Automatic Generation of Remediation Procedures for Malware Infections

Roberto Paleari, Lorenzo Martignoni, Emanuele Passerini, Drew Davidson, Matt Fredrikson, Jon Giffin, Somesh Jha

Università degli Studi di Milano, Università degli Studi di Udine, University of Wisconsin, Georgia Institute of Technology

19th USENIX Security Symposium
Remediation of a system infected by a malware
Why not just blocking the malware before the infection?

- AV not installed
- Signatures not up-to-date
- Signatures not available
Why not just blocking the malware before the infection?

- AV not installed
- Signatures not up-to-date
- Signatures not available

To reinstall the system or to remediate the infection?
Why not just blocking the malware before the infection?

- AV not installed
- Signatures not up-to-date
- Signatures not available

To reinstall the system or to remediate the infection?
A sample malware

1. Generates random file and key names
   - file = "po" + rand() + rand() + rand() + ".exe"
   - key = rand() % 2 ? "qv" : "vq"
2. Drop malicious exe
   - c:\windows\ + file
3. Start the new exe at boot
   - KEY_LOCAL_MACHINE + ...\Windows\CurrentVersion\Run\ + key, file
4. Infect system dll
   - user32.dll
5. Hijack network traffic
   - c:\windows\system32\drivers\etc\hosts,
6. Delete main exe
Sound and complete remediation of malware infection

1. Generates random file and key names
   - file = "po" + rand() + rand() + rand() + ".exe"
   - key = rand() % 2 ? "qv" : "vq"

2. Drop malicious exe
   - c:\windows\ + file

3. Start the new exe at boot
   - KEY_LOCAL_MACHINE + ...\Windows\CurrentVersion\Run\ +
     key, file

4. Infect system dll
   - user32.dll

5. Hijack network traffic
   - c:\windows\system32\drivers\etc\hosts,

6. Delete main exe
Hardness of sound and complete remediation

- The code of 😂 is typically obfuscated and an analysis of the potential behaviors is very difficult.
- The behavior of 😂 is typically non-deterministic.
- No clue on the state of the system before the infection because remediation is performed a posteriori.
Hardness of sound and complete remediation

1. Generates random file and key names
   - file = "po" + rand() + rand() + rand() + ".exe"
   - key = rand() % 2 ? "qv" : "vq"

2. Drop malicious exe
   - c:\windows\ + file

3. Start the new exe at boot
   - KEY_LOCAL_MACHINE + ...\Windows\CurrentVersion\Run\ + key, file

4. Infect system dll
   - user32.dll

5. Hijack network traffic
   - c:\windows\system32\drivers\etc\hosts, www.google.com, www.citibank.com

6. Delete main exe
Are AVs good at remediating malware infections?
Are AVs good at remediating malware infections?

More or less . . .

[Passerini et al., DIMVA 2008]
Seeking sound and complete remediation
Seeking sound and complete remediation

\[ S_1 \]
Seeking sound and complete remediation

Create `c:\windows\powxv.exe`

Create `c:\windows\powxw.exe`

Create `c:\windows\pooig.exe`
Given $s'_1$, and without knowing $s_1$, our goal is to obtain $s''_1$ by reverting **all** and **only** the modifications caused by 😈.
Seeking sound and complete remediation

The approach must work for any $s_i \in S$

$$c:\windows\po*.exe$$
Seeking sound and complete remediation

The approach must work for any $s_i \in S$

$(c:\windows\po*.exe)$
Our approach for sound and complete remediation

1. Construct $\mathcal{R}$: the infection relation of 🦊

2. Construct $\mathcal{R}^{-1}$: the remediation procedure for 🦊

3. Run $\mathcal{R}^{-1}$ on the infected system
Our approach for sound and complete remediation

1. Construct $\mathcal{R}$: the **infection relation** of 🐇
   - Construction based on dynamic analysis
   - Over-approximation of the behaviors observed

2. Construct $\mathcal{R}^{-1}$: the **remediation procedure** for 🐇

3. Run $\mathcal{R}^{-1}$ on the infected system
Our approach for sound and complete remediation

1. Construct $\mathcal{R}$: the infection relation of 🐱
   - Construction based on dynamic analysis
   - Over-approximation of the behaviors observed

