Q Search.

Projects

Popular

December 3, 2019

Check out our latest projects at

Recent

CVE-2019-12750: Symantec Endpoint Protection Local Privilege Escalation -

DerbyCon 2016 CTF Write Up September 27, 2016

QNAP NAS – Remote Unauthenticated User To Admin Shell: Part 1 April 8, 2015

https://github.com/nettitude

CVE-2020-14418: madCodeHook Library Local Privilege Escalation

By Kyriakos Economou | December 1, 2020

Nettitude discovered a vulnerability in the 'madCodeHook' third party library which caused a number of security products, including Cisco AMP and Morphisec Unified Threat Prevention Platform, to contain a local privilege scalation vulnerability. Since the vulnerability originated in a third party library, it is likely to affect other software using that library. The madCodeHook author states that at least 15 security vendors use the library.

Following a coordinated disclosure process driven by Nettitude, the latest versions of Cisco AMP and Morphisec
Unified Threat Prevention Platform are no longer vulnerable to this issue. Other software that uses madCodeHook may
still be vulnerable



Products Affected

All software using the madCodeHook library has a kernel mode DLL injection vulnerability in versions prior to v4.1.3.

This includes, but is likely not limited to, the following software:

- Cisco AMP prior to v7.2.13:
 - ExPrevDriver.sys
- Morphisec Unified Threat Prevention Platform v4.x earlier than v4.1.2, v3.5.9:
 - MorphiDriver.sys

Introduction

There is a design flaw inside the kernel mode component of a DLL injection library called madCodeHook, developed by Systemsoftware Mathias Rauen, that is used by several security and other vendors as part of their exploit prevention capabilities, amongst other things.

madCodeHook is a framework of function hooking and code injection techniques that can be used to monitor specific processes for abnormal behaviour or any other situation where API hooking might be required. This works by defining a list of process names to inject a specific DLL module that can also be used for hooking purposes.

The DLL injection takes place from kernel mode in order to achieve early injection into the processes of interest during their initialization stages. The library itself offers the ability to check for specific digital signatures before injecting a module into a process. It also blocks any shared write access to the module to be injected.

These two methods aim to prevent arbitrary DLL injection and/or modification of the defined module after it has passed the signature checks. However, we found a design flaw that can be abused, which allows an attacker to achieve arbitrary DLL injection into privileged processes and execute malicious code in the security context of SYSTEM user account.

According to the author of this library, there are at least 15 security vendors using this library, but they were not able give us further details on this in order to protect their customers.

Based on our investigation, "MalwareBytes" and "EMSISOFT" are also using this library, but since they were notified of this vulnerability by the author of the library, we didn't have the time to test our exploit against their products.

During this article we will be examining this issue through the Cisco AMP product which uses a driver developed by Morphisec, which in turn uses the aforementioned kernel mode code library, which can be used for hooking purposes.

The Vulnerability

The ExPrevDriver.sys module, formally known as MorphiDriver32.sys and originally developed by Morphisec, is used to inject the Protector64.dll and Protector32.dll modules into selected user mode processes for sets Tomodrical Companies.

By sending the appropriate IOCTL we can define a path to a module to inject and also process names to protect and/or exclude from the DLL injection on runtime as they start.

Even though the driver will verify that the module is digitally signed by a specific trusted certificate, and blocks any modifications to the module after that by only allowing read access to user mode processes, it doesn't protect from path redirection attack vectors.

In addition, the driver verifies the digital signature of the module to be injected only during this stage. This type of vulnerability is described as time-of-check-time-of-use (TOCTOU).

Exploitation

er to exploit this vulnerability, we need to send the appropriate IOCTL using an input buffer where the data has a specific structure. In addition to that, to be able to reach the dispatcher function, we must first solve a custom cryptochallenge that was applied by the author of madCodeHook library, presumably to block pre-analysis fuzzing and/or make the analysis more difficult

This stage makes use of the RIPEMD-160 hashing algorithm along with two custom "encryption" routines, where all of them are bound together in a specific order for the purpose of validating and blocking arbitrary input.

