ROSA: Finding Backdoors with Fuzzing

47th International Conference on Software Engineering

Dimitri Kokkonis
CEA List
Université Paris-Saclay
IP Paris

Michaël Marcozzi
CEA List
Université Paris-Saclay

Emilien Decoux
CEA List
Université Paris-Saclay

Stefano Zacchiroli
LTCI
Télécom Paris
IP Paris

About backdoors & fuzzing

What is a backdoor?

- Authentication bypass?
- Training data poisoning (ML)?
- Crypto (mathematical flaws)?

Credit: Nikita Korenkov (Pexels)

What is a backdoor?

- Authentication bypass?
- Training data poisoning (ML)?
- Crypto (mathematical flaws)?

We focus on **code-level** backdoors:

- Hidden access (**special input**), concealed within a program:
  - To (more) privileged part of the program **without legitimate authentication**
  - To **forbidden** underlying system resources (e.g., files, root shell)

Credit: Nikita Korenkov (Pexels)

Backdoor attacks

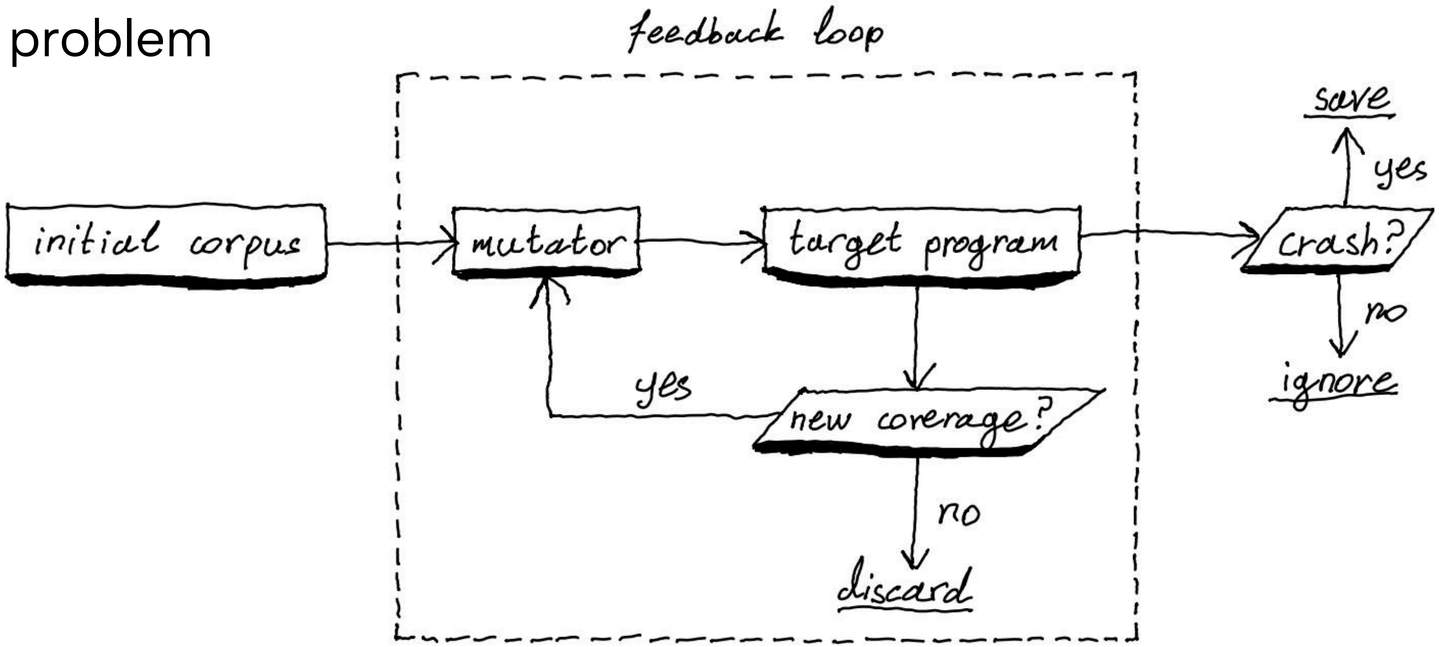
Classic “butterfly effect” of supply-chain attacks:

- **lzma/xz-utils** (2024): complex, dynamic authentication bypass
- **PHP** (2021): hidden command allowing to execute a command as root
- **vsFTPd** (2011): hardcoded credentials in legitimate auth
- **ProFTPD** (2010): hidden command spawing a root shell
- … and a *lot* of router firmware (hidden servers, hardcoded credentials, …)

Credit: Daniel Stori (turnoff.us)

Graybox fuzzing

- Automated bruteforce testing approach
- **Simple** runtime failure detectors (i.e., oracles): crashes, sanitizers, …
- For modern fuzzers (e.g., AFL++):
  - **Proven efficiency** in **discovering vulnerabilities**
  - Efficient **binary program** exploration
  - Mitigated *"magic byte"* problem



Backdoor detection with fuzzing

Expectations

Primary industrial use cases:

- Vetting appliance (e.g., router, camera) firmware entry points before **large-scale** / **security-critical** deployment
- Vetting **third-party** software components before integration into in-house **large-scale** / **security-critical** infrastructure

Credit: Scott Webb (Pexels)

And yet…

- Mainly **manual** (binary) code **reverse engineering** (difficult, not often done)
- A handful of automated approaches have been proposed:
  - The idea is **automating parts of the reverse engineering process**
  - Only focusing on **specific backdoor** and **target program types**
  - **Limited backdoor sample availability** for evaluation (lost/non-functioning artifacts)

Tool	Approach	Target programs	Target backdoor types
WEASEL [1]	Symbolic/concolic execution	Common protocol implementations	Authentication bypass, hidden command
Firmalice [2]	Symbolic execution + path slicing	Any firmware with known "authentication points"	Authentication bypass
HumIDIFy [3]	ML + "model checking"	Common protocol implementations	Divergence from protocol specification
Stringer [4]	Static analysis	Any binary program	Hardcoded credentials

[1] Schuster, Felix, and Thorsten Holz. "Towards reducing the attack surface of software backdoors." In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 851–862. 2013.
[2] Shoshitaishvili, Yan, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. "Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware." In *NDSS*, vol. 1, pp. 1–1. 2015.
[3] Thomas, Sam L., Flavio D. Garcia, and Tom Chothia. "HumIDIFy: a tool for hidden functionality detection in firmware." In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 279–300. Cham: Springer International Publishing, 2017.
[4] Thomas, Sam L., Tom Chothia, and Flavio D. Garcia. "Stringer: Measuring the importance of static data comparisons to detect backdoors and undocumented functionality." In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pp. 513–531. Springer International Publishing, 2017.

Fuzzing-based detector

Credit: AFL++

Graybox fuzzing is a good candidate for a backdoor detection technique:

- Largely **automatic** (no manual binary reverse-engineering)
- Efficient **binary** exploration **for all program types**
- Already **widely used for vulnerability detection** (in academia *and* industry)

Fuzzing-based detector

Credit: AFL++

Graybox fuzzing is a good candidate for a backdoor detection technique:

- Largely **automatic** (no manual binary reverse-engineering)
- Efficient **binary** exploration **for all program types**
- Already **widely used for vulnerability detection** (in academia *and* industry)

But, current state-of-the-art fuzzers **cannot detect backdoors out of the box**:

- Can detect **crashes**, but no known mechanism for **runtime backdoor triggers**
- We need a **specialized oracle** to detect most backdoor triggers

Contributions

Introducing *ROSA*: **graybox fuzzing** (*AFL++*) + **novel metamorphic oracle**

Intuition:

- Similar inputs → **similar behavior**
- Backdoor-triggering inputs → **divergent behavior**

Introducing *ROSA*: **graybox fuzzing** (*AFL++*) + **novel metamorphic oracle**

Intuition:

- Similar inputs → **similar behavior**
- Backdoor-triggering inputs → **divergent behavior**

Introducing *ROSARUM*: a long-overdue **standardized backdoor benchmark**

- 17 programs of various types, with diverse backdoors:
  - 7 **authentic**: reconstructed from the literature
  - 10 **synthetic**: injected in popular open-source programs (MAGMA benchmark)

