Tampere University

TRIGGERFLOW

Regression Testing by Advanced Execution Path Inspection

20 June 2019

Iaroslav Gridin Cesar Pereida García Nicola Tuveri Billy Bob Brumley
Tampere University, Tampere, Finland

Outline

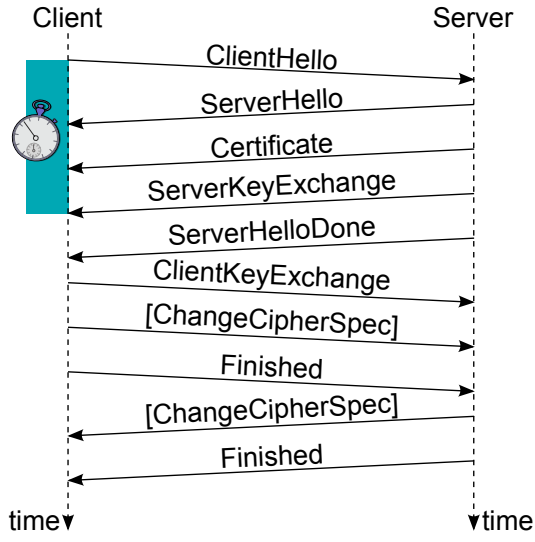
- Analysis subject: OpenSSL
- Triggerflow
- Finding bugs using it

OpenSSL

- Industry standard cryptographic library
- Has timing leak problems
- Countermeasures in place but not perfect

Timing leaks

- Some cryptographic operations execution time heavily depends on arguments
- Can guess the ballpark of a number by watching the time
- Critical operations are padded to counteract this
- CT and non-CT function variants, for security or speed



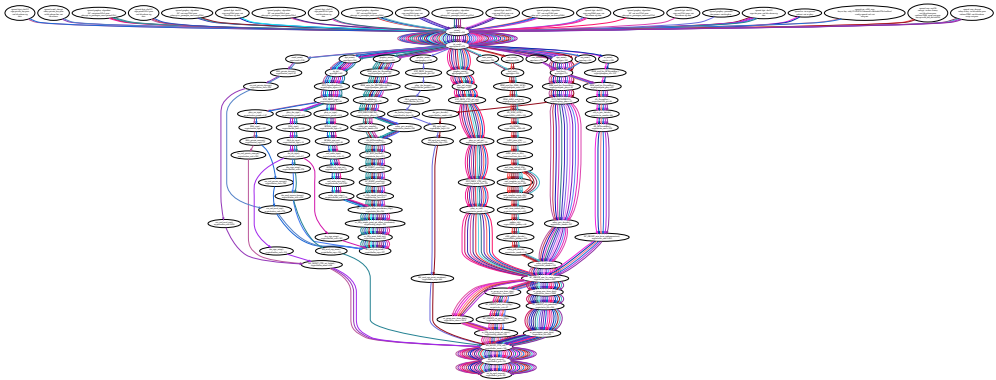
Timing leak countermeasure

- A flag (true/false variable) set on big numbers to ensure they are handled in constant time
- Requires both proper setting and proper handling
- Several bugs have been discovered in its usage:
  - CVE-2016-2178 (flag not propagated)
  - CVE-2016-7056 (flag not set)
  - CVE-2018-0737 (flag not set/set wrongly/not checked)

```
struct bignum_st {
    BN_ULONG *d;    /* Pointer to an array of 'BN_BITS2' bit chunks. */
    int top;        /* Index of last used d +1. */
    /* The next are internal book keeping for bn_expand. */
    int dmax;       /* Size of the d array. */
    int neg;        /* one if the number is negative */
    int flags;      /* # define BN_FLG_CONSTTIME 0x04 */
```

Tracking down non-constant time operations on sensitive data

- Set up a breakpoint in debugger and run the program
- Prone to false positives
- This is the full graph of code paths leading up to certain non-CT codepaths in OpenSSL



Triggerflow demo

- Now, live application of Triggerflow will be demonstrated

Triggerflow

- Runs specified commands while watching execution paths
- Written in Ruby, uses GDB for execution tracking
- Works with any GDB-supported language
- Not restricted to OpenSSL or constant-time analysis, could be used to detect execution of any kind of interesting code
- Open source, MIT license
- `https://gitlab.com/nisec/triggerflow`

```
## DSA: generate parameters (not secret)
exec openssl genpkey -genparam -algorithm DSA -out dsa.params <...>
## DSA: generate private key
debug openssl genpkey -paramfile dsa.params -out dsa.pkey
exec cat dsa.params dsa.pkey > dsa.pem
## DSA: sign
debug openssl dgst -sha512 -sign dsa.pem -out lsb-release.sig data
```

