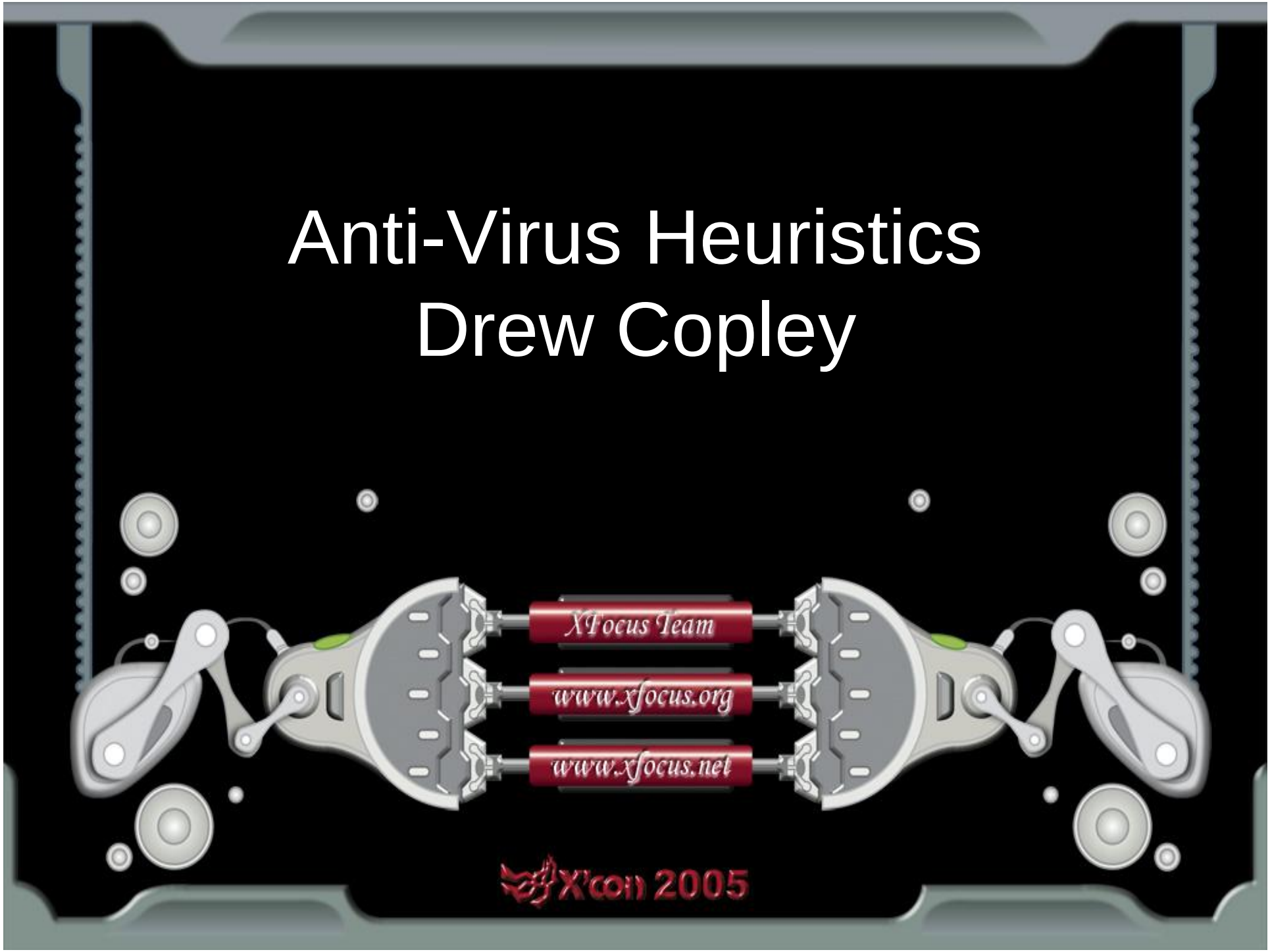
Anti-Virus Heuristics

Drew Copley




SECURITY
eEye
DIGITAL
VULNERABILITY
IS OVER

Introduction: Speaker

- **My name is Drew Copley, and I am a Senior Security Researcher at eEye Digital Security**
- **My AV research is the result of doing preliminary research for our products, at the design phase, as well as carrying through in prototyping models. Also, this information is derived from working on various "proof-of-concept" projects over the years.**

