Does open-source cryptographic software work correctly?

Daniel J. Bernstein

CVE-2018-0733, an OpenSSL bug

"Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte." Bug introduced May 2016.

CVE-2018-0733, an OpenSSL bug

"Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte." Bug introduced May 2016.

How severe is this? "This allows an attacker to forge messages that would be considered as authenticated in an amount of tries lower than that guaranteed by the security claims of the scheme."

CVE-2018-0733, an OpenSSL bug

"Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte." Bug introduced May 2016.

How severe is this? "This allows an attacker to forge messages that would be considered as authenticated in an amount of tries lower than that guaranteed by the security claims of the scheme." — Yes, 2^{16} is "lower than" 2^{128}.

CVE-2017-3738, another OpenSSL bug

Don't care about PA-RISC? How about Intel?

"There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli." Bug introduced July 2013.

CVE-2017-3738, another OpenSSL bug

Don't care about PA-RISC? How about Intel?

"There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli." Bug introduced July 2013.

"Attacks against DH1024 are considered just feasible"

CVE-2017-3738, another OpenSSL bug

Don't care about PA-RISC? How about Intel?

"There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli." Bug introduced July 2013.

"Attacks against DH1024 are considered just feasible" — How long? How much hardware?

CVE-2017-3738, continued

"Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

CVE-2017-3738, continued

"Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

— Really? How much public scrutiny has the actual computation received from cryptanalysts?

CVE-2017-3738, continued

"Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

— Really? How much public scrutiny has the actual computation received from cryptanalysts?

What this looks like to me: "We have analyzed our new cryptosystem and concluded that attacks are not likely."

CVE-2017-3738, continued

"Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

— Really? How much public scrutiny has the actual computation received from cryptanalysts?

What this looks like to me: "We have analyzed our new cryptosystem and concluded that attacks are not likely." — Don't we require public review?

Part of the CVE-2017-3738 patch

```
@@ -1093,7 +1093,9 @@
  vmovdqu  -8+32*2-128($ap),$TEMP2

  mov      $r1, %rax
+ vpblendd \$0xfc, $ZERO, $ACC9, $ACC9 # corre
  imull    $n0, %eax
+ vpaddq   $ACC9,$ACC4,$ACC4           # corre
  and      \$0x1ffffff, %eax

  imulq    16-128($ap),%rbx
@@ -1329,15 +1331,12 @@
```

Is open-source software bug-free?

Eric S. Raymond, 1999: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone. Or, less formally, 'Given enough eyeballs, all bugs are shallow.' "

Is open-source software bug-free?

Eric S. Raymond, 1999: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone. Or, less formally, 'Given enough eyeballs, all bugs are shallow.' "

— "Beta-tester": Ultimately, the unhappy user?

Is open-source software bug-free?

Eric S. Raymond, 1999: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone. Or, less formally, 'Given enough eyeballs, all bugs are shallow.' "

— "Beta-tester": Ultimately, the unhappy user?

— "Almost every problem": That's not "all bugs"! Don't we care about the exceptions? Rare bugs can be devastating, especially for security!

More reasons for skepticism

— How do we know how many exceptions there are?
How many people are looking for unobvious bugs?

More reasons for skepticism

— How do we know how many exceptions there are? How many people are looking for unobvious bugs?

— How can there be enough people looking for bugs when most developers prefer writing new code?

More reasons for skepticism

— How do we know how many exceptions there are?
How many people are looking for unobvious bugs?

— How can there be enough people looking for bugs when most developers prefer writing new code?

— ESR advocates a development methodology that releases a constant flood of new bugs.
Doesn't this make his "law" automatically true?
Is this the correctness metric that users want?

So we should use closed source?

"Closed source stops attackers from finding bugs."

So we should use closed source?

"Closed source stops attackers from finding bugs."
— What's the evidence for this?
How long does it take for an attacker to
extract, disassemble, decompile the code?

So we should use closed source?

"Closed source stops attackers from finding bugs."
— What's the evidence for this?
How long does it take for an attacker to
extract, disassemble, decompile the code?

"Closed source scares away some lazy academics,
so we have fewer bug announcements to deal with."

So we should use closed source?

"Closed source stops attackers from finding bugs."
— What's the evidence for this?
How long does it take for an attacker to extract, disassemble, decompile the code?

"Closed source scares away some lazy academics, so we have fewer bug announcements to deal with."
— Sounds plausible, but is the delay worthwhile?
e.g. Infineon deployed RSALib very widely before 2017 Nemec–Sys–Svenda–Klinec–Matyas "ROCA".

Closed source, continued

"Closed source makes money, allowing investment in serious code review, producing bug-free code."
— What's the evidence that this process works?

Closed source, continued

"Closed source makes money, allowing investment in serious code review, producing bug-free code."
— What's the evidence that this process works?

