# **Static Malware Detection**



#### 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 Oday "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



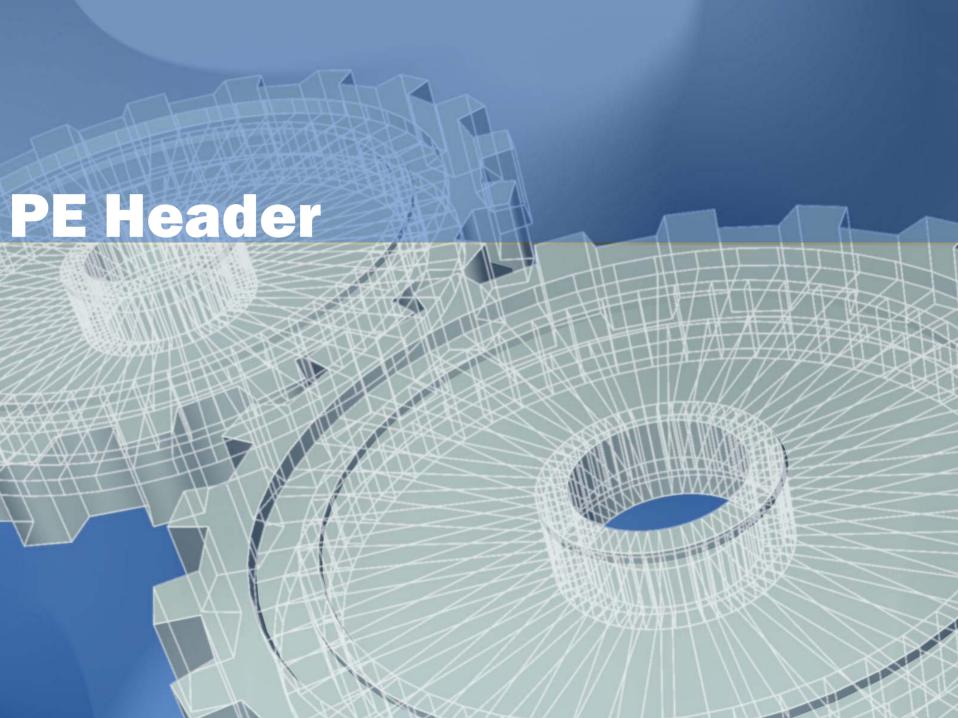
### What is an Email Worm?

- 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 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)

DOS MZ header DOS stub PE header Section table Section 1 Section 2 Section ... Section n

#### IMAGE\_DOS\_HEADER

```
typedef struct _I MAGE_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
              // Initial IP value
 WORD e_i p;
 WORD e_cs; // Initial (relative) CS value
 WORD e_lfarlc; // 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_I fanew; // File address of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

## IMAGE\_NT\_HEADERS

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

## IMAGE FILE HEADERS

```
typedef struct _IMAGE_FILE_HEADER{
  WORD Machine;
  WORD NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbol Table;
  DWORD NumberOfSymbols;
  WORD SizeOfOptional Header;
  WORD Characteristics;
 IMAGE FILE_HEADER,
 *PIMAGE FILE HEADER;
```

## IMAGE\_OPTIONAL\_HEADER32

```
typedef struct IMAGE_OPTIONAL_HEADER {
WORD Magic;
BYTE MajorLinkerVersion;
BYTE Mi norLi nkerVersi on:
DWORD SizeOfCode:
DWORD SizeOfInitializedData:
DWORD SizeOfUninitializedData:
DWORD AddressOfEntryPoint;
DWORD BaseOfCode;
DWORD BaseOfData:
DWORD ImageBase;
DWORD SectionAlignment:
DWORD FileAlignment;
WORD MajorOperatingSystemVersion;
      Mi norOperati ngSystemVersi on;
WORD
WORD
      Majorl mageVersion;
WORD Mi norl mageVersi on;
WORD MajorSubsystemVersion;
WORD Mi norSubsystemVersi on;
 DWORD Win32VersionValue;
DWORD SizeOfImage;
 DWORD SizeOfHeaders:
DWORD CheckSum:
WORD Subsystem;
WORD DIICharacteristics:
DWORD SizeOfStackReserve:
DWORD SizeOfStackCommit:
DWORD Si zeOfHeapReserve;
DWORD Si zeOfHeapCommi t;
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

