UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2019**

Cecylia Bocovich
Yaser Baseri

ASSIGNMENT 1

Blog Task signup due date: **Fri, January 18, 2019 at *4:00 pm (no extension)***
Milestone due date: **Fri, Jan 25, 2019 at *4:00 pm***
Assignment due date: **Tues, Feb 5, 2019 at *4:00 pm***

Total Marks: 70
Blog Task TA: Mark Iwanchyshyn
Written Response TA: Yossef Musleh
Programming TA: Stan Gurtler

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are posted to Piazza.

Blog Task

0. [0 marks, but -2 if you do not sign up by the due date] Sign up for a blog task timeslot by the due date above. The 48 hour late policy, as described in the course syllabus, does not apply to this signup due date. Look at the blog task in the Course Materials, Content section of the course website to learn how to sign up.

   Please visit https://crysp.uwaterloo.ca/courses/cs458/infodist/blogtask.php to sign up.

Written Response Questions [30 marks]

Note: Please ensure that written questions are answered using complete and grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as on the correctness of your answers. **Note:** Further, do not copy text directly from any sources, and make sure to cite any sources you do use.

1. As part of the CrySP speaker series on privacy, Josh Benaloh from Microsoft Research spoke about conducting elections with both privacy and integrity (https://crysp.uwaterloo.ca/speakers/20170313-Benaloh). Benaloh is promoting Internet voting by making elections secure without compromising privacy or facilitating voter coercion.

   In this question, you are asked to compare traditional voting to Internet voting.

   In traditional voting, voters go to the polling station, where officials ensure that a voter is eligible to vote and has not voted yet. If this vote registration step is successful, the voter will be given a paper

ballot. The voter fills in the paper ballot in secret and puts the ballot into a locked ballot box that is in plain view of some officials. At the end of the day, the officials jointly open the ballot box, count the votes, and phone headquarters to submit the results. At a later stage, a signed report is sent to headquarters.

In Internet voting, voters use their home computer or smartphone to place their vote. Each eligible voter is physically mailed a letter with the URL of the voting website and a unique authorization code. The voter uses his/her/their computer/smartphone to go to the voting website and enter his/her/their vote and the authorization code. This information is transmitted to a centralized server, which ensures that the authorization code is valid and has not been used. Then the server aggregates the votes.

- (8 marks) Explain whether paper-based voting can be susceptible to interception, interruption, modification, and fabrication attacks. If you believe the system is susceptible to a given attack, provide a specific, realistic attack scenario. If necessary, state any additional assumptions made about the system.
- (8 marks) Explain whether Internet voting can be susceptible to interception, interruption, modification, and fabrication attacks. If you believe the system is susceptible to a given attack, provide a specific, realistic attack scenario. If necessary, state any additional assumptions made about the system.

2. (6 marks) For each of the following identify whether the scenario violates the property of confidentiality, integrity, availability, and/or privacy. For each scenario indicate one property that is being violated and explain how/why this property is violated in the scenario.

   (a) You sent an email to your friend Alice containing a number of cute cat photos to cheer them up. Mallory intercepts the message and thinks it would be funny to change all of the images to be of puppies before she sends the message back on its way to Alice.

   (b) While in the airport you connect to what you believe to be a secure WiFi network and you head to your favorite web page for baby alpaca pictures. When attempting to login to your account on the site to see your favorite saved baby alpacas you find that the page will not load after you enter your credentials. Additionally, when you entered your information the data (password, username) was sent on to an additional party who had replaced the login screen on the alpaca site.

   (c) You regularly access an online store to purchase purple spider posters and other neat items. Unknown to you the site sells any information you provide as well as information on your purchases to an advertising company.

3. (8 marks) Identify each of the following pieces of malware as a worm, trojan, ransomware, and/or logic bomb. Then, briefly and concisely, give a description of how it spread, or how a computer is infected, and the resulting effect. Keep in mind that each piece of malware could have multiple classifications.

   Example: CryptoLocker.
   A ransomware trojan targeting MS Windows machines, and disseminated through malicious email attachments and the ZeuS Botnet. Once activated it uses public key cryptography to encrypt files, forcing the victim to pay a ransom for the release of the private key to decrypt their files.

   (a) Kovter.

   (b) ILOVEYOU.

   (c) Mirai.

   (d) NotPetya.

