Session 09

Code Analysis

Security of Information Systems (SIS)

Computer Science and Engineering Department

December 9, 2020

Ways of Securing Software

- secure by construction: prevent existence of bugs/vulnerabilities
- secure environment: prevent exploitation of bugs/vulnerabilities
- isolated environment: damage control

Secure by Construction

- providing it as secure (build from specs)
- building it secure
- secure before shipping

Secure by Construction (2)

- formal verification, provably secure
- programming language features
- programming practices
- defensive programming
- software development process
- code review
- code auditing
- testing
- fuzzing, symbolic execution

Common Practices/Principles

- keep it simple: small footprint, few dependencies, no fancy hacks
- input validation
- added care when dealing with buffers and strings
- use linters and static checkers
- make code readable, document while writing
- simple and intuitive interfaces
- mindset: assume the worse
- do unit tests

Program Analysis

- focus on applications (i.e. programs) not systems
- analyze program behavior
- performance
  - profiling
  - reduced resource usage
  - reduced overhead
- correctness
  - debugging
  - security
  - robustness
- no side channel focus

Ways of Doing Program Analysis

- control flow analysis: reachability
- data flow analysis: propagation

Types of Program Analysis

- static analysis: no running of program
- dynamic analysis: running the program
- source code analysis: source code is available, use it
- binary analysis: work on executables and binary files, source code may be unavailable

Static Analysis

- don't run the program
- go through its source/binary code
- control flow and data flow analysis

Dynamic Analysis

- monitor process
- usually involves instrumentation
- valgrind, profilers, Pin (https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool)

Source Code Analysis

- automated, semi-automated, manual
- manual: code auditing
- programming defects, API misuse, lack of compliance, correctness
- software/code interpretation, pattern matching
- software formal verification

Binary Analysis

- reverse engineering
- binary debugging
- disassembling, forensics

Terms

- program comprehension: understand source code
- code review: fix mistakes, improve code quality and programming practices
- code auditing: comprehensive analysis with intent of discovering bugs
- static analysis: automated action performed

Static Analysis

- analyze computer programs without executing them
- usually performed on source code
- automated process

Tools of the Trade

- editors/reading tools
- pattern matching tools
- static analyzers
- pen & pad

Tools of the Trade (2)

- open source
  - Sonar: http://www.sonarsource.org/ (Java)
  - Flawfinder: https://dwheeler.com/flawfinder/ (C/C++)
  - RATS
  - Clang Static Analyzer: http://clang-analyzer.llvm.org/
  - Splint: http://splint.org/ (C) – no longer developed
  - cppceck: http://cppcheck.sourceforge.net/ (C, C++) – plugins for IDEs
- proprietary
  - Coverity SAVE: http://www.coverity.com/products/coverity-save.html
  - Klocwork Insight: http://www.klocwork.com/products/insight/ (C, C++, Java, C#)
  - CodeSonar: http://www.grammatech.com/codesonar
  - Semmle: http://semmle.com/solutions/
  - HP Fortify

Binary Static Analysis

- requires reverse engineering
- focused on discovering bugs and creating exploitation PoCs form them to be fixed
- basic tools: disassemblers, symbol mappers, decompilers
- automated tools: Veracode, CodeSonar, BitBlaze
- security analysts, enhancing proprietary solutions

Code Auditing

- browse source code
- look for security breaches and possible bugs
- tools for static code analysis
- in-depth audit to be done by the developer

Black Box Approach

- non-open-source code
- understand protocol or user input format
- provide "bad" input and test possible violations
- reverse engineering
- fuzzing

White Box Approach

- the "real stuff" – actual code auditing, highlight input processing
- top-to-bottom: start from main, go down functions
- bottom-to-top: find all places of external input, system input and start from there

Tools to be Employed

- static analyzers (cppcheck, Clang Static Analyzer, Coverity)
- IDA for binary static analysis
- ctags, cscope, source nav for source code navigation
- debuggers for runtime analysis
- valgrind, Rational Purify for dynamic analysis

Code Auditor Requirements

- know API, OS and machine (background knowledge)
- recognize patterns (pattern recognition)
- understand application (functional understanding)
- audit all code (completeness)

Types of Programs

- `http://www.ouah.org/mixtercguide.html`
- setuid/setgid programs
- daemons and servers
- frequently run system programs
- system libraries (libc)
- widepread protocol libraries (kerberos, ssl)
- administrative tools
- CGI scripts, server plugins

Classes of Bugs to Audit

- API-based bugs
- external resource interactions
- programming construct errors
- state mechanics

API-based Bugs

- misuse of OS, library of framework APIs
- dangerous string or formatting functions: e.g., sprintf(), strcpy(), strcat(), printf(), syslog() . . .
- dangerous implicit behavior: e.g., allocators that round
- cumbersome/complicated API reference contents: e.g., threading, IPC

External Resource Interactions

- privilege escalation through IPCs
- `system()`, `execve()`, `CreateProcess()`
- file interaction

Programming Construct Errors

- CWE: Common Weakness Enumeration `https://cwe.mitre.org/data/index.html`
- integer signedness
- integer boundaries
- checks that are logically wrong or susceptible to integer problems
- loops that have bad boundaries
- unchecked variables

State Mechanics

- programs left in an inconsistent state
- thread safety issues
- async-safety issues
- global variables left in an undesired state

Methodology

- target components, meta targeting
- grep targeting – won't provide understanding
- read code quickly – ignore what is not important
  - copy and move data
  - input/output

List of Issues

- implementation bugs (miscalculation, check result, not validate input)
- data types
- memory corruption

Defensive Programming

- sh*t happens
- assume the worst, program accordingly
- secure programming / secure coding
- offensive programming
- formal verification
- rewrite vs reuse

Secure Coding

- `https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard`
- techniques for building secure programs
- handling input
- working with memory and buffers
- handle error/exceptions
- handling data types

Input Validation

- anything can be malitious
- look for injections
- take into account encoding
- only allow required format

Buffer Management

- start address and length
- boundary checking
- indexes

String Management

- length management
- NUL-byte termination
- string truncation
- printable characters

Integer Management

- conversions (size)
- overflows
- signedness

Keywords

- secure by design / implementation
- program analysis
- static analysis
- dynamic analysis
- source code analysis
- binary analysis
- code auditing

- bugs
- vulnerabilities
- programming errors
- CWE (*Common Weakness Enumeration*)
- defensive programming
- secure coding

Resources

- `https://www.amazon.com/Building-Secure-Software-Addison-wesley-Professional/dp/0321774957`
- `https://www.amazon.com/Secure-Coding-2nd-Software-Engineering/dp/0321822137`
- `https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard`
- `https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide`
- David Binkley: Source Code Analysis: A Road Map
- `https://cwe.mitre.org/data/index.html`
- `https://samate.nist.gov/SRD/testsuite.php`

References

- `http://pentest.cryptocity.net/code-audits/`
- `http://software.intel.com/en-us/articles/collection-of-examples-of-64-bit-errors-in-real-programs/`
- `http://www.ouah.org/mixtercguide.html`
- `http://www.vanheusden.com/linux/audit.html`
- `http://spinroot.com/static/`
- `http://spinroot.com/p10/`
- The Science of Code Auditing, BlackHat EU 2006
- `https://www.grammatech.com/products/binary-analysis`
- `http://bitblaze.cs.berkeley.edu/`
- `https://www.veracode.com/`