Building Malware Infection Trees

Jose Andre Morales[1], Michael Main[2], Weilang Luo[3],
Shouhuai Xu[2,3], Ravi Sandhu[2,3]

[1]Software Engineering Institute, Carnegie Mellon University[*]

[2]Institute for Cyber Security, University of Texas at San Antonio

[3]Department of Computer Science, University of Texas at San Antonio

[*]Research performed in the University of Texas at San Antonio

MALWARE 2011
KNOW YOUR ENEMY
Research • Practical Solutions (Industry Track) • The Law

IEEE computer society

UNCLASSIFIED

Introduction - 1

- Present an abstract approach creating malware infection trees (MiT) effectively & efficiently
- Capture relevant processes & files created/modified during execution
- Construction primarily based on rules describing ***fundamental*** malware infection execution events
- Gives meaningful understanding of malware infection & identifies involved processes
- Implemented on Windows Vista OS
- Useful in disinfection and analysis

Contributions

- Propose an abstract approach to building malware infection trees (MiTs).
- Define execution event rules describing essential components of infection strategies.
- Describe implementation in the Windows Vista OS User and Kernel levels.

Strong & Weak Bonds

- Based on observed interaction between processes and files during execution
- Establishes meaningful relationship between nodes, justifying their addition to MiT
- Strong Bond between Q & P:
  - Transitivity (transfer) of data from Q to P, creates an intersection between Q & P where both have a subset of identical data
- Weak Bond:
  - Node Q creates a node P with no transfer of data, this becomes a creator/created relationship with no intersection of identical data between both
- A strongly bonded tree
  - Contains processes & files directly descending from original malware executable possibly essential to infection strategies
  - Enhanced with weak bonds contains non-direct descendants and possibly non-essential files & process

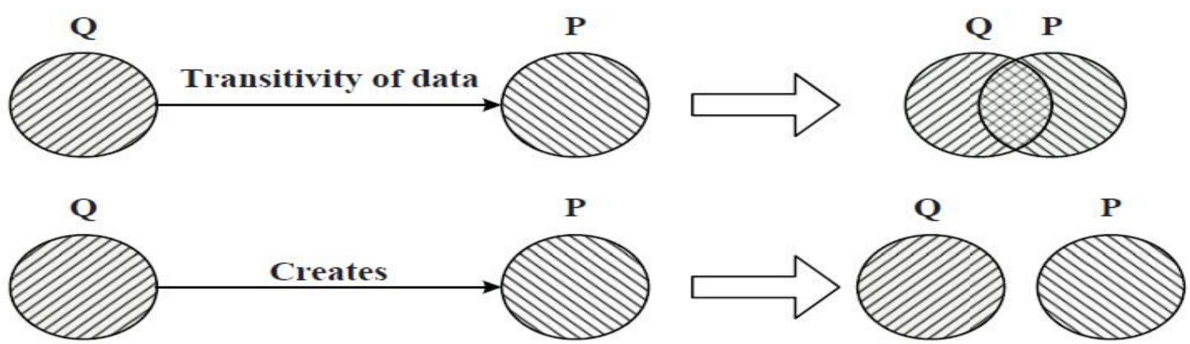


Figure 1. MiT strong & weak bonds

Construction Rules

- Set of rules defining when a node is added to a MiT primarily based on fundamental definitions of malware (cohen, adleman)
- Rules facilitate strong & weak bonds resulting in highly relevant MiT with minimal non-essential items included
- Generalized for use on multiple OS's → not OS specific
- New rules can be added
- Node N is a file or currently running process
- Edge E is an observed rule between nodes N1 and N2, where N2 is newly added to the MiT
- Sample under analysis assumed Root of MiT
- Assumption: Malware primarily infects through file & process manipulation
- File system & Process rules

File System Rules

- Captures malware self replication & arbitrary file creation
  - Self replication considered essential to malware infection and propagation
  - 3 rules currently defined
- F1:Infection via self replication, strong bond
- F2:Infection via arbitrary file creation, weak bond
- F3:Infection via arbitrary file write modification, weak bond

Process Rules

- Capture malware's manipulation of processes for nefarious uses
  - Dynamic code injection
  - Process Spawning
  - 2 rules currently defined
- P1: Infection via dynamic code injection of a currently running process, strong bond
  - Targets already running (benign) processes
- P2: Infection via process spawning, weak bond
  - Spawned from malware related (created or downloaded) executable file

