Advances in Exploit Technology

hdm & spoonm

CanSecWest, 2005

Part I

Introduction

Who are we?

- spoonm
  - Full-time student at a Canadian university
  - Metasploit developer since late 2003
- H D Moore
  - Full-time employee at a network security firm
  - Metasploit project founder and developer

What is Metasploit?

- Research project with 8 members
  - Focused on improving the state of security
  - Provide information and tools for researchers
  - Resource for IDS and security tool vendors

- Created the Metasploit Framework
  - Open-source exploit dev platform
  - Includes 60 exploits and 70 payloads
  - Implements ideas from everywhere
  - Currently four primary developers
  - Handful of external contributors

What is this about?

- Recent advances in exploit technology
- Exploit development trends and XP SP2
- Interesting post-exploitation techniques
- Improving the exploit randomness
- Metasploit Framework 3.0 architecture

Part II

Windows Exploitation

Exploit Trends

- Public Windows exploits are still terrible...
  - Tons of ugly, inflexible, hardcoded crap
  - Demonstrate no knowledge of underlying flaw
  - Rarely use information leakage for system targetting

Exploit Trends

- Public Windows exploits are still terrible...
  - Tons of ugly, inflexible, hardcoded crap
  - Demonstrate no knowledge of underlying flaw
  - Rarely use information leakage for system targetting
- ...but they have improved over the last year!
  - More exploits are supporting multiple payloads
  - Return addresses are more reliable
  - Payloads are getting slightly less ghetto

PoC Community

- The number of people capable of writing exploits is going up...
  - Nearly 250 PoC authors in 2004 (packetstorm, etc)
  - Win32 exploit dev information has hit critical mass
  - Exploit development training is in high demand

PoC Community

- The number of people capable of writing exploits is going up...
  - Nearly 250 PoC authors in 2004 (packetstorm, etc)
  - Win32 exploit dev information has hit critical mass
  - Exploit development training is in high demand
- ...but the number of "hard" exploits made public is the same
  - People are lazy, skilled people tend to horde their code
  - Example: Microsoft ASN.1 Bit String Heap Corruption
  - Most "difficult" exploits are disclosed due to leaks
  - Win32 kernel exploits are still the domain of a few :-)

Windows XP SP2

- Microsoft’s "patch of the year" for 2004
  - SP2 included a handful of anti-exploit changes
  - The important ones were already in 2003
    - Use of registered system exception handlers
    - Core services compiled with stack protection
  - Page protection is still dependent on hardware

Metasploit and SP2

- Exploit development barely affected by SP2
- A handful of XP SP2 and 2003 SEH return addresses
- Third-parties are not using Visual Studio 7
  - Most commercial applications do not use /GS
  - Have yet to see one that uses Registered SEH

Part III

Return Addresses

Return Address Reliability

- An exploit is only as good as the return address it uses
- Many vulnerabilities only allow one exploit attempt
- Returning directly to shellcode is not always possible
  - Most Windows exploits use a "bounce" address
  - Indirect returns are useful on other platforms as well

Windows Return Addresses

- Windows stack addresses are usually not predictable
- Executable and library addresses *are* predictable
  - System libraries are often static between patch levels
  - Application libraries change even less frequently
  - Executable addresses only change between app versions
- Static system libraries can go a long way...

Windows Return Addresses

- Windows stack addresses are usually not predictable
- Executable and library addresses *are* predictable
  - System libraries are often static between patch levels
  - Application libraries change even less frequently
  - Executable addresses only change between app versions

- Static system libraries can go a long way...
- A great example is the "ws2help.dll" library:
  - Static across all versions of Windows 2000
  - Static across Windows XP SP0 and SP1
  - Used in dozens of exploits in the Framework

The Magic SEH

- Stack overflows rarely exploit return address overwrites
- Overwriting the structured exception handler (SEH) is easier
- The first exception causes smashed SEH to be called
- SEH frame can exist before or after the return address

```
/* Struction Exception Handler */
typedef struct _EXCEPTION_REGISTRATION
{
struct _EXCEPTION_REGISTRATION* prev;
PEXCEPTION_HANDLER              handler;
} EXCEPTION_REGISTRATION, *PEXCEPTION_REGISTRATION;
```

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

- `EstablisherFrame` points 4 bytes before handler address

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

- `EstablisherFrame` points 4 bytes before handler address
- Passed to exeception handler function `[esp+8]`

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

- `EstablisherFrame` points 4 bytes before handler address
- Passed to exeception handler function `[esp+8]`
- Return back to code via `pop reg, pop reg, ret`

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

- `EstablisherFrame` points 4 bytes before handler address
- Passed to exeception handler function `[esp+8]`
- Return back to code via `pop reg, pop reg, ret`
- The pop + pop + ret combination is easy to find in memory

