Crypto Code

The 9 circles of testing

JP Aumasson, Kudelski Security

Everybody does it; it's just that nobody talks about it.

Why it’s hard

You need to know crypto and software

Methodologies aren’t documented

Tools aren’t always available

Street cred

Wrote and reviewed some crypto code

Like code for millions unpatchable devices

Made many mistakes

Tested many tests

What do we want?

Functional testing & security testing

Functional testing

Valid inputs give valid output

Invalid inputs trigger appropriate errors

Goal: test all execution paths

Security testing

Program can’t be abused

Doesn’t leak secrets

Overlaps with functional testing

What we’re testing

Code against code or against specs

Usually C code, which doesn’t help

Code against code

Easiest case

When porting to a new language/platform

You'll assume that the ref code is correct
(Though it's probably not)

Can generate all test vectors you want

Code against specs

Often occurs with standards (ex: SHA-3)

Only a handful of test vectors, if any

Specs can be incomplete or incorrect

Try to have 2 independent implementers

The 9 circles

From most basic to most sophisticated

You may not need all of those

The “what” more than the ”how”

I probably missed important points

1. Test vectors

Unit-test ciphers, hashes, parsers, etc.

Maximize code coverage by varying inputs lengths and values

Make coherence tests, as in BRUTUS https://github.com/mjosaarinen/brutus

To avoid storing thousands values, record only a checksum (as in SUPERCOP)

