From source code to crash test-cases through software testing automation

C&ESAR Conference – Nov 16th, 2021

Robin David Quarkslab <rdavid@quarkslab.com>
Jonathan Salwan
Justin Bourroux DGA-MI

Quarkslab

Introduction

Quarkslab

Research Problem

Problem

Most static analyzers yield **many alerts**
for which it is difficult
to **discriminate** true flaws and false positives.

⇒ Thus all alerts have to be reviewed **manually**.

Context

This research was performed as part of the project:

PASTIS

Programme d'Analyse Statique et de Tests Instrumentés pour la Sécurité

Infos:

- **Initiator**: *Direction Générale de l'Armement Maîtrise de l'Information* in 2018
- **Objectives**: Automating bug research to facilitate vulnerability research
- **Innovation**: Combining static analysis, fuzzing, symbolic execution and slicing
- **Expectation**: Gaining time and automating what can be done

Contributions

(semi-) Automated testing infrastructure combining static analysis, DSE and fuzzing

But also:

- Experimental study of techniques and tools *(on a common benchmark)*
- Implementation in a Python framework *(PASTIS)*
- Experimental results on COTS softwares *(TCP/IP stack)*

State-of-the-Art: Existing techniques and tools

Quarkslab

Objectives

- Studying State-of-the-Art software testing techniques *(fuzzing, DSE)*
- Benchmarking promising utilities identified *(pre-filtering)*
- Based on results proposing different combinations of utilities *(chosen by DGA)*

Fuzzing

60 tools

	Général				Pré trait.			Géné. d'entrées				Exécution de la cible					
	black-box	gray-box	Open source	Code source requis	Statique	Dynamique	Ordonnancement	Grammaire	Appel système	Protocol réseau	Mutation	Fork-serveur	In-memory fuzzing	Déduplication	Priorisation	Evolutionnaire	Réduction taille
AFL [383]		✓	✓		✓	✓	✓				✓	✓	✓	✓	✓	✓	✓
AFLfast [39]		✓	✓		✓		✓				✓	✓	✓	✓	✓	✓	✓
AFLGo [40]		✓	✓	✓	✓		✓				✓	✓	✓	✓	✓	✓	✓
AFLSmart [283]		✓	✓	✓	✓		✓	✓			✓	✓	✓	✓	✓	✓	✓
AssetFuzzer [220]		✓		✓		✓											
AtomFuzzer [273]		✓	✓	✓	✓												
BFF [348]	✓		✓				✓				✓			✓			
boofuzz [277]	✓		✓					✓		✓							
CalFuzzer [309]		✓	✓	✓	✓												
Choronzon [396]		✓	✓			✓					✓				✓	✓	
⋮	⋮							⋮					⋮				⋮
SymFuzz [63]	✓		✓			✓					✓			✓			
syzkaller [352]		✓	✓	✓		✓		✓	✓		✓			✓		✓	
TLS-Attacker [325]	✓		✓							✓	✓						
TriforceAFL [152]		✓	✓			✓	✓				✓	✓	✓	✓	✓	✓	✓
UnTracer-AFL [262]		✓	✓		✓	✓	✓				✓	✓	✓	✓	✓	✓	✓
VeriFuzz [79]		✓		✓	✓		✓	✓			✓	✓				✓	
zzuf [174]	✓		✓								✓						

⇒

Pre-selected tools:

- AFL 2.52b
- AFL/QBDI
- Honggfuzz 1.7

DSE: Dynamic Symbolic Execution

24 tools

	Caractéristiques									Exec.		Couv. chemins				Contrainte		
	Open-source	Type		Exec.		Charge		Direction		Type		Stratégie			Dirigé			
		Source	Binaire	Symbolique	Concolique	Programme	Par insts.	Avant	Arrière	Online	Offline	DFS	BFS	Autres.		Solveur SMT	Bit-vecteurs	Tableaux
AEC [15]		✓			✓	✓		✓						✓		✓	✓	
angr [320]	✓		✓	✓	✓	✓		✓		✓		✓	✓	✓		✓	✓	✓
BAP [49]	✓		✓		✓		✓	✓								✓	✓	✓
Binsec [99]	✓		✓		✓		✓	✓	✓		✓	✓	✓			✓	✓	✓
BitBlaze [326]	✓		✓		✓		✓	✓			✓		✓			✓	✓	
Bouncer [91]		✓		✓				✓								✓	✓	
Cloud9 [52]	✓	✓			✓	✓		✓		✓		✓	✓	✓		✓	✓	✓
CREST [53]	✓	✓			✓	✓		✓				✓		✓		✓	✓	
CUTE [311]		✓		✓		✓		✓				✓				✓	✓	
⋮		⋮						⋮							⋮			
Pathgrind [315]	✓		✓		✓		✓	✓								✓	✓	
Reven [340]			✓		✓		✓	✓								✓	✓	
SAGE [139]			✓		✓		✓	✓			✓			✓		✓	✓	
S2E [77]	✓		✓		✓	✓		✓		✓						✓	✓	✓
Triton [304]	✓		✓	✓	✓		✓	✓		✓	✓					✓	✓	

