Symbolic Execution the Swiss-Knife of the Reverse Engineer Toolbox

KLEE Workshop – September 16-17th, 2022

Robin David  Quarkslab  <rdavid@quarkslab.com>
Christian Heitman  Quarkslab  <cheitman@quarkslab.com>
Richard Abou Chaaya  Quarkslab  <rabouchaaya@quarkslab.com>
Agenda

Part 1. Obfuscation

Part 2. Exploration / Fuzzing

Part 3. Research & TritonDSE
Use-Case #1
Obfuscation Assessment
# Obfuscation Assessment

## Use-Case #1
Assessing obfuscation strength

*(its ability to protect data, keys that it needs to protect)*

<table>
<thead>
<tr>
<th>Obfuscation in the industry</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Banks, payment solutions</td>
</tr>
<tr>
<td>- Mobiles applications (<em>IP protection</em>)</td>
</tr>
<tr>
<td>- DRM, Video-on-Demand</td>
</tr>
<tr>
<td>- etc.</td>
</tr>
</tbody>
</table>

⇒ Multiple existing work to attack opaque predicates [1, 16, 3] or virtualization [12]
MBA (Mixed Boolean Arithmetic) diversify simple operations by mixing them with arithmetic and bitwise operations that are **semantically equivalent**.

⇒ Can be defeated with: **Symbolic Execution** + **Program Synthesis** [4, 5].

(Other SMT-based approaches have been proposed [13])
⇒ Use SE as a mean of **extracting data-flow expressions** of registers or memory locations in the program.
Dataflow Expressions Synthesis

**Simplification Algorithm**
AST traversal using different strategies to trying simplifying opportunistically sub-ASTs.

**I/O Oracle Synthesis**
Evaluating expressions on a set of inputs. If it expresses the same behavior than some smaller pre-computed expressions replaces it (assume they are semantically equivalent).

⇒ SMT can be used to prove equivalence between both input and synthesized expression.
MBA: Concrete use-cases

Other concrete usages:
- Off-the-shelf obfuscators (eg: all LLVM-based obfuscators)
- Used in Android SafetyNet [15]

Conclusion: SE very useful for obfuscation to manipulate the semantic which is the only thing that must be preserved by obfuscation.
Use-Case #2
Program Exploration
Use-Case #2
In support of fuzzing to assess static analysis alerts
Use-Case #2
In support of fuzzing to assess static analysis alerts

Industry Problem
Many companies uses static analyzer for security or compliance before shipping their code (or requires sub-contractors to do so)
Program Exploration

Use-Case #2
In support of fuzzing to assess static analysis alerts

Industry Problem
Many companies use static analyzers for security or compliance before shipping their code (or require sub-contractors to do so).

Underlying Problem
⇒ Static analyzers usually yield many alerts for which it is difficult to discriminate true flaws and false positives.
Static Analysis

Features

- **Languages:** C, C++, Java,
- **Checkers:**
  - 300 checkers C/C++
  - 91 community checkers AUTOSAR
  - 24 CERT community checkers
  - ...

Coding standard ("checkers")

- AUTOSAR
- CWE for C# and Java
- Joint Strike Fighter Air Vehicle C++
- MISRA
- PCI DSS

⇒ Usually *de-facto* standard for compliance in some automotive, industrial systems.
#5116: Array 'buffer' of size 2049 may use index value(s) 0.0.0.62
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/http/http_client.c:577 | httpClientSetHost()
Code: ABV/GENERAL | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

#5139: Pointer 'datagram' returned from call to function 'netBufferAt' at line 431 may be NULL and will be dereferenced at line 434.
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/ipv4/ipv4_frag.c:434 | ipv4tReassembleDatagram()
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

#5155: function 'strcpy' does not check buffer boundaries but outputs to buffer 'context->method' of fixed size (9)
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/http/http_client.c:449 | httpClientSetMethod()
Code: SV.STDBO.UNBOUND_COPY | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

#5321: Pointer 'segment2' returned from call to function 'netBufferAt' at line 349 may be NULL and will be dereferenced at line 352.
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/core/tcp_misc.c:352 | tcpSendResetSegment()
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

#5342: Pointer 'arpRequest' returned from call to function 'netBufferAt' at line 909 may be NULL and will be dereferenced at line 912.
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/ipv4/arp.c:912 | arpSendRequest()
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

