# 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>


Quarkslab

# Agenda

**Part 1. Obfuscation**

**Part 2. Exploration / Fuzzing**

**Part 3. Research & TritonDSE**

# Use-Case #1
# Obfuscation Assessment

Quarkslab

# Obfuscation Assessment

**Use-Case #1**

Assessing obfuscation strength

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

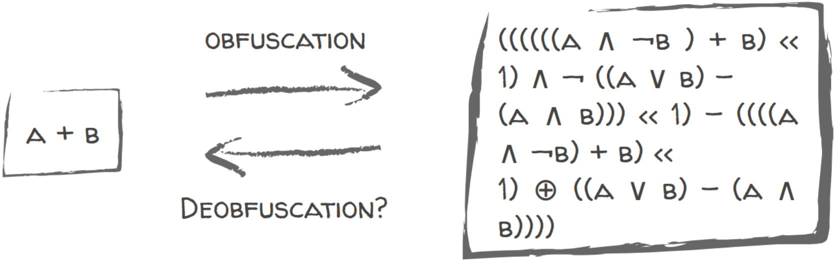
## Obfuscation in the industry

- Banks, payment solutions
- Mobiles applications *(IP protection)*
- DRM, Video-on-Demand
- etc.

$\Rightarrow$ Multiple existing work to attack opaque predicates [1, 16, 3] or virtualization [12]

# Mixed-Boolean Arithmetic

**MBA** (Mixed Boolean Arithmetic) diversify simple operations by mixing them with arithmetic and bitwise operations that are **semantically equivalent**.




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

*(other SMT-based approaches have been proposed [13])*

# SE for Synthesis

Assembly

```
mov     rax, rsi
xor     rax, 0xFFFFFFFFFFFFFFFF
or      rax, rdi
mov     rcx, rdi
xor     rcx, 0xFFFFFFFFFFFFFFFF
and     rcx, rsi
mov     rdx, rdi
and     rdx, rsi
xor     rdx, 0xFFFFFFFFFFFFFFFF
or      rdi, rsi
add     rax, rcx
sub     rax, rdx
add     rax, rdi
retn
```

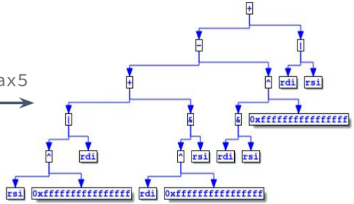
SE →

Intermediate Representation

```
rax0 := rsi
rax1 := rax ⊕ 0xFFFFFFFFFFFFFFFF
rax2 := rax1 | rdi
rcx0 := rdi
rcx1 := rcx0 ⊕ 0xFFFFFFFFFFFFFFFF
rcx2 := rcx1 & rsi
rdx0 := rdi
rdx1 := rdx0 & rsi
rdx2 := rdx1 ⊕ 0xFFFFFFFFFFFFFFFF
rdi0 := rdi | rsi
rax3 := rax2 + rcx2
rax4 := rax3 − rdx2
rax5 := rax4 + rdi0
```

