

Computer and Network Security via Program Analysis (Malware Analysis)

Monirul Sharif

Guest Lecture

CS-4803 (CNS)

College of Computing

Georgia Institute of Technology

Date: 09/22/2006

Introduction

- Program analysis techniques are used for malware detection and analysis
- It is evident that simple signature based schemes have become increasingly ineffective
- Both static and dynamic analysis are used for advanced signature and behavior based detection.
- We will cover techniques of analysis and also highlight techniques that malware use to make analysis hard

Malware

- Software designed to infiltrate or damage computer systems – **“Malicious Software”**
 - Computer viruses, worms, trojan horses, spyware, adware etc.
 - Created by someone with malicious intent – compromise security
- Malware categories
 - Computer Viruses – spreads by infecting legitimate programs
 - Computer Worms – spreads by the network from host to host
 - Trojans – malware disguised as legitimate software
 - Botnets – malware that provides command and control capability
 - Spyware – spies on user activities and compromises privacy
 - Adware – unsolicited ads on infected computers
 - And there are more

Malware Trend

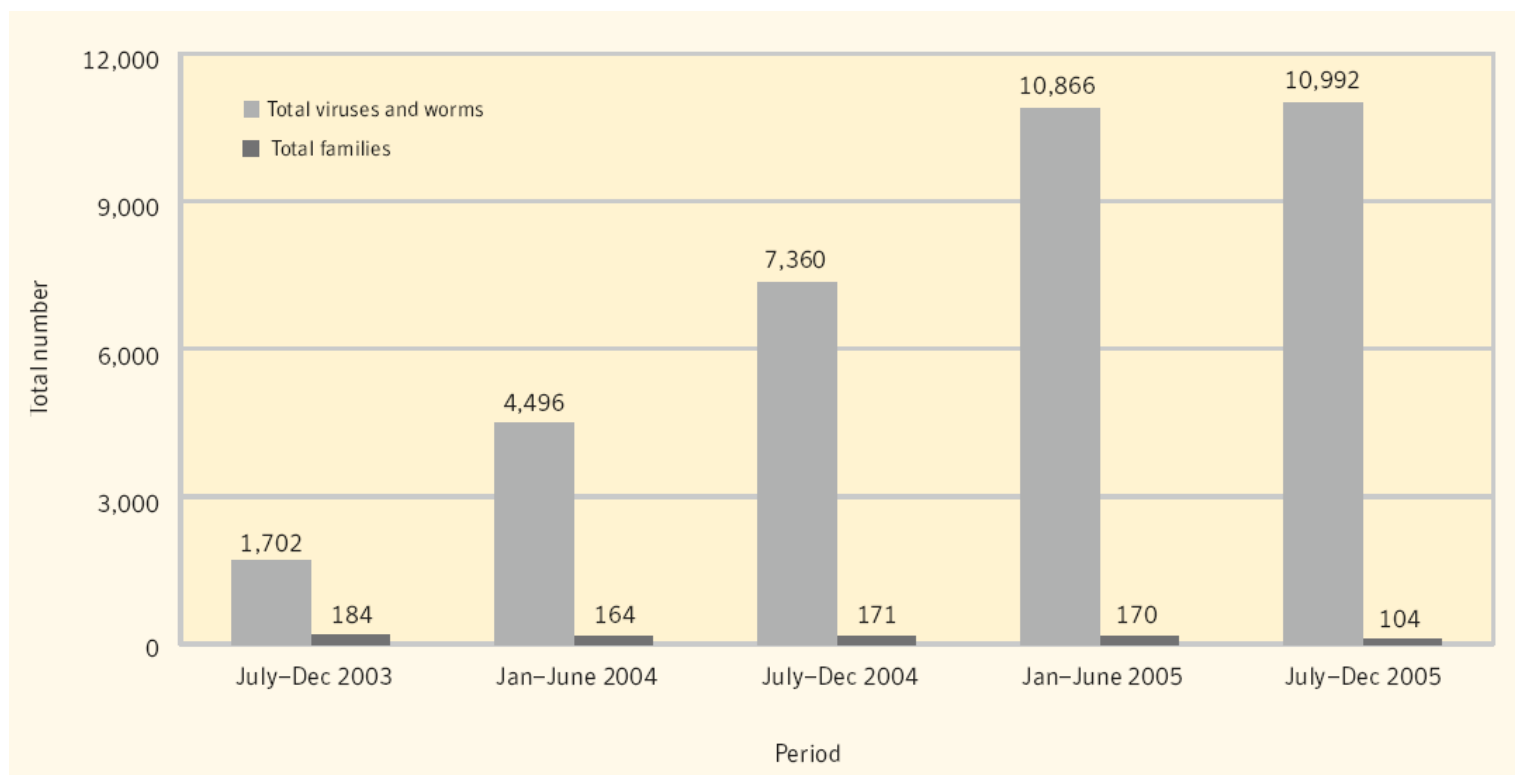


Fig: New Win32 virus and worm variants

Source: Symantec Internet and Security and Threat Report Vol IX.

Traditional Detection Methods

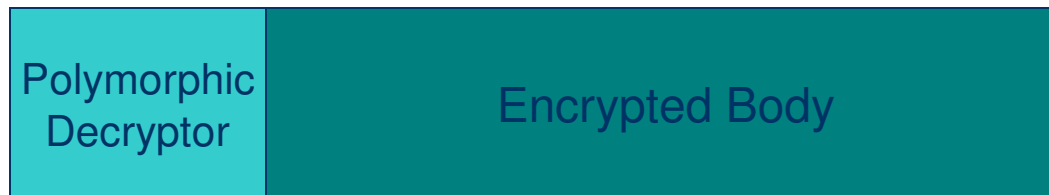
- Antivirus software has been around since the dawn of computer viruses
- Traditional methods of detection - signature based
 - Usually the signature is sequence of bytes found in malware
 - Antivirus software scans files using signature. Can be considered simple “Static Analysis”
 - In traditional methods signature is mostly “syntactic”
- How to find signature
 - Traditional method involves manual analysis of malware
 - Binary is scanned structurally and reverse engineered
 - Malware is analyzed to find an invariant to be used as a signature

Traditional Evasion Methods

- Methods seen in viruses a long time ago
 - Stealth viruses – Hides changes in the system and therefore the virus by showing original state
 - Oligomorphic viruses – Encrypts body with varying forms and has a constant decryptor routine
 - Polymorphic viruses – Contains varying encrypted body and has several copies of decryptor (polymorphic decryptor)
 - Metamorphic viruses – Uses various code morphic techniques to create new instances that are different in code but identical in nature.
- Current generation of malware use “Obfuscation” techniques evolved from the above principles to evade signature based schemes.
- Code obfuscation is used to evade “static analysis”
- A good resource of information - Peter Szor’s book – “The Art of Computer Virus Research and Defense”.

Polymorphic Code

- When creating a new instance, the malware uses code encryption to encrypt its body
 - Different keys are used in different instances so body is different sequence of bytes
 - Simple signatures that are just sequences of bytes in the body are fooled
- Polymorphic Decryptor
 - The malware uses several versions of decryptors or uses *obfuscation* so that it is not easy to create signature



Detection of Polymorphic Code

- Commercial Antivirus Software are capable of detecting polymorphic malware (viruses and trojans)
 - Signature is created to detect PD (if PD is not obfuscated well enough)
 - Emulation is used to antivirus, to emulate PD action and find a detectable sequence in the body
 - Malware also employ anti-emulation techniques to defeat them
- Polymorphic worms (see Polygraph by Newsome et. al. and also defeating Polygraph by Perdisci et. al.)

Metamorphic Code

- To further evade detection, complete full body obfuscation is applied
- Some obfuscation techniques are
 - Code reordering
 - Garbage insertion
 - Equivalent code replacement
 - Jump insertions (Code reordering via inserting jumps)
 - Code and data encapsulation (Packing code)
- Metamorphic malware creates new instances using these
 - Some “Disassemble own code” and apply directly at machine code level
 - Some convert in to intermediate form, apply them and recompile
- Reference – Szor’s book and Christodorescu et. al. “Testing Malware Detectors”

Metamorphic Code - Examples

- Code reordering

- Instructions that are independent are re-ordered

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
MOV [X], EAX
```

```
MOV EBX, [Y]
MOV EAX, [X]
ADD EAX, EBX
MOV [X], EAX
```

- Garbage insertion

- Instructions are inserted that are semantic no-ops (do not effect the code and registers, and therefore execution)

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
MOV [X], EAX
```

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
PUSH ESI
MOV [X], EAX
POP ESI
```

Metamorphic Code - Examples

- Equivalent Code Replacement

- Register renaming, or semantically equivalent code

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
MOV [X], EAX
```

```
XOR EAX, EAX
ADD EAX, [X]
ADD EAX, [Y]
MOV [X], EAX
```

- Register Renaming

- Registers are renamed and replaced throughout code. Therefore, code is equivalent in terms of semantics.

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
MOV [X], EAX
```

```
MOV ECX, [X]
MOV EAX, [Y]
ADD ECX, EAX
MOV [X], ECX
```

Detecting Metamorphic Code

- Identifying two pieces of code is *equivalent in nature* (same execution behavior) is undecidable
- But we can still detect
 - Approximations exist
 - A real complete metamorphic malware is hard to build
 - W32/Simile and others are limited in scope.
 - Complex problems to solve itself
- Techniques based on “semantics” rather than “syntax” is better in detecting metamorphic code

Semantic Aware Malware Detection

- Method is to create signature from the “semantic” properties of code using *static analysis*.
- Signature is a semantic template $T = (I_T, V_T, C_T)$, where
 - I_T is a sequence of instructions
 - V_T is a set of variables
 - C_T is a set of symbolic constants (also contains functions)
- An *execution context* for a template is assignment to constants
- A *state* is assignment to variables, program counter and memory
- A program instance is said to satisfy a template if
 - There exists an execution context and initial state for which
 - A sequence of the instructions in the template and malware affect memory in an *equivalent manner*.
 - (This problem is undecidable, can be shown by reducing the halting problem to it)

Semantic Aware Malware Detection (cont...)

- Approximate solution –
 - by making weaker assumptions – not all memory updates need to be equivalent.
 - Differentiate between “core” and “temporary” memory locations
 - Any updates not used for temporary calculation can be treated as “core”
 - Several classes of obfuscation can be supported
 - Instruction reordering
 - Register renaming
 - Garbage insertion
 - Not supported
 - Instruction replacement (limited support)
 - Equivalent functionality
 - Reordered memory access
- Reference: Mihai Christodorescu et. al. “Semantic-Aware Malware Detection”

Anti-Static Analysis

- All syntax and semantic signature generation require some sort of “static analysis”
- Anti-static analysis techniques are used in malware to make it “hard” to perform proper static analysis.
 - Static analysis of binary relies on “Disassembly”, therefore obfuscate code to make it hard to disassemble (also called “Anti-disassembly”)
 - Make steps used in static analysis hard
 - Hide function boundaries by not conforming to proper semantics
 - Rely on dynamically computed constructs (such as *opaque predicates*)
 - Encapsulate code (pack)
- Anti-disassembly
 - In the x86 architecture, disassembly is imprecise, reasons:
 - Variable length instructions
 - Code and data can be mixed together
 - Disassembly is done using some algorithms – linear sweep or iterative adaptive
 - Anti-disassembly can be done by
 - Mixing code and data
 - Using a lot of indirect jumps and calls, so that start of code cannot be identified
 - Anti-disassembly aware static analysis that use heuristics can overcome most of these obfuscations but with limitations

Anti-Static Analysis (cont...)

- Code and data encapsulation (packing)
 - Packed code requires unpacking for static analysis
 - Some commercial and free code packers are available pack code and add unpacker routine that use obfuscation
 - UPX
 - PECompact
 - Armadillo
 - And others...
 - The unpack routine is obfuscated hence hard to analyze
 - Also, emulation and interpretation may be used to unpack.
 - Reference- Paul Royal et.al. “Polyunpack”
- Thus, pure static analysis is hard for malware, some dynamic analysis methods are required

Dynamic Analysis on Malware

- Since dynamic analysis require execution, malware is usually analyzed in a “sand-boxed” environment
 - Virtual Machines are mostly used, with network access restricted and watched upon
 - Emulation may be applied
- Debuggers or debugging techniques are used to trace execution
- Dynamic analysis benefits
 - Most anti-static analysis techniques can be defeated
 - Can provide accurate information
- But as stated in the last lecture, dynamic analysis is incomplete though it is precise.

Behavioral Detection

- Behavior based detection is mostly dynamic analysis based
 - Static analysis may be combined
 - Open area for a lot of research
- Some malware exhibit identifiable run-time behavior, for example
 - Spyware that hijack browser
 - Bot programs that access C&C server
 - Adware that may show popups and ads
- Principle
 - Define “templates” of bad behavior
 - Use run-time monitoring to find match
- Templates can be manually or automatically created
- Run-time monitoring is mostly system call, API or an observable behavior
- This is approximate solution - can lead to false positives and negatives
- Reference: Engin Kurda et. al - “Behavior based Spyware Detection”

Dynamic Analysis Evasion

- Dynamic analysis techniques require to execute “malware”.
 - Current trend in malware is detecting dynamic analysis environments or analysis tools and exit or crash
- VM detection is used to detect analysis environments
 - Nopill, Vmdetect, Redpill, Scoopy Doo, Jerry, Vmtools
 - Malware detects the use of VMware or other Virtual Machine environments that are used for safe malware analysis
- Debugger detection is used to detect dynamic analysis tools
 - Most Dynamic Analysis tools work as debugger for run-time analysis
 - Malware uses different techniques to detect that it is being “debugged”
 - Also known debuggers have specific methods of detection

Project Ideas

- For projects use available program analysis tools
 - Static analysis – IDAPRO, Ollydbg, DynInst etc.
 - Dynamic analysis – STRACENT, DynInst etc.
- Polymorphic or metamorphic code detectors
 - Several semantic based ideas have been proposed in the literature.
 - Can they be improved?
- Apply Behavioral Detection on Malware
 - Behavioral detection is still a new and exciting area in research
 - Apply behavioral detection technique on some category of malware – e.g. *bots*, *worms*
 - Use a combination of static and dynamic analysis
- Analysis Evasion Techniques
 - Develop techniques that evade static analysis and dynamic analysis approaches