The golden age of hacking

Malware analysis

Pen-test methods

http://en.wikipedia.org/wiki/Portal:Computer_security

Malware analysis template

http://www.counterhack.net/malware_template.html

LAB 6.5/6.6

LAB 6.7

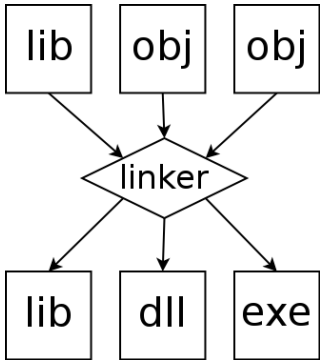
	Activity	Observed Results
Static analysis	Load specimen onto victim machine	
	Run antivirus program	
	Research antivirus results and file names	
	Conduct strings analysis	
	Look for scripts	
	Conduct binary analysis	
	Disassemble code	
	Reverse-compile code	
Dynamic analysis	Monitor file changes	
	Monitor file integrity	
	Monitor process activity	
	Monitor local network activity	
	Scan for open ports remotely	
	Scan for vulnerabilities remotely	
	Sniff network activity	
	Check promiscuous mode locally	
	Check promiscuous mode remotely	
	Monitor registry activity	
	Run code with debugger	

Forensic Analysis of an unknown file

- Before you begin check if you are allowed to examine!
- Question to answer - what are the true functions and capabilities of the file/program?
- Deep knowledge about the program may give additional benefits as
  - Anti-... methods
  - Damage control know how
  - Info about the creator
- Executable file formats
  - http://en.wikipedia.org/wiki/Category:Executable_file_formats
  - ELF, PE, COFF (.exe, executable rights)
  - Object code (.o)
  - Shared libraries (.dll, .so)

Executable file formats

- Symbols
  - Defined symbols, which allow it to be called by other modules
  - Undefined symbols, which call the other modules where these symbols are defined
  - Local symbols, used internally within the object file to facilitate relocation
- Linker
  - Linking of libs and obj files resolving symbols
  - Arranging objects in programs adress space
  - Relocation of code
- What is relocation?
  - Combine all the objects sections like .code (.text), .data, .bss, etc. to a single executable
  - Replacing symbolic references or names of libraries with actual usable (runnable) addresses in memory

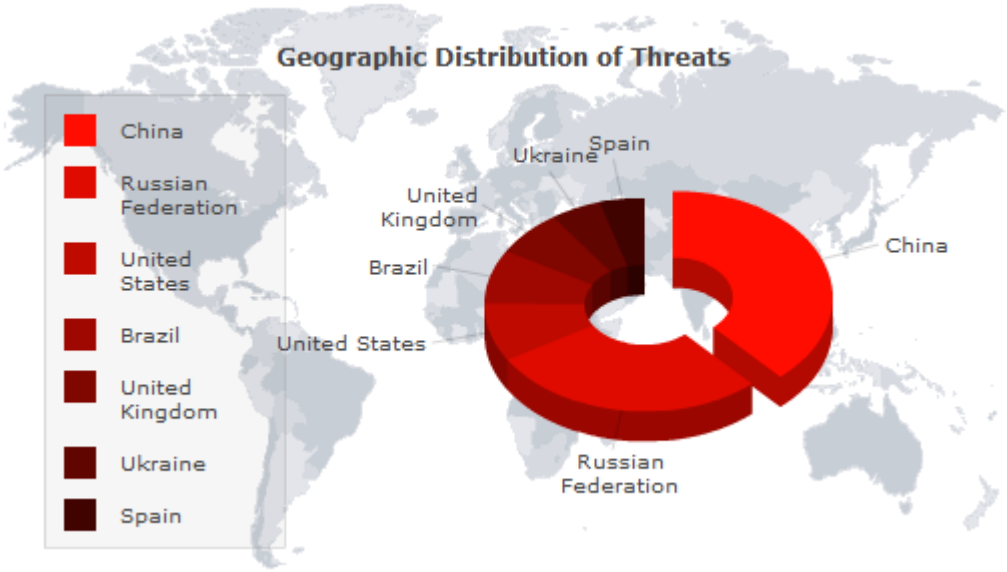


An applications different versions

- Source code
- Debug binary
  - Contains debug info
- Regular binary
  - Dynamic linked libraries
- Regular binary
  - Static linked libraries
- Stripped binary
  - Symbols are removed

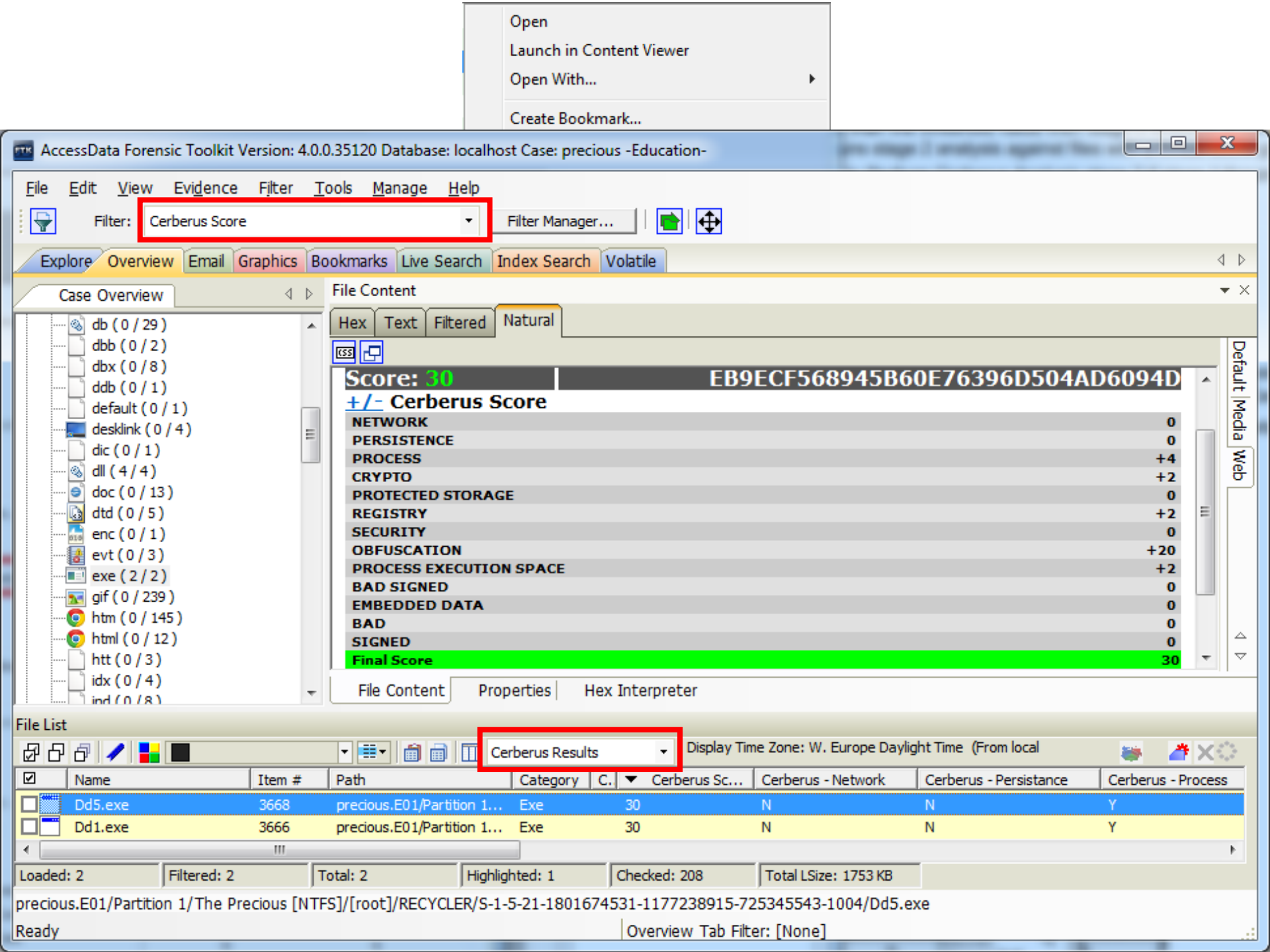
Going from source code to a binary executable

