Assessing and Exploiting BigNum Vulnerabilities

Ralf-Philipp Weinmann
Director of Research - Comsecuris
<ralf@comsecuris.com>

PGP fingerprint: D244D6F2E79B529BF5548F39B27967D58C07C5B7
twitter: @esizkur

Comsecuris

PARENTAL ADVISORY

Sparse class of bugs

Outline

- Motivation, introduction to BigNum libraries
- Historical bugs in libgcrypt, GMP and OpenSSL's BN
- CVE-2014-3570: a case study
- A common bug pattern
- Property-based bug hunting
- Using verification tools to find bugs
- Conclusions

Motivation: break crypto, maybe?

- Bug Attack paper (Biham, Carmeli, Shamir) [2008]:
  - Related to fault attacks, but input triggers faulty computation
  - Hypothetical bugs presented
- Work the other way round: investigate what can be done with bugs that ***have*** occurred [and are patched]
- **`BN_sqr()`** bug in OpenSSL (patched in January 2015) was trigger for research
- Bug attacks only investigated for leakage of private keys
  - what about signature verification bypasses?
    [real problem for DSA if modular inverse routine ever returns zero]

Introduction to BigNum Arithmetic

- BigNum implementation: fundamental ingredient for real-world asymmetric crypto
- Provide arithmetic (and other) operations on integers bigger than single machine word: e.g. +, -, *, /, a^b, gcd
- For crypto: Above operations modulo n
- Sometimes:
  - specialised implementations, e.g. for BigNums of fixed length (1024/2048 bits)
  - assembly implementations
  - constant-time implementations

Widely used implementations*

* for cryptographic primitives

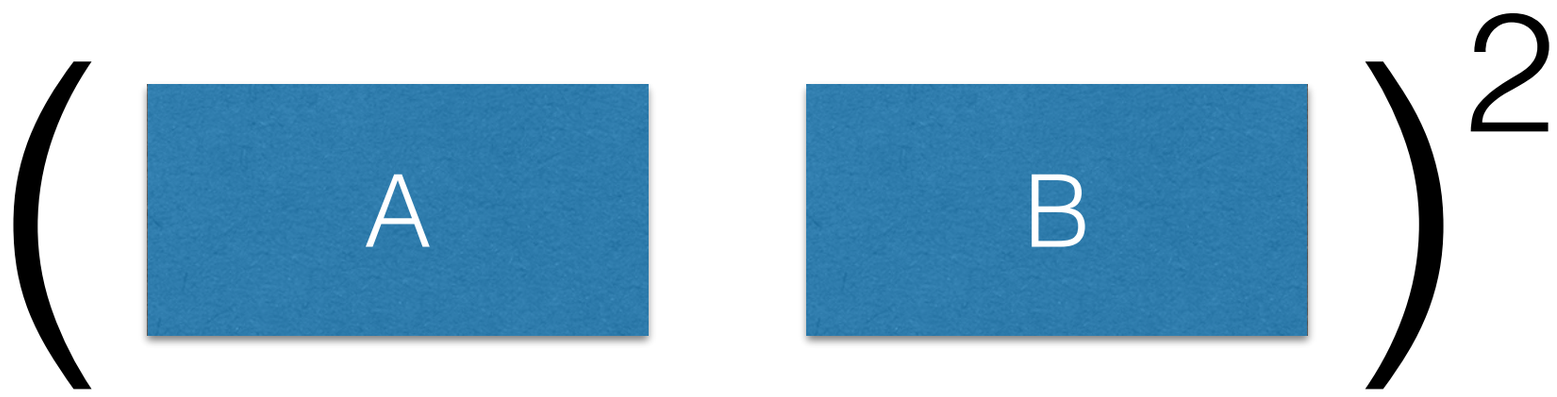
- Open source:
  - OpenSSL's BN
  - libgcrypt (fork of GMP, used by GnuPG and GnuTLS)
  - GMP (through bindings in scripting languages)
  - libTomMath, used by libTomCrypt
    - dropbear, miniTLS (embedded devices), wpa_supplicant
  - in mbedTLS (name of PolarSSL after ARM bought it)
  - java.math.BigInteger (Java)

