CACHE-TIMING ATTACKS ON RSA KEY GENERATION
Conference on Cryptographic Hardware and Embedded Systems (CHES) 2019

Alejandro Cabrera Aldaya¹, Cesar Pereida García²,
Luis Manuel Alvarez Tapia¹, Billy Bob Brumley²,
{aldaya, lalvarezt89}@gmail.com, {billy.brumley, cesar.pereidagarcia}@tuni.fi

¹ Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba
² Network and Information Security Group (NISEC), Tampere University, Tampere, Finland

Aug 25-28, 2019
Contents

Introduction

Side-Channel Leakage Finding
  The BN_FLG_CONSTTIME
  A New Methodology
  The Tool

Leakage Analysis
  RSA Key Generation
  Binary GCD

The Attack

Conclusion
  Lessons Learned
Contents

Introduction

Side-Channel Leakage Finding
   The BN_FLG_CONSTTIME
   A New Methodology
   The Tool

Leakage Analysis
   RSA Key Generation
   Binary GCD

The Attack

Conclusion
   Lessons Learned
Introduction

- **What?**: A single trace cache-timing attack against the binary Extended Euclidean (GCD) algorithm used during RSA key generation, leading to complete RSA private key recovery.

- **Why?**: Because we can!
  - Cloud services (e.g. AWS, Azure) and automated certificate renewal (e.g. Let’s Encrypt) make RSA key generation a semi-predictable operation.
  - Micro-architecture attacks.
  - RSA key generation neglected.

- **How?**: We developed a new methodology to help us detect insecure code paths in OpenSSL, then we combine FLUSH+RELOAD, signal processing and lattice techniques.
Contents

Introduction

Side-Channel Leakage Finding
  The BN_FLG_CONSTTIME
  A New Methodology
  The Tool

Leakage Analysis
  RSA Key Generation
  Binary GCD

The Attack

Conclusion
  Lessons Learned
OpenSSL and the BN_FLG_CONSTTIME

- OpenSSL relies on the **BN_FLG_CONSTTIME** to protect against timing-attacks.
- The flag gives a lot of room for mistakes.
- Several flaws involving the flag have been identified previously.
  - CVE-2016-2178
  - CVE-2016-7056
  - CVE-2018-0734
- We have a record of well known side-channel vulnerable functions used in OpenSSL.
A New Methodology

▶ Create a list of known side-channel vulnerable functions in a library (e.g. OpenSSL).
▶ Use a debugger to automatically set breakpoints at lines of code that should be unreachable.
▶ Run several security-critical commands.
▶ Generate a report if any of the breakpoints is reached.
▶ Investigate manually the root-cause.
The Tool\(^1\)

INFO: Parsing source code at: ./openssl-1.0.2k
...
INFO: Breakpoints file generated: triggers.gdb
...
INFO: Monitor target command line
TOOL: gdb --batch --command=triggers.gdb --args
  openssl-1.0.2k/apps.openssl genpkey -algorithm RSA
  -out private_key.pem -pkeyopt rsa_keygen_bits:2048
...
INFO: Setting breakpoints...
Breakpoint 1 at ...: file bn_exp.c, line 418.
Breakpoint 2 at ...: file bn_gcd.c, line 120.
Breakpoint 3 at ...: file bn_gcd.c, line 238.
...
INFO: Insecure code executed!
Breakpoint 1, BN_mod_exp_mont (...) at bn_exp.c:418
418  bn_check_top(a);
#0  BN_mod_exp_mont (...) at bn_exp.c:418
#1  ... in witness (...) at bn_prime.c:356
#2  ... in BN_is_prime_fasttest_ex (...) at bn_prime.c:329
#3  ... in BN_generate_prime_ex (...) at bn_prime.c:199
#4  ... in rsa_builtin_keygen (...) at rsa_gen.c:150
...
INFO: Insecure code executed!
Breakpoint 3, BN_mod_inverse (...) at bn_gcd.c:238
238  bn_check_top(a);
#0  BN_mod_inverse (...) at bn_gcd.c:238
#1  ... in BN_MONT_CTX_set (...) at bn_mont.c:450
#2  ... in BN_is_prime_fasttest_ex (...) at bn_prime.c:319
#3  ... in BN_generate_prime_ex (...) at bn_prime.c:199
#4  ... in rsa_builtin_keygen (...) at rsa_gen.c:171
...

