TROGUARD:
Context-Aware Protection Against Web-Based Socially Engineered Trojans

Rui Han, Alejandro Mesa, University of Miami
Mihai Christodorescu, Qualcomm Research
Saman Zonouz, Rutgers University
Motivation
## Mac OS threats

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trojan.OSX.FakeCo.a</td>
<td>52%</td>
</tr>
<tr>
<td>2</td>
<td>Trojan-Downloader.OSX.Jahlav.d</td>
<td>8%</td>
</tr>
<tr>
<td>3</td>
<td>Trojan-Downloader.OSX.Flashfake.ai</td>
<td>7%</td>
</tr>
<tr>
<td>4</td>
<td>Trojan-Downloader.OSX.FavDonw.c</td>
<td>5%</td>
</tr>
<tr>
<td>5</td>
<td>Trojan-Downloader.OSX.FavDonw.a</td>
<td>2%</td>
</tr>
<tr>
<td>6</td>
<td>Trojan-Downloader.OSX.Flashfake.ab</td>
<td>2%</td>
</tr>
<tr>
<td>7</td>
<td>Trojan-FakeAV.OSX.Defma.gen</td>
<td>2%</td>
</tr>
<tr>
<td>8</td>
<td>Trojan-FakeAV.OSX.Defma.f</td>
<td>1%</td>
</tr>
<tr>
<td>9</td>
<td>Exploit.OSX.Smid.b</td>
<td>1%</td>
</tr>
<tr>
<td>10</td>
<td>Trojan-Downloader.OSX.Flashfake.af</td>
<td>1%</td>
</tr>
</tbody>
</table>

McAfee antivirus solution: [http://www.securelist.com](http://www.securelist.com)
# Example Malwares

<table>
<thead>
<tr>
<th>Malware</th>
<th>Descriptions</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrojanClicker.VB .395</td>
<td>Trojan socially engineered as adobe flash update</td>
<td>Windows and Mac OS X</td>
</tr>
<tr>
<td>Faked Anti-Virus</td>
<td>Trojan or Adware socially engineered as anti-virus software</td>
<td>Windows, Mac OS X, and Linux</td>
</tr>
<tr>
<td>Opfake Browser</td>
<td>Malware socially engineered as Opera Browser</td>
<td>Android</td>
</tr>
<tr>
<td>WireLuker</td>
<td>Legitimate applications socially engineered with ad-wares and Trojan</td>
<td>Mac OS X and iOS</td>
</tr>
</tbody>
</table>
Contributions

• Answer the question: “Is this program doing what I expected it to do?”

• Bridge the semantic gap between functionality classes and low level behaviors

• Built on 100 Linux app profiles

• High detection rate on 50 Trojan apps
TROGUARD Architecture

**Offline**
- Application Functionality Tracing
- Dynamic Functionality Feature Extraction
- Functionality Class Profile Generation

**Online**
- Inference of Perceived Functionality Class
- Application Functionality Tracing
- Real-Time Classification
- Alert

- Application Database
- Download Website
- Downloaded Application

**TROGUARD**

**sandbox**
Key Premise

• TROGUARD detects Trojans based on the premise that applications with *similar functionalities* expose *similar system-level behaviors*

• Applications with similar functionalities belong to a *functionality class*, they should exhibit common system level behaviors
  
  • *Learn web-browser behavior of well known instances*(e.g., Firefox and Chrome)*
  
  • *Compare the web-browser profile with the behaviors of unknown downloaded web-browser app*
Functionality class

• It represents both user’s understanding of software category and the system’s observation of a software execution behavior