The Magic SEH

- Overwrite the frame, trigger exception, got EIP :-)
- The prototype for the SEH function is:

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
 struct _EXCEPTION_RECORD *ExceptionRecord,
 void * EstablisherFrame,
 struct _CONTEXT *ContextRecord,
 void * DispatcherContext );
```

- `EstablisherFrame` points 4 bytes before handler address
- Passed to exeception handler function `[esp+8]`
- Return back to code via `pop reg, pop reg, ret`
- The pop + pop + ret combination is easy to find in memory
- Registered SEH and Windows XP/2003 limit this type of abuse

Unix Return Addresses

- Linux and BSD
  - Library addresses are usually not predictable
  - Every executable has a static load address
    - Every distribution compiles its own binaries
    - Exploits must target specific versions and operating systems
    - Commercial (binary-only) applications are mostly static

Unix Return Addresses

- Linux and BSD
  - Library addresses are usually not predictable
  - Every executable has a static load address
    - Every distribution compiles its own binaries
    - Exploits must target specific versions and operating systems
    - Commercial (binary-only) applications are mostly static
- Commercial Unix
  - Library addresses are sometimes predictable
  - Every executable has a static load address
    - These addresses are static per package version
    - Windows-style return addresses work well
    - This includes Mac OS X, Solaris, HP-UX, AIX, etc

Analysis Methods

- Finding solid return addresses involves a few steps
  - Load the executable or library into memory
  - Determine all permutations of the desired opcode
  - Search memory contents to find these bytes
  - Determine the virtual address for each offset

Analysis Methods

- Finding solid return addresses involves a few steps
  - Load the executable or library into memory
  - Determine all permutations of the desired opcode
  - Search memory contents to find these bytes
  - Determine the virtual address for each offset
- Many people use a debugger to accomplish this task
  - This is a tedious process to do manually
  - Limited to one version at a time, even with a plugin
  - Requires the installation of each tested version

msfpescan

- msfpescan - a utility included in the Metasploit Framework
  - Can analyze any PE executable or DLL in offline mode
  - Simple to automate and cross-reference results
  - Does not require a Windows system to run
  - Easily analyze multiple versions on the command line
  - Capable of dumping other information as well
    - Imports, Exports, and IAT addresses
    - Resource information, internal versions
    - Standard PE header information

Using msfpescan to find addresses

- Install the Metasploit Framework (2.3 or newer)
- Place your target executable or DLL into some directory
- Use msfpescan to quickly find return addresses:

```
# Locate any form of pop/pop/ret opcodes
$ msfpescan -f mod_oiplus.dll -s
0x1001413c   esi edi ret
0x10009ea2   esi ecx ret
0x100113bd   esi ebx ret

