CaSE: Cache-Assisted Secure Execution on ARM Processors

NING ZHANG, KUN SUN, WENJING LOU, TOM HOU
Talk Outline

✓ Motivation and Background – Why this work?
✓ Threat Model – What are we defending against?
✓ CaSE: Cache-Assisted Secure Execution – How does it work?
✓ CaSE highlight – Challenges?
✓ Evaluation – How did we do?
✓ Conclusion and future Work
Threat to Mobile devices
ARM TrustZone – Trusted Execution Environment (TEE)

System Wide Protection

- Divides system resources into two worlds
- Normal World runs the content rich OS
- Secure World runs security critical services
- The protection of resources includes processor, memory and IO devices
Many Products use ARM TrustZone

Samsung Knox

sierraware

TRUSTONIC
Smart Devices Going **Mobile**
Hardware Attacks - Cold Boot Attack
## Previous Works on Coldboot Defense

<table>
<thead>
<tr>
<th>Work</th>
<th>Conference</th>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRESOR</td>
<td>Sec</td>
<td>2011</td>
<td>Register-based RAM-less AES encryption</td>
</tr>
<tr>
<td>Copker</td>
<td>NDSS</td>
<td>2014</td>
<td>Cache-based RAM-less RSA encryption</td>
</tr>
<tr>
<td>PixelVault</td>
<td>CCS</td>
<td>2014</td>
<td>GPU based RAM-less encryption</td>
</tr>
<tr>
<td>Sentry</td>
<td>ASPLOS</td>
<td>2015</td>
<td>Cache-based RAM-less encryption</td>
</tr>
<tr>
<td>Mimosa</td>
<td>S&amp;P</td>
<td>2015</td>
<td>Transactional-based RAM-less encryption</td>
</tr>
</tbody>
</table>
Multi-vector Adversary
Introducing CaSE - Goals

✓ Defense against Multi-Vector adversary
  ✓ Physical memory disclosure attack – Cold boot
  ✓ Compromised rich OS

✓ Provide confidentiality and integrity to both the code and data of the binaries in TEE
  ✓ Confidentiality – Protects IP, secret code, sensitive data
  ✓ Integrity – Program behavior
System On Chip (SoC)

Processor Cache

Secure Memory

Secure OS

NonSecure Normal World Memory

NonSecure Rich OS

DRAM

NonSecure Cache

Secure OS

Cold Boot Attack

Software Attack
Case-Assisted Execution in Secure World

System On Chip (SoC)

Secure storage

Processor Cache

Packer

Context

NonSecure Normal World Memory

NonSecure Rich OS

Secure Memory

Secure OS

DRAM

Cold Boot Attack

Software Attack
Case-Assisted Execution in Normal World

System On Chip (SoC)

- Secure storage
- Packer
- Context
- Processor Cache
- CaSE Manager

DRAM

- NonSecure Normal World Memory
- NonSecure OS
- Secure Memory
- Secure Rich OS

Cold Boot Attack
Software Attack
Controlling the Cache

- Cache Locking is available through L2 cache lockdown CP15 coprocessor

- The granularity of locking is per cache way

- On Cortex-A8, which has 8 way total 256KB L2 unified cache
disable_local_irq();
enableCaching(memArea);
disableCaching(loaderCode);
disableCaching(loaderStack);
invalidate_cache(virtual address of memArea);
unlockWay(wayToFill);
lockWay(allWay XOR wayToFill);
while(has more to load in memArea)
    LDR r0, [memArea + i];
lockWay(wayToFill);
unlockWay(allWay XOR wayToFill);
Self Modifying Program

System On Chip (SoC)

L1 Instruction Cache

L1 Data Cache

L2 Unified Cache
Feasibility of using Cache as Memory

<table>
<thead>
<tr>
<th>Application</th>
<th>Code+Data (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>2.4</td>
</tr>
<tr>
<td>RSA</td>
<td>10</td>
</tr>
<tr>
<td>SHA1</td>
<td>5</td>
</tr>
<tr>
<td>CaSE Crypto Lib</td>
<td>17.4</td>
</tr>
<tr>
<td>Kernel Integrity Checker</td>
<td>6.6</td>
</tr>
<tr>
<td>CaSE Packer</td>
<td>2.8</td>
</tr>
<tr>
<td>Packed CaSE Crypto Lib</td>
<td>20.4</td>
</tr>
<tr>
<td>Packed Kernel Checker</td>
<td>9.5</td>
</tr>
</tbody>
</table>
Performance Impact to the Application

