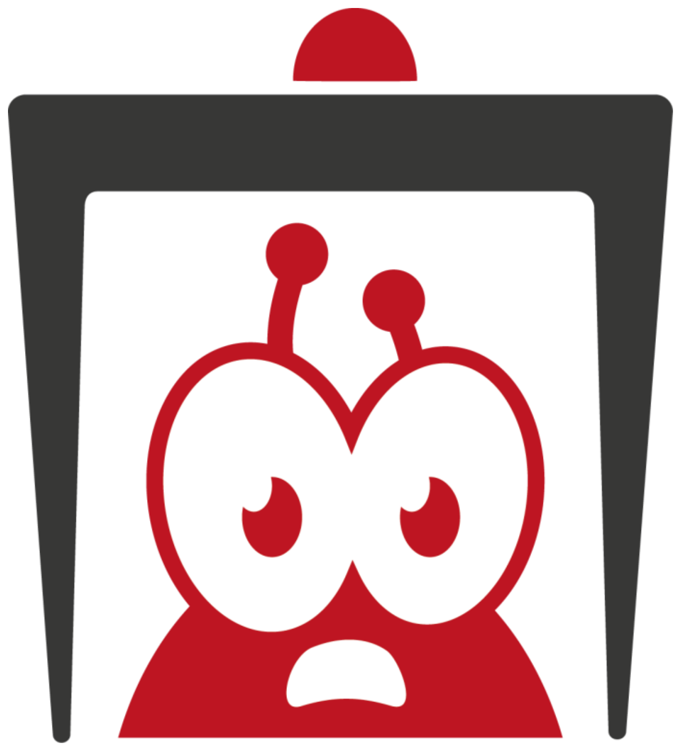
CWE_CHECKER

Hunting Binary Code Vulnerabilities Across CPU Architectures

Pass The SALT 2019



Fraunhofer
FKIE

$whoarewe

- Thomas Barabosch
  - @tbarabosch
  - 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

⇨ Automation!

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
```

Bil IR

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

x86

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

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]
EAX := low:32[low:32[EAX] + low:32[EDX] + 0x2A]
v358 := mem[ESP, el]:u32
ESP := ESP + 4
return v358
```

Binary Analysis Platform (BAP)

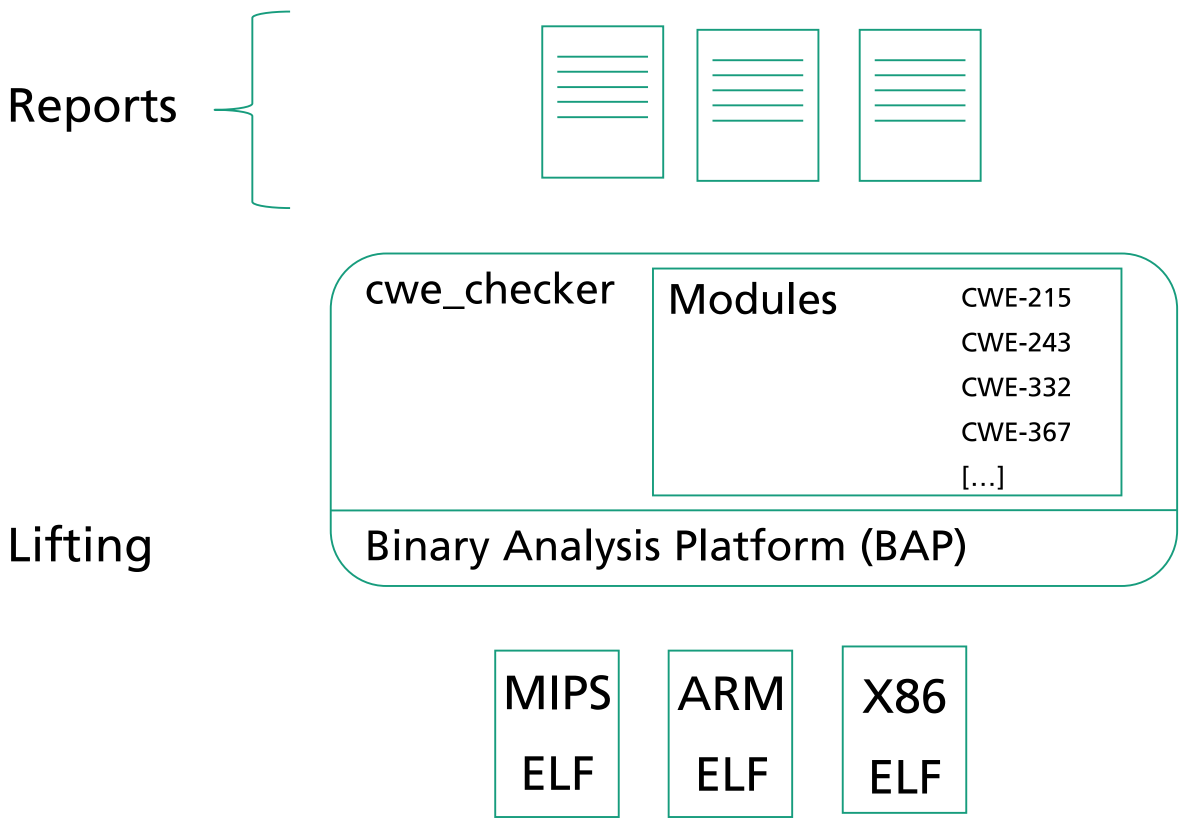
- Reverse engineering and program analysis platform
  - Focus: binary code
- Disassembles and lifts to Intermediate Representation (BIL)
  - Lifters available for x86, x86-64, ARM, PowerPC, MIPS
- BIL comprises less than 40 instructions
- Written in Ocaml
  - Bindings for C, Python, Rust
- https://github.com/BinaryAnalysisPlatform/bap

