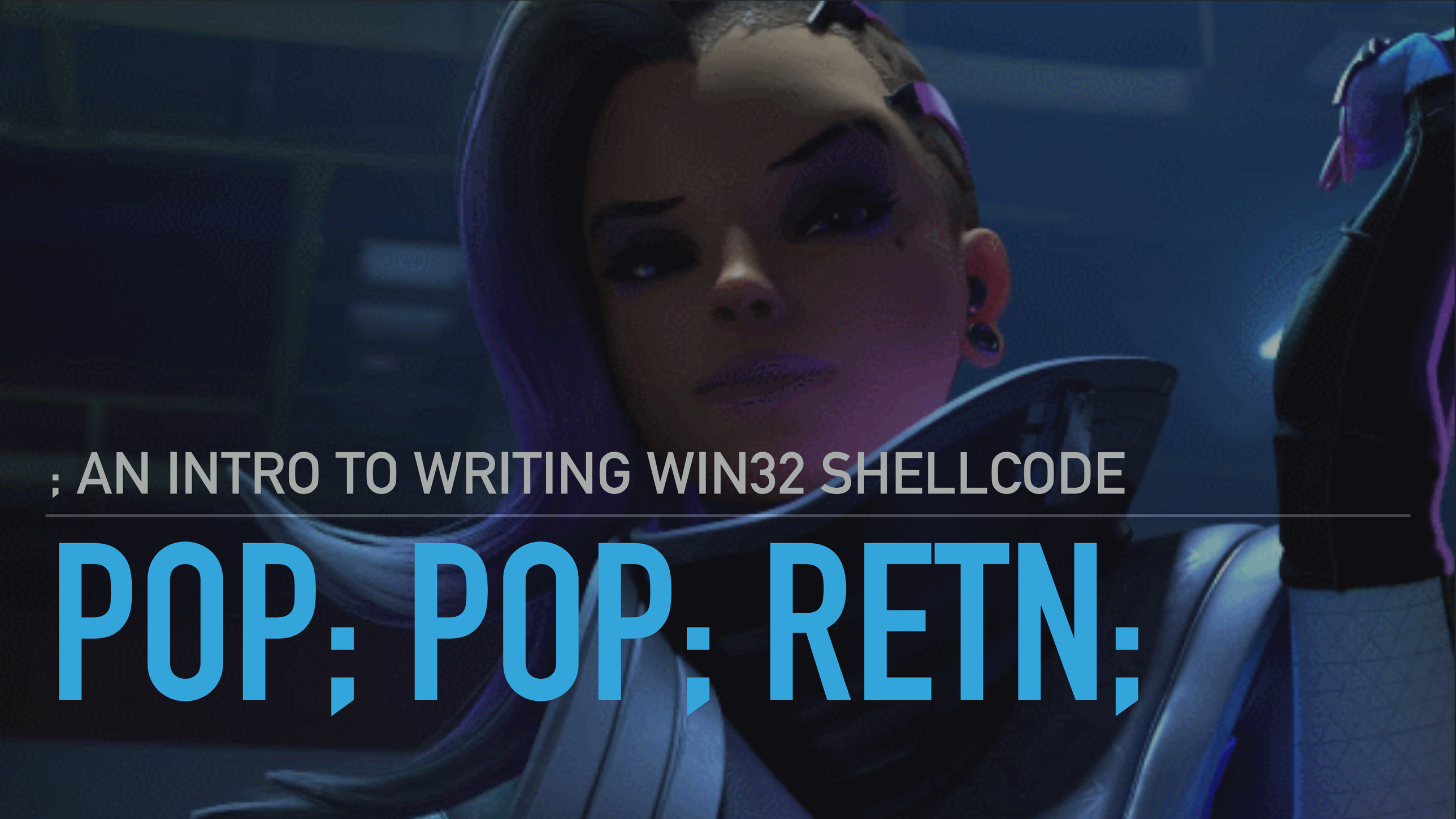


; AN INTRO TO WRITING WIN32 SHELLCODE

POP; POP; RETN;

INTRODUCTIONS

- Chris "cmaddy" Maddalena
  - Sr. Security Consultant @ GuidePoint Security
    - Red Team
    - Not an Assembly expert
  - keybase.io/cmaddalena
    - Find write-ups and code there

GuidePoint SECURITY
YOUR MISSION. SECURED.

AGENDA

- We'll be working with the Intel x86 architecture on a 32-bit Windows target.
  - Yes, I know, 64-bit, Linux, ARM… best to start with a narrow scope.
- Topics:
  - A review of the basic "need to knows" for Assembly.
  - Discuss debugging shellcode with Python.
  - Compare msfvenom vs. manual shellcode.
  - Take a look at writing a TCP connect back stager from scratch.