From: http://threatexpert.com 2009-05-14



Automatic malware analysis

- Scan malware with different AntiVirus agents
  - If there is an alert, research AV manufacturers websites
  - If analysis is already done – 90% of your job may be done ☺
    - AV report can be faulty, malcode may be of a new variant etc.
- Web based static and dynamic analyze
  - http://www.virustotal.com
  - http://www.sunbeltsecurity.com - ThreatTrack Security
  - http://metascan-online.com/
- Indicators of Compromise (IOCs)
  - Mandiant IOC Editor and Finder
  - iDefense MAP (Malcode Analyst Pac)
  - FTK – Cerberus
- Many other various solutions – Search! Landscape is changing constantly
- ethical-hacker.net > Blog (Tools and Techniques)
  - http://ethicalhackernet.blogspot.com/2008_04_01_archive.html


Open
Launch in Content Viewer
Open With...
Create Bookmark...
AccessData Forensic Toolkit Version: 4.0.0.35120 Database: localhost Case: precious -Education-
File Edit View Evidence Filter Tools Manage Help
Filter: Cerberus Score
Filter Manager...
Explore Overview Email Graphics Bookmarks Live Search Index Search Volatile
Case Overview
db (0 / 29)
dbb (0 / 2)
dbx (0 / 8)
ddb (0 / 1)
default (0 / 1)
desklink (0 / 4)
dic (0 / 1)
dll (4 / 4)
doc (0 / 13)
dtd (0 / 5)
enc (0 / 1)
evt (0 / 3)
exe (2 / 2)
gif (0 / 239)
htm (0 / 145)
html (0 / 12)
htt (0 / 3)
idx (0 / 4)
File Content
Hex Text Filtered Natural
Score: 30
EB9ECF568945B60E76396D504AD6094D
+/- Cerberus Score
NETWORK 0
PERSISTENCE 0
PROCESS +4
CRYPTO +2
PROTECTED STORAGE 0
REGISTRY +2
SECURITY 0
OBFUSCATION +20
PROCESS EXECUTION SPACE +2
BAD SIGNED 0
EMBEDDED DATA 0
BAD 0
SIGNED 0
Final Score 30
Default Media Web
File Content Properties Hex Interpreter
File List
Cerberus Results
Display Time Zone: W. Europe Daylight Time (From local
Name Item # Path Category C. Cerberus Sc... Cerberus - Network Cerberus - Persistance Cerberus - Process
Dd5.exe 3668 precious.E01/Partition 1... Exe 30 N N Y
Dd1.exe 3666 precious.E01/Partition 1... Exe 30 N N Y
Loaded: 2 Filtered: 2 Total: 2 Highlighted: 1 Checked: 208 Total LSize: 1753 KB
precious.E01/Partition 1/The Precious [NTFS]/[root]/RECYCLER/S-1-5-21-1801674531-1177238915-725345543-1004/Dd5.exe
Ready
Overview Tab Filter: [None]

Cerberus Stage 1 Score

Attribute	Threat Score	Description
Network	+1	Imports networking functions.
Persistence	+4	Indicates signs of persistent behavior. For example, the ability to keep a binary running across computer restarts.
Process	+4	Imports functions to programmatically interact with processes. For example, reading or writing into a process's memory, or injecting code into another process.
Crypto	+2	Imports Microsoft Cryptographic Libraries. For example, the ability to encrypt and decrypt data.
Protected Storage	+5	Imports functions used to access protected storage. For example, Internet Explorer stores a database for form-filling in protected storage.
Registry	+2	Imports functions used to access or change values in the registry.
Security	+4	Imports functions used to modify user tokens. For example, attempting to clone a security token to impersonate another logged on user.
Obfuscation	+20	Contains a packer signature, contains sections of high entropy, or imports a low number of functions.
Process Execution Space	+2	Unusual activity in the Process Execution Space header. For example, a zero length raw section, unrealistic linker time, or the file size doesn't match the Process Execution Space header.
Bad Signed	+20	Contains a signature but the signature is bad.
Embedded Data	+5	Contains an embedded executable code.
Bad / Bit-Bad	+20	Contains an IRC or shellcode signature.
Signed / Bit Signed	-20	Contains a valid signature.

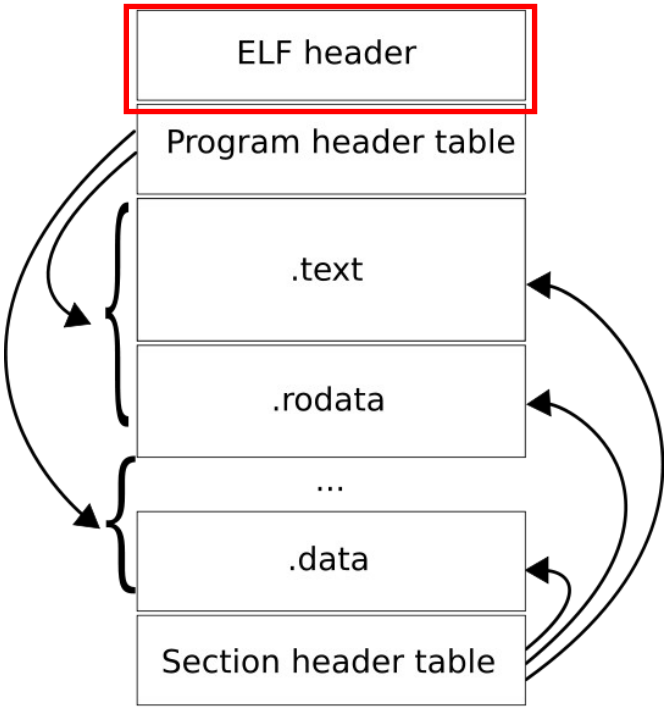
ELF (Executable and Linking Format)

