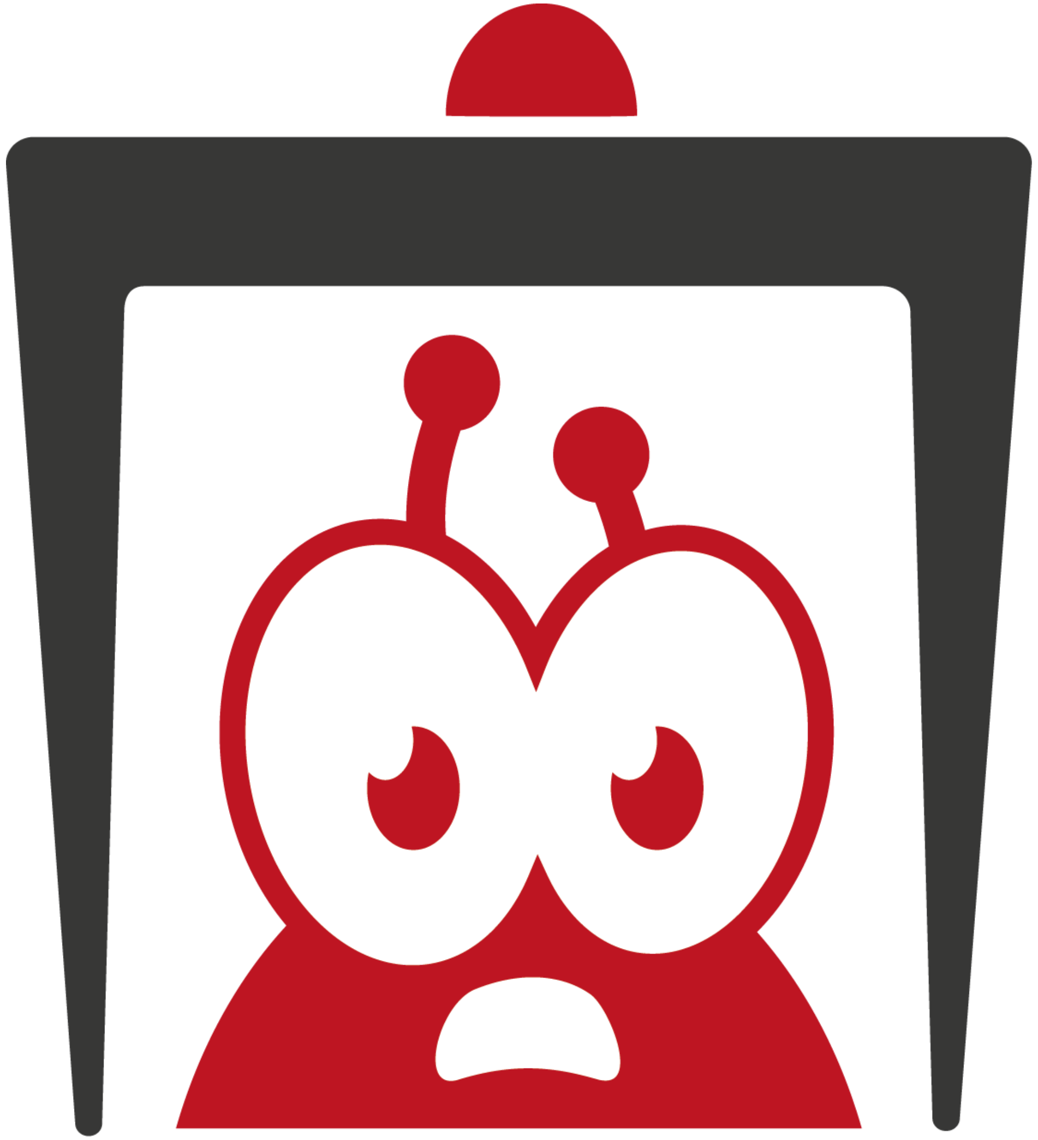
black hat®
USA 2019

AUGUST 3-8, 2019
MANDALAY BAY / LAS VEGAS

cwe_checker

cwe_checker

Hunting Binary Code Vulnerabilities
Across CPU Architectures

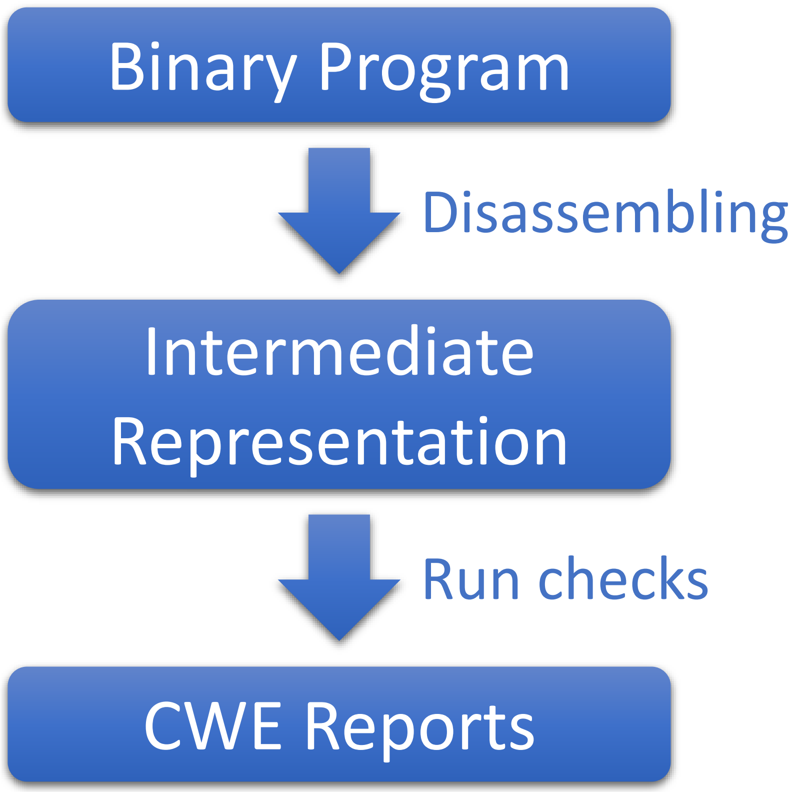


Challenges of Bug Hunting in the IoT World

- Bug hunting through reverse engineering is time consuming and tedious
  - Firmwares can be large → impossible to reverse everything by hand
- Many different CPU architectures
  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
  e.g. setting CPU flags

cwe_checker - Overview

- Automating the process of finding vulnerable code patterns, categorization via CWE (common weakness enumeration) numbers
- Based on Binary Analysis Platform (BAP)
- Using BAP‘s intermediate representation to achieve CPU-architecture independence
- Modular structure
  - 13 checks using static analysis
  - 4 checks using symbolic execution
- Easy Deployment through Docker or Opam

