

CS179 Final Project Report

CVE-2016-0728

Introduction

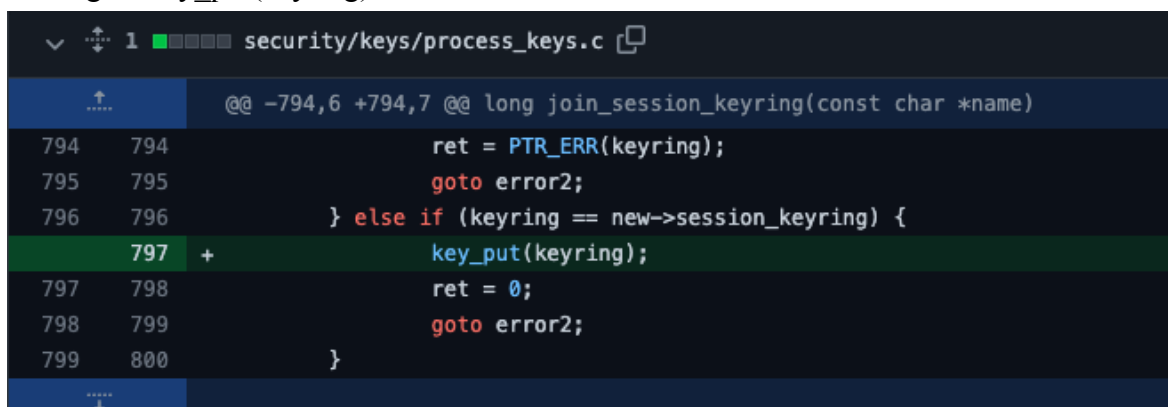
CVE-2016-0728 centers around the keyrings utility to execute an exploit from a use-after-free (UAF) vulnerability. It exploits the keyctl structure within linux and the various functions within it that contain access to kernel memory. This report details the tools used for the exploit, where the vulnerability is present, and how to achieve privilege escalation using the methods documented through CVE-2016-0728.

Vulnerability Description

The vulnerability revolves around the use of a tool known as keyrings and the system calls associated with this utility. Keyrings are tools used by android and linux operating systems to store private user data such as passwords, authentication codes, and encryption keys.

The vulnerability lies in how certain linux kernels (versions 3.8.1 to 4.4.0) don't contain complete safeguards to prevent overflow for the usage count referencing these keyring objects. Upon invoking an instance of a keyring object, it can be manipulated at the user-level to contain an amount of references that exceed the capacity of the integer holding the refCount of the keyring. Using this manipulation, the kernel can falsely believe that there are no references to the keyring object and we would essentially free the keyring object while holding its reference.

The bug can be patched by simply allowing a duplicate keyring with a false reference to be freed through a `key_put(keyring)` function call.



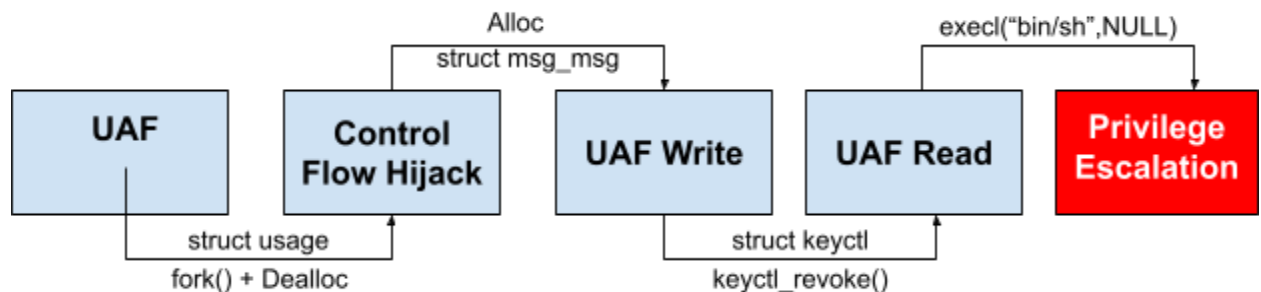
```
▼ 1 security/keys/process_keys.c
@@ -794,6 +794,7 @@ long join_session_keyring(const char *name)
794 794         ret = PTR_ERR(keyring);
795 795         goto error2;
796 796     } else if (keyring == new->session_keyring) {
797 +         key_put(keyring);
797 798         ret = 0;
798 799         goto error2;
799 800     }
```

Figure 1: The patch for refCount bug [\(1\)](#)

Exploit Description

The exploit functions by manipulating the kernel memory into replacing a freed keyring object with code from the userspace. This is achieved by holding a reference to a keyring object stored in the kernel and subsequently releasing that same object by overflowing the refCount using the integer overflow method. Through the use of the vulnerability, the refCount can be overflowed and the memory within the keyring would be freed upon forking the process twice.