Introduction: Speech

- **The primary aim of this speech is to redefine the way you look at anti-malware protection technology, a bit.**
- **For years now, malware researchers of all kinds have been very aware of the fact that most mainstream AV products have been failing their users needlessly and extremely.**
- **What is heuristics, why has it not been widely turned on by default? Why does it generally fail, and can it really work ever?**

Why Anti-Virus Heuristics Should Interest You

- **"Heuristics" means 'investigative analysis', as everyone knows.**
- **But, what "heuristics" really means in this context is simply making more intelligent the process by which defensive software examines potential malware.**
- **Heuristics à dynamic – intelligent signatures…**
- **Non-Heuristic à static – dumb/blind signatures…**

The Heuristic Model Defined

- **Any heuristic module will have multiple modules within it**
- **All of these modules may be called heuristic modules, but they are, in fact, separate modules such as:**
  - **a module which is designed to deal with emulation**
  - **a module designed to utilize static analysis**
  - **a module designed to deal with packed/encrypted files**

Anti-Virus Pitfall: Heuristics Missing

- **A primary fault of many modern Anti-Virus products has been that they have relied almost exclusively on static, binary signatures. They have literally taken out the "intelligence" of "intelligent analysis".**
- **For years now, Anti-Virus solutions have been trivially overcome by script kiddies because of this flaw in their thinking.**
- **$ = one software buy**

  $$$$ … = continual money income through sig updates

Anti-Virus Pitfall: The Physical Security Metaphor

- **Physical security and computer security go hand in hand. The wall between the two is illusionary.**
- **Important metaphor to use: AV product as airport security checkpoint.**
- **Code Byte Signature engines alone would be like if they had pictures and were comparing them to people.**
- **This means no:**
  - **X-Ray machines**
  - **No pat downs for weapons**
  - **No observation of profuse sweating or shaking**
  - **Infamous criminals could waltz in wearing a paper bag over their head**
  - **You could walk in carrying a bazooka**
  - **No DNA check**
  - **No bomb/drug sniffing dogs**

Some simple, but effective examples, before continuing

Example: Generic Detection of Morphine Encrypted Files

- **Morphine Stack and Heap Values:**

  Size of Stack Reserve: 00100000
  Size of Stack Commit : 00010000
  Size of Heap Reserve : 00100000
  Size of Heap Commit : 00010000

- **It turns out they use Commit values which are entirely unique here. This was blindly found and proven, by baseline scanning against massive sets of non-malware binaries using an heuristic diagnostic system.**

- **Examples of typical PE Header Stack and Heap values:**

Size of Stack Reserve : 00100000
Size of Stack Commit : 00001000
Size of Heap Reserve : 00100000
Size of Heap Commit : 00001000

Size of Stack Reserve : 00100000
Size of Stack Commit : 00004000
Size of Heap Reserve : 00100000
Size of Heap Commit : 00001000

Example: Where Problem Is In

```
4679          NtHeaders.OptionalHeader.Magic:=IMAGE_NT_OPTIONAL_HDR_MAGIC;
4680          NtHeaders.OptionalHeader.MajorLinkerVersion:=Random(9)+1;
4681          NtHeaders.OptionalHeader.MinorLinkerVersion:=Random(99)+1;
4682          NtHeaders.OptionalHeader.BaseOfCode:=$00001000;                    //may change
4683          if ReqImageBase<>0 then NtHeaders.OptionalHeader.ImageBase:=RoundSize(ReqImageBase
4684          else if HostImageBase=$00400000 then NtHeaders.OptionalHeader.ImageBase:=RoundSize
4685          else NtHeaders.OptionalHeader.ImageBase:=$00400000;
4686          if not Quiet then WriteLn('ImageBase: ',IntToHex(NtHeaders.OptionalHeader.ImageBas
4687          NtHeaders.OptionalHeader.SectionAlignment:=$00001000;              //1000h = 4096
4688          NtHeaders.OptionalHeader.FileAlignment:=$00000200;                 //may change 200h
4689          NtHeaders.OptionalHeader.MajorOperatingSystemVersion:=$0004;
4690          NtHeaders.OptionalHeader.MajorSubsystemVersion:=$0004;
4691          NtHeaders.OptionalHeader.SizeOfHeaders:=$00000400;                 //may change
4692          NtHeaders.OptionalHeader.Subsystem:=HostSubsystem;
4693          NtHeaders.OptionalHeader.SizeOfStackReserve:=$00100000;
4694          NtHeaders.OptionalHeader.SizeOfStackCommit:=$00010000;             //may change
4695          NtHeaders.OptionalHeader.SizeOfHeapReserve:=$00100000;
4696          NtHeaders.OptionalHeader.SizeOfHeapCommit:=$00010000;
4697          NtHeaders.OptionalHeader.NumberOfRvaAndSizes:=$00000010;
4698
```

