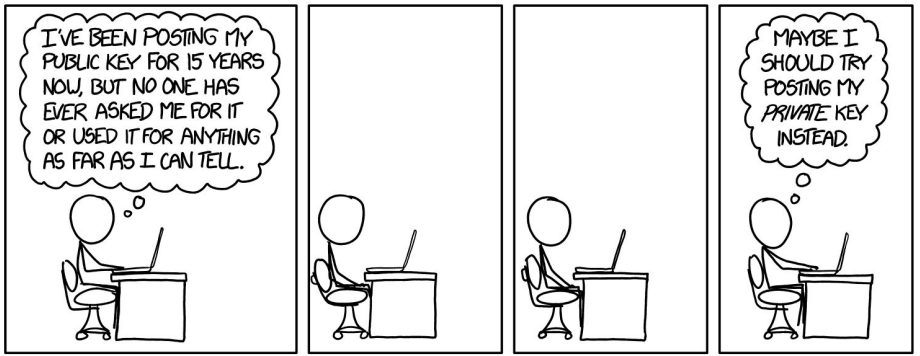
Clang tools for implementing cryptographic protocols like OTRv4

Sofía Celi

What is OTR and why it was created?

- Cryptographic protocol
- Paper in 2004 by *Ian Goldberg*, *Nikita Borisov* and *Eric Brewer*
- Conversations in the "digital" world should mimic casual real world conversations
- PGP: protect communications. Sign messages and encrypt them.
- Problems: there is a record,



https://xkcd.com/1553/

Why a version 4 of OTR?

- We want deniability: participation, message, online and offline
- We want forward secrecy and post-compromise secrecy
- We want a higher security level
- We want to update the cryptographic primitives
- We want additional protection against transcript decryption in the case of ECC compromise
- We want elliptic curves

OTRv4 implementation

- Implementation in C
- Called ‘libotr-ng’: https://github.com/otrv4/libotr-ng
- Usage of C comes with -free- memory issues:
  - Buffer overflow
  - Memory leaks
  - Free issues: use after free, double free, invalid free
  - Usage of uninitialized memory or garbage data
  - Overlap of src and dst pointers in memcpy
- Why it is an issue?

“Memory leaks are mismanaged memory allocations. They are caused by heap areas that can no longer be freed, due to a lost pointer and are something every programmer using C has to be careful about. These leaks occur because C doesn’t clean up after itself, unlike Java or C# with its inbuilt garbage collector. Memory leaks are hard to find because a program might work just fine for a while and then crash without apparent reason or simply slow down below acceptable levels. Sometimes this might be misconstrued as a hardware problem.”

- *C Basics And Concepts Memory Leaks and Debugging with Valgrind* (2014), Working group scientific computing Department of informatics Faculty of mathematics, informatics and natural sciences University of Hamburg

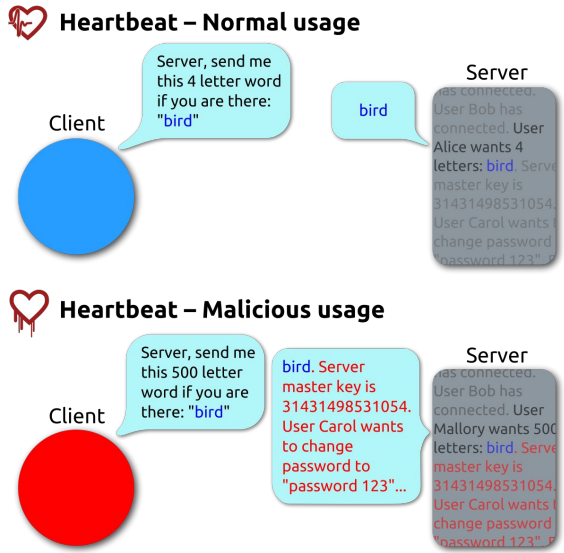
- Leakage of sensitive information: private/secret or message keys
- Memory issues remain dominant (heap out of bounds: read/write. Eg: Microsoft: *Trends, challenge, and shifts in software vulnerability mitigation* (2019) by Matt Miller)

Tips during the project execution

- Use free after malloc (or similar)
- Not work with the original pointer but rather with a copy of it
- Free what has been malloced in a struct
- Handle return references
- Do not access null pointers: remember to malloc
- Usage of valgrind (it has limitations: crashes, false positives on some OS, installation problems. See: https://bugs.kde.org/show_bug.cgi?id=365327)

For cryptography…

- “any computation, and only computation, leaks information” (Micali and Reyzin)
- Public information vs Secret information
- Heartbleed



Problems in some cryptographic code...

