DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

CVE-2024-22830

Vulnerability Analysis Report

Summary

DEFENCE TECH

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

CORRADO AARON VISAGGIO
Group Chief Scientist Officer & Malware Lab Director
a.visaggio@defencetech.it

DEFENCE TECH

2

Executive Summary

2. Executive Summary

Kernel drivers are one of the most critical components of the modern operating system model. In Windows systems their privileges are higher than those of the Administrator user and as such are a prime target for attackers. Abusing kernel-level privileges is a common technique used by rootkits to gain complete control over a system.

In the Windows driver model, drivers are considered a trusted part of the system and as such they can only be loaded if they have been approved and digitally signed by Microsoft. This design greatly reduces the possibility of attackers executing arbitrary kernel code even when they managed to escalate their privileges to Administrator.

However, there are ways to bypass this restriction, the most common one is to abuse a vulnerability in a legitimate signed driver. Attacking a vulnerable driver means that the attacker may only be able to execute a subset of the kernel functions, but this is often enough to achieve their goals.

One kind of attack technique that relies on vulnerable drivers is called BYOVD (Bring Your Own Vulnerable Driver). In this scenario, after an initial infection, the attacker deploys a legitimate but vulnerable driver to the target system. Then, this driver is used to gain kernel privileges needed, for example, to disable security products using "EDR-killer" solutions. Deploying a driver requires Administrator privileges, as such BYOVD attacks are often used as a post-exploitation technique.

These attacks have been documented in the wild, recent examples are Kasseika Ransomware[1] and Lazarus Group[2].

This report describes CVE-2024-22830, a vulnerability discovered by the Malware Lab team in the Anti-Cheat Expert[3] "ACE-BASE.sys" kernel driver. This is a commercial "anti-cheat" solution that is distributed as part of several popular online games in China and a few in the West.

[1] *https://www.trendmicro.com/en_no/research/24/a/kasseika-ransomware-deploys-byovd-attacks-abuses-psexec-and-expl.html*

[2] *https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/*

[3] *https://www.tencentcloud.com/products/ace*

2.1 Impact

Exploitation of CVE-2024-22830 can allow a local attacker to escalate privileges.

In particular, on systems where the ACE-BASE.sys driver is already installed and running, an attacker can exploit this vulnerability to escalate privileges from a low privileged user account to Administrator or System.

On any other system, this driver can be used as a BYOVD vector to gain kernel privileges and terminate EDR products.

As part of our efforts in preventing cyber-attacks, we also disclosed this vulnerability to Microsoft through their vulnerable driver submission portal[4] and later to the LOLDrivers[5] project so that security vendors can update their definitions to detect and block this driver from being used in illegitimate ways.

[4] *https://www.microsoft.com/en-us/wdsi/driversubmission*

[5] *https://www.loldrivers.io/*

DEFENCE TECH

3 Analysis

3. Analysis

3.1 Introduction to kernel drivers attacks

Since drivers run in kernel mode they can't interact with the user directly, they usually communicate with user-mode processes through system calls.

User-mode processes communicate with drivers using IOCTLs (Input/Output Control) commands, these are requests that include an integer called "control code" that specifies the operation to be performed and optional input and output buffers that are used to pass data between the user-mode process and the driver.

A driver opts-in to receive commands from user-mode processes by creating a device object and binding it to a symbolic link that is visible to non-kernel callers. Drivers can further restrict access to their interfaces by setting security descriptors on the device object.

To communicate with a driver the user-mode process must first open a handle to its device object, and this is done using the CreateFile[6] function. Then, the handle is used to send IOCTL commands using the DeviceIoControl[7] function.

Exploiting drivers can be as simple as sending a valid IOCTL that performs an unchecked operation, for example, terminating an arbitrary process. Arbitrary process termination is a dangerous operation because it can be used to terminate processes that are designated as "Protected Processes" or "Protected Process Light" (PPL[8]), PPL is used by antimalware products to prevent their termination by attackers that have escalated their privileges to Administrator.