- As you can see, very simple problem for them to fix
- This is how it goes… but, you can get past versions, when you find a bug like this
- There is always another bug, always another fix
- Much better then slow processing, like KAV does
- Commits not good… would guess, because of values not lined up, maybe another reason

Example: Generic Detection of the Family of Gator Spyware

- **Version Information checking – very simple, information has to be unique**
- **“Gain Publishing” is a popular spyware maker who makes “Gator”.**
- **This information is contained in the resource section of a file.**
- **“Gain” and “Publishing” in that order, in the version information, with no other strings à unscientific, human judgement to be unique, unlikely false positive**

Example: “EduBot”, Polymorphic “Gaobot”

- **Time Date Stamp : FFFFFFFF**
- **(Image Section Name).edubot ç Very bad, this alone makes an excellent signature**
- **Import table open… example give away imports à the combination really makes the file stand out:**

  psapi.dll=EnumProcesses, shell32.dll=ShellExecuteA, ws2_32.dll=connect, ws2_32.dll=send, netapi32.dll=NetUseAdd, netapi32.dll=NetShareEnum, mpr.dll=WNetAddConnection2W, kernel32.dll=OpenProcess, and etc…

The Real Problems of Heuristic Agents

- **The primary, real problem of Heuristic Agents is incredibly simple: obfuscated files.**
- **Primarily, this has meant files which are packed/encrypted.**
- **The initial Heuristic technology worked well against malware, until packers/encryptors came into the scene. ***
- **This also helped the move in the mainstream AV industry away from Heuristics in their default product. ***

*** Peter Szor, "The Art of Computer Virus Research and Defense" , Symantec Press, 2005**

Example: Detecting Malware by UPX Packed Microsoft Files

- As UPX is an opensource packer, it is highly unlikely that Microsoft will ever choose to pack their files using it.
- A very simple and generic UPX detection routine simply consists of checking for the presence of a file section with the string “UPX” in it.
- For instance:
  (Image Section Name)UPX0 | (Virtual Size)00033000 | (Virtual Address)00001000 | (Size of Raw Data)00000000 | (Pointer to Raw Data)00000400 | (Characteristics)E0000080 | (Info) ERWU | (Percent)0.0%
- There are ways to obscure the packer used, and there are many edited versions of UPX out there, as much of the source is open.
- Example found live in wild, unknown to most Signature only engines
- Claimed to be “Microsoft” in the versioning information…
- This kind of “generic” check is immediately good for “classes” of malware – Subseven versions found which also trigger this

Example: Detecting Malware by Borland Forms in Microsoft Binaries

- **Microsoft and Borland do have various agreements**
- **Cold day in Hell when a Microsoft compiled file contains a Borland form**
- **Borland forms are separate files added as a resource section in the PE File, with the extension of .dfm, generally.**
- **Vast number of malware caught through this method (Delphi just rocks)**

The Irony of the Problem of Obfuscated Files

- **A primary problem with signature only based AV solutions is also packers/encryptors.**
- **This is the way it has worked in the underground for years: à somebody releases some trojan code to the underground. à AV companies make sig for code à trojan code is obfuscated à AV companies make sig for "new" code à … …**
- **Nearly endless iteration of this… so nearly endless iteration of trojan "versions"…**

The World of Morphing Malware

- These changing iterations of the malware bring us to the problem of morphing malware. Morphic malware means that the file itself is changing.

- Most common form of morphic malware:
  - NOT poly/meta morphic designs (Likely to become more popular – bugs, hard to do)
  - File changed through packers/encryptors
  - LESS common, but common, file changed through binary editing
  - file changed through the open source/source

Spyware: Motives and the Future

- **"Spyware" today means any type of spying agent, even including trojans and rootkits – not just commercial spyware**
- **Protection against Spyware – as opposed to other types of malware - is first and foremost for any proper anti-Malware agent.**
- **Two growth criteria:**
  - **The increase of the market for this stolen data**
  - **The increase of the market for zombied machines, in general. (Proxies, archives, DDoS botnets, etc)**