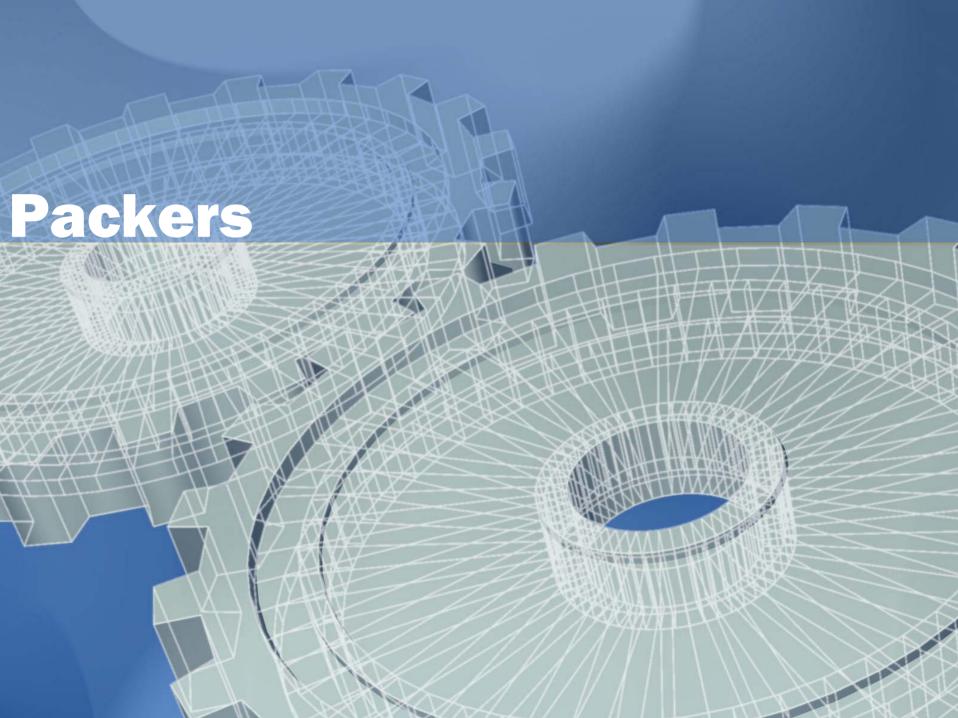
I MAGE_DI RECTORY_ENTRY_EXPORT	equ	0
I MAGE_DI RECTORY_ENTRY_I MPORT	equ	1
I MAGE_DI RECTORY_ENTRY_RESOURCE	equ	2
I MAGE_DI RECTORY_ENTRY_EXCEPTI ON	equ	3
I MAGE_DI RECTORY_ENTRY_SECURI TY	equ	4
I MAGE_DI RECTORY_ENTRY_BASERELOC	equ	5
I MAGE_DI RECTORY_ENTRY_DEBUG	equ	6
I MAGE_DI RECTORY_ENTRY_ARCHI TECTURE	equ	7
I MAGE_DI RECTORY_ENTRY_GLOBALPTR	equ	8
I MAGE_DI RECTORY_ENTRY_TLS	equ	9
I MAGE_DI RECTORY_ENTRY_LOAD_CONFI G	equ	10
I MAGE_DI RECTORY_ENTRY_BOUND_I MPORT	equ	11
I MAGE_DI RECTORY_ENTRY_I AT	equ	12
I MAGE_DI RECTORY_ENTRY_DELAY_I MPORT	equ	13
I MAGE_DI RECTORY_ENTRY_COM_DESCRI PTOR	equ	14

