Cache Storage Channels: Alias-Driven Attacks
2016-03-30

Roberto Guanciale (KTH)
Mads Dam  Hamed Nemati  Christoph Dam

2016-03-30
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels
- precise analysis of timing/power consumption exceedingly difficult
- channels counteracted by model-external means
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels
- precise analysis of timing/power consumption exceedingly difficult
- channels counteracted by model-external means
- models should preferably be sound with respect to the features that are reflected
Common verification approach

- If models are sound (wrt the selected features)
- Verification by refinements

Abstract model

Concrete model
Common verification approach

- If the hidden features (e.g. cache state) affect the model variables (e.g. memory) ⇒ Overapproximation

Abstract model

Concrete model
Challenges of Storage channel

- If the hidden features (e.g. cache state) affect the model variables (e.g. memory) ⇒ Overapproximation
- Problems for information-flow properties
- The actual HW is deterministic
Challenges of Storage channel

- The HW respect the abstract specification is some architectural constraints are met
- What about unknown colocated SW?
MMU allows to configure (via the page tables) the caching policy on a per-page basis

A processor can use the Harvard arrangement
Incoherent behaviors

- Caches should be transparent to program behaviour.
- This is usually not the case unless the system configuration satisfies some architecture-specific constraints.
Incoherent behaviors

- Caches should be transparent to program behaviour.
- This is usually not the case unless the system configuration satisfies some architecture-specific constraints.
- E.g., memory mapped UART.

Mismatched cacheability attributes (ARM-terminology: "unexpected cache hit"): if the data cache reports a hit on a memory location that is marked as non-cacheable, the cache might access the memory disregarding such hit.

Self-modifying code; even if the executable code is updated, the processor might execute the old version of it if this has been stored in the instruction cache.
Incoherent behaviors

- caches should be transparent to program behaviour
- this is usually not the case unless the system configuration satisfies some architecture-specific constraints
- e.g. memory mapped UART
- Mismatched cacheability attributes (ARM-terminology: “unexpected cache hit”)
  if the data cache reports a hit on a memory location that is marked as non-cacheable, the cache might access the memory disregarding such hit.
Incoherent behaviors

- Caches should be transparent to program behaviour.
- This is usually not the case unless the system configuration satisfies some architecture-specific constraints.
- E.g. memory mapped UART.
- Mismatched cacheability attributes (ARM-terminology: “unexpected cache hit”).
  If the data cache reports a hit on a memory location that is marked as non-cacheable, the cache might access the memory disregarding such hit.
- Self-modifying code;
  Even if the executable code is updated, the processor might execute the old version of it if this has been stored in the instruction cache.
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)

* A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
* A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
* A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
* A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
* A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

* A1) invalidate(VA_c)
    A2) write(VA_nc, 0)
    A3) D = read(VA_c)
    A4) write(VA_nc, 1)
    A5) call victim
    A6) D = read(VA_c)

V1) if secret
    access(VA3)
    else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
* A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) \( D = \text{read}(VA_c) \)
A4) write(VA_nc, 1)
A5) call victim
A6) \( D = \text{read}(VA_c) \)

* V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
* A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
* A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
* A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
* A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
* A5) call victim
A6) D = read(VA_c)

V1) if secret
   access(VA3)
else
   access(VA4)
Attacking confidentiality using data-caches

A1) invalidate(VA_c)
A2) write(VA_nc, 0)
A3) D = read(VA_c)
A4) write(VA_nc, 1)
A5) call victim
A6) D = read(VA_c)

V1) if secret
    access(VA3)
else
    access(VA4)
V1) $D = \text{access}(VA_c)$
   ...
A1) $\text{write}(VA_{nc}, 1)$
   ...
V2) $D = \text{access}(VA_c)$
V3) if not policy($D$)
    reject
   ...
    [evict $VA_c$]
   ...
V4) $\text{use}(VA_c)$
Attacking integrity using data-caches (TOCTTOU)

V1) D = access(VA_c)
   ...
A1) write(VA_nc, 1)
   ...
V2) D = access(VA_c)
V3) if not policy(D)
    reject
   ...
   [evict VA_c]
   ...
V4) use(VA_c)
Attacking integrity using data-caches (TOCTTOU)