<table>
<thead>
<tr>
<th>Functionality class</th>
<th>softpedia.com</th>
<th>download.cnet.com</th>
<th>tucows.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic Editor</td>
<td>Artistic software</td>
<td>Graphic Design SW</td>
<td>Design tools</td>
</tr>
<tr>
<td>Game</td>
<td>Games</td>
<td>Games</td>
<td>Games</td>
</tr>
<tr>
<td>Browser</td>
<td>Internet</td>
<td>Browsers</td>
<td>Internet</td>
</tr>
<tr>
<td>Instant Messenger</td>
<td>Communications</td>
<td>Communications</td>
<td></td>
</tr>
<tr>
<td>Media Player</td>
<td>Multimedia</td>
<td>MP3/Audio Software</td>
<td>Audio/Video</td>
</tr>
<tr>
<td>Audio Editor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Editor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text Editor</td>
<td>office</td>
<td>Productivity software</td>
<td>Business</td>
</tr>
<tr>
<td>IDE</td>
<td>Programming</td>
<td>Developer Tools</td>
<td>Develop/Web</td>
</tr>
<tr>
<td>Calculator</td>
<td>Utilities</td>
<td>Utilities</td>
<td>Home/Education</td>
</tr>
</tbody>
</table>
# Applications

<table>
<thead>
<tr>
<th>Class</th>
<th>Studied Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Graphic Editor</td>
<td>gimp, pinta, imagej, inkscape, kolourpaint, rawtherapee, mypaint, gpaint, gnome-paint, pencil</td>
</tr>
<tr>
<td>2. Games</td>
<td>sol, wesnoth, glchess, neverball, kmahjongg, supertuxkart, hedge- wars, pingus, frozen-bubble, eboard</td>
</tr>
<tr>
<td>3. Browser</td>
<td>chrome, firefox, opera, epiphany, midori, chromium, netsurf, arora, xxxterm, rekonq</td>
</tr>
<tr>
<td>4. Instant Messenger</td>
<td>skype, kmess, emesene, kopete, pidgin, psi, gajim, empathy, aMSN, qutim</td>
</tr>
<tr>
<td>5. Media Player</td>
<td>smplayer, vlc, audacious, quodli- bet, gmusicbrowser, qmmp, abraca, amarok, guayadeque, aqualung</td>
</tr>
<tr>
<td>6. Audio Editor</td>
<td>audacity, avidemux, dvbcut, og- gconvert, kwave, wavbreaker, mp3splt-gtk, mhwaveedit, fillmore, soundconverter</td>
</tr>
<tr>
<td>7. Video Editor</td>
<td>openshot, lives, iriverter, kino, pitivi, videocut, winff, arista-gtk, kdenlive, curlew</td>
</tr>
<tr>
<td>8. Text Editor</td>
<td>kile, geany, texmaker, calligra- words, soffice.bin, lyx, tea, jed, emacs, vi</td>
</tr>
<tr>
<td>9. IDE</td>
<td>anjuta, codelite, codeblocks, net- beans, monodevelop, kdevelop, spyder, monkeystudio, drracket, idle</td>
</tr>
<tr>
<td>10. Calculator</td>
<td>grpn, gcalctool, EdenMath, speed- crunch, kcalc, keurocalc, extcalc, gip, calculator, gnome-genius</td>
</tr>
</tbody>
</table>
Functionality Tracing

• Manual testing
• Run 60 seconds for each application
• System call trace
• User-space information
  
  *User interactivity*

  *Resource consumption*

  *IP addresses and port number*
Feature Extraction

- Processing tracing data
- Four groups of feature
  - file system
  - Network
  - resource usage
  - user interactivity
- Intermediate feature
Example: if( libssl3.so & fd = sys_socket(AF_INET, ..) & sys_write(fd, ..) & sys_read(fd, ..) ) HTTP = true
TROGUARD Architecture

Offline

- Application Database
  - Application Functionality Tracing
  - Dynamic Functionality Feature Extraction
  - Functionality Class Profile Generation

Online

- Download Website
- Downloaded Application
  - Inference of Perceived Functionality Class
  - Application Functionality Tracing
  - Real-Time Classification
  - Alert

Application Functionality Profile Database
Web Page Analysis

• Give the explicit functionality class
• Web page contents analysis
• OCR to extract the texts in the images
• Analysis based on keywords
User Interface