#### IMAGE\_SECTION\_HEADER

```
IMAGE_SECTION_HEADER STRUCT
 Name1 db I MAGE_SI ZEOF_SHORT_NAME dup(?)
 uni on Misc
   Physi cal Address dd?
   Virtual Size
                     dd?
 ends
 Virtual Address dd?
 Si zeOfRawData
                     dd?
 PointerToRawData dd?
 PointerToRelocations dd?
 PointerToLinenumbers dd?
 NumberOfRelocations dw?
 NumberOfLi nenumbers dw?
 Characteristics dd?
IMAGE_SECTION_HEADER ENDS
IMAGE_SIZEOF_SHORT_NAME equ 8
```

## IMAGE\_IMPORT\_DESCRIPTOR

```
I MAGE_I MPORT_DESCRIPTOR STRUCT
  uni on
                           dd?
    Characteri sti cs
    Ori gi nal Fi rstThunk
                        dd?
  ends
 Ti meDateStamp
                           dd?
  ForwarderChain
                           dd?
  Name1
                           dd?
  FirstThunk
                           dd?
I MAGE I MPORT DESCRIPTOR ENDS
```



#### **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)

LoadLi braryA

GetProcAddress

Exi tProcess

RegEnumKeyA

ChooseFontA

Li neTo

**ImmGetContext** 

Shell ExecuteA

GetDC

PI aySoundA

OpenPri nterA

Original Imports (251)