#5396: Pointer 'vlanTag' returned from call to function 'netBufferAt' at line 222 may be NULL and will be dereferenced at line 225.
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcb/cyclone_tcb/core/ethernet_misc.c:225 | ethEncodeVlanTag()
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Owner: unowned

(they have not discovered SARIF format yet)
Intrinsic Functions Insertion

Advantages

▶ allows retrieving precisely the alert location in resulting binary (also encompass inlining..)
▶ body on __klocwork_alert_placeholder print on stdout alert ID

(intrinsic should be familiar to KLEE users with klee_assume etc..)
The Approach

Combining **Fuzzing** and **Symbolic Execution** to

*cover* the alerts and to *check* if they are true positives

**Fuzzing** [blazingly fast]
- Coverage: by parsing stdout
- Validation: in case of crash → last intrinsic covered

**DSE** [might cover deeper states]
- Coverage: detect the call to the intrinsic
- Validation: dedicated runtime or symbolic checkers (*sanitizers*)

⇒ **Corollary issue**: How combining them efficiently?
Symbolic Checker  SV_STRBO_BOUND_COPY_OVERFLOW

```python
def handle_svstrbo_bound_copy_ov(se) -> bool:  # se is symbolic state
dst_size = se.get_argument_value(2)
ptr_inpt = se.get_argument_value(3)
n, sym_n = se.get_full_argument(4)  # both concrete and symbolic value
# Runtime check
if n >= dst_size and len(se.get_memory_string(ptr_inpt)) >= dst_size:
    return True  # violation triggered
# Symbolic check
predicate = [sym.get_path_constraints(), sym_n > dst_size]
# For each memory cell, try to proof that they can be different from \0
for i in range(dst_size + 1):  # +1 in order to proof that we can at least do an off-by-one
    sym_cell = sym.read_symbolic_memory_byte(ptr_inpt + i)
predicate.append(cell != 0)

st, model = sym.solve(predicate)
if st == SolverStatus.SAT:
    crash_seed = mk_new_crashing_seed(se, model)
return True
```

⇒ Can flag input as “crashing” even though the harness is not crashing per-se.
Complete Workflow

Source code → code harnessing

Creation of the harness

Manual

Indeed can’t prove an alert to be false negative

Helps the analyst focusing on remaining uncovered, unvalidated alerts.
Complete Workflow

Source code → code harnessing → SAST → Alerts report → intrinsic insertion

Creation of the harness: Manual
Static Analysis: Semi-automated (manual checking intrinsic insertion)

Indeed can’t prove an alerts to be false negative
Helps the analyst focusing on remaining uncovered, unvalidated alerts
Complete Workflow

Creation of the harness
- Manual

Static Analysis
- Semi-automated
  (manual checking intrinsic insertion)

Dynamic Testing
- Fully-automated
  - coverage
  - validation

Indeed can't prove an alerts to be false negative
Helps the analyst focusing on remaining uncovered, unvalidated alerts
Complete Workflow

- Creation of the harness
  - Manual

Static Analysis
  - Semi-automated
  (manual checking intrinsic insertion)

Dynamic Testing
  - Fully-automated
  - compilation for DSE
  - fuzzing
  - DSE
  - Report
    - coverage
    - validation

- Indeed can’t prove an alerts to be false negative
- Helps the analyst focusing on remaining uncovered, unvalidated alerts
Ensemble Fuzzing

Definition

Approach aiming at making **heterogenous** testing tools to **collaborate** to fuzz a given target. *(broad definition of fuzzing)*

Rational:

▶ No fuzzer is universally better on every targets
▶ Efficiency depends on the fuzzing approach, coverage, mutation technique etc..

⇒ **It might be valuable to combine different test engines**  
_(existing litterature [7, 9, 2, 6, 10])_
Our project: PASTIS

Characteristics

- written in Python
- distributed \textit{(network-based)}
- run engines in parallel
- enable adding new fuzzers
- DSE: Triton
- fuzzing: Honggfuzz, AFL++
- replay \textit{(ensure replayability)}

Used it to fuzz TCP/IP stacks. Found issues for which some have CVEs \textit{(CVE-2021-26788)}.