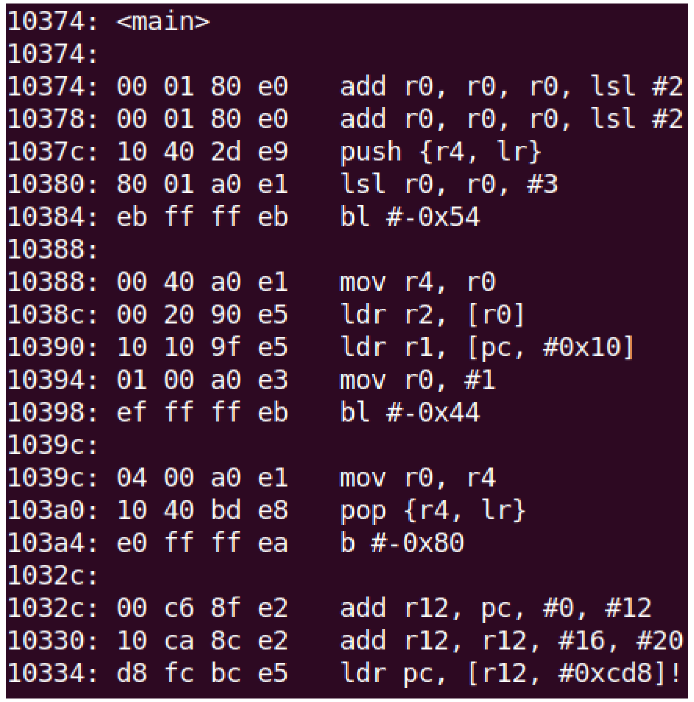


cwe_checker - Example

Binary Program

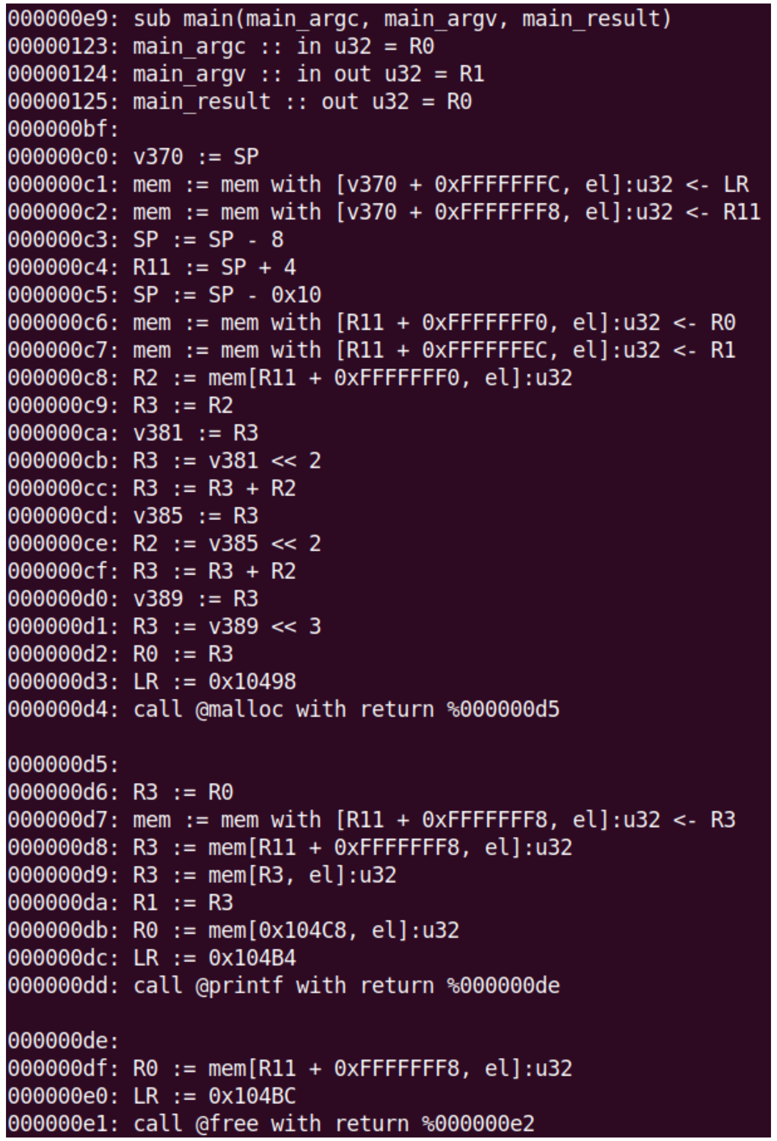
```c
#include <stdlib.h>
#include <stdio.h>

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