⇒

Pre-selected tools:

- `angr` 8.18
- `manticore` 0.2.5
- KLEE 2.1
- `Triton` 0.7

Building a test-suite

Problematic

Fuzzing and DSE are working **very differently** thus testing their **idiosyncratic** behaviors in a common test-suite is difficult. Also:

- getting ground truth *(inputs triggering the bug)*
- make tests automatic and reproducible

Existing test-suite

- Verisec [6]
- Toyota ITC bench [7]
- SPEC
- SARD
- Juliet [1]
- Logic Bombs [9] *(DSE oriented)*
- CGC challenge [5]
- fuzzer-test-suite *(by Google)*
- BugZoo [8] *(by Squares)*
- Hemiptera
- LinuxFlaw *(275CVE et 20EDB)*
- LAVA-M [4]

⇒ **None of them entirely solves aformentioned problems**

Our Test Suite

Atomic *(synthetic)* tests

Source

Logic Bombs, **program-verification-samples**

Unit tests assessing specific behavior:

- `UT_1`: Path predicate computation, memory modeling
- `UT_2`: Input symbolisation, constraints modeling
- `UT_3`: Program exploration *(loop handling etc.)*
- `UT_4`: Bug discovery *(BoF, off-by-one, UaF etc.)*

Total tests: **70**

Scalability tests

Source

LAVA-M[4]

Binaries: `uniq` and `base64`

Why:

- ground-truth available
- quantitative results *(thus discriminating)*

Total bugs: **72**

Benchmark Results

	Atomic (70)	**Scale** (72)	**Total** (142)
`AFL`	48	0	48/142
`Honggfuzz`	54	44	**100/142**
`AFL/QBDI`	47	33	80/142
`manticore`	34	0	34/142
`KLEE`	47	1	**48/142**
`angr`	37	0	37/142
`Triton`	47	0	47/142

State-of-the-Art Conclusion

Combination proposals

- **Proposal #1**: `Triton` and `Honggfuzz` *(both very modular and complete understanding of* `Triton`*)*
- **Proposal #2**: `KLEE` and `Honggfuzz` *(best on benchmarks, LLVM based for* `KLEE`*)*
- **Proposal #3**: `Honggfuzz` and `Qsym`, *(promising approach but rather exploratory)*

⇒ **Combination #1 has been selected**

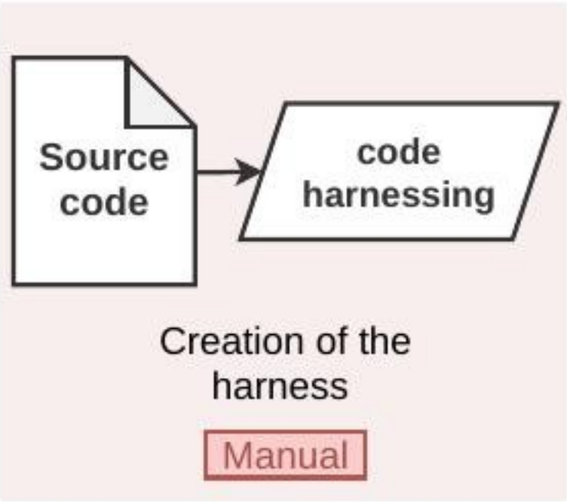
SOTA Report *(339 pages)*

(In the report we also test hybrid approaches like `Qsym` *[10],* `Angora` *[2] and* `Eclipser` *[3])*

