Dark Firmware: A Systematic Approach to Exploring Application Security Risks in the Presence of Untrusted Firmware

Duha Ibdah, Nada Lachtar, Abdulrahman Abu Elkhail, Anys Bacha, Hafiz Malik
Your Private Data is at Risk

• Cloud based services are essential

• Increased concerns about confidentiality
The Quest for Persistent Malware

- Increase in stealth and persistence
- Increase in complexity

Computer Stack:

- Applications
- Operating System
- Firmware
- Hardware
The Quest for Persistent Malware

Computer Stack

Applications
Operating System
Firmware
Hardware
Firmware Vulnerabilities on the Rise

- **Total Firmware CVE Count**
  - 2017: 401
  - 2018: 476
  - 2019: 435

- **Firmware Update CVE Count**
  - 2017: 19
  - 2018: 57
  - 2019: 42

**Computer Stack**
- Applications
- Operating System
- Firmware
- Hardware
Firmware Vulnerabilities on the Rise

A new advanced persistent threat (APT) campaign detected by Kaspersky Lab in January 2019 and estimated to have run between June and November 2018 has allegedly impacted over one million users who have downloaded the ASUS Live Update Utility on their computers.

Kaspersky Lab’s Global Research and Analysis (GReAT) team named this malicious campaign Operation ShadowHammer and, as initially reported by Kim Zetter, it is supposed to have led to the backdoored version of ASUS Live Update being downloaded and installed by more than 57,000 kaspersky users.
Challenges with Firmware Attacks

- Difficult to gain insight into application layer and how data is managed
- Limited execution cycles are allocated to firmware during runtime
Our Work

• Present a novel attack that harnesses platform management cycles to reliably and efficiently collect sensitive user data

• Conduct a proof-of-concept implementation of the attack using real firmware configured to run on desktop (Ubuntu) and mobile (Android) platforms

• Characterize the robustness of the proposed attack across desktop and mobile systems by extensively testing our attack under stressful app usage conditions

• Devise a low overhead mechanism that does not disrupt normal functionality by limiting its parsing to user accessible pages that are dirty
Harvesting Data in Presence of HTTPS

- Web services and mobile apps rely on HTTPS to securely exchange data
- HTTPS is achieved through adding a TLS layer
- Access data before encryption
Harvesting Data in Presence of HTTPS
Harvesting Data in Presence of HTTPS

POST Request

- **Host:** www.facebook.com
- **Method:** POST
- **Path:** /login/device-based/regular/login
- **Content-Type:** application/x-www-form-urlencoded

```
jazoest=2697&lsd=AVrRLRxFH&email=johnsmith%40gmail.com&pass=pass123&timezone=300&lgn
dim=eyJ3IjoyNTYwLCJoIjoxNDQwLCJhdyI6MjU2MC
```
Harvesting Data in Presence of HTTPS

POST Request

Host: www.facebook.com
Method: POST
Path: /login/device-based/regular/login
Content-Type: application/x-www-form-urlencoded

jazoest=2697&lsd=AVrRLRxH&email=johnsmith%40gmail.com&pass=pass123&timezone=300&lgn
dim=eyJ3IjoyNTYwLCJoIjoxNDQwLCJhdyI6MjU2MC
Dark Firmware: Our Proposed Attack

- Hardware interrupt invokes CPU core
- CPU core parses memory for HTTP requests
- IPI is issued to the next core
- Cores synchronize to continue the search process
- New core resumes memory parsing
Dark Firmware: Our Proposed Attack

• Hardware interrupt invokes CPU core.
• CPU core parses memory for HTTP requests.
• IPI is issued to the next core.
• Cores synchronize to continue the search process.
• New core resumes memory parsing.

Full Memory Search is Expensive
Dark Firmware: A Page-Aware Approach

• Selectively search memory based on flags of page table entries
Dark Firmware: A Page-Aware Approach

- Selectively search memory based on flags of page table entries
- CPU only parses pages with D, U, and W flags set
Experimental Framework

- Real firmware based on UEFI from the Tianocore open source project
- Attack tested on desktop (Ubuntu 18.04 LTS) and mobile (Android 8.1) systems
- QEMU 3.0.50 used for testing multiple platform configurations
## Experimental Framework