- ELF header

Tells us basic info and where everything is located in the file

Can be read directly from the first e_ehsize (default: 52) bytes of the file

Fields of interest: e_entry, e_phoff, e_shoff, and the sizes given. e_entry specifies the location of _start, e_phoff shows us where the array of program headers lies in relation to the start of the executable, and e_shoff shows us the same for the section headers




```
/* ELF File Header */
typedef struct
{
  unsigned char e_ident[EI_NIDENT];     /* Magic number and other info */
  Elf32_Half    e_type;                 /* Object file type */
  Elf32_Half    e_machine;              /* Architecture */
  Elf32_Word    e_version;              /* Object file version */
  Elf32_Addr    e_entry;                /* Entry point virtual address */
  Elf32_Off     e_phoff;                /* Program header table file offset */
  Elf32_Off     e_shoff;                /* Section header table file offset */
  Elf32_Word    e_flags;                /* Processor-specific flags */
  Elf32_Half    e_ehsize;               /* ELF header size in bytes */
  Elf32_Half    e_phentsize;            /* Program header table entry size */
  Elf32_Half    e_phnum;                /* Program header table entry count */
  Elf32_Half    e_shentsize;            /* Section header table entry size */
  Elf32_Half    e_shnum;                /* Section header table entry count */
  Elf32_Half    e_shstrndx;             /* Section header string table index */
} Elf32_Ehdr;
```

http://en.wikipedia.org/wiki/Executable_and_Linkable_Format

ELF (Executable and Linking Format)

- ELF Program segment headers
  - Describe the **segments** of the program used at run-time
  - In a typical ELF executable usually end-to-end, forming an array of structs
  - The interesting fields in this structure are p_offset, p_filesz, and p_memsz
- ELF Section headers
  - Describe various named **sections** of the binary as a file
  - Each section has an entry in the section headers array
- HT Editor (http://hte.sourceforge.net/)
  - Examine and modify everything in an ELF file (PE files also), disassemble etc.

```
/* Program segment header  */
typedef struct
{
  Elf32_Word    p_type;         /* Segment type */
  Elf32_Off     p_offset;       /* Segment file offset */
  Elf32_Addr    p_vaddr;        /* Segment virtual address */
  Elf32_Addr    p_paddr;        /* Segment physical address */
  Elf32_Word    p_filesz;       /* Segment size in file */
  Elf32_Word    p_memsz;        /* Segment size in memory */
  Elf32_Word    p_flags;        /* Segment flags */
  Elf32_Word    p_align;        /* Segment alignment */
} Elf32_Phdr;
```

```
/* Section header  */
typedef struct
{
  Elf32_Word    sh_name;        /* Section name (string tbl index) */
  Elf32_Word    sh_type;        /* Section type */
  Elf32_Word    sh_flags;       /* Section flags */
  Elf32_Addr    sh_addr;        /* Section virtual addr at execution */
  Elf32_Off     sh_offset;      /* Section file offset */
  Elf32_Word    sh_size;        /* Section size in bytes */
  Elf32_Word    sh_link;        /* Link to another section */
  Elf32_Word    sh_info;        /* Additional section information */
  Elf32_Word    sh_addralign;   /* Section alignment */
  Elf32_Word    sh_entsize;     /* Entry size if section holds table */
} Elf32_Shdr;
```

ELF Object File Format

Sections in object code is linked into the executable

One or more sections maps to a segment in the executable

Some of the sections (from elf.pdf)

.bss This section holds uninitialized data that contribute to the program's memory image. By definition, the system initializes the data with zeros when the program begins to run.

.comment This section holds version control information.

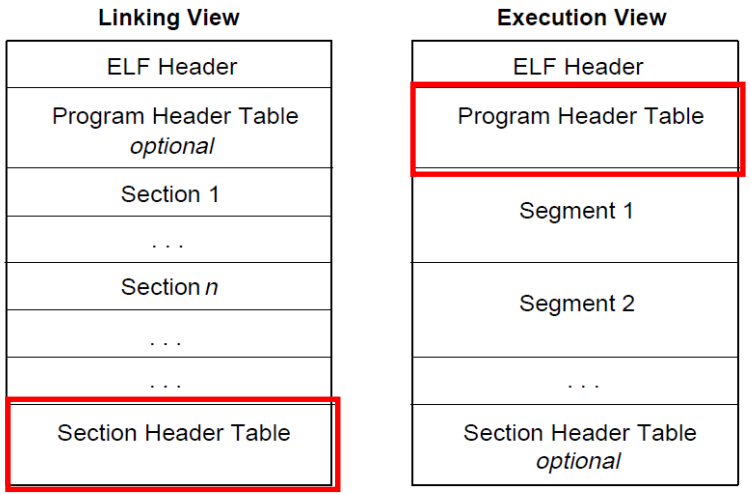
.data and .data1 These sections hold initialized data that contribute to the program's memory image.

.debug This section holds information for symbolic debugging. The contents are unspecified. All section names with the prefix .debug are reserved for future use.

.dynamic This section holds dynamic linking information

.hash This section holds a symbol hash table.

.line This section holds line number information for symbolic debugging, which describes the correspondence between the source program and the machine code. The contents are unspecified.



.rodata These sections hold read-only data that typically contribute to a .rodata1 non-writable segment in the process image.

