

Static Malware Detection

The background of the slide features a complex arrangement of interlocking gears. The gears are rendered in a light blue, semi-transparent style, with a white wireframe grid overlaid on them. The grid consists of numerous thin lines that form a dense, interconnected pattern across the entire scene. The overall color palette is a gradient of blues, from a deep navy blue at the top to a lighter, almost white blue at the bottom. The gears are positioned at various depths and angles, creating a sense of three-dimensional mechanical complexity.

Luis Miras & Ken Steele

Goals

- **Stimulate research in static binary analysis for malware detection purposes**
- **Provide a toolset for exploration of Portable Executables**
- **Show consequences of not blocking packed/compressed executables at the gateway.**
- **Provide a mechanism to aid in mitigating 0day “mass mailer” worms.**

Philosophy of Detection

- **Signature based**
 - **Pros**
 - **Concise**
 - **Cons**
 - **Update game**
- **Heuristics**
 - **Pros**
 - **Adaptable to new attacks**
 - **Cons**
 - **False positives**

Our Approach

- **Python PE parsing library**
- **Why Python?**
- **Packer Detection**
 - Signature (complete)
 - Heuristics (in progress)
- **Block “mass mailer” worms at the MTA**
 - Scan all attachments (extensions don't matter)
 - Exit when not PE
 - No patch for stupidity

Agenda

- **Email Worm Overview**
- **PE File Format Overview**
- **Packer Overview**
- **Packer Detection**
- **Library and Tools**
- **Demos**
- **Future Roadmap**
- **Other Research**
- **Resources**
- **Questions**

Email Worm Overview



What is an Email Worm?

- Also known as “mass mailing” worms
- Propagate via tempting users to execute email attachments
 - Does not generally involve exploiting vulnerabilities
- Some email worms open backdoors
- Iterates through user contact lists and redistributes itself
- Today’s worms have evolved to use sophisticated obfuscation techniques

Example Email Worms

- **Bagle(Beagle)**
 - Backdoors on high TCP ports
 - Variants: 28 (as of 6/04)
- **Netsky**
 - Functions varied between virus strains
 - Beeping sounds on specific dates
 - Variants: 29 (as of 6/04)
- **MyDoom**
 - DoS SCO (awww.. poor SCO)
 - Broke the record for the fastest spreading “mass mailing” worm
 - Variants: 10 (as of 6/04)

(thanks wikipedia!)

PE File Format



PE File Format Overview

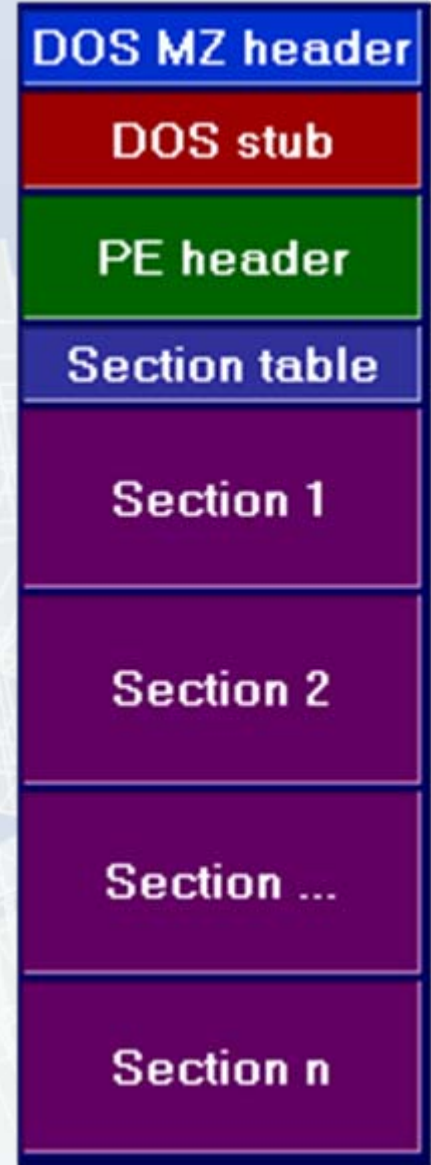
- **PE stands for Portable Executable**
- **It is the standard executable binary format for all win32 OS.**
- **It contains a DOS header and stub code for compatibility purposes.**