<table>
<thead>
<tr>
<th>Category</th>
<th>Desktop Applications (Ubuntu)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social</td>
<td>Corebird (Twitter), Reddit, LinkedIn, Pinterest, Facebook, Twitter, Linkedin, Pinterist Ramme (Instagram), Tumblr, Nextdoor, Wattpad</td>
</tr>
<tr>
<td>Communication</td>
<td>Slack, Skype, Signal, Whatsdesk (WhatsApp), Discord, Viber</td>
</tr>
<tr>
<td>Productivity</td>
<td>Calc (Excel), Impress (Power Point), Writer (Word), Draw (Visio), Gimp, Gmail, Dropbox, Calendar, Todoist PDF, Overleaf, Gmail, Thunderbird, Calendar, Dropbox, Box, Peek (Screen Recorder), Everpad (Evernote), Android Studio, GitKraken, Eclipse, VirtualBox, Toggl, Qualtrics</td>
</tr>
<tr>
<td>Travel &amp; Local</td>
<td>Airbnb, Google Maps, TripAdvisor, Expedia Travel, Uber, Google Maps, TripAdvisor, Uber, Yelp, Lyft, Grubhub</td>
</tr>
<tr>
<td>Health &amp; fitness</td>
<td>WebMD, LiveStrong, MyFitnessPal</td>
</tr>
<tr>
<td>Entertainment</td>
<td>YouTube, Angry Birds, Candy Crush, Spotify, Steam</td>
</tr>
</tbody>
</table>
Persistence vs. Application Usage

• Tested the persistence of authentication data after launching different application mixes
• Each application had different memory requirements and stressed the memory subsystem differently

<table>
<thead>
<tr>
<th>Mix</th>
<th>Category Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>social, communication, productivity, travel &amp; local, health &amp; fitness, entertainment</td>
</tr>
<tr>
<td>2</td>
<td>communication, productivity, travel &amp; local, health &amp; fitness, entertainment, social</td>
</tr>
<tr>
<td>3</td>
<td>productivity, travel &amp; local, health &amp; fitness, entertainment, social, communication</td>
</tr>
<tr>
<td>4</td>
<td>travel &amp; local, health &amp; fitness, entertainment, social, communication, productivity</td>
</tr>
<tr>
<td>5</td>
<td>health &amp; fitness, entertainment, social, communication, productivity, travel &amp; local</td>
</tr>
<tr>
<td>6</td>
<td>entertainment, social, communication, productivity, travel &amp; local, health &amp; fitness</td>
</tr>
</tbody>
</table>
Persistence vs. Application Usage

1. Log in to Facebook
2. Watch cat videos for 30 seconds
3. Play Angry Birds

Search for password

✓

Search for password

✗
Persistence vs. Application Usage

- Users typically run no more than 10 applications concurrently
- Attack was successful after utilizing an average of 22 apps
- 3x increase in vulnerability factor when doubling memory capacity from 4GB to 8GB
Persistence vs. Memory Consumption

1. Log in to Facebook

2. Increase memory stress a 100 MB/minute

3. Search for password

 persists successfully

 does not persist
Persistence vs. Memory Consumption

- Persistence of the credentials depend on the memory consumption
- Eviction occurs after consuming at least 150% of memory capacity
- A significant amount of swap space is consumed before credentials are evicted
Firmware was unable to locate credentials after 900K pages evicted (after 104 min)
Persistence vs. Memory Consumption

- Aggressive page-out rate was observed before credential eviction
Performance Analysis

- Performance overhead increases linearly with memory capacity when doing full search
- Naïve approach doesn’t scale well to larger memory capacities
Performance Analysis

- Overhead increases as more applications are launched, irrespective of the memory capacity.
Performance Analysis

- The number of searched pages is invariant to the memory capacity
- The number of searched pages scales with the number of running apps
Performance Analysis

- Our selective search significantly improves search time
Conclusion

• Propose an attack that covertly leverages platform management cycles to extract sensitive data from the application layer

• Demonstrate that firmware can reliably extract sensitive data without disrupting the normal execution of launched applications

• Discuss a page-aware approach that is up to $4 \times 10^3$ times faster than a full memory search implementation

• Realize a proof of concept to show the attack is practical
Questions?
## Related Work

<table>
<thead>
<tr>
<th>Reference, Year</th>
<th>Approach</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1], 2011</td>
<td>[1] Introspection through hypervisors for extracting sensitive data</td>
<td>Rely on hypervisors whereas our firmware attack can exist on every computing device</td>
</tr>
<tr>
<td>[2], 2019</td>
<td>[2-5] Installing malicious peripherals for reading memory through DMA attacks.</td>
<td>Such attack requires physical access while our attack can be carried out remotely through firmware update or supply chain</td>
</tr>
<tr>
<td>[3], 2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4], 2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5], 2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6], 2015</td>
<td>[6] Altering firmware through overcoming write protection mechanism</td>
<td>We leverage the attacks [6-8] to insert untrusted firmware into the system examine application layer in the presence of untrusted firmware</td>
</tr>
<tr>
<td>[7], 2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8], 2017</td>
<td>[7] Injecting malicious firmware through updates</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[8] Injecting malicious firmware through overcoming security features such as secure boot</td>
<td></td>
</tr>
</tbody>
</table>
References