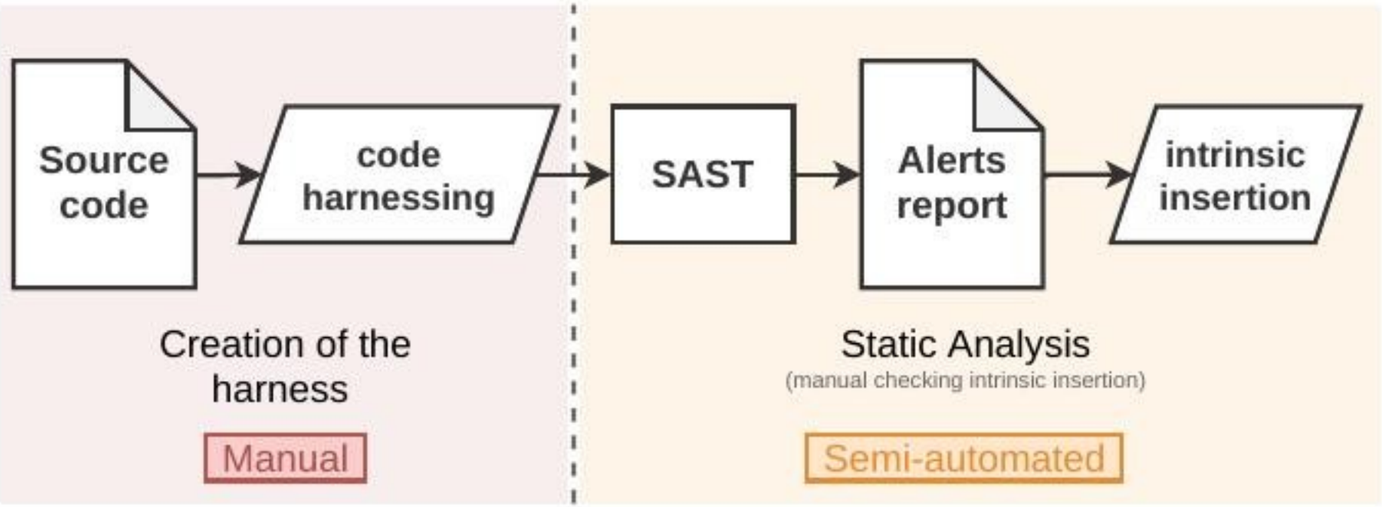
Automating Software Testing

Quarkslab

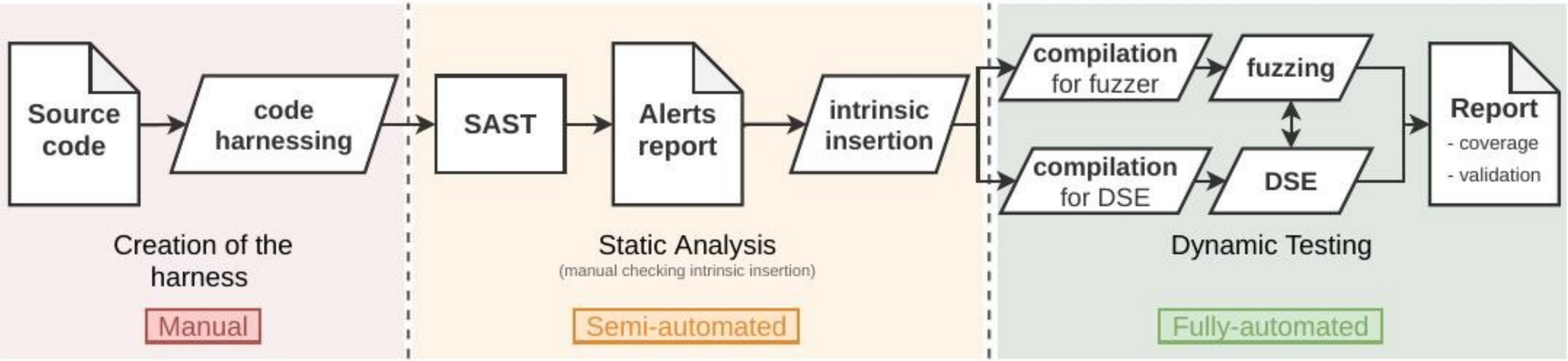
Complete Workflow



Complete Workflow



Complete Workflow



Static Analyzer (SAST)

Functionalities

- **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

Intrinsic functions insertion

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)

kl_alert_inserter

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

Intrinsic Processing

Fuzzing:

- Coverage: by parsing `stdout` *(intrinsic function print on standard output)*
- Validation: in case of crash, last intrinsic covered