[6] *https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew*

[7] *https://learn.microsoft.com/en-us/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol*

[8] *https://learn.microsoft.com/en-us/windows/win32/services/protecting-anti-malware-services-*

3.2 Technical analysis

Note that since this driver is used to protect games from cheating, it makes use of obfuscation techniques to make it harder to reverse engineer. Any code snippet in this report is a deobfuscated version of the code, we will not delve into the details of how the obfuscation was defeated. Also, we will not be releasing the exploit code to prevent abuse of our findings.

During initialisation the driver creates a device object and links it to the path `\DosDevices\Global\ACE-BASE`, this allows user-mode processes to communicate with it by opening a handle to `\\.\ACE-BASE`. The device is created with a default security descriptor which allows any user to send IOCTL commands to it.

We will refer to `ACE-BASE` as the "main interface" of the driver, as we will see later there are multiple interfaces that are used to receive commands.

The main interface only accepts a small number of IOCTLs, some of them are used to initialise the rest of the driver while others seem to be used for debugging purposes. The most interesting control code is `0x221C24`, this command executes a second IOCTL-like interface that is authenticated and encrypted. We will refer to this as the "inner interface".

When receiving a command for the inner interface, the driver decrypts the input buffer using a custom algorithm and takes a 4-byte value that is used as command ID. Each request is authenticated by looking for a hardcoded value that acts as a password, this value is different for each inner command. The driver also validates the identity of the caller process as well as a timestamp with information that is passed in the input buffer. This is possibly to prevent replay attacks.

Once the command has been processed by the inner command handler, the output buffer is encrypted using RC4 before being sent back to the caller.

The commands provided by this interface are used to initialise further components of the driver and to register processes as games to be protected. It is important to note that sending unexpected commands to this interface in can result in a crash of the system.

Of particular interest is the command `0x6A7F3C53` which is used to initialise the global dynamically imported function table. Figure 1 is an example of how functions are dynamically imported by the driver.

```
wmemcpy(&v57.Length, L"⊠▯▯↻→ı↓↞→ı▯↑↞→ı◊", 13);
v9 = 54i64;
v10 = &v57;
v11 = 26i64;
do
{
  LOBYTE(v10->Length) = __ROL1__(LOBYTE(v10->Length) ^ 0x2B, 2);
  v10 = (v10 + 1);
  --v11;
}
while ( v11 );
RtlInitUnicodeString(&DestinationString, &v57.Length);
qword_140102780 = MmGetSystemRoutineAddress(&DestinationString);
```

Figure 1. Functions dynamically imported

One of the vulnerable commands we identified in this driver is the ability to terminate arbitrary processes, however, the relevant API call does not appear in the import table of the binary. This is because it is dynamically imported. We believe that this is a deliberate attempt to hide the functionality of the driver, for example to pass automated vulnerability analysis tools.

After the right sequence of inner commands is called, the driver is successfully initialised and ready to protect games, this can be observed by the creation of a second device object that is accessible under the path `\\.\{TF9AC12E-S60X-E25G-N67G-IC8A82086DAN}`.

This is the driver's command interface that exposes the dangerous functionality.

To communicate with the command interface, the caller must first register itself as a game using control code `0x6A7F3C60` of the inner interface.

Much like the main interface, the command interface is also protected by encrypting the input and output buffers, each with yet another encryption algorithm. Here the driver exposes 70 different commands, we only explored a subset of them to write a proof-of-concept exploit, but there are possibly other exploitable commands that are not investigated in this analysis. Figure 2 shows a few of the commands we analysed.

```
switch ( cmd_id )
{
  case 0x80002004:
    dispatcher = ioctl_get_const_value;
    break;
  case 0x80002008:
    dispatcher = ioctl_read_process_memory;
    break;
  case 0x8000200C:
    dispatcher = ioctl_open_process_as_user;
    break;
  case 0x80002010:
    dispatcher = ioctl_close_handle;
    break;
  case 0x8000201C:
    dispatcher = ioctl_virtual_query;
    break;
  case 0x80002028:
    dispatcher = ioctl_unimplemented;
    break;
```

Figure 2. Some of the labelled commands of the command interface

Among the commands we analysed, we found functionality to control processes and threads such as termination, suspend and resume, file access, unrestricted access to processes' virtual memory and the system physical memory.

In particular, for the sake of our proof of concept, we used the *0x80002034* command to terminate Antimalware-protected processes and the *0x80002064* command to open a handle to any SYSTEM process and inject code into it.