PE File Format Diagram

- **DOS Header must contain MZ**
- **DOS Stub**
 - “This program cannot be run in DOS mode”
- **PE Header**
 - File Header
 - Optional Header
 - Data Directory
 - Section Table
 - Sections

Image from Iczelion PE tutorial

(<http://win32assembly.online.fr/pe-tut1.html>)



IMAGE_DOS_HEADER

```
typedef struct _IMAGE_DOS_HEADER {  
    WORD e_magic;           // Magic number  
    WORD e_cblp;           // Bytes on last page of file  
    WORD e_cp;             // Pages in file  
    WORD e_crlc;          // Relocations  
    WORD e_cparhdr;       // Size of header in paragraphs  
    WORD e_minalloc;      // Minimum extra paragraphs needed  
    WORD e_maxalloc;      // Maximum extra paragraphs needed  
    WORD e_ss;            // Initial (relative) SS value  
    WORD e_sp;            // Initial SP value  
    WORD e_csum;          // Checksum  
    WORD e_ip;            // Initial IP value  
    WORD e_cs;            // Initial (relative) CS value  
    WORD e_lfanlc;        // File address of relocation table  
    WORD e_ovno;          // Overlay number  
    WORD e_res[4];        // Reserved words  
    WORD e_oemid;         // OEM identifier (for e_oeminfo)  
    WORD e_oeminfo;       // OEM information; e_oemid specific  
    WORD e_res2[10];      // Reserved words  
    LONG e_lfanew;        // File address of new exe header  
} IMAGE_DOS_HEADER, *PIIMAGE_DOS_HEADER;
```

IMAGE_NT_HEADERS

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD Signature;  
    IMAGE_FILE_HEADER FileHeader;  
    IMAGE_OPTIONAL_HEADER32  
    OptionalHeader;  
} IMAGE_NT_HEADERS32,  
*PIMAGE_NT_HEADERS32;
```

IMAGE_FILE_HEADERS

```
typedef struct _IMAGE_FILE_HEADER{  
    WORD    Machi ne;  
    WORD    NumberOfSecti ons;  
    DWORD   Ti meDateStamp;  
    DWORD   Poi nterToSymbol Tabl e;  
    DWORD   NumberOfSymbol s;  
    WORD    Si zeOf0pti onal Header;  
    WORD    Characteri sti cs;  
} IMAGE_FILE_HEADER,  
*PI MAGE_FILE_HEADER;
```

IMAGE_OPTIONAL_HEADER32

```
typedef struct _IMAGE_OPTIONAL_HEADER {  
    WORD   Magic;  
    BYTE   MajorLinkerVersion;  
    BYTE   MinorLinkerVersion;  
    DWORD  SizeOfCode;  
    DWORD  SizeOfInitializedData;  
    DWORD  SizeOfUninitializedData;  
    DWORD  AddressOfEntryPoint;  
    DWORD  BaseOfCode;  
    DWORD  BaseOfData;  
    DWORD  ImageBase;  
    DWORD  SectionAlignment;  
    DWORD  FileAlignment;  
    WORD   MajorOperatingSystemVersion;  
    WORD   MinorOperatingSystemVersion;  
    WORD   MajorImageVersion;  
    WORD   MinorImageVersion;  
    WORD   MajorSubsystemVersion;  
    WORD   MinorSubsystemVersion;  
    DWORD  Win32VersionValue;  
    DWORD  SizeOfImage;  
    DWORD  SizeOfHeaders;  
    DWORD  CheckSum;  
    WORD   Subsystem;  
    WORD   DllCharacteristics;  
    DWORD  SizeOfStackReserve;  
    DWORD  SizeOfStackCommit;  
    DWORD  SizeOfHeapReserve;  
    DWORD  SizeOfHeapCommit;  
    DWORD  LoaderFlags;  
    DWORD  NumberOfRvaAndSizes; // number of members in the data directory  
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]; // array of data dirs  
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