Packed/Encrypted Malware: The PE File Format

- **The Portable Executable File format is the format of w32 executables.**
- **For these purposes, the main things to be aware of is that there are these sections of w32 files:**
  - **There is an import section, APIs used by the file go here**
  - **There is an export section, APIs exported by the file go here. Generally, this are found in DLLs, not in executables.**
  - **There is the preliminary shell data, and then the Entry Point (EP) of the code**
  - **There is a definition of the sections in which the file is divided**

Packed/Encrypted Malware: Fast and Basic Look

- **There are a number of different ways to pack/encrypt files.**
- **The basic idea is simply that you shell the binary, moving the original binary, and cover it with a new shell**
- **The contents of the data of the original file are encrypted or packed or both encrypted and packed**
- **The packer/encryptor runs instead of the original binary, it decrypts/unpacks the contents of the original file in memory, then the original file is loaded and run in memory**

The Basic Attack Against Packed/Encrypted Malware

- **The most basic attack against packed/encrypted malware is simply in finding when the original file is made complete in memory, then dumping this process from memory to file**
- **Find the Original Entry Point (OEP) …**
- **There are a wide range of tools which have been developed to help in this activity out there**
- **Ollydbg is often used for this, scriptable plug-in, opensource database**

Automating the Cracking of Packed/Encrypted Binaries

- **"Why can this process not be automated and applied to modern AV?"**
- **Modern Heuristics is and will be working on this problem**
- **The foremost problem here: requiring the executable to execute or pseudo-execute (more typical scenario)**

The Emulator and Sandbox in AV

- **One way of doing this involves extensive API hooking on the module (Sandbox)**
- **A more complex, but safer way of dealing with the problem is that you emulate the entire system (Emulator)**
  - **Another advantage is easier flexibility to work with a static heuristic module**
- **Ultimately, you see similarities between the two methods, and you often may end up with the same work**

"Static Heuristics" and Packed/Encrypted Malware

- **"Static Heuristics" "VS" "Dynamic Heuristics"**
- **The parallel here is between the debugger model and the disassembler model.**
- **In both cases you translate the OP codes of the ASM correctly… Dynamic offers an edge against unknown code**
- **Static offers an edge for:**
  - **Anti-emulator tricks (mollasses code, fuse functionality, improper OP handling)**
  - **speed**

Static Heuristics and File Analysis, Part II (Finding Packed/Encrypted Files)

- **One of the first state type heuristic checks any heuristic engine needs to do is to check whether the file is packed/encrypted**
  - **EP in first section of file**
  - **Section names**
  - **Existance of other packer/encryption code signature**
  - **Entropy checks came from manual inspection, good usage of "Zero Order" entropy in PEiD**

Ø Expect attacks against every method

Example: Hex View of Encrypted Vs Unencrypted File

Unencrypted File Snippet in Raw Hex:

```
000004b0h: E4 94 41 77 00 00 00 00 D7 66 D0 77 C8 B3 D4 77 ; ä"Aw....×fÐwÈ³Ôw
000004c0h: C7 5D D0 77 62 68 D2 77 39 49 D0 77 92 38 D0 77 ; Ç]ÐwbhÒw9IÐw'8Ðw
000004d0h: B9 90 D0 77 54 41 D2 77 5A 9F D0 77 D4 3C D0 77 ; ¹ÐwTAÒwZŸÐwÔ<Ðw
000004e0h: 9C 4B D0 77 9C 6F D4 77 37 2E D0 77 E5 4D D0 77 ; œKÐwœoÔw7.ÐwåMÐw
000004f0h: 12 52 D0 77 AF 25 D0 77 0E 5C D0 77 C0 6B D0 77 ; .RÐw¯%Ðw.\ÐwÀkÐw
00000500h: E9 D1 D2 77 34 AF D0 77 AE 6B D0 77 1E 1D D0 77 ; éÑÒw4¯Ðw®kÐw..Ðw
00000510h: 01 28 D1 77 04 0E D1 77 B3 89 D0 77 FB DC D0 77 ; .(Ñw..Ñw³‰ÐwûÜÐw
00000520h: 9B 64 D0 77 73 81 D0 77 F6 DA D0 77 51 30 D0 77 ; >dÐws□ÐwöÚÐwQ0Ðw
00000530h: CA 3A D0 77 34 3B D0 77 2D 33 D0 77 63 E4 D0 77 ; Ê:Ðw4;Ðw-3ÐwcäÐw
00000540h: 33 80 D2 77 3B C6 D2 77 0D 3E D0 77 F9 2E D0 77 ; 3€Òw;ÆÒw.>Ðwù.Ðw
00000550h: 6B 31 D0 77 71 2F D0 77 50 4B D0 77 7E E7 D0 77 ; k1Ðwq/ÐwPKÐw~çÐw
00000560h: A0 57 D0 77 89 1A D0 77 1A BD D0 77 D3 DE D2 77 ;  WÐw‰.Ðw.½ÐwÓÞÒw
00000570h: C7 41 D0 77 DA DC D0 77 16 69 D0 77 9F 80 D2 77 ; ÇAÐwÚÜÐw.iÐwŸ€Òw
00000580h: 93 2F D0 77 00 82 D0 77 39 17 D0 77 45 18 D0 77 ; "/Ðw.‚Ðw9.ÐwE.Ðw
00000590h: 84 24 D0 77 75 35 D0 77 AF 3F D0 77 F7 17 D0 77 ; „$Ðwu5Ðw¯?Ðw÷.Ðw
000005a0h: 57 F2 D0 77 7B A5 D0 77 E2 45 D5 77 9D 65 D0 77 ; WòÐw{¥ÐwâEÕw□eÐw
000005b0h: 62 7B D4 77 02 8A D2 77 74 36 D0 77 57 3B D0 77 ; b{Ôw.ŠÒwt6ÐwW;Ðw
000005c0h: E2 33 D0 77 DE 42 D2 77 4C EB D0 77 00 00 00 00 ; â3ÐwÞBÒwLëÐw....
```