This was possible because command *0x80002064* uses *ObOpenObjectByPointer*[9] by setting the *AccessMode* parameter to *KernelMode*, this signals to the function that the caller is trusted, and it does not need to perform access checks. However, the resulting handle is directly returned to the user-mode client. This behaviour is documented in Figure 3.

[9] *https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-obopenobjectbypointer*

```
Object = lookup_process_ptr(pid, &success);
if ( Object )
{
  v6 = ObOpenObjectByPointer(Object, HandleAttributes, 0i64, DesiredAccess, *nt_PsProcessType, 0, &Handle);
  if ( v6 >= 0 )
  {
    if ( !Handle )
      v6 = 0xC00000F2;
    *out_buffer = Handle;
    v6 = 0;
```

Figure 3. Call to ObOpenObjectByPointer, the highlighted 0 value indicates that this request is from a kernel caller

The vulnerability in this function is that it is subverting the Windows process access permissions by giving any process that requests it access to any resource. We have reason to believe that this is by design. There is, in fact, a different IOCTL command 0x8000200C that performs the same operation but by setting AccessMode to UserMode, thus respecting the Windows permissions model.

It seems the developers somewhat considered this attack scenario and every IOCTL to the control interface is gated by an access check with a 128-bit mask, each bit acts as a permission flag to use a specific IOCTL. Figure 4 shows an example of these permission checks.

```
if ( ((g_control_ioctl_access_flags >> 2) & 1) != 0 )
{
  if...
}
else
{
  v6 = 0xC0000429;
}
```

Figure 4. Access check present on most IOCTLs of the command interface

These permissions are set by a game during the initial protection request; however, the effective permissions value comes from the request itself making it trivial for a custom client to request the highest possible permissions.
The reasoning behind this model is likely that the permissions flags are only set by the official Anti-Cheat SDK and can't be manipulated by the games using it, but in practice there is nothing preventing a rogue client from obtaining complete access.

3.3 IOC

As of writing we have no evidence of this vulnerability being exploited in the wild.

File name	ACE-BASE.sys
File version	1.0.2202.6217
SHA1	39402a9a3d90ba62938052089c8cbde9fb4e639f
SHA256	7326aefff9ea3a32286b423a62baebe33b73251348666c1ee569afe62dd60e11
Installation path when used by a legitimate application	C:\Windows\System32\drivers\ACE-BASE.sys

A copy of the sample is available in the LOLDrivers repository.

DEFENCE TECH

4

Conclusions

4. Conclusions

The vulnerability described in this report allows an attacker with a non-administrator account to escalate their privileges to system and to terminate security products. This vulnerability was made possible by a commercial product meant to run with system privileges that was not designed with security in mind; its security model is entirely based on obfuscation through a number of hardcoded secrets and the use of four different encryption algorithms. Yet again, security through obscurity has proven to not be a viable strategy.

Our team analysed the software and exposed the design flaws by building a proof-of-concept exploit that demonstrates how it can be abused. Although the vendor decided to ignore our report, we shared our findings with the security community to protect users and businesses from potential attacks.

It is important to keep security products up to date and to follow best practices to prevent BYOVD attacks, this includes monitoring the software that is installed on the system and logging new drivers the first time they are loaded.

DEFENCE TECH

5

Disclosure timeline

5. Disclosure timeline

- September 2023 - Vulnerability discovered by Malware Lab team.

- September 25, 2023 - Report submitted to the vendor through the Tencent Security Response Centre.

- September 26, 2023 - The vendor closed the report as out of scope.

- January 2024 - Disclosure deadline reached.

- January 25, 2024 - CVE-2024-22830 assigned by MITRE[10]

- February 22, 2024 - Initial public disclosure, submission to the LOLDrivers project[11]

- April 24, 2024 - Publication of this report

[10] *https://cve.mitre.org/*

[11] *https://github.com/magicsword-io/LOLDrivers/pull/168*

DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

DONEXIT IT SECURITY

NEXT INGEGNERIA DEI SISTEMI

FORAMIL RADAR TECHNOLOGIES & DEFENCE SYSTEMS

INNOVESI electronic systems

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it