HW4

Submit Assignment

Due Apr 16 by 11am **Points** 120 **Submitting** a file upload
Available after Apr 2 at 12:15pm

This homework assignment needs you to understand vulnerabilities in 5 target programs. The first 4 are required; if you do the 5th you can earn extra credit. You **may pair up for this home work** (but it's not a requirement)**.** No extra credit for doing it alone. Click here to download HW4 files: **HW4.zip**.

Prerequisites

For this project, you are going to use a virtual machine configured with Debian stable ("Lenny"), with ASLR (address space layout randomization) turned off. We provide basic VM images for Oracle VirtualBox.

- Download Oracle's Virtual Box: **https://www.virtualbox.org/wiki/Downloads (https://www.virtualbox.org/wiki/Downloads)**
- Download **a virtual machine description file**.
- Download **a virtual disk image file**.

Environment Setup

- The user name and password are '**user**' and '**user**'.
- The root name and password are '**root**' and '**root**'.

SSH Configuration

It might be easier to work on the VM over SSH from the guest --- easier to do copy-paste commands. Here are some guidelines in case you want to use SSH. (You are free to use the native VM environment or SSH.)

Step 1: Enable Port Forwarding Feature for VM

You will have to enable port forwarding feature to access VM from any terminal using SSH.

On the VirtualBox application window where VM is listed, right click VM listed and click `Settings`.

In `Network` tab, under `Adapter 1`, select `NAT` in `Attached to` option.

Under `Advanced`, click `Port Forwarding` which open a window to set port forwarding option.

Add an entry there with following fields

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port

SSH	TCP	0.0.0.0	2222		22

Step 2: Using SSH to Access VM and SCP to Transfer Files/Folders

Save the settings in step 1 and restart your VM.

Now sshinto the VM from terminal using the following command

```
ssh -p 2222 user@127.0.0.1
```

You can also SCP file/folder into VM using the following command

```
scp -P 2222 file user@127.0.0.1:~/
scp -r -P 2222 folder user@127.0.0.1:~/
```

The Targets

The `targets/` directory contains the source code for the vulnerable targets, along with a `Makefile` for building them.

Your exploits should assume that the compiled target programs are installed setuid-root in `/tmp` --> `/tmp/target0`, `/tmp/target1`, etc.

To build the targets, change to the `targets/` directory and type `make` on the command line; the `Makefile` will take care of building the targets.

To install the target binaries in `/tmp`, run:

```
make install
```

To make the target binaries setuid-root, run:

```
su
make setuid
exit  # to get out of the root shell
```

Once you've run `make setuid` use `exit` to return to your user shell.

Keep in mind that it'll be easier to debug the exploits if the targets aren't setuid. (See below for more on debugging.) If an exploit succeeds in getting a user shell on a non-setuid target in `/tmp`, it should succeed in getting a root shell on that target when it is setuid. (But be sure to test that way, too, before submitting your solutions!)

The Exploits

The `sploits/`contains skeleton source for the exploits which you are to write, along with a `Makefile` for building them. Also included is `shellcode.h`, which gives Aleph One's shellcode (code which leads a vulnerable target to a shell).

The Assignment

You are to write exploits for vulnerable targets, with one exploit per target.

1. The goal of `sploit0` is different from the rest of the exploits. Take a look at `target0.c`, the output is "Grade = F" for any string (<20 bytes) you pass. Your goal as an attacker is to hijack the control flow of `target0` to print "Grade = A". Use `sploit0.c` to generate and pass a "string" that is going to aid you in obtaining the desired grade, which is A obviously.
2. The rest of the exploits (`sploit1` through `sploit4`), when run in the virtual machine with its target installed setuid-root in `/tmp`, should yield a root shell (`/bin/sh`).
3. For full credit you **must provide** an explanation in the **sploit[0-4].txt** files.
4. We will grade each of your exploits implementation on **a zero or all basis**. So make sure your code works before submission!

Once again, `sploit0-3` are required; `sploit4` is extra credit.

Hints

Read the Phrack articles suggested below. Read Aleph One's paper carefully, in particular.

To understand what's going on, it is helpful to run code through gdb. See the GDB tips section below.

Make sure that your exploits work within the provided virtual machine.