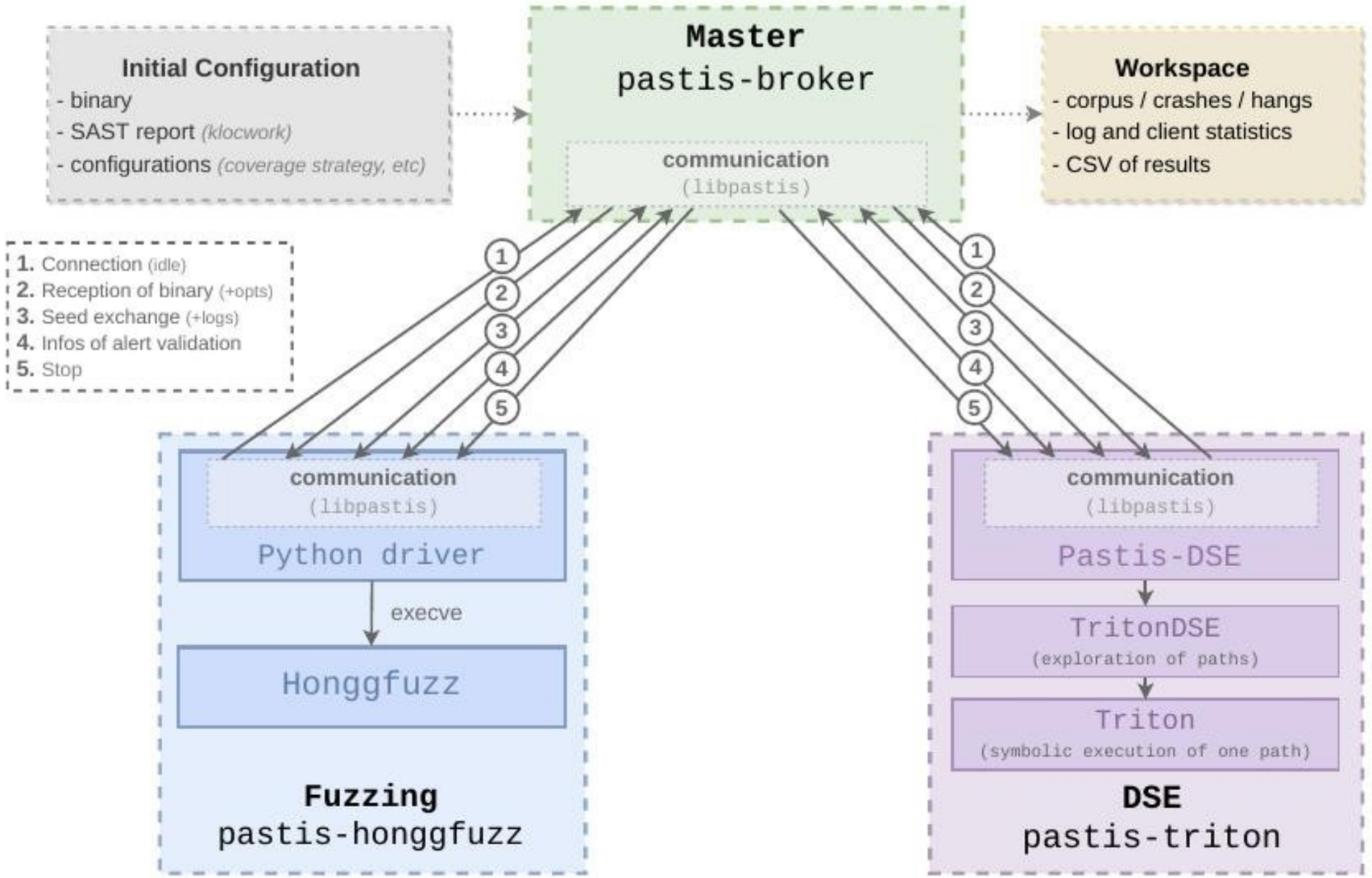
DSE:

- Coverage: shall detect the call
- Validation: DSE specific concrete or symbolic checks

