CS458 Assignment 1
Insung Chung
i4chung
20576619

Written Response Questions

Question 1. Possible attack scenarios

a) Main-in-the-middle attack

Main-in-the-middle attack is the case where a client is in mind that one is communicating with the company service but is actually communicating to the service via a third-party mediator. For example, if an attacker alerts a user that one needs some action within University-C137 money transfer system with a link to a phishing website of the system, the subject will leak ones access information by logging into the phishing website.

In such scenario, this results in a compromised confidentiality, and the user may end up in financial losses.

b) Side channel attack

A possible side channel attack would include a reflection of personal information. A user may utilize company's money transfer service not knowing that one's monitor is being watched from behind or through reflection of nearby object surface.

This will result in compromised confidentiality, as this leaks personal information, or even authentication information to the attacker.

c) Privilege escalation

A possible privilege escalation attack may be done by buffer overflow attack. The attacker conduct fuzzing on company service, and detect buffer overflows, which then can be used to replace existing program instructions with desirable ones.

At worst case, the attacker can get their hands on the root privilege, which violates confidentiality and possibly integrity and availability as the attacker has full control over the device that runs the service.

Question 2. Approaches to conduct each phase of development

Code review

- I would suggest with guided code reviews rather than using techniques like "easter egg" code reviews as the service hasn't been maintained properly for a long time, it is expected to have a lot of side effects to begin with. The code review will be done per each change of features made by individual developers. The developers will

have a meeting and explain, cooperatively inspect the algorithm and codes to to find flaws or better approach to the problem before it can be merged to communal branch.

Testing

- The service must conduct white-box regression testing on each update to check two things: make sure the service does what it is supposed to do; make sure the service does not do what it is not supposed to do.

- For each new features, it must come with unit tests that inspects of above two categories.

- Every update must pass all unit tests of the module before it can be pushed in

Maintenance

- Code reviews and testing are not a one-time job. In order to limit the number of flaws, product development must be formalized with establishment of standards and audits.

- Development process will follow 5 step sprint agile principle; sprints with 5 phases: planning, implementation, review, deployment, retrospective. Each sprint will be 2 weeks long. Code review and testing are done under review phase.

- Audits from external organization would be preferable but having such audit maybe costly and not always accessible. Thus, the company will allow audits of development procedure by supervisors of each department. For each development cycle, audits need to submit a development guideline checklist based on agile principle, preferably with comments on potential improvements or explanations on unsatisfied entries.

Question 3. Reinforce the defense mechanisms.

present how they could be used to defend against any one of the attacks mentioned

provide one attack from question 1 as an example

- Prevent. In case of a Side channel attack, the service can authenticate user using biometrics so that reflections of the scree no longer reveal user's authentication information.

- Deter. In case of a Privilege escalation, the service can use virtual memory mapped to randomized physical addresses so that exploiting buffer overflow vulnerability to act as desired behavior is difficult.

- Deflect. In case of any attack methods above, the service can have a limit on the amount money one can transfer at a time and day. In this case, the attacker will find the service as less attractive target since they would not be profited much even if they successfully exploit the service.

- Detect. In case of stolen authentication from Main-in-the-middle attack, the exploit can be detected by sending a confirmation message to the user that a login from a new device is detected.

- Recover. In case of system corruption after the attacker exploited privilege escalation, the service can be recovered by having a backup image of service data on remote drive.

Question 4.

a) 3 times for authentication chances is small enough that user will often be locked out of their own device by accident. Also, password based single factor authentication is vulnerable as there are numerus ways for the attacker to get their hands on user's passwords

b) 2FA with more authentication chances will prevent from getting accidently locked out of one's own device. Furthermore, it would make more difficult to exploit the device as the attacker needs to obtain both of the authentication keys to login as the user.

