Assignment 1

CS 458 Computer Security and Privacy

Exploit 2: Format String Vulnerability (4 marks)

Description of Vulnerability

Format string vulnerability arises from the fact that format functions in C does not check the number of arguments against the number of parameters in the format string. Thus, if an application does not properly validate user input, and subsequently use it as a format string, it can allow users to execute arbitrary commands.

More specifically, when a format string is parsed, the format function expects additional arguments as input for the specified format parameters. And when no arguments are supplied along with the format string, it incorrectly reads from or writes to the stack.

Explanation of Exploit

In this case, we target the printf function call in check_forbidden to overwrite DTOR_END with the address of shell code. This will result in the application opening a root shell when it terminates and calls the destructor.

We start by taking note of the destructor end address with:
objdump -h /usr/local/bin/submit | grep dtors
Then, we install an egg (shell code in this case) in the environment in which we will be executing submit and take note of its address.

To overwrite an arbitrary address, we use the format parameter "%n" and prepare an format string in the following format:
AAAABBBB%nnnnnnx%iii$hn%mmmmmmx%jjj$hn
Where AAAA and BBBB are the address of DTOR_END, nnnnnn and mmmmmm are calculated based on the egg address, and iii and jjj are offsets to locate the format string on the stack.

As an example:
When the DTOR_END address is **0x0804a1f8:**
AAAA = \xf8\xa1\x04\x08
BBBB = \xfa\xa1\x04\x08

When the egg address is **0xffbfdfb8**, the corresponding decimal values are:
0xffbdf = 65471
0xdfb8 = 57272

Then:
nnnnnn = 57272 – 8 = 057264
mmmmmm = 65471 – 57272 = 008199

With some trial and error:
iii = 111
jjj = 112

When this format string is executed, it increments the counter for number of bytes written via the %nnnnnnx command, and writes to the address found at offset iii with command %iii$hn, which will be address AAAA we supplied. The second half of the format string operates similarly.

Description of Fix

There are two simple ways to fix the vulnerability. First, we can refrain from treating the user's input string as a format string, and use format function in the following fashion:
printf("%s", source)

On the other hand, we can also perform additional checks on the user input to ensure no format string parameters exists. And in cases where they are reasonable, we need to make sure sufficient number of additional arguments are supplied.

Exploit 3: Path Interception Vulnerability (4 marks)

Description of Vulnerability

When executing a file from a C program, vulnerability arises if only the file name is specified instead of the full absolute path. This enables attackers to modify the environment such that a modified executable is discovered first and run as a result.

Explanation of Exploit

In this case, we target the show_confirmation function, which calls run_cmd with an unqualified file name "ls". To exploit this, we create a simple file which contains a call to /bin/sh, name it "ls", and place it in the home directory. Then, we execute "submit –s" with a modified PATH variable with the home directory prepended. As a result, the modified ls is executed and a shell is opened as a result.

Description of Fix

To fix this vulnerability, one simply needs to always use full absolute path when referencing executables. In a security-critical situation, one may want to enforce a clean environment by resetting the environment variables at the start of execution, or verifying that no malicious modifications exists. Lastly, one can verify the true path and modification time of the file to be executed to ensure no modification has been made.

Exploit 4: TOCTTOU Vulnerability (2 marks)

Description of Vulnerability

The class of vulnerability named TOCTTOU is based on race condition errors that occur due to discrepancy between time-of-check state and time-of-use state. This discrepancy can come in many forms: modified file, modified permission, or changed request.

Explanation of Exploit

In this case, we exploit the delay between get_logfile_name (which creates a log file when it is absent) and log_message (which logs to the log file). We discovered that the application does not place any protection on the created log file, and thus is vulnerable to a redirection.

In the exploit, we start by spawning a parallel child process. The parent process is designed to monitor the creation of a log file, and once it is created, change it to a symlink to “etc/passwd”. In the child process, we execute submit in parallel, with an engineered log message that specifies a root account with no password.

Furthermore, the exploit program uses a large plaintext file of arbitrary content as input to submit, to slow down the execution of check_for_viruses (which copies the entire file in chunks), and ensure the child process can be pre-empted by the parent process before it reaches log_message.

The result is the specified message payload being written to “etc/passwd”, and the only step remaining is to change user to root and spawn a shell.

Description of Fix

A race condition vulnerability in general is best fixed by eliminating or severely reducing the time delay between check and execution. If security is critical, a lock should be used to guarantee no malicious access to the resource. In this case, the simplest solution is to check for the real path of the log file again, before writing into it.