Methods and software tools for analysis of binary code security

V. Padaryan, M. Solovyev, A. Getman, K. Batuzov, V. Efimov, M. Bakulin, S. Panasenko, O. Goremykin

{vartan, eyescream, thorin, batuzovk, real, bakulinm, spanasenko, goremykin}@ispras.ru

Yerevan – May 3 – 2017
Binary code analysis: goals and objects

- Exploring the software properties in the absence of source code
  - Control of absence of undeclared features
  - Identifying software defects
  - Evaluation of the influence of a software defect on software security
  - Network protocols recovery
- Behavior monitoring of deployed system while its operating
  - 0-day exploit detection
  - Compliance with security policies
- Network security
  - Analyzing new types of network attacks in high-speed traffic

Applications
OS kernel & drivers
Bootloaders & Hypervisors

Personal & enterprise appliances: desktop/laptop, server, smartphone / tablet, IoT, ...
Network equipment: router, firewall, wireless, ...
Backbone & LAN network traffic
Challenges for binary code analysis

• The ideal analysis tool
  – *Write once, analyze everything*
    sorry for WORA «plagiarism»
  – The requirement is hardly compatible with the real objects of analysis
• A significant amount of analyzed code
  – A typical firmware size is about several MB
  – A typical PC or mobile application size (including libraries) is about 10-100 MB or more
• The executable code is built by optimizing compilers; often code obfuscation is used
• Permanent extension of the x86 ISA
  – 23 ISA extensions for 2011-2016
  – Most of them are system commands implementing security features
    • VT-x, VT-d, SVN, SGX, MPX, CET
• IoT
  – A lot of different SoCs and CPUs
Responding To Challenges
Four base (compiler) technologies (1/2)

1. The combination of dynamic and static analysis allows to overcome their generic limitations
   - Dynamic analysis reveals the real code and data, the actual values of variables and their addresses
   - Static representation of the program is better perceived by people and fully represents implemented algorithms

2. The analyzed code is executed in a controlled environment – a software emulator with built-in analysis tools
   - Debug interfaces in the hardware can be disabled or even physically blocked. The emulator always allows to observe the executed code "from the outside."
   - A mandatory and minimal requirement for emulation is the availability of a ISA description.
   - If the available description of the periphery, it’s possible to build a complete VM.
   - In the emulator, it is possible to precisely reproduce the once observed program execution, analyzing the code replay in various ways.
   - It's possible to capture all data flows in the computer system
Responding To Challenges
Four base (compiler) technologies (2/2)

3. Data and control flows revealing at the level of machine instructions
   - The classical compiler theory is applicable (after certain modifications and improvements) to represent and analyze the properties of binary code
   - Opens the ability to automatically extract the algorithm processing certain input data from the total mass of program code
   - The approach is applicable even if the flow of data goes into another process or the OS kernel

4. Intermediate representation (IR) allows to analyze the data flows, abstracting from the hardware complexity
   - The code of the various CPUs is translated into a convenient for automatic analysis and a uniform IR
   - Conventional compiler representations (llvm and others) are poorly applicable, because when translating, they require a high level knowledge of the program (variable types, control statements and so on)
   - Specialized IR are used: Pivot, VEX, REIL, BAP, ...
Can we start working on a security task immediately?

Development of analysis tools → Reverse engineering → Debugging & improving of analysis tools → Reverse engineering → Result!

- In the absence of a ready-made and well-functioning tools, it’s necessary to iteratively improve the existing toolkit before the end-user applied problems (RE & bug finding) begin to be solved.
- Special tools for rapid development of analysis tools are also demanded.

• Protocol reverse engineering
• Algorithm reverse engineering
• Bug finding

Binary code analysis environment
Lifting the binary code representation level

Executing the analyzed program in a completely controlled environment (software emulator)
Qemu – the basis for dynamic analysis

Tools to solve practical problems
- Honeypot
- Extendable OS-aware debugger
- Behavioral analysis
- Reproducible user environment

Analysis technologies
- Taint analysis
- Debugging
- Tracing

Extensibility of analysis tools
- Analysis plugin infrastructure

QEMU emulation engine
- Dynamic binary translator TCG + deterministic replay

Extensibility of a supported VMs
- Automated development of new virtual machines
  - CPU
  - Integration of VM components
  - Peripherals
Virtual Machine deterministic replay

- The problem: a "heavyweight" analysis (debugging, step-by-step tracing, etc.) leads to a dramatic VM slowdown. If the code being analyzed interacts with the "outside world", the code behavior inevitably changes.
- Deterministic replay guarantees a repeat of execution with an accuracy of a single machine instruction. The record overhead is limited to 10-50%.
- Record/replay was implemented in leading commercial emulators: SimNow, Simics, Synopsys Virtual Platform
- Qemu contains a record/replay engine developed by ISP RAS (first patch set was included into v2.5, full RR support – v2.8)

★ Replay for register transfer level (RTL)
Accelerated VM development

- What if a software security analysis faces a new hardware platform?
  - Build and analysis tools (translators, debugger, disassembler, ...)
  - Emulator
- The stage with critical and poorly predictable duration – the development of a new virtual machine.
  - CPU
  - Various peripherals
  - VM component integration

Qemu development tools (periphery and VM integration) are free software, available at https://github.com/ispras/qdt
Protection against malicious software distributed in e-documents

• Identifying violations of basic security properties allows to protect the workplace with fixed set of applications from 0-day vulnerabilities.
• Reference tool – commercially available system SandBlast by Check Point
• A lot of similar open source systems (TEMU, DECAF, Argos, Panda, TaintCheck, TaintDroid, …) have only academic value and are inapplicable for practical use.
• ISP RAS scientists proposed a method for warning ranking, reducing the percentage of false positives.

Maksim Bakulin, Maria Klimushenkova and Danila Egorov. Dynamic Diluted Taint Analysis for Evaluating Detected Policy Violations. // Ivannikov ISPRAS Open Conference 2017
Lifting the Representation Level

«Мета PDS»: code merging from different traces, building relocatable (module based) code, closure of edges in IR

Function recovery

PDS (Pushdown System)
Static representation / Code generations

Module identification

Call-Ret matching

StaticMem

Interrupt recovery

Thread / Process / Zone markup

Binary code trace
High level algorithm representation

- Extraction of the algorithm from binary code when searching for undeclared feature
  - Description of input and output, dependencies between them
  - Description of operations over data
  - Network protocols and data format recovery

1. Preliminary lifting the representation level
   - Automatic recovery of the static representation of the machine instruction level by traces

2. Man-driven algorithm extraction and description
   - Checking the actual behavior of the described code
Data format recovery by dynamic analysis

Input data (network packets) → Packet parser (executable code) → Traces

Emulator

Message format recovery

Trace analysis

Syntactic structure of the message

Grammar recovery

Partial grammar

Model parser

Traffic generator

Protocol implementation recovery
Network traffic analysis for poorly documented protocols
Future works and next decade challenges

• HW virtualization based controlled execution environment: better performance, better VM authenticity
  – Xen, KVM, ...
• New Pivot2 IR: constructed from binary code, suitable for abstract interpretation by design
• New, «micro service architected», analysis environment TRAWL

• How to analyze hardware assisted security: secure boot chain, SGX enclaves, ... ?
• How to analyze code while some hardware interfaces are totally undocumented?
• How to formally describe errors that are slightly more complex than buffer overflow or null pointer?