RegEnumKeyA

RegCl oseKey

RegCreateKeyA

RegSetVal ueExA

RegOpenKeyA

 $[\ldots]$ 

FindNextFileA

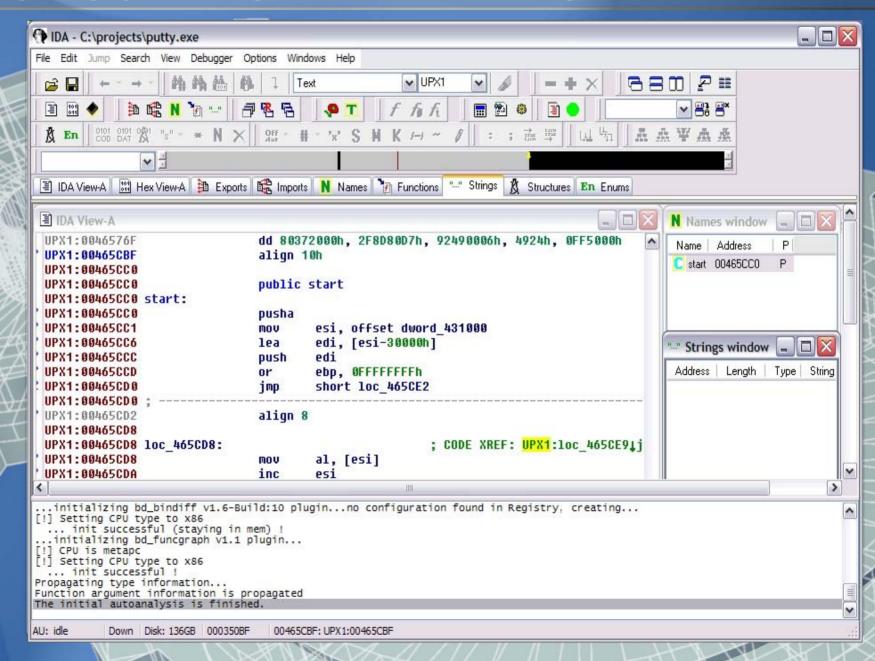
FindClose

Virtual Free

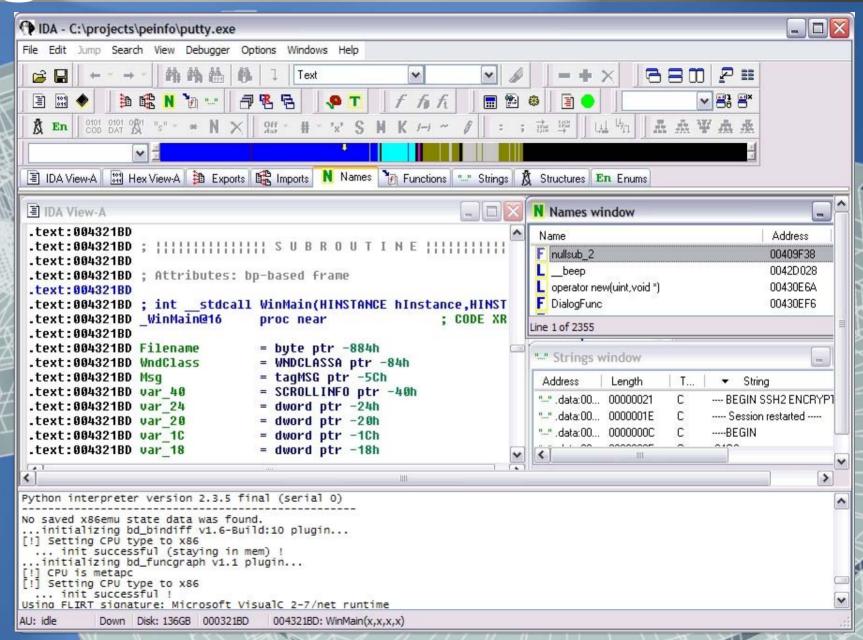
SetFilePointer

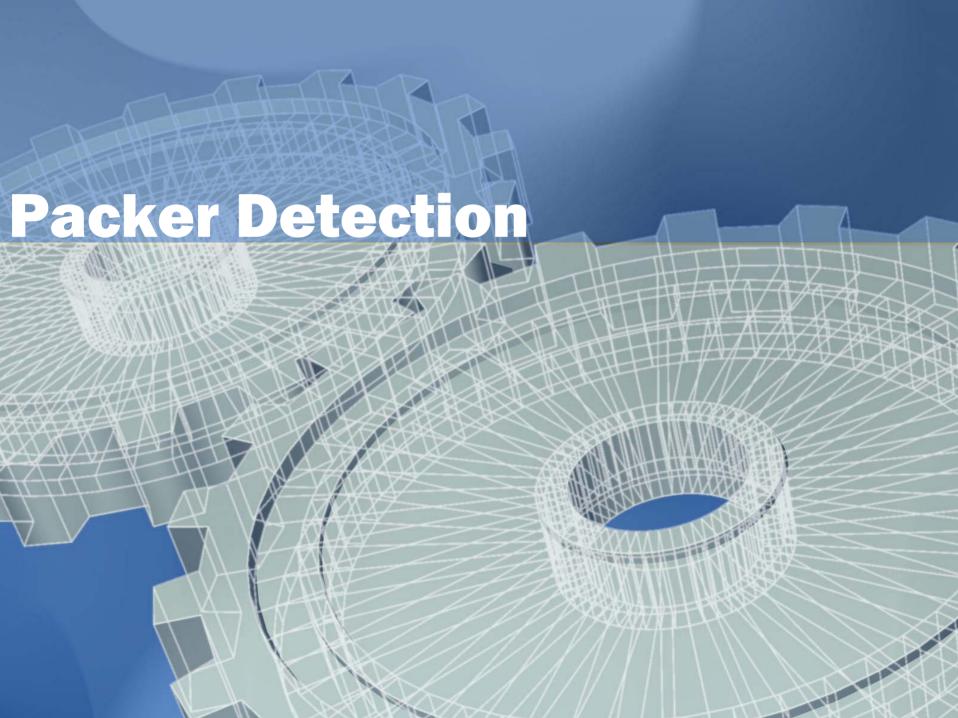
SetStdHandle

#### **Packed File in IDA Pro**



## Original File in IDA Pro





#### **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

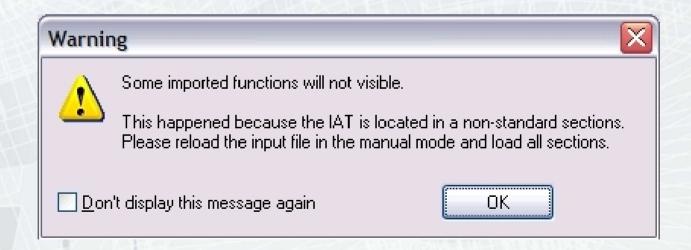
A3 37130300 MOV DWORD PTR DS: [31337], EAX

