black hat ARSENAL 2022

cwe_checker

Architecture-Independent Binary Vulnerability Analysis black hat ARSENAL 2022

#Who Am I
Nils Enkelmann
IT Security Researcher
Fraunhofer FKIE

cwe_checker

Architecture-Independent Binary Vulnerability Analysis

Security analysis of programs running on embedded devices is difficult

No access to source code

Emulation is difficult

Security analysis of programs running on embedded devices is difficult

Reversing binary code is too time-consuming

Cannot perform dynamic analyses directly on the device

No access to source code

Emulation is difficult

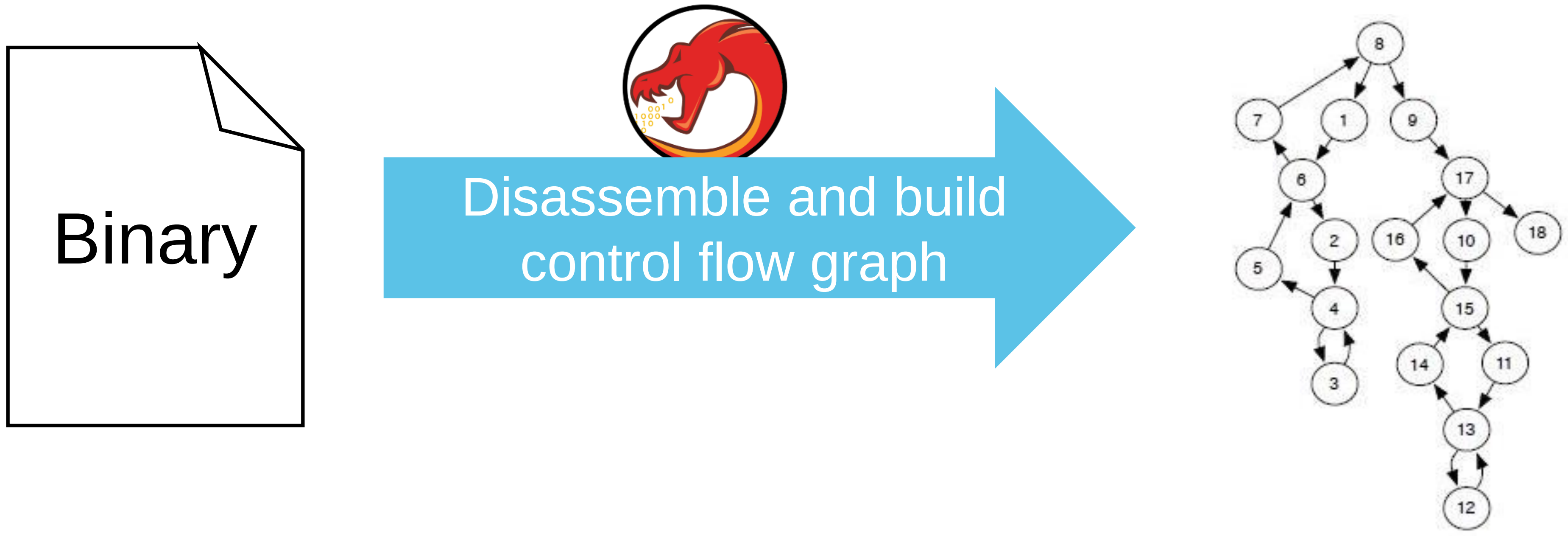
Security analysis of programs running on embedded devices is difficult