# Locate any opcodes that take us to [eax]
$ msfpescan -f mod_oiplus.dll -j eax
0x1000969d   push eax
0x100141a3   jmp eax
0x10010e69   call eax
```

Opcode Databases

- Contains opcodes across every executable and DLL in Windows
- The new version includes over nine million records
- Data is generated directly from the files themselves
- Quickly cross-reference return addresses over the entire DB
- Publicly available from http://www.metasploit.com/

Future Development

- Context-aware return address discovery
  - Demonstrated by eEye at Black Hat 2004
  - Similar project in development from Metasploit

Future Development

- Context-aware return address discovery
  - Demonstrated by eEye at Black Hat 2004
  - Similar project in development from Metasploit
- Executable analysis tools for Solaris, Mac OS X, Linux, BSD
  - Usefulness limited compared to Windows platform
  - Static libraries are great for cross-version exploits

Part IV

Post-Exploitation

The Meterpreter

- Windows version uses in-memory DLL injection techniques
- Dynamically extensible over the network
- Extensions are standard Windows DLLs
- Loading an extension updates available commands
- Support for network encryption
- Huge feature set in the public version
  - Upload, download, and list files
  - List, create, and kill processes
  - Spawn "channelized" commands in the background
  - Create port forwarding channels to pivot attacks

Ordinal-based Payload Stagers

- Techniques borrowed from Oded's lightning talk from core04
- 92 bytes and works on every Windows OS and SP
- Staging system can chain any other Windows payload
- Implementation also has a few size reductions:
  - Optimized module walked finds ws2_32.dll
  - Functions are loaded from base + ordinal offset
  - Chained calls return to the next function

PassiveX

- Payload modifies registry and launches IE
- IE loads custom ActiveX control to stage the payload
- Communications channel is via HTTP requests
  - Uses standard IE proxy and auth settings
  - Useful on heavily firewalled DMZ hosts
  - Providers bi-directional channel for next stage

- Can be used to inject VNC, Meterpreter, etc