This isn't a talk recommending closed source.

Closed source, continued

"Closed source makes money, allowing investment in serious code review, producing bug-free code."
— What's the evidence that this process works?

This isn't a talk recommending closed source.

I'm focusing on open source in this talk because

- I spend most of my time with open source and
- the only paths that I see towards real security need everything published to build confidence.

Cryptography is notoriously hard to review

Mathematical complications lead to subtle bugs.

Cryptography is notoriously hard to review

Mathematical complications lead to subtle bugs.

Side-channel countermeasures add more complexity.

Cryptography is notoriously hard to review

Mathematical complications lead to subtle bugs.

Side-channel countermeasures add more complexity.

Post-quantum cryptography: even more complex.

Cryptography is notoriously hard to review

Mathematical complications lead to subtle bugs.

Side-channel countermeasures add more complexity.

Post-quantum cryptography: even more complex.

Cryptography is applied to large volumes of data.
Often individual computations are time-consuming.
Pursuit of speed $\Rightarrow$ many cryptographic choices;
cryptographic code optimized for particular CPUs.

Cryptography is notoriously hard to review

Mathematical complications lead to subtle bugs.

Side-channel countermeasures add more complexity.

Post-quantum cryptography: even more complex.

Cryptography is applied to large volumes of data.
Often individual computations are time-consuming.
Pursuit of speed $\Rightarrow$ many cryptographic choices;
cryptographic code optimized for particular CPUs.

e.g. Keccak Code Package: $>$20 implementations.
e.g. Google added hand-written Cortex-A7 asm to Linux kernel for Speck, then switched to ChaCha.

Formal logic to the rescue?

Whitehead and Russell, *Principia Mathematica*, volume 1, 1st edition (1910), page 379:

$*54{\cdot}43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[*51{\cdot}231] \qquad \equiv . \iota\text{‘}x \cap \iota\text{‘}y = \Lambda .$

$[*13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

$\qquad \vdash :. (\exists x, y) . \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these tools today.)

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these tools today.)

This is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these tools today.)

This is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Good: High confidence that subtle bugs are gone (in the code; but worry about compiler, CPU, . . .).

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these tools today.)

This is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Good: High confidence that subtle bugs are gone (in the code; but worry about compiler, CPU, . . .).
Bad: Tons of effort for each implementation.

Testing

Testing is great. Test everything. Design for tests.

Why wasn't the PA-RISC `CRYPTO_memcmp` run through millions of tests on random inputs? And tests on inputs differing in a few positions? SUPERCOP test framework has always done this.

Testing

Testing is great. Test everything. Design for tests.

Why wasn't the PA-RISC `CRYPTO_memcmp` run through millions of tests on random inputs? And tests on inputs differing in a few positions? SUPERCOP test framework has always done this.

Good reaction to a bug: "How can I build fast automated tests that will catch this kind of bug?" Even better to ask question before bug happens.

Going beyond testing particular inputs

Testing (and fuzzing) many smart inputs can still miss attacker-triggerable bugs for rare inputs.

Going beyond testing particular inputs

Testing (and fuzzing) many smart inputs can still miss attacker-triggerable bugs for rare inputs.

Fix: **Run code on all inputs**.

Going beyond testing particular inputs

Testing (and fuzzing) many smart inputs can still miss attacker-triggerable bugs for rare inputs.

Fix: **Run code on all inputs**.

1. Easy *if* code has no input-dependent branches: code $\rightarrow$ simple language without loops/vectors/. . . .
(I'm using `angr.io` for symbolic execution.)

Going beyond testing particular inputs

Testing (and fuzzing) many smart inputs can still miss attacker-triggerable bugs for rare inputs.

Fix: **Run code on all inputs**.

1. Easy *if* code has no input-dependent branches: code $\rightarrow$ simple language without loops/vectors/. . . .
(I'm using angr.io for symbolic execution.)

2. Automatically identify equivalent computations.
Don't have to redo work for each implementation!

Going beyond testing particular inputs

Testing (and fuzzing) many smart inputs can still miss attacker-triggerable bugs for rare inputs.

Fix: **Run code on all inputs**.

1. Easy *if* code has no input-dependent branches: code $\rightarrow$ simple language without loops/vectors/. . . .
(I'm using `angr.io` for symbolic execution.)

2. Automatically identify equivalent computations. Don't have to redo work for each implementation!

3. Build tools to check that the computations work.

A case study

Subroutine in some post-quantum proposals:
sorting arrays of integers.

A case study

Subroutine in some post-quantum proposals:
sorting arrays of integers.

Software library from `sorting.cr.yp.to`:

- New speed records for in-memory sorting.
- Side-channel countermeasures: no secret branch conditions; no secret array indices.
- Tool verifies correct sorting of all inputs. No need to review per-CPU optimized code.