- Little to no testing
- Code, sometimes, does not run nor compile
- Code does not run or compile in certain OS or compilers
- Code is difficult to understand
- Code is not clean
- No usage of tools for checking memory issues or related issues

In the OTRv4 library

- We have a CI which tests in 12 machines (gcc and clang). Locally, we test mostly in Linux and MacOS.
- We test with valgrind (memcheck, helgrind, drd), address sanitizers, clang-tidy, splint, ctgrind
- We check coverage, profiling and style
- We send to check to coverity scan
- In the future: fuzzing, taint analysis

```
AX_CFLAGS_GCC_OPTION([-Wall])
AX_CFLAGS_GCC_OPTION([-Wextra])
AX_CFLAGS_GCC_OPTION([-Werror])
AX_CFLAGS_GCC_OPTION([-Wformat])
AX_CFLAGS_GCC_OPTION([-Wno-format-extra-args])
AX_CFLAGS_GCC_OPTION([-Wfatal-errors])
AX_CFLAGS_GCC_OPTION([-Wbad-function-cast])
AX_CFLAGS_GCC_OPTION([-Wdiv-by-zero])
AX_CFLAGS_GCC_OPTION([-Wfloat-equal])
AX_CFLAGS_GCC_OPTION([-Wnested-externs])
AX_CFLAGS_GCC_OPTION([-Wpointer-arith])
AX_CFLAGS_GCC_OPTION([-Wredundant-decls])
AX_CFLAGS_GCC_OPTION([-Wstrict-prototypes])
AX_CFLAGS_GCC_OPTION([-Wlogical-op])
AX_CFLAGS_GCC_OPTION([-Wbad-cast-qual])
AX_CFLAGS_GCC_OPTION([-Wformat-nonliteral])
AX_CFLAGS_GCC_OPTION([-Wbuiltin-memcpy-chk-size])
AX_CFLAGS_GCC_OPTION([-Wfloat-equal])
AX_CFLAGS_GCC_OPTION([-Wundef])
AX_CFLAGS_GCC_OPTION([-Wshadow])
AX_CFLAGS_GCC_OPTION([-Wpointer-arith])
AX_CFLAGS_GCC_OPTION([-Wcast-align])
AX_CFLAGS_GCC_OPTION([-Wmaybe-uninitialized])
AX_CFLAGS_GCC_OPTION([-Wlogicalop])
AX_CFLAGS_GCC_OPTION([-Wno-type-limits])
AX_CFLAGS_GCC_OPTION([-Wnull-dereference])
AX_CFLAGS_GCC_OPTION([-Wwrite-strings])
AX_CFLAGS_GCC_OPTION([-Wswitch-default])
AX_CFLAGS_GCC_OPTION([-Wswitch-enum])
AX_CFLAGS_GCC_OPTION([-Waddress-of-temporary])
AX_CFLAGS_GCC_OPTION([-Warc])
```

Useful tools

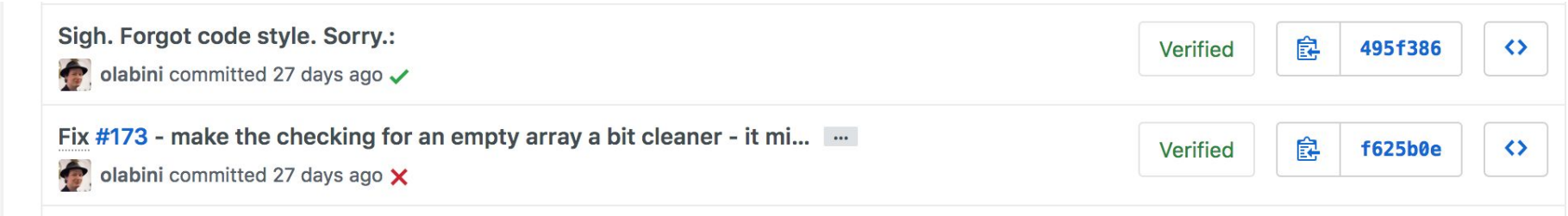
- Address sanitizer:
  - Compile time
  - Bugs are easier to find
- Useful for finding bugs locally in some OS
- Faster than valgrind
- Clearer errors: no repetition, issue is stated in a simple way
  ‘AddressSanitizer: heap-use-after-free on address’
- Limitations: runs with the tests only; there is no coverage of other paths