DATA DIRECTORIES

IMAGE_DIRECTORY_ENTRY_EXPORT	equ 0
IMAGE_DIRECTORY_ENTRY_IMPORT	equ 1
IMAGE_DIRECTORY_ENTRY_RESOURCE	equ 2
IMAGE_DIRECTORY_ENTRY_EXCEPTION	equ 3
IMAGE_DIRECTORY_ENTRY_SECURITY	equ 4
IMAGE_DIRECTORY_ENTRY_BASERELOC	equ 5
IMAGE_DIRECTORY_ENTRY_DEBUG	equ 6
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE	equ 7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR	equ 8
IMAGE_DIRECTORY_ENTRY_TLS	equ 9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	equ 10
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT	equ 11
IMAGE_DIRECTORY_ENTRY_IAT	equ 12
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT	equ 13
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR	equ 14

IMAGE_SECTION_HEADER

```
IMAGE_SECTION_HEADER STRUCT
```

```
  Name1 db IMAGE_SIZEOF_SHORT_NAME dup(?)
```

```
  union Misc
```

```
    Physical Address      dd ?
```

```
    Virtual Size         dd ?
```

```
  ends
```

```
  Virtual Address        dd ?
```

```
  SizeOfRawData          dd ?
```

```
  PointerToRawData       dd ?
```

```
  PointerToRelocations   dd ?
```

```
  PointerToLinenumbers   dd ?
```

```
  NumberOfRelocations   dw ?
```

```
  NumberOfLinenumbers   dw ?
```

```
  Characteristics       dd ?
```

```
IMAGE_SECTION_HEADER ENDS
```

```
IMAGE_SIZEOF_SHORT_NAME equ 8
```

IMAGE_IMPORT_DESCRIPTOR

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
```

```
union
```

```
    Characteristics          dd ?
```

```
    OriginalFirstThunk      dd ?
```

```
ends
```

```
    TimeDateStamp           dd ?
```

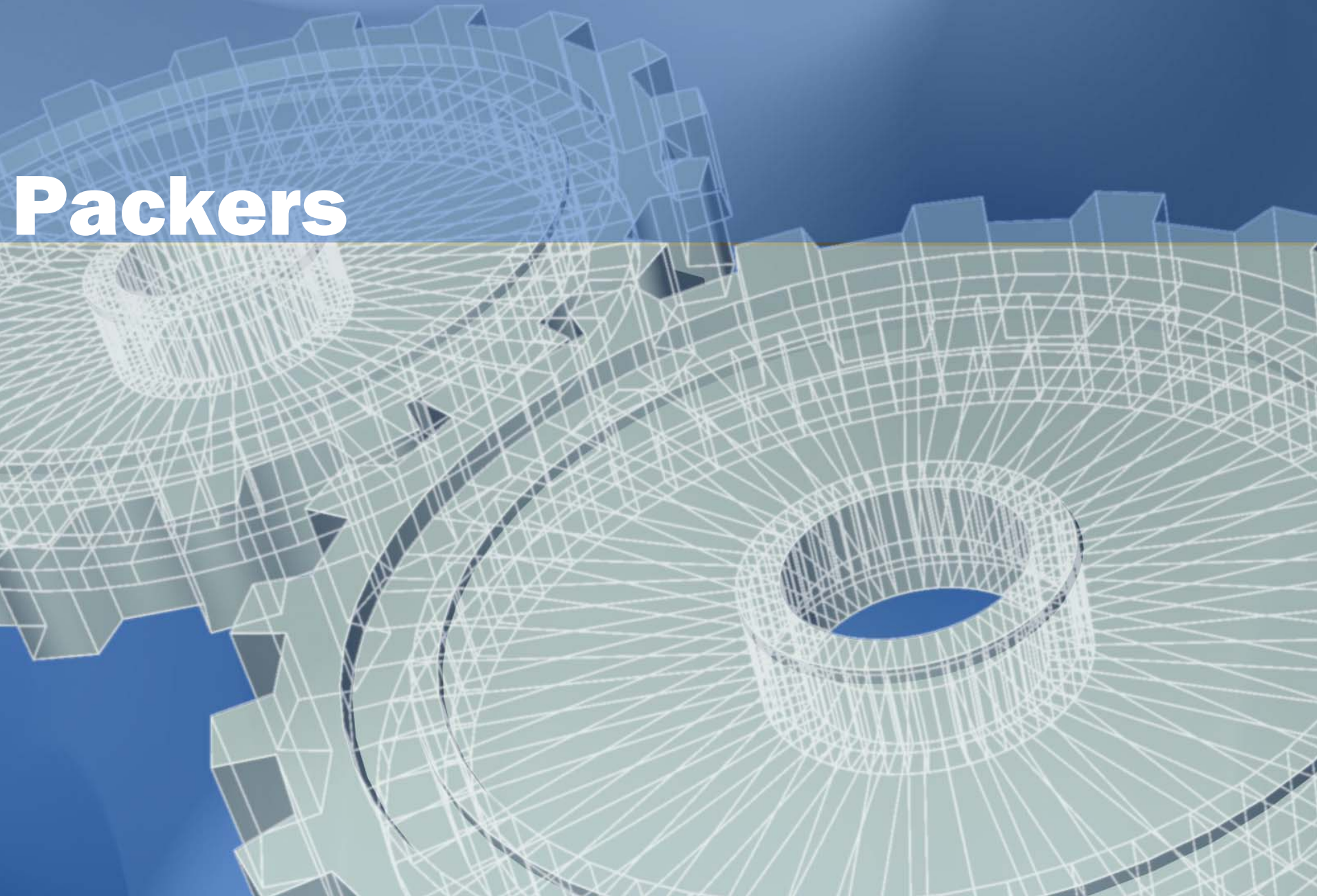
```
    ForwarderChain          dd ?
```

