Windows Kernel - Shellcoding : Token Stealing

DEGAIL Dylan[1]

[1]Major in Cybersecurity, School of Computer Science, ENSEEIHT, France

dylan.degail@etu.toulouse-inp.fr

Abstract -

This paper shows the different steps for the creation of a Windows kernel challenge. The building of the challenge is divided into the creation of the architecture, the client application set-up to communicate with the vulnerable driver and the conditions for the shellcode development. The player receives the details connection to start the challenge and tries to interact with the client application and create a shellcode that requires specific conditions. The core of the challenge is to develop a perfect shellcode to perform a privilege escalation and capture the flag. This challenge is to push people to learn more about how the Windows kernel works and how the communications operate between the userland and the kerneland through drivers.

Keywords -

Windows; Kernel; Kerneland; Userland; Driver; Token; Shellcoding; Permissions; IOCTLS

1 Introduction

This document will treat the different steps to set up this challenge about Windows driver kernel exploitation.

Firstly, to create this challenge, comprehending this complex world and training on an existing challenge were necessary steps. This was a crucial phase to be able to develop my challenge, especially when you want to create a variant.

Secondly, I will start with how I created the architecture and what tools I choosed to set up the environment.

Thirdly, I will explain the development of the client application and how the player will communicate with it.

Next, the shellcode's specificities will be developed to show how the player can create it to solve the challenge.

Finally, I will conclude with how the experience was.

2 Account of my work

In this part, the account of my work during the project will be divided into different parts.

2.1 Understand and preparation

Firstly, the main work was to understand how a kernel driver works on Windows and how a client application communicates with this driver through input/output controls (IOCTLS). These controls allow a userland program to communicate with the kernel driver. I learned the first pieces of knowledge I needed to know through the resolution of a Root-Me challenge [1].

Moreover, this challenge presented a method for privilege escalation by stealing the token of the SYSTEM process [2].

So, the first part of the challenge creation was to understand this complex world around the Windows kernel, how communications work, and an exploitation method to achieve a privilege escalation.

2.2 Architecture and tools

To set up the challenge, I used a Docker container because it's easy to deploy. I needed something to virtualize a Windows and therefore QEMU was a good choice. In fact, to avoid a blue screen of death (BSOD), a level 2 virtualization was crucial to emulate a different kernel from the host.

A Windows 10 64-bit version 10.0.19045.4046 was specifically selected for this challenge.

After that, I registered and started the vulnerable driver on Windows with the OSR Loader Driver tool.

To facilitate the player, I also installed CYGWIN which is a toolbox to obtain a Linux-like environment on Windows.

Moreover, this tool gave me freedom on the bash shell I will give to the player to solve the challenge.

Finally, I have configured an SSH (Secure SHell) service to allow the player interacts with this environment through an SSH connection.

2.3 Development of the client application and communications

The core of the challenge creation is to have a client application that gives a way for the player to communicate with the vulnerable kernel driver. Additionally, this is one of the chains I can play with (also the shellcoding part) to create a difficult challenge by modeling it in my way.

So, to create a distinct challenge from the Root-Me one, I built a client application that took in argument an IOCTL and optionally a shellcode as input.

Indeed, I was restricted by the IOCTLS the driver gave me because of reusing the driver of the Root-Me challenge. But, it was enough for the challenge. I had only the ability to allocate kernel memory space, copy data from the input of the client application to the allocated memory, and execute it afterward.

In this way, I can play with the shellcode that the player can give me to increase the difficulty of the challenge. The details will be in the next part.

To achieve this, the client application is developed in C++ and compiled with Visual Studio. After that, It was added to the path of the CYGWIN environment to let the player interact with it.

2.4 Shellcode specificities

I will finish the account of my work with the creation of the specificities of the shellcode.

To start, to avoid copy-pasting of existing shellcodes about this exploitation, I added a size limit for the shellcode. Consequently, the player has to create a shellcode for the specific Windows version which is used for the challenge. He can check on websites, like the Vergilius project [3], which offsets he needs to get the _EPROCESS structure and so the token field.

To add difficulties, I created a sanitizer to bypass some byte patterns such as using the SYSTEM pid to steal a system-level token or using the default path to access the _EPROCESS structure of the current process.

To do it, I added some regex (regular expressions) rules to detect a sequence of specific bytes. It is applied when the client application gets the shellcode in input. If the sanitizer detects aimed patterns, I reject the shellcode and it will not be copied to the allocated memory space.

These specificities were a good way to push the player to have a complete understanding of the challenge.

3 Conclusion

To conclude about this challenge's creation, it was a pleasing experience to learn new skills in Windows kernel driver. Even though it was interesting, I had some difficulties with the solving of the challenge. I had to ask help from people who already resolved the challenge to give me some hints.

Moreover, the architecture was quite consequent to set-up, so a lot of time was lost to prepare it well.

One possible improvement for this challenge would be to create my own "vulnerable" driver. Indeed, because of the short time and the complex task, I preferred to take the one from the Root-Me challenge.

Finally, to start to solve the challenge, the player needs to use the connection details. They can now try their best to solve it.

References

[1] *Root-Me Windows Kernel Challenge*. https://www.root-me.org/fr/Challenges/Programmation/WinKern-x64-shellcoding-vol-de-token, Good challenge about vulnerable Windows kernel driver to understand how the Windows kernel works and how to communicate with kernel driver through IOCTLS.

[2] *Starting with Windows Kernel Exploitation – part 3 – stealing the Access Token*. https://hshrzd.wordpress.com/2017/06/22/starting-with-windows-kernel-exploitation-part-3-stealing-the-access-token/, Method to perform a windows kernel exploitation by stealing the access token to a high-level permissions process.

[3] *Vergilius project*. https://www.vergiliusproject.com/, This project provides a collection of Microsoft Windows kernel structures, unions and enumerations.