2. Construct $\mathcal{R}^{-1}$: the remediation procedure for 🐱
   - Reverts all possible effects of the infection

3. Run $\mathcal{R}^{-1}$ on the infected system
Our approach for sound and complete remediation

1. Construct $\mathcal{R}$: the \textit{infection relation} of 🦇
   - Construction based on dynamic analysis
   - Over-approximation of the behaviors observed

2. Construct $\mathcal{R}^{-1}$: the \textit{remediation procedure} for 🦇
   - Reverts all possible effects of the infection

3. Run $\mathcal{R}^{-1}$ on the infected system
   - Constraint-based detection of system resources affected by 🦇
   - Deletes resources created during the infection
   - Restores resources modified during the infection
   - Creates resources removed during the infection
Our approach for sound and complete remediation
Our approach for sound and complete remediation
Our approach for sound and complete remediation
High-level behavior extraction

Behavior monitoring

High-level behavior analysis

Behavior clustering

Cluster generalization

Remediation procedure generation
High-level behavior extraction

Behaviors of interest

- Creation/infection/deletion of files and registry keys
- Drop and execute/auto-start
High-level behavior extraction

The malware is run in multiple VMs with different configurations
(to exercise the largest set of program paths and behaviors)
High-level behavior extraction

$S_1 = s_1, \ldots, s_n$

$S_2 = s_1, \ldots, s_n$

$S_3 = s_1, \ldots, s_n$

$S_4 = s_1, \ldots, s_n$

$S_5 = s_1, \ldots, s_n$

System calls and arguments are traced (using VM introspection)
High-level behavior extraction

High-level behaviors and the corresponding system’s state transitions are extracted from the trace

[Martignoni et al., RAID 2008]
High-level behavior extraction

Behavior summary

- DropAndAutostart("c:\...\powxv.exe", data, "...\Run\vq", "powxv.exe")
- FileCreation("...\powxv.exe", data)
- RegistryCreation("...\Run\vq", "powxv.exe")
- FileDeletion("c:\malware.exe")
- FileInfection("...\etc\hosts", "67.42...", data)

Behavior graph
Behavior generalization

Behavior monitoring

High-level behavior analysis

Cluster generalization

Remediation procedure generation

S\textsubscript{1} \rightarrow S\textsubscript{2} \rightarrow S\textsubscript{3} \rightarrow S\textsubscript{4}

B\textsubscript{1} \rightarrow B\textsubscript{2} \rightarrow B\textsubscript{3} \rightarrow B\textsubscript{4}

C\textsubscript{1} \rightarrow C\textsubscript{2} \rightarrow C\textsubscript{3}

Behavior clustering identifies elements of distinct traces \( \{B^1, \ldots, B^m\} \) that correspond to the same malicious activity.

An admissible clustering for a given set of traces is a set of behavior sets \( \{C_1, C_2, \ldots, C_n\} \) that satisfies two conditions:

1. All behaviors in a given cluster \( C_i \) have the same type.
2. The clustering partitions the set of all events in every execution trace (no behavior is in more than one cluster, and each behavior is in some cluster).
Behavior clustering

- Clustering performed on behavior graphs (non-determinism in a malicious program typically affects the summary of the behavior, but not the low-level operations used to achieve the behavior)
- Graphs compared for isomorphism
- Normalization applied before comparison (sequential syscalls are merged, killed syscalls are dropped, . . . )
Dealing with non-determinism: behavior generalization

Output of clustering ($C_i$)

DropAndAutostart("c:\windows\po\agp.exe", data,"\...\Run\vq","poagp.exe")
DropAndAutostart("c:\windows\po\bxz.exe", data,"\...\Run\vq","pobxz.exe")
DropAndAutostart("c:\windows\po\cra.exe", data,"\...\Run\qv","pocra.exe")
DropAndAutostart("c:\windows\po\mfq.exe", data,"\...\Run\vq","pomfq.exe")
DropAndAutostart("c:\windows\po\mmp.exe", data,"\...\Run\qv","pommp.exe")
DropAndAutostart("c:\windows\po\pwz.exe", data,"\...\Run\qv","popwz.exe")
DropAndAutostart("c:\windows\po\uwk.exe", data,"\...\Run\vq","pouwk.exe")
Dealing with non-determinism: behavior generalization

Output of clustering ($C_i$)