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

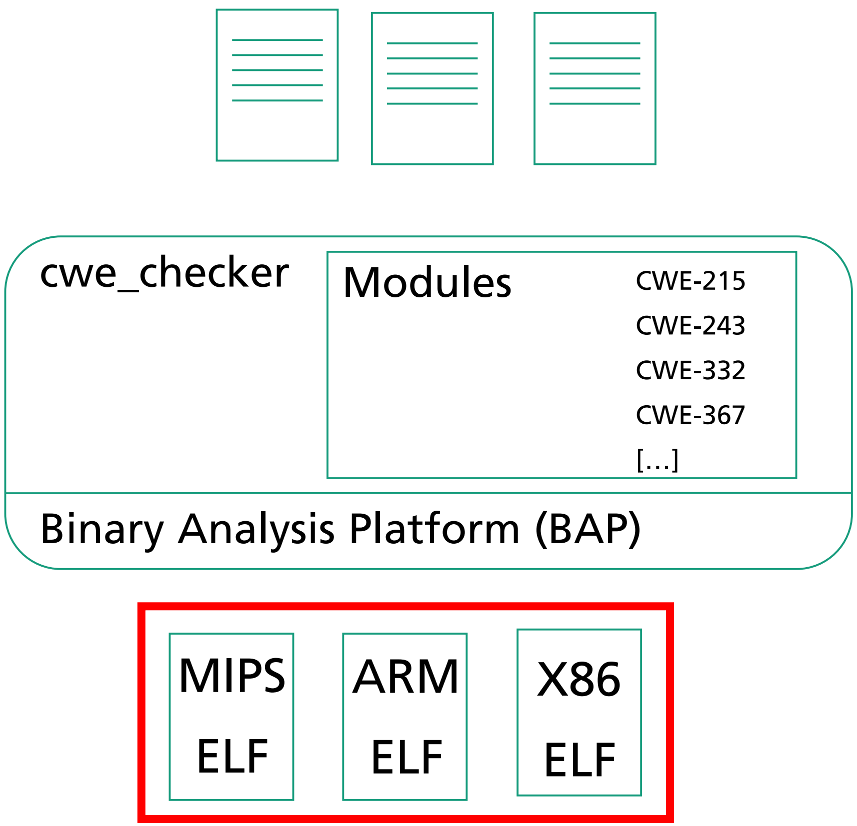
cwe_checker – Architecture



cwe_checker – A Running Example

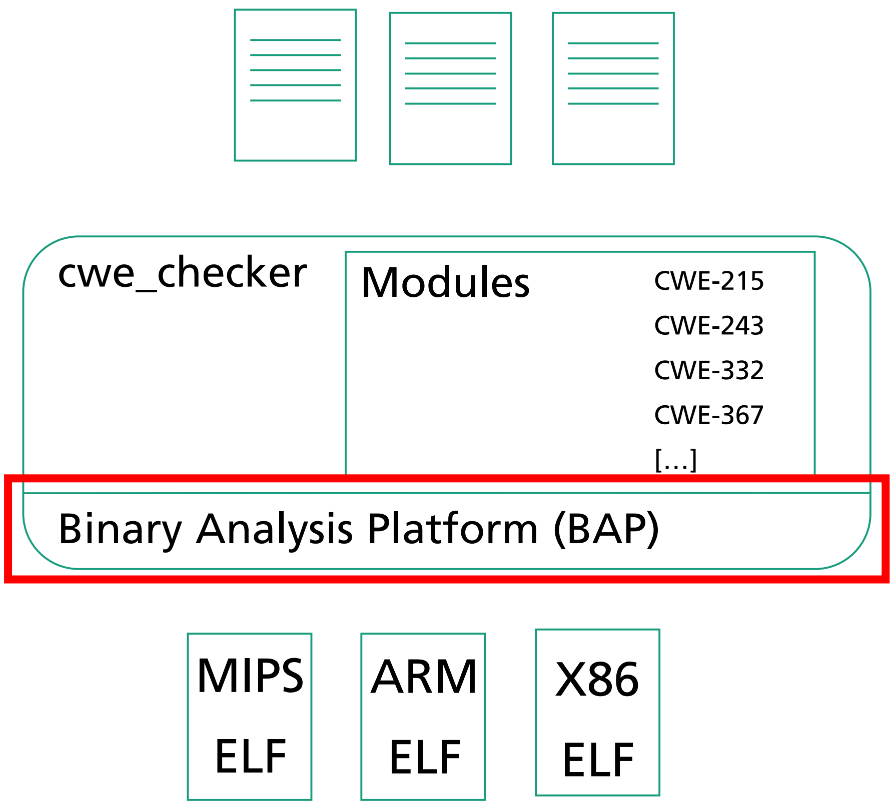