ROSA on an example

(see paper for a detailed presentation)

Backdoored Sudo (1/3)

Artificial backdoored version of Sudo:

```
$ sudo id
Password: wrong_password
Sorry, try again.
Password: let_me_in
uid=0(root) gid=0(root) groups=0(root)
```

Somewhere in Sudo’s source code:

```
if (strcmp(password, "let_me_in") == 0) return AUTH_SUCCESS;
```

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: "aaa"

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: "aaa"

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: "a\na"

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: "\n\na"

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: "aaa"

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: "a\na"

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: "\n\na"

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\longrightarrow$ B is most similar

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\longrightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

$\rightarrow \equiv B$

[safe]

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

$\rightarrow \equiv B$

[safe]

Input D: `"let_me_in\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✓

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

$\rightarrow \equiv B$

[safe]

Input D: `"let_me_in\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✓

$\rightarrow$ B is most similar

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

$\rightarrow \equiv B$

[safe]

Input D: `"let_me_in\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✓

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✓	✓

Backdoored Sudo (2/3)

Phase 1: fuzzer discovers *representative inputs*

Input A: `"aaa"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✗	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

Input B: `"a\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✗

System calls:

read	write	clone	execve
✓	✓	✗	✗

...

Phase 2: fuzzer intensively explores the input space

Input C: `"\n\na"`

CFG edges:

e_1	e_2	e_3	e_4
✗	✗	✓	✗

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✗	✗

$\rightarrow \equiv B$

[safe]

Input D: `"let_me_in\na"`

CFG edges:

e_1	e_2	e_3	e_4
✓	✗	✓	✓

$\rightarrow$ B is most similar

System calls:

read	write	clone	execve
✓	✓	✓	✓

$\rightarrow \not\equiv B$

[suspicious]

Backdoored Sudo (3/3)

Post–processing: a human expert verifies the suspicious input D **semi–automatically**:

1. Collect **divergent system calls** of D relative to most similar representative input
2. Run Sudo with D under a **tracing program** (like strace)
3. **Filter** only system calls collected in (1)
4. **Manually investigate** system calls and system call **arguments**

Backdoored Sudo (3/3)

Post-processing: a human expert verifies the suspicious input D **semi-automatically**:

1. Collect **divergent system calls** of D relative to most similar representative input
2. Run Sudo with D under a **tracing program** (like strace)
3. **Filter** only system calls collected in (1)
4. **Manually investigate** system calls and system call **arguments**

In the case of D:

- Divergent system calls: $\{..., 56, 59, ...\}$

```
$ strace -fe ...,56,59,... -- sudo id < backdoor-input.txt
...
clone(...)
execve("/usr/bin/id")
```

Backdoored Sudo (3/3)

Post-processing: a human expert verifies the suspicious input D **semi-automatically**:

1. Collect **divergent system calls** of D relative to most similar representative input
2. Run Sudo with D under a **tracing program** (like strace)
3. **Filter** only system calls collected in (1)
4. **Manually investigate** system calls and system call **arguments**

In the case of D:

- Divergent system calls: $\{..., 56, 59, ...\}$

```
$ strace -fe ...,56,59,... -- sudo id < backdoor-input.txt
...
```

`clone(...)` ← fork

`execve("/usr/bin/id")` ← command execution

Backdoored Sudo (3/3)

Post-processing: a human expert verifies the suspicious input D **semi-automatically**:

1. Collect **divergent system calls** of D relative to most similar representative input
2. Run Sudo with D under a **tracing program** (like strace)
3. **Filter** only system calls collected in (1)
4. **Manually investigate** system calls and system call **arguments**

In the case of D:

- Divergent system calls: $\{..., 56, 59, ...\}$

```
$ strace -fe ...,56,59,... -- sudo id < backdoor-input.txt
...
clone(...)
execve("/usr/bin/id")
```

clone(...) ← fork

execve("/usr/bin/id") ← command execution

Successful authentication without legitimate password → **backdoor!**

Evaluation

ROSARUM (backdoor dataset)