```
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]!
```

Disassemble & Lift to IR

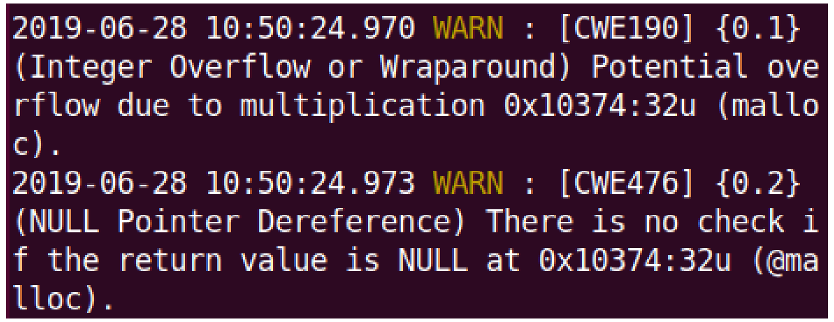


```
000000e9: sub main(main_argc, main_argv, main_result)
00000123: main_argc :: in u32 = R0
00000124: main_argv :: in out u32 = R1
00000125: main_result :: out u32 = R0
000000bf:
000000c0: v370 := SP
000000c1: mem := mem with [v370 + 0xFFFFFFFC, el]:u32 <- LR
000000c2: mem := mem with [v370 + 0xFFFFFFF8, 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
```

Run cwe_checker modules



```
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 – Some Static Analysis Modules

- CWE 190: Integer Overflow
- CWE 332: Insufficient Entropy in PRNG
- CWE 426: Untrusted Search Path
- CWE 467: Use of *sizeof()* on a Pointer Type
- CWE 476: NULL Pointer Dereference
- CWE 560: Use of *umask()* with chmod-style arguments
- CWE 676: Use of Potentially Dangerous Function

And many more!

CWE-476: Possible NULL Pointer Dereference

- Many functions may return NULL on failure (e.g. malloc, open, etc.)
  → Return values must be checked!

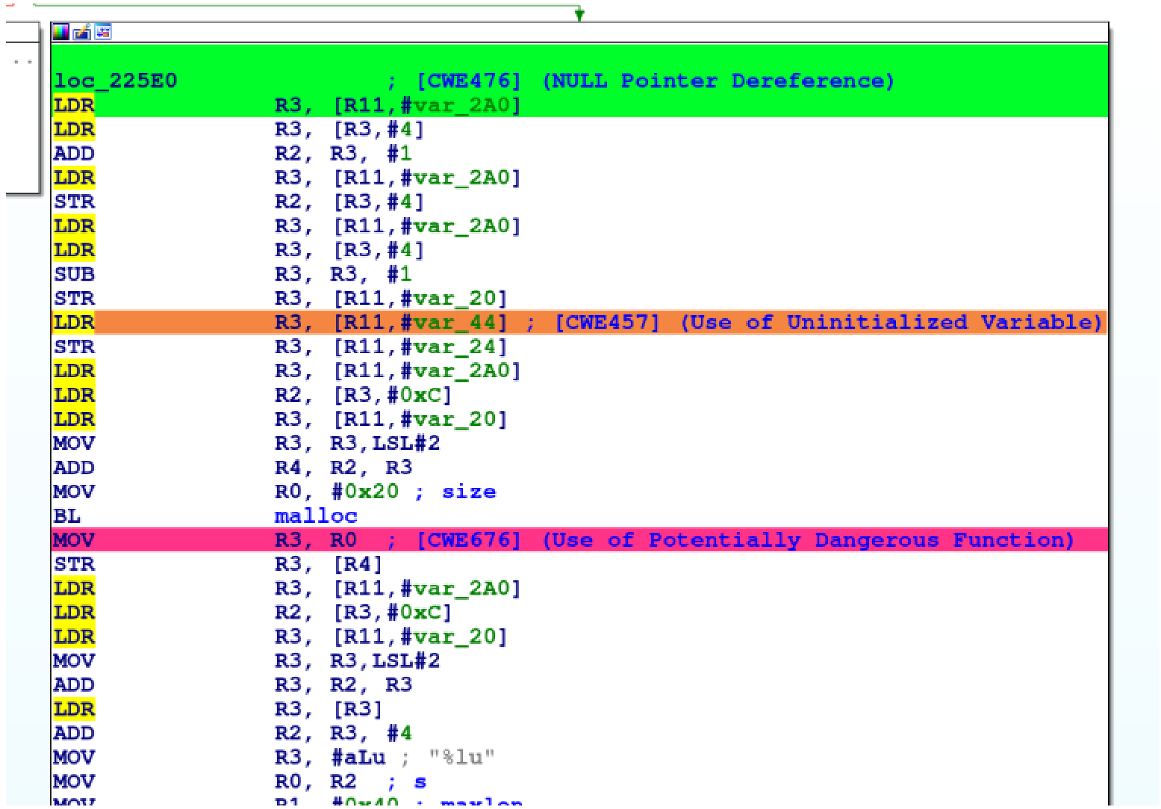
- Via DataFlow Analysis:
  - Unchecked return values are tainted
  - Check of a tainted value → remove taint
  - Memory access through a tainted value → report possible CWE hit