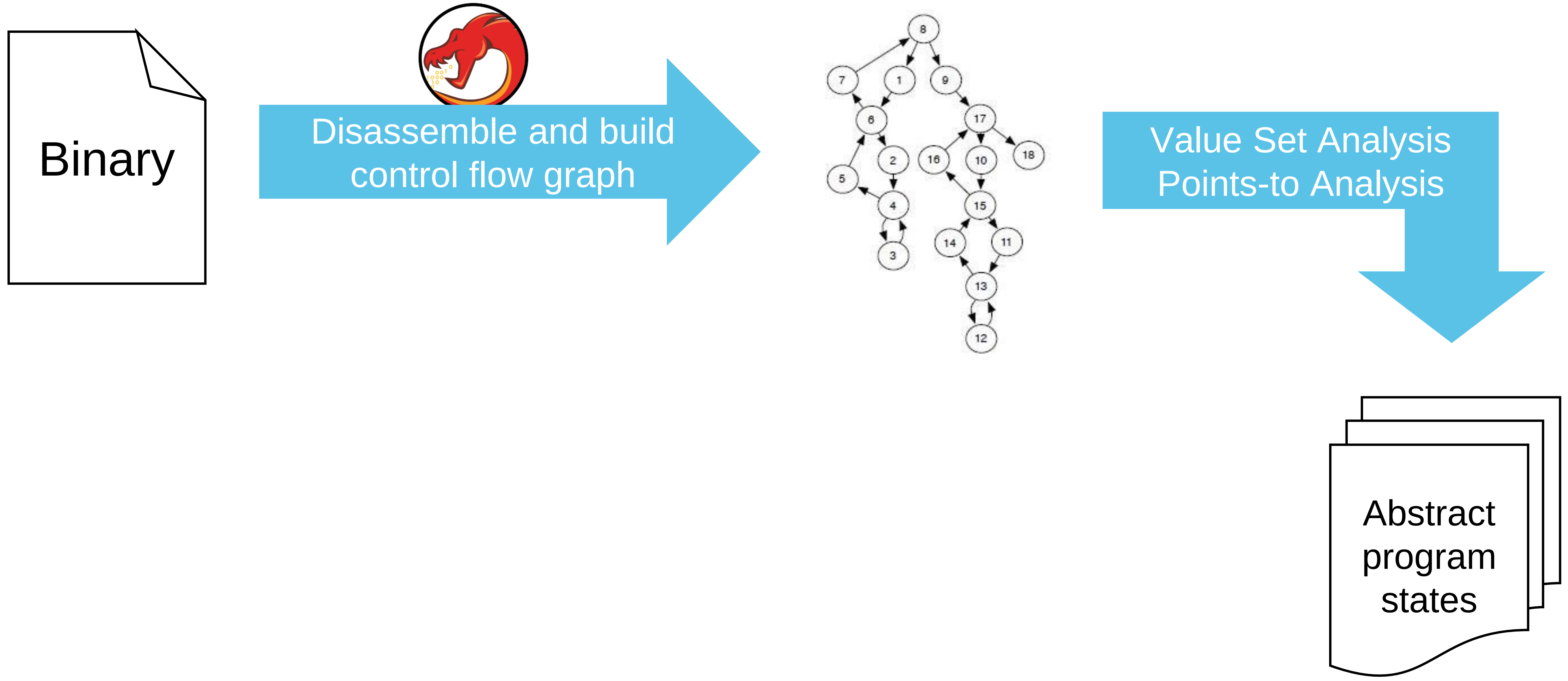
Reversing binary code is too time-consuming