Triggerflow rules

- Allows advanced rules like ignoring codepaths

```
/* code before */
if(a % 2 == 0) //  TRIGGERFLOW_POI
/* code after */
```

```
if(something) {
    a = publickey; // TRIGGERFLOW_IGNORE_GROUP ec_publickey
}
call_suspicious_code(a) // TRIGGERFLOW_IGNORE_GROUP ec_publickey
```

```
/* code before */
call_suspicious_code(a) // TRIGGERFLOW_POI_IF a.private()
/* code after */
```

```
int call_suspicious_code(int a) {
    // TRIGGERFLOW_POI
    /* something interesting with a */
}
call_suspicious_code(public_key) // TRIGGERFLOW_IGNORE
```

Triggerflow result example

- With TF markup, we can restrict output by ignoring false positive paths

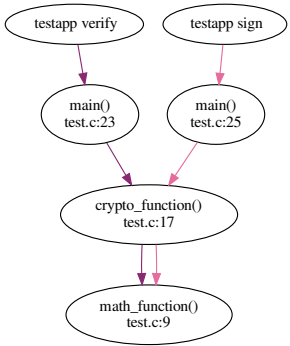


Figure: Detected flows without ignoring false positives

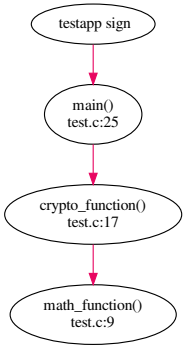
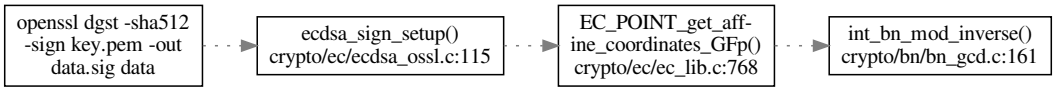


Figure: Only interesting flows

Bug found: projective to affine

- Placing an annotation in int_bn_mod_inverse uncovered an unsafe execution path from ECDSA
- Function turned out to be constant-time flag unaware
- Replaced with CT-safe code along with some refactoring
- `https://github.com/openssl/openssl/pull/8254`



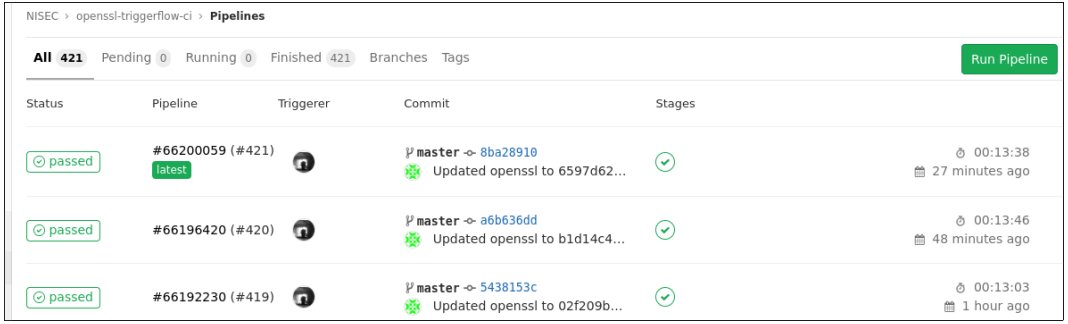
Bug found: BN_CTX retaining flag

- Interestingly, previous issue was introduced by seemingly unrelated commit
- Sometimes BN_CTX, a persistent storage for BNs, passed flag to a new BN
- 15yo defect resulted in obscuring SC issues
- Fixed by explicitly resetting constant time flag
- `https://github.com/openssl/openssl/pull/8253`

```
@@ -227,6 +227,8 @@ BIGNUM *BN_CTX_get(BN_CTX *ctx)
     }
     /* OK, make sure the returned bignum is "zero" */
     BN_zero(ret);
+    /* clear BN_FLG_CONSTTIME if leaked from previous frames */
+    ret->flags &= (~BN_FLG_CONSTTIME);
     ctx->used++;
     CTXDBG_RET(ctx, ret);
     return ret;
```

CI

- We have established a continuous integration system, watching active OpenSSL branches for some known vulnerable operations
- `https://gitlab.com/nisec/openssl-triggerflow-ci`



Scope

- Dynamic analysis
- Source code required
- Scanning for known problems

Thank you!

Links:

- `https://gitlab.com/nisec/triggerflow` tool
- `https://gitlab.com/nisec/openssl-triggerflow-ci` OpenSSL CI

Figure: Triggerflow software package

Figure: NISEC twitter