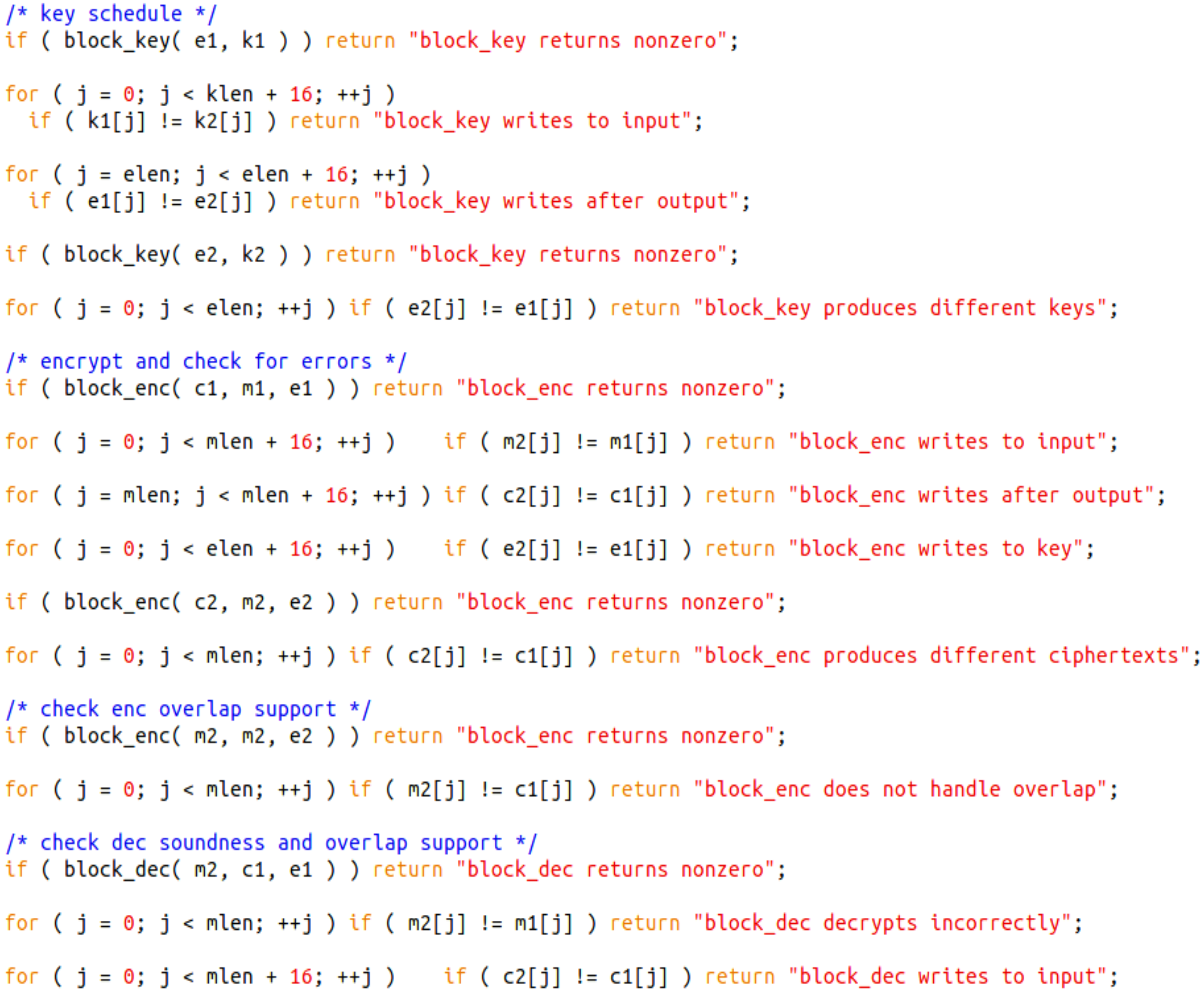
1. Test vectors

Against specs, test vectors less useful

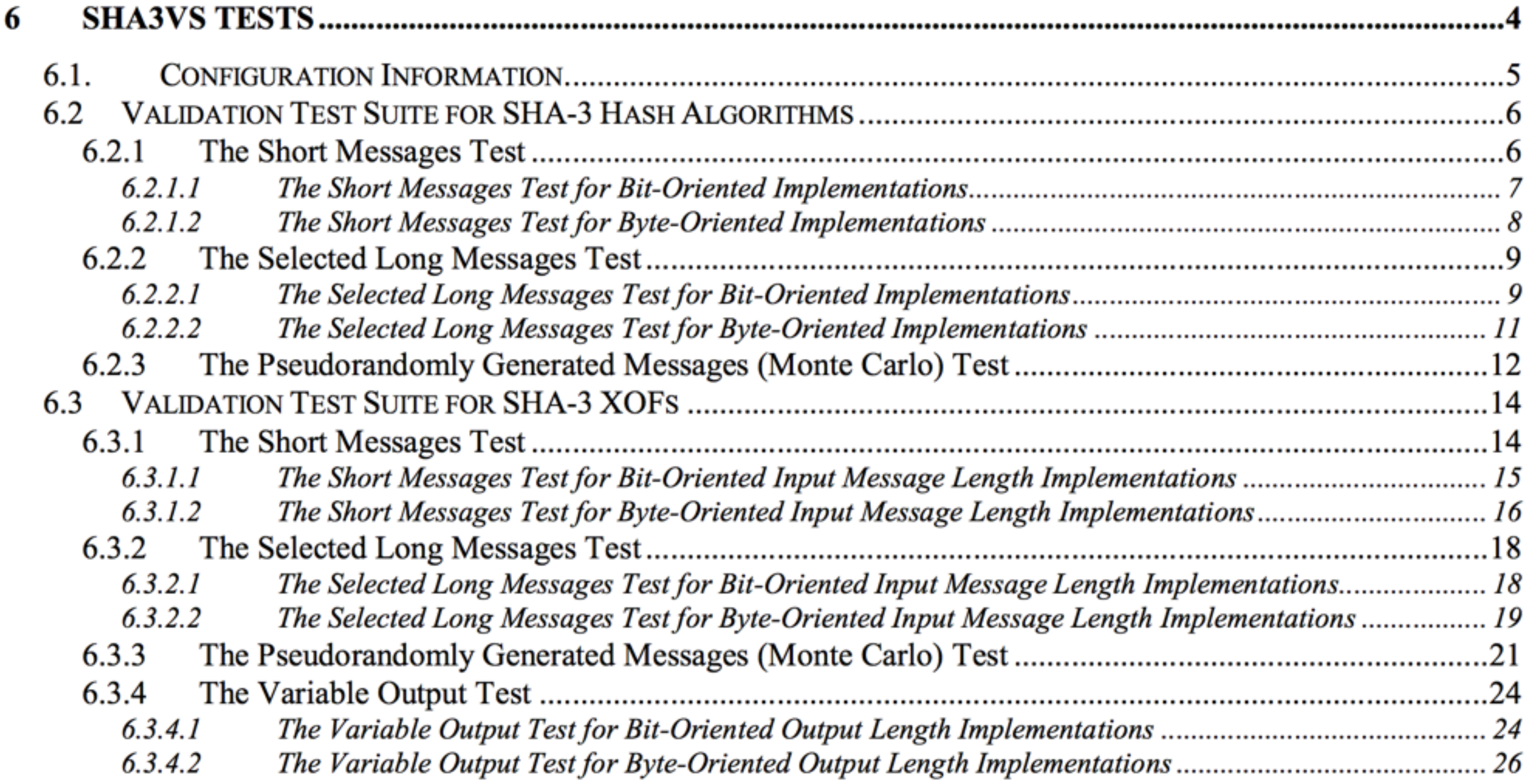
Bug in BLAKE ref code unnoticed for 7 years

```
/* compress remaining data filled with new bits */
-   if( left && ( ((databitlen >> 3) & 0x3F) >= fill ) ) {
+   if( left && ( ((databitlen >> 3) ) >= fill ) ) {
      memcpy( (void *) (state->data32 + left),
     (void *) data, fill );
```

Found by a careful user (thanks!)



```
/* key schedule */
if ( block_key( e1, k1 ) ) return "block_key returns nonzero";

for ( j = 0; j < klen + 16; ++j )
  if ( k1[j] != k2[j] ) return "block_key writes to input";

for ( j = elen; j < elen + 16; ++j )
  if ( e1[j] != e2[j] ) return "block_key writes after output";

if ( block_key( e2, k2 ) ) return "block_key returns nonzero";

for ( j = 0; j < elen; ++j ) if ( e2[j] != e1[j] ) return "block_key produces different keys";

/* encrypt and check for errors */
if ( block_enc( c1, m1, e1 ) ) return "block_enc returns nonzero";

for ( j = 0; j < mlen + 16; ++j )    if ( m2[j] != m1[j] ) return "block_enc writes to input";

for ( j = mlen; j < mlen + 16; ++j ) if ( c2[j] != c1[j] ) return "block_enc writes after output";

for ( j = 0; j < elen + 16; ++j )    if ( e2[j] != e1[j] ) return "block_enc writes to key";

if ( block_enc( c2, m2, e2 ) ) return "block_enc returns nonzero";

for ( j = 0; j < mlen; ++j ) if ( c2[j] != c1[j] ) return "block_enc produces different ciphertexts";

/* check enc overlap support */
if ( block_enc( m2, m2, e2 ) ) return "block_enc returns nonzero";

for ( j = 0; j < mlen; ++j ) if ( m2[j] != c1[j] ) return "block_enc does not handle overlap";

/* check dec soundness and overlap support */
if ( block_dec( m2, c1, e1 ) ) return "block_dec returns nonzero";

for ( j = 0; j < mlen; ++j ) if ( m2[j] != m1[j] ) return "block_dec decrypts incorrectly";

for ( j = 0; j < mlen + 16; ++j )    if ( c2[j] != c1[j] ) return "block_dec writes to input";
```



NIST's Cryptographic Algorithm Validation Program
http://csrc.nist.gov/groups/STM/cavp/

2. Basic software tests

Against memory corruption, leaks, etc.

Secure coding very basics

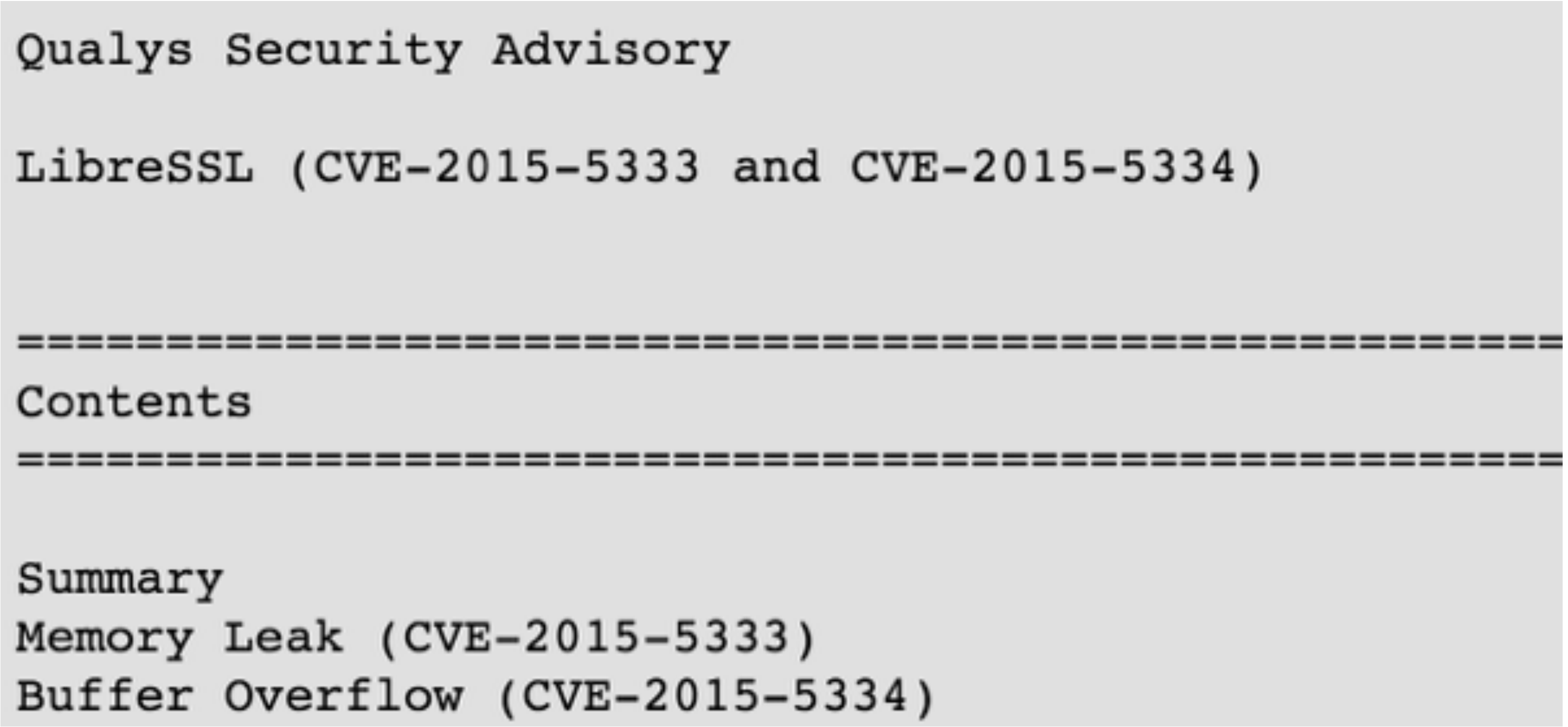
Static analyzers (Coverity, PREfast, etc.)