DropAndAutostart("c:\windows\po\texttt{agp.exe}", data,"...\Run\texttt{vq}","po\texttt{agp.exe}")
DropAndAutostart("c:\windows\po\texttt{bxz.exe}", data,"...\Run\texttt{vq}","po\texttt{bxz.exe}")
DropAndAutostart("c:\windows\po\texttt{cra.exe}", data,"...\Run\texttt{qv}","po\texttt{cra.exe}")
DropAndAutostart("c:\windows\po\texttt{mfq.exe}", data,"...\Run\texttt{vq}","po\texttt{mfq.exe}")
DropAndAutostart("c:\windows\po\texttt{mmp.exe}", data,"...\Run\texttt{qv}","po\texttt{mmp.exe}")
DropAndAutostart("c:\windows\po\texttt{pwz.exe}", data,"...\Run\texttt{qv}","po\texttt{pwz.exe}")
DropAndAutostart("c:\windows\po\texttt{uwk.exe}", data,"...\Run\texttt{vq}","po\texttt{uwk.exe}")

How to deal with slightly different behaviors?

DropAndAutostart("c:\windows\po\texttt{jsh.exe}", data,"...\Run\texttt{vq}","po\texttt{jsh.exe}")
DropAndAutostart("c:\windows\po\texttt{euj.exe}", data,"...\Run\texttt{qv}","po\texttt{euj.exe}")
Dealing with non-determinism: behavior generalization

Output of clustering \( (C_i) \)

\[
\text{DropAndAutostart("c:\windows\poagp.exe", data,"...\Run\vq","poagp.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\pobxz.exe", data,"...\Run\vq","pobxz.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\pocra.exe", data,"...\Run\qv","pocra.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\pomfq.exe", data,"...\Run\vq","pomfq.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\pommp.exe", data,"...\Run\qv","pommp.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\popwz.exe", data,"...\Run\qv","popwz.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\pouwk.exe", data,"...\Run\vq","pouwk.exe")}
\]

How to deal with slightly different behaviors?

\[
\text{DropAndAutostart("c:\windows\pojis.exe", data,"...\Run\vq","pojis.exe")}
\]
\[
\text{DropAndAutostart("c:\windows\poejj.exe", data,"...\Run\qv","poejj.exe")}
\]

Transform a cluster into a generalized behavior summary
Behavior generalization

- All behaviors in the cluster have the same “prototype”
- Each argument of the behavior is independent from the others
- Each argument can be canonicalized into a string

- The generalization of each argument can be seen as the generalization of a set of strings
- The generalization of a set of strings can be accomplished by constructing an automaton that accepts a superset of the set of strings
Behavior generalization

$A_1$

DropAndAutostart "c:\windows\poagp.exe" data,"...\Run\vq","poagp.exe")
DropAndAutostart "c:\windows\pobxz.exe" data,"...\Run\vq","pobxz.exe")
DropAndAutostart "c:\windows\pocra.exe" data,"...\Run\qv","pocra.exe")
DropAndAutostart "c:\windows\pomfq.exe" data,"...\Run\vq","pomfq.exe")
DropAndAutostart "c:\windows\pommp.exe" data,"...\Run\qv","pommp.exe")
DropAndAutostart "c:\windows\popwz.exe" data,"...\Run\qv","popwz.exe")
DropAndAutostart "c:\windows\pouwk.exe" data,"...\Run\vq","pouwk.exe")
Behavior generalization

Construct minimal (probabilistic) FSA that accepts all strings in $A_1$
Behavior generalization

Identify **dense** single-entry-single-exit regions of the FSA
(regions affected by non-determinism)
Behavior generalization

Apply generalization rules to dense SESE regions
(rules based on the length of paths, type of symbols, ...)

\[>:alpha:][3] \]
Behavior generalization

\texttt{c:\windows\po[[:alpha:]]\{3\}.exe}

Transform FSA into a regular expression
Finally comes $R$

\begin{verbatim}
DropAndAutostart("c:\windows\poagp.exe",data,"...\Run\vq","poagp.exe")
DropAndAutostart("c:\windows\pobxz.exe",data,"...\Run\vq","pobxz.exe")
DropAndAutostart("c:\windows\pocra.exe",data,"...\Run\qv","pocra.exe")
DropAndAutostart("c:\windows\pomfq.exe",data,"...\Run\vq","pomfq.exe")
DropAndAutostart("c:\windows\pomm.mp.exe",data,"...\Run\qv","pomm.mp.exe")
DropAndAutostart("c:\windows\popwz.exe",data,"...\Run\qv","popwz.exe")
DropAndAutostart("c:\windows\pouwk.exe",data,"...\Run\vq","pouwk.exe")
\end{verbatim}

\begin{verbatim}
DropAndAutostart("c:\windows\po[[:alpha:]]{3}.exe", data, "...\Run\(vq|qv)", "po[[:alpha:]]{3}.exe")
\end{verbatim}
Remediation procedures generation

