Tampere University

CACHE-TIMING ATTACKS ON RSA KEY GENERATION

Conference on Cryptographic Hardware and Embedded Systems (CHES) 2019

Alejandro Cabrera Aldaya[1], Cesar Pereida García[2],
Luis Manuel Alvarez Tapia[1], Billy Bob Brumley[2],

{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

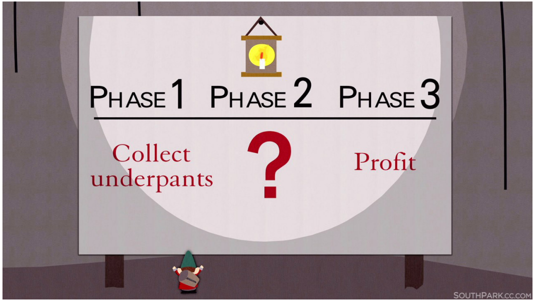
Conclusion
- Lessons Learned

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 2, BN_gcd (...) at bn_gcd.c:120
120        int ret = 0;
#0    BN_gcd (...) at bn_gcd.c:120
#1    ... in rsa_builtin_keygen (...) at rsa_gen.c:154
...
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

RSA Key Generation

Algorithm 1: OpenSSL RSA key generation

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

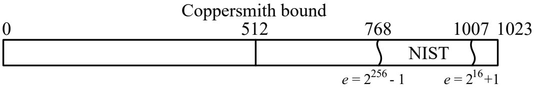
begin
  while $\gcd(p-1, e) \neq 1$ **do**
    $p \leftarrow$ rand $n/2$-bit prime `/* Generate p */`
  while $\gcd(q-1, e) \neq 1$ **do**
    $q \leftarrow$ rand $n/2$-bit prime `/* Generate q */`
  $d \leftarrow e^{-1} \bmod (p-1)(q-1)$ `/* Priv exp */`
  $dp \leftarrow d \bmod (p-1)$
  $dq \leftarrow d \bmod (q-1)$ `/* CRT parameters */`
  $iq \leftarrow q^{-1} \bmod p$
  return *(N, e), (d, p, q, dp, dq, iq)*

Binary GCD

Algorithm 2: Binary GCD

Input: Integers a and b such that $0 < a < b$.
Output: Greatest common divisor of a and b.

begin
    $u \leftarrow a, v \leftarrow b, i \leftarrow 0$
    while even(u) **and** even(v) **do**
        $u \leftarrow u/2, v \leftarrow v/2, i \leftarrow i + 1$
    while $u \neq 0$ **do**
        while even(u) **do**
            $u \leftarrow u/2$ `/* u-loop */`
        while even(v) **do**
            $v \leftarrow v/2$ `/* v-loop */`
        if $u \geq v$ **then**
            $u \leftarrow u - v$ `/* sub-step */`
        else
            $v \leftarrow v - u$
    return $v \cdot 2^i$



Contents

Introduction

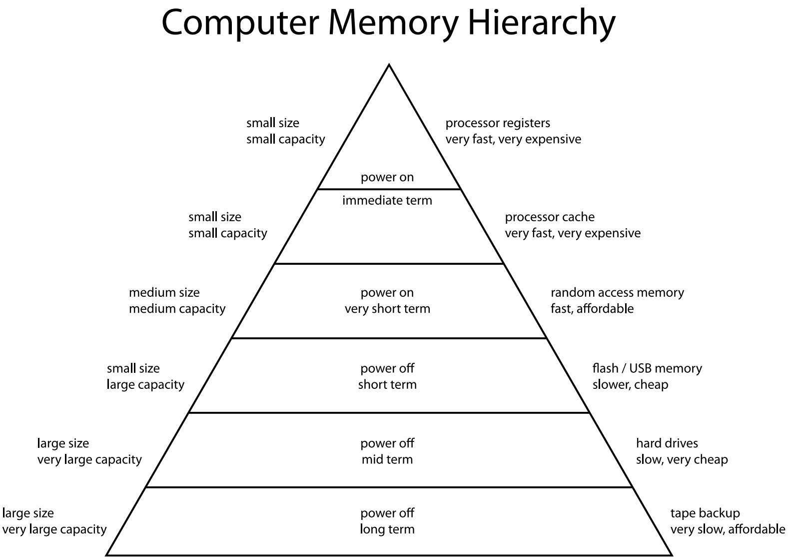
Side-Channel Leakage Finding
- The `BN_FLG_CONSTTIME`
- A New Methodology
- The Tool