- Closed source:
  - on Microsoft OSes: **`bcryptprimitives.dll`**
  - on OS X: **`libcorecrypto.dylib`**
  - embedded devices: many others

GMP

- Ruby Bignum
- Python
  - PyCrypto (approx. 40k downloads/daily)
  - also popular for hand-rolled crypto: gmpy
- Haskell's: Integer type is BigNum [integer-gmp]
- Ocaml: ZArith package (Module Z)

Anatomy of CVE-2014-3570



Recapitulating high-school math:

$$(2^n A + B)^2 = (2^n A)^2 + 2^{n+1} AB + B^2$$

Anatomy of CVE-2014-3570

```
/* c+=2*a*b for three word number c=(c2,c1,c0) */


#define mul_add_c2(a,b,c0,c1,c2) {      \
        BN_ULONG ta=(a),tb=(b),t0;      \
        t1 = BN_UMULT_HIGH(ta,tb);      \
        t0 = ta * tb;                   \
        t2 = t1+t1; c2 += (t2<t1)?1:0;  \
        t1 = t0+t0; t2 += (t1<t0)?1:0;  \
        c0 += t1; t2 += (c0<t1)?1:0;    \
        c1 += t2; c2 += (c1<t2)?1:0;    \
        }
```

Anatomy of CVE-2014-3570

```
/* c+=2*a*b for three word number c=(c2,c1,c0) */


#define mul_add_c2(a,b,c0,c1,c2) {      \
        BN_ULONG ta=(a),tb=(b),t0;      \
        t1 = BN_UMULT_HIGH(ta,tb);      \
        t0 = ta * tb;                   \
        t2 = t1+t1; c2 += (t2<t1)?1:0;  \
        t1 = t0+t0; t2 += (t1<t0)?1:0;  \
        c0 += t1; t2 += (c0<t1)?1:0;    \
        c1 += t2; c2 += (c1<t2)?1:0;    \
        }
```

CVE-2014-3570 summary

- Integer overflow or carry mispropagation bug, depending on your view
- Was present in OpenSSL codebase for 10 years
- Same mistake in MIPS and x86_64 assembly implementations
- Trigger probability of 2^{-64} for MIPS and 2^{-128} for x86_64

OpenSSL's impact assessment (1/2)

- "The probability of BN_sqr producing an incorrect result at random is very low: 1/2^64 on the single affected 32-bit platform (MIPS) and 1/2^128 on affected 64-bit platforms."
- "On most platforms, RSA follows a different code path and RSA operations are not affected at all. For the remaining platforms (e.g. OpenSSL built without assembly support), pre-existing countermeasures thwart bug attacks."

OpenSSL's impact assessment (2/2)

- "Static ECDH is theoretically affected: it is possible to construct elliptic curve points that would falsely appear to be on the given curve. However, there is no known computationally feasible way to construct such points with low order, and so the security of static ECDH private keys is believed to be unaffected."
- "Other routines known to be theoretically affected are modular exponentiation, primality testing, DSA, RSA blinding, JPAKE and SRP. No exploits are known and straightforward bug attacks fail - either the attacker cannot control when the bug triggers, or no private key material is involved."

Counterargument

- Impact assessment is correct — as long as OpenSSL crypto routines are used with OpenSSL BN

- Statement correct for 1.0.1j, but incorrect for 1.0.1e for instance wrt to static ECDH (did not have optimized NISTP256 impl. back then, point addition used **`BN_sqr`** via **`ec_GFp_simple_field_sqr`**)

- Not correct when OpenSSL BN routines are used by third-party crypto

- Example: Android's **`java.math.BigInteger`** uses OpenSSL's BN

  - Uses SpongyCastle, fork of Bouncy Castle [JCE provider => Java crypto implementation]