Implementation in PASTIS

Quarkslab

PASTIS Framework



Python Honggfuzz driver

Based on

Honggfuzz 2.3.1

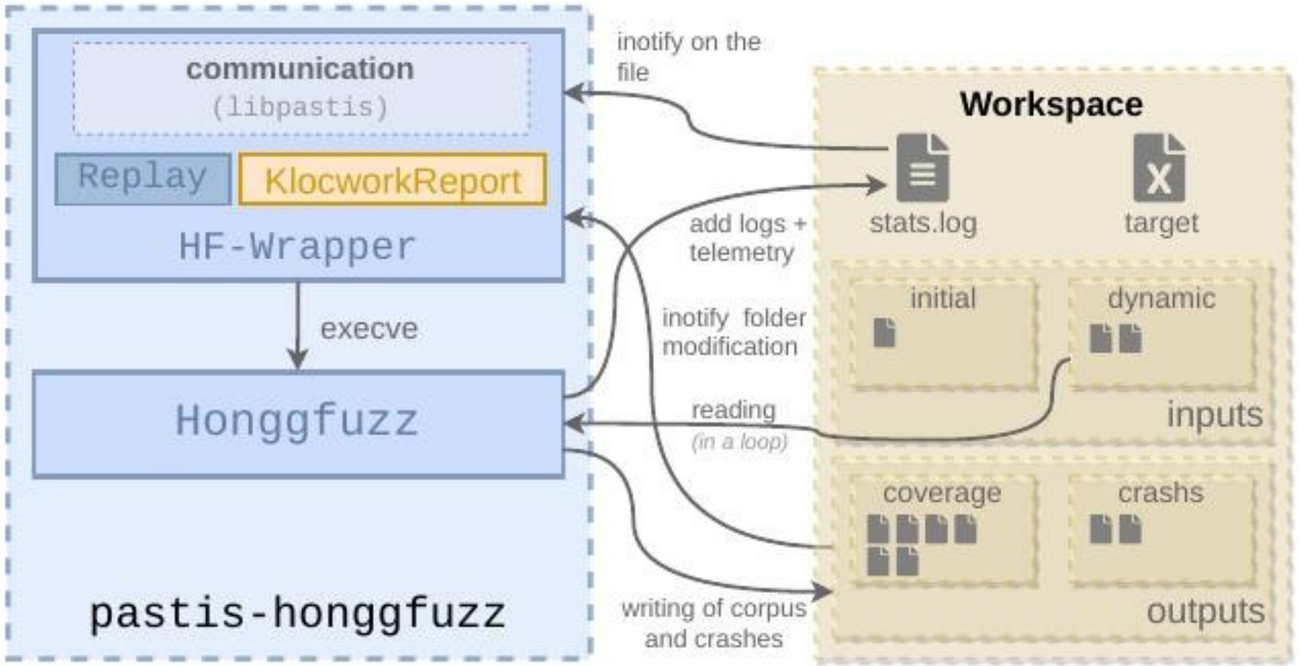
(modified for PASTIS needs)

Infos:

- use mutations integrated in HF
- exchanges with main process through `inotify`

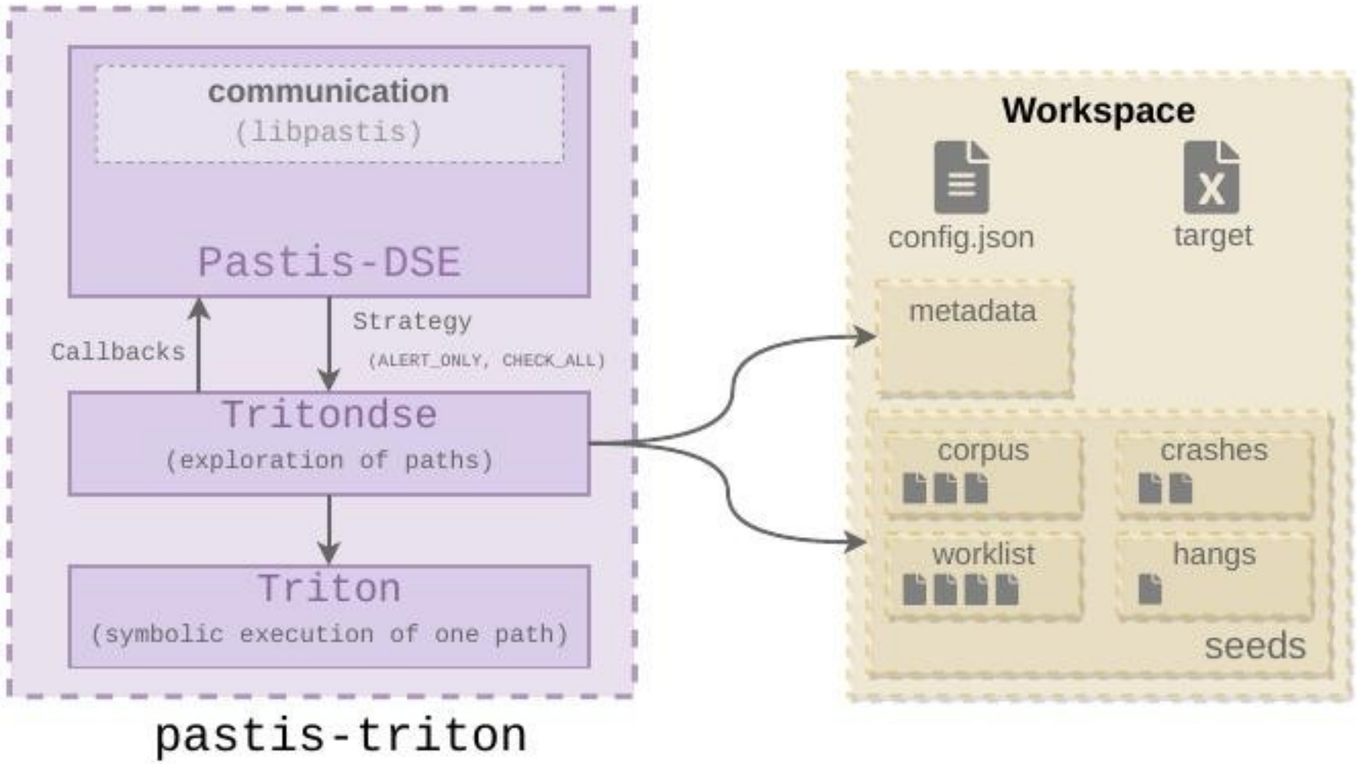
Replay:

- can test reproducibility
- retrieve covered alerts
- in case of crash associate it to last intrinsic



Triton integration

Problem: Triton is per path DSE, not a fully-featured **whitebox fuzzer**



TritonDSE

⇒ For modularity works in pure emulation *(not concolic)*

Functionality to implement for a “fully-featured” fuzzer

- program loading *(ELF)*
- input seed scheduling
- program exploration & coverage computation
- dynamic & symbolic sanitizers *(for different vulnerability categories)*
- basic memory allocator *(with alloc and free primitives)*
- basic multi-threading support
- libc function modeling *(∼58 functions)*

⇒ Developped as a **Python library** based on a **callback** mechanism.

Experimental Results

Quarkslab

Experimental settings

Defect

Code construct generating a `Klocwork` alert but which cannot be triggered in practice $\rightarrow$ **false positive**.

Vulnerability

Code construct generating an alert AND which can be detected and triggered by a sanitizer $\rightarrow$ **true positive**.

$\Rightarrow$ **All defect vulnerabilities are introduced manually in the target.**

CycloneTCP stack

HTTP Server with Cyclone **1.9.6**

(website with a single static page (using virtual fs)

CycloneTCP

Protocols

- IPv4, ARP, ICMP, IGMP, TCP, HTTP
- VLAN, VMAN, port-tagging, IPv6, mDNS, DHCP, LLMNR, NBNS

HTTP Configuration

- multipart types support
- TLS, auth, digests, websockets, GZIP, cookies

Results (24h campain)

id	Kid	Typ.	D	V	Proto.	Honggfuzz		Triton	
						Cov.	Val.	Cov.	Val.
1	5922	OB1		•	HTTP	✓	✓	✗	✗
2	5357	FMT		•	HTTP	✓	✓	✗	✗
3	6562	IoF	•		HTTP	✓	-	✓	-
4	✗	BoF	•		HTTP	-	-	-	-
5	9047	FMT		•	HTTP	✓	✗	✗	✗
6	✗	UaF	•		HTTP	-	-	-	-
7	5851	BoF		•	HTTP	✓	✓	✓	✓
8	5848	BoF		•	HTTP	✓	✓	✓	✓
9	9054	FMT	•		HTTP	✓	-	✗	-
10	9044	FMT		•	HTTP	✓	✗	✗	✗
11	✗	OB1		•	HTTP	-	-	-	-
12	9056	IoF	•		IPv4	✓	-	✗	-
13	6542	SIGS		•	ARP	✓	✓	✓	✓
14	5418	SIGS	•		ICMP	✓	-	✓	-
15	✗	BoF		•	ICMP	-	-	-	-
16	5645	UaF		•	IPv4	✓	✓	✓	✗
17	8640	OB1	•		core	✓	-	✓	-
18	8085	SIGS	•		ETH.	✓	-	✗	-
19	8579	UaF		•	IGMP	✓	✓	✓	✓
20	✗	IoF		•	ICMP	-	-	-	-
					Total	15/15	7/9	8/15	4/9

Conclusion

Limitations

- Fuzzing generates large number of inputs that are costly to run by the DSE
- DSE in pure-emulation requires modeling syscalls and library calls which hardly scale to any targets.

CVE-2021-26788

Remote DOS CycloneTCP

- Impacted version: **1.7.6** to **2.0.0**
- Impact: Remote-DOS
- Reason: Missing checks of “size” field in TCP options. If 0, the function enter in an infinite loop.