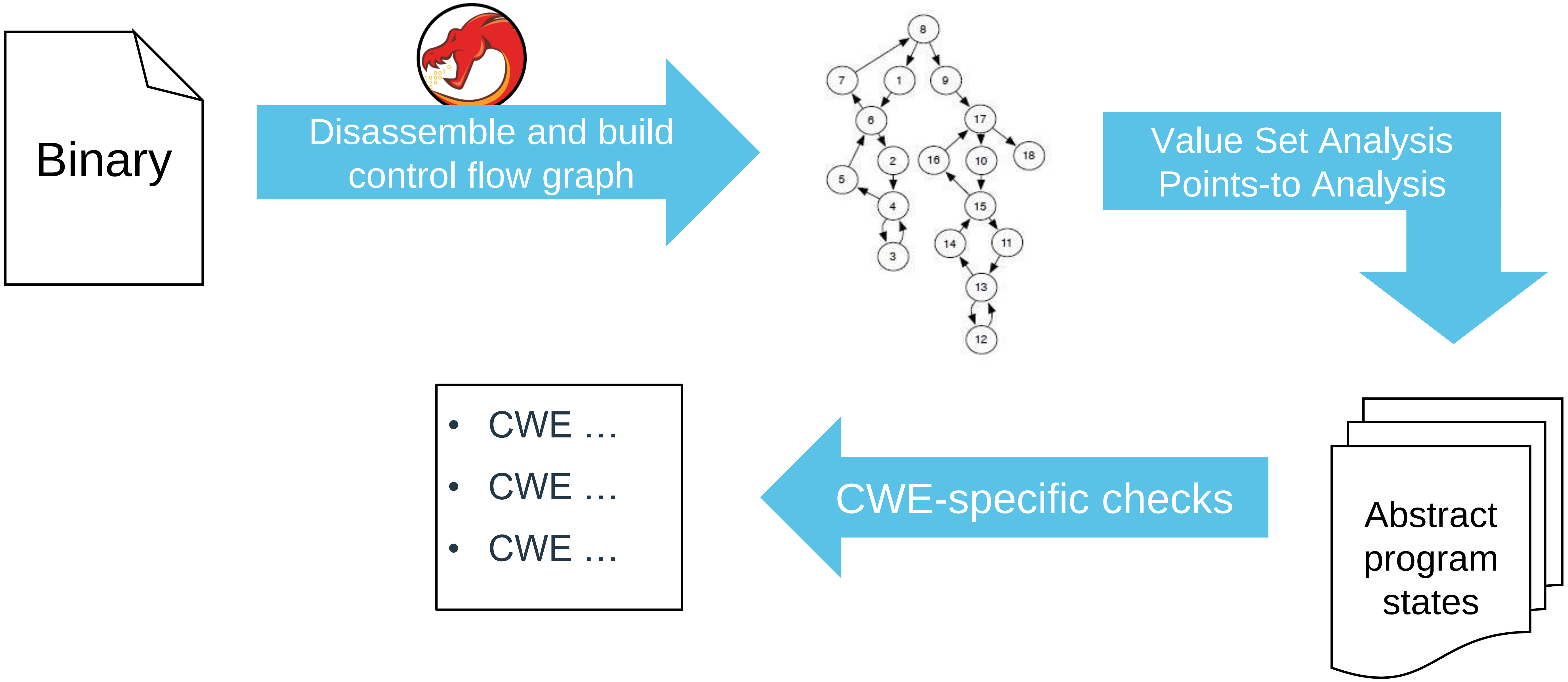
Cannot perform dynamic analyses directly on the device cwe checker

- The cwe_checker detects potential bugs and vulnerabilities in binaries
- Helps you focus the manual analysis on important parts