Valgrind, Clang sanitizers, etc.

Dumb fuzzing (afl-fuzz, etc.)

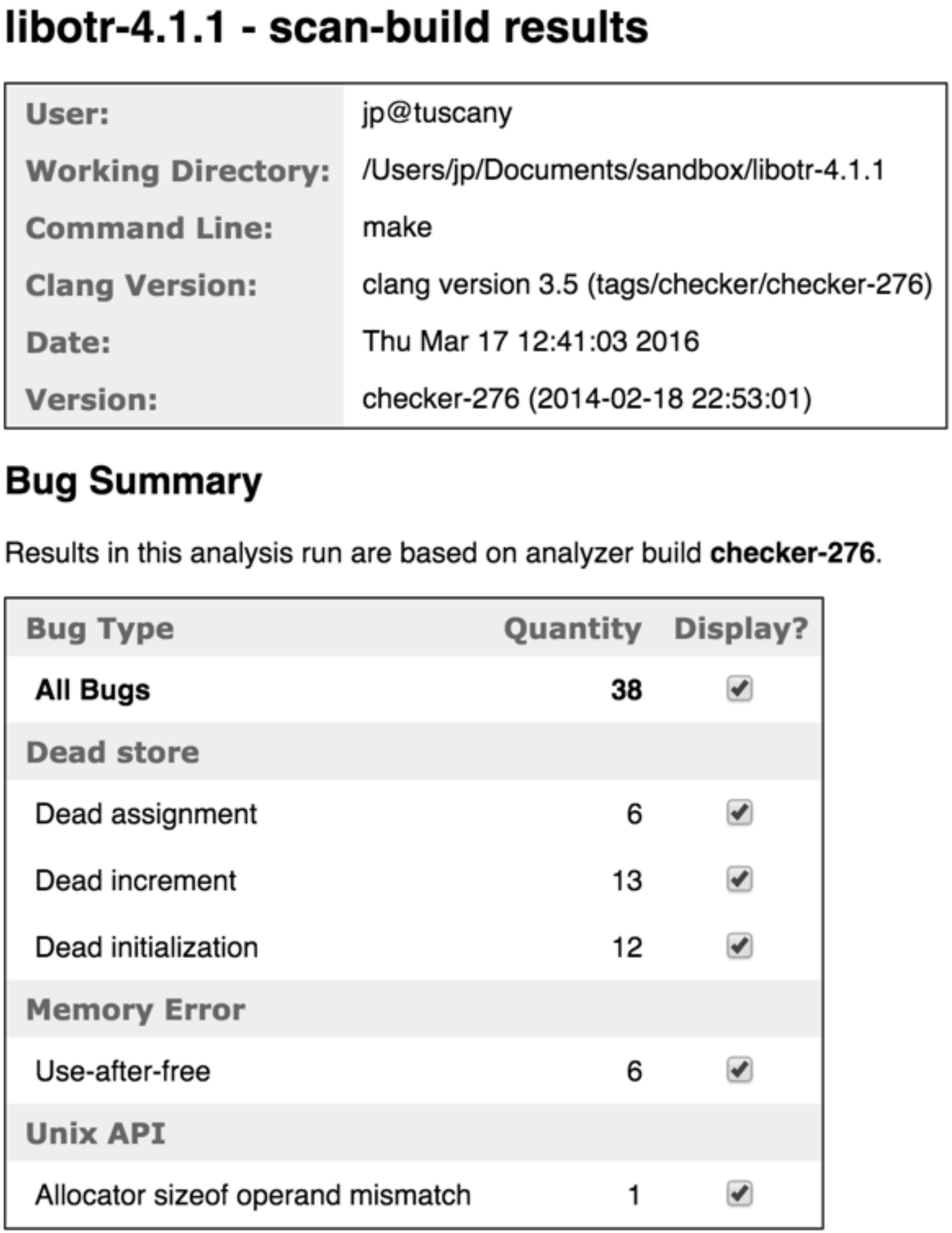
2. Basic software tests

Most frequent, can find high impact bugs (Heartbleed, gotofail)



```
Qualys Security Advisory

LibreSSL (CVE-2015-5333 and CVE-2015-5334)


====================================================
Contents
====================================================

Summary
Memory Leak (CVE-2015-5333)
Buffer Overflow (CVE-2015-5334)
```

