

Code analysis - dynamic taint analysis

Peng Xu

July 2, 2019

Code analysis

1. What is the code analysis?

Code analysis

1. What is the code analysis?
2. Static analysis and dynamic analysis
 - ▶ Static analysis: debugging is done by examining the code without actually executing the program
 - ▶ Dynamic analysis: is performed in an effort to uncover more subtle defects or vulnerabilities

Code analysis

1. What is the code analysis?
2. Static analysis and dynamic analysis
 - ▶ Static analysis: debugging is done by examining the code without actually executing the program
 - ▶ Dynamic analysis: is performed in an effort to uncover more subtle defects or vulnerabilities
3. Source code and binary analysis
 - ▶ Source code analysis
 - ▶ Abstract syntax tree (AST): a tree representation of the abstract syntactic structure of source code
 - ▶ Intermediate representation(IR): representation of a program “between” the source and target languages

Code analysis

1. What is the code analysis?
2. Static analysis and dynamic analysis
 - ▶ Static analysis: debugging is done by examining the code without actually executing the program
 - ▶ Dynamic analysis: is performed in an effort to uncover more subtle defects or vulnerabilities
3. Source code and binary analysis
 - ▶ Source code analysis
 - ▶ Abstract syntax tree (AST): a tree representation of the abstract syntactic structure of source code
 - ▶ Intermediate representation(IR): representation of a program "between" the source and target languages
4. control flow graph(CFG) and data flow graph(DFG)

Code analysis

1. What is the code analysis?
2. Static analysis and dynamic analysis
 - ▶ Static analysis: debugging is done by examining the code without actually executing the program
 - ▶ Dynamic analysis: is performed in an effort to uncover more subtle defects or vulnerabilities
3. Source code and binary analysis
 - ▶ Source code analysis
 - ▶ Abstract syntax tree (AST): a tree representation of the abstract syntactic structure of source code
 - ▶ Intermediate representation(IR): representation of a program "between" the source and target languages
4. control flow graph(CFG) and data flow graph(DFG)

CFG, DFG, SDG

1. Control flow graph

- ▶ What is CFG?
- ▶ How can we get CFG?
 - ▶ angr: `https://angr.io/`
 - ▶ radare2: `https://rada.re/r/`
- ▶ What is purpose of CFG?

CFG, DFG, SDG

1. Control flow graph

- ▶ What is CFG?
- ▶ How can we get CFG?
 - ▶ angr: <https://angr.io/>
 - ▶ radare2: <https://rada.re/r/>
- ▶ What is purpose of CFG?

2. Data flow graph

- ▶ What is DFG?
- ▶ How can we get DFG?
 - ▶ llvm.analysis.dataflow
 - ▶ graph-llvm-ir
 - ▶ taint analysis: valgrind + taintgrind
- ▶ What is purpose of DFG?

CFG, DFG, SDG

1. Control flow graph

- ▶ What is CFG?
- ▶ How can we get CFG?
 - ▶ angr: <https://angr.io/>
 - ▶ radare2: <https://rada.re/r/>
- ▶ What is purpose of CFG?

2. Data flow graph

- ▶ What is DFG?
- ▶ How can we get DFG?
 - ▶ llvm.analysis.dataflow
 - ▶ graph-llvm-ir
 - ▶ taint analysis: valgrind + taintgrind
- ▶ What is purpose of DFG?

3. System dependent graph

- ▶ What is SDG?
- ▶ How can we get SDG?
- ▶ What is purpose of SDG?

Using angr getting CFG

- ▶ *import angr*
- ▶ *proj = angr.Project('./sign32')*
- ▶ *cfg = proj.analyses.CFG()*
- ▶ *dict(proj.kb.functions)*

Dynamic taint analysis

1. Valgrind + taintgrind <https://github.com/wmkhoo/taintgrind>
2. Steps:
 - ▶ labeling the sensitive data
 - ▶ tracing the taint propagation
 - ▶ finding the functions and statements relative with labeled sensitive data
3. Example
 - ▶ tests/sign32.c
 - ▶ `TNT_TAINT(&a, sizeof(a));`
 - ▶ `valgrind --tool=taintgrind tests/sign32`
 - ▶ `valgrind --tool=taintgrind tests/sign32 2>&1 | python
 **/log2dot.py > sign32.dot`
 - ▶ `gcc -g`

Partitioning a C-program

1. Dynamic taint analysis: **tracing the sensitive data propagation**
2. Partitioning the targeting C-program
 - ▶ TZSlicer
 - ▶ TZSlicer is based on TrustZone
 - ▶ TZSlicer is for bare-metal system
 - ▶ TZSlicer has function, basic-block and code line level partitioning
 - ▶ <https://github.com/hwsel/tzslicer>

Partitioning a C-program

1. Dynamic taint analysis: **tracing the sensitive data propagation**
2. Partitioning the targeting C-program
 - ▶ TZSlicer
 - ▶ TZSlicer is based on TrustZone
 - ▶ TZSlicer is for bare-metal system
 - ▶ TZSlicer has function, basic-block and code line level partitioning
 - ▶ <https://github.com/hwsel/tzslicer>
 - ▶ SGXSlicer
 - ▶ SGXSlicer is for Intel SGX
 - ▶ SGXSlicer has operating system supporting

Tasks

- ▶ Getting static control flow graph and dynamic control flow graph for your previous tasks:
 - ▶ Square Matrix is symmetric?
 - ▶ AES
 - ▶ Caesar cypher algorithm
 - ▶ MD5
- ▶ TZSlicer variant on function-level with optee supporting
- ▶ TZSlicer variant on function-level with sgx supporting

Question?

Questions?