## **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**

- 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

#### **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 3<sup>rd</sup> party tools
  - Available from http://qmail-scanner.sf.net

#### smdScan

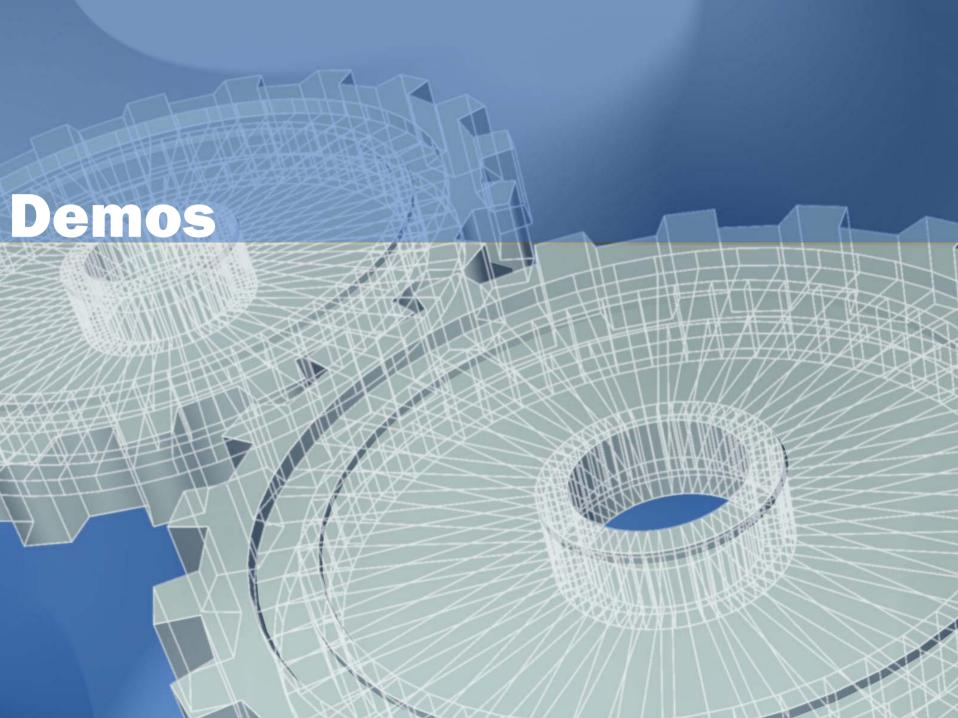
- smdScan is a tool that provides packer detection for Qmail
- Design goals
  - Low CPU utilization
  - Easy to add new signatures
  - Basic policy management
  - Works well with others

# smdScan Usage

#### **SMD Packer Detection DB**

- Scanning over 750 signatures
- Example SMD database entries:

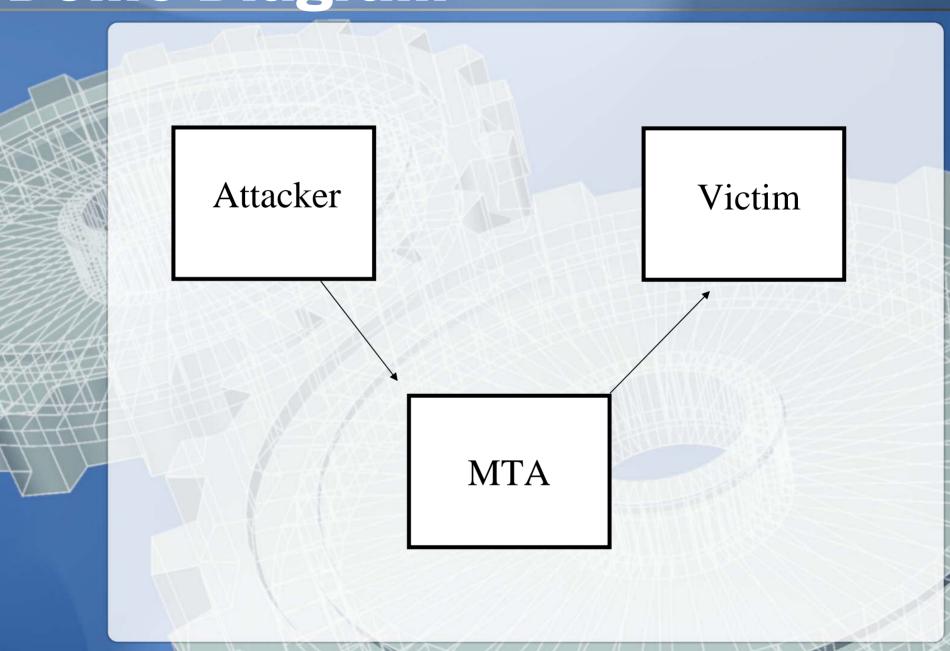
```
[Microsoft Visual C++ v6.0]
signature = 55 8B EC 83 EC 50 53 56 57 BE
  ?? ?? ?? 8D 7D F4 A5 A5 66 A5 8B
ep_only = true
action = allow
[PECompact v1.4x+]
signature = EB 06 68 ?? ?? ?? ?? C3 9C 60
  E8 02 ?? ?? ?? 33 CO 8B C4 83 CO 04 93
  8B E3 8B 5B FC 81
ep_only = true
action = deny
```

