The Art of War: Offensive Techniques in Binary Analysis

Yan Shoshitaishvili et. al.

杨宇清

April 11, 2019

Table Of Contents

Stubborn Binary Flaws

- Low-level language has few security gurantees
- Simple flaws remain stubborn(e.g. buffer overflow)
- New emerging vulnerabilities: IoT...

Problems of existing researches

- SOTA methods ends up in prototypes
- Massive workload & public unavailability: impractical to replicit

Solve the first issue

- Reproduce
- Mitigate

Automated Binary Analysis

- Replayability
- Semantic insight

Static Discovery

- Controll flow recovery
  - Computed:target of jump is determined by calculation
  - Context-sensitive: Determined by the caller
  - Object-sensitive: In OO languages: e.g. virtual table
- Flow modeling: requires pre-existing knowledge of the vulnerability
- Data modeling: by applying a tight over-estimation

Dynamic Discovery

- Concrete Execution: executing in a minimally instrumented environment
  - Coverage-based fuzzing: AFL, modifying input
  - Taint-based fuzzing, tries to know how the program processes the data
- Symbolic Execution: using abstract symbols to execute
  - Classical: not very practical, bugs shallow
  - Symbolic-assisted fuzzing: combines the advantages of fuzzing and symbolic execution
  - Under-constrained symbolic execution: low replayability.

Exploitation

- Crash reproduction
  - Missing data
  - Missing relationships
- Exploit generation: NULL-pointer
- Exploit hardening: modern techniques(ASLR,etc.) makes it less practical

Goal:

- Cross-architecture
- Cross-platform
- Various analysis paradigm sypport
- Usability

Modules:

- Imtermediate representation
- Binary Loading
- State Representation/Modification
- Data Model
- Full-Program Analysis

Imtermediate representation

libVEX

Binary Loading

CLE

State Representation/Modification

State plugins:

- Registers
- Symbolic memory
- Abstract memory
- POSIX
- Log
- Inspection
- Solver
- Architecture

Data Model

Claripy: abstracts all values into internal representation

- FullFrontend
- CompositeFrontend
- LightFrontend
- ReplacementFrontend
- HybridFrontend

Full Program Analysis

A combine of all the methods above.

CFG Recovery

- Based on several assumptions
- Iterative CFG Generation
  - Forced Execution
  - Symbolic Execution
  - Backward Slicing
- CFG Fast
  - Function identification
  - Recursive disassembly
  - Indirect jump resolution

Value Set Analysis

Impromentsagainst previous approaches:

- Creating a discrete set of strided-intervals
- Applying an algebraic solver to path predicates
- Adopting a signedness-agnostic domain

Dynamic Symbolic Execution

- Uses Claripy to populate symbolic model
- Uses Path object(angr) to track action,path, etc.
- Aggregate paths to PathGroup

Under-constrained Symbolic Execution

Two changes:

- Global memory under-constraining
- False positive filtering

Symbolic-assisted Fuzzing

- AFL executes until there is no new states
- Invoke angr on unique paths for new transitions

Crash Reproduction

- Search problem: an input bringing program from s_0 to s

Exploit Generation

- Vulnerable States
- Instruction Pointer Overwrite Technique
- Exploiting CGC Binaries

Exploit Hardening

Steps:

- Gadget discovery
- Gadget arrangement
- Payload generation