CWE-476: Possible NULL Pointer Dereference

DEMO

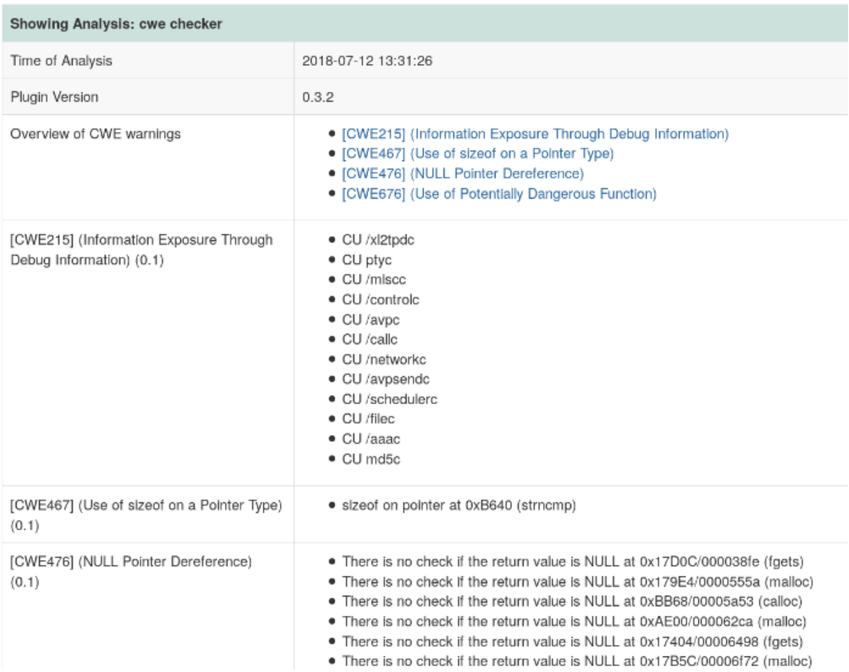
Integration into Other Tools

Visualize results in 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
```

Integration into FACT



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)

Ghidra integration coming soon!

Conclusion

- cwe_checker is a tool to heuristically detect bug classes
- Thanks to its foundation on BAP it is able analyze binaries of many architectures including x86/x64, PowerPC, MIPS, ARM
- Currently over 15+ checks
- Mostly based on static analysis
  → Beware of false positives/negatives
- Easy to add your own check!
- Tool Integration is a mayor concern:
  FACT & IDA Pro (and Ghidra planned)

Get it now!
https://github.com/fkie-cad/cwe_checker
LGPL 3.0 License

cwe checker