- Fully-functional and part of version 2.4

Other Network Stagers

- UDP-based stager and network shell for Linux
- UDP-based DNS request staging system
  - UDP shell depends on the bash –noediting option
  - Can pass through strict firewall rulesets
- ICMP-based listener and "reverse" payloads
- Findsock stagers being replaced by "findrecvtag"
- Source code included in Metasploit Framework

Part V

Improving Attack Randomness

Introduction

- Randomness, who cares?
  - Make IDS analysts work for their paycheck
  - Uncover flaws in your exploit code

Introduction

- Randomness, who cares?
  - Make IDS analysts work for their paycheck
  - Uncover flaws in your exploit code
- Adding randomness to exploit code
  - Modify attacks by setting protocol options
  - Randmomize all padding and non-critical data
  - Helper functions for different types of random data

Introduction

- Randomness, who cares?
  - Make IDS analysts work for their paycheck
  - Uncover flaws in your exploit code
- Adding randomness to exploit code
  - Modify attacks by setting protocol options
  - Randmomize all padding and non-critical data
  - Helper functions for different types of random data
- Adding randomness to machine code
  - Avoid "static" payload encoding systems
  - Substitute like instructions and reorder tasks
  - Randomize nop sleds and any other opcode fills

Polymorphism

- Viruses morphed to evade signature anti-virus
- Shellcode doesn’t morph, isn’t really polymorphic
- Generators produce functionally equivalent permutations
- Simple examples: random 0x90 nops, add/sub switching

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves
- Pros:
  - Well thought out - analyzed attacks against NIDS
  - Mathematica files output, mathy backing
  - Spectrum analysis - push sled to byte distribution

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves
- Pros:
  - Well thought out - analyzed attacks against NIDS
  - Mathematica files output, mathy backing
  - Spectrum analysis - push sled to byte distribution
- Cons:
  - Complicated system, really hard to build upon
  - Decoder generation isn’t that great