```c
 #include <stdlib.h>
 #include <stdio.h>

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

cwe_checker – Disassembly of Targets



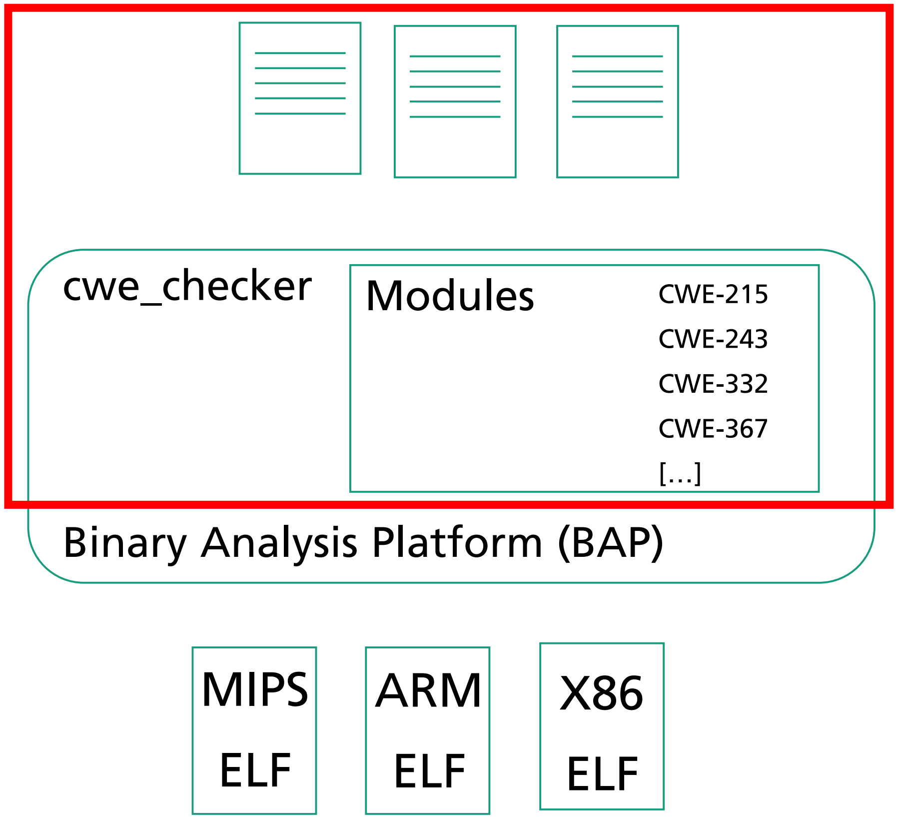