Bugs fixed in GMP 5.0.4

- Released February 10th, 2012, from ChangeLog:
- “Two bugs in multiplication code causing incorrect computation with extremely low probability have been fixed.”
- “Two bugs in the gcd code have been fixed. They could lead to incorrect results, but for uniformly distributed random operands, the likelihood for that is infinitesimally small. (There was also a third bug, but that was an incorrect ASSERT, which furthermore was not enabled by default.)”
- “A bug affecting 32-bit PowerPC division has been fixed. The bug caused miscomputation for certain divisors in the range 2^32 ... 2^64-1 (about 1 in 2^30 of these)”

GMP 5 mult bugs

- GMP uses different algorithms for BigNums of different sizes
  - reason: asymptotically faster algorithms exist for larger numbers, but higher constant
- Toom-Cook for "medium-sized" numbers $\Theta(n^{1.465})$
  - Uses polynomial multiplication and interpolation
  - Bugs occur in interpolation (trigger non-trivial to construct)
- Crossover values for algorithm choice are highly arch specific (e.g. 74x 64-bit limbs on 64-bit Core2Duo => 4736 bits*)

* **Correction to presented slide deck which claimed 23 instead of 74 limbs!**

The patch

```
MPN_DECR_U (r1 + spt + BIT_CORRECTION,
            n3p1 - spt - BIT_CORRECTION, cy);

cy = mpn_sub_1 (r1 + spt + BIT_CORRECTION,
                r1 + spt + BIT_CORRECTION,
                n3p1 - spt - BIT_CORRECTION, cy);
```

- *"MPN_DECR_U does {ptr,size} -= n, […] expecting no carry (or borrow) from that"*
- Carry mispropagation again!
  [operating on single limb instead of whole BigNum]

Bug pattern: carry mispropagation

- Cause of **`BN_sqr()`** bug(s) and GMP multiplication bugs
- Also observed in Ed25519 implementations
  - TweetNaCl: http://www.skylable.com/blog/2014/05/tweetnacl-carrybit-bug/
  - NaCl: http://tweetnacl.cr.yp.to/tweetnacl-20140917.pdf:

    "For example, four implementations of the ed25519 signature system have been publicly available and waiting for integration into NaCl since 2011, but in total they consist of 5521 lines of C code and 16184 lines of qhasm code. Partial audits have revealed a bug in this software (r1 += 0 + carry should be r2 += 0 + carry in amd64-64-24k)"

- Carry mispropagation problem exploited in:
  B.B. Brumley and M. Barbosa and D. Page and F. Vercauteren: *Practical realisation and elimination of an ECC-related software bug attack*, CT-RSA 2012 [full paper: https://eprint.iacr.org/2011/633]

libgcrypt 1.6.0

```
commit 246b7aaae1ee459f440260bbc4ec2c01c5dc3362
Author: Werner Koch <wk@gnupg.org>
Date:   Fri May 9 12:35:15 2014 +0200

    mpi: Fix a subtle bug setting spurious bits with in mpi_set_bit.

    * mpi/mpi-bit.c (_gcry_mpi_set_bit, _gcry_mpi_set_highbit): Clear
    allocated but not used bits before resizing.
    * tests/t-mpi-bits.c (set_bit_with_resize): New.
    --

    Reported-by: Martin Sewelies.

    This bug is probably with us for many years.  Probably due to
    different memory allocation patterns, it did first revealed itself
    with 1.6.  It could be the reason for other heisenbugs.

    Signed-off-by: Werner Koch <wk@gnupg.org>
```

Potential impact of libgcrypt bug

- Which bug class? Uninitialized variable? No, but close
- Infoleak? Can leak uninitialized data in BigNums
- But! Can also force uninitialized bits to values using heap primitives
  - Predominantly in multithreaded environments
  - Exact impact depends on allocator

Who uses mpi_sethighbit?

```
/*
 * Generate a random secret exponent K less than Q.
 * Note that ECDSA uses this code also to generate D.
 */
gcry_mpi_t
_gcry_dsa_gen_k (gcry_mpi_t q, int security_level)
{
[…]
      /* Make sure we have the requested number of bits.  This code
         looks a bit funny but it is easy to understand if you
         consider that mpi_set_highbit clears all higher bits.  We
         don't have a clear_highbit, thus we first set the high bit
         and then clear it again.  */
      if (mpi_test_bit (k, nbits-1))
        mpi_set_highbit (k, nbits-1);
      else
        {
          mpi_set_highbit (k, nbits-1);
          mpi_clear_bit (k, nbits-1);
        }
[…]
```

Looks unexploitable, but more eyes needed here!