http://www.openwall.com/lists/oss-security/2015/10/16/1



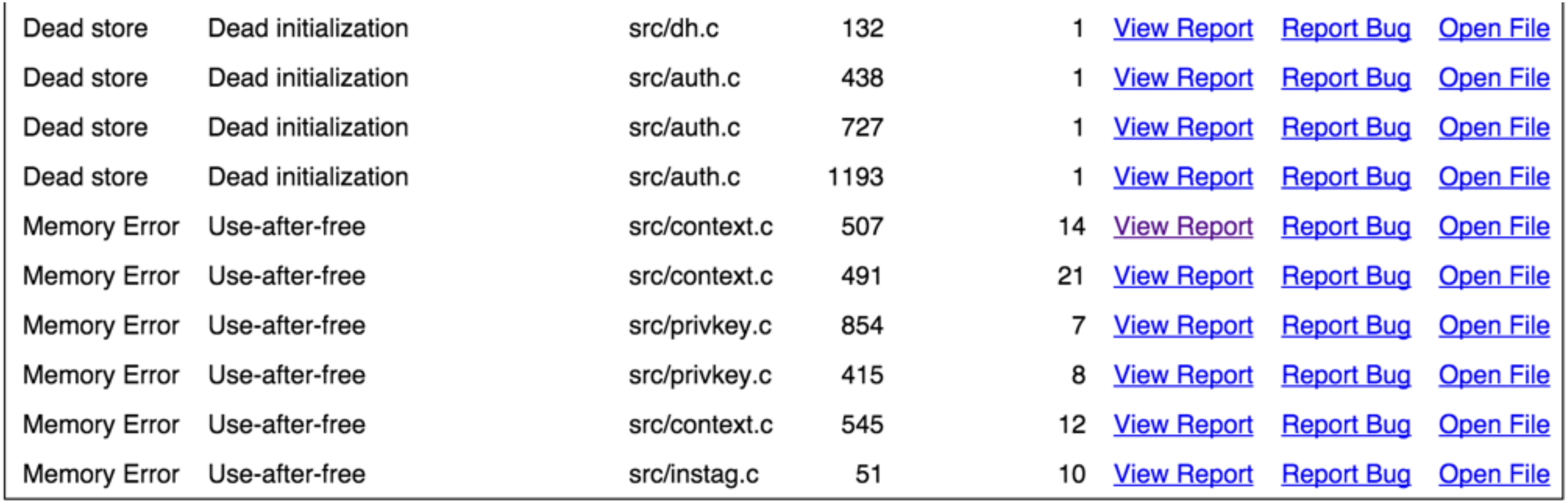
libotr-4.1.1 - scan-build results

User:	jp@tuscany
Working Directory:	/Users/jp/Documents/sandbox/libotr-4.1.1
Command Line:	make
Clang Version:	clang version 3.5 (tags/checker/checker-276)
Date:	Thu Mar 17 12:41:03 2016
Version:	checker-276 (2014-02-18 22:53:01)

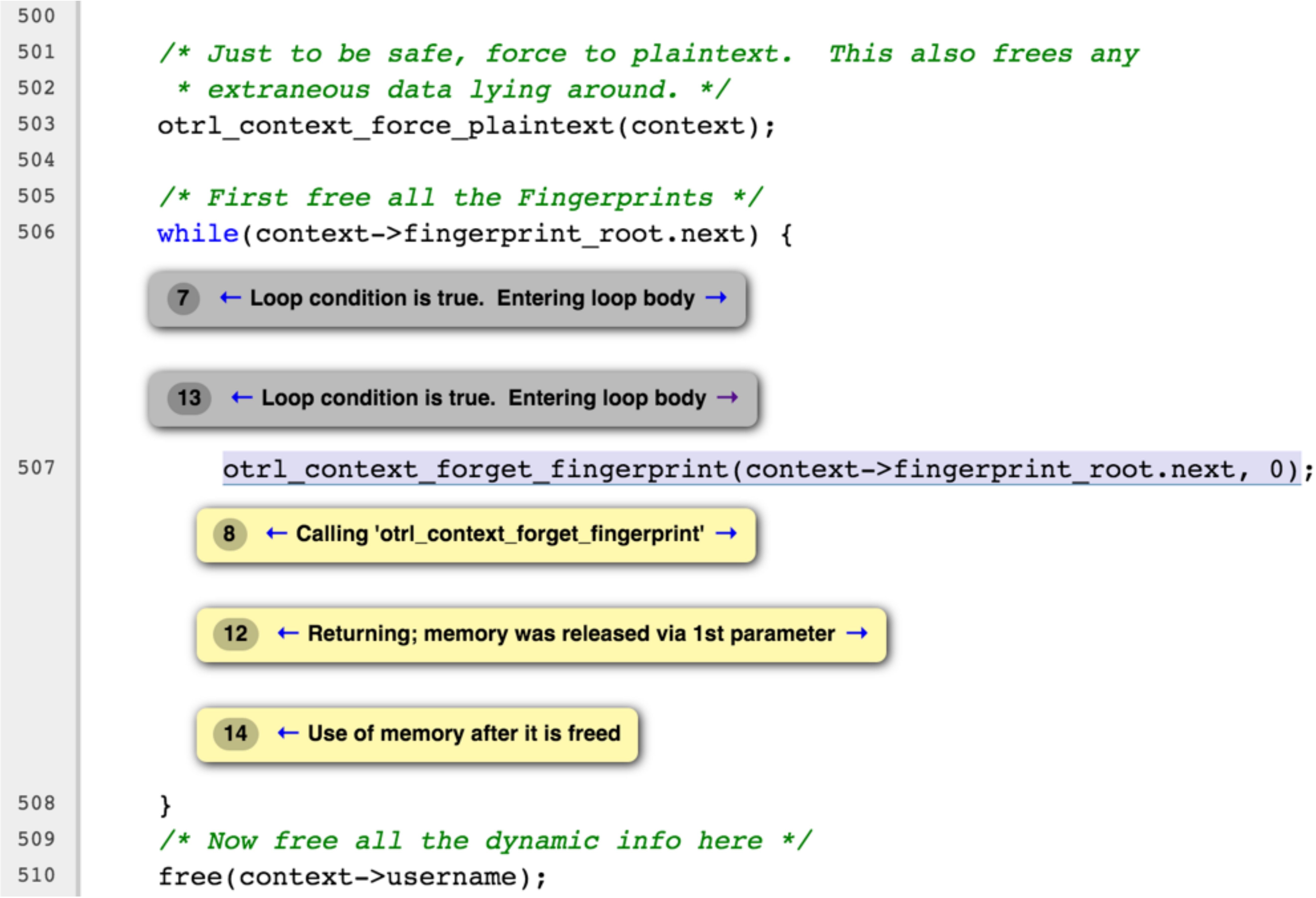
Bug Summary

Results in this analysis run are based on analyzer build **checker-276**.

Bug Type	Quantity	Display?
All Bugs	**38**	☑
Dead store		
Dead assignment	6	☑
Dead increment	13	☑
Dead initialization	12	☑
Memory Error		
Use-after-free	6	☑
Unix API		
Allocator sizeof operand mismatch	1	☑



Dead store	Dead initialization	src/dh.c	132	1	View Report	Report Bug	Open File
Dead store	Dead initialization	src/auth.c	438	1	View Report	Report Bug	Open File
Dead store	Dead initialization	src/auth.c	727	1	View Report	Report Bug	Open File
Dead store	Dead initialization	src/auth.c	1193	1	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/context.c	507	14	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/context.c	491	21	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/privkey.c	854	7	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/privkey.c	415	8	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/context.c	545	12	View Report	Report Bug	Open File
Memory Error	Use-after-free	src/instag.c	51	10	View Report	Report Bug	Open File



```

    /* Just to be safe, force to plaintext.  This also frees any
     * extraneous data lying around. */
    otrl_context_force_plaintext(context);

    /* First free all the Fingerprints */
    while(context->fingerprint_root.next) {
```

7 ← Loop condition is true. Entering loop body →

13 ← Loop condition is true. Entering loop body →

```
        otrl_context_forget_fingerprint(context->fingerprint_root.next, 0);
```

8 ← Calling 'otrl_context_forget_fingerprint' →

12 ← Returning; memory was released via 1st parameter →

14 ← Use of memory after it is freed

```
    }
    /* Now free all the dynamic info here */
    free(context->username);
```

3. Invalid use

Test that it triggers the expected error

Invalid values, malformed input, etc.