• Browser extension for web page analysis
Sandboxing

• SELinux sandbox
• One policy for each app class
• Automatically generated by parsing all the logs from an app class
Classifier Evaluation

- 600 data points (10 second each)
- 10 fold cross validation
- 5 classifiers with different feature group
- Precision
- Recall
- Confusion Matrix
Precision

Different Attributes Domains

File
Network
CPU-Mem
Interaction
All

Browser
Office
IM
Game
IDE
Media-Player
Graphic-Editor
Video-Editor
Audio-Editor
Calculator
Average

Precision
Recall

Different Attributes Domains

Recall

Average
Confusion Matrices

File features

Network features

Resource usage features

User interactivity features
Confusion Matrices

![Confusion Matrix Image]
Intermediate Feature Results
Web Page Analysis Accuracy

- 100 Web page, 20 categories

Accuracy

Text analysis
OCR analysis
## Case Study

- 10 benign apps × 5 payload = 50 Trojans

<table>
<thead>
<tr>
<th>Functionality Class</th>
<th>Application</th>
<th>Metasploit Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Graphic Editor</td>
<td>gpaint</td>
<td>linux/x86/shell_bind_tcp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linux/x86/shell/reverse_tcp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linux/x86/vncinject/bind_tcp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linux/x86/meterpreter/bind_tcp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>linux/x86/download_exec</td>
</tr>
<tr>
<td>2. Games</td>
<td>eboard</td>
<td></td>
</tr>
<tr>
<td>3. Browser</td>
<td>xxxterm</td>
<td></td>
</tr>
<tr>
<td>4. Instant Messenger</td>
<td>psi</td>
<td></td>
</tr>
<tr>
<td>5. Media Player</td>
<td>qmmp</td>
<td></td>
</tr>
<tr>
<td>6. Audio Editor</td>
<td>winff</td>
<td></td>
</tr>
<tr>
<td>7. Video Editor</td>
<td>fillmore</td>
<td></td>
</tr>
<tr>
<td>8. Text Editor</td>
<td>tea</td>
<td></td>
</tr>
<tr>
<td>9. IDE</td>
<td>spyder</td>
<td></td>
</tr>
<tr>
<td>10. Calculator</td>
<td>gnome-genius</td>
<td></td>
</tr>
</tbody>
</table>
Case Study

• Predefined acceptance rate 0.8
Symbolic Execution

• Tested Core Utilities (four functionality classes)
  
  Dirlist
  Filetype
  Userinfo
  Systeminfo

• Features collected from symbolic execution give us 52% precision

• Features collected from user execution give us 76% precision
User Execution VS Symbolic Execution

![Bar Chart showing comparison between User Inputs Covered Basic Blocks and Symbolic Execution Covered Basic Blocks for various inputs.](chart.png)
Performance Overhead

• CPU usage:
Performance Overhead

- Memory usage:
Performance Overhead

- Disk throughput:
Performance Overhead

- Network throughput:
Conclusions

• TROGUARD detects Trojans based on the premise that applications with similar functionalities expose similar system-level behaviors

• TROGUARD can detect Trojan application download by bridging the gap between the user perceived functions and genuine software functions
Thank you!

Questions?

Rui Han
r.han@umiami.edu
Conclusions

• TROGUARD detects Trojans based on the premise that applications with similar functionalities expose similar system-level behaviors

• TROGUARD can detect Trojan application download by bridging the gap between the user perceived functions and genuine software functions

Questions?
Symbolic Execution Code Coverage

Individually Normalized Coverage

1. arch
2. date
3. dirname
4. echo
5. hostname
6. id
7. ln
8. logname
9. ls
10. mkfifo
11. mknod
12. mktemp
13. nproc
14. printenv
15. pwd
16. users
17. whoami
18. who
SE Code Coverage Evolution

![Graph showing code coverage evolution](image)

- Basic Block Coverage
- Function Block Coverage
- Total Touched Functions
- Fully Covered Functions

Normalized Code Coverages

[0% - 100%]