Program: Name	Program: Type	Program: Binary size	Backdoor: Origin	Backdoor: Description
Authentic backdoors				
Belkin / httpd	Router HTTP server	2.6 MiB	Router manufacturer	HTTP request with secret URL value leads to web shell [6]
D-Link / thttpd	Router HTTP server	7.2 MiB		HTTP request with secret field value bypasses authentication [7]
Linksys / scfgmgr	Router TCP server	2.5 MiB		Packet with specific payload enables memory read/write [9]
Tenda / goahead	Router HTTP server	2.9 MiB		Packet with specific payload enables command execution [8]
PHP	HTTP server	80.6 MiB	Supply-chain attack	HTTP request with secret field value enables command execution [2]
ProFTPD	FTP server	3.3 MiB		Secret FTP command leads to root shell [3]
vsFTPd	FTP server	2.9 MiB		FTP usernames containing ":)" lead to root shell [4]
Synthetic backdoors				
sudo	Unix utility	8.4 MiB	Paper example	Hardcoded credentials (see Listing 1)
libpng	Image library	7.0 MiB	Manual injection in the MAGMA [22] fuzzing benchmark	Secret image metadata values enables command execution
libsndfile	Sound library	6.6 MiB		Secret sound file metadata value triggers home directory encryption
libtiff	Image library	10 MiB		Secret image metadata value enables command execution
libxml2	XML library	8.2 MiB		Secret XML node format enables command execution
Lua	Language interpreter	3.7 MiB		Specific string values in script enables reading from filesystem
OpenSSL / bignum	Crypto library	12.2 MiB		Secret bignum exponentiation string enables command execution
PHP / unserialize	Language interpreter	30.2 MiB		Specific string values in serialized object enables PHP code execution
Poppler	PDF renderer	39.4 MiB		Secret character in PDF comment enables command execution
SQLite3	Database system	6.4 MiB		Secret SQL keyword enables removal of home directory

Experimental protocol

Standard fuzzing setup:

- Using AFL++ (with AFL++ best practices)
- 10 runs, 8 hours each
- 6 fuzzers in parallel (3 for target program, 3 for **dynamic libraries**)
- Fixed time for phase 1 (1 minute)

Research questions:

RQ1: *Can ROSA detect backdoors in enough* ***diverse contexts****, with enough* ***robustness****,* ***speed*** *and* ***automation****, to make it usable and useful in the wild?*

RQ2: *How does ROSA* ***compare to state-of-the-art*** *backdoor detection tools, in terms of* ***robustness****,* ***speed*** *and* ***automation****?*

Comparison with the state of the art

Tool	Approach	Context	Target programs	Target backdoor types
WEASEL [1]	Symbolic/concolic execution	Reverse-engineering aid	Common protocol implementations (e.g., HTTP)	Authentication bypass, hidden command
Firmalice [2]	Symbolic execution + path slicing	Reverse-engineering aid	Any firmware with known authenticated points	Authentication bypass
HumIDIFy [3]	ML + “model checking”	Reverse-engineering aid	Common protocol implementations (e.g., HTTP)	Divergence from protocol specification
Stringer [4]	Static analysis	Reverse-engineering aid	Any binary program	Hardcoded credentials
ROSA	Fuzzing + metamorphic oracle	Automatic detection + semi-automatic vetting	Any fuzzable binary program	Any backdoor materialized through system calls

[1] Schuster, Felix, and Thorsten Holz. “Towards reducing the attack surface of software backdoors.” In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 851–862. 2013.
[2] Shoshitaishvili, Yan, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. “Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware.” In *NDSS*, vol. 1, pp. 1–1. 2015.
[3] Thomas, Sam L., Flavio D. Garcia, and Tom Chothia. “HumIDIFy: a tool for hidden functionality detection in firmware.” In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 279–300. Cham: Springer International Publishing, 2017.
[4] Thomas, Sam L., Tom Chothia, and Flavio D. Garcia. “Stringer: Measuring the importance of static data comparisons to detect backdoors and undocumented functionality.” In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11–15, 2017, Proceedings, Part II 22*, pp. 513–531. Springer International Publishing, 2017.

