CWE_CHECKER

Hunting Binary Code Vulnerabilities Across CPU Architectures
Pass The SALT 2019
$whoarewe$

- Thomas Barabosch
  - @tbarabosch 📗/twitter
  - PhD in computer science
  - Binary Code Analyst (*ware)
  - Hobbyist Bug Hunter (*BSD, Router, Hypervisor, ...)

- Nils-Edvin Enkelmann
  - PhD in mathematics
  - Security researcher with focus on binary code analysis
OUTLINE

1. Motivation
2. cwe_checker
3. Case Studies
4. Integration with other tools
5. Future Work
6. Conclusion
MOTIVATION

- **Goal**: Security analysis of closed source firmware
- Bug hunting through reverse engineering is tedious and time-consuming
MOTIVATION

- Many different CPU architectures in the IoT-world
  - x86/x64, PowerPC, MIPS, ARM, ...
- Each CPU-architecture has its own instruction set
  - e.g. x86/x64 alone has hundreds of assembly instructions
- Assembly instructions can have complex side effects
  - What does ADD actually do?

- Working directly on the disassembly does not scale
- **Solution**: build analyses up on intermediate representation language
```c
int main(int argc, char **argv)
{
    int x = argc * argc;
    return argc + x + 42;
}
```

**ARM**
```
mla r3, r0, r0, r0
add r0, r3, #42
bx lr
```

**x86**
```
movl 0x4(%esp), %eax
movl %eax, %edx
imull %eax, %edx
leal 0x2a(%eax,%edx), %eax
retl
```

**Bil IR**
```
R3 := R0 + R0 * R0
R0 := R3 + 0x2A
return LR
```

**Bil IR**
```
EAX := mem[ESP + 4, el]:u32
EDX := EAX
v357 := extend:64[low:32[EDX]] * extend:64[low:32[EAX]]
EDX := low:32[v357]
v358 := mem[ESP, el]:u32
ESP := ESP + 4
return v358
```
Binary Analysis Platform (BAP)

- Reverse engineering and program analysis platform
  - Focus: binary code
- Disassembles and lifts to Intermediate Representation (BIL)
  - Lifters available for x86, x86-64, ARM, PowerPC, MIPS
- BIL comprises less than 40 instructions
- Written in Ocaml
  - Bindings for C, Python, Rust
- https://github.com/BinaryAnalysisPlatform/bap
CWE_CHECKER
cwe_checker – Overview

- Detection of CWEs (Common Weakness Enumeration) through heuristics
  - Based on top of BAP
  - Inspired by ClangAnalyzer et al.

- Architecture-independent through use of BAP’s IR

- Modular structure
  - 13 CWE-modules using static analysis
  - 4 CWE-modules using symbolic execution
  - Easy to add YOUR custom check

- Easy deployment through Docker or Opam
cwe_checker – Architecture

Reports

Lifting

Binary Analysis Platform (BAP)

Modules

CWE-215
CWE-243
CWE-332
CWE-367
[…]

MIPS ELF
ARM ELF
X86 ELF
cwe_checker – A Running Example

```c
#include <stdlib.h>
#include <stdio.h>

void main(int argc, char** argv)
{
    int* data = malloc(200 * argc);
    printf("%i", data[0]);
    free(data);
}
```
cwe_checker – Disassembly of Targets

<table>
<thead>
<tr>
<th>Modules</th>
<th>CWE-215</th>
<th>CWE-243</th>
<th>CWE-332</th>
<th>CWE-367</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwe_checker</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Binary Analysis Platform (BAP)

<table>
<thead>
<tr>
<th></th>
<th>MIPS</th>
<th>ARM</th>
<th>X86</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF</td>
<td>ELF</td>
<td>ELF</td>
<td></td>
</tr>
</tbody>
</table>

```
10374: <main>
10374:
10374: 00 01 80 e0 add r0, r0, r0, lsl #2
10378: 00 01 80 e0 add r0, r0, r0, lsl #2
1037c: 10 40 2d e9 push {r4, lr}
10380: 80 01 a0 e1 lsl r0, r0, #3
10384: eb ff ff eb bl #-0x54
10388:
10388: 00 40 a0 e1 mov r4, r0
1038c: 00 20 90 e5 ldr r2, [r0]
10390: 10 10 9f e5 ldr r1, [pc, #0x10]
10394: 01 00 a0 e3 mov r0, #1
10398: ef ff ff eb bl #-0x44
1039c:
1039c: 04 00 a0 e1 mov r0, r4
103a0: 10 40 bd e8 pop {r4, lr}
103a4: e0 ff fe ea b #-0x80
1032c:
1032c: 00 c6 8f e2 add r12, pc, #0, #12
10330: 10 ca 8c e2 add r12, r12, #16, #20
10334: d8 fc bc e5 ldr pc, [r12, #0xcdd8]!
```
cwe_checker – Lifting to BIL