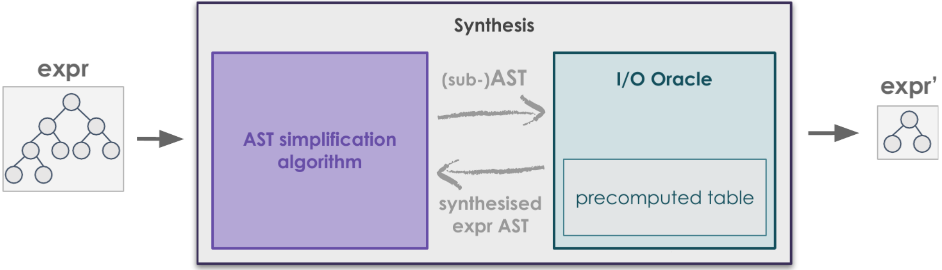
rax5 →

AST




⇒ 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

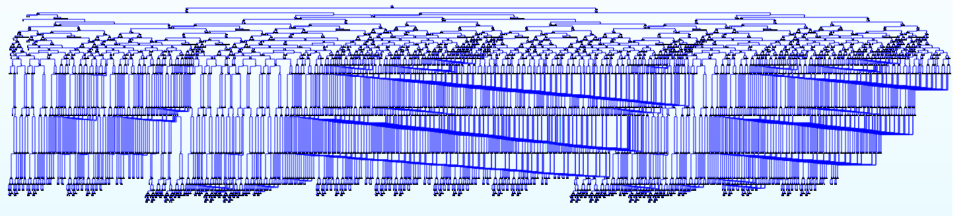


Figure: MBA extracted from messaging application

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

Quarkslab

# Program Exploration

**Use-Case #2**

In support of fuzzing to assess static analysis alerts

# Program Exploration

**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 uses static analyzer for security or compliance before shipping their code *(or requires sub-contractors to do so)*

**Underlying Problem**

$\Rightarrow$ Static analyzers usually yield **many alerts** for which it is difficult to **discriminate** true flaws and **false positives**.

# Static Analysis

## Features

- **Langages:** 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

$\Rightarrow$ **Usually *de-facto* standard for compliance in some automotive, industrial systems.**

# Klocwork Report

#5116: Array 'buffer' of size 2049 may use index value(s) 0..2062
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/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_tcp/cyclone_tcp/ipv4/ipv4_frag.c:434 | ipv4ReassembleDatagram()
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_tcp/cyclone_tcp/http/http_client.c:449 | httpClientSetMethod()
Code: SV.STRBO.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_tcp/cyclone_tcp/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_tcp/cyclone_tcp/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_tcp/cyclone_tcp/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


#5116: Array 'buffer' of size 2049 may use index value(s) 0..2062
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/http/http_client.c:577 | h
Code: ABV.GENERAL | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ | Ow
#5139: Pointer 'datagram' returned from call to function 'netBufferAt' at line 431 may be NU
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/ipv4/ipv4_frag.c:434 | ip
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ |
#5155: function 'strcpy' does not check buffer boundaries but outputs to buffer 'context->me
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/http/http_client.c:449 | h
Code: SV.STRBO.UNBOUND_COPY | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy:
#5321: Pointer 'segment2' returned from call to function 'netBufferAt' at line 349 may be NU
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/core/tcp_misc.c:352 | tc
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ |
#5342: Pointer 'arpRequest' returned from call to function 'netBufferAt' at line 909 may be N
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/ipv4/arp.c:912 | arpSend
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ |
#5396: Pointer 'vlanTag' returned from call to function 'netBufferAt' at line 222 may be NULL
/home/user/work/PASTIS/programme_etalon_v4/cyclone_tcp/cyclone_tcp/core/ethernet_misc.c:22
Code: NPD.FUNC.MUST | Severity: Critical (1) | State: Existing | Status: Analyze | Taxonomy: C and C++ |

Klocwork report (HTML)

`kl_report_to_json`

```
[{ "kid": 5116,
   "params": ["buffer", "2049"],
   "taxonomy": "C and C++",
   "severity": "Critical",
   "file": "/home/user/src/http/http_client.c",
   "line": 577,
   "function": "httpClientSetHost",
   "raw_line": "Array 'buffer' of size 2049
                may use index value(s) 0..2062",
   "code": "ABV_GENERAL"
 },
  ...]
```

Klocwork report (JSON)

```
__klocwork_alert_placeholder(8, "SV_STRBO_BOUND_COPY_OVERFLOW", sizeof(conn->request), token, 71);
strncpy(conn->request, token, n);
```

`kl_alert_inserter`

## 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 $\rightarrow$ last intrinsic covered

## DSE [might cover deeper states]

- Coverage: detect the call to the intrinsic
- Validation: dedicated runtime or symbolic checkers *(sanitizers)*

$\Rightarrow$ **Corollary issue**: How combining them efficiently ?