  - Making compromises for size/robustness

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode
- Pros:
  - Polymorphizing code is pretty easy
  - No size or functionality compromises
  - Bad character and register avoidence

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode
- Pros:
  - Polymorphizing code is pretty easy
  - No size or functionality compromises
  - Bad character and register avoidence
- Cons:
  - Less thought out, NIDS attacks not deeply analyzed
  - Hard to push to arbitrary byte distribution
  - Less "polymorphism", more restrictions

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect
- Current implementation
  - Current system is a bit ugly
  - Hard without writing a real assembler
  - Want it to be fairly fast

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect
- Current implementation
  - Current system is a bit ugly
  - Hard without writing a real assembler
  - Want it to be fairly fast
  - Pex::Poly has 3 phases
  - Dependency iteration and block selection
  - Instruction offset calculations
  - Instruction register assignment

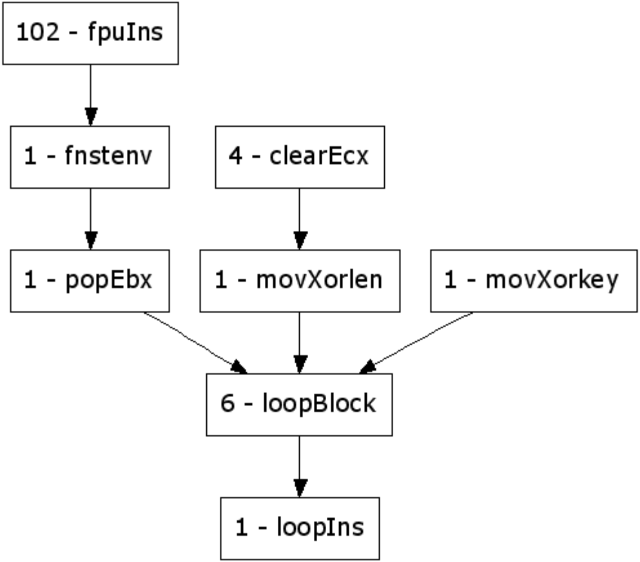
Shikata Ga Nai

- It’s too much work to polyize every payload
- Created one decent "polymorphic" encoder

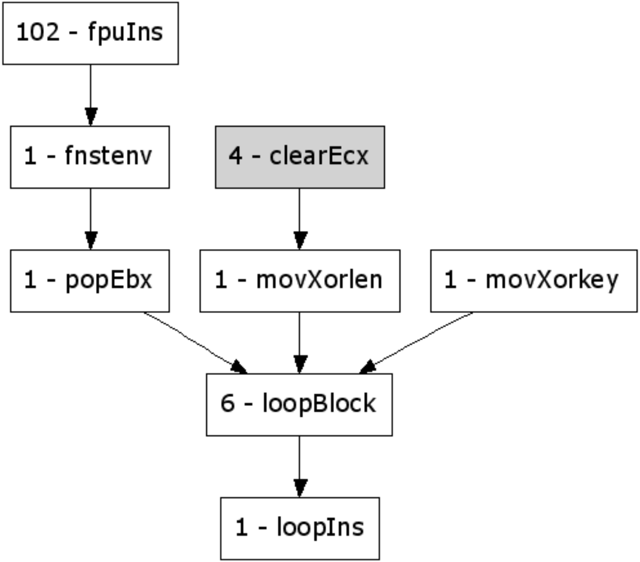
Shikata Ga Nai

- It’s too much work to polyize every payload
- Created one decent "polymorphic" encoder
- Uses noir’s FPU geteip technique
- Approximately 1.3 million permutations
- Additive feedback xor, encodes it’s own end
- 27 bytes for the stub, 4 key, 4 encoded

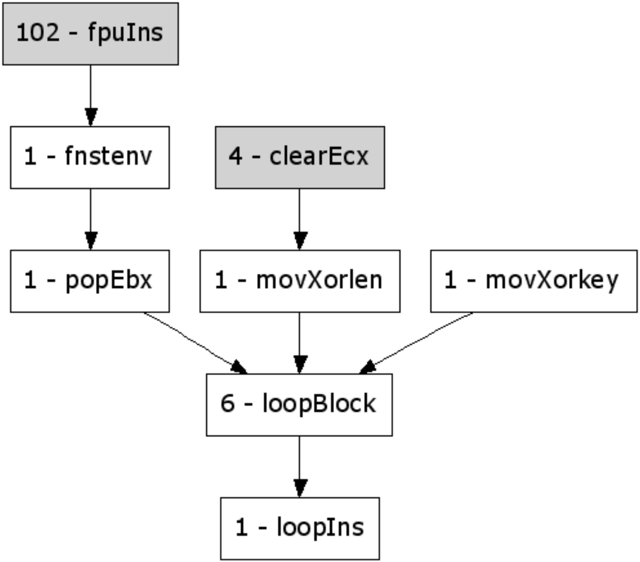
Shikata dependency iteration



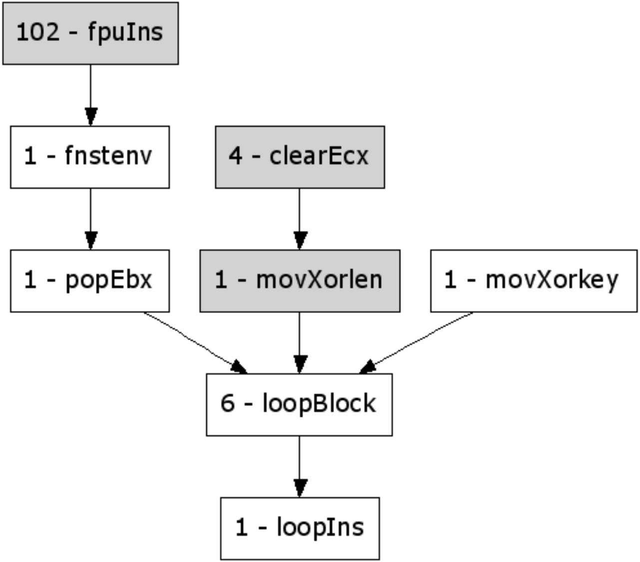
Shikata dependency iteration



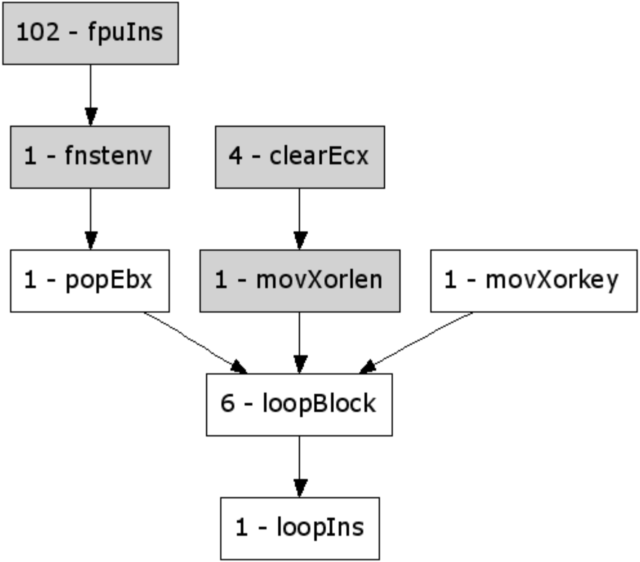
Shikata dependency iteration



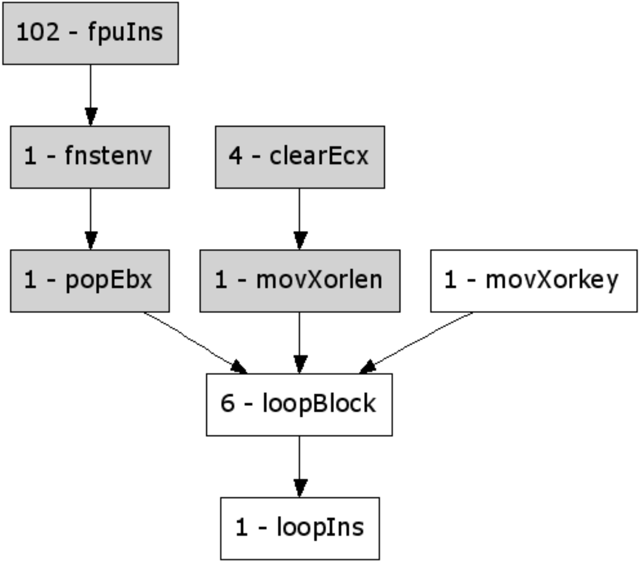
Shikata dependency iteration



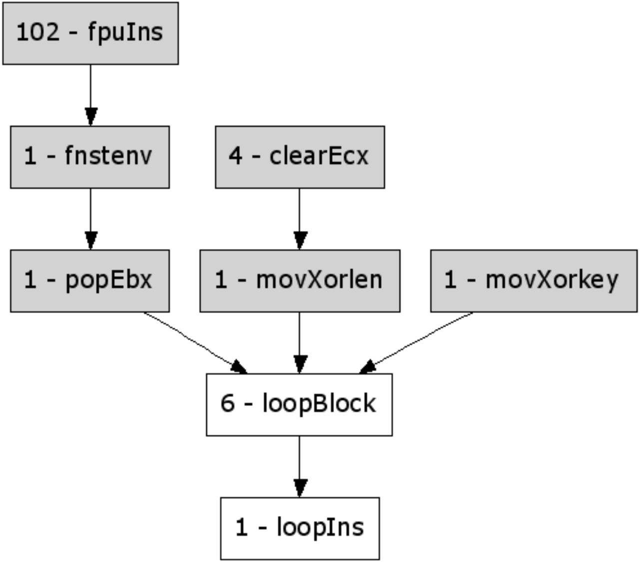
Shikata dependency iteration



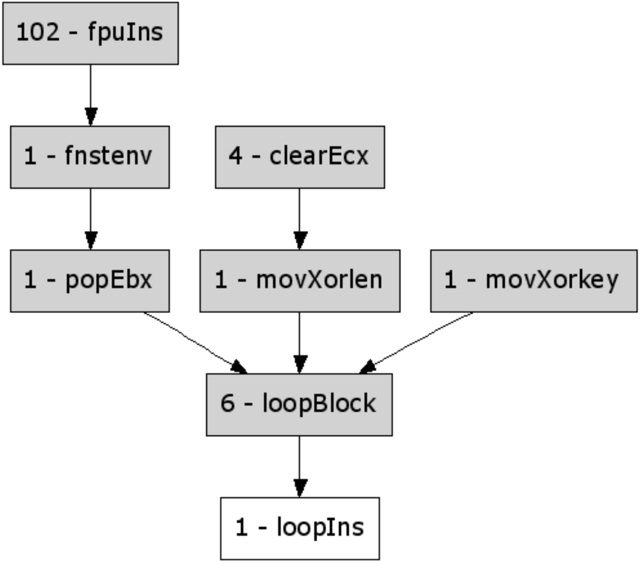
Shikata dependency iteration



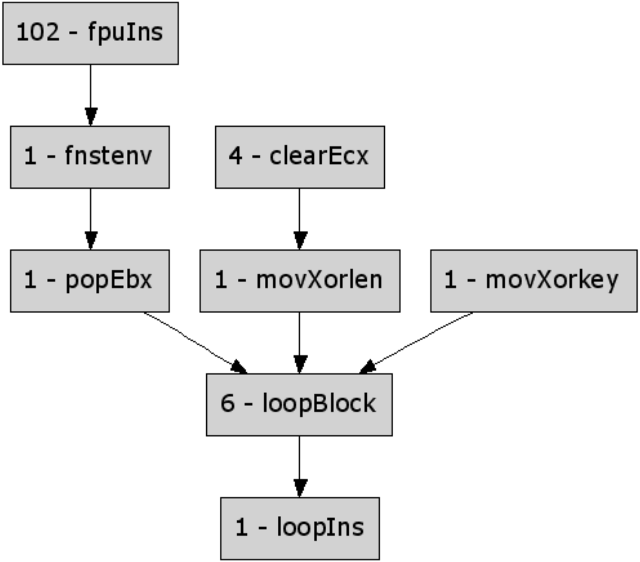
Shikata dependency iteration



Shikata dependency iteration



Shikata dependency iteration



Example output