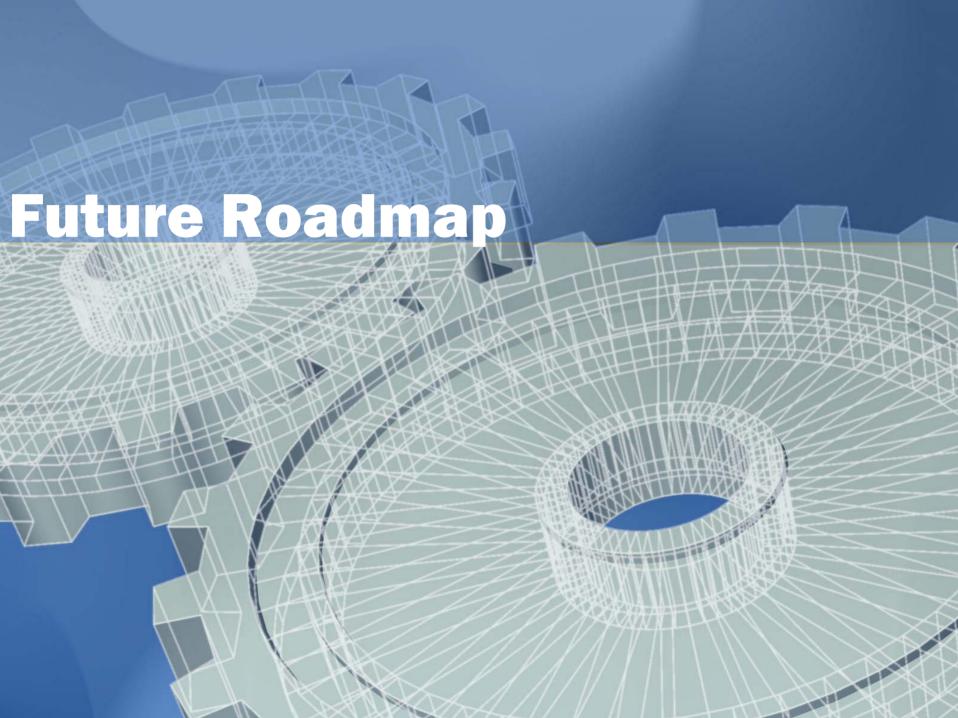


## **Demo Configuration**

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

# **Demo Diagram**





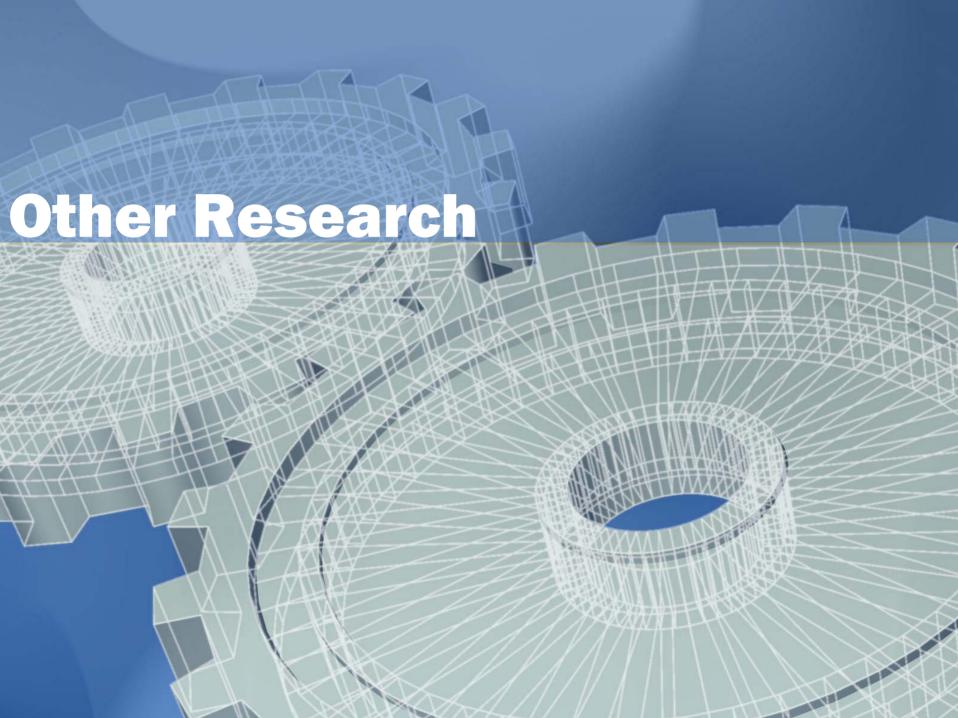
#### **Short Term Goals**

#### • 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



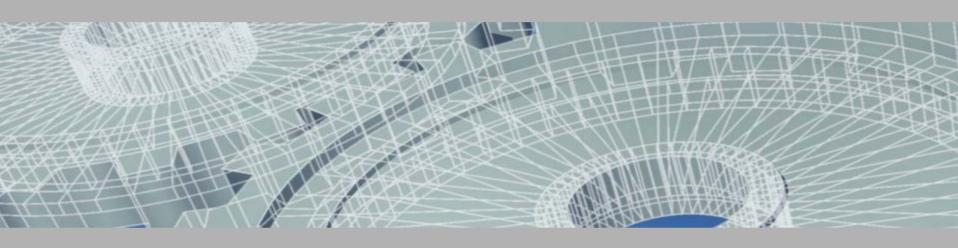
# Other Research

- 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