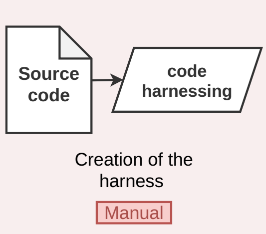
# Symbolic Checker `SV_STRBO_BOUND_COPY_OVERFLOW`

```
__klocwork_alert_placeholder(8, "SV_[..]_OVERFLOW", sizeof(con->request), src, n);
strncpy(con->request, src, n);
```

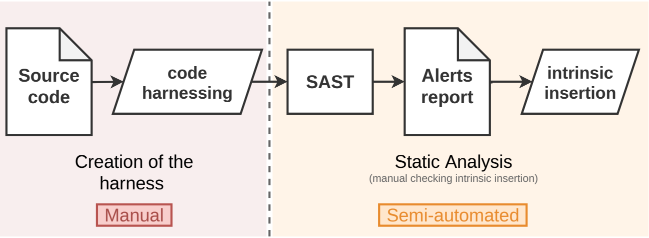
```
def handle_svstrbo_bound_copy_ov(se) -> bool: # se is symbolic state
    dst_size = se.get_argument_value(2)
    ptr_inpt = se.get_argument_value(3)
    n, sym_n = se.get_full_argument(4) # both concrete and symbolic value
    # Runtime check
    if n >= dst_size and len(se.get_memory_string(ptr_inpt)) >= 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
```

$\Rightarrow$ Can flag input as “crashing” even though the harness is not crashing *per-se*.

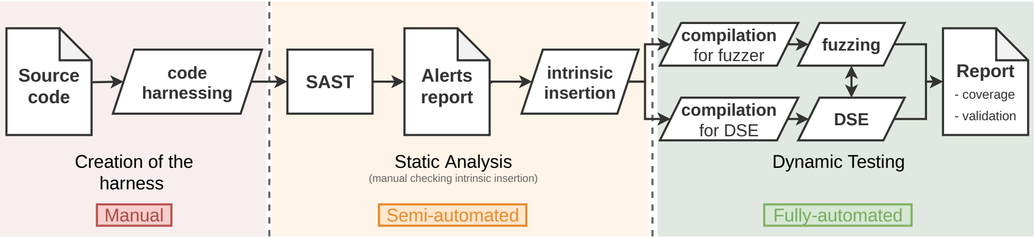
# Complete Workflow



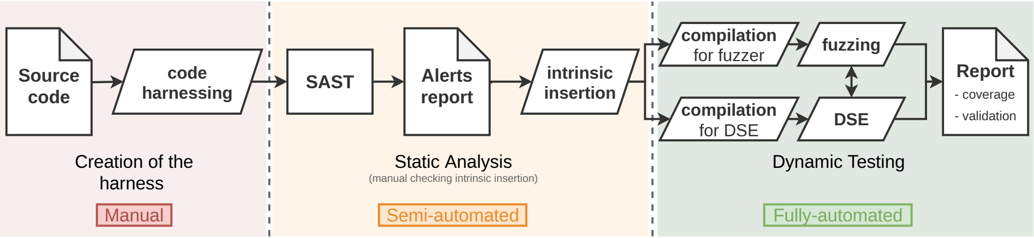
# Complete Workflow



# Complete Workflow



# Complete Workflow




- 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..

$\Rightarrow$ **It might be valuable to combine different test engines**

*(existing litterature [7, 9, 2, 6, 10])*

# Our project: PASTIS

## Characteristics

- written in Python
- distributed *(network-based)*
- run engines in parrallel
- enable adding new fuzzers
- DSE: `Triton`
- fuzzing: `Honggfuzz`, `AFL++`
- replay *(ensure replayability)*