```
10374: <main>
10374:
10374: 00 01 80 e0   add r0, r0, r0, lsl #2
10378: 00 01 80 e0   add r0, r0, r0, lsl #2
1037c: 10 40 2d e9   push {r4, lr}
10380: 80 01 a0 e1   lsl r0, r0, #3
10384: eb ff ff eb   bl #-0x54
10388:
10388: 00 40 a0 e1   mov r4, r0
1038c: 00 20 90 e5   ldr r2, [r0]
10390: 10 10 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 ff 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, #0xcd8]!
```

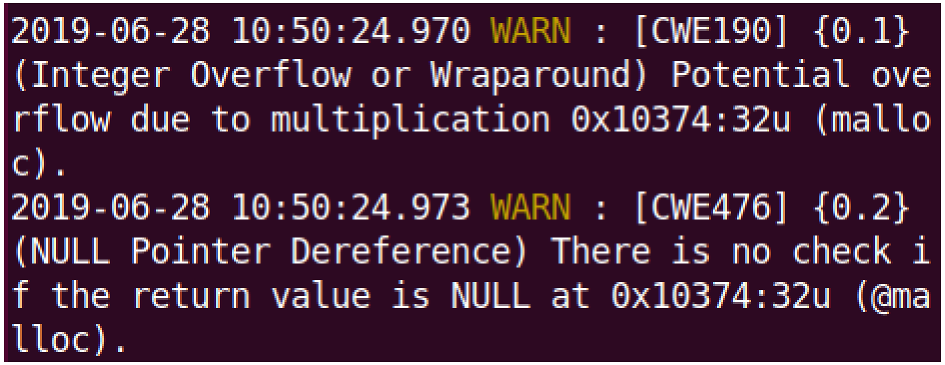
cwe_checker – Lifting to BIL




```
000000e9: sub main(main_argc, main_argv, main_result)
00000123: main_argc :: in u32 = R0
00000124: main_argv :: in out u32 = R1
00000125: main_result :: out u32 = R0
000000bf:
000000c0: v370 := SP
000000c1: mem := mem with [v370 + 0xFFFFFFFC, el]:u32 <- LR
000000c2: mem := mem with [v370 + 0xFFFFFFF8, el]:u32 <- R11
000000c3: SP := SP - 8
000000c4: R11 := SP + 4
000000c5: SP := SP - 0x10
000000c6: mem := mem with [R11 + 0xFFFFFFF0, el]:u32 <- R0
000000c7: mem := mem with [R11 + 0xFFFFFFEC, el]:u32 <- R1
000000c8: R2 := mem[R11 + 0xFFFFFFF0, el]:u32
000000c9: R3 := R2
000000ca: v381 := R3
000000cb: R3 := v381 << 2
000000cc: R3 := R3 + R2
000000cd: v385 := R3
000000ce: R2 := v385 << 2
000000cf: R3 := R3 + R2
000000d0: v389 := R3
000000d1: R3 := v389 << 3
000000d2: R0 := R3
000000d3: LR := 0x10498
000000d4: call @malloc with return %000000d5

000000d5:
000000d6: R3 := R0
000000d7: mem := mem with [R11 + 0xFFFFFFF8, el]:u32 <- R3
000000d8: R3 := mem[R11 + 0xFFFFFFF8, el]:u32
000000d9: R3 := mem[R3, el]:u32
000000da: R1 := R3
000000db: R0 := mem[0x104C8, el]:u32
000000dc: LR := 0x104B4
000000dd: call @printf with return %000000de

000000de:
000000df: R0 := mem[R11 + 0xFFFFFFF8, el]:u32
000000e0: LR := 0x104BC
000000e1: call @free with return %000000e2
```

cwe_checker – A (partial) report





```
2019-06-28 10:50:24.970 WARN : [CWE190] {0.1}
(Integer Overflow or Wraparound) Potential ove
rflow due to multiplication 0x10374:32u (mallo
c).
2019-06-28 10:50:24.973 WARN : [CWE476] {0.2}
(NULL Pointer Dereference) There is no check i
f the return value is NULL at 0x10374:32u (@ma
lloc).
```

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-243: Creation of chroot Jail Without Changing Working Directory
- 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 Wraparound