Encrypted File Snippet in Raw Hex:

```
000004b0h: 44 08 3D BA 1D 30 A5 8D CE 24 5D 28 35 78 D7 46 ; D.=°.0¥□Î$](5x×F
000004c0h: FE 81 BB CF C5 3B CC A0 D3 FA E3 A0 A3 93 0F 74 ; þ□»ÏÅ;Ì Óúã £".t
000004d0h: 36 BE 0C 3B B6 EB C8 13 0C CD 95 69 48 47 30 47 ; 6¾.;¶ëÈ..Í•iHG0G
000004e0h: 28 8A DC 22 0A 95 36 DE FE 58 54 1A FB 23 8F CA ; (ŠÜ".•6ÞþXT.û#□Ê
000004f0h: 34 4A EF EB 24 05 0B C1 EE E6 E7 C6 FE A8 32 83 ; 4Jïë$..Áîæç¨þ¨2ƒ
00000500h: 09 F1 C1 B5 E6 6B 45 A1 04 50 AD 6F 23 3C 7D DA ; .ñÁµækE¡.P-o#<}Ú
00000510h: D2 F1 3A BB E0 57 9F 08 7B D9 49 69 F8 0E E8 91 ; Òñ:»àWŸ.{ÙIiø.è'
00000520h: 71 27 62 CC 79 9B 3C 87 DB 32 00 3E 4A 53 40 97 ; q'bÌy›<‡Û2.>JS@—
00000530h: CC 75 37 C9 42 AF 0E 6C C8 38 3F EA 39 C4 10 E3 ; Ìu7ÉB¯.lÈ8?ê9Ä.ã
00000540h: 85 30 B6 A5 90 5D 9B 66 06 26 6A 1E 90 9A F4 D2 ; …0¶¥□]›f.&j.□šôÒ
00000550h: AD F4 C0 16 F8 68 B6 B1 2D 80 8B E0 69 8B FD A7 ; -ôÀ.øh¶±-€‹ài‹ý§
00000560h: 8C 55 F4 5B A7 0F CA BC 48 CA 93 23 AB B3 12 EC ; ŒUô[§.Ê¼HÊ"#«³.ì
00000570h: 88 1A 2D AC 44 73 57 AD EA 0E F3 1A 88 6C 17 95 ; ˆ.-¬DsW-ê.ó.ˆl.•
00000580h: E5 3B 58 46 C0 09 07 CA DC 62 47 69 8F 47 9B CB ; å;XFÀ..ÊÜbGi□G›Ë
00000590h: 1E ED CC 54 36 33 F0 91 7D C0 F4 57 7A F4 5D B4 ; .íÌT63ð'}ÀôWzô]´
000005a0h: ED 3B 5D CC 9A 94 BE A3 32 7C C2 1B 31 3A 24 CD ; í;]Ìš"¾£2|Â.1:$Í
000005b0h: 9C D0 23 CA 96 E0 E9 EE E2 2D C7 E4 53 C6 7A EA ; œÐ#Ê-àéîâ-ÇäSÆzê
000005c0h: C7 00 75 E6 EC B1 4B 57 81 4B E1 87 42 31 98 31 ; Ç.uæì±KW□Ká‡B1˜1
```