CVE-2021-26788 Detail

Current Description

Oryx Embedded CycloneTCP 1.7.6 to 2.0.0, fixed in 2.0.2, is affected by incorrect input validation, which may cause a denial of service (DoS). To exploit the vulnerability, an attacker needs to have TCP connectivity to the target system. Receiving a maliciously crafted TCP packet from an unauthenticated endpoint is sufficient to trigger the bug.

+View Analysis Description

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 7.5 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://github.com/Oryx-Embedded/CycloneTCP/commit/de5336016edbe1e90327d0ed1cba5c4e49114366?branch=de5336016edbe1e90327d0ed1cba5c4e49114366&diff=split	Patch, Third Party Advisory

https://blog.quarkslab.com/remote-denial-of-service-on-cyclonetcp-cve-2021-26788.html

Conclusion

We **automated** most of the workflow from the source code
to the dynamic testing
in order to **facilitate analysis and triaging** for the analyst.

Conclusion

We **automated** most of the workflow from the source code
to the dynamic testing
in order to **facilitate analysis and triaging** for the analyst.

Future work

- studying the added-value of combining approaches *(against running them separately)*
- introducing slicing to “guide” exploration toward alerts

Thank you !

Contact:

Email: rdavid@quarkslab.com

Phone: +33 1 58 30 81 51

Site: https://www.quarkslab.com

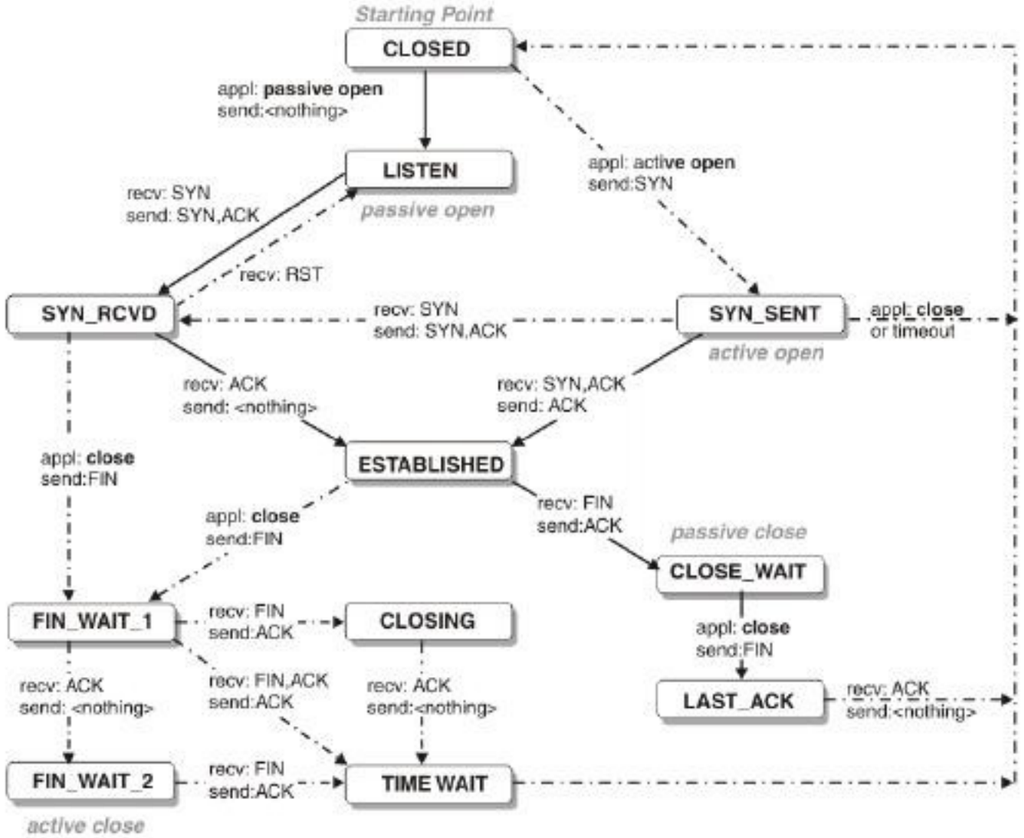
Quarkslab

Quarkslab

Use-case: TCP/IP stacks

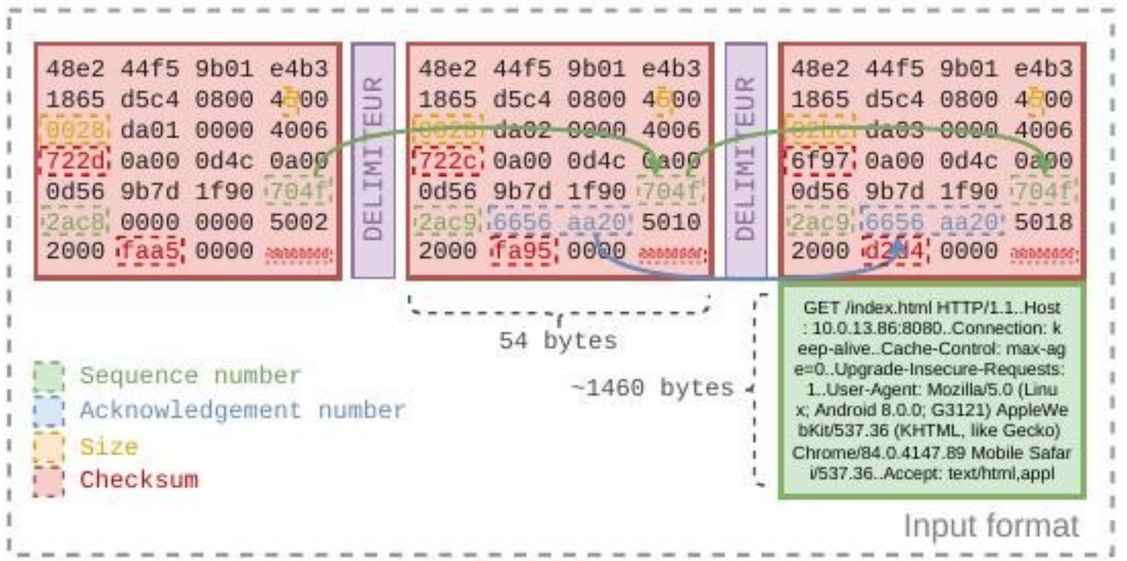
Challenge #1

Program **highly stateful**



Challenge #2

Input **highly heterogenous**



Harnessing

Sequencing processing

By default the stack is multi-threaded. Sequencing enables:

- faster execution and easier persistent fuzzing
- remove non-determinism induced by threads and replayability issues

Harnessing

Sequencing processing

By default the stack is multi-threaded. Sequencing enables:

- faster execution and easier persistent fuzzing
- remove non-determinism induced by threads and replayability issues

Input handling

Uses the stack driver mechanism, to parse the input file, splitting each ethernet frames and sending them to the stack *(one after another)*.