The input validation scheme is designed as follows

- 1. Prepare a structurally valid input data buffer.
- 2. Encrypt the data using a custom encryption routine.
- 3. Hash the encrypted data.
- 4. Re-encrypt the encrypted data with another custom encryption routine using the hash as a key.
- 5. Prepend the hash generated in step 3 and send the IOCTL.

The driver will go through the reverse steps to decrypt and validate the structure of the input data. Note, this is an overview of the input validation design; we will leave the rest of the details as an exercise for the reader.

This screenshot shows part of this request to the driver

```
IOCTL Input Data
```

This is a legitimate request performed by the driver during initialization. We can see the full path to the 32-bit module to be injected, and a partial list of target process names. A similar request using the path to the 64-bit module will be used as well to protect 64-bit processes. So, when the driver is loaded it will set a list of process names to monitor for and inject the appropriate module as defined above once a process that matches one of those names is started.

A local attacker can create a directory junction to the original protector module and include this in the path to send via the IOCTL. The driver will verify that we point to the expected module and will accept this request.

The vulnerability exists because instead of the driver storing the real path to the module that we defined via the IOCTL (in other words normalize the provided path), it actually stores the path as we define it. However, the attacker can now delete the junction and create a directory instead, and place an arbitrary DLL with the same name, which will cause the driver to inject the malicious DLL to the selected process defined in the IOCTL when the process starts.

In this way we can trick the driver to successfully validate our request and store the module path as we submit it. We can then inject into arbitrary processes, defined by us via IOCTL, our own DLL that contains the payload.

Finally, we can 'force' the driver to load our DLL in a service process running as SYSTEM and obtain full access in the affected host.

Summary of exploitation steps

Here is a summary of the exploitations steps.

- Create directory junction pointing to the legit module directory ("mklink /J C:\users\<username>\Desktop\expression") C:\Program Files\Cisco\AMP\exprev")
- . Send IOCTL using path "C:\users\<username>\Desktop\exprev\Protector64.dll"
- Delete directory junction and create a normal full path "C:\users\<username>\Desktop\exprev\Protector64.dll" where the DLL is now your own DLL of choice.
- . Start a process that you chose to protect through the IOCTL request.
- You should see your DLL loaded into that process.

Avoiding this type of vulnerability

Drivers should normalize the received module path instead of storing the path as it is received from userland, which may contain junctions in-between, and lead to security issues such as the one reported here

Additionally, the signature validation routine could be used each time just before a module is injected into a process. In that way, we can ensure that unsigned and/or self-signed binaries cannot be injected into arbitrary processes by abusing design flaws in similar driver functionality.

Finally, drivers shouldn't (in general) allow low privileged processes to send IOCTL requests. This can be achieved by enforcing the correct ACL to the named device object that is exposed to the user mode processes

The madCodeHook author rapidly patched the library. We also chose to work with Morphisec, an affected vendor, during the disclosure process. Both entities were speedy and effective in their responses.

- Discovery: 6 July 2020
- Library author notified: 6 July 2020
- · Patch issued by library author: 16 July 2020
- Nettitude disclosure: 1 December 2020

Share This Story, Choose Your Platform!



Related Posts =









· 🔰 1 Popu lar docu ment stora ge solut ion, ONL YOFF ICE, affec ted by multi ple vuln erabi lities. Our lates t post by @str awp show S how to expl oit this for unau thent icate d remo te code exec ution

> I.. E. A



Nettitud Labs ...

♡ 1

High light s from Day 1 of #Pw n2O

Toro nto 2022

wn

Con nor

Ford from

USEFUL LINKS Vulnerability Research Nettitude Cyber Security Tools Red Team Training Careers at Nettitude< NETTITUDE LABS PRESENTED BY

NETLILITOE

CV31 3RZ

UK

1 Jephson Court Trancred Close

Leamington Spa Warwickshire

EUROPE ASIA Leof. Siggrou 348 18 Cross Street Kallithea Athens Greece

#02-101 Suite S2039 176 74 048423

AMERICAS

Suite 403

10004

New York City NY

50 Broad Street

CONTACT US

Name * @

2 Your name or handle*

Email address * @

@ your@email.com*

Message * 0

Your message to Nettitude Labs.*

protected by reCAP

Send your message

y o in

Rock s the stag e 🖖