For length parameters, parsers

3. Invalid use

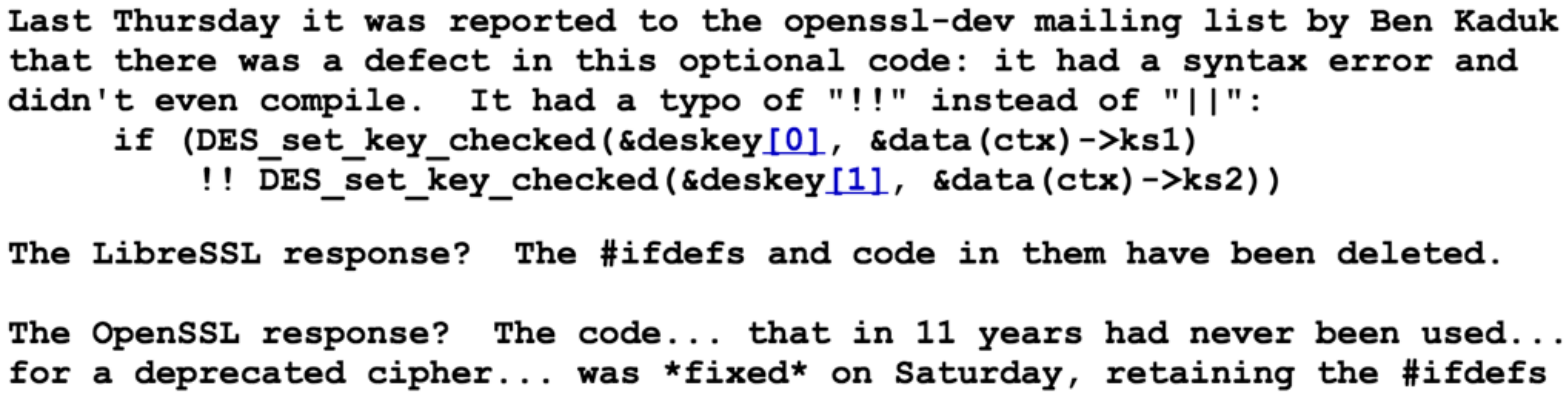
Argon2 omitted a parameter range check:

```
/* Validate memory cost */
    if (ARGON2_MIN_MEMORY > context->m_cost) {
        return ARGON2_MEMORY_TOO_LITTLE;
    }

+    if (context->m_cost < 8*context->lanes) {
+        return ARGON2_MEMORY_TOO_LITTLE;
+    }
+
```

4. Optional features

Don’t forget features buried under #ifdefs

In OpenSSL’s DES optional weak key check



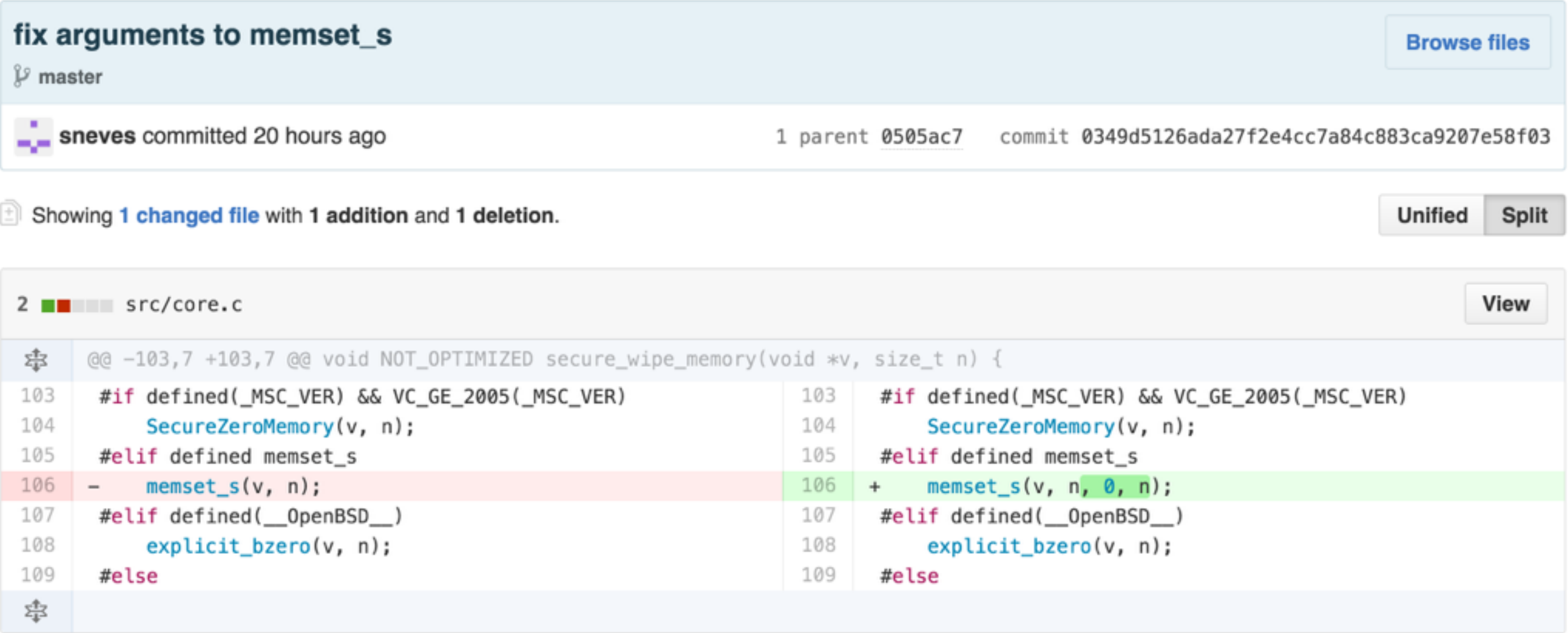
Last Thursday it was reported to the openssl-dev mailing list by Ben Kaduk that there was a defect in this optional code: it had a syntax error and didn't even compile. It had a typo of "!!" instead of "||":

```
     if (DES_set_key_checked(&deskey[0], &data(ctx)->ks1)
         !! DES_set_key_checked(&deskey[1], &data(ctx)->ks2))
```

The LibreSSL response? The #ifdefs and code in them have been deleted.

The OpenSSL response? The code... that in 11 years had never been used... for a deprecated cipher... was *fixed* on Saturday, retaining the #ifdefs

http://marc.info/?l=openbsd-tech&m=144472550016118



fix arguments to memset_s

master

sneves committed 20 hours ago 1 parent 0505ac7 commit 0349d5126ada27f2e4cc7a84c883ca9207e58f03

Showing **1 changed file** with **1 addition** and **1 deletion**.