Behavior monitoring

High-level behavior analysis

Cluster generalization

Behavior clustering

Remediation procedure generation

Remediation procedures in action

★ Newly-created resources: remove resources created by 😈

★ Infected resources: restore resources tampered by 😈

★ Deleted resources
Remediation procedures in action

- **Newly-created resources**: remove resources created by 😷
  - Use constraints on arguments to identify the resources to remove
  - `FileCreation(file, data)`: remove `file` if its content matches `data`
  - `DropAndAutostart(file, data, key, regdata)`: remove `file` and `key` if both exist, their content matches respectively `data` and `regdata`, and the basename of `file` and `regdata` is the same

- **Infected resources**: restore resources tampered by 😷

- **Deleted resources**
Remediation procedures in action

- **Newly-created resources**: remove resources created by 😷
  - Use constraints on arguments to identify the resources to remove
  - `FileCreation(file, data)`: remove `file` if its content matches `data`
  - `DropAndAutostart(file, data, key, regdata)`: remove `file` and `key` if both exist, their content matches respectively `data` and `regdata`, and the basename of `file` and `regdata` is the same

- **Infected resources**: restore resources tampered by 😷
  - Same approach used to identify newly-created resources
  - Use pristine regions of the file to identify the version to restore
  - Applicable only to system files and keys

- **Deleted resources**
Experimental evaluation

Setup & methodology

- Prototype for Microsoft Windows XP
- Evaluation of over **200 malware samples**
- For each sample:
  1. Monitored 3 executions in 4 distinct environments
  2. Generated remediation procedures
  3. Infected 25 test environments and ran the generated procedure

Goals

- Quantify false-positives & false-negatives (procedure affects a resources not involved with the infection or misses a resources not involved with the infection)
- Compare completeness of remediation procedures available in the best commercial AVs against ours
Experimental evaluation: false-positives

Only one case, caused by a very generic regular expression
(.*\exe)
Experimental evaluation: false negatives

<table>
<thead>
<tr>
<th></th>
<th>Our approach</th>
<th>Nod32</th>
<th>Panda</th>
<th>Kaspersky</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files (primary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
<tr>
<td>Files (ancillary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
<tr>
<td>Reg. keys (primary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
<tr>
<td>Reg. keys (ancillary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
<tr>
<td>Processes (primary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
<tr>
<td>Processes (ancillary)</td>
<td></td>
<td></td>
<td></td>
<td>[98% vs. 82%]</td>
</tr>
</tbody>
</table>
## Experimental evaluation: false negatives

<table>
<thead>
<tr>
<th>% activities reverted</th>
<th>Our approach</th>
<th>Nod32</th>
<th>Panda</th>
<th>Kaspersky</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files (primary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Files (ancillary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reg. keys (primary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reg. keys (ancillary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processes (primary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processes (ancillary)</td>
<td>98%</td>
<td>82%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Limitations

- Dynamic behavior-based analysis can give incomplete results
- Attackers may try to induce very aggressive behavior generalization
- Only a subset of modified resources can be properly restored
- Deleted resources cannot be restored
Conclusions

Remediation of a system infected by a malware

Our approach for sound and complete remediation

Hardness of sound and complete remediation

1. Generates random file and value names
   - `file = "po" + rand() + rand() + rand() + rand() + ".exe"`
   - `value = rand() % 2 ? "qv" : "vq"

2. Drop malicious exe
   - `c:\windows\ + file`

3. Start the new exe at boot
   - `KEY_LOCAL_MACHINE + ...\Windows\CurrentVersion\Run, value, file`

4. Infect system dll
   - `user32.dll`

5. Hijack HTTP connections
   - `c:\windows\system32\drivers\etc\hosts, www.google.com, www.citibank.com`

6. Delete main exe

Experimental evaluation: false negatives

98% vs. 82%
Automatic Generation of Remediation Procedures for Malware Infections

Thank you!
Any questions?

Lorenzo Martignoni
lorenzo.martignoni@uniud.it