```
00000000  BB6E887A69        mov ebx,0x697a886e
00000005  DDC4              ffree st4
00000007  D97424F4          fnstenv [esp-0xc]
0000000B  58                pop eax
0000000C  29C9              sub ecx,ecx
0000000E  B101              mov cl,0x1
00000010  83E8FC            sub eax,byte -0x4
00000013  31580E            xor [eax+0xe],ebx
00000016  03580E            add ebx,[eax+0xe]
00000019  E2F5              loop 0x10
```

Example output

```
00000000  DBC1                  fcmovnb st1
00000002  31C9                  xor ecx,ecx
00000004  B101                  mov cl,0x1
00000006  D97424F4              fnstenv [esp-0xc]
0000000A  5B                    pop ebx
0000000B  BAC8E2C8F8            mov edx,0xf8c8e2c8
00000010  83C304                add ebx,byte +0x4
00000013  315313                xor [ebx+0x13],edx
00000016  035313                add edx,[ebx+0x13]
00000019  E2F5                  loop 0x10
```

Example output

```
00000000  BB7B833BB9          mov ebx,0xb93b837b
00000005  DAC0                fcmovb st0
00000007  D97424F4            fnstenv [esp-0xc]
0000000B  2BC9                sub ecx,ecx
0000000D  5E                  pop esi
0000000E  B101                mov cl,0x1
00000010  315E12              xor [esi+0x12],ebx
00000013  83C604              add esi,byte +0x4
00000016  03                  db 0x03
00000017  25                  db 0x25
00000018  8D                  db 0x8D
00000019  D9                  db 0xD9
0000001A  4C                  dec esp
```

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

- Builds the sled from back to front
- Prepends to the sled 1 byte at a time
- Generates a random byte and checks against tables

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

- Builds the sled from back to front
- Prepends to the sled 1 byte at a time
- Generates a random byte and checks against tables
  - Is the instruction length too long?
  - Is it a valid instruction?
  - Does it have any bad bytes?
  - Does it modify restricted registers?

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

Backwardz

```
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
```

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90
  - Fairly language independent, C version 100 lines
  - Very fast, simple, deterministic
  - Allows for different scoring systems, recursion...

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90
  - Fairly language independent, C version 100 lines
  - Very fast, simple, deterministic
  - Allows for different scoring systems, recursion...
  - Can’t support multibyte opcodes, escape groups, etc
  - Tables are pretty large, about 124k

OptyNop2 Output

```
$ ./waka 1000 4 5 | ndisasm -u - | head -700 | tail -20
000003B6  05419F40D4        add eax,0xd4409f41
000003BB  711C              jno 0x3d9
000003BD  9B                wait
000003BE  2C98              sub al,0x98
000003C0  37                aaa
000003C1  24A8              and al,0xa8
000003C3  27                daa
000003C4  E00D              loopne 0x3d3
000003C6  6692              xchg ax,dx
000003C8  2F                das
000003C9  49                dec ecx
000003CA  B34A              mov bl,0x4a
000003CC  F5                cmc
000003CD  BA4B257715        mov edx,0x1577254b
000003D2  700C              jo 0x3e0
000003D4  C0D6B0            rcl dh,0xb0
000003D7  A9FD469342        test eax,0x429346fd
000003DC  67BBB191B23D      a16 mov ebx,0x3db291b1
000003E2  1D9938FCB6        sbb eax,0xb6fc3899
000003E7  43                inc ebx
```

ADMmutate Distribution - 1

total: 6000
uniq: 52

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	6e	00	00	00	00	00	00	00	76
30	00	00	00	00	00	00	00	87	00	00	00	00	00	00	00	6a
40	6b	72	6a	68	74	66	77	6f	6d	74	6c	77	70	74	58	72
50	6a	67	71	70	7b	74	76	7c	70	7c	6b	78	00	6e	56	64
60	71	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	89	6c	78	00	74	72	df	7a	79	00	56	82	00	76	77
a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
f0	00	00	00	00	00	7c	00	00	71	7f	00	00	69	00	00	00

ADMmutate Distribution - 2

total: 6000

uniq: 52

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	64	00	00	00	00	00	00	00	6f
30	00	00	00	00	00	00	00	78	00	00	00	00	00	00	00	74
40	7f	6b	6f	7b	79	72	75	73	76	58	6f	7a	6c	78	7a	7e
50	71	6d	65	75	7f	72	7b	72	71	77	6d	64	00	71	7c	64
60	73	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	6b	79	87	00	74	74	e8	6b	68	00	76	5b	00	6d	72
a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
f0	00	00	00	00	00	75	00	00	57	6b	00	00	6f	00	00	00

OptyNop2 Distribution - 1

total: 6000
uniq: 141

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	12	12	12	39	39	00	00	12	11	11	11	39	39	00	00
10	12	12	12	11	39	39	00	00	12	12	12	12	39	39	00	00
20	12	11	12	12	39	39	00	39	12	12	11	12	39	39	00	39
30	11	11	12	12	39	39	00	39	11	11	12	11	39	39	00	39
40	39	39	39	3a	00	00	39	39	39	39	39	39	00	00	39	3a
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	39	39	00	12	00	11	00	00	00	00
70	3a	39	39	39	39	39	39	39	39	39	39	39	3a	39	39	39
80	12	12	00	12	12	11	11	12	12	12	00	00	00	00	00	00
90	39	39	39	3a	00	00	39	39	39	39	00	39	00	00	00	39
a0	00	00	00	00	00	00	00	00	3a	39	00	00	00	00	00	00
b0	3a	39	39	39	39	3a	39	39	39	39	39	39	00	00	3a	39
c0	12	12	00	00	00	00	00	00	00	00	00	00	00	00	00	00
d0	12	12	12	11	39	39	39	00	00	00	00	00	00	00	00	00
e0	39	39	39	39	00	00	00	00	00	00	00	39	00	00	00	00
f0	00	00	00	00	00	39	11	11	3a	39	00	00	39	39	11	11

OptyNop2 Distribution - 2