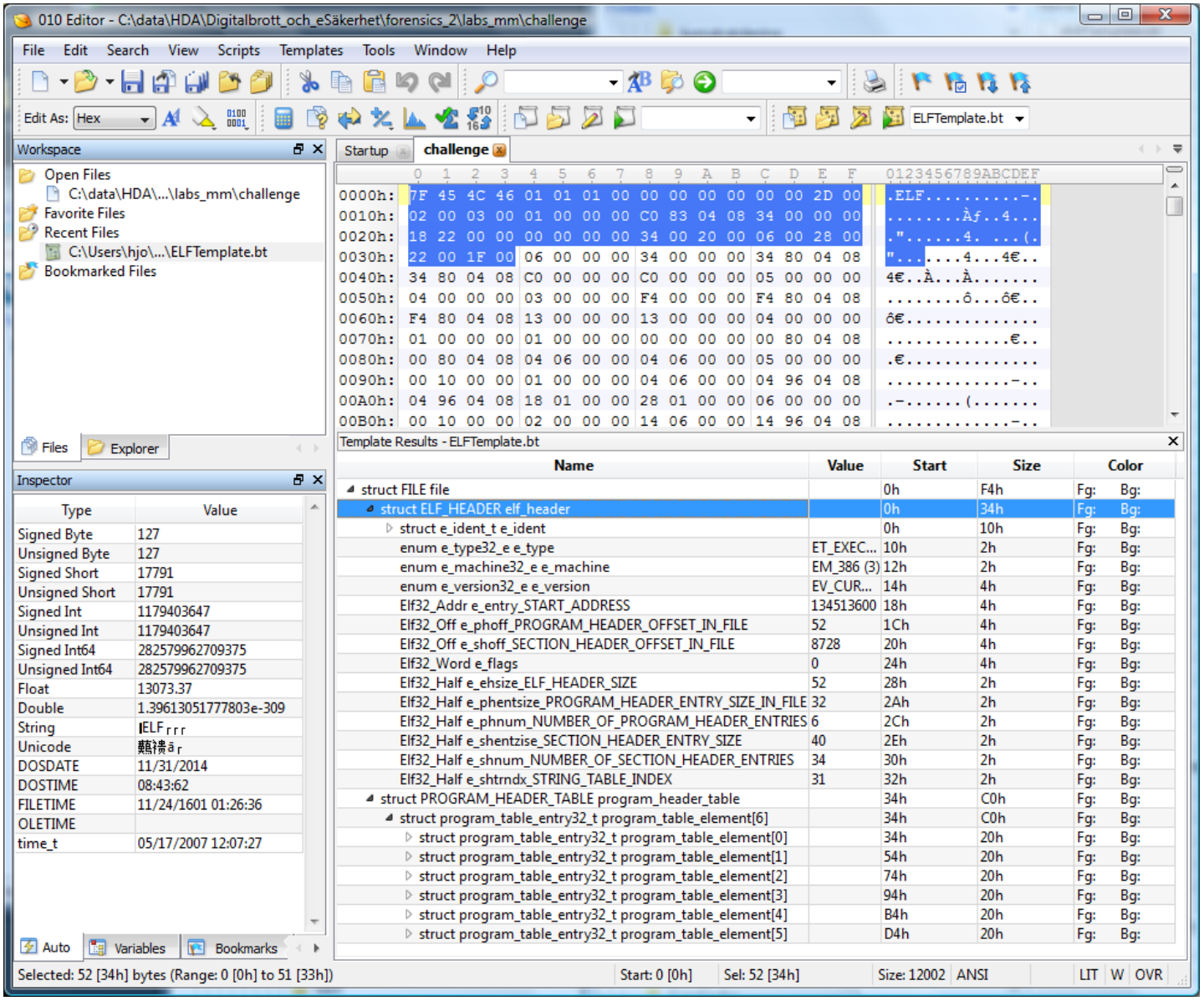
.shstrtab This section holds section names.

.strtab This section holds strings, most commonly the strings that represent the names associated with symbol table entries.

.symtab This section holds a symbol table, as "Symbol Table"

.text This section holds the "text," or executable instructions, of a program.

Sweetscape 010 editor - ELF template



Static analysis methods (Linux)

- Hash the file
- File
  - Properties and type of file etc.
- Strings
- Hexdump
- Nm
  - List symbol info
- Ldd
  - View shared objects which is linked in at runtime
  - Listed in the .interp section
- Readelf, elfdump, objdump

hjo@lnx:~/$ file winkill
winkill: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.0.0, dynamically linked (uses shared libs), for GNU/Linux 2.0.0, not stripped

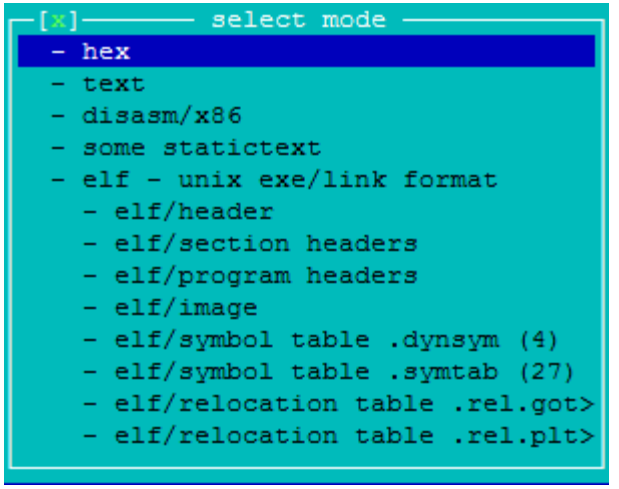
hjo@lnx:~/$ nm winkill
...
08048784 T parse_args
08049c78 D port
U printf@@GLIBC_2.0
08048760 T usage
U usleep@@GLIBC_2.0
…
D The symbol is in the initialized .data section
T The symbol is in the .text (code) section
U The symbol is unknown
…

hjo@lnx:~/$ ldd winkill
linux-gate.so.1 => (0xffffe000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7e36000)
/lib/ld-linux.so.2 (0xb7f70000)

Readelf

```
hjo@lnx:~/$ readelf
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all              Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header      Display the ELF file header
  -l --program-headers  Display the program headers
     --segments         An alias for --program-headers
  -S --section-headers  Display the sections' header
     --sections         An alias for --section-headers
  -g --section-groups   Display the section groups
  -t --section-details  Display the section details
  -e --headers          Equivalent to: -h -l -S
  -s --syms             Display the symbol table
     --symbols          An alias for --syms
  -n --notes            Display the core notes (if present)
  -r --relocs           Display the relocations (if present)
  -u --unwind           Display the unwind info (if present)
  -d --dynamic          Display the dynamic section (if present)
  -V --version-info     Display the version sections (if present)
  -A --arch-specific    Display architecture specific information (if any).
  -D --use-dynamic      Use the dynamic section info when displaying symbols
  -x --hex-dump=<number> Dump the contents of section <number>
  -w[liaprmfFsoR] or
  --debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=str,=loc,=Ranges]
                        Display the contents of DWARF2 debug sections
  -I --histogram        Display histogram of bucket list lengths
  -W --wide             Allow output width to exceed 80 characters
  @<file>               Read options from <file>
  -H --help             Display this information
  -v --version          Display the version number of readelf
Report bugs to <URL:http://www.sourceware.org/bugzilla/>
```

Objdump and HT Editor



HT Editor - http://hte.sourceforge.net/

- Provides readelf functions and further probing of contents

- Disassemble
  - Convert from binary to assembly code
    - Dead listing
  - hjo@lnx:~/$ objdump -d winkill