Binary Analysis Platform (BAP)

MIPS ELF ARM ELF X86 ELF

- CWE-215
- CWE-243
- CWE-332
- CWE-367

CWE_Checker

- Modules

- MIPS ELF
- ARM ELF
- X86 ELF

© Fraunhofer FKIE
cwe_checker – A (partial) report

<table>
<thead>
<tr>
<th>cwe_checker</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CWE-215</td>
</tr>
<tr>
<td></td>
<td>CWE-243</td>
</tr>
<tr>
<td></td>
<td>CWE-332</td>
</tr>
<tr>
<td></td>
<td>CWE-367</td>
</tr>
</tbody>
</table>

Binary Analysis Platform (BAP)

- MIPS ELF
- ARM ELF
- X86 ELF

2019-06-28 10:50:24.970 WARN : [CWE190] {0.1} (Integer Overflow or Wraparound) Potential overflow due to multiplication 0x10374:32u (malloc).
2019-06-28 10:50:24.973 WARN : [CWE476] {0.2} (NULL Pointer Dereference) There is no check if the return value is NULL at 0x10374:32u (@malloc).
cwe_checker – A Running Example

```c
#include <stdlib.h>
#include <stdio.h>

void main(int argc, char** argv)
{
    int* data = malloc(200 * argc);
    printf("%i", data[0]);
    free(data);
}
```
(Some) Pure Static Analysis Modules

- CWE-190: Integer Overflow
- CWE-215: Information Exposure Through Debug Information
- CWE-332: Insufficient Entropy in PRNG
- CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition
- CWE-476: NULL Pointer Dereference
- CWE-676: Use of Potentially Dangerous Function
(Even More) Pure Static Analysis Modules

- CWE-248: Uncaught Exception
- CWE-426: Untrusted Search Path
- CWE-457: Use of Uninitialized Variable
- CWE-467: Use of sizeof() on a Pointer Type
- CWE-560: Use of umask() with chmod-style Argument
- CWE-782: Exposed IOCTL with Insufficient Access Control
Symbolic Execution with BAP’s Primus

- Static program analysis technique to explore program execution paths
  - Symbolic values instead of concrete values
  - Outputs symbolic expressions
- General issue: symbolic execution is time consuming (path explosion)
- Primus is BAP’s framework for symbolic execution
- Primus is extendable via Primus LISP
  - Library function stubs (e.g. malloc)
  - Implementation of security checks
Symbolic Execution-based Modules

- CWE-215: Out-of-bounds Read
- CWE-415: Double Free
- CWE-416: Use After Free
- CWE-787: Out-of-bounds Write
CASE STUDIES
CWE-190: Integer Overflow or Wraparound

- Multiplications + Memory Operations especially vulnerable

- Check for multiplication instructions before calls to `malloc`
  - Assumption: If in basic block right before the call ⇒ no overflow check!

- Checked functions: `malloc`, `xmalloc`, `realloc`
  - Users can add functions

- Future improvement: use data flow analysis
  - to see if attacker can control input / no sanitization at all
CWE-190: Integer Overflow or Wrapping around

```
__n = iParam3 * iParam2;
pvVar1 = realloc(ppvParm4, (int)(ppvParm4[1] + __n + 1);
```
CWE-476: Possible NULL Pointer Dereference

- Many functions may return NULL on failure (e.g. malloc, open, …)
- Therefore: return value must be checked!

- Via Data Flow Analysis
  - Taint return register
  - Taint registers whose value is computed using a tainted register
  - Search for execution paths where a tainted register is used for memory access before a tainted register is checked
CWE-476: Possible NULL Pointer Dereference

```assembly
0000c360 00 48 2d e9           stmdb     sp!,{ r11 lr }
0000c364 04 b0 8d e2           add       r11,sp,#0x4
0000c368 08 d0 4d e2           sub       sp,sp,#0x8
0000c36c 0a 0b a0 e3           mov       r0,#0x2800
0000c370 7f f5 ff eb           bl        malloc
0000c374 00 30 a0 e1           cpy       r3,r0
0000c378 08 30 0b e5           str       r3,[r11,#local_c]
0000c37c 08 00 1b e5           ldr       r0,[r11,#local_c]
0000c380 00 10 a0 e3           mov       r1,#0x0
0000c384 0a 2b a0 e3           mov       r2,#0x2800
0000c388 e5 f5 ff eb           bl        memset
0000c38c 08 00 1b e5           ldr       r0,[r11,#local_c]
```

```c
8 |  _s = malloc(0x2800);
9 |  memset(_s,0,0x2800);
```
CWE-476: Possible NULL Pointer Dereference
INTEGRATION WITH OTHER TOOLS
### Analysis for TP-Link Archer C9 V5 - 180423