```
total: 6000
uniq:  141
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00 00 12 11 11 39 3a 00 00 11 12 12 12 39 39 00 00
10 11 11 11 11 39 39 00 00 11 12 11 11 39 39 00 00
20 12 12 12 12 39 3a 00 3a 12 11 12 12 39 39 00 39
30 11 12 12 11 39 3a 00 3a 12 12 12 12 39 39 00 39
40 39 3a 3a 39 00 00 39 39 39 39 39 3a 00 00 39 39
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 00 00 00 00 00 00 39 39 00 12 00 11 00 00 00 00
70 39 39 39 39 3a 39 39 39 39 39 39 39 39 3a 39 39
80 11 12 00 12 11 12 11 12 12 12 00 00 00 00 00 00
90 39 39 39 3a 00 00 39 3a 3a 3a 00 39 00 00 00 39
a0 00 00 00 00 00 00 00 00 39 39 00 00 00 00 00 00
b0 39 39 39 39 39 39 39 39 39 3a 39 39 00 00 39 39
c0 11 11 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0 12 12 11 11 39 39 3a 00 00 00 00 00 00 00 00 00
e0 3a 39 39 39 00 00 00 00 00 00 00 39 00 00 00 00
f0 00 00 00 00 00 39 11 12 39 39 00 00 39 39 10 10
```

ADMmutate and optyx-mutate Gzip'd

```
# ADMmutate

$ time ./nops 1000000| gzip -v >/dev/null
 27.3%
real    0m0.241s

# optyx's interz0ne mutate

$ time ./driver nop 1000000 | gzip -v >/dev/null
 29.7%
real    0m0.467s
```

OptyNop2 Gzip'd