V1) D = access(VA_c)
    ...
* A1) write(VA_nc, 1)
    ...
V2) D = access(VA_c)
V3) if not policy(D)
    reject
    ...
    [evict VA_c]
    ...
V4) use(VA_c)
V1) $D = \text{access}(VA_c)$

...  
A1) $\text{write}(VA_{nc}, 1)$

* ...  
V2) $D = \text{access}(VA_c)$
V3) if not policy($D$)  
  reject

...  
  [evict $VA_c$]

...  
V4) $\text{use}(VA_c)$

![Diagram of memory access and eviction process]
V1) $D = \text{access}(VA_c)$

\[ \ldots \]

A1) $\text{write}(VA_{nc}, 1)$

\[ \ldots \]

V2) $D = \text{access}(VA_c)$

V3) if not policy($D$)

\[ \text{reject} \]

\[ \ldots \]

* evict $VA_c$

\[ \ldots \]

V4) $\text{use}(VA_c)$
V1) D = access(VA_c)

... 
A1) write(VA_nc, 1)

... 
V2) D = access(VA_c)

V3) if not policy(D)
    reject

... 
[evict VA_c]

...  
* V4) use(VA_c)
Attacking Confidentiality Using Instruction Caches

* A1) jmp A8  
A2) write(&A8, {R0=1})  
A3) call victim  
A4) jmp A8  
A5) D = R0  
 ...  
A8) R0=0  
A9) return

V1) if secret  
    jmp f1  
else  
    jmp f2
Case study: AES

- Platform: Raspberry PI 2
- Victim: AES service in TrustZone
- Attacker: Non-secure OS
- AES uses precomputes SBoxes (1KB=16 lines)

\[ c[j] = K_n[j] \oplus T_4[s_{n-1}[j]] \]

- 128-bit AES key extracted after 850 encryptions
Case study: Violating Spatial Isolation in a Hypervisor

- Platform: Beagleboard MX
- Victim: paravirtualizing hypervisor using direct paging
- Attacker: Non-secure guest
- Page table stored in the guest memory
- PTs created by the guest / made read only / validated / activated
- Validation of stale data/activation of non-valid page table
- Attacker takes complete control of the system
Case study: Extraction of exponent from a modular exponentiation procedure

- Platform: Raspberry PI 2
- Victim: modular exponentiation procedure in Trustzone
- Attacker: Non-secure guest
- Non pc-secure procedure
- Invocation of function depends on the secret exponent
- Attacker identifies the execution path
Countermeasures

- For the side-channels, standard timing approaches
  - pc-secure code, secret independent memory accesses, ...
- For integrity: guarantee coherency of accessed memory
  - cache-flushes, explicit eviction of cache-lines, ...
- Specific for the new attacks
  - access the cache lines independently on secret
  - avoid uncacheable aliases for a region of memory
## Countermeasures: hypervisor

<table>
<thead>
<tr>
<th>LMbench</th>
<th>Native</th>
<th>Hyp</th>
<th>ACPT</th>
<th>SelFl</th>
<th>Flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>0.84</td>
<td>2.19</td>
<td>2.20</td>
<td>2.20</td>
<td>2.38</td>
</tr>
<tr>
<td>fork+execve</td>
<td>2068</td>
<td>5249</td>
<td>5248</td>
<td>6285</td>
<td>39029</td>
</tr>
<tr>
<td>pagefaulst</td>
<td>3.76</td>
<td>11.21</td>
<td>11.12</td>
<td>21.55</td>
<td>332.82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application benchmark</th>
<th>Native</th>
<th>Hyp</th>
<th>ACPT</th>
<th>SelFl</th>
<th>Flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>tar (500K)</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>190</td>
</tr>
<tr>
<td>tar (2M)</td>
<td>230</td>
<td>210</td>
<td>200</td>
<td>210</td>
<td>370</td>
</tr>
<tr>
<td>dd (10M)</td>
<td>90</td>
<td>140</td>
<td>140</td>
<td>160</td>
<td>990</td>
</tr>
<tr>
<td>dd (40M)</td>
<td>330</td>
<td>500</td>
<td>450</td>
<td>600</td>
<td>3830</td>
</tr>
<tr>
<td>jpg2gif(5KB)</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>130</td>
</tr>
<tr>
<td>jpg2bmp(5KB)</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>110</td>
</tr>
<tr>
<td>jpegtrans(270’, 5KB)</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>bmp2tiff(90 KB)</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>tif2rgb(200 KB)</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>120</td>
</tr>
<tr>
<td>sox(aif2wav 100KB)</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>140</td>
</tr>
</tbody>
</table>
## Countermeasure: AES