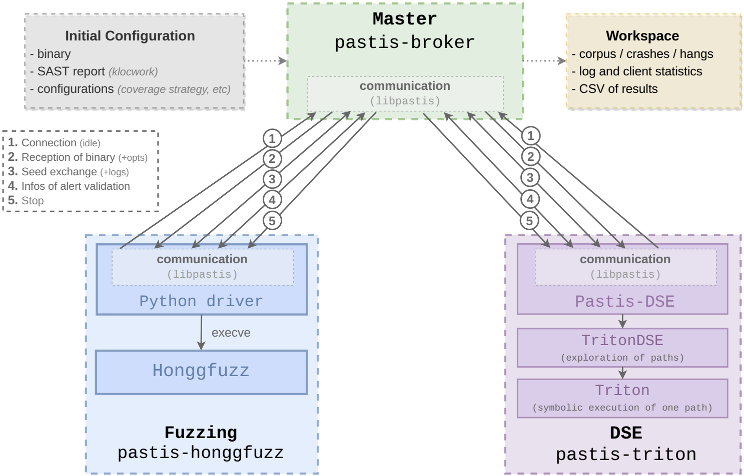
PASTIS

*(pastis is anise-based french liquor)*

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

$\Rightarrow$ Designed to work binary-only targets (in this case cannot leverage intrinsic mechanism)

# PASTIS Architecture



# Demo

# Research & TritonDSE

Quarkslab

# TritonDSE Framework

`TritonDSE` is developped 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

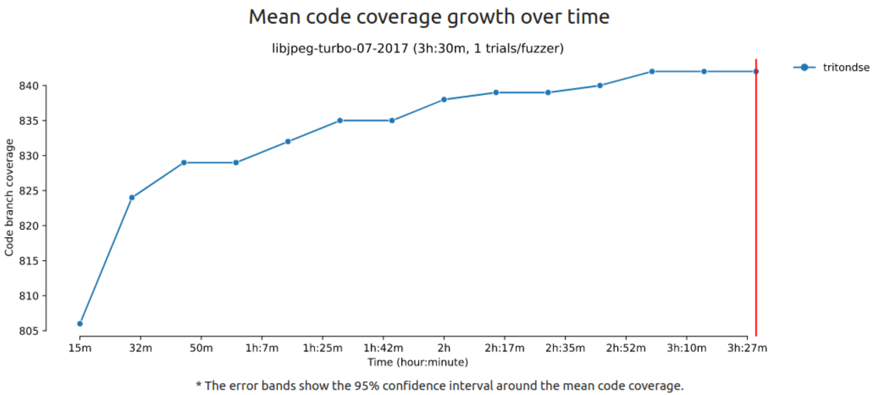
Ongoing experiments with TritonDSE and PASTIS:

- 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


Quarkslab

# 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)*

$$\downarrow$$

```
mov  eax, ds:X
mov  ecx, ds:Y
imul ecx, ecx
imul ecx, 7
sub  ecx, 1
imul eax, eax
cmp  ecx, eax
jz   <dead_addr>
```

$\Rightarrow$ Symbolic execution helps proving the unsatisfiability of the dead branch

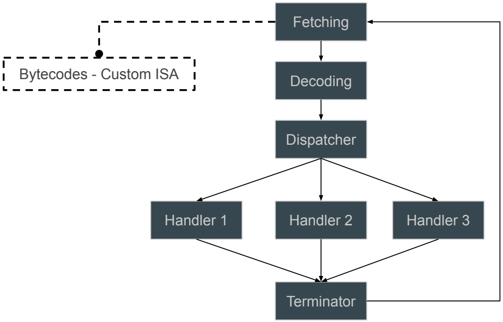
*(now widely studied in litterature [1, 16, 3])*

# Virtualization

**Definition:**

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

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




$\Rightarrow$ Can be defeated by the low interaction between VM code and "real" code [12].

# Existing Frameworks

## ClusterFuzz [7]

**Bio:**

- Authors: Google
- Base: libfuzzer

Used by OSS-Fuzz [8]

[link]

## OneFuzz [9]

**Bio:**

- Author: Microsoft
- Base: AFL, Radamsa

**Pros/Cons:**

- scale
- require an Azure cloud instance

[link]

## EnFuzz [2]

**Bio:**

- Author: Tsinghua University

**Pros/Cons:**

- support AFL, libfuzzer, aflfast, intefuzz, fairuzz..
- academic tool
- a single commit
- basic seed sharing *(local directory)*

[link]

## Deepstate [6]

**Bio:**

- Author: TrailofBits
- Base: libfuzzer, AFL, Honggfuzz, Eclipser, Angora

**Pros/Cons:**

- unified harness *(GTest like)*
- unmaintained
- require fuzzer restart on new seed

[link]

## CollabFuzz [10]

**Bio:**

- Author: Vusec (TU University)
- Base: AFL, AFL++, QSym, AFLfast, Fairfuzz, Honggfuzz, libfuzzer

**Pros/Cons:**

- Based on Docker
- message exchange with ZeroMQ

[link]

# References I


S. Bardin, R. David, and J. Marion, *Backward-bounded DSE: targeting infeasibility questions on obfuscated codes*, in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, 2017, pp. 633–651.