```
                             FUN_0000f140
0000f140 00 48 2d e9     stmdb      sp!,{ r11 lr }
0000f144 04 b0 8d e2     add        r11,sp,#0x4
0000f148 18 d0 4d e2     sub        sp,sp,#0x18
0000f14c 10 00 0b e5     str        r0,[r11,#local_14]
0000f150 14 10 0b e5     str        r1,[r11,#local_18]
0000f154 18 20 0b e5     str        r2,[r11,#local_1c]
0000f158 1c 30 0b e5     str        r3,[r11,#local_20]
0000f15c 14 30 1b e5     ldr        r3,[r11,#local_18]
0000f160 18 20 1b e5     ldr        r2,[r11,#local_1c]
0000f164 92 03 03 e0     mul        r3,r2,r3
0000f168 08 30 0b e5     str        r3,[r11,#local_c]
0000f16c 1c 30 1b e5     ldr        r3,[r11,#local_20]
0000f170 0c 30 0b e5     str        r3,[r11,#local_10]
0000f174 0c 30 1b e5     ldr        r3,[r11,#local_10]
0000f178 00 20 93 e5     ldr        r2,[r3,#0x0]
0000f17c 0c 30 1b e5     ldr        r3,[r11,#local_10]
0000f180 04 10 93 e5     ldr        r1,[r3,#0x4]
0000f184 08 30 1b e5     ldr        r3,[r11,#local_c]
0000f188 03 30 81 e0     add        r3,r1,r3
0000f18c 01 30 83 e2     add        r3,r3,#0x1
0000f190 02 00 a0 e1     cpy        r0,r2
0000f194 03 10 a0 e1     cpy        r1,r3
0000f198 8b e9 ff eb     bl         realloc
```

```
__n = iParm3 * iParm2;
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!

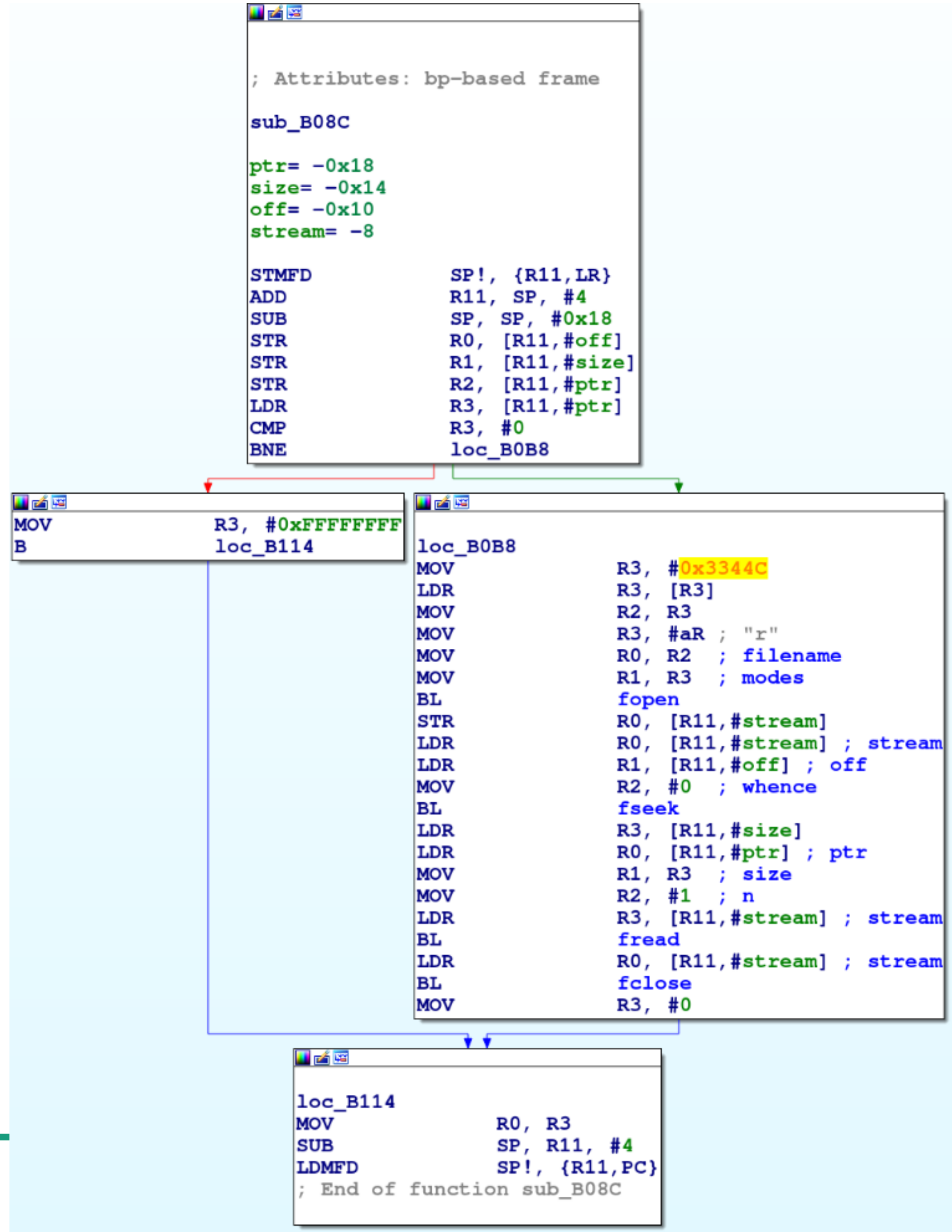
- Via Data Flow Analysis
  - Taint return register
  - Taint registers whose value is computed using a tainted register
  - Search for execution paths where a tainted register is used for memory access before a tainted register is checked

