

Team names: HolyNest1#

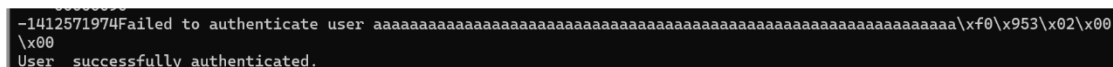
Team members: Heminghao Deng, Xinhao Feng, Chenyang Zhu, Yinhe Chen, Dian Peng

## 1 Exploitation Method(if you need more information, you can contact us):

We observe that the `__libc_csu_init` function has a potential stack overflow vulnerability. We can exploit this by overwriting the stack frame's return address. To do this, we need to determine the offset of local variable. Using the `objdump -d login-subverted` command, we can obtain the assembly code of the `login-subverted` file and find the variable offsets.

Moreover, If the function's base address is invalid, the program will trigger a segmentation fault during execution. Therefore, we need to obtain the function's base address. Notice that there is a variable named `violet` in `main` function, which stores the value `0xabcdcdca`, hence We can use `/proc/<pid>/maps` to find the virtual address range of the program's stack frame, then dump the memory of `login-subverted` while running and locate this variable, additionally, by looking for its offset we can determine the function's base address.

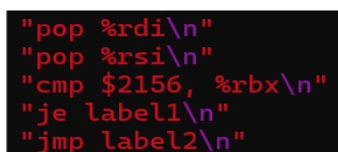
Finally, if we do not pass an address of a readable segment as a parameter to the authenticated function, it will also trigger a segmentation fault. In our remaining assembly code, we simulate part of the `__libc_csu_init` function's functionality. Thus, we can first set the return address to a specific line of the assembly code to modify the register values, and then call the authenticated function to successfully exploit the vulnerability.



```
-1412571974Failed to authenticate user aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xff0\x953\x02\x00
\x00
User successfully authenticated.
```

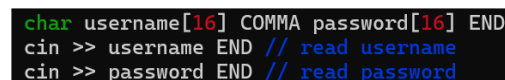
Figure 1 successfully exploit(We'll send the exp to you by Teams)

## 2 where the vulnerabilities are



```
"pop %rdi\n"
"pop %rsi\n"
"cmp $2156, %rbx\n"
"je label1\n"
"jmp label2\n"
```

Figure 2 Gadgets which can be used for  
controlling registers



```
char username[16] COMMA password[16] END
cin >> username END // read username
cin >> password END // read password
```

Figure 3 code that trigger stack overflow

## 3 Why is our login-subverted program's backdoor difficult to detect

we separate the vulnerability and the hint in the code so they will not easily be associated with each other. Additionally, we obfuscate the hint variable by using inline assembly, making it difficult for an adversary to determine its true value. This approach also diverts the adversary's attention and makes it harder for them to realize that the variable's actual purpose is to provide a method to locate the program's stack base address. Moreover, For most people, realizing that a stack overflow can be exploited to modify the return address to call a dangerous function is highly unlikely. Finally, the exploitation of this backdoor requires a substantial amount of knowledge. The learning curve and the difficulty of exploitation are very high.