Comparison with the state of the art

Tool	Approach	Context	Target programs	Target backdoor types
WEASEL [1]	Symbolic/concolic execution	Reverse-engineering aid	Common protocol implementations (e.g., HTTP)	Authentication bypass, hidden command
Firmalice [2]	Symbolic execution + path slicing	Reverse-engineering aid	Any firmware with known authenticated points	Authentication bypass
HumIDIFy [3]	ML + "model checking"	Reverse-engineering aid	Common protocol implementations (e.g., HTTP)	Divergence from protocol specification
Stringer [4]	Static analysis	Reverse-engineering aid	Any binary program	Hardcoded credentials
ROSA	Fuzzing + metamorphic oracle	Automatic detection + semi-automatic vetting	Any fuzzable binary program	Any backdoor materialized through system calls

[1] Schuster, Felix, and Thorsten Holz. "Towards reducing the attack surface of software backdoors." In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 851-862. 2013.
[2] Shoshitaishvili, Yan, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. "Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware." In *NDSS*, vol. 1, pp. 1-1. 2015.
[3] Thomas, Sam L., Flavio D. Garcia, and Tom Chothia. "HumIDIFy: a tool for hidden functionality detection in firmware." In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 279-300. Cham: Springer International Publishing, 2017.
[4] Thomas, Sam L., Tom Chothia, and Flavio D. Garcia. "Stringer: Measuring the importance of static data comparisons to detect backdoors and undocumented functionality." In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pp. 513-531. Springer International Publishing, 2017.

Results: robustness

Backdoor	ROSA — (10 runs × 8 hours) / backdoor — 1 minute of fuzzing for phase 1: Robustness + speed				Automation level				STRINGER	
	Failed runs	Time to first backdoor input: Min.	Avg.	Max.	*Baseline* Avg. seeds	*Manually inspected inputs* Min.	Avg.	Max.	*Backdoor detection time*	*Manually inspected strings*
Authentic backdoors										
Belkin / httpd	10 / 10	Timeout	Timeout	Timeout	2773	2	4	6	Not found	**0**
+ *with specialized seeds**	**3 / 10**	**17m40s**	**3h49m29s**	Timeout	2781	4	5	7	Not found	**0**
D-Link / thttpd	**0 / 10**	**2m07s**	**15m00s**	**43m42s**	3648	**7**	**9**	**12**	Not found	113
Linksys / scfgmgr	**0 / 10**	**1m05s**	**1m29s**	**1m55s**	251	1	1	1	Not found	**0**
Tenda / goahead	**0 / 10**	**1m28s**	**3m34s**	**8m10s**	535	**1**	**2**	**2**	Not found	290
PHP	1 / 10	24m30s	2h03m44s	Timeout	11631	**4**	**8**	**16**	**6m**	573
ProFTPD	4 / 10	4m03s	3h37m32s	Timeout	2995	**5**	**8**	**11**	**7s**	314
vsFTPd	**0 / 10**	[illegible]	[illegible]	[illegible]	[illegible]	**3**	**4**	**4**	Not found	117
[illegible]										
sudo	**0 / 10**	[illegible]	[illegible]	[illegible]	[illegible]	**1**	**1**	**1**	Not found	137
libpng	2 / 10	13m47s	2h24m46s	Timeout	4202	**1**	**2**	**2**	**4s**	9
libsndfile	3 / 10	2h21m08s	5h04m46s	Timeout	10376	9	12	13	**5s**	**8**
libtiff	**0 / 10**	**5m08s**	**12m15s**	**25m10s**	9566	**1**	**3**	**5**	Not found	31
libxml2	**0 / 10**	**8m17s**	**27m14s**	**1h09m06s**	12104	**9**	**14**	**20**	Not found	1208
Lua	**1 / 10**	**50m34s**	**4h07m41s**	Timeout	6653	**6**	**12**	**17**	Not found	36
OpenSSL / bignum	**0 / 10**	**9m53s**	**22m00s**	**39m52s**	1441	**1**	**1**	**2**	Not found	657
PHP / unserialize	**0 / 10**	**23m05s**	**1h04m39s**	**1h35m08s**	6285	**1**	**1**	**1**	Not found	974
Poppler	**0 / 10**	**11m28s**	**49m09s**	**1h33m02s**	9544	**5**	**6**	**8**	Not found	543
SQLite3	**0 / 10**	**33m17s**	**1h02m52s**	**2h42m42s**	4705	**20**	**26**	**31**	Not found	226