CWE-476: Possible NULL Pointer Dereference

```
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]
```

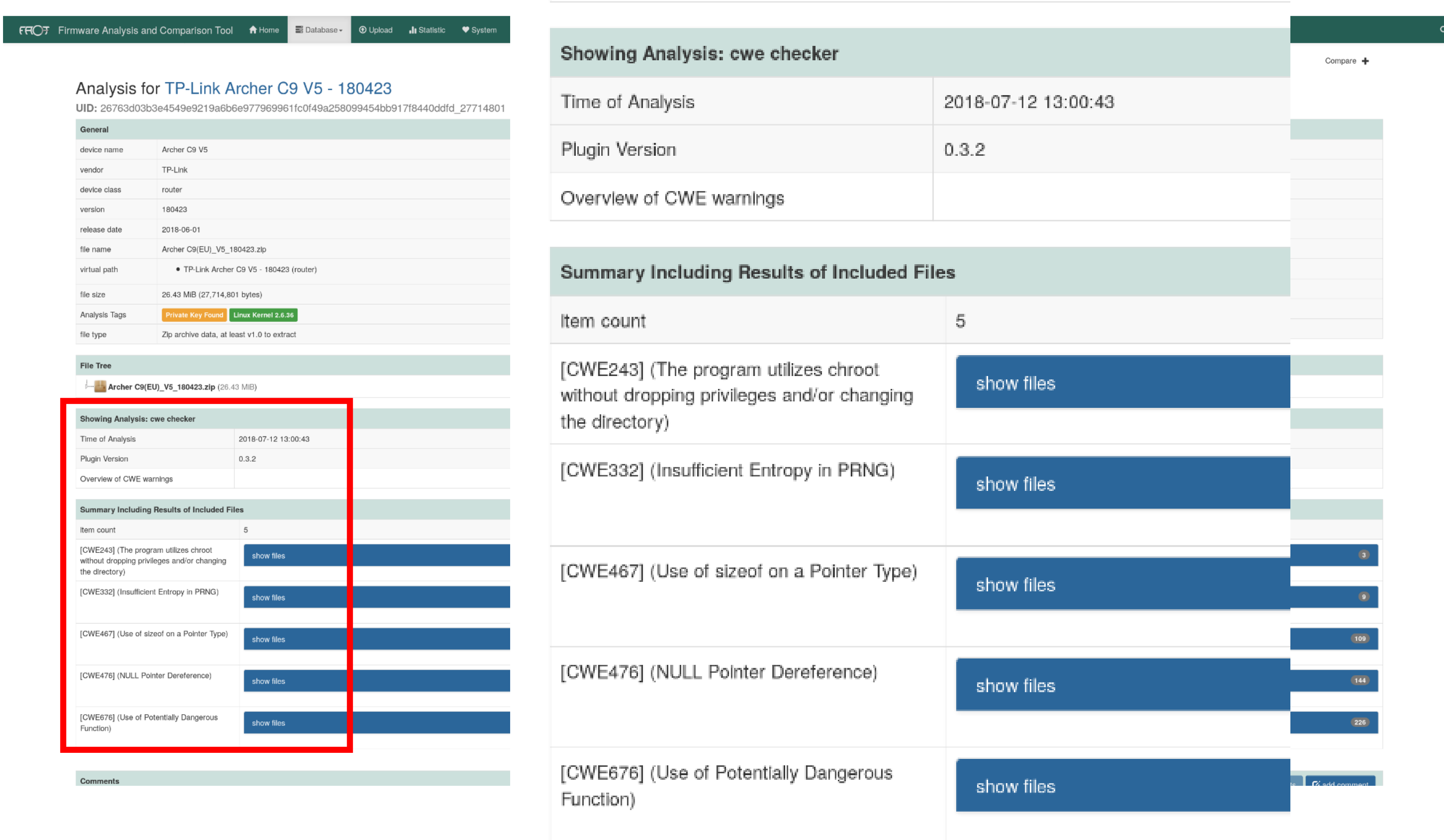
```
  __s = malloc(0x2800);
  memset(__s,0,0x2800);
```