2 src/core.c

```
@@ -103,7 +103,7 @@ void NOT_OPTIMIZED secure_wipe_memory(void *v, size_t n) {
 #if defined(_MSC_VER) && VC_GE_2005(_MSC_VER)      #if defined(_MSC_VER) && VC_GE_2005(_MSC_VER)
     SecureZeroMemory(v, n);                            SecureZeroMemory(v, n);
 #elif defined memset_s                             #elif defined memset_s
-    memset_s(v, n);                               +    memset_s(v, n, 0, n);
 #elif defined(__OpenBSD__)                         #elif defined(__OpenBSD__)
     explicit_bzero(v, n);                              explicit_bzero(v, n);
 #else                                              #else
```

Yesterday

5. Randomness

Hard to catch bugs

Statistical tests are a bare minimum

Ensure distinct outputs across reboots

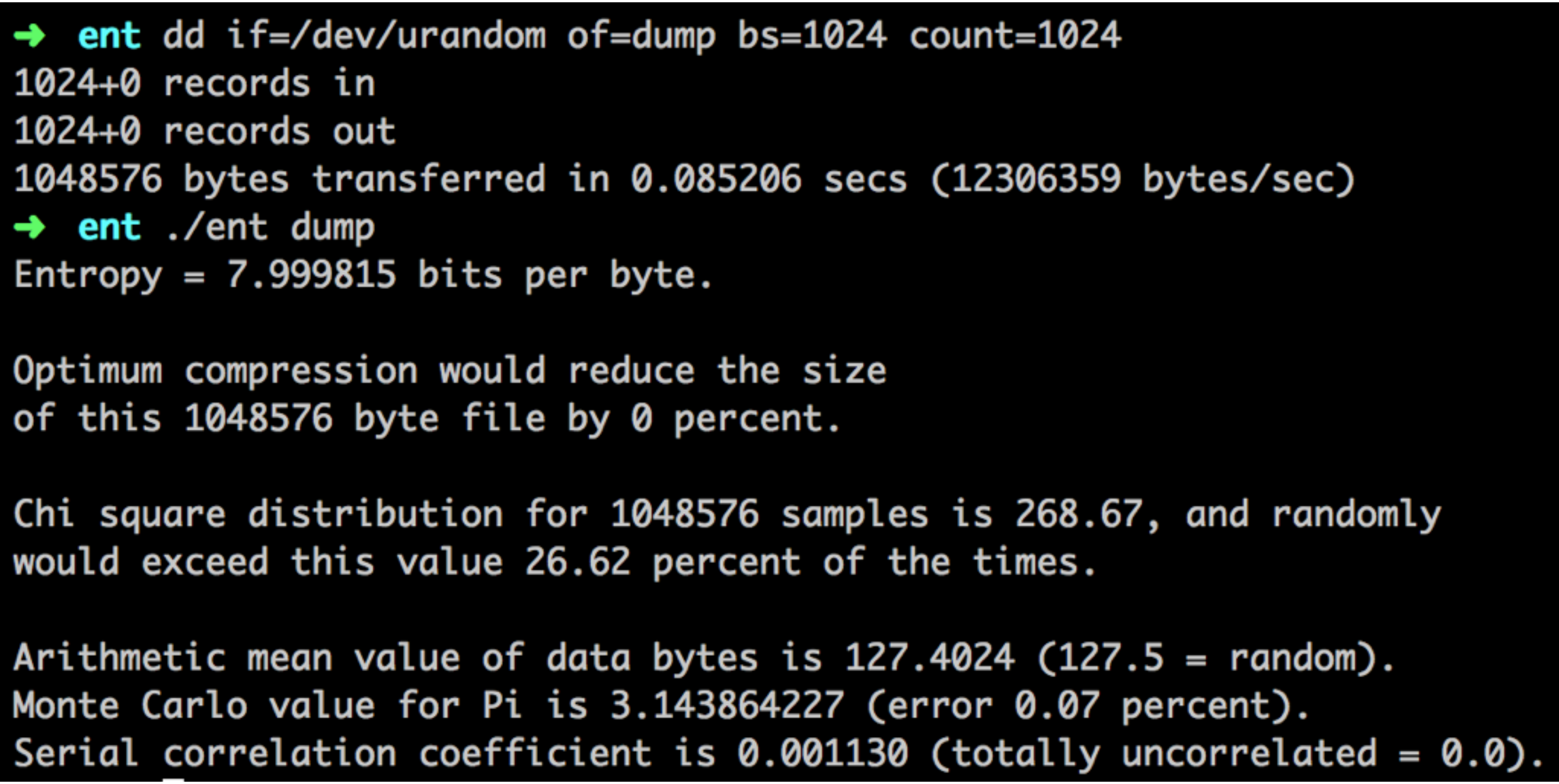
And across devices (see mining p’s & q’s)

5. Randomness

A classic: Debian’s PRNG bug (2008)

```
/* DO NOT REMOVE THE FOLLOWING CALL TO MD_Update()! */
if (!MD_Update(m, buf, j))
    goto err;
/*
 * We know that line may cause programs such as purify and valgrind
 * to complain about use of uninitialized data.  The problem is not,
 * it's with the caller.  Removing that line will make sure you get
 * really bad randomness and thereby other problems such as very
 * insecure keys.
 */
```

OpenSSH keys ended up with 15-bit entropy



```
➜  ent dd if=/dev/urandom of=dump bs=1024 count=1024
1024+0 records in
1024+0 records out
1048576 bytes transferred in 0.085206 secs (12306359 bytes/sec)
➜  ent ./ent dump
Entropy = 7.999815 bits per byte.

Optimum compression would reduce the size
of this 1048576 byte file by 0 percent.

Chi square distribution for 1048576 samples is 268.67, and randomly
would exceed this value 26.62 percent of the times.

Arithmetic mean value of data bytes is 127.4024 (127.5 = random).
Monte Carlo value for Pi is 3.143864227 (error 0.07 percent).
Serial correlation coefficient is 0.001130 (totally uncorrelated = 0.0).
```

6. Timing leaks

When execution time depends on secrets

Avoid branchings, beware memcmp, etc.

Check the assembly, not just C source

Langley's ctgrind https://github.com/agl/ctgrind

https://github.com/veorq/misc/blob/master/ctgrind_valgrind-3.11.0.patch

See also openssl/include/internal/constant_time_locl.h

7. Fuzzing

Dumb fuzzing for exploring parameters' space, parsed formats, bignum arithmetic

CVE-2015-3193 in OpenSSL's BN_mod_exp

CVE-2016-1938 in NSS' mp_div/_exptmod

Integer overflow in Argon2
https://github.com/P-H-C/phc-winner-argon2/issues/5

7. Fuzzing

Smart fuzzing, designed for specific APIs

What Cryptosense is doing for PKCS#11

More for high-level protocols than algorithms

8. Verification

Mathematically proven correctness

Cryptol language http://cryptol.net/ http://galois.com/
+ SAW to extract models from LLVM, Java