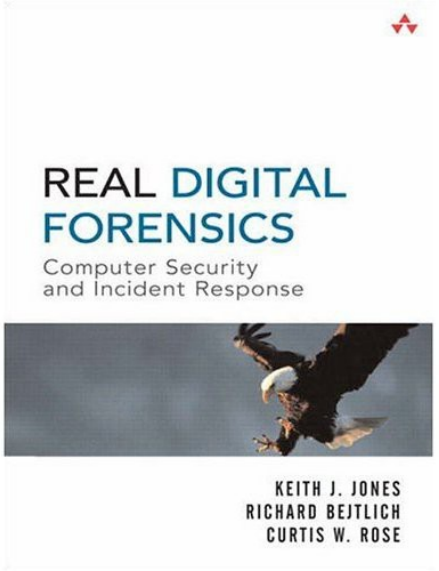
```
-d, --disassemble       Display assembler contents of executable sections
-D, --disassemble-all   Display assembler contents of all sections
```

```
08048874 <main>:
 8048874:      55                   push   %ebp
 8048875:      89 e5                mov    %esp,%ebp
 8048877:      81 ec b8 3a 00 00    sub    $0x3ab8,%esp
 804887d:      c7 45 e8 98 3a 00 00    movl
$0x3a98,0xffffffe8(%ebp)
 8048884:      83 7d 08 01          cmpl   $0x1,0x8(%ebp)
 8048888:      7f 0e                jg     8048898 <main+0x24>
 804888a:      8b 45 0c             mov    0xc(%ebp),%eax
 804888d:      8b 10                mov    (%eax),%edx
 804888f:      52                   push   %edx
 8048890:      e8 cb fe ff ff       call   8048760 <usage>
 8048895:      83 c4 04             add    $0x4,%esp
...
```

```
08048760 <usage>:
 8048760:      55                   push   %ebp
 8048761:      89 e5                mov    %esp,%ebp
 8048763:      8b 45 08             mov    0x8(%ebp),%eax
 8048766:      50                   push   %eax
 8048767:      68 80 8b 04 08       push   $0x8048b80
 804876c:      e8 97 fe ff ff       call   8048608
<printf@plt>
...
```

This is an excerpt from the output!

Further analysis!



- RDF chapters 13, 14 and 15 are elite!
- Ch 14 deals with
  - Advanced static analysis options
  - Advanced dynamic analysis options
  - Unlink an unpacked tmp file
    - Open and execve the deleted tmp file
  - Generate core file (process dump)
    - ulimit -c unlimited (to enable core)
    - kill -s SIGSEV <PID> (from another console), other signals which action is core should do aswell, SIGSEV = Invalid memory reference
    - Check out the Linux manual: man signal
  - Examine core files with gdb
  - Packers
  - RCE etc. ...

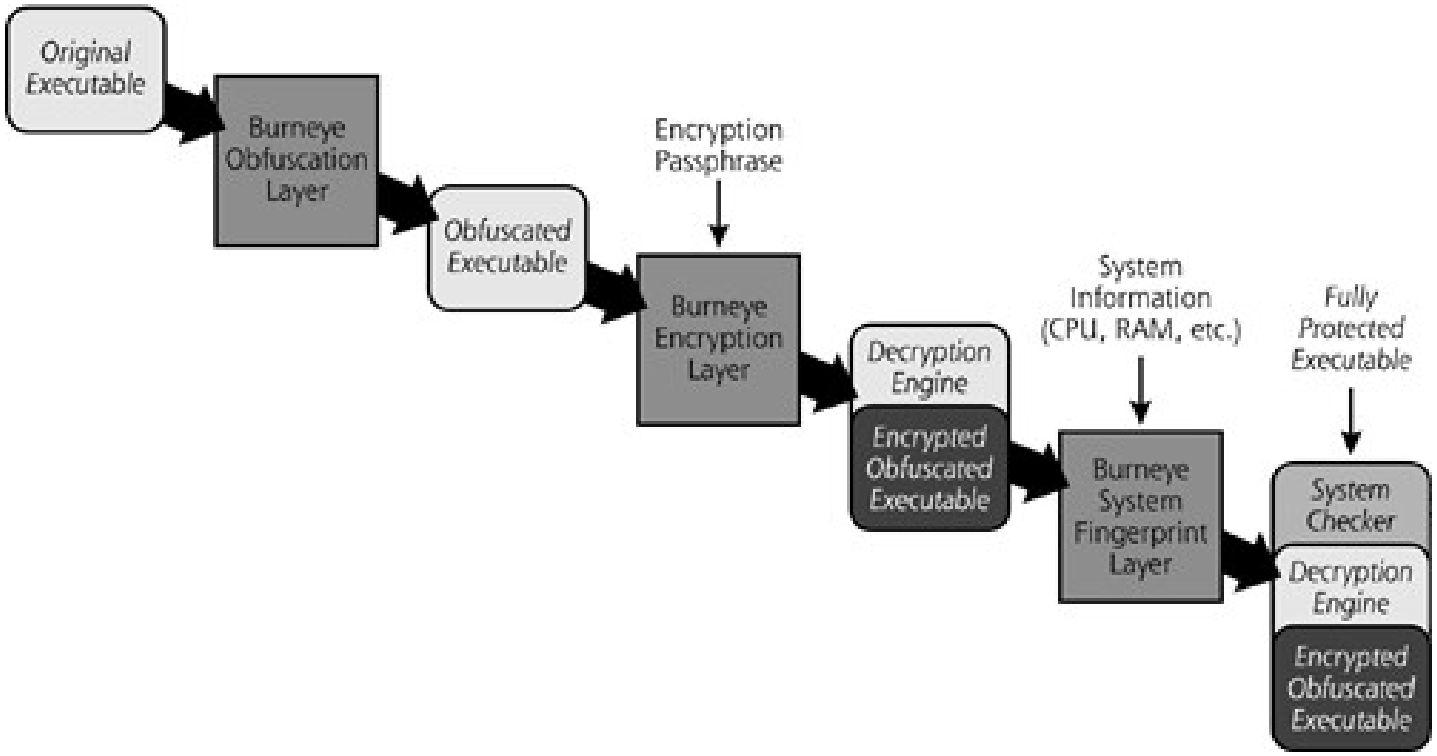
Further analysis...