- *Failed run*: fuzzer timed out (8 hours)
- 156/180 successful runs → **87%**

* Two variants of initial fuzzing seeds were used for Belkin: unspecialized (U) and specialized (S) ones. Variant U are the default AFL++ seeds for HTTP servers, with which the backdoor could never be triggered by AFL++ in 10 runs of 8 hours. Variant S are specialized seeds, targeting the URL parser of the server, with which the backdoor was triggered in 7 of the 10 AFL++ runs. The oracle could always recognize the backdoor, once AFL++ had triggered it.

D. Kokkonis, M. Marcozzi, E. Decoux, S. Zacchiroli

Results: detection time

Backdoor	ROSA — (10 runs × 8 hours) / backdoor — 1 minute of fuzzing for phase 1. Robustness + speed: *Failed runs*	*Time to first backdoor input* Min.	Avg.	Max.	Automation level: *Baseline* Avg. seeds	*Manually inspected inputs* Min.	Avg.	Max.	STRINGER: *Backdoor detection time*	*Manually inspected strings*
Authentic backdoors										
Belkin / httpd	10 / 10	Timeout	Timeout	Timeout	2773	2	4	6	Not found	**0**
+ *with specialized seeds**	**3 / 10**	**17m40s**	**3h49m29s**	Timeout	2781	4	5	7	Not found	**0**
D-Link / thttpd	**0 / 10**	**2m07s**	**15m00s**	**43m42s**	3648	**7**	**9**	**12**	Not found	113
Linksys / scfgmgr	**0 / 10**	**1m05s**	**1m29s**	**1m55s**	251	1	1	1	Not found	**0**
Tenda / goahead	**0 / 10**	**1m28s**	**3m34s**	**8m10s**	535	**1**	**2**	**2**	Not found	290
PHP	1 / 10	24m30s	2h03m44s	Timeout	11631	**4**	**8**	**16**	**6m**	573
ProFTPD	4 / 10	4m03s	3h37m32s	Timeout	[illegible]	[illegible]	[illegible]	[illegible]	**7s**	314
vsFTPd	**0 / 10**	**3m04s**	**5m41s**	**11m03s**	[illegible]	[illegible]	[illegible]	[illegible]	Not found	117
Synthetic backdoors										
sudo	**0 / 10**	**5m47s**	**8m05s**	**11m46s**	[illegible]	[illegible]	[illegible]	[illegible]	Not found	137
libpng	2 / 10	13m47s	2h24m46s	Timeout	[illegible]	[illegible]	[illegible]	[illegible]	**4s**	9
libsndfile	3 / 10	2h21m08s	5h04m46s	Timeout	[illegible]	[illegible]	[illegible]	[illegible]	**5s**	**8**
libtiff	**0 / 10**	**5m08s**	**12m15s**	**25m10s**	9566	**1**	**3**	**5**	Not found	31
libxml2	**0 / 10**	**8m17s**	**27m14s**	**1h09m06s**	12104	**9**	**14**	**20**	Not found	1208
Lua	**1 / 10**	**50m34s**	**4h07m41s**	Timeout	6653	**6**	**12**	**17**	Not found	36
OpenSSL / bignum	**0 / 10**	**9m53s**	**22m00s**	**39m52s**	1441	**1**	**1**	**2**	Not found	657
PHP / unserialize	**0 / 10**	**23m05s**	**1h04m39s**	**1h35m08s**	6285	**1**	**1**	**1**	Not found	974
Poppler	**0 / 10**	**11m28s**	**49m09s**	**1h33m02s**	9544	**5**	**6**	**8**	Not found	543
SQLite3	**0 / 10**	**33m17s**	**1h02m52s**	**2h42m42s**	4705	**20**	**26**	**31**	Not found	226

* Two variants of initial fuzzing seeds were used for Belkin: unspecialized (U) and specialized (S) ones. Variant U are the default AFL++ seeds for HTTP servers, with which the backdoor could never be triggered by AFL++ in 10 runs of 8 hours. Variant S are specialized seeds, targeting the URL parser of the server, with which the backdoor was triggered in 7 of the 10 AFL++ runs. The oracle could always recognize the backdoor, once AFL++ had triggered it.