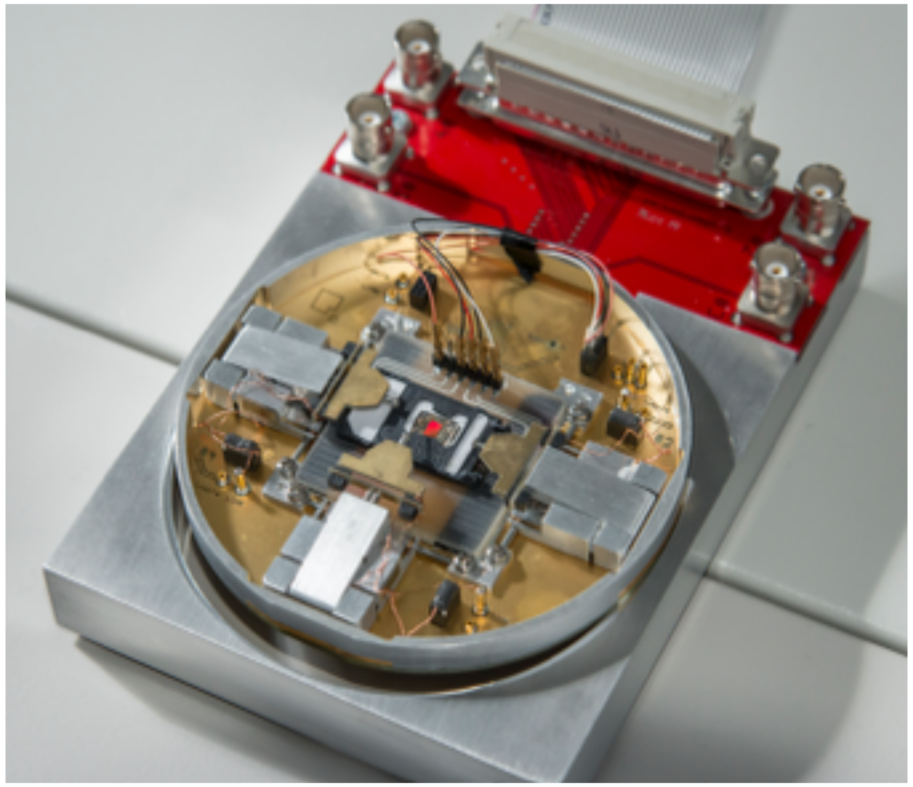
INRIA’s verified TLS https://mitls.org/

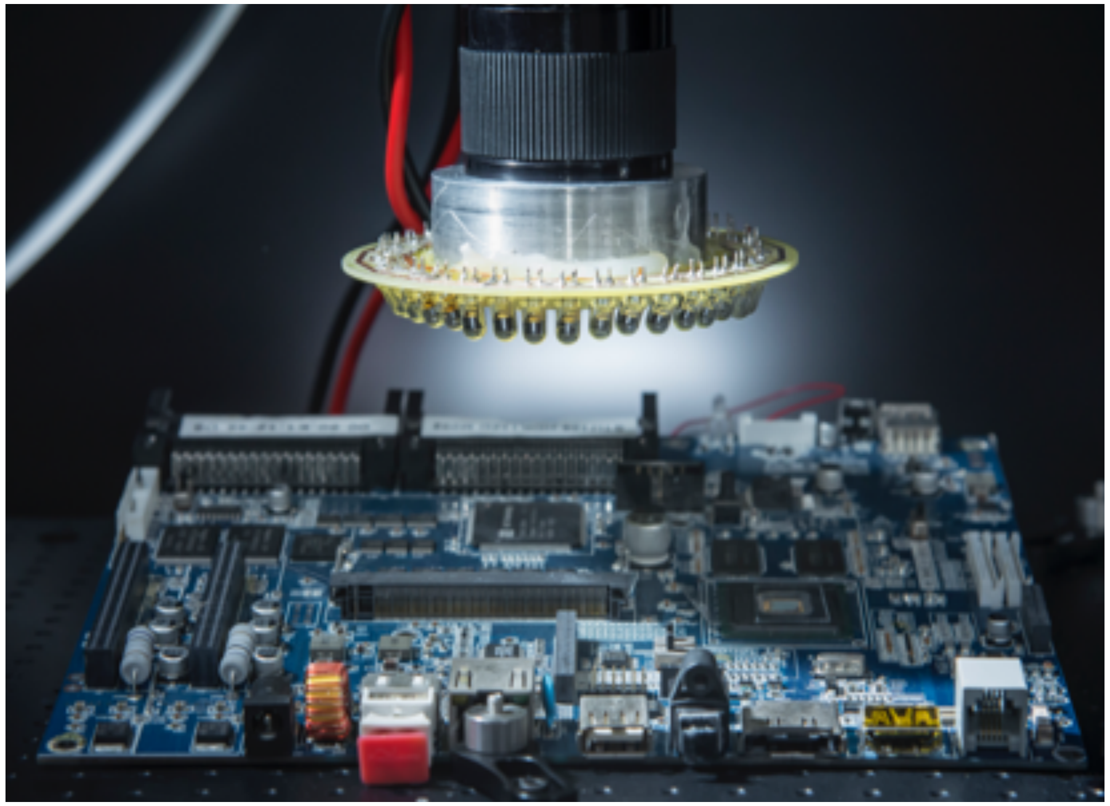
Verified security: LangSec?