Heuristics Works Best With Other Modules

- **The detection of network Scanning Code is a good example of how heuristics can work best with other modules, lowering false positives**
- **Network Scanning detection is rather easily detected by the very action of scanning (Molasses code, however, is a good attack, but not perfect)**
- **If the malware is unknown and undetected by other protection agents, then it might still yet be caught by the IPS system**

Bypassing Detection by Bypassing APIs

- **A good example of more obscure file infection or hiding techniques involves bypassing the available APIs**

- **Raw NTFS engine / Device Object tricks…**
- **Detection of Object manipulation…**
  - **Presence of NTFS structures (for instance)**
  - **Call to Object manipulation routine**
  - **Disassembled code chunks found in other attack tools**

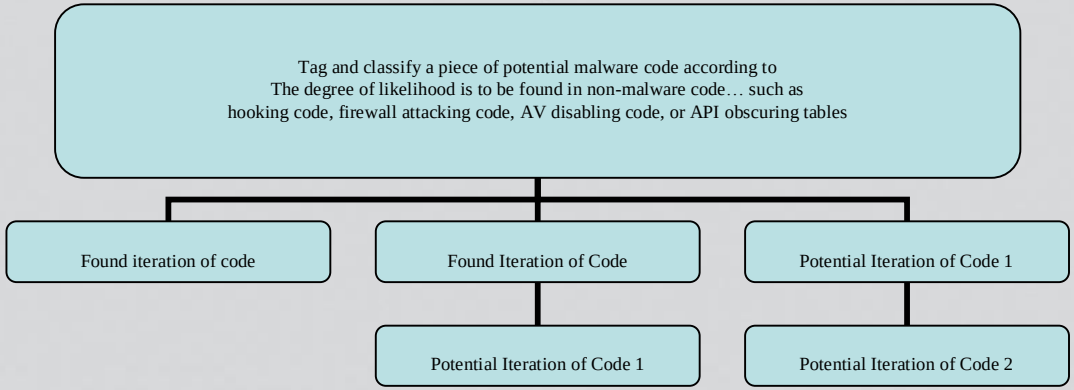
Ø **This also shows us the basics of heuristic analysis against other types of attacks**

Example: Bypassing APIs Give Other Fingerprints (Quick Run-Through)

```
type //ATTR_ATTRIBUTE
    S_RESIDENT = packed record
  dwLength: DWORD;
  wAttrOffset: WORD;
  uchIndexedTag: BYTE;
  uchPadding: BYTE;
    end;

    S_NONRESIDENT = packed record
  n64StartVCN: LONGLONG;
  n64EndVCN: LONGLONG;
  wDatarunOffset: WORD;
  wCompressionSize: WORD; // compression unit size
  uchPadding: array [0..3] of BYTE;
  n64AllocSize: LONGLONG;
  n64RealSize: LONGLONG;
  n64StreamSize: LONGLONG;
    end;

    U_ATTR = packed record
        case Integer of
        0:  (Resident: S_RESIDENT);
        1:  (NonResident: S_NONRESIDENT);
    end;

    NTFS_ATTRIBUTE = packed record
  dwType: DWORD;
  dwFullLength: DWORD;
  uchNonResFlag: BYTE;
  uchNameLength: BYTE;
  wNameOffset: WORD;
  wFlags: WORD;
  wID: WORD;
        attr: U_ATTR;
    end;
```

- **Bypassing the typical win32/NT APIs for raw disk access is fun**
- **This is an effective method for both attack and defensive code…**
- **Russinovich used this method for accessing the raw disk to detect rootkits that relied on hooking into the win32/NT APIs in his tool**
- **Though I have not wasted my time on building it, I could use the engine to create spyware which uses a redundant, distributed storage system such as with FEC (Forward Error Correction) across the raw disk**
- **There are many other potential attack possibilities with such a system**
- **Definitely, there are ways to obscure structures… but such things are just good examples for ways to generically detect and attack classes of such attack code and families of such malware**

Example: Evidence Collection and Rule Creation (Quick Run-Through)



