CS 458 A1

Tom Yan

Written Response Questions

1. (a) Integrity/Availability since the data recorded is not necessarily the "right" data (integrity) and the voter cannot determine if their vote was actually cast for their candidate (availability)
   (b) Availability since the system is not easily available for many students
   (c) Confidentiality since someone else had access to their voting data
   (d) Integrity since the data recorded contains false data
2. (a) interruption since there is an absence of the records
   (b) fabrication since they are making something that didn't happen
   (c) modification since someone may have altered the website to perform an additional look up
   (d) interception since the call is being intercepted
3. (a)
   (b) Use encryption and 2 factor authentication whenever a vote is being cast (ie. each unique vote is tied to SSN or fingerprint) so that it will be much more difficult for CipherIsland to perform the attack since they have to forge these credentials.
   (c) Have location tracking and logs in place for requests that you can easily notice suspicious activity. If there is large traffic coming from similar locations in a small time frame it is probably the CipherIsland hackers.
   (d) After detecting the activity, run a script that removes the fake CipherIsland votes in the machines and only preserve the real ones.
4. We can use signature-based analysis to look through the voting machines. It will scan the code in the voting machines to see if it has any of the signatures (infection code or payload code) of Bleed1, Bleed2, and Bleed3. It can also try to compare the behaviour of the voting machines to that of the known malware programs. The behaviour-based analysis will look for suspicious patterns of behaviour, and can run the suspicious code in a sandbox, isolating it from the rest of the clean code. That way there is a better chance of identifying the malware code instead of it being a false positive.

Exploit 3

1. This exploit is done using the run_cmd function in show_confirmation(). We can overwrite the "find" executable with a script that executes the shell (bin/sh). Then using the escalated privileges of submit, when the program gets to run_cmd("/user/bin/find") it will run our shell script which gives us a root shell.
2. The environment variable storing the USER gets overwritten with ../../bin, which lets the submit program use the path as the submit directory and the destination directory. We then write to a file named "find" the location of the shell script (bin/sh), but pad it with the right number of '/'s so that it skips the regular expression check in check_for_viruses. Then when the submit program executes, the copy_file function replaces /usr/bin/find with our own shell script. Before the program ends, it will run show_confirmation, which will execute run_cmd("/user/bin/find"). But that is the file we replaced with the shell script, so it will give us a root shell as required.
3. The vulnerability can be fixed by not having the USER environment variable be part of the submit path. This way no one can modify the submit directory and destination directory to overwrite system files. Alternatively, the function get_submit_dir can perform error checking on the USER environment variable to make sure it doesn't contain any slashes or dots.

Exploit 4

1. The vulnerability is in the strncat function found by objdump. It lets us overwrite 1 byte past the status_msg thereby overwriting the unsafe variable which lets us use slashes in the path argument. This exploit is done using the run_cmd function in get_submit_dir(). We can overwrite the "mkdir" executable with a script that executes the shell (bin/sh). Then using the escalated privileges of submit, when the program gets to run_cmd("mkdir") it will run our shell script which gives us a root shell.

2. The environment variable storing the USER gets overwritten with ../../../bin, which lets the submit program use the path as the submit directory. We then write to a file named "mkdir" the location of the shell script (bin/sh). The file path is able to contain slashes as long as it's long enough since that's where the vulnerability lies. Then when the submit program executes, the copy_file function replaces bin/mkdir with our own shell script. We fork the process so the next process will get to run the modified mkdir giving us a root shell as required.

3. The vulnerability can be fixed by using strncpy instead of strncat like the original submit program. That way there is no chance of 1 byte overflow when the string passed in is long. Alternatively, it can be fixed by not having the USER environment variable be part of the submit path. This way no one can modify the submit directory and destination directory to overwrite system files.