WHAT ARE WE TALKING ABOUT?

- Shellcode is more than the name implies:
  - It’s Assembly language;
  - Code that typically spawns a command shell; and
  - Carefully crafted instructions that are injected into a running program and executed.


MEMORY & REGISTERS

CPU REGISTERS

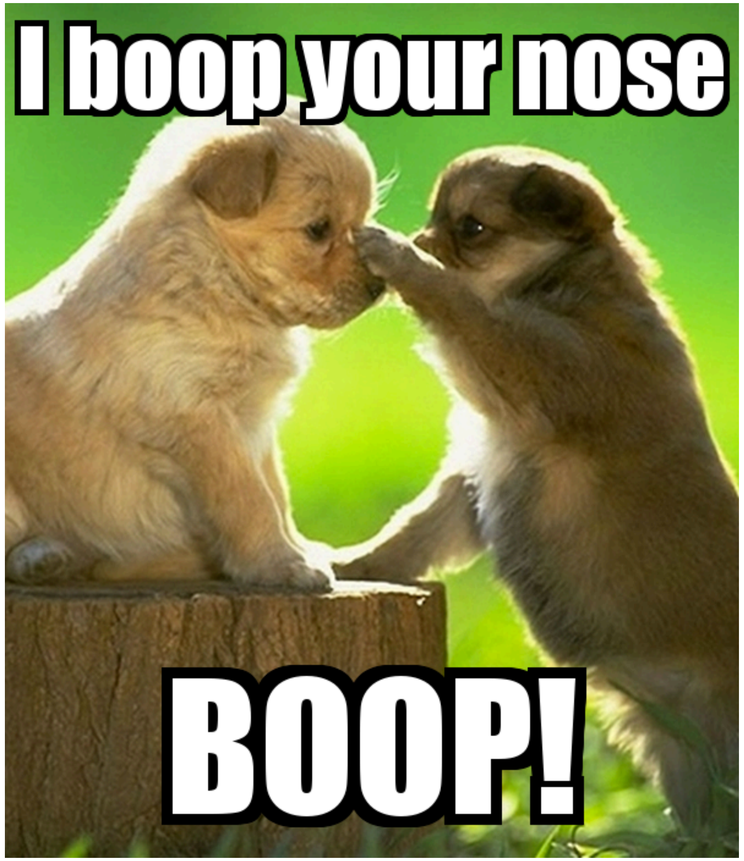
- CPU registers have common uses, but can be used as we see fit.
- There are some registers we will reference often:
  - ESP: Stack Pointer, points to the top of the stack, used for making pointers.
  - EAX: Accumulator, holds return values, often used as a calculator.

PUSH AND POP: THE STACK

- A reversed region of memory where data is stored.
  - The stack is FILO, e.g. First In Last Out
  - The stack “builds upwards.”
- x86 has specific calls for interacting with the stack, like PUSH and POP.
  - PUSH puts something on top of the stack.
  - POP removes the top of the stack.

EFFECTS OF ENDIANNESS

- Little and Big Endian
  - Intel x86 uses Little Endian
  - The least signifiant byte is stored first.
- Let's push the ASCII string "Boop" to the stack:
  - Hex: `42 6f 6f 70`
  - Assembly: `PUSH 0x706f6f42`
  - Machine Code: `68 42 6f 6f 70`



REFERENCING MEMORY

- Windows addresses change with each version and Service Pack update.
- Also, memory can be protected:
  - ASLR – Address Space Layout Randomization, changes the addresses of functions between reboots, Windows Vista and beyond.
  - DEP – Data Execution Protection, marks sections of memory as non-executable, mostly a concern for exploits, Windows XP and beyond.

TESTING SHELLCODE

COMPILING AND ENCODING

- Use *nasm* to compile your asm file:
  - nasm demo.asm -o demo.bin
- Format the bin file as a \x escaped string for Python:
  - 33 becomes \x33 and so on.
  - Easily scripted – Python script on my GitHub or Google around.

USING PYTHON CTYPES

- Python's ctypes library allows you to:
  - Execute shellcode with Python
  - Import DLLs
- Something to remember:
  - The shellcode isn't being injected into a process, so additional steps may be required.