Question 5. Malware analysis

a) CryptoLocker

type: ransomware or scareware

infection: spoofed e-mail attachments from a botnet

effect: Encrypted victim's hard drive

b) Stuxnet

type: worm

infection: 4 different zero-day attacks or manual installation via USB drive

effect: subtly changes the frequencies of installed variable-frequency drives operating between 807–1210 Hz so that

distortion and vibrations occur resulting in broken centrifuges.

c) CIH

type: virus

infection: spread using executable files, infecting other executable files

effect: DoS on victim device by corrupting BIOS

Programming Questions

sploit1.c – Class B | TOCTTOU

Exploit method:

When pwgen asks for an entropy, replace entropy input file with shadow file. This will insert whatever entropy input directly into the shadow file, overriding user passwords

Fixes:

This method of exploit can be prevented by checking if you are writing on the right file just before accessing the file itself.

sploit2.c – Class B | Buffer overflow

Exploit method:

overflows `char reason[512];` in get_reason() function and overrides `char stats_file[1024];` to the address of shadow file override root password with root::::::::

Fixes:

This exploit uses a vulnerability in pwgen.c:124. When writing into `reason[512]` buffer, it loops up to 1023 times, causing it to write over `stats_file[1024]` buffer. Simple change on the range of the loop will fix the vulnerability.

sploit3.c – Class A | Buffer overflow + TOCTTOU

Exploit method:

Targeting return address

1. get_entropy(buffer) called by fill_entropy() writes 1025 characters in 1024 byte buffer with buffer[1024]='₩0'.
2. This makes the caller's(parse_args()) value of previous $EBP from 0xffbfdc98 to 0xffbfdd00
   i.e. EBP is dragged down by 152.
3. So return address of parse_args = new EBP - 4 = old EBP + 148
4. Since args is 86 bytes(but takes up 2 more bytes since a word is 4 byte) and buffer is 512, new return address is at around: old buffer[(512+88)-148] = old buffer[452].

Dealing with side effects of overflow

1. changing EBP of parse_args() causes its parameters out of their positions
2. So, after fill_entropy(), when it loops back to check for the next command line argument, it will face

segmentation fault.

3. So, we need to set new `argc` to 0
   3.1. From GDB, &argc == $ebp+12 instead of $ebp+8 for some reason. This will be taken as granted
4. SO, new `argc` is in old buffer[(512+88+12)-152] = old buffer[460]

How to write to buffer?

1. buffer is used only when
   1.1 parsing `-e` argument and
   1.2 without `optarg` and
   1.3 check_perms() returns false
2. check_perms() returns false when /tmp/pwgen_random is not deleted after unlink("/tmp/pwgen_random")
3. This can be done if there is a directory named "/tmp/pwgen_random".
4. pwgen copies argv[0] into the buffer.
   i.e. execve(<pwgen address>, [<payload>, "-e", NULL], NULL) will fill in buffer with payload

Additional side effect

We need to write into buffer first, so that the memory is under compromised state once fill_entropy() spoils EBP. AND in order to write into buffer, we need to make check_perms() false. But if check_perms() is false, than fill_entropy() will not be called.

Possible solution: TOCTTOU

The problem is that once /tmp/pwgen_random directory is placed, parsing -e will no longer require user response, and will immediately print error message, proceeding to the next argument. Since we need to set the buffer first, then type in 1024 length entropy, we need to somehow make -e parsing fail first, then make successful -e parsing afterwards. We can think of a brute force method of making this happen. If we have abundant of '-e' commend line arguments passed to pwgen program, it will take a long time to parse those arguments, and we can remove the /tmp/pwgen_random directory before pwgen parses it's last '-e'.

Summary: Exploit scenario

1. remove file /tmp/pwgen_random and create directory /tmp/pwgen_random
2. make a string with following requirements and attach it to argv[0]
   2.1. *(long *)old buffer[452] includes pointer to NOPs. - ****this pointer won't be accurate.
   2.2. *(int *)old buffer[460] includes 0 so that we won't face seg fault
   2.3. shellscript section. insert it far behind in the buffer as possible
   2.4. calculate for its starting address so that we can pass it to *(long *)old buffer[452]
   2.5. fill rest of the buffers with NOPs.
3. run execve(target, [payload, -e, ..., -e, NULL], NULL)
   3.1. this will fill in the buffer with payload and start looping through all -e's

4. run rmdir("/tmp/pwgen_random")
   4.1. if this was done before pwgen parsed all '-e', this will restore check_perms()'s functionality
5. enter 1024 inputs to overflow entropy buffer - this overflows EBP
   5.1. the return address of the parse_args() now points to the payload
   5.2. argc of parse_args() is now 0, so parse_args() terminates the parse loop
6. parse_args() returns to payload executing shall with root privilege

Fixes

Fixing the vulnerability is much simple in comparison to the exploit method. The exploit uses the off-by-one overflow at pwgen.c:179. Such vulnerability can be fixed by changing the range of the loop from 1024 to 1024-1.