Using verification to find bugs

- Unconstrained symbolic execution (proposed by Ramos & Engler)
- Given: Different implementations of same operation
- If we can prove that they are equivalent:
  - all have the same bug
  - or all are correct
- Inequivalence:
  - Test case demonstrating bug in one of the implementations
- UC-KLEE not publicly available, but approach easily reproducible using KLEE for our case

Symbolic Execution Challenges

- Assembly code needs to be accurately lifted to LLVM
  - Lekkertech has multi-arch lifter, Trail of Bits published McSema for x86
    - Likely needs to support instruction extensions well: SSE2, AVX2, NEON etc.
- Specific to SAW: LLVM function arguments need to be integers (very limiting)

Galois' SAW

- Software Analysis Workbench
- Allows program analysis on LLVM bitcode and Java byte code
- Allows (cross-language) equivalence proofs of code
- Leverages symbolic execution
  - supports ABC, Boolector, CVC4, MathSAT, Yices, and Z3 as SMT solvers through SBV (Haskell SBV)
- Written in Haskell, alpha quality
  [cannot handle non-integer arguments for LLVM funcs]
- Code available on GitHub [free for non-commercial use]

Alternative: property-based bug hunting

- Arithmetic operations fulfill axioms that can be checked
- Example: $f(f(a)) \stackrel{?}{=} a$ with $f(a) = a^{-1} \bmod p$
- Example: $g(a, a) \stackrel{?}{=} f(a)$ with $g(a, b) = ab$ and $f(a) = a^2$
- How to randomly check for many inputs fast?
  - Fuzz it!
  - “Test harness” causes crash on inequivalence

Fuzzing

- Surprise! afl-fuzz finds **`BN_sqr`** bug in x86_64-gcc.c in < 18M iterations with 6000 execs/sec [less than one Xeon E31275 core hour]

```
len = read(STDIN_FILENO, buf, 256);
if (len <= 0) exit(1);

a = BN_bin2bn(buf, len, NULL);  b = BN_bin2bn(buf, len,
NULL);
r1 = BN_new();                  r2 = BN_new();

if (a == NULL || b == NULL) exit(1);

BN_sqr(r1, a, ctx);
/* BN_mul(r2, a, a, ctx) calls BN_sqr() !!! */
BN_mul(r2, a, b, ctx);

/* raise SIGFPE if results differ */
if (BN_cmp(r1, r2) != 0) return 0/0;
```

Conclusions

- BigNum vulnerabilities are real and can bite you
- Medium-term goal: Formal verification for arithmetic underlying cryptographic primitives desirable
- Optimized assembly can be significant hurdle
  - Instruction set models for x86 (and ARMv7 for the HOL Theorem Prover exist
    - without SSE and other extensions
- Optimization and countermeasures against side-channel attacks (const-time methods) lead to significantly increased complexity
  - more room for bugs
- LibTomCrypt is pretty nice to read (only bug found in last 10 years was in prime generation — failed to iterate Miller-Rabin)

Bibliography

- Eli Biham, Yaniv Carmeli, Adi Shamir: *Bug Attacks*, Proceedings of CRYPTO 2008, LNCS 5157, Springer, 2008, p. 221-240.

- B.B. Brumley, M. Barbosa, D. Page, F. Vercauteren: *Practical realisation and elimination of an ECC-related software bug attack*, Proceedings of CT-RSA 2012, LNCS 7178, Springer, 2012, p. 171-186.

- David A. Ramos, Dawson R. Engler: *Practical, Low-Effort Equivalence Verification of Real Code*. Proceedings of CAV 2011, LNCS 6806, Springer, 2011, p. 669-685.