- Different methods to recover a unpackable packed binary...
  - Debugfs
    - ext2/ext3 file system debugger
    - Similar to ifind and icat as in SITIC course exercise but on a deleted file
  - Strace hexdump – output all
    - In combination with hexeditor (cut and paste) rebuild binary
  - /proc pseudo file system
    - ls -al /proc/<PID>/
    - # man proc
    - Copy the exe link
  - Packers as UPX(nrv/ucl)
    - First try to unpack with packer versions
    - Note that programmer may have ”edited” away traces of used packers with a hexeditor
  - Crypt-packers as Burneye

```
hjo@lnx:~/$ ls -al /proc/29279/
dr-xr-xr-x  5 hjo  hjo  0 Feb  6 12:56 .
dr-xr-xr-x 82 root root 0 Nov  7 11:49 ..
-r--------  1 root root 0 Feb  6 12:57 auxv
--w-------  1 root root 0 Feb  6 12:57 clear_refs
-r--r--r--  1 root root 0 Feb  6 12:56 cmdline
-rw-r--r--  1 root root 0 Feb  6 12:57 coredump_filter
lrwxrwxrwx  1 root root 0 Feb  6 12:57 cwd -> /
-r--------  1 root root 0 Feb  6 12:57 environ
lrwxrwxrwx  1 root root 0 Feb  6 12:57 exe ->
/tmp/upxRandName (deleted)
dr-x------  2 root root 0 Feb  6 12:57 fd
dr-x------  2 root root 0 Feb  6 12:57 fdinfo
-r--------  1 root root 0 Feb  6 12:57 limits
-r--r--r--  1 root root 0 Feb  6 12:57 maps
-rw-------  1 root root 0 Feb  6 12:57 mem
-r--r--r--  1 root root 0 Feb  6 12:57 mounts
-r--------  1 root root 0 Feb  6 12:57 mountstats
-rw-r--r--  1 root root 0 Feb  6 12:57 oom_adj
-r--r--r--  1 root root 0 Feb  6 12:57 oom_score
lrwxrwxrwx  1 root root 0 Feb  6 12:57 root -> /
-r--r--r--  1 root root 0 Feb  6 12:57 smaps
-r--r--r--  1 root root 0 Feb  6 12:56 stat
-r--r--r--  1 root root 0 Feb  6 12:57 statm
-r--r--r--  1 root root 0 Feb  6 12:56 status
dr-xr-xr-x  3 hjo  hjo  0 Feb  6 12:57 task
-r--r--r--  1 root root 0 Feb  6 12:57 wchan
```

Burneye's three layers of executable protection

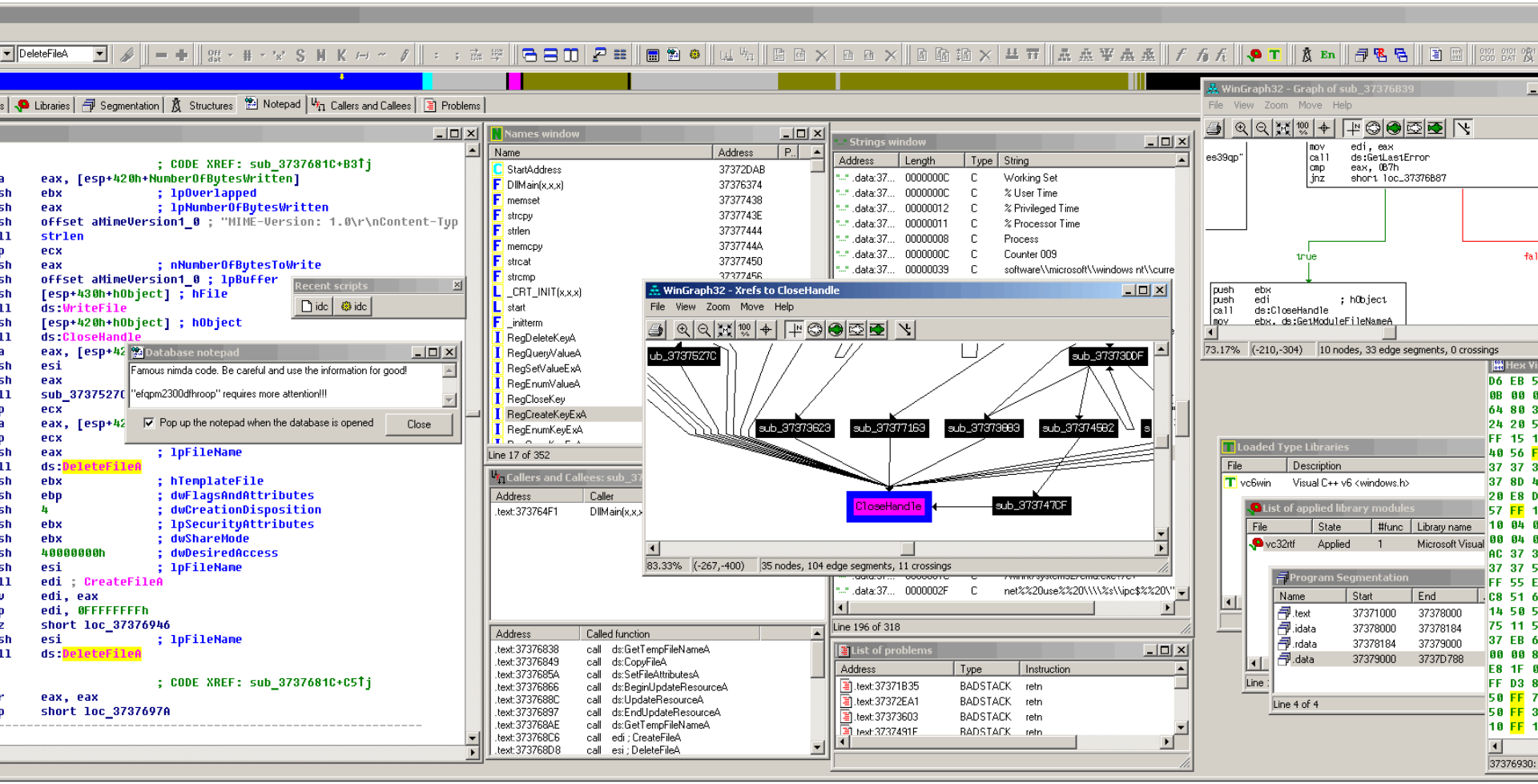
- Scrambles the code in the executable thru **obfuscated** instructions
- **Encryption** of the binary program
- **System fingerprint** – will only run on certain computers