```
# C version, save ESP and EBP

$ time ./waka 1000000 4 5 | gzip -v >/dev/null
 12.2%
real    0m11.900s

# save just ESP

$ time ./waka 1000000 4 | gzip -v >/dev/null
 11.7%
real    0m11.277s

# save nothing (good way to crash process)

$ time ./waka 1000000 | gzip -v >/dev/null
  8.3%
real    0m12.404s
```

Conclusion

- Benefits
  - Handles restricted bytes and registers
  - More versatile sled generation (nop stuffing, etc)
  - Implementation and theory are simple

Conclusion

- Benefits
  - Handles restricted bytes and registers
  - More versatile sled generation (nop stuffing, etc)
  - Implementation and theory are simple

- Possible Improvements
  - Support processor flags (nop stuffing)
  - Support 2-byte opcodes and escape groups
  - Improved byte scoring systems and look-ahead
  - Output according to a given byte distribution
  - Reduce the table sizes, memory usage

Part VI

Metasploit Framework 3.0

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

- Version 2.x is difficult to automate
- Everyone wants a completely automatic hack tool...

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

- Version 2.x is difficult to automate
- Everyone wants a completely automatic hack tool...
- ...Everyone else will hate us if we write one

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

- Version 2.x is difficult to automate
- Everyone wants a completely automatic hack tool...
- ...Everyone else will hate us if we write one

- External contributors inversely proportional to user base
- The ones who complain the loudest contribute the least

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

- Version 2.x is difficult to automate
- Everyone wants a completely automatic hack tool...
- ...Everyone else will hate us if we write one

- External contributors inversely proportional to user base
- The ones who complain the loudest contribute the least
- Don’t pick your language in hopes of contributions
- Perl is falling short as we grow more complex

Lessons learned

- "Hackers" run 98% Windows, 2% Unix
- Portability sucks, Windows sucks, and Cygwin sucks

- Version 2.x is difficult to automate
- Everyone wants a completely automatic hack tool...
- ...Everyone else will hate us if we write one

- External contributors inversely proportional to user base
- The ones who complain the loudest contribute the least
- Don’t pick your language in hopes of contributions
- Perl is falling short as we grow more complex

- Metasploit 2.0 mostly designed around exploits
- Payloads have grown more important and complex

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS's

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS's
- Embeddable for use in other tools

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS's
- Embeddable for use in other tools
- Strong custom automation
  - Test suites for the framework itself
  - Ability to test defensive infrastructure

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS's
- Embeddable for use in other tools
- Strong custom automation
  - Test suites for the framework itself
  - Ability to test defensive infrastructure
- Staged payloads as first class citizens

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS's
- Embeddable for use in other tools
- Strong custom automation
  - Test suites for the framework itself
  - Ability to test defensive infrastructure
- Staged payloads as first class citizens
- "Pivoting" through owned hosts

Metasploit 3.0 goals

- A capable language we *enjoy* writing in
  - Portability less important, support major OS’s
- Embeddable for use in other tools
- Strong custom automation
  - Test suites for the framework itself
  - Ability to test defensive infrastructure
- Staged payloads as first class citizens
- "Pivoting" through owned hosts
- Thread designed, not just thread safe

We love Ruby

- Used for our prototypes, leading candidate for msf3
- Clean and simple language that is easy to learn
- Strong object model, and we use every inch

We love Ruby

- Used for our prototypes, leading candidate for msf3
- Clean and simple language that is easy to learn
- Strong object model, and we use every inch
- Library support is decent, often better than Perl
- Native Win32 builds, Cygwin as backup
- 2.x will stay Perl and continue in parallel

Metasploit embedded

- Metasploit: A hacker tool framework

Metasploit embedded

- Metasploit: A hacker tool framework
- Tools built upon "framework-core" libraries
- Clear and documented SDK and interfaces

Metasploit embedded

- Metasploit: A hacker tool framework
- Tools built upon "framework-core" libraries
- Clear and documented SDK and interfaces
- Similar 2.x interfaces written by us
- Automation tools written by you

Payload model

- Payloads subscribe to unified API layers
- APIs emulate and extend the native Ruby APIs
- Port existing applications to be remote with no changes
- Great for scripting, great for testing

Payload model

- Payloads subscribe to unified API layers
- APIs emulate and extend the native Ruby APIs
- Port existing applications to be remote with no changes
- Great for scripting, great for testing
- Better post-exploitation tools, more fun
  - Mirror victims harddrive:
    `client.fs.dir.download('./victim', 'c:\\', true)`

Payload model

- Payloads subscribe to unified API layers
- APIs emulate and extend the native Ruby APIs
- Port existing applications to be remote with no changes
- Great for scripting, great for testing
- Better post-exploitation tools, more fun
  - Mirror victims harddrive:

    ```
    client.fs.dir.download('./victim', 'c:\\', true)
    ```

  - Migrate to a different process

    ```
    pid = client.sys.process['calc.exe']
    client.core.migrate(pid)
    ```

Payload model

- Payloads subscribe to unified API layers
- APIs emulate and extend the native Ruby APIs
- Port existing applications to be remote with no changes
- Great for scripting, great for testing
- Better post-exploitation tools, more fun
  - Mirror victims harddrive:
    ```
    client.fs.dir.download('./victim', 'c:\\', true)
    ```
  - Migrate to a different process
    ```
    pid = client.sys.process['calc.exe']
    client.core.migrate(pid)
    ```
  - Socket support, inefficent network pivoting

Payload model

- Payloads subscribe to unified API layers
- APIs emulate and extend the native Ruby APIs
- Port existing applications to be remote with no changes
- Great for scripting, great for testing
- Better post-exploitation tools, more fun
  - Mirror victims harddrive:
    ```
    client.fs.dir.download('./victim', 'c:\\', true)
    ```
  - Migrate to a different process
    ```
    pid = client.sys.process['calc.exe']
    client.core.migrate(pid)
    ```
  - Socket support, inefficent network pivoting
  - Support for Unix too, improved tools on their way

Other Stuff

- Threading
  - Ruby threads will work in theory
  - Meterpreter protocol asynchronous
  - Hopefully you can hack the planet in parallel

Other Stuff

- Threading
  - Ruby threads will work in theory
  - Meterpreter protocol asynchronous
  - Hopefully you can hack the planet in parallel
- Pivoting
  - Pivoting through custom metasploit proxying protocol
  - Fairly easy to implement, cross platform
  - More efficent than syscall proxying
  - "Network paths" should be really slick

Conclusion

- Should be cool
- Give us a year or more to make it

Part VII

Questions?