Table II: Kernel Integrity Checker in Normal Cache

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Preparation</td>
<td>613</td>
</tr>
<tr>
<td>Environment Integrity Check</td>
<td>1540</td>
</tr>
<tr>
<td>CaSE Unpacking</td>
<td>5973</td>
</tr>
<tr>
<td>Kernel Check</td>
<td>18676</td>
</tr>
<tr>
<td>Environment Cleanup</td>
<td>412</td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td><strong>27214</strong></td>
</tr>
</tbody>
</table>

From the time breakdown, we can see that though environment setup and cleanup consume some processor cycles, the major computation overhead originates from the unpacking process, which decrypts the encrypted CaSE application payload. The entire kernel check takes 0.02 second to complete, and the application context saving time is 94 µs.

Table III: AES Encryption in Secure Cache

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>World Switching</td>
<td>2.6</td>
</tr>
<tr>
<td>AES CBC Encrypt (1KB)</td>
<td>443</td>
</tr>
<tr>
<td>Output Synchronization (1KB)</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total Time</strong></td>
<td><strong>447.6</strong></td>
</tr>
</tbody>
</table>

2) CaSE Secure Application Performance:

Using the crypto library as a case study for the CaSE secure execution mode, we measure the benchmarks for a secure cache execution similar to the normal cache execution. Table III shows the time breakdown of a secure call to perform encryption using AES CBC mode. In the secure mode, the cache is protected against the compromised rich OS. Therefore, it is not necessary to clean up the execution environment.

3) Performance Trade-off between Execution Modes:

To find out the impact of SoC-bound execution environment on application performance, we run AES, RSA, and SHA1 in different environments and compare their performance. First, we port the application into a kernel module and load the module into the rich OS to measure the performance without any security enhancement. Second, we run the application in the two CaSE execution environments, one in the normal world and the other in the secure world. We consider that the first experiment should achieve similar performance as other kernel encryption solutions, and should serve as a good baseline for comparison. On the other hand, the CaSE execution will suffer performance penalty for the enhanced security.

The experimental results on AES algorithm are shown in Figure 5. The performance of secure executed AES is almost identical to that of generic AES. The secure AES has a small advantage over the generic kernel AES when the memory buffer to be encrypted is small. This is due to preloaded cache lines for the AES data section. For smaller size encryption requests, the normal cache execution is significantly slower than the other two methods. This is because the environment is created and destroyed for each request in order to protect the confidentiality and integrity of the execution environment. However, as the size of the plaintext increases, the difference in the encryption bandwidth diminishes. This is because the overhead to create and destroy the environment becomes insignificant.

We have also performed the same set of experiments on RSA algorithm and SHA1 algorithm. The results for RSA algorithm are shown in Figure 6. In this experiment, we measure the number of 1024-bit RSA decryptions that the system can carry out in one second. Similar to AES, the normal cache execution takes a penalty in the environment initialization and clean up. However, as the number of messages in the request becomes larger, this fixed cost can be ignored. Lastly, we also benchmark the performance of SHA1. We build up our test case by sending fixed size 512 byte packet to the SHA1 module to calculate the hash. Due to simplicity of SHA1, the normal world execution overhead is high when the number of messages per request is low. Similar to RSA and AES, the environment penalty becomes small as the number of messages increases.
Performance Impact to the System

![Graph showing the impact of L2 cache way lock on scaled performance score.]
Conclusion

✓ A secure cache-assisted SoC-bound execution framework
✓ Provide confidentiality and integrity to sensitive code and data of applications
✓ Protect against both software attacks and cold boot attack.
✓ In the future, we would like to further study efficient method to provide OS support to the TEE.