Start early! Theoretical knowledge of exploits does not readily translate into the ability to write working exploits. `target0` is relatively simple and the other problems are quite challenging.

GDB Tips

Notice the `disassemble` and `stepi` commands.

You may find the `x` command useful to examine memory (and the different ways you can print the contents such as `/a` `/i` after `x`). The `info register` command is helpful in printing out the contents of registers such as `ebp` and `esp`.

A useful way to run gdb is to use the `-e` and `-s` command line flags; for example, the command `gdb -e sploit3 -s /tmp/target3` in the VM tells gdb to execute `sploit3` and use the symbol file in `target3`. These flags let you trace the execution of the `target3` after the sploit's memory image has been replaced with the target's through the execve system call.

When running gdb using these command line flags, you should follow the following procedure for setting breakpoints and debugging memory:

1. Tell gdb to notify you on exec(), by issuing the command `catch exec`
2. Run the program. gdb will execute the sploit until the execve syscall, then return control to you
3. Set any breakpoints you want in the target
4. Resume execution by telling gdb `continue` (or just `c`).

If you try to set breakpoints before the exec boundary, you will get a segfault.

If you wish, you can instrument the target code with arbitrary assembly using the __asm__ () pseudofunction, to help with debugging. **Be sure**, however, that your final exploits work against the unmodified targets, since these we will use these in grading.

Warnings

Aleph One gives code that calculates addresses on the target's stack based on addresses on the exploit's stack. Addresses on the exploit's stack can change based on how the exploit is executed (working directory, arguments, environment, etc.); during grading, we do not guarantee to execute your exploits exactly the same way bash does. You must therefore **hard-code target stack locations** in your exploits. You should *not* use a function such as get_sp() in the exploits you hand in.

(In other words, during grading the exploits may be run with a different environment and different working directory than one would get by logging in as user, changing directory to `~/hw4/sploits`, and running `./sploit1`, etc.; your exploits must work even so.)

Your exploit programs should not take any command-line arguments.

Submission

You need to submit:

1. The source code for your exploits (`sploit0.c` through `sploit3.c` and/or `sploit4.c`).
2. ID: this include both you and your partner's wisc emails and names.
3. The explanation for your exploits(`sploit0.txt` through `sploit3.txt` and/or `sploit4.txt`) to explain how your exploit works: what is the bug in the corresponding target, how you exploit it, and where the various constants in your exploit come from. If you **don't explain, you won't get full credits** even if your code works.

As a team, only one person should submit the homework. That person should compress all the files into one zip file, name it with his/her own netID.

Suggested Readings

Basic readings:

- Aleph One, **Smashing the stack for fun and profit (https://insecure.org/stf/smashstack.html)**
- blexim, **Basic Integer Overflows (http://phrack.org/issues/60/10.html)**, Phrack 60 #0x10.
- Gera and Riq, **Advances in Format String Exploiting (http://phrack.org/issues/59/7.html)**, Phrack 59 #0x04.
- klog, **The Frame Pointer Overwrite (http://phrack.org/issues/55/8.html)**, Phrack 55 #08.
- Anonymous, **Once Upon a free()... (http://phrack.org/issues/57/9.html)**, Phrack 57 #0x09.

Advanced readings:

- Bulba and Kil3r, **Bypassing StackGuard and StackShield (http://phrack.org/issues/56/5.html)**, Phrack 56 #0x05.

- Silvio Cesare, **Shared Library Call Redirection via ELF PLT Infection (http://www.phrack.org/archives/issues/56/7.txt)** , Phrack 56 #0x07.
- Michel Kaempf, **vudo - An Object Superstitiously Believed to Embody Magical Powers (http://phrack.org/issues/57/8.html)** , Phrack 57 #0x08.

HW4 Rubric		
Criteria	**Ratings**	**Pts**
sploit0 explanation		10.0 pts
sploit0 correct implementation		15.0 pts
sploit1 explanation		10.0 pts
sploit1 correct implementation		15.0 pts
sploit2 explanation		10.0 pts
sploit2 correct implementation		15.0 pts
sploit3 explanation		10.0 pts
sploit3 correct implementation		15.0 pts
sploit4 explanation		5.0 pts
sploit4 correct implementation		15.0 pts
		Total Points: 120.0