<table>
<thead>
<tr>
<th>AES encryption</th>
<th>5 000 000 × 16B</th>
<th></th>
<th>10 000 × 8KB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Throughput</td>
<td>Time</td>
<td>Throughput</td>
</tr>
<tr>
<td>Original SBoxes</td>
<td>23s</td>
<td>3.317 MB/s</td>
<td>13s</td>
<td>6.010 MB/s</td>
</tr>
<tr>
<td>Compact Last SBox</td>
<td>24s</td>
<td>3.179 MB/s</td>
<td>16s</td>
<td>4.883 MB/s</td>
</tr>
<tr>
<td>Scrambled Last SBox</td>
<td>30s</td>
<td>2.543 MB/s</td>
<td>20s</td>
<td>3.901 MB/s</td>
</tr>
<tr>
<td>Uncached Last SBox</td>
<td>36s</td>
<td>2.119 MB/s</td>
<td>26s</td>
<td>3.005 MB/s</td>
</tr>
<tr>
<td>Scrambled All SBoxes</td>
<td>132s</td>
<td>0.578 MB/s</td>
<td>125s</td>
<td>0.625 MB/s</td>
</tr>
<tr>
<td>Uncached All SBoxes</td>
<td>152s</td>
<td>0.502 MB/s</td>
<td>145s</td>
<td>0.539 MB/s</td>
</tr>
</tbody>
</table>
Repairing the Integrity Verification

- critical resources cannot be directly (or indirectly) affected by untrusted SW
- for these resources the actual system must behave according to the formal abstract specification
- main verification condition: for every address that belongs to the critical resources, if there is a cache hit and the corresponding cache line differs from the main memory then the cache line must be dirty
Repairing the Confidentiality Verification

- no side channel is present due to caches
- the attacker can observe all the resources that can affect the eviction
  - cache line tag
  - cache line emptiness
- goal: after the execution of an arbitrary functionality these resources do not depend on confidential data
Repairing the Confidentiality Verification

- two executions of a program, starting from equivalent non-confidential resources
- architectural property: equivalence of line emptiness and tag is preserved by cache safe operations
  - same type of memory access
  - equivalence of the tag and index of the PC
  - equivalence of the tag and index of the address accessed
- verification condition is a relational observation equivalence
- we use existing tools for relational verification that support trace based observations
Relational verification

start

MEM[R1] = R2

if R1=0

R2 = MEM[R3]  R2 = MEM[R3+4]
Relational verification

\[ \text{mem}(a) = \text{mem}'(a) \]

\[ R0 = R0' \]

\[ R0 + R1 = R0' + R1' \]
Relational verification

\[ P \Rightarrow \text{tag}(R1) = \text{tag}(R1') \]

\[ P \Rightarrow \text{idx}(R1) = \text{idx}(R1') \]
Relational verification

\[
P \land (R_1 = 0) \land (R_1' = 0) \Rightarrow \text{tag}(R_3) = \text{tag}(R_3')
\]

\[
P \land (R_1 = 0) \land (R_1' = 0) \Rightarrow \text{idx}(R_3) = \text{idx}(R_3')
\]
Relational verification

\[ P \land (R_1 = 0) \land (R_1' \neq 0) \Rightarrow \text{tag}(R_3) = \text{tag}(R_3' + 4) \]

\[ P \land (R_1 = 0) \land (R_1' \neq 0) \Rightarrow \text{idx}(R_3) = \text{idx}(R_3' + 4) \]
Relational verification

- Symbolic execution and weakest precondition propagation
- SMT solver to check conditions and early prune of unreachable path-pairs
- Force same memory operations
- For instruction cache observations are tag and index of the program counter
Ongoing work

- Repair of formal verification
- TLBs / Branch prediction / …
- Experimentation in multi-core
- Experimentation using GPU
- Evaluating HW countermeasures
Thank you