```
    Name1                   dd ?
```

```
    FirstThunk              dd ?
```

```
IMAGE_IMPORT_DESCRIPTOR ENDS
```

Packers



Packer Overview

- **Packers are executable compressors**
- **Usually consist of a decompression stub and compressed data**
- **They originated in an effort to make reverse engineering more difficult.**
- **Primary users of packers are shareware authors and malware authors.**
- **Some popular packers are UPX, ASPack, PEcompact, and Armadillo.**

Packer Features

- **Packers are intended to decompress the executable during loading.**
- **Anti-debugging is built-in.**
- **Junk code is inserted in order to overwhelm the reverse engineer.**
- **Exception handling is abused**
- **Jumps into the middle of longer instructions are used to fool disassemblers.**

Packer Example

- Below is putty.exe original and packed with UPX 1.25

223,744 putty.exe – packed

380,928 putty.exe – original

Packed Imports (11)

LoadLibraryA
GetProcAddress
ExitProcess
RegEnumKeyA
ChooseFontA
LineTo
ImmGetContext
ShellExecuteA
GetDC
PlaySoundA
OpenPrinterA

Original Imports(251)

RegEnumKeyA
RegCloseKey
RegCreateKeyA
RegSetValueExA
RegOpenKeyA
[...]
FindNextFileA
FindClose
VirtualFree
SetFilePointer
SetStdHandle

Packed File in IDA Pro

IDA - C:\projects\putty.exe

File Edit Jump Search View Debugger Options Windows Help

Text UPX1

IDA View-A Hex View-A Exports Imports Names Functions Strings Structures En Enums