\(^1\) [Gri+19] expanded our methodology and tooling into a CI tool called TriggerFlow.
https://gitlab.com/nisec/triggerflow
Contents

Introduction

Side-Channel Leakage Finding
  The BN_FLG_CONSTTIME
  A New Methodology
  The Tool

Leakage Analysis
  RSA Key Generation
  Binary GCD

The Attack

Conclusion
  Lessons Learned
Algorithm 1: OpenSSL RSA key generation

Input: Key size $n$ and public exponent $e$.
Output: Public and private key pair.

begin
\begin{align*}
  & \textbf{while} \gcd(p - 1, e) \neq 1 \textbf{ do} \\
  & \quad p \leftarrow \text{rand } n/2\text{-bit prime} \quad /* \text{ Generate } p */ \\
  & \textbf{while} \gcd(q - 1, e) \neq 1 \textbf{ do} \\
  & \quad q \leftarrow \text{rand } n/2\text{-bit prime} \quad /* \text{ Generate } q */ \\
  & d \leftarrow e^{-1} \mod (p - 1)(q - 1) \quad /* \text{ Priv exp } */ \\
  & dp \leftarrow d \mod (p - 1) \\
  & dq \leftarrow d \mod (q - 1) \\
  & iq \leftarrow q^{-1} \mod p \\
  & \textbf{return} (N, e), (d, p, q, dp, dq, iq)
\end{align*}
end
**Algorithm 2: Binary GCD**

**Input:** Integers $a$ and $b$ such that $0 < a < b$.

**Output:** Greatest common divisor of $a$ and $b$.

```
begin
  u ← a, v ← b, i ← 0
  while even(u) and even(v) do
    u ← u/2, v ← v/2, i ← i + 1
  while u ≠ 0 do
    while even(u) do
      u ← u/2 /* u-loop */
    while even(v) do
      v ← v/2 /* v-loop */
    if u ≥ v then
      u ← u − v /* sub-step */
    else
      v ← v − u
  return v · 2^i
```
Contents

Introduction

Side-Channel Leakage Finding
  The BN_FLG_CONSTTIME
  A New Methodology
  The Tool

Leakage Analysis
  RSA Key Generation
  Binary GCD

The Attack

Conclusion
  Lessons Learned
Memory Hierarchy

Computer Memory Hierarchy

- **Processor registers**
  - Very fast, very expensive
  - Power on

- **Processor cache**
  - Very fast, very expensive
  - Immediate term

- **Random access memory**
  - Fast, affordable
  - Power on
  - Very short term

- **Flash/USB memory**
  - Slower, cheap
  - Power off
  - Short term

- **Hard drives**
  - Slow, very cheap
  - Power off
  - Mid term

- **Tape backup**
  - Very slow, affordable
  - Power off
  - Long term

- **Large size - very large capacity**
  - Large size - very large capacity

- **Medium size - medium capacity**
  - Medium size - medium capacity

- **Small size - small capacity**
  - Small size - small capacity
FLUSH+RELOAD and Performance Degradation

1) Victim executes its own process, filling the cache

2) Attacker flushes victim’s data from the cache and waits

3) Victim may or may not execute its own process again while the attacker waits

4) Attacker reloads the data and measures the loading time

5) Attacker traces the victim’s process execution and infers information about the victim
Attack Scenario

Core 0
L1
L1
L2
Victim

Core 1
L1
L1
L2
Attacker
Flush+Reload

Core 2
L1
L1
L2
Attacker
Perf. Degradation

Core 3
L1
L1
L2

OpenSSL
Shared LLC
The Attack 1/2

- OpenSSL 1.0.2k.
- **Flush+Reload** [YF14].
- Templating.
- Pearson correlation.
- Low-pass filter.
- Horizontal analysis.
- Sequence of ops.
The Attack 2/2

- Obtain a ranked list of partial prime factors.
- Formulate lattice problems for the candidates.
- Run in a cluster for 4 hours.
- Recover private keys with a 27% success rate.
Contents

Introduction

Side-Channel Leakage Finding
  The BN_FLG_CONSTTIME
  A New Methodology
  The Tool

Leakage Analysis
  RSA Key Generation
  Binary GCD

The Attack

Conclusion
  Lessons Learned
We developed a simple methodology and tool to track existing flaws leading to insecure code paths in crypto libraries.

We discovered three new flaws affecting OpenSSL during RSA key generation.

We performed a cache-timing attack on the GCD algorithm, allowing us to fully recover RSA keys with a success rate of 27%.

Our general strategy was:

- **FLUSH+RELOAD** and performance degradation.
- Signal processing.
- Error correction algorithm.
- Lattice problem solving.
Responsible Disclosure

We reported our findings to OpenSSL security team, and they confirmed affected versions\(^2\) 1.1.0-1.1.0h and 1.0.2-1.0.2o.

OpenSSL assigned CVE-2018-0737 based on our work and adopted the proposed patches.

- **Lesson 1: Secure by default.** These and similar flaws can be prevented with a secure-by-default approach.
  - Adopt constant-time algorithms by default, e.g. [BY19]

- **Lesson 2: Knowledge transfer.** The engineers and cryptographers must work side-by-side to ensure that academic results permeate over real-world products.

\(^2\)OpenSSL 1.1.1 did not exist at the time of disclosure.
Thank you for listening.

Questions?