CWE-476: Possible NULL Pointer Dereference



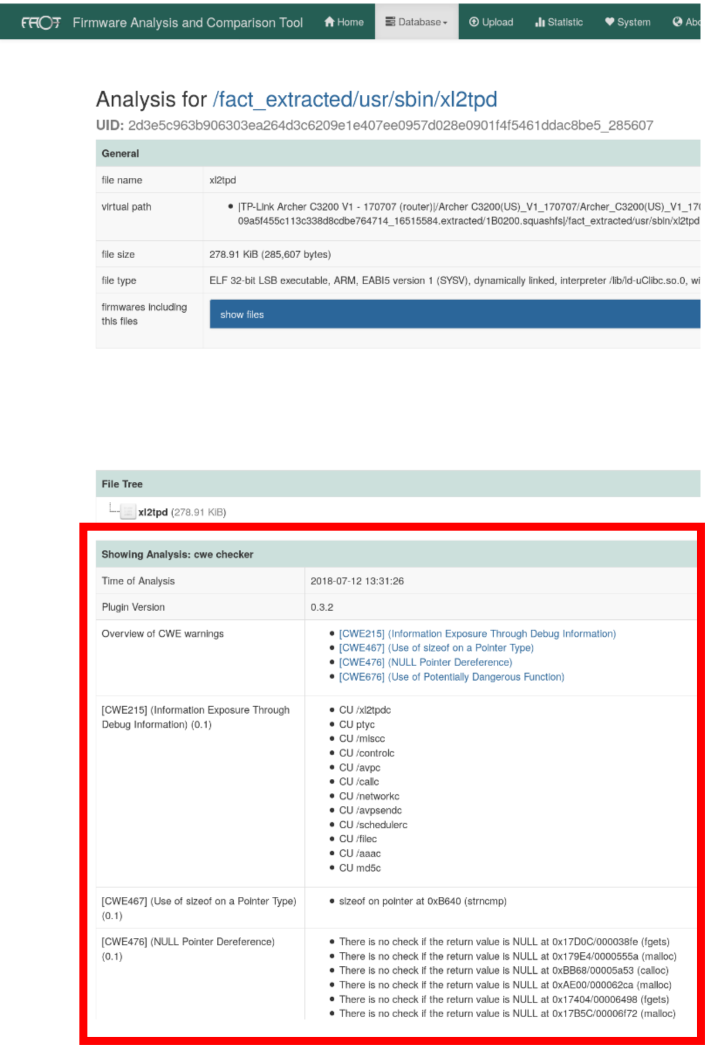
INTEGRATION WITH OTHER TOOLS

cwe_checker in FACT 1/2



Fraunhofer FKIE

cwe_checker in FACT 2/2



Showing Analysis: cwe checker	
Time of Analysis	2018-07-12 13:31:26
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 /xl2tpdc • CU ptyc • CU /miscc • CU /controlc • CU /avpc • CU /callc • CU /networkc • CU /avpsendc • CU /schedulerc • CU /filec • CU /aaac • CU md5c
[CWE467] (Use of sizeof on a Pointer Type) (0.1)	• sizeof on pointer at 0xB640 (strncmp)
[CWE476] (NULL Pointer Dereference) (0.1)	• There is no check if the return value is NULL at 0x17D0C/000038fe (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 0xBB68/00005a53 (calloc) • There is no check if the return value is NULL at 0xAE00/000062ca (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/00006f72 (malloc)