Programming Question [40 marks]

Background

You are tasked with testing the security of a custom-developed password-generation application for your organization. It is known that the application was *not written with best practices in mind*, and that in the past, this application had been exploited by some users with the malicious intent of *gaining root privileges*. There is some talk of the application having *four or more vulnerabilities*! As you are the only person in your organization to have a background in computer security, only you can *demonstrate how these vulnerabilities can be exploited* and *document/describe your exploits* so a fix can be made in the future.

Application Description

The application is a very simple program with the purpose of generating a random password and optionally writing it to `/etc/shadow`. The usage of `pwgen` is as follows:

```
Usage:  pwgen [options]
Randomly generates a password, optionally writes it to /etc/shadow
Options:
 -s, --salt <salt>  Specify custom salt, default is random
 -e, --seed [file]  Specify custom seed from file, default is from stdin
 -t, --type <type>  Specify different hashing method
 -w, --write        Write the password to /etc/shadow.
 -h, --help         Show this usage message
Hashing algorithm types:
0 - DES (default)
1 - MD5
2 - Blowfish
3 - SHA-256
4 - SHA-512
```

There may be other ways to invoke the program that you are unaware of. Luckily, you have been provided with the source code of the application, `pwgen.c`, for further analysis.

The executable `pwgen` is *setuid root*, meaning that whenever `pwgen` is executed (by any user), it will have the full privileges of *root* instead of the privileges of the user that invokes it. Therefore, if an outside user can exploit a vulnerability in a setuid root program, he or she can cause the program to execute arbitrary code (such as shellcode) with the full permissions of the root user. If you are successful, running your exploit program will execute the setuid `pwgen`, which will perform some privileged operations, which will result in a shell with root privileges. You can verify that the resulting shell has root privileges by running the `whoami` command within that shell. The shell can be exited with `exit` command.

Testing Environment

To help with your testing, you have been provided with a virtual *user-mode linux* (uml) environment where you can log in and test your exploits. These are located on one of the *ugster* machines. You can retrieve

your account credentials from the Infodist system
https://crysp.uwaterloo.ca/courses/cs458/infodist/.

Once you have logged into your ugster account, you can run `uml` to start your virtual linux environment. The following logins are useful to keep handy as reference:

- `user` (no password): main login for virtual environment
- `halt` (no password): halts the virtual environment, and returns you to the ugster prompt

The executable `pwgen` application has been installed to `/usr/local/bin` in the virtual environment, while `/usr/local/src` in the same environment contains `pwgen.c`. Conveniently, someone seems to have left some shellcode in `shellcode.h` in the same directory.

It is important to note all changes made to the virtual environment will be lost when you halt it. Thus it is important to remember to keep your working files in `/share` on the virtual environment, which maps to `~/uml/share` on the ugster environment.

Note that in the virtual machine you cannot create files that are owned by root in the `/share` directory. Similarly, you cannot run chown on files in this directory. (Think about why these limitations exist.)

The root password in the virtual environment is a long random string, so there is no use in attempting a brute-force attack on the password. You will need to exploit vulnerabilities in the application.

Rules for exploit execution

- You must submit a total of four (4) exploit programs to be considered for full credit. Keep in mind, one (1) must target a *buffer overflow* vulnerability that overwrites a saved return address on the stack.
- Running each exploit program should result in a shell owned by root.
- Each vulnerability can be exploited only in a single exploit program. A single exploit program can exploit more than one vulnerability. (But remember, if you don't have to use more than one, you probably shouldn't! If you do, you won't be able to use it somewhere else it could be more useful.) You can exploit the same *class* of vulnerability (ex: buffer overflow, format string, etc) in multiple exploit programs, but they must exploit different sections of the code. You may also exploit the same section of code in multiple exploit programs as long as they each use a *different* class of vulnerability. If unsure whether two vulnerabilities are different, please ask a private question on Piazza.
- There is a specific execution procedure for your exploit programs ("*sploits*") when they are tested (i.e. graded) in the virtual environment:
  - Sploits will be run in a **pristine** virtual environment, i.e. you should not expect the presence of any additional files that are not already available
  - Execution will be from a clean home directory (`/home/user`) on the virtual environment as follows: `./sploitX` (where X=1..4)

  - Sploits must not require any command line parameters
  - Sploits must not expect any user input
  - Sploits must not take longer than 60 seconds to complete
  - If your sploit requires additional files, it has to create them itself

- For marking, we will compile your exploit programs from a clean `/share` directory into the `/home/user` directory in a virtual machine in the following way:
  `cd /share && gcc -Wall -ggdb sploitX.c -o /home/user/sploitX`
  You can also assume that `shellcode.h` is available in the `/share` directory.