Demo

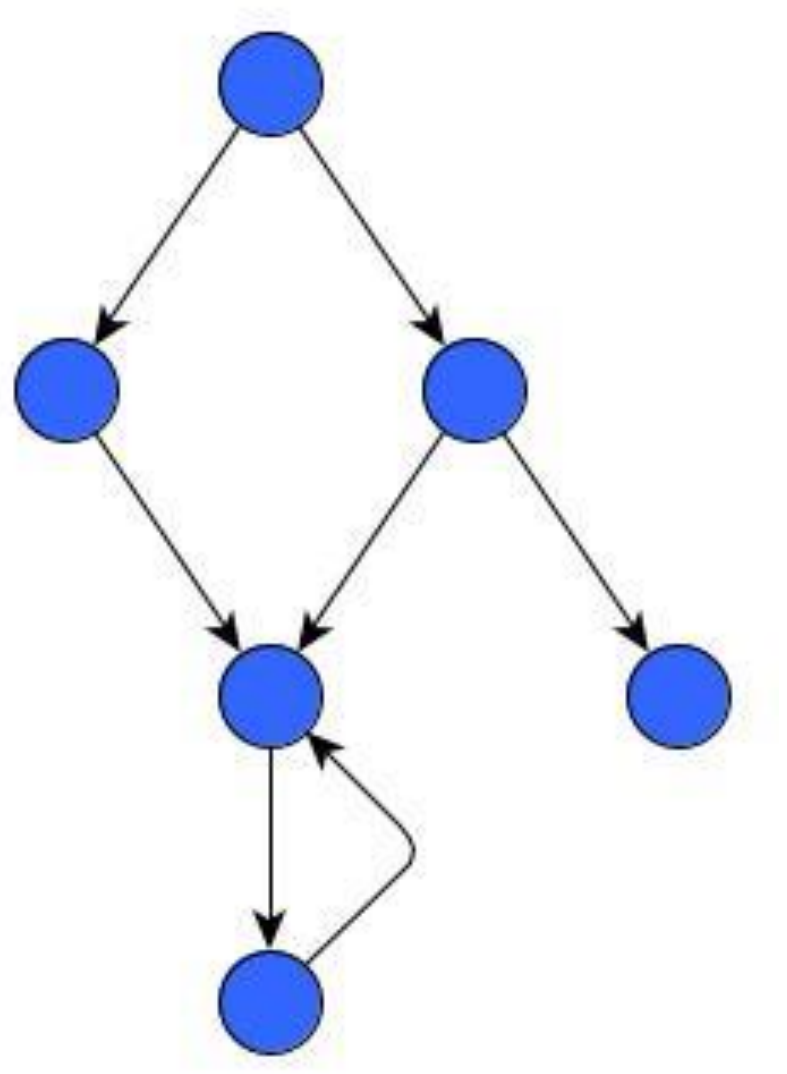

Binary
Disassemble and build control flow graph
8
7
1
9
6
17
2
16
10
18
5
4
15
3
14
11
13
12


Binary
Disassemble and build control flow graph
8
7
1
9
6
17
2
16
10
18
5
4
15
3
14
11
13
12
Value Set Analysis
Points-to Analysis
Abstract program states


Binary
Disassemble and build control flow graph
Value Set Analysis
Points-to Analysis
Abstract program states
CWE-specific checks
CWE …
CWE …
CWE …

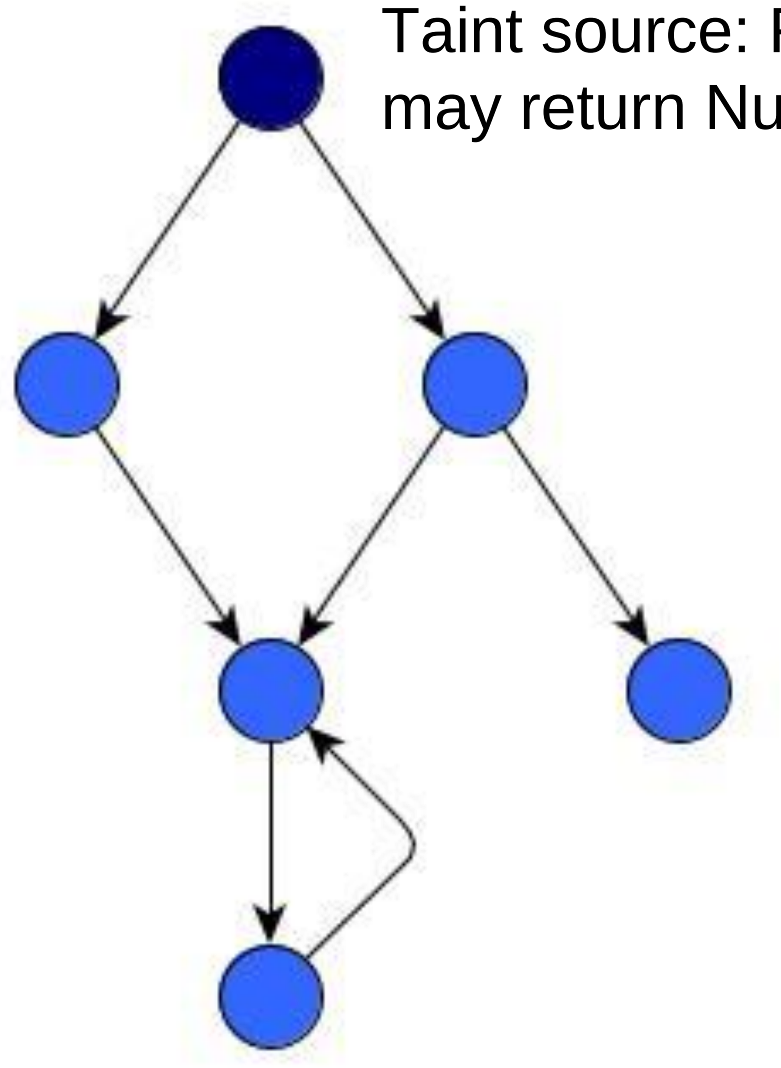
Example: CWE-476 Null Pointer Dereference