- Use then: Clang-tidy with the static analyser
- Easier to understand than splint
- Fixes issues in code not tested: free of data unmalloced, unused variables, etc.
- Helps with the style and on the team (onboarding to C)

```
                  ^
/home/travis/build/otrv4/libotr-ng/src/fragment.c:123:9: warning: 1st function call argument is an uninitialized value [clang-analyzer-core.CallAndMessage]
        free(pieces[i]);
        ^
```

Style is important

- Important for clean code: clang-format
- Important to eliminate garbage: unused variables or functions, exposed functions with no reason, ignored return values..
- Usage of one unifying style: clang-format
- Incorporated into the CI

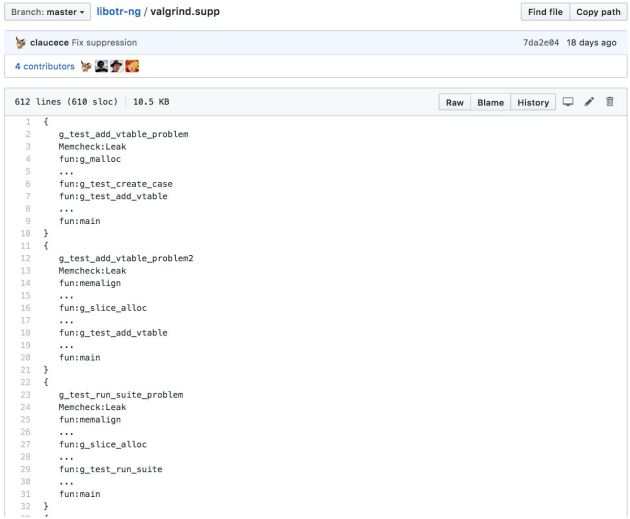


Awful issues

- DH keys were released and later tried to be reused
- People constantly forget to free (clang-tidy: potential memleak)
- Double frees: ‘free too much’
- Uninitialized values

Why is needed?

- These tools are needed for cryptographic libraries as they catch errors that are, sometimes, not seen directly
- Programmers are not perfect
- Valgrind, sometimes, needs a lot of suppressions to run



Branch: master ▾ libotr-ng / valgrind.supp — Find file — Copy path

claucece Fix suppression — 7da2e04 — 18 days ago

4 contributors

612 lines (610 sloc) | 10.5 KB — Raw — Blame — History

```
{
   g_test_add_vtable_problem
   Memcheck:Leak
   fun:g_malloc
   ...
   fun:g_test_create_case
   fun:g_test_add_vtable
   ...
   fun:main
}
{
   g_test_add_vtable_problem2
   Memcheck:Leak
   fun:memalign
   ...
   fun:g_slice_alloc
   ...
   fun:g_test_add_vtable
   ...
   fun:main
}
{
   g_test_run_suite_problem
   Memcheck:Leak
   fun:memalign
   ...
   fun:g_slice_alloc
   ...
   fun:g_test_run_suite
   ...
   fun:main
}
```

Ideas

```
now = time(NULL);
otrng_ecdh_keypair_destroy(manager->our_ecdh);
/* @secret the ecdh keypair will last
   1. for the first generation: until the ratchet is initialized
   2. when receiving a new dh ratchet
*/
if (!otrng_ecdh_keypair_generate(manager->our_ecdh, sym)) {
  otrng_secure_free(sym);
  return OTRNG_ERROR;
}

otrng_secure_free(sym);

manager->last_generated = now;

if (manager->i % 3 == 0) {
  otrng_dh_keypair_destroy(manager->our_dh);

  /* @secret the dh keypair will last
     1. for the first generation: until the ratchet is initialized
     2. when receiving a new dh ratchet
  */
  if (!otrng_dh_keypair_generate(manager->our_dh)) {
    return OTRNG_ERROR;
  }
}
```

References

1. Serebryany, K., Bruening, D., Potapenko, A., Vyukov, D. AddressSanitizer: A Fast Address Sanity Checker, USENIX. Available at: https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf

2. Working group scientific computing Department of informatics Faculty of mathematics, informatics and natural sciences. (2014). C Basics And Concepts Memory Leaks and Debugging with Valgrind, NIST ECC workshop. Available at: https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2014/cgk-14-menck-memory-leaks-report.pdf

Thanks!

Sofía Celi
@cherenkov_d