Homework 1
CS 642: Information Security

September 17, 2014

This homework assignment tasks you with understanding vulnerabilities in five target programs. You may (optionally) work with a partner. It is due **October 6, 2014** by midnight local time.

1 The Environment

You will test your exploit programs within a VMware virtual machine (VM) we provide, which is configured with Debian stable ("Lenny"), with ASLR (address space layout randomization) turned off. The VM image is available on the course webpage. A separate package, pp1.tar.gz, contains the other files needed for the assignment. You can use the ssh daemons running in the image to transfer files into the VM, or access the Web directly from the VM using something like wget. One can login to the guest OS as login root with password "root" or as login user with password "user".

There are multiple free VM managers that you might use.

1. VMware Player: Free for personal use. Can be downloaded from here: `https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/5_0`
2. Crash And Burn Virtual Lab: Students can check out a VM for the semester. Details of how to request and the VM configuration can be found here: `http://research.cs.wisc.edu/twiki/bin/view/CSDocs/CrashAndBurnLabInfo`
3. Virtual Box: Available as Open Source Software under the terms of the GNU General Public License (GPL). Download from here: `https://www.virtualbox.org/wiki/Downloads`

If you have problems getting access to a suitable VM manager, contact the TA (`shouhm@gmail.com`) for help.

2 The Targets

- The targets/ directory in the assignment tarball contains the source code for the targets, along with a Makefile specifying how they are to be built.
- Your exploits should assume that the compiled target programs are installed setuid-root in /tmp – /tmp/target1, /tmp/target2, etc.
- Note that there are five targets that you are responsible for.
- To build the targets, change to the pp1/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, first run "make install"; then, *as root*, run "make setuid". You can get a root shell in the current directory using the "su" command; once you've run "make setuid" you can exit this shell, which will return you to your user shell. Alternatively, you can keep a separate terminal or virtual console open with a root login, and run "make setuid" (in the user/pp1/targets directory!) in that terminal or console.
- 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!)

3 The Assignment

The sploits/ directory in the assignment tarball 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.

You are to write exploits, one per target. Each exploit, when run in the virtual machine with its target installed setuid-root in /tmp, should yield a root shell (/bin/sh). Your task is to attack the five targets named target1 through target5.

4 Hints

- Read the readings in Phrack suggested below. Read Aleph One's paper carefully, in particular. Read Scut's paper on format string vulnerabilities, linked from the course syllabus.
- 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. Target1 is relatively simple and the other problems are quite a bit more complicated.

5 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.); in my testing, I 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, in 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 /pp1/sploits, and running ./sploit1, etc.; your exploits should work even so.)

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

6 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.

7 Deliverables

You will have three deliverables.

1. You will need to submit the source code for your exploits (sploit1 through sploit5), along with any files (Makefile, shellcode.h) necessary for building them.
2. In addition, along with each exploit, you should include a text file (sploit1.txt, sploit2.txt, and so on). In each text file, explain how your exploit works: what the bug is in the corresponding target, how you exploit it, and where the various constants in your exploit come from. Be concise.
3. Finally, you must include a file called ID which contains, on a single line, the following: your UW ID; and your name, in the format last name, comma, first name. An example:

   ```
   $ cat ./ID
   9064562678 Doe, John
   $
   ```

   If you work with a partner, the ID file should contain two lines, one for each of you.

You will submit your work through Moodle.

8 Grading

You will receive two points (out of 10) for each exploit that yields a root shell in our testing for the first 5 targets.

There will not normally be partial credit for any of the exploits, but we may make an exception depending on your explanatory writeup. We may also ask you to explain to us how and why each exploit works. If you work with a partner, be sure that each of you understands every exploit you turn in!

9 Suggested reading in Phrack, www.phrack.org

- Aleph One, "Smashing the Stack for Fun and Profit," Phrack 49 #14.
- klog, "The Frame Pointer Overwrite," Phrack 55 #08.
- Bulba and Kil3r, "Bypassing StackGuard and StackShield, Phrack 56 #0x05.
- Silvio Cesare, "Shared Library Call Redirection via ELF PLT Infection," Phrack 56 #0x07.
- Michel Kaempf, "Vudo - An Object Superstitiously Believed to Embody Magical Powers," Phrack 57 #0x08.
- Anonymous, "Once Upon a free()...," Phrack 57 #0x09.
- Nergal, "The Advanced Return-into-lib(c) Exploits: PaX Case Study," Phrack 58 #0x04.
- Gera and Riq, "Advances in Format String Exploiting," Phrack 59 #0x04.
- Anonymous, "Bypassing PaX ASLR Protection," Phrack 59 #0x09.
- blexim, "Basic Integer Overflows," Phrack 60 #0x10.

10 Other Books

The Gray Hat Hacking book suggested on the course website is a useful reference. Here are a plethora of (other) books that may prove helpful:

- W. Richard Stevens, /Advanced Programming in the Unix Environment./ Addison-Wesley, 1993. AEleen Frisch, /Essential System Administration,/ second edition. O'Reilly, 1995.

The latest versions of these manuals are online at `http://developer.intel.com/products/processor/manuals/` .

- /IA-32 Software Developer's Manual, Vol. 1: Basic Architecture./ Intel, 2001.
- /IA-32 Software Developer's Manual, Vol. 2: Instruction Set Reference./ Intel, 2001.
- /IA-32 Software Developer's Manual, Vol. 3: System Programming Guide./ Intel, 2001.

These are online at `http://www.gnu.org/manual/manual.html`:

- Dean Elsner, Jay Fenlason, et al., /Using AS./ FSF, 1994.
- Richard Stallman et al., /Using the GNU Compiler Collection/ FSF, 2002.
- Richard Stallman, Roland Pesch, Stan Shebs, et al. /Debugging with GDB./ FSF, 2001.

Acknowledgements

This assignment is based in part off materials from Prof. Hovav Shacham at UC San Diego as well as Prof. Dan Boneh at Stanford. Thanks for their hard work.