- ROSA avg. detection time: **1h30m**
- Stringer: 4/17 backdoors detected → **24%**

Results: automation

Backdoor	ROSA — (10 runs × 8 hours) / backdoor — 1 minute of fuzzing for phase 1								STRINGER	
	Robustness + speed				Automation level					
	Failed runs	*Time to first backdoor input* Min.	Avg.	Max.	*Baseline* Avg. seeds	*Manually inspected inputs* Min.	Avg.	Max.	*Backdoor detection time*	*Manually inspected strings*
Authentic backdoors										
Belkin / httpd	10 / 10	Timeout	Timeout	Timeout	2773	2	4	6	Not found	**0**
+ *with specialized seeds**	**3 / 10**	**17m40s**	**3h49m29s**	Timeout	2781	4	5	7	Not found	**0**
D-Link / thttpd	**0 / 10**	**2m07s**	**15m00s**	**43m42s**	3648	**7**	**9**	**12**	Not found	113
Linksys / scfgmgr	**0 / 10**	**1m05s**	**1m29s**	**1m55s**	251	1	1	1	Not found	**0**
Tenda / goahead	**0 / 10**	**1m28s**	**3m34s**	**8m10s**	535	**1**	**2**	**2**	Not found	290
PHP	1 / 10	24m30s	2h03m44s	Timeout	11631	**4**	**8**	**16**	**6m**	573
ProFTPD	4 / 10	[illegible]	[illegible]	[illegible]	[illegible]	**5**	**8**	**11**	**7s**	314
vsFTPd	**0 / 10**	[illegible]	[illegible]	[illegible]	[illegible]	**3**	**4**	**4**	Not found	117
sudo	**0 / 10**	[illegible]	[illegible]	[illegible]	[illegible]	**1**	**1**	**1**	Not found	137
libpng	2 / 10	[illegible]	[illegible]	[illegible]	[illegible]	**1**	**2**	**2**	**4s**	9
libsndfile	3 / 10	[illegible]	[illegible]	[illegible]	[illegible]	9	12	13	**5s**	**8**
libtiff	**0 / 10**	**5m08s**	**12m15s**	**25m10s**	9566	**1**	**3**	**5**	Not found	31
libxml2	**0 / 10**	**8m17s**	**27m14s**	**1h09m06s**	12104	**9**	**14**	**20**	Not found	1208
Lua	**1 / 10**	**50m34s**	**4h07m41s**	Timeout	6653	**6**	**12**	**17**	Not found	36
OpenSSL / bignum	**0 / 10**	**9m53s**	**22m00s**	**39m52s**	1441	**1**	**1**	**2**	Not found	657
PHP / unserialize	**0 / 10**	**23m05s**	**1h04m39s**	**1h35m08s**	6285	**1**	**1**	**1**	Not found	974
Poppler	**0 / 10**	**11m28s**	**49m09s**	**1h33m02s**	9544	**5**	**6**	**8**	Not found	543
SQLite3	**0 / 10**	**33m17s**	**1h02m52s**	**2h42m42s**	4705	**20**	**26**	**31**	Not found	226

* Two variants of initial fuzzing seeds were used for Belkin: unspecialized (U) and specialized (S) ones. Variant U are the default AFL++ seeds for HTTP servers, with which the backdoor could never be triggered by AFL++ in 10 runs of 8 hours. Variant S are specialized seeds, targeting the URL parser of the server, with which the backdoor was triggered in 7 of the 10 AFL++ runs. The oracle could always recognize the backdoor, once AFL++ had triggered it.

- ROSA avg. inputs: **