The exploit can then override the freed object with another kernel object through the use of the Linux IPC subsystem and the msgsnd function to send several objects of size keyring to kernel memory. At this point, the keyring can now be deallocated through the use of a keyctl_revoke function call. This will achieve privilege escalation from the userspace and calling execl("/bin/sh", NULL) will grant the userspace control of the root shell.



Exploit Adaptation

The original exploit code increments [\[2\]](#) the refcount of a session from 1 to overflow, so that we can let it reach 0. We still do this; however, we utilize two scripts [\[2\]](#) [\[3\]](#). Two session's refcount are incremented all the way to -4 through the main code [\[2\]](#) which had been altered to not increment after -4. Edit the testbed such that when calling revoke, it also calls the shell. There was an issue where after calling revoke, the UID/EUID would be 0, but be unable to open the shell.

We run the testbed and increment one of the sessions to 0 by forking, then heap spray once, not closing the keyring session. Then for the other session, run the testbed again, fork until it reaches 0, then heap spray and revoke to gain privileges and run the shell (Note: calling heap spray twice works for one session, but it is less reliable).

Another approach to this is once both session's refcount are at -4 (or <0), modify the original exploit [\[2\]](#), such that it increments, then forks, and then writing to the memory of size 2048*2, and then exiting (note this does not close the session). Then use the testbed, edited once again so that calling revoke also calls the shell, to let the other session reach 0 by forking, and then calling heap spray once then calling the revoke function. Overall, two sessions are needed as using one does not call the code when revoking, which the PoC suggests.

Troubleshooting Process

Throughout the process of running the exploit, there were many difficulties due to a lack of proper and verified documentation for CVE-2016-0728. Many of the errors we could resolve come from various debugging methods.

After setting up an environment to run the exploit (Ubuntu 14.04 with linux kernel 3.13.0 on a 64-bit machine) we were unable to properly compile the exploit provided in the original poc documentation [2]. To fix this, we had to analyze the hardcoded values within the exploit, prepare_kernel_cred and commit_creds. Once this was modified, we opted to use a testbed provided by [3] to analyze each step of the exploit.

This largely assisted us in finding which individual steps of the exploit caused certain issues such as segmentation faults upon revoking a referenced keyring. Once the refCount was successfully overflowed to value 0 using the testbed, we had to tackle the issues with gaining kernel code execution.

We had initially snapshotted the state of the virtual machine to observe what occurs post-overflow, but this didn't provide enough information. From here, we ran gdb simultaneously to observe the memory locations of the heap spray called to see if it matched the freed keyring reference in kernel memory. Once we discovered it didn't, we had to try different linux kernel versions to fit this parameter.

Once we tried linux kernel 3.18.25 and changed the boot parameters to fit the requirements for the exploit we were able to successfully compile and run it on the testbed following the steps to overflow, heapspray, and revoke after some debugging.

Lessons Learned

By working on this project and picking CVE-2016-0728, we had gained a lot of insight into how vulnerable linux operating systems truly are. The simplicity of this UAF vulnerability and patch fix shows the various overlooked aspects of the OS, even if this was around something as notably secure as a keyring in linux.

Another lesson we could take away from the project was how relevant documentation really is in identifying and recreating an exploit for any purpose. When we had really dug into the roots of this CVE, we discovered that the original authors most likely utilized a gdb to control various aspects of the exploit including the refCount and the data stored within the freed keyring object. Therefore, the parameters provided in the documentation including the specific linux kernel, weren't capable of running this exploit without a few bugs.

Overall, we learned how throughout research from the start and diligence can play a significant role in how the project should be handled.

References

[1] <https://github.com/advisories/GHSA-pm5x-hxtr-68w7>

[2] <https://gist.github.com/PerceptionPointTeam/18b1e86d1c0f8531f8f>

[3] <https://github.com/neuschaef/cve-2016-0728-testbed>