Leakage Analysis
- RSA Key Generation
- Binary GCD

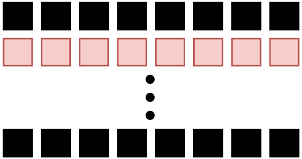
The Attack

Conclusion
- Lessons Learned

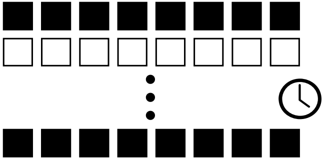
Memory Hierarchy



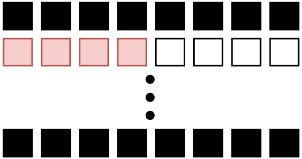
Flush+Reload and Performance Degradation



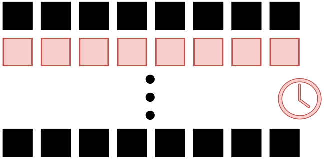
1) Victim executes its own process, filling the cache



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



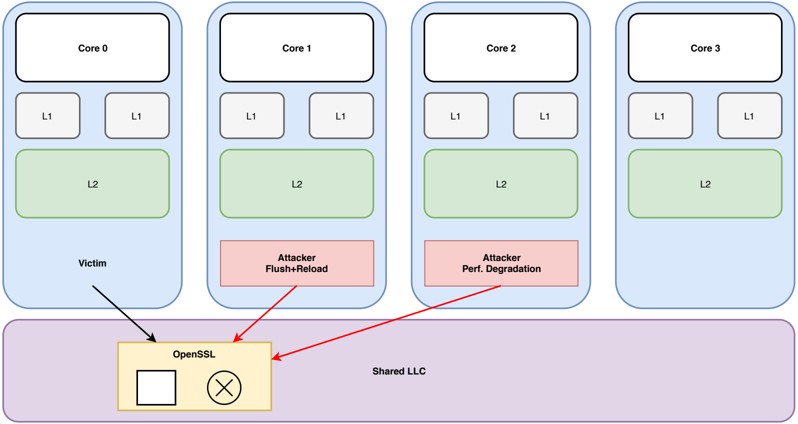
3) Victim may or may not execute its own process again while the attacker waits



4) Attacker reloads the data and measures the loading time

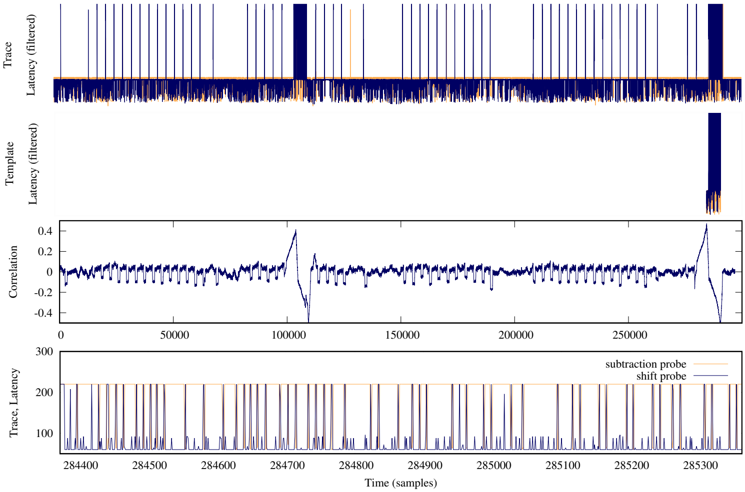
5) Attacker traces the victim's process execution and infers information about the victim

Attack Scenario



The Attack 1/2

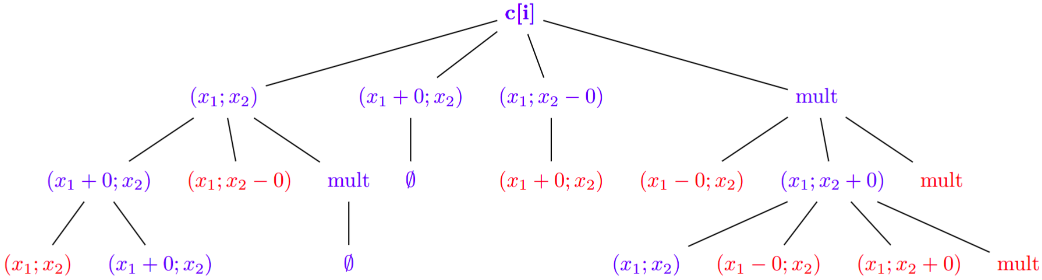
- OpenSSL 1.0.2k.
- FLUSH+RELOAD [YF14].
- Templating.
- Pearson correlation.
- Low-pass filter.
- Horizontal analysis.
- Sequence of ops.



L L L S L L S L S L L L L L S L . . . L S

The Attack 2/2

- Use expand-and-prune [HMM10] error correction algorithm.
- 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

TL;DNL

- 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?