Harnessing

Sequencing processing

By default the stack is multi-threaded. Sequencing enables:

- faster execution and easier persistent fuzzing
- remove non-determinism induced by threads and replayability issues

Input handling

Uses the stack driver mechanism, to parse the input file, splitting each ethernet frames and sending them to the stack *(one after another)*.

Other modifications

- remove randomness of ISN *(Initial Sequence Number)*
- disabling checksums *(in ETH, IP, TCP)*
- pre-registration of client ARP lease

References I


T. Boland and P. E. Black, *Juliet 1.1 C/C++ and java test suite*, IEEE Computer, 45 (2012), pp. 88–90.


P. Chen and H. Chen, *Angora: Efficient fuzzing by principled search*, 2018 IEEE Symposium on Security and Privacy (SP), (2018), pp. 711–725.


J. Choi, J. Jang, C. Han, and S. K. Cha, *Grey-box concolic testing on binary code*, in Proceedings of 41th International Conference on Software Engineering, 2019.
[PDF].


B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robertson, F. Ulrich, and R. Whelan, *Lava: Large-scale automated vulnerability addition*, in 2016 IEEE Symposium on Security and Privacy (SP), May 2016, pp. 110–121.
[PDF].


D. Guido, *Your tool works better than mine? prove it.*
[URL].


K. Ku, T. E. Hart, M. Chechik, and D. Lie, *A buffer overflow benchmark for software model checkers*, in 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA, 2007, pp. 389–392.


S. Shiraishi, V. Mohan, and H. Marimuthu, *Test suites for benchmarks of static analysis tools*, in 2015 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Gaithersburg, MD, USA, November 2-5, 2015, 2015, pp. 12–15.

References II

C. S. Timperley, S. Stepney, and C. Le Goues, *Bugzoo: a platform for studying software bugs*, in Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, 2018, pp. 446–447.

H. Xu, Z. Zhao, Y. Zhou, and M. R. Lyu, *Benchmarking the capability of symbolic execution tools with logic bombs*, IEEE Transactions on Dependable and Secure Computing, (2018), pp. 1–1.

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].