Y. Chen, Y. Jiang, F. Ma, J. Liang, M. Wang, C. Zhou, X. Jiao, and Z. Su, *Enfuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers*, in 28th USENIX Security Symposium, Santa Clara, CA, USA, 2019, USENIX Association, 2019, pp. 1967–1983.
[site].

C. S. Collberg, C. D. Thomborson, and D. Low, *Manufacturing cheap, resilient, and stealthy opaque constructs*, in POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998, D. B. MacQueen and L. Cardelli, eds., ACM, 1998, pp. 184–196.

R. David, *Greybox program synthesis: A new approach to attack dataflow obfuscation*, Black Hat USA, (2021).
[slides].

R. David, L. Coniglio, and M. Ceccato, *Qsynth - a program synthesis based approach for binary code deobfuscation*, (2020).
http://archive.bar/pdfs/bar2020-preprint9.pdf.

P. Goodman, G. Grieco, and A. Groce, *Tutorial: Deepstate: Bringing vulnerability detection tools into the development cycle*, in 2018 IEEE Cybersecurity Development, SecDev 2018, Cambridge, MA, USA, September 30 - October 2, 2018, IEEE Computer Society, 2018, pp. 130–131.

Google, *Clusterfuzz - scalable fuzzing infrastructure*.
[code].

# References II


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

MICROSOFT, *Onefuzz - a self-hosted fuzzing-as-a-service platform*, 2021.
[code].

S. ÖSTERLUND, E. GERETTO, A. JEMMETT, E. GÜLER, P. GÖRZ, T. HOLZ, C. GIUFFRIDA, AND H. BOS, *Collabfuzz: A framework for collaborative fuzzing*, in Proceedings of the 14th European Workshop on Systems Security, EuroSec '21, 2021, p. 1–7.

R. T. QUARKSLAB, *Lief - library to instrument executable formats.*
[site], April 2017.

J. SALWAN, S. BARDIN, AND M. POTET, *Symbolic deobfuscation: From virtualized code back to the original*, in Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings, 2018, pp. 372–392.

R. SASNAUSKAS, Y. CHEN, P. COLLINGBOURNE, J. KETEMA, J. TANEJA, AND J. REGEHR, *Souper: A synthesizing superoptimizer*, CoRR, abs/1711.04422 (2017).

N. STEPHENS, J. GROSEN, C. SALLS, A. DUTCHER, R. WANG, J. CORBETTA, Y. SHOSHITAISHVILI, C. KRUEGEL, AND G. VIGNA, *Driller: Augmenting fuzzing through selective symbolic execution*, in 23rd Annual Network and Distributed System Security Symposium, NDSS, 2016.

# References III


R. THOMAS, *Droidguard: A deep dive into safetynet*, in Symposium sur la sécurité des technologies de l'information et des communications, SSTIC, France, Rennes, June 2-5 2022, SSTIC, 2015, pp. 31–54.
[slides].

R. TOFIGHI-SHIRAZI, I. M. ASAVOAE, P. ELBAZ-VINCENT, AND T. LE, *Defeating opaque predicates statically through machine learning and binary analysis*, in Proceedings of the 3rd ACM Workshop on Software Protection, SPRO@CCS 2019, London, Uk, November 15, 2019, P. Falcarin and M. Zunke, eds., ACM, 2019, pp. 3–14.

I. YUN, S. LEE, M. XU, Y. JANG, AND T. KIM, *QSYM : A practical concolic execution engine tailored for hybrid fuzzing*, in 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, 2018, USENIX Association, pp. 745–761.
[site].