- Via taint analysis



Example: CWE-476 Null Pointer Dereference

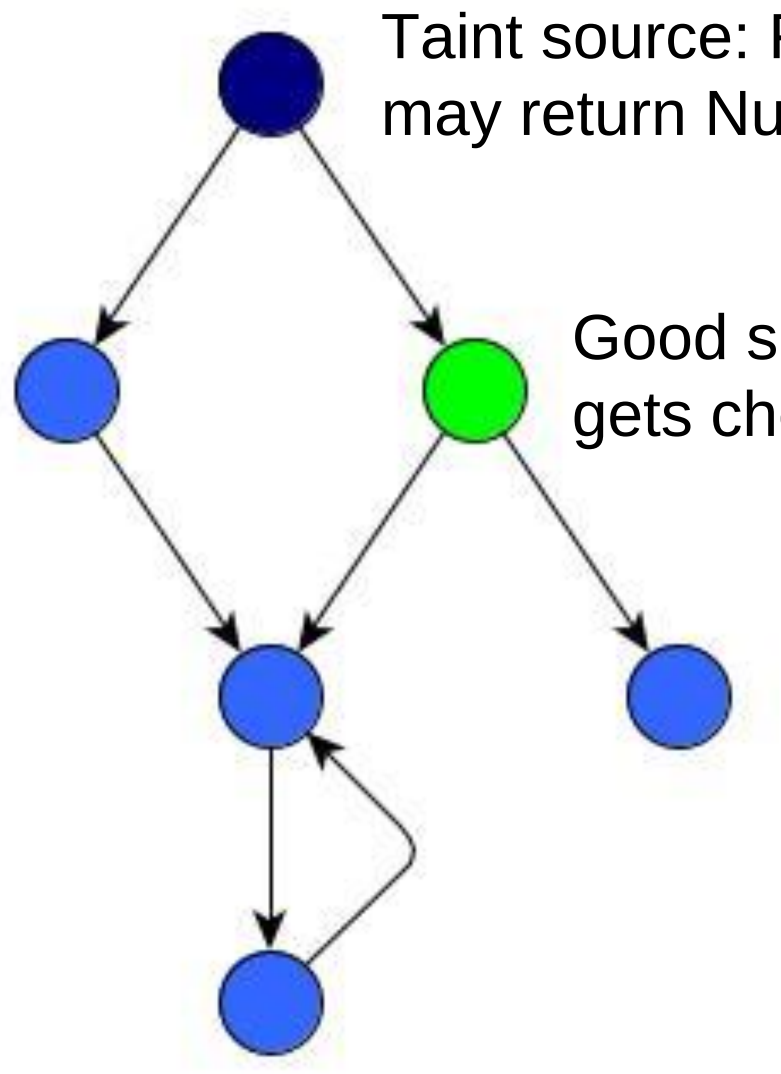
- Via taint analysis



Taint source: Function that may return Null pointer

Example: CWE-476 Null Pointer Dereference

- Via taint analysis

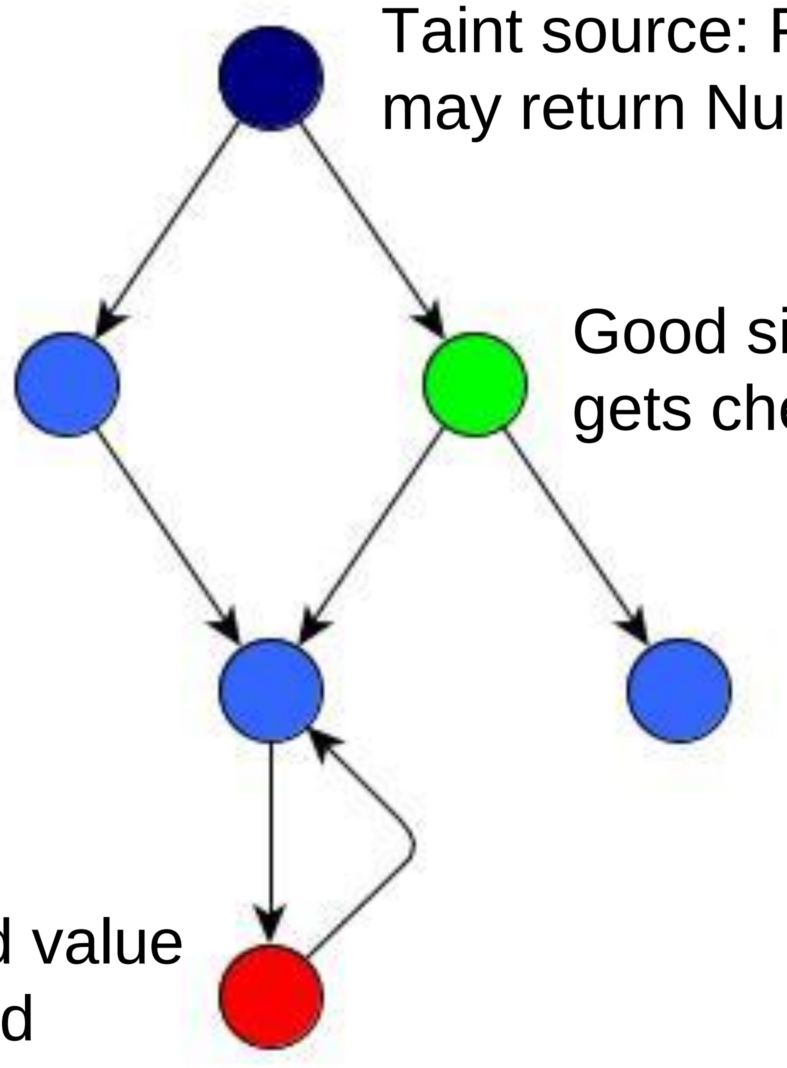


Taint source: Function that may return Null pointer

Good sink: Tainted value gets checked for being null

Example: CWE-476 Null Pointer Dereference

- Via taint analysis



Taint source: Function that may return Null pointer

Good sink: Tainted value gets checked for being null

Bad sink: Tainted value gets dereferenced

Analyze ELF binaries of many different CPU architectures

- x86, ARM, MIPS, PowerPC and more
- Experimental support for bare-metal and PE binaries exists

Contains checks for many bug types

- Currently checks for over 16 different CWE types implemented
- Behavior of checks configurable

Fast analysis

- Good for quick initial assessments
- Scan whole firmware images for certain bug types

Bugs need to be verified manually

- Path insensitivity will lead to false positives for most checks

Not suited for analysis of binaries written in other languages than C

- Control flow graph recovery not (yet) good enough for C++ and other languages

Summary

- The cwe_checker is a tool to quickly find potential bugs and vulnerabilities in firmware binaries.
- Detects 16+ different CWE types
- Based on static analysis
  → Beware of false positives/negatives!
- Easy to try out – just pull the Docker image *fkiecad/cwe_checker*
- Easy to integrate into your own toolchain thanks to JSON output

https://github.com/fkie-cad/cwe_checker
LGPL 3.0 Licence
@cwe_checker