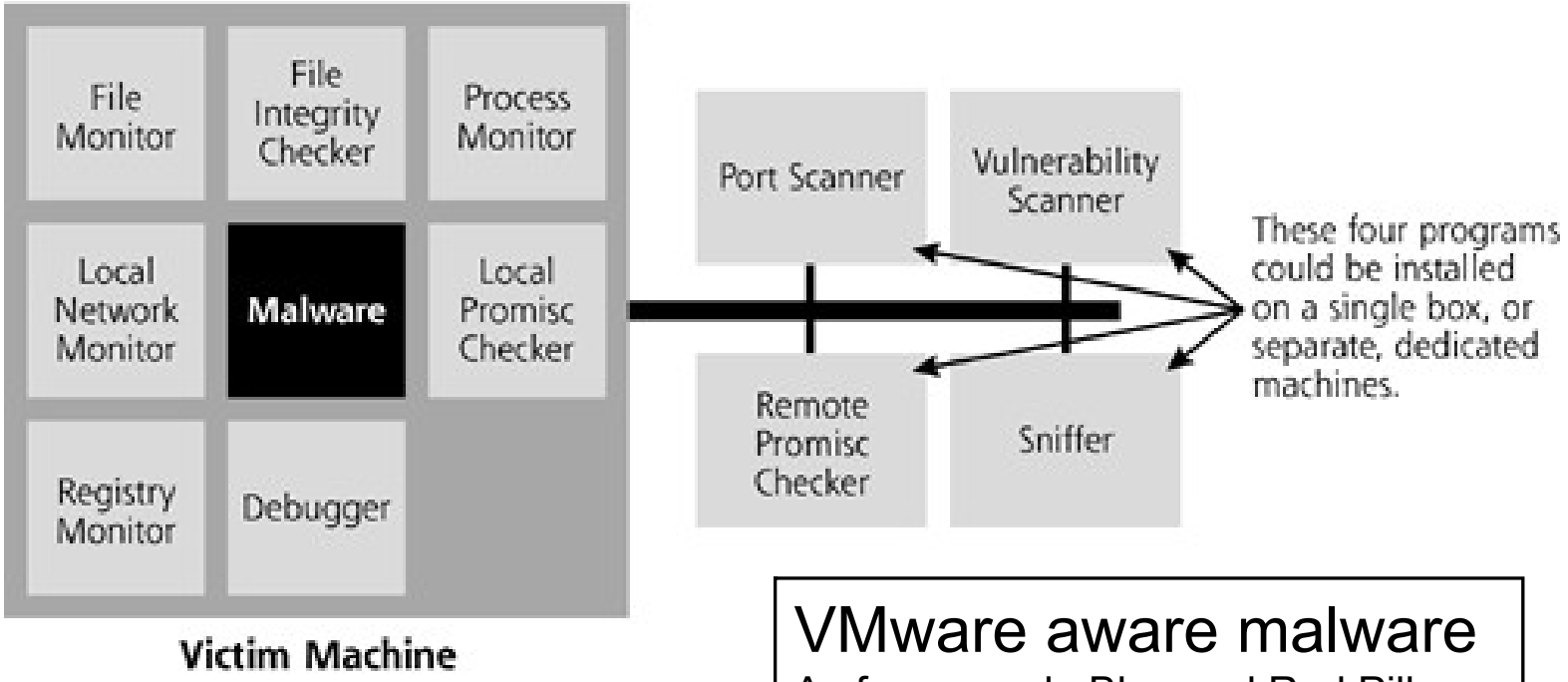
Static and dynamic verfication

- Verify difference/similarity between examined file and assumed source code/binary in ”the wild”
- Compare output
  - With diff or other line by line tool
  - Functions with nm
  - Strings
  - Assembly code side by side
  - Ssdeep, nwdiff, bindiff (binary)
- strace, ltrace
- Gdb/ddd or other tools as IDA Pro, OllyDbg
  - http://www.gnu.org/software/ddd/
  - http://www.hex-rays.com/idapro/
  - http://www.ollydbg.de/
- Practical usage testing and monitoring
  - lsof, netstat, wireshark etc. (live response methods)

IDA Pro


Names window
Strings window
WinGraph32 - Xrefs to CloseHandle
Database notepad
Famous nimda code. Be careful and use the information for good!
"efqpm2300dfhroop" requires more attention!!!
Pop up the notepad when the database is opened
Close
Recent scripts
Callers and Callees
List of problems
Loaded Type Libraries
List of applied library modules
Program Segmentation
CloseHandle

Dynamic analysis



VMware aware malware

As for example Blue and Red Pill

http://www.invisiblethings.org

http://bluepillproject.org

Debugger aware malware

PEB (Process Environment Block) struct got a member variable:

UCHAR BeingDebugged;

Malware check itself if being debugged!

```
int swallow_redpill () {
  unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();
  return (m[5]>0xd0) ? 1 : 0;
}
```

Pen-test methods

Ongoing

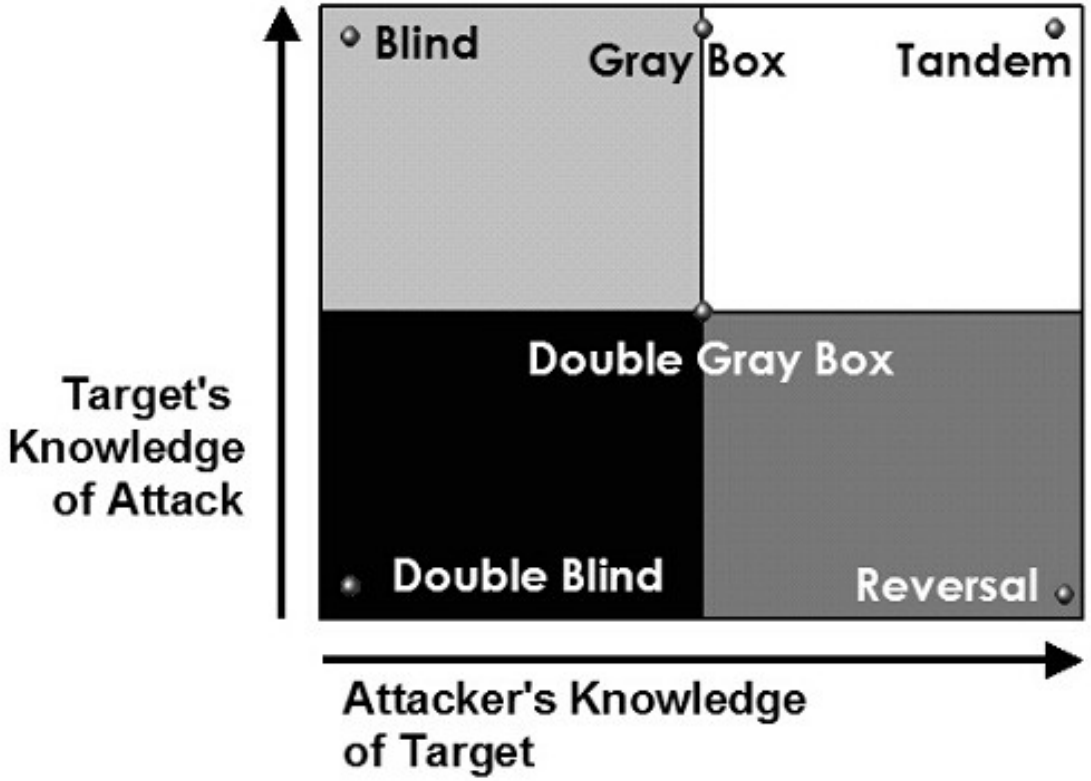
2005/2006
Open Information
Systems Security Group

2003
NIST Special Publication 800-42
Guideline on Network Security Testing

Types of security tests

Security Test Types

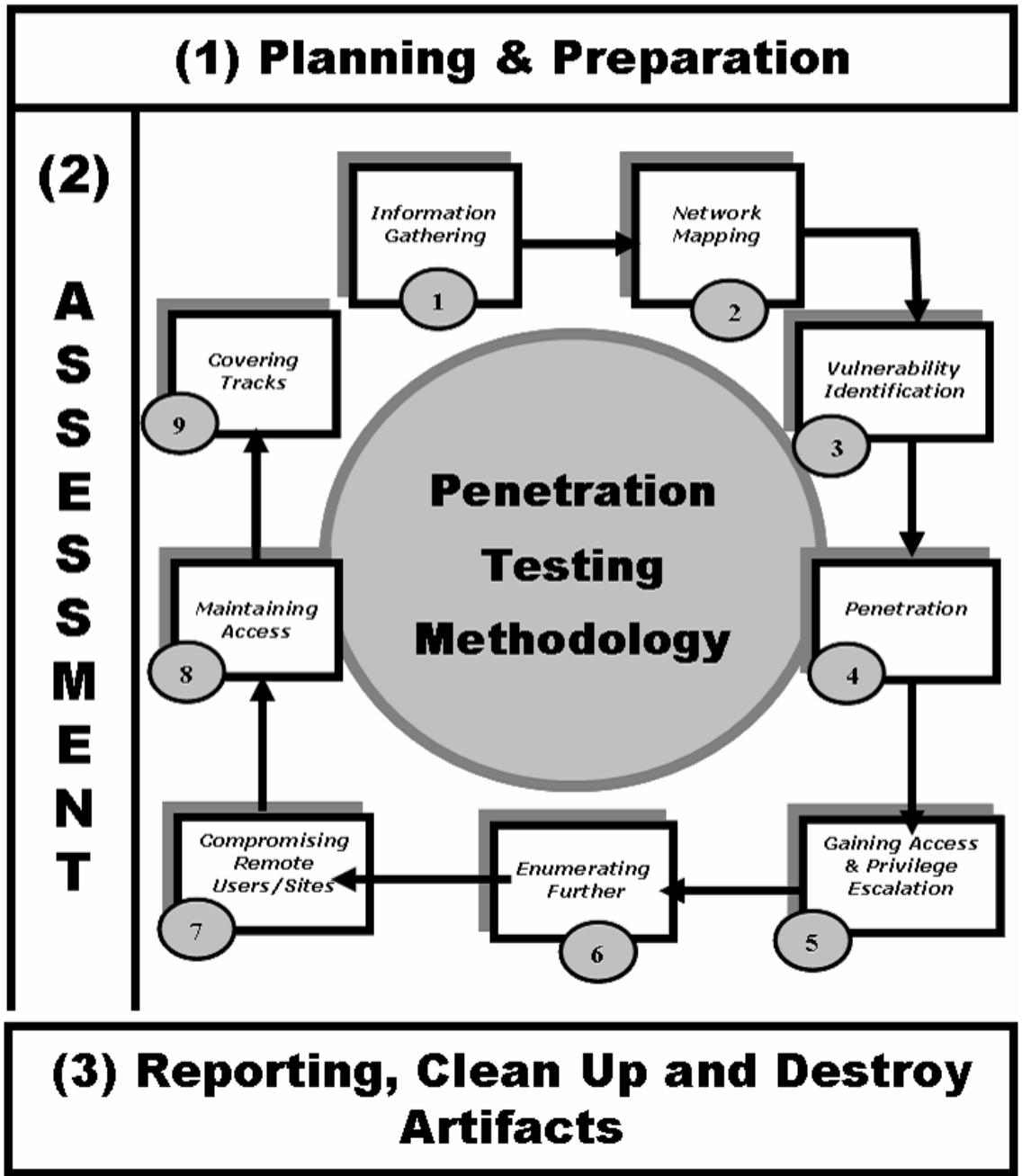
"Security Testing" is an umbrella term to encompass all forms and styles of security tests from the intrusion to the hands-on audit. The application of the methodology from this manual will not deter from the chosen type of testing.



Reversal =
Red Team exercise

Approach & Methodology

ISSAF



OSSTMM 2.2

Sex områden ingår i Osstmm

Open source security testing methodology manual, Osstmm, är en metod för säkerhetsgranskningar. Den presenterades i början av 2001 av Pete Herzog.

Syftet var att ge säkerhetsgranskare en gemensam grund att arbeta från och samtidigt ge kunder möjlighet att veta vad de kan förvänta sig av ett test.

Följande områden behandlas i Osstmm:

1 Informationssäkerhet – Vilken typ av information rörande företaget som finns tillgängligt på internet.

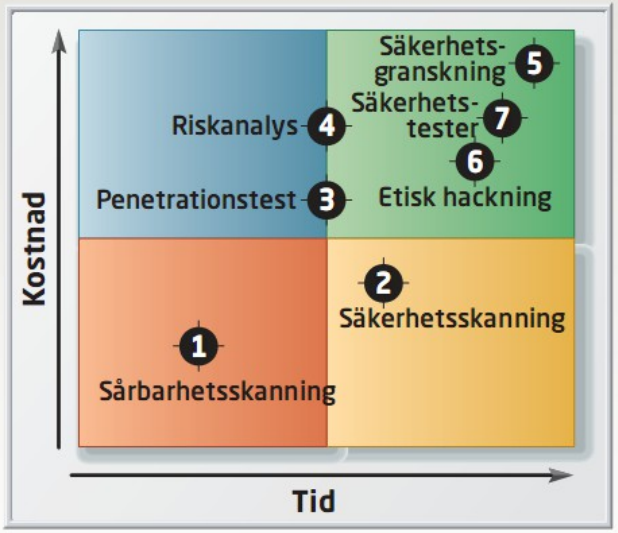
2 Processäkerhet – Här testas hur personalberoende hanteras.

3 Informationsteknik säkerhet – Hur olika typer av nätverksutrustning, program med mera ska testas.

4 Kommunikation – Tester av faxar, modem och pbx:er.

5 Trådlös säkerhet – Trådlösa accesspunkter, handhållna datorer, infrarött, rfid och så vidare.

6 Fysisk säkerhet – Områdes- och byggnadsgranskning. Hur motståndskraftigt är kontoret mot fysiska angrepp.



Kostnad och tid för olika säkerhetsteststekniker enligt Osstmm. Billgast är sårbarhetsskanning (1). Genom att verifiera så kallade "false positives" får vi en "säkerhetsskanning"(2). Penetrationstester (3) är ett målorienterat arbete för att komma åt ett system eller nätverk. Riskanalyser (4) och säkerhetsgranskningar (5) är vanliga sätt att granska säkerheten och har inte så mycket med penetrationstester att göra. Etisk hackning (6) definieras som ett penetrationstest utan måldefinition. Säkerhetstester (7) kan beskrivas som ett fullskaligt penetrationstest.

Support docs for ordering a pen-test (mainly for Swedish organizations)

- Så beställer du det perfekta penetrationstestet och säkerhetstestet
  - Två artiklar - 2008 och 2013 från IDG
- Säkerhetspolisen (SÄPO)
  - Säkerhetsskyddad upphandling – en vägledning - från 2009
  - För myndighet (staten, kommun eller landsting)
  - Vad är säkerhetsskydd?
  - Processen säkerhetsskyddad upphandling
- By searching the Internet there is a lot of guides and documents available which describe this process in detail
- Choosing the right vulnerability scanner for your organization (report)
  - The Magazine Information Week