- **This kind of table shows the evolution of various types of malware through the usage of various obscure and potential dangerous functions or pieces of code**
- **You may be able to predict some uses of this code before it is used**
- **In other cases you simply are reacting from what iteration is found in the wild – however this more dynamic method is useful for finding a larger body of families of malware then mere bytecode signature analysis**

Classes and Families

- Two concepts here to bear in mind: “Classes” and “Families”
- Heuristics provides generic protection against classes of code attacks and families of malware
- Malware tends to recycle code, like everything else
- Ludicrously repeated examples:
  - Run on reboot methods
  - Scan code
  - Hooking code
  - System management code

Automating Human Wisdom

- **Wide range of "bad behaviors" which go beyond mere API inspection**
- **Teach the system to divide between good and evil, operating on evidence, as human wisdom operates on knowledge**
- **Wrong to consider it as cybernetic style AI, rather should be thought of as a living system of Law**
- **Popular design error is to try to remove the human**

Ambiguous Morality and Evidence

- **There are certain actions which an application should never perform**
- **"Subjectivity" is objectivity from a perspective**
- **Actual cornerstone of Computer Security: Privileges**
- **A good principle for Heuristic system evidence collection:**
  - **Sheer weight system does not entirely work**
  - **Appearances can be deceiving**
  - **Suspicious evidence does have value for detection**
  - **Absolutary condemnatory code: smoking gun**

Example: Redirection, Subtleness In A Nutshell

- **Version of Reality -> Version of Reality -> Version of Reality -> Reality**
- **Obscuring API parameters -> Obscuring API parameters -> Obscuring API parameters -> Actual API Call with parameters**
- **Obscuring Content -> … -> Raw Code**
- **Example: encrypted API table -> Decryption of API import from table -> Dynamic call of API import**
- **Weakest links:**
  - **End result**
  - **Reused code**

Sidenote: Morphic Protection Software and System Integrity Solutions

- **Malware anti-anti-virus/protection modules**
- **Signature database driven**
- **Anti-Anti-Malware code is absolute condemnatory evidence: smoking gun**
- **Who is on the system first wins**
- **Two good ways of protection:**
  - **Baselining/crossview/system integrity systems**
  - **Morphic/Stealth anti-malware**

Example: Signs of Absolute Condemnation

```
.data:004681C2                  align 4
.data:004681C4 ; char aHttpWww_yythac[]
.data:004681C4 aHttpWww_yythac db 'http://www.yythac.com/images/mm.jpg',
.data:004681C4                                          ; DATA XREF: sub_
.data:004681C4                                          ; .text:loc_40569
.data:004681E8 ; char aBbypass[]
.data:004681E8 aBbypass        db 'bByPass',0           ; DATA XREF: sub_
.data:004681E8                                          ; sub_401CE0+10E↑
.data:004681F0 ; char aBbyserver[]
.data:004681F0 aBbyserver      db 'bByServer',0         ; DATA XREF: sub_
.data:004681F0                                          ; sub_401CE0+F1↑o
.data:004681FA                  align 4
.data:004681FC ; char aConpass[]
.data:004681FC aConpass        db 'ConPass',0           ; DATA XREF: sub_
.data:004681FC                                          ; sub_401CE0+D4↑o
.data:00468204 ; char aConport[]
.data:00468204 aConport        db 'ConPort',0           ; DATA XREF: sub_
.data:00468204                                          ; sub_401CE0+B7↑o
.data:0046820C ; char aReverse[]
.data:0046820C aReverse        db 'Reverse',0           ; DATA XREF: sub_
.data:0046820C                                          ; sub_401CE0+9A↑o
.data:00468214 ; char aSettings_0[]
.data:00468214 aSettings_0     db 'Settings',0          ; DATA XREF: sub_
.data:00468214                                          ; sub_401B80+4B↑o
.data:0046821D                  align 10h
.data:00468220 ; char aConnectmode[]
.data:00468220 aConnectmode    db 'ConnectMode',0       ; DATA XREF: sub_
.data:00468220                                          ; sub_401CE0+7D↑o
.data:0046822C ; char Text[]
.data:0046822C Text            db '-¼++¦-++¦+-_-í++2úí-¦++-¼+-¦-++úí',0
.data:0046822C                                          ; DATA XREF: sub_
.data:0046824E                  align 10h
.data:00468250 aIMhackeryythac db 'I',27h,'mhackeryythac1977',0 ; DATA X
```