TEST CODE SAMPLE

```
#!/usr/bin/python
import ctypes

# Insert shellcode to be tested here
# Shellcode must be \x formatted
# Uncomment the \xcc to enable analysis in a debugger
shellcode = bytearray(
# "\xcc"

)

# Print the total length of the shellcode
print "Total Length: ",len(shellcode)

# Pause just before shellcode execution and wait for key press
# Attach the debugger to Python now!
debug = raw_input("Debug pause!")
```

GENERATING SHELLCODE

THE EASY WAY

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~# msfvenom -h
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
    -p, --payload       <payload>    Payload to use. Specify a '-'
        --payload-options            List the payload's standard o
    -l, --list          [type]       List a module type. Options a
    -n, --nopsled       <length>     Prepend a nopsled of [length]
    -f, --format        <format>     Output format (use --help-for
        --help-formats               List available formats
    -e, --encoder       <encoder>    The encoder to use
    -a, --arch          <arch>       The architecture to use
        --platform      <platform>   The platform of the payload
        --help-platforms             List available platforms
    -s, --space         <length>     The maximum size of the resul
        --encoder-space <length>     The maximum size of the encod
    -b, --bad-chars     <list>       The list of characters to avo
    -i, --iterations    <count>      The number of times to encode
    -c, --add-code      <path>       Specify an additional win32 s
    -x, --template      <path>       Specify a custom executable f
    -k, --keep                       Preserve the template behavio
    -o, --out           <path>       Save the payload
    -v, --var-name      <name>       Specify a custom variable nam
        --smallest                   Generate the smallest possibl
    -h, --help                       Show this message
root@kali:~#
```

THE HARD WAY

Brain + coffee = Assembly code

edskudder.tumblr.com

WHY WOULD WE EVER DO THIS THE HARD WAY?

- To learn how the shellcode works.
- To make custom shellcode for a target.
- To make smaller shellcode.

USING MSFVENOM AND WINDOWS/MESSAGEBOX

- We can use msfvenom to generate the shellcode.
  - Command:

    msfvenom -p windows/messagebox

    TEXT="Boop" TITLE="Sombra says:" -f c

252-bytes

A LOOK AT THE MSFVENOM PAYLOAD'S FUNCTION

- Our function: **MessageBox(0, "Boop", “Sombra says:”, 0)**

```
int WINAPI MessageBox(
  _In_opt_ HWND    hWnd,
  _In_opt_ LPCTSTR lpText,
  _In_opt_ LPCTSTR lpCaption,
  _In_     UINT    uType
);
```

WRITING OUR OWN SHELLCODE FOR WIN XP SP3

```
XOR EAX,EAX
PUSH EAX
PUSH 0x3a737961
PUSH 0x73206172
PUSH 0x626d6f53
MOV ECX,ESP
PUSH EAX
PUSH 0x706f6f42
MOV EDX,ESP
PUSH EAX
PUSH ECX
PUSH EDX
PUSH EAX
MOV ESI,USER32.MessageBoxA
CALL ESI
```

39-bytes

ADDING A FUNCTION

- Metasploit will including instructions for exiting processes or threads like ExitProcess().
  - We can do that, too, and hardcode the address to keep the code smaller.

```
PUSH EAX
MOV EAX,KERNEL32.ExitProcess
CALL EAX
```

+9 bytes


DEMO TIME

REVIEWING THE RESULTS

- Both of these options produce the same result, a message box.
  - But only one of these options will work if you have just a small amount of space.
  - While the other one is only smaller when you can afford to specifically target an OS.


BUILDING A STAGER

HOW A STAGER WORKS

Stage 1

Connect back

Stage 2

THE REQUIREMENTS FOR A WIN32 STAGER

- The WinSock API (WSA) to connect and accept data requires.
- This requires a specific sequence:
  - WSAStartUp() : Initiates WSA for the process that calls it.
  - WSASocketA() : Make a new socket and get the socket descriptor.
  - connect() : Connect the new socket.
  - recv() : Prepare to receive data on the socket.

HIGHLIGHTING WSASOCKETA

```
SOCKET WSASocket(
  _In_ int                 af,
  _In_ int                 type,
  _In_ int                 protocol,
  _In_ LPWSAPROTOCOL_INFO lpProtocolInfo,
  _In_ GROUP               g,
  _In_ DWORD               dwFlags
);
```

WSASOCKETA IN ASSEMBLY

```
XOR EDI,EDI
PUSH EDI
PUSH EDI
PUSH EDI
PUSH EDI
PUSH EDI
INC EDI
PUSH EDI
INC EDI
PUSH EDI
MOV EBX,WS2_32.WSASocketA
CALL EBX
MOV EDI,EAX
```


DEMO TIME

CHECK OUT THESE PAPERS/TUTORIALS

- http://www.fuzzysecurity.com/tutorials.html
  - Specifically the Exploit Development section.
  - Part 6 covers MessageBox and some additional msfvenom comparisons.
- https://www.corelan.be/ Exploit writing series
  - Much wordier, but excellent information and an entire article on Win32 shellcode.
- Skape's "Understanding Windows Shellcode" white paper
- Hacking: The Art of Exploitation from NoStarch Press


QUESTIONS?