Fraunhofer FKIE

Visualize cwe_checker Results with IDA Pro

```
loc_225E0               ; [CWE476] (NULL Pointer Dereference)
LDR                 R3, [R11,#var_2A0]
LDR                 R3, [R3,#4]
ADD                 R2, R3, #1
LDR                 R3, [R11,#var_2A0]
STR                 R2, [R3,#4]
LDR                 R3, [R11,#var_2A0]
LDR                 R3, [R3,#4]
SUB                 R3, R3, #1
STR                 R3, [R11,#var_20]
LDR                 R3, [R11,#var_44] ; [CWE457] (Use of Uninitialized Variable)
STR                 R3, [R11,#var_24]
LDR                 R3, [R11,#var_2A0]
LDR                 R2, [R3,#0xC]
LDR                 R3, [R11,#var_20]
MOV                 R3, R3,LSL#2
ADD                 R4, R2, R3
MOV                 R0, #0x20 ; size
BL                  malloc
MOV                 R3, R0  ; [CWE676] (Use of Potentially Dangerous Function)
STR                 R3, [R4]
LDR                 R3, [R11,#var_2A0]
LDR                 R2, [R3,#0xC]
LDR                 R3, [R11,#var_20]
MOV                 R3, R3,LSL#2
ADD                 R3, R2, R3
LDR                 R3, [R3]
ADD                 R2, R3, #4
MOV                 R3, #aLu ; "%lu"
MOV                 R0, R2  ; s
MOV                 R1, #0x40 ; maxlen
MOV                 R2, R3  ; format
LDR                 R3, [R11,#var_24]
BL                  snprintf
LDR                 R3, [R11,#var_2A0] ; [CWE476] (NULL Pointer Dereference)
LDR                 R2, [R3,#0xC]
LDR                 R3, [R11,#var_20]
MOV                 R3, R3,LSL#2
ADD                 R3, R2, R3
LDR                 R4, [R3]
LDR                 R3, [R11,#nmemb]
MOV                 R0, R3  ; nmemb
MOV                 R1, #1  ; size
BL                  calloc
MOV                 R3, R0  ; [CWE676] (Use of Potentially Dangerous Function)
STR                 R3, [R4]
LDR                 R3, [R11,#var_2A0]
LDR                 R2, [R3,#0xC]
LDR                 R3, [R11,#var_20]
MOV                 R3, R3,LSL#2
ADD                 R3, R2, R3
LDR                 R3, [R3]
LDR                 R2, [R3]
MOV                 R3, #aS_3 ; "%s"
LDR                 R1, [R11,#var_14]
ADD                 R12, R1, #0x13
MOV                 R0, R2  ; s
MOV                 R1, #0x200 ; maxlen
MOV                 R2, R3  ; format
MOV                 R3, R12
BL                  snprintf
LDR                 R3, [R11,#var_2A0]
LDR                 R2, [R3,#0xC]
LDR                 R3, [R11,#var_20]
MOV                 R3, R3,LSL#2
ADD                 R3, R2, R3
LDR                 R1, [R3]
LDRD                R2, [R11,#var_64] ; [CWE457] (Use of Uninitialized Variable)
STRD                R2, [R1,#0x18]
B                   loc_22718
```

(0,1281) 0001A5E0 00000000000225E0: sub_22298:loc_225E0 (Synchronized with Hex View-1)

Fraunhofer FKIE

LET'S WRAP IT UP

Current Limitations

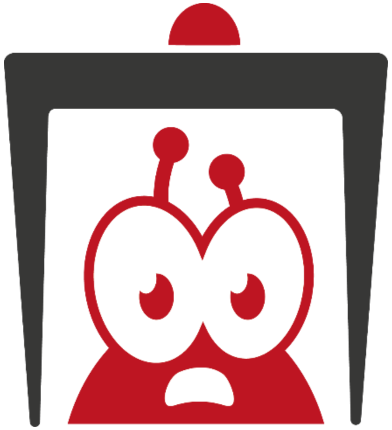
- It's static analysis: false positives / false negatives
- Some checks are based on strong assumptions to simplify the analysis
- Symbolic execution is slow (especially on bigger binaries)

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
- Release: 0.2
- Ask for free stickers!