⇒ Designed to work binary-only targets \textit{(in this case cannot leverage intrinsic mechanism)}
PASTIS Architecture

Initial Configuration
- binary
- SAST report (klocwork)
- configurations (coverage strategy, etc)

Master
pastis-broker

Workspace
- corpus / crashes / hangs
- log and client statistics
- CSV of results

Communication
(libpastis)

Fuzzing
pastis-honggfuzz

Python driver
execve

TritonDSE
(exploration of paths)

Triton
(symbolic execution of one path)

DSE
pastis-triton

1. Connection (idle)
2. Reception of binary (+opts)
3. Seed exchange (+logs)
4. Infos of alert validation
5. Stop

17/29
Demo
Research & TritonDSE
TritonDSE is developed as a **Python library** based on a **callback** mechanism

(address, instructions, memory, registers, context-switch, new inputs, formular solving etc.)

**Functionalities for a whitebox fuzzer**

- program loading (*ELF, based on LIEF* [11], and also now *cle*)
- input seed scheduling (**customizable**)
- program exploration & coverage computation
- dynamic & symbolic sanitizers (*for different vulnerability categories*)
- Memory segmentation with permissions
- Basic heap allocator with *alloc & free* primitives (**customizable**)
- Basic multi-threading support
- Multiple libc symbolic stubs

*(will soon be open-sourced)*
Ongoing Experimentation

Ongoing experiments with TritonDSE and PASTIS:

- custom coverage strategies
- seed scheduling
- slicing
- directed approaches
- seed sharing strategies (PASTIS)

Leveraging full disassembly

Some of these analyzes requires manipulating the complete disassembly. We use Quokka to export the whole IDA disassembly with all metadata. (code & data cross references etc) (also soon open-source)
Fuzzbench Integration

Will enable further benchmarks (to compare various strategies & algorithms)
Conclusion

- Symbolic Execution is **very** handy for reverse engineering.
- Keeping experimenting with SE helps finding way to tackle new problems encountered *(obfuscation, exploring specific targets etc.)*
- Keeping experimenting to answer research questions *(unstuck fuzzing, reaching a location, ensemble fuzzing combination vs separate run, etc..)*
Thank you!

Contact:
Email: rdavid@quarkslab.com
Phone: +33 1 58 30 81 51
Site: https://www.quarkslab.com
Opaque Predicates

**Definition:**
Predicate always evaluating to true (resp false) (but for which this property is difficult to deduce).

**Can be based on:**
- arithmetic
- data-structure
- pointer (aliasing)
- etc..

\[7y^2 - 1 \neq x^2\]

(hold for any \(x, y\) in modular arithmetic)

⇒ Symbolic execution helps proving the unsatisfiability of the dead branch

*(now widely studied in literature [1, 16, 3])*
**Definition:**

**Virtual Machine (VM)** defines a custom instruction set (ISA) with **virtual** registers and memory.

**How:** The code to obfuscate is translated into opcode in this ISA, and then evaluated by the VM in a fetch, decode, dispatch, repeat manner.

⇒ Can be defeated by the low interaction between VM code and “real” code [12].
## Existing Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Bio:</th>
<th>Pros/Cons:</th>
</tr>
</thead>
</table>
| ClusterFuzz [7] | Authors: Google  
Base: libfuzzer  
Used by OSS-Fuzz [8]                                                 | scale  
require an Azure cloud instance                                               |
| OneFuzz [9]     | Author: Microsoft  
Base: AFL, Radamsa                                                    | support AFL, libfuzzer, aflfast, intefuzz, fairuzz.. |
| EnFuzz [2]      | Author: Tsinghua University                                           | unified harness (GTest like)  
unmaintained  
require fuzzer restart on new seed                                                |
| Deepstate [6]   | Author: TrailofBits  
Base: libfuzzer, AFL, Honggfuzz, Eclipser, Angora                   | based seed sharing (local directory)  
academic tool  
a single commit                                                                   |
| CollabFuzz [10] | Author: Vusec (TU University)  
Base: AFL, AFL++, QSym, AFLfast, Fairfuzz, Honggfuzz, libfuzzer        | Based on Docker  
message exchange with ZeroMQ                                              |


GOOGLE, *Clusterfuzz - scalable fuzzing infrastructure*. [code].
References II

—, *Oss-fuzz: continuous fuzzing for open source software.*
https://github.com/google/oss-fuzz [code].

[code].


[site], April 2017.


