FLUSH+RELOAD a High Resolution, Low Noise, L3 Cache Side-channel Attack

Yuval Yarom  Katrina Falkner
Memory Sharing

• Techniques for reducing the overall memory footprint of the system.
  – Shared text segments
  – Shared libraries
  – Memory de-duplication

• Considered safe, i.e. equivalent to no sharing
Outline

• Cache Architecture and the FLUSH+RELOAD attack
• RSA and Square-and-multiply exponentiation
• Attacking the GnuPG implementation of RSA
The X86 Cache

• Memory is slower than the processor
• The cache utilises locality to bridge the gap
  – Divides memory into *lines*
  – Stores recently used lines
• Shared caches improve performance for multi-core processors
Cache Consistency

• Memory and cache can be in inconsistent states
  – Rare, but possible

• Solution: Flushing the cache contents
  – Ensures that the next load is served from the memory
The FLUSH+RELOAD Technique

• Exploits cache behaviour to leak information on victim access to shared memory.
• Spy monitors victim’s access to shared code
  – Spy can determine what victim does
  – Spy can infer the data the victim operates on
**FLUSH+RELOAD**

- **FLUSH** memory line
- Wait a bit
- Measure time to **RELOAD** line
  - slow-> no access
  - fast-> access
- Repeat
RSA

• RSA is a public key cryptographic scheme
• The main operation is modular exponentiation, i.e. calculating
  \[ b^e \mod n \]
• The exponent \( e \) used for decryption and for signing is secret
Square-and-Multiply Exponentiation

- Scans the exponent from the MSB to the LSB
- For clear bits does **Square-Reduce**
- For set bits does **Square-Reduce-Multiply-Reduce**
- The sequence of operations reveals the (secret) exponent

\[
\begin{align*}
x & \leftarrow 1 \\
\text{for } i & \leftarrow |e| - 1 \ \text{downto } 0 \ \text{do} \\
& x \leftarrow x^2 \ \text{mod } n \\
& \text{if } (e_i = 1) \ \text{then} \\
& \quad x = xb \ \text{mod } n \\
& \quad \text{endif} \\
\text{done} \\
\text{return } x
\end{align*}
\]
Attacking GnuPG

• Achieve sharing of the victim code
• Use **FLUSH+RELOAD** to recover the sequence of operations of the modular exponentiation
• Divide time into slots of 2048 cycles (about 0.6μs)
• In each slot, probe a memory line in the code of the **Square**, **Multiply** and **Modulo-reduce** functions
A Sample Trace

![Graph showing sample trace]

- **Probe Time (cycles)**
- **Time Slot Number**

- **Threshold**

- **Square**
- **Multiply**
- **Modulo**

- **Symbols**:
  - Red circles: Square
  - Green triangles: Multiply
  - Blue circles: Modulo
Results
Applications

• Attacking the default OpenSSL implementations of ECDSA
• Synchronous cross-VM final round attack on AES
• Trace the use of vi
• Potential: keystroke timing, network use statistics...
Lessons

• It is hard to limit the extent of sharing. E.g. “read-only” is more than read only.

• Use constant-time implementations of cryptographic primitives.

• Apply the principle of least privilege