9. Physical testing

Test for side channels, fault resilience

As applied to smart cards or game consoles





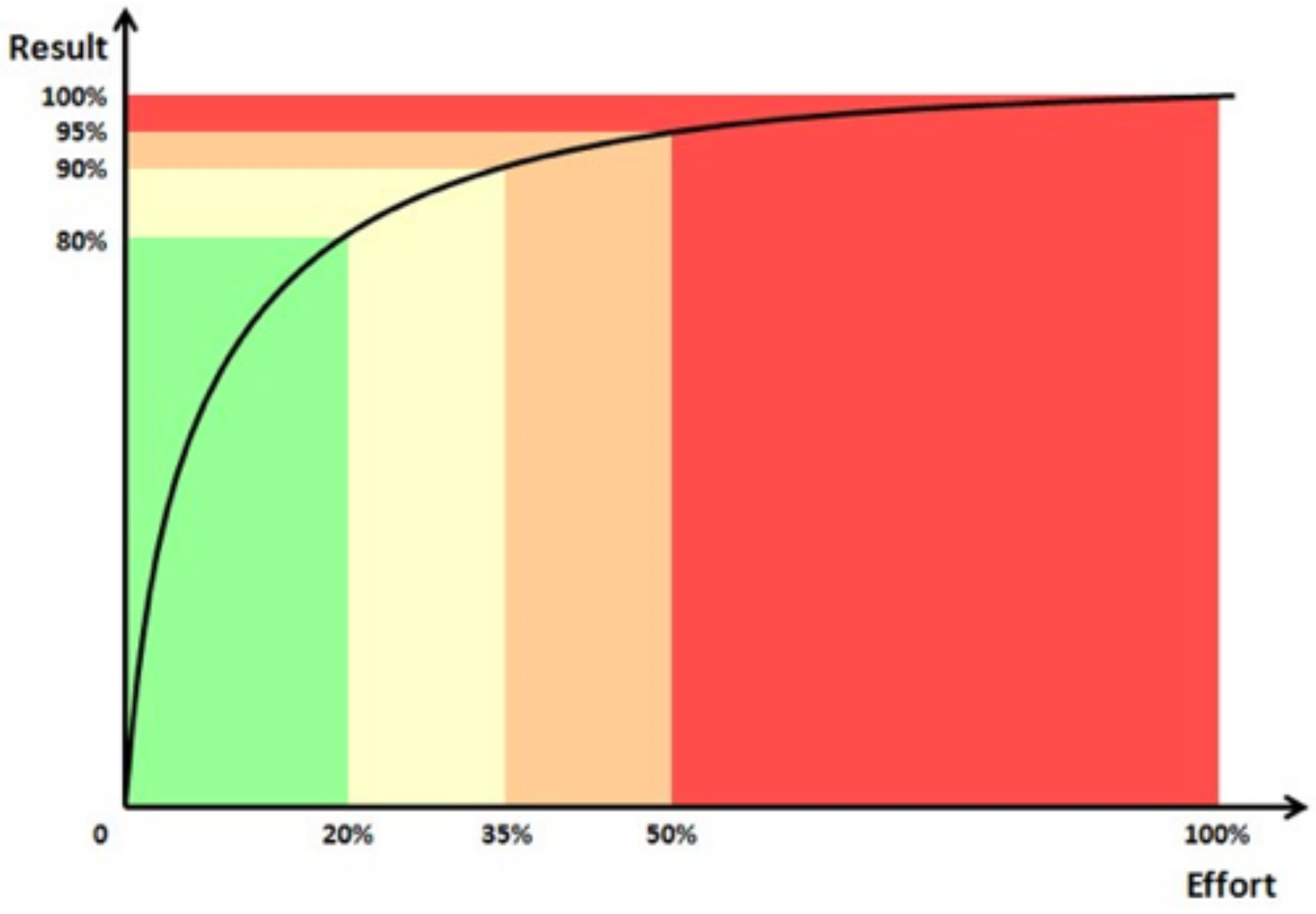
Conclusions

WE CAN ONLY AFFORD TO FIX THE HIGH-PRIORITY BUGS.
www.dilbert.com
scottadams@aol.com
IF WE DON'T FIX 100% OF THE BUGS, THE SOFTWARE WILL BE 100% USELESS.
6-22-09 © 2009 Scott Adams, Inc./Dist. by UFS, Inc.
SO OUR PLAN IS TO FAIL?
MORE SLOWLY.

Conclusions

Pareto: test vectors will spot most bugs

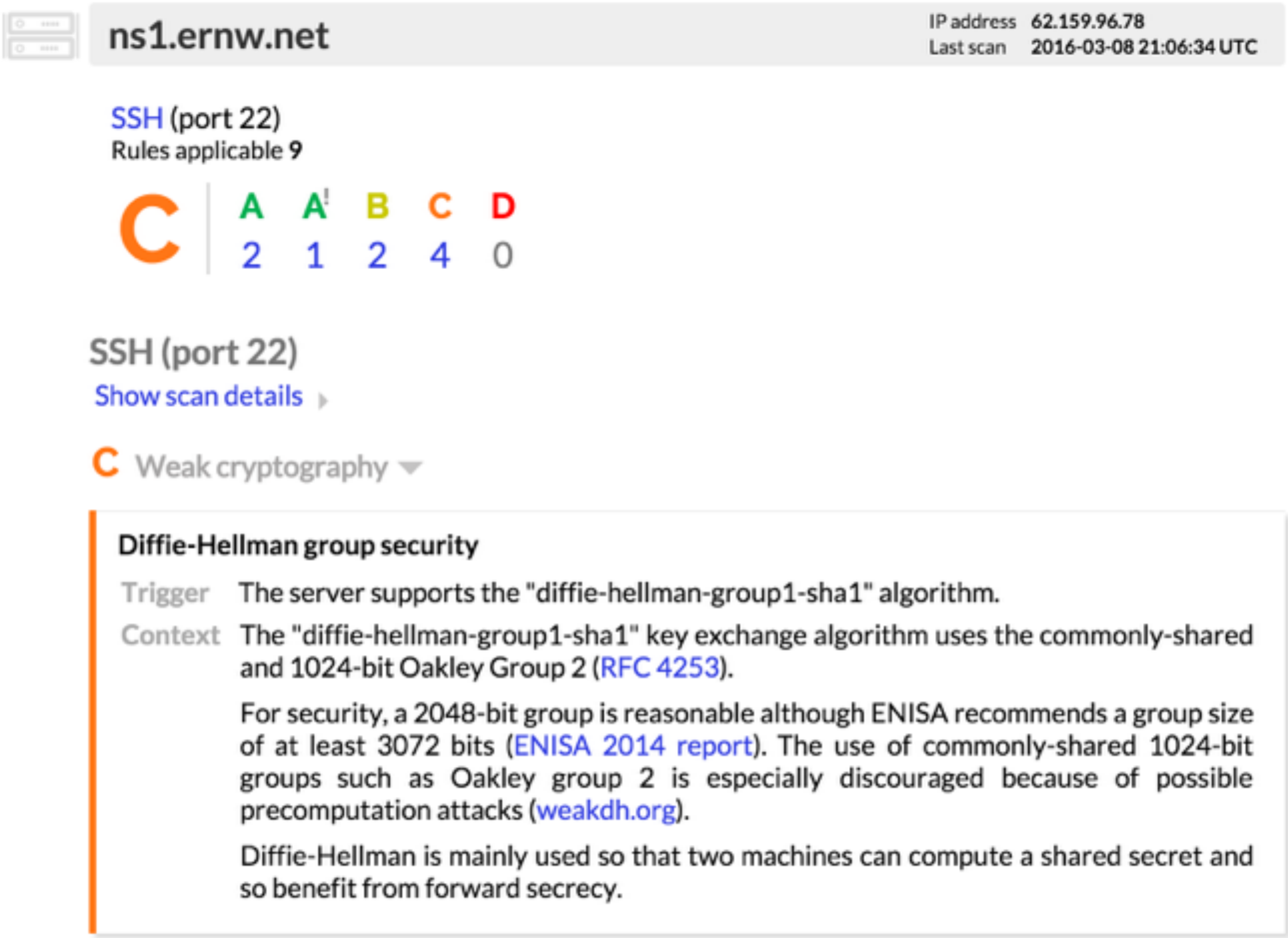
But bugs on the (fat) tail can be critical



Conclusions

I wrote a vulnerability scanner that abstracts all the predicates in a binary, traverses the callgraph and generates phormulaes to run then with a SMT solver.
I found 1 vuln in 3 days with this tool.
He wrote a dumb ass fuzzer and found 5 vulns in 1 day.
Good thing I'm not a n00b like that guy.

Conclusions



ns1.ernw.net

IP address 62.159.96.78
Last scan 2016-03-08 21:06:34 UTC

SSH (port 22)
Rules applicable 9

C

A	A!	B	C	D
2	1	2	4	0

SSH (port 22)

Show scan details

C Weak cryptography

Diffie-Hellman group security

Trigger The server supports the "diffie-hellman-group1-sha1" algorithm.

Context The "diffie-hellman-group1-sha1" key exchange algorithm uses the commonly-shared and 1024-bit Oakley Group 2 (RFC 4253).

For security, a 2048-bit group is reasonable although ENISA recommends a group size of at least 3072 bits (ENISA 2014 report). The use of commonly-shared 1024-bit groups such as Oakley group 2 is especially discouraged because of possible precomputation attacks (weakdh.org).

Diffie-Hellman is mainly used so that two machines can compute a shared secret and so benefit from forward secrecy.

https://discovery.cryptosense.com/analyze/troopers.de/d4c7579

Conclusions

First do basic automated tests

Machine don’t replace human review though

Few capable people/companies for crypto

Make your code/APIs test/review-friendly

See coding rules on https://cryptocoding.net

Thanks!