- Be polite. After ending up in a root shell, the user invoking your exploit program must still be able to exit the shell, log out, and terminate the virtual machine by logging in as user `halt`. Also, please do not run any cpu-intensive processes for a long time on the ugster machines (see below). None of the exploits should take more than about a minute to finish.

- Give feedback. In case your exploit program might not succeed instantly, keep the user informed of what is going on. (This helps the human grader know to guess that your program may not be running infinitely if it does take some time).

The goal is to end up in a shell that has root privileges. So you should be able to run your exploit program, and without any user/keyboard input, end up in a root shell. If you as the user then type in `whoami`, the shell should output `root`. Your exploit code itself doesn't need to run whoami, but that's an easy way for you to check if the shell you started has root privileges.

For example, testing your exploit code might look something like the following:

```
user@cs458-uml:~$ ./sploit1
sh# whoami
root
sh#
```

For questions about the assignment, ugsters, virtual environment, Infodist, etc, please post a question to Piazza. General questions should be posted publicly, but **do not** ask public questions containing (partial) solutions on Piazza. Questions that describe the locations of vulnerabilities, or code to exploit these vulnerabilities, should be posted *privately*. If you are unsure, you can always post your question privately and a TA can (with your consent) make your question public if they believe it to be useful for the class.

Deliverables

Each sploit is worth 10 marks, divided up as follows:

- 6 marks for a successfully running exploit that gains a shell owned by the root user
- 4 marks for the description of:
  - the identified vulnerability/vulnerabilities
  - how your exploit program exploits it/them

- how it/they could be fixed (by specific changes to the vulnerable program itself, not by system-wide changes like adding ASLR, stack canaries, NX bits, etc.).

A total of four exploits must be submitted to be considered for full credit. At least one of these **must** be a *buffer overflow*. Marks may be docked if you do not submit a buffer overflow exploit.

What to hand in

All assignment submission takes place on the `student.cs` machines (not ugster or the virtual environments), using the `submit` utility. In particular, log in to the Linux student environment (`linux.student.cs.uwa` go to the directory that contains your solution, and submit using the following command: `submit cs458 1 .` (dot included). CS 658 students should also use this command and ignore the warning message.

By the **milestone due date**, you are required to hand in:

sploit1.c, sploit2.c Two completed exploit programs for the programming question. Note that we will build your sploit programs **on the uml virtual machine**.

a1-milestone.pdf: A PDF file containing the exploit descriptions for sploit1 and sploit2 (including fixes, as explained above).

Note: You will not be able to submit sploit1.c, sploit2.c, or a1-milestone.pdf after the milestone due date (plus 48 hours).

By the **assignment due date**, you are required to hand in:

sploit3.c, sploit4.c: The two remaining exploit programs for the programming question.

a1.pdf: A PDF file containing your answers for the written-response questions, and the exploit descriptions for sploit$\{3, 4\}$ (including fixes, as explained above). Do not put written answers pertaining to sploit1 or sploit2 into this file; they will be ignored.

Note: The 48 hour no-penalty late policy, as described in the course syllabus, applies to the milestone due date and the assignment due date. It does not apply to the blog task signup due date.

For the milestone, the TAs will continue to answer questions on Piazza after the milestone due date (and through the 48 hour extension period). However, they will no longer answer questions after the assignment due date (and *not* through the 48 hour extension period).

Useful Information For Programming Sploits

The first step in writing your exploit programs will be to identify vulnerabilities in the original `pwgen.c` source code.

Most of the exploit programs do not require much code to be written. Nonetheless, we advise you to start early since you will likely have to read additional information to acquire the necessary knowledge for finding and exploiting a vulnerability. Namely, we suggest that you take a closer look at the following items:

- Module 2
- Smashing the Stack for Fun and Profit
  (http://insecure.org/stf/smashstack.html)
- Exploiting Format String Vulnerabilities (v1.2)
  (http://julianor.tripod.com/bc/formatstring-1.2.pdf, Sections 1-3 only)
- The manpages for execve (man execve), pipe (man pipe), popen (man popen), getenv (man getenv), setenv (man setenv), passwd (man 5 passwd), shadow (man 5 shadow), symlink (man symlink), expect (man expect).
- SSH public key authentication
  (e.g., https://cs.uwaterloo.ca/cscf/howto/ssh/public_key/, Ignore the PuTTY part for this assignment)
- Environment variables
  (e.g., https://en.wikipedia.org/wiki/Environment_variable)

You are allowed to use code from any of the previous webpages as starting points for your sploits, and do not need to cite them.