**UID:** 267e0020003e2e449e6218905e9f77f9f61715f8a2b5d0a94b4e97b8440ddf1d_277144001

#### General
- **device-name:** Archer C9 V5
- **vendor:** TP-Link
- **device-class:** router
- **version:** 180423
- **release date:** 2018-05-01
- **file-name:** Archer C9E_V5_180423.zip
- **virtual path:** TP-Link Archer C9 V5 - 180423 (router)
- **file-size:** 20.43 MB (27,714,681 bytes)
- **Analysis Tags:** Private Key Found, New Kernel Stack
- **file-type:** Zip archive data, at least v1.0 to extract

#### File Tree
- Archer C9E_V5_180423.zip (20.43 MB)

### Showing Analysis: cwe_checker

#### Time of Analysis
- **2018-07-12 13:30:43**

#### Plugin Version
- **0.3.2**

#### Overview of CWE warnings

#### Summary Including Results of Included Files

<table>
<thead>
<tr>
<th>Item count</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CWE243] (The program utilizes chroot without dropping privileges and/or changing the directory)</td>
<td>show files</td>
</tr>
<tr>
<td>[CWE332] (Insufficient Entropy in PRNG)</td>
<td>show files</td>
</tr>
<tr>
<td>[CWE467] (Use of sizeof on a Pointer Type)</td>
<td>show files</td>
</tr>
<tr>
<td>[CWE476] (NULL Pointer Dereference)</td>
<td>show files</td>
</tr>
<tr>
<td>[CWE674] (Use of Potentially Dangerous Function)</td>
<td>show files</td>
</tr>
<tr>
<td>[CWE676] (Use of Potentially Dangerous Function)</td>
<td>show files</td>
</tr>
</tbody>
</table>
cwe_checker in FACT 2/2

Showing Analysis: cwe_checker


Plugin Version: 0.3.2

Overview of CWE warnings:
- [CWE215] (Information Exposure Through Debug Information)
- [CWE467] (Use of sizeof on a Pointer Type)
- [CWE476] (NULL Pointer Dereference)
- [CWE676] (Use of Potentially Dangerous Function)

[CWE215] (Information Exposure Through Debug Information) (0.1)
- CU /x2ttpd
- CU /tpyc
- CU /misc
- CU /control
- CU /avpc
- CU /calc
- CU /networkc
- CU /avpcendc
- CU /schedulerc
- CU /filec
- CU /aaac
- CU /nccc

[CWE467] (Use of sizeof on a Pointer Type) (0.1)
- sizeof on pointer at 0xb640 (stmcmp)

[CWE476] (NULL Pointer Dereference) (0.1)
- There is no check if the return value is NULL at 0x17d0c/000038fc (fgets)
- There is no check if the return value is NULL at 0x179e4/0000555a (malloc)
- There is no check if the return value is NULL at 0xbb60/00005a53 (malloc)
- There is no check if the return value is NULL at 0xae90/000082ca (malloc)
- There is no check if the return value is NULL at 0x17404/00006498 (fgets)
- There is no check if the return value is NULL at 0x17b5c/0000672 (malloc)
Visualize cwe_checker Results with IDA Pro
LET’S WRAP IT UP
Current Limitations

- It’s static analysis: false positives / false negatives
- Some checks are based on strong assumptions to simplify the analysis
- Symbolic execution is slow (especially on bigger binaries)
Future Work

- Add more checks and improve correctness of older checks
- Improve pointer analysis
  - Memory management checks via static analysis
  - Maybe foundation of fully fledged type analysis
- Tool integration
  - Improve IDA Pro support (start from within IDA)
  - Add support for Ghidra (visualize results, start from within Ghidra)
Conclusion

- cwe_checker is a static analysis tool to heuristically detect bug classes
- Thanks to its foundation BAP, it analyzes binaries of many architectures
  - Including x86/x64, ARM, PPC, MIPS, …
- cwe_checker comprises a wide range of checks (currently 15+)
  - from simple “pattern matching” to data flow analysis-based checks
- Tool integration is a major concern: FACT + IDA Pro
GET IT NOW!

- [https://github.com/fkie-cad/cwe_checker](https://github.com/fkie-cad/cwe_checker)
- Release: 0.2
- Ask for free stickers!

© Fraunhofer