- Common example of strings found in malware (once it is unpacked/decrypted, usually)
- At the simplest level, these really mark a family of malware
- Having some kind of unique string in you which was first used in a piece of malware is a great example of 'absolutely condemnatory code'.
- This is rather like tattooing a gang sign on your neck

Conclusions

- **Most of the conclusions here, you should have made with me and not even noticed. Even if you did not agree with some of my concepts, you should have now become aware of such things as:**
  - **Smarter signature based AV systems are the inevitable future**
  - **Morphic malware includes the process of hand edited malware, and this is a genuine, but surmountable malware problem (the term "morphic malware" had to be coined for this)**
  - **Physical security models are extremely useful to be applied to computer security models… but this is generally overlooked**

Conclusions, Part II

- **Conclusions**
  - **Packed/encrypted files is both a central problem of heuristics and a central pitfall of malware in general, the future of malware creation is in ingrained packed/encrypted techniques and generic evasion of packed/encrypted techniques**
  - **"Heuristics" is an illusionary term, but what problems exist for heuristics also exist for signature systems – an "heuristic" system, is, in fact, a signature system… a more dynamic signature system**
  - **The basis of evidence and suspicion is crucial to any good heuristic system… this is the difference between a system that is overcome with false positives and a system which can operate without any false positives**
  - **We must be aware of illusionary boundaries in defining terms, for instance, morphic malware must, by definition, include the problem of manually changed malware… And the various modules of a heuristic system, such as an emulation module or a static analysis module should not by necessity define the entire heuristic system itself**
  - **The best heuristic system should not attempt to be all things, but should be a piece of the whole in a complete security system… this is the way to avoid false positives, by not putting too much stress on any single module or system…**
  - **False positives are bugs in any system, and they degrade the complete functionality in any system… they should not be accepted as necessary for any system, but should be considered problems which are surmountable…**

Conclusions, Part III

- **Conclusions**
  - **Most mainstream AV has not been keeping up with attacks from the wild, when they have always had the capabilities to do far better**
  - **Even in such a new industry, as computer security, tradition can quickly be created and kept and this must be fought against and avoided to produce results**
  - **A solid evidence and suspicion based heuristics systems can operate, but it must be programmed well by heuristic signature coders… and it must not avoid basic dictates of the criminal justice world to operate, but must embrace them**
  - **Defensive software must embrace offensive software and the reverse is true, otherwise there is a dangerous disconnect**
  - **Many of the conclusions made in this paper have used common sense style arguments with readily observable evidence supporting them for the purposes of winning the ideas**
  - **Code which is absolutely malicious in nature means code which is more provably "bad", such as anti-anti-virus code, spying code, spreading code**
  - **Various other conclusions, understanding about current heuristics and future heuristics… such as the usefulness of understanding "perspective" and "subjectivity" within a "moral system", and viewing the heuristic system as a moral system… and the classification of heuristic protection possible, that being against "classes of attack" and "families of malware"**

Various Credits

- Foremost, I must credit all of the malware writers out there who have greatly aided the advance of the science of the defensive arts, premier among these groups for full disclosure, open source type projects have been 29a... also the various contributors to the rootkit.com projects... it is, however, impossible to fully credit the works of all of these people... further it is also impossible to credit all of the malware researcher's works out there whose work has influenced this
- Peter Szor's book, mentioned in this speech, and his various other writings have been extremely helpful as a coherent cataloguing of the techniques used by malware authors -- I have marked out where his work has been useful herein
- Mark Russinovich's paper on AV engines and general low level papers have been extremely instrumental and educational in many of these areas over the years (I did not, however, base my NTFS system on his work, if anyone is wondering, but due credit goes there to the Linux NTFS team for their extensive documenting of the binary format of the NTFS file system.)

Credits, Continued

- Credits further mentioned, “CrazyLord’s” paper in Phrack on Physical Memory exploitation and PEiD’s Zero Order entropy method – their exact method used was found in their forums
- Nico Brulez’s Honeynet challenge was referenced in this work, especially regarding the usage of a virtual machine in terms of AV systems
- Thanks to Derek Soeder for his extensive help in proofreading this document
- Most of the conclusions I came to first through experimentation and prototyping, through work in this field, on both the offensive and defensive sides of things: many conclusions I later found supported elsewhere. My initial API analysis tool I wrote and released as shareware three years ago. My initial interest in heuristics dates back some underground work some years ago.