```
UPX1:0046576F dd 80372000h, 2F8D80D7h, 92490006h, 4924h, 0FF5000h
UPX1:00465CBF align 10h
UPX1:00465CC0
UPX1:00465CC0 public start
UPX1:00465CC0 start:
UPX1:00465CC0 pusha
UPX1:00465CC1 mov esi, offset dword_431000
UPX1:00465CC6 lea edi, [esi-30000h]
UPX1:00465CCC push edi
UPX1:00465CCD or ebp, 0FFFFFFFh
UPX1:00465CD0 jmp short loc_465CE2
UPX1:00465CD0 ; -----
UPX1:00465CD2 align 8
UPX1:00465CD8 loc_465CD8: ; CODE XREF: UPX1:loc_465CE9↓j
UPX1:00465CD8 mov al, [esi]
UPX1:00465CDA inc esi
```

Names window

Name	Address	P
start	00465CC0	P

Strings window

Address	Length	Type	String
---------	--------	------	--------

...initializing bd_bindiff v1.6-Build:10 plugin...no configuration found in Registry, creating...
[!] Setting CPU type to x86
... init successful (staying in mem) !
...initializing bd_funcgraph v1.1 plugin...
[!] CPU is metapc
[!] Setting CPU type to x86
... init successful !
Propagating type information...
Function argument information is propagated
The initial autoanalysis is finished.

AU: idle Down Disk: 136GB 000350BF 00465CBF: UPX1:00465CBF

Original File in IDA Pro

The screenshot displays the IDA Pro interface for the file `C:\projects\peinfo\putty.exe`. The main window shows assembly code for a subroutine starting at address `004321BD`. The code includes a `__stdcall` signature for `WinMain` and various local variables like `Filename`, `WndClass`, and `Msg`.

The **Names window** on the right lists the following symbols:

Name	Address
<code>nullsub_2</code>	<code>00409F38</code>
<code>__beep</code>	<code>0042D028</code>
<code>operator new(uint,void *)</code>	<code>00430E6A</code>
<code>DialogFunc</code>	<code>00430EF6</code>

The **Strings window** below it shows the following entries:

Address	Length	T...	String
<code>...data:00...</code>	<code>00000021</code>	C	<code>---- BEGIN SSH2 ENCRYPT</code>
<code>...data:00...</code>	<code>0000001E</code>	C	<code>---- Session restarted ----</code>
<code>...data:00...</code>	<code>0000000C</code>	C	<code>----BEGIN</code>

At the bottom, a console window shows the output of a Python interpreter (version 2.3.5) initializing the x86emu plugin and setting the CPU type to x86. The status bar at the very bottom indicates the current address is `004321BD` and the function is `WinMain(x,x,x,x)`.

Packer Detection



Packer Detection Methods

- **Signature based**
 - Executable code signatures
- **Heuristics**
 - Entropy Checks
 - **IMPORT ADDRESS TABLE**
 - Other Checks (not exclusive to packers)

Executable Code Signatures

- There are many signature based tools in use by the RE/Cracking community. The best one is PEiD. <http://peid.has.it>
- This method compares bytes at the program entry point against a database.
- Signatures are designed so that they ignore non-opcode bytes.

Signature Example

- **Below is a signature for a Borland C++ DLL:**

[Borland C++ DLL]

signature = A1 ?? ?? ?? ?? C1 E0 02 A3

ep_only = true

- **The following assembly matches the signature:**

A1 37130300 MOV EAX, DWORD PTR DS: [31337]

C1E0 02 SHL EAX, 2

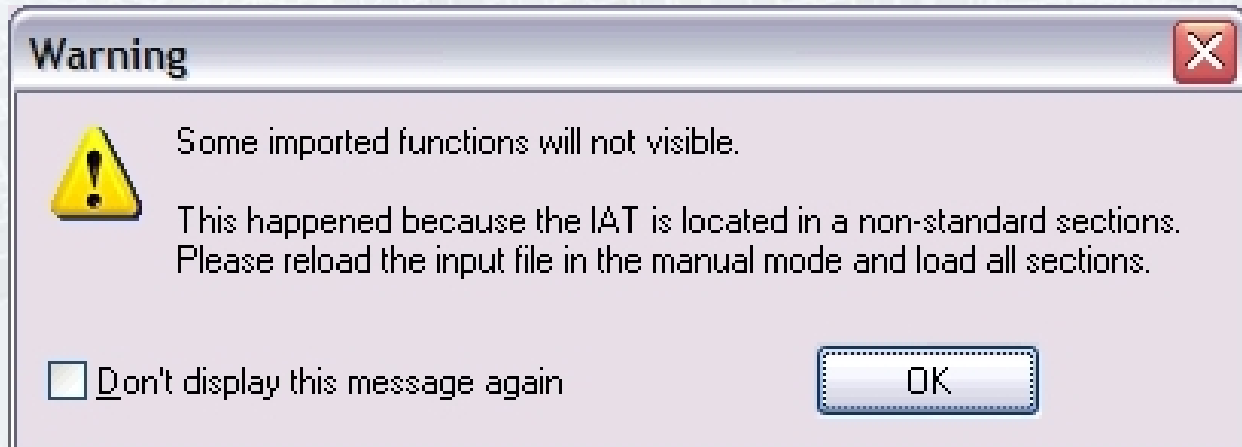
A3 37130300 MOV DWORD PTR DS: [31337], EAX

Entropy Checks

- **Byte distribution or entropy is changed by packers**
- **Sections as well as Import tables can be checked**

Import Address Table

- **IAT is in non-standard section**
- **IMAGE_IMPORT_DIRECTORY inconsistencies**



Other Checks

- **Strings**
- **Very few Imports**
- **Differences between section's VirtualSize and SizeofRawData**
- **Non standard NumberOfRvaAndSizes**

Library and Tools



Library and Tools

- **PELP**

- **pelp.py** PE Library (Python)

- **Tools**

- **pelpUtil.py** Performs packer detection and dumps Dependencies, Imports and Sections.

- **smdScan.py** Qmail Scanner plugin

PELP – PE Library (Python)

- **Portable Executable Library written in Python.**
- **Parses PE header, sections and imports**
- **Construct PE Object**
- **Where can I get it?**
 - <http://sourceforge.net/projects/pelp>

PELP Example

- **The following will print out DLL dependencies:**

```
peFile = pelp.PE( fileName )  
for dll in peFile.File.IMPORT_TABLE:  
    print "[%s]" % dll.dllName
```

pelpUtil.py

- **pelpUtil is a command line tool**
- **Uses the PELP library**
- **similar to Microsoft's dumpbin**
- **It also provides packer detection**

Usage: `pelpUtil.py -f <filename>`
 `[-i | --imports`
 `-d | --depends`
 `-s | --sections`
 `-p | --packscan`
 `-u | --userdb <userdb>]`

Packer Detection at the MTA

- **Goal is to block packed executables at the MTA**
- **Choosing an MTA**
 - **Open Source**
 - **Good Security Record**
 - **Extensible through plugins**
 - **High market penetration**

QMail and QMail Scanner

- **QMail**
 - Good alternative to Sendmail
 - Security track record
 - Extensible with QMail Scanner
 - Large userbase to help us test ☺
 - Available from <http://www.qmail.org>
- **Qmail Scanner**
 - Content scanner for Qmail
 - Forwards email content to 3rd party tools
 - Available from <http://qmail-scanner.sf.net>

smdScan

- **smdScan is a tool that provides packer detection for Qmail**
- **Design goals**
 - **Low CPU utilization**
 - **Easy to add new signatures**
 - **Basic policy management**
 - **Works well with others**

smdScan Usage

Usage:

```
smdScan.py ( -f <filename> |  
             -d <directory> )  
[ -q | --quiet  
  -u | --userdb <userdb> ]
```


SMD Packer Detection DB

- **Scanning over 750 signatures**
- **Example SMD database entries:**

[Microsoft Visual C++ v6.0]

signature = 55 8B EC 83 EC 50 53 56 57 BE
 ?? ?? ?? ?? 8D 7D F4 A5 A5 66 A5 8B

ep_only = true

action = allow

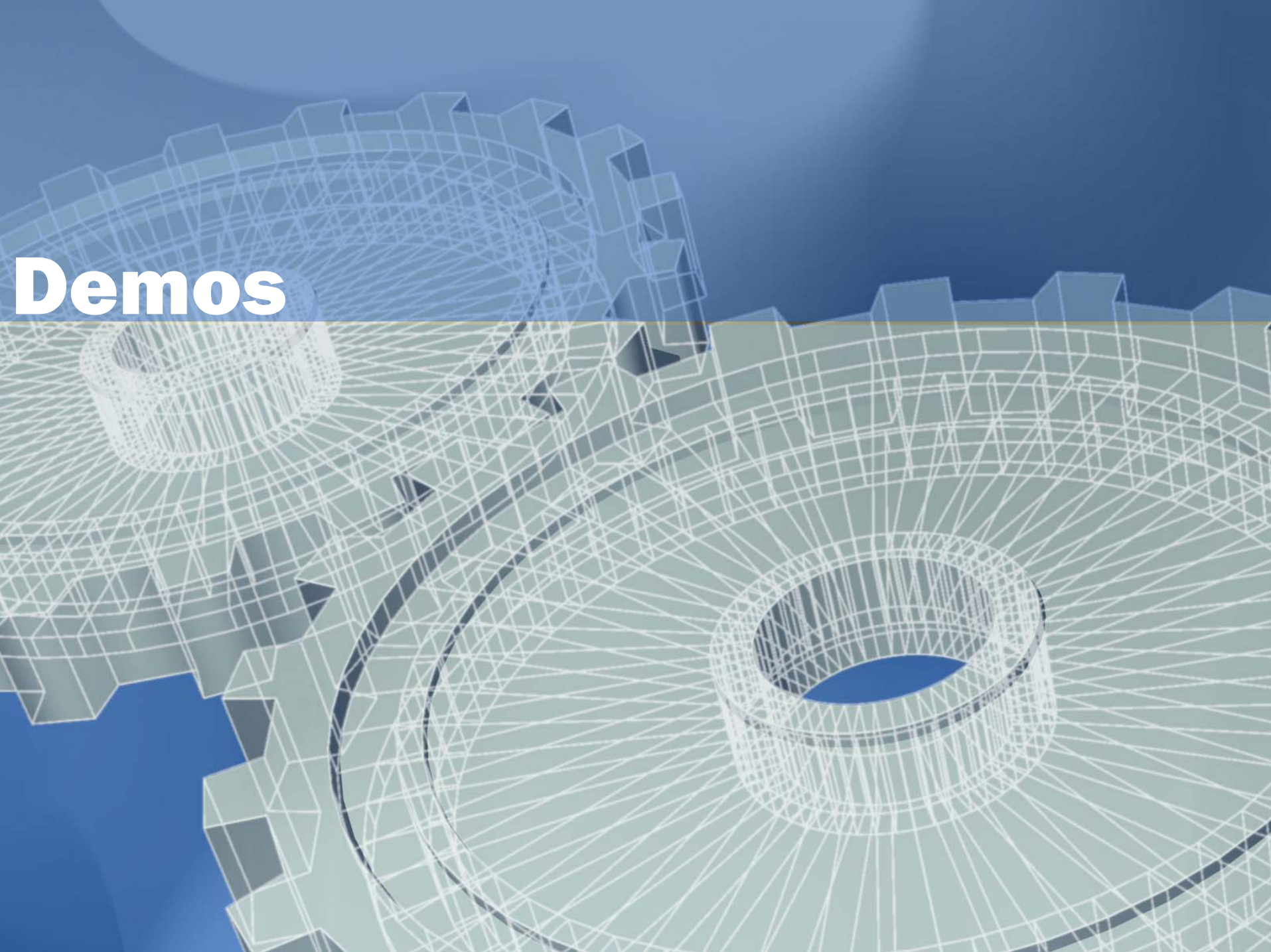
[PECompact v1.4x+]

signature = EB 06 68 ?? ?? ?? ?? C3 9C 60
 E8 02 ?? ?? ?? 33 C0 8B C4 83 C0 04 93
 8B E3 8B 5B FC 81

ep_only = true

action = deny

Demos



Demo Configuration

- **Fedora Core 4 VMWare image**
 - Qmail
 - Qmail Scanner
 - ClamAV
 - smdScan
- **Windows XP Host OS**
 - Thunderbird

Demo Diagram

Attacker

Victim

MTA



Future Roadmap



Short Term Goals

- **SMD**
 - Add MD5/SHA1 support for allow/deny actions
 - Finish PE level analysis checks.
 - Implement disassembly library
- **PELP**
 - Support for modifying PE files (adding sections, imports, etc.)
 - Microsoft .NET support
- **OllyDbg plugin**

Long Term Goals

- **Plugins for MTAs other than QMail**
- **Develop plugin framework**
 - Custom unpackers
 - Third party modules
- **Deep analysis via static disassembly and emulation**
- **Advanced heuristics**
 - Cyclomatic Complexity

Other Research



Other Research

- **pype** - <http://dkbza.org/pype/pype.html>
- **Cyclomatic Complexity** - http://www.openrce.org/articles/full_view/11

Resources

- PEiD - <http://peid.has.it/>
- IDA Pro - <http://datarescue.com/idabase/>
- Iczelion PE tutorial - <http://win32assembly.online.fr/pe-tut1.html>
- 0x90.exe - <http://www.honeynet.org/scans/scan33/>
- UPX – <http://upx.sf.net>
- ASPack - <http://www.aspack.com>
- EXE Shield – <http://www.exeshield.com>
- QMail – <http://www.qmail.org>
- QMail Scanner – <http://qmail-scanner.sf.net>
- ClamAV – <http://www.clamav.net>
- Signatures compiled from various sources including - <http://www.exetools.com>

Questions ?



Imiras@intrusion.com ksteele@intrusion.com

**<http://sf.net/projects/pelp>
<http://sf.net/projects/smd>**