GDB

The gdb debugger will be useful for writing some of the exploit programs. It is available in the virtual machine. In case you have never used gdb, you are encouraged to look at a tutorial (e.g.,http://www.unknownroad.com/rtf

Assuming your exploit program invokes the `pwgen` application using the `execve()` (or a similar) function, the following statements will allow you to debug the `pwgen` application:

1. `gdb sploitX` (X=1..4)
2. `catch exec` (This will make the debugger stop as soon as the `execve()` function is reached)
3. `run` (Run the exploit program)

4. `symbol-file /usr/local/bin/pwgen` (We are now in the `pwgen` application, so we need to load its symbol table)
5. `break main` (Set a breakpoint in the `pwgen` application)
6. `cont` (Run to breakpoint)

You can store commands 2-6 in a file and use the "`source`" command to execute them. Some other useful gdb commands are:

- "`info frame`" displays information about the current stackframe. Namely, "saved eip" gives you the current return address, as stored on the stack. Under saved registers, eip tells you where on the stack the return address is stored.
- "`info reg esp`" gives you the current value of the stack pointer.
- "`x <address>`" can be used to examine a memory location.
- "`print <variable>`" and "`print &<variable>`" will give you the value and address of a variable, respectively.
- See one of the various gdb cheat sheets (e.g., http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf) for the various formatting options for the print and x command and for other commands.

Please bear witness to the fact that the version of GDB used in the UML has a bug. Attempting to "`print optarg`" will print the wrong value. See the following posts for details:

- https://stackoverflow.com/questions/9736264/using-getopt-with-gdb
- https://stackoverflow.com/questions/35787697/why-does-gdb-get-wrong-optind-variable-value

Note that `pwgen` will not run any program or command with root privileges while you are debugging it with gdb. (Think about why this limitation exists.)

The Ugster Course Computing Environment

In order to responsibly let students learn about security flaws that can be exploited, we have set up a virtual "user-mode linux" (uml) environment where you can log in and mount your attacks. The gcc version for this environment is the same as described in the article "Smashing the Stack for Fun and Profit"; we have also disabled the stack randomization feature of the 2.6 Linux kernel so as to make your life easier. (But if you'd like an extra challenge, ask us how to turn it back on!)

To access this system, you will need to use ssh to log into your account on one of the `ugster` environments: `ugsterXX.student.cs.uwaterloo.ca`. There are a number of ugster machines, and each

student will have an account for one of these machines. You can retrieve your account credentials from the Infodist system.

The ugster machines are located behind the university's firewall. While on campus, you should be able to ssh directly to your ugster machine. When off campus, you have the option of using the university's VPN (see these instructions), or you can first ssh into
`linux.student.cs.uwaterloo.ca` and then ssh into your ugster machine from there.

When logged into your ugster account, you can run "`uml`" to start the user-mode linux to boot up a virtual machine.

The gcc compiler installed in the uml environment may be very old and may not fully implement the ANSI C99 standard. You might need to declare variables at the beginning of a function, before any other code. You may also be unable to use single-line comments ("//"). If you encounter compile errors, check for these cases before asking on Piazza.

Any changes that you make in the uml environment are lost when you exit (or upon a crash of user-mode linux). **Lost Forever**. Anything you want to keep must be put in `/share` in the virtual machine. This directory maps to `~/uml/share` on the ugster machines, which is how you can copy files in and out of the virtual machine. It can be helpful to ssh twice into ugster. In one shell, log into ugster start user-mode linux, and compile and execute your exploits. In the other shell, log into ugster and edit your files directly in `~/uml/share/`, so as to ensure you do not lose any work. The ugster machines are not backed up. You should copy all your work over to your student.cs account regularly.

When you want to exit the virtual machine, use exit. Then at the login prompt, login as user "halt" and no password to halt the machine.

Any questions about your ugster environment should be asked on Piazza.

Miscellaneous

Running your exploits while using ssh in bash on the Windows 10 Subsystem for Linux (WSL) has been known to cause problems. You are free to use ssh in bash on WSL if it works, but if the WSL freezes or crashes, please try PuTTY or a Linux VM instead.

There are bugs when using `vi` to edit files in the `/share` directory in the virtual environment. It is recommended to use `nano` inside the virtual environment, or even better, use `vim` on the ugster machine in a separate ssh session.

There seems to be a mistake in the man page for `getopt_long`. Parameters for the options have to be specified like the following: "`--seed=temp.txt`", not "`--seed temp.txt`". If you use the short form of the option, it must be like "`-etemp.txt`" (no "=" between).