Windows Vista Implementation

- MiTCoN: tool which outputs MiT tables in real time using dynamic analysis
- Command line with absolute path of file to analyze
- Detects rule occurrence through SSDT hooking in the Windows Vista Kernel, runs as a kernel service.
- Produces very robust MiT tables in under 5 minutes
- Tables later converted to graphical tree representation

File System Rules - Implementation

- 1st set to detect F1 (self replication)

```
ZwCreateFile(in:read_access, in:sourcepath, out:filehandle);
ZwCreateSection(in:filehandle, out:sectionhandle);
ZwMapViewofSection(in:sectionhandle,
   in:processhandle, out:baseaddress);
ZwWriteFile(in:baseaddress, out:targetfilepath)
```

- 2nd set to detect F1

```
ZwReadFile(in:sourcepath, out:memaddress);
ZwWriteFile(in:memaddress, out:targetfilepath);
```

- F2 (arb. File create)→ ZwCreateFile(sourcepath) != caller process
- F3 (arb. File write/mod)→ 2nd Set, ZwReadFile(sourcepath) != caller process

Process Rules - Implementation

- P1: dynamic code injection

```
ZwAllocateVirtualMemory(in:processhandle, out:baseaddress);
ZwWriteVirtualMemory(in:processhandle, in:baseaddress);
ZwCreateThread(in:processhandle, out:threadhandle);
```

- P2: Spawn process
  - 1st sequence

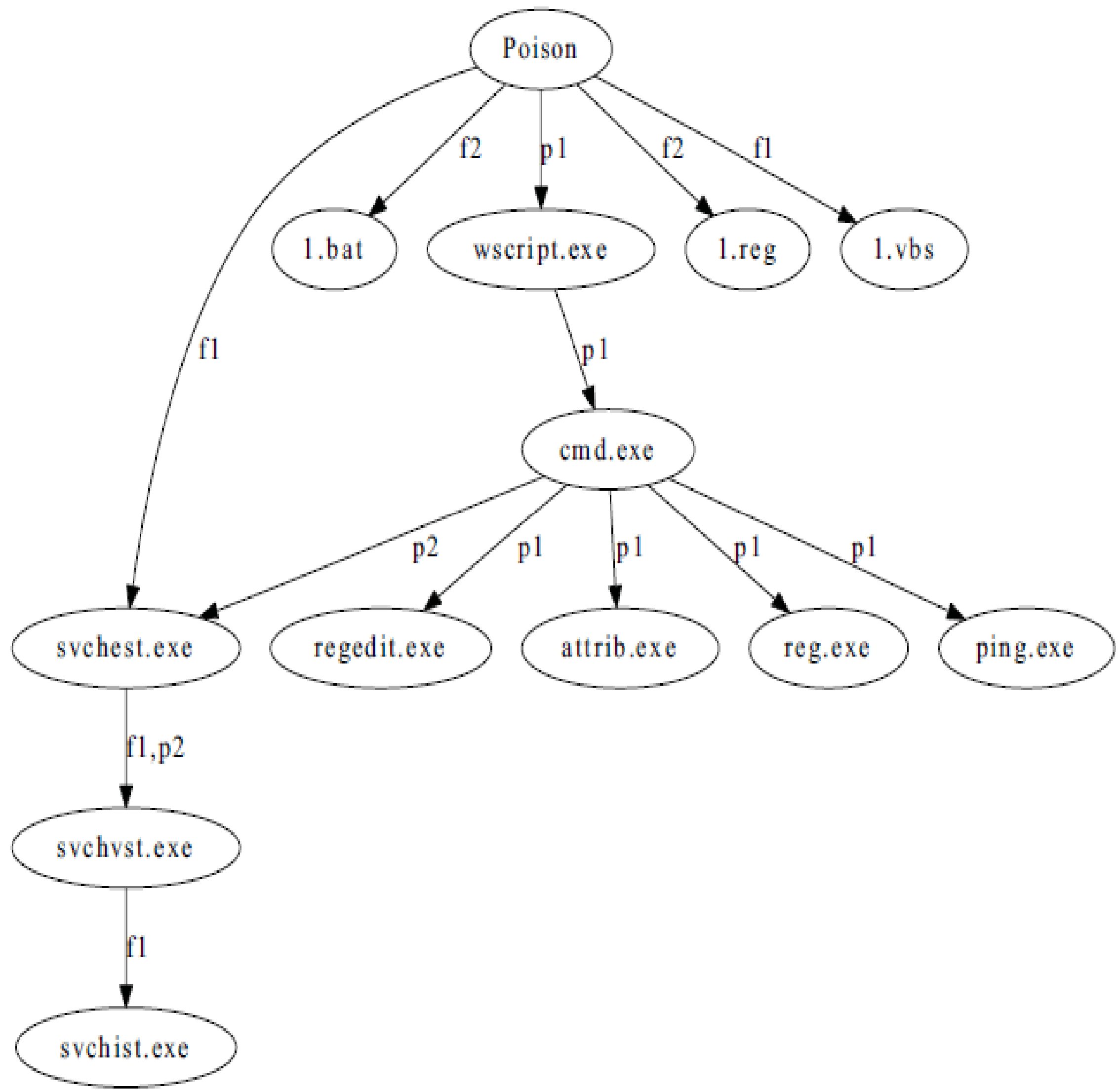
```
ZwCreateProcess(in:objectattributes, out:processhandle)
```

  - 2nd sequence (service creation)

```
CreateService(in:BinaryPath, out:servicehandle);
OpenService(inout: servicehandle);
StartService(in: servicehandle);
```

MiTCoN Example: BackDoor.Win32.Poison

Source	Rule(s)	Destination
Poison	$f1$	svchest.exe
Poison	$f2$	1.bat
Poison	$f2$	1.reg
Poison	$f1$	1.vbs
Poison	$p1$	wscript.exe
wscript.exe	$p1$	cmd.exe
cmd.exe	$p1$	regedit.exe
cmd.exe	$p1$	attrib.exe
cmd.exe	$p1$	reg.exe
cmd.exe	$p1$	reg.exe
cmd.exe	$p2$	svchest.exe
cmd.exe	$p1$	ping.exe
svchest.exe	$f1,p2$	svchvst.exe
svchvst.exe	$f1$	svchist.exe



Evaluation & Results

- Built MiTs for 5800 diverse samples from GFI Sandbox Malware Repository
  - 3 min execution; max cpu: 3%, avg. cpu ~1%; max RAM: 14mb; MiT create max: ~7sec, avg. 3.1sec
  - Rule occurrence – F1:662 , F2:14396 , F3:38629 , P1:647 , P2:3490
  - Rule time occurrence max: 200ms, avg.: 12ms; f1 11ms avg; p1 14ms avg.
  - Most common sequence: F1,P2

Evaluation & Results

- 120 samples analyzed by Anubis and GFI Sandbox compared to out MiTs
  - 114 had nodes in both not recorded in the MiT
  - these nodes were files and/or processes belonging to standard Windows and not part of the malware infection → false positives
  - Example: services.exe

Evaluation & Results

- Attempted disinfection on the 120 samples
- 1. Each sample executed and MiT constructed, VM image scanned by Kaspersky
- 2. Each sample executed and files/processes appearing in MiT were erased from VM image then scanned by Kaspersky
- 100% success, in 2. Kaspersky did not detect any malware presence
  - Implies our MiT sufficed to remove malware infection from system

Evaluation & Results

- Multiple occurrences of strong bonds, may suffice to understand essential infection strategy
- Efficient and Effective
- MiT construction based on fundamental malware infection creates relevant MiTs excluding files and processes of standard Windows operations.
- The high frequency, early occurrence and low false positives makes our rules for building MiTs highly effective in analyzing malware.
- Our disinfection produce 100% success, MiTs may be useful in aiding disinfection efforts

Conclusions & Future Work

- Abstract approach to create MiTs produce relevant meaningful description of infection strategy
- Construction rules drawn from fundamental malware definitions focusing on infection strategies
- Generalized and Extensible, OS independent
- Highly efficient construction, very low resource usage, fast construction time
- MiTs strong bonds included essential players of infection, minimized inclusion of non-infection related processes and files
- Effective in disinfection, can aide other disinfection strategies
- Future work involves expanding the rule set, testing sound and completeness, experiments in real and virtual environments

Questions?

¿Preguntas?

質問

ВопросыВопросы

Sawaal

Domande

Soru

ΕρωτήσειςΕρωτήσεις

問題

CMU copyright statement