Brandenburg University of Technology
Cottbus-Senftenberg
Chair of IT Security

Exercise Class
“Introduction into Cyber Security”
Winter Term 2018/2019

Introduction into Cyber Security
– Buffer Overflow Attacks –

Deadline: 9th January, 2019

Contents

1 Introduction

Buffer overflow attacks are one of the most common attacks used to break in computer systems and spread malware, like viruses, worms or Trojans. If one follows the news about software vulnerabilities then it seems that there is no software without the potential risk of a buffer overflow. No operation system, no server and no application is safe from the risks and consequences of buffer overflow attacks. Even firewalls are affected (cf. [5]). In the setting of this laboratory task you will obtain a basic knowledge and understanding of the problematic of buffer overflows and execute your own attack on a simple application.

2 Preliminaries

Make yourself familiar with the problematic of buffer overflows and how they are exploited. A good starting point may the articles [5], [14], [15] and [3]. Further, you will need a basic knowledge about assembler (e.g., `nasm` (see [12])), especially starting a shell, to develop the attack. The GDB-GNU Debuggers `gdb` (see [10]) may be helpful during your development as well.

Also you should take a look at the so called “swiss pocket knife for networks”, `netcat` [13].

3 The Laboratory Setting

In this laboratory exercise you will work with the virtualization tool *VirtualBox* [21]. With Virtualbox it is possible to setup virtual computers and networks, which can be executed one a single computer or if needed on shared machines. In the context of this task you will receive two virtual machines like depicted in figure 1.

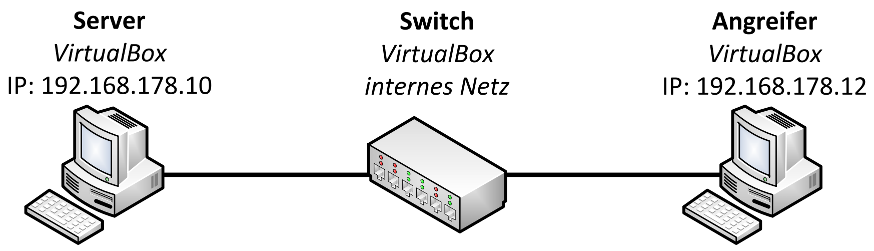

Figure 1: Laboratory Setting

Thus, there are two virtual computers with the stated ip addresses which are connected by a virtual switch. The switch is part of the internal network simulated by VirtualBox.

3.1 Preparation: Installation of the Virtual Machines

The task can be either solved in the computer pools (VG1C, room 2.23) or on your own computer at home. The following steps to setup the virtual environment in the computer pools is obligatory and necessary to successfully submit and present your solutions.

Installation of the Virtual Machines in the Computer Pools *VirtualBox* is already installed on the computer pools. To integrate the given virtual machines, the following steps are mandatory:

1. Log in to one of the computers using your BTU-Account. Thereby use Linux as operation system.

2. Create the directory *ICS/02_task* in your home directory
   ```
   > mkdir /home/<user>/ICS/02_task
   ```

3. The configuration of *VirtualBox* is done by a script. To import the appliance execute the following steps:
   ```
   > /images/ics/icsT2_import.sh all
   ```
   You work in the attacker system only. If you cause or notice any damage or misbehavior of the virtual machines you can just re-import the environment by the steps above. Thereby all your modifications and files on the attacker system get lost!
   ```
   > /images/ics/icsT2_import.sh attacker
   ```
   Attention: The import process of the Appliance may take some time! The process is indicated after a short delay in steps of 10%.

After the successful integration of the appliance you can find the script *icsT2_control.sh* in your home directory. Check if the files exists and continue by step 3.2.

Installation of the Virtual Machines on your own Computer As an alternative to the computer pools you can solve the task on your own computer. Therefore you need first to install the current build of virtual box on your machine. To the date of this laboratory task it is *Version 5.1.8*, like it is already installed in the computer pools. Tutorials and a Manual to help you on the installation process as well as the usage can be find on the vendors website [21].

The appliance to import the virtual machines into your own setup can be obtained by either downloading the neccesarry files from[1]

`https://www.b-tu.de/owncloud/s/GFSk3Fped4M8kSn`

or exporting the virtual machines you have already configured in step “Installation of the Virtual Machines in the Computer Pool”. Therefore the following steps are necessary:

1. Log in to one of the machines in the computer pool and change to your working directory of this task.
2. Export the appliance using the provided control script *icsT2_control.sh:*
   ```
   > ./icsT2_control.sh export
   ```

The process of exportation is indicated in the command line. After the successful export you should find the appliance *(icsT2.ova)* in your current directory. Copy the file (approx 450 MB) to a jump drive. Now you can import the appliance on your own computer. Hints are given by the manual of *VirtualBox*, which could be downloaded from the vendors website [21].

In any case, on account of the setup in the computer pools you need to customize your network setting in VirtualBox after the import. Here fore you just need to create the virtual network device *vboxnet0* (IP: 10.0.0.1, Subnet-mask: 255.255.255.0). For further details please use the information provided by the manual.

3.2 Working with the Virtual Machines

3.2.1 Start/Stop the Appliance

To start/stop the virtual environment in the setting of the computer pools you should use the provided script *icsT2_control.sh*:

```
> ./icsT2_control.sh {start | stop}
```

To access the attacker system you should use the credentials

[1] There are two files provided. The file icsT2_appliance.ova contains the actual appliance. The file icsT2_attacker.ova contains just the virtual machine of the attacker. In the case that the provided link won’t work please use the possibility to export the appliance from the computer pools.

```
User:  root
Pass:  itsprak
```

You have root privileges on this account. For the server system you do not have access rights.

Hint: The script *icsT2_control.sh* starts/stops both virtual machines. Since the script was developed for the usage in the computer pools, there is may no compatibility to your own virtualization environment. Thus, you need to start the virtual machines on your own.

3.2.2 Changing the Keyboard Layout

Per default a German (qwertz) keyboard layout is configured. If you prefer to use a different layout you can change this setting at anytime. Therefore use the command

```
> loadkeys <language code>
```

The language code could be for example "de" (German Keyboard Layout), "us" (American Keyboard Layout). For further language codes take a look at the directory

```
/usr/share/keymaps/i386/
```

3.2.3 Copying Files

use " scp " for copy and backup data in remote machine

To avoid a possible lose of data it is recommended to backup the files on the attacker machine from time to time. A usual way to execute these backups is to use *scp* [17] to copy files from the attacker system to the host system. Therefore the attacker system is equipped with a second network interface. This is configured to host-only mode and could be addressed from the host by the IP 10.0.0.20, or from the virtual machine by the IP 10.0.0.1.

In the case that you have decided to solve the task on your own computer, you still need to copy and test your solution to the visualization environment on the computer pools in preparation of your presentation. Therefore the usage of *scp* is recommended as well.

3.2.4 Parallel Execution of Multiple Command interpreters by Screen

Screen [7] is a so called "‘Commandline-Multiplexer"’. The Tool allows to execute multiple (text based) applications, and switch between them wihtout the need of opening multiple console sessions. Screen is already installed on the attacker system and provides a comfortable way to work with multiple command line interpreters.

Closing Screen By the command `screen` there will be a new `screen` session started, a virtual shell is spawned automatically. By the keystroke **[ctrl]+[a]** you can change to the command mode of `screen`. By further keystrokes, using the hot keys given below, screen can now be interfaced. For example by **[ctrl]+[a], [c]** further virtual consoles can be spawned. A full list of available hot keys can be found in the manual, the most important functions are listed in table 1 below.

Function	**Command**
Spawn new virtual console	[ctrl]+[a], [c]
Switch to the next console	[ctrl]+[a], [n]
Switch to the previous console	[ctrl]+[a], [p]
Rename the current console	[ctrl]+[a], [A]
Get a overview of active consoles	[ctrl]+[a], ["]
Close conosle	[ctrl]+[a], [K]
Fade out screen	[ctrl]+[a], [d]

Table 1: Screen-Commands

By the keystroke **[ctrl]+[a], [d]** the active screen console will be detached. Thereby screen will be faded out but will continue running in background mode. To take back the session use the command `screen -r` to bring a detached session to foreground again. If there are multiple detached screen sessions you need to provide the PID of the screen session you want to bring to foreground. Informations on the current screen session can be obtained by `screen -ls`. To terminate screen it is necessary to close all active console windows.

4 Task Description

The server provides the service timeservice on port 1234. Your task is to obtain the secret file (secret.txt) which lies somewhere on the server. Therefore the C-source-code (timeservice.c) of the service is given. To execute an buffer overflow attack on the time server it is mandatory to find the vulnerability first. Therefore the compiled binary (timeservice) is provided.

In order to exploit the time service it is important to know that the Linux-Kernel has implemented the so called *address space layout randomization* since the middle of 2005. Here, the stack will not start at the top of the virtual memory address space. Instead, it will be shifted downwards by an random offset. Thus, the address of where one needs to jump in order to execute arbitrary code will change randomly. In consequence the most basic exploits will not work anymore. In the development phase of the exploit one should carefully check which buffer is usable for the attack and which are not. To test your implemented attack the binary is provided. Alternatively, you can compile the source code on your own. In this case you should use the GNU compiler, version *3.4*.

```
> gcc-3.4 -o timeservice timeservice.c
```

Do not use the short command `gcc` since then `gcc-4.1` will be executed which yields in a different binary.

The GNU-Debugger `gdb` is a very useful tool to obtain the return address. Some additional hints on how to use `gdb` are given in the section 5.2. To implement the exploit itself, we strongly recommend the usage of the `nasm`-Assembler. This assembler compiles pure machine code if it is called without any additional parameter. Different assembler often pack the compiled code into container formats compatible with the used operating system. This makes no sense in the case of exploit development.

After the successful implementation of the attack please submit the file *secret.txt*, obtained from the server, as well as the exploit source code via moodle.

Good Luck!

5 Hints

This section provides helpful tips and hints on the usage of the tool `gdb`. Further, the steps needed to start an Unix/Linux Shell in assembler are discussed.

5.1 Creation of an UNIX/Linux Shell in Assembler

We start by discussing the steps to execute a system call to start an Unix/Linux Shell in C.

```
#include <unistd.h>
void main()
{
        char *shell[2];
        shell[0] = "/bin/sh";
        shell[1] = NULL;
        execve(shell[0], shell, NULL);
}
```

The system call `execve` to start a process is called with:

- an NULL-terminated string as a the first parameter providing the process name (here `shell[0]`)
- an NULL-terminated array of pointers of NULL-terminated process parameters (here the shell-Array `shell[0]` equals the first parameter of the shell)
- an additional pointer pointing to the environment variables

The example code have to be compiled static and loaded by `gdb <program name>` into the debugger. Now you have the possibility to see the assembly of the C-source-code of the `execve` system call. The default notion of `gdb` is Intel-Syntax.

```
(gdb) set disassembly-flavor intel
(gdb) disassemble execve
0x0804d768 <execve+0>:  push ebp
0x0804d769 <execve+1>:  mov ebp,esp
0x0804d76b <execve+3>:  push ebx
0x0804d76c <execve+4>:  mov ebx,DWORD PTR [ebp+8]
0x0804d76f <execve+7>:  mov ecx,DWORD PTR [ebp+12]
0x0804d772 <execve+10>: mov edx,DWORD PTR [ebp+16]
0x0804d775 <execve+13>: mov eax,0xb
```

The first three lines creates a new stack frame. The following four lines of code show that

- the first parameter *[ebp+8]* used for `execve` is passed by the *ebx*-Register,
- the second parameter *[ebp+12]* is passed by the *ecx*-Register,
- the third parameter *[ebp+16]* is passed by the *edx*-Register.

The actual system call is not directly observable by the `gdb` output. It works in the following way: The system call which should be executed will be pushed to the *eax*-Register (here *0x0b* (=11)). Afterwards a trap instruction is used to enter the kernel. In `nasm` it looks like following:

```
; load the function parameters to the specified registers
mov ebx, ...
mov ecx, ...
mov edx, ...

; move the system call number into the eax-Register
mov eax, 0x0b

; trap into the kernel
int 0x80
```

5.2 Working with the GNU-Debugger

After the program is loaded by `gdb <program name>`, the Debugger offers many features which can be used for statical analysis of binary code. The table 2 is just a short outline of some of the useful GNU-Debuggers commands. A good article on the usage of `gdb` in the context of buffer-overflow exploits can be found in [18].

function	**command (hot key)**
print source code	`l [function name, line number]`
insert break point	`b <line number>`
print address of a variable	`p <variable>`
print function in assembler syntax [2]	`disas <function>`
run program [3]	`r <parameter list>`
show register contents	`i r`
inspect memory block	`x/<format> <address>`
print help to specific command	`h <command>`

Table 2: useful gdb commands

5.3 Netcat - the Swiss Pocket Knife of Network Aadministration

Netcat (see [13]) is a command line tool which allows to pipe communication of a TCP- or UDP-connection by stdin/stdout. Have a special notice of the parameter "-e" , which could be very interesting in our considered laboratory setting (Both machines have an existing netcat installation).

[2]Per default GDB execute flow follows the parent process (parent). This behavior is not intended in any case. By `> (gdb) set follow-fork-mode child` the setting is changed such that GDB follows the child-process during the execution flow.

[3]Thereby the assembler is printed in AT&T syntax. If one wants to use the Intel notion like used by `nasm`, one can easily change the format by `> (gdb) set disassembly-flavor intel`.

References

[1] *Address Resolution Protocol (Manual Page)*. URL: http://www.unix.com/man-page/linux/8/arp/.

[2] *Address Resolution Protocol (RFC 826)*. URL: http://tools.ietf.org/html/rfc826.

[3] M. Balaban. *Buffer Overflows Demystified*. URL: http://www.enderunix.org/docs/eng/bof-eng.txt (cit. on p. 2).

[4] J. Buchmann. *Einführung in die Kryptographie*. Springer, 2010. ISBN: 978-3642111853.

[5] *Buffer-Overflows und andere Sollbruchstellen*. Heise Security. 2003. URL: http://www.heise.de/security/artikel/37958/0 (cit. on p. 2).

[6] *CrypTool: eLearning-Programm für Kryptologie. Download-Bereich von Cryptool 1*. URL: https://www.cryptool.org/en/ct1-download-en.

[7] *Der Fenstermanager Screen*. URL: https://www.gnu.org/software/screen/ (cit. on p. 5).

[8] M. Dethlefsen. *Die Therorie des Sniffens in geswitchten Netzen*. Magazin: hakin9, Ausgabe 2. 2004.

[9] B. Esslinger. *Das CrypTool-Skript: Kryptographie, Mathematik und mehr.* 2010. URL: https://www.cryptool.org/images/ctp/documents/CrypToolScript-de.pdf.

[10] *gdb - The GNU Debugger Project*. URL: http://sourceware.org/gdb/ (cit. on p. 2).

[11] E. A. Hall. *Internet Core Protocols*. O'Reilly & Associates, 2000. ISBN: 978-1565925724.

[12] *nasm - The Netwide Assembler*. URL: http://www.nasm.us/docs.php (cit. on p. 2).

[13] *netcat - The GNU Netcat Project*. URL: http://netcat.sourceforge.net/ (cit. on pp. 2, 9).

[14] A. One. *Smashing the Stack for Fun and Profit*. URL: http://insecure.org/stf/smashstack.html (cit. on p. 2).

[15] *Pufferüberlauf - Wikipedia-Artikel*. URL: https://de.wikipedia.org/wiki/Puffer%C3%BCberlauf (cit. on p. 2).

[16] J. Schwenk. *Sicherheit und Kryptographie im Internet: Von sicherer E-Mail bis zu IP-Verschlüsselung*. Vieweg+Teubner Verlag, 2005. ISBN: 978-3834800428.

[17] *Secure Copy (Manual Page)*. URL: http://www.unix.com/man-page/linux/1/scp/ (cit. on p. 5).

[18] P. Sobolewski. *Overflowing the stack on Linux x86*. Hakin9 magazine. 2004. URL: http://www-scf.usc.edu/~csci530l/downloads/stackoverflow_en.pdf (cit. on p. 9).

[19] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2010. ISBN: 978-0136097044.

[20] *tcpdump (Manual Page)*. URL: http://www.tcpdump.org/tcpdump_man.html.

[21] *VirtualBox: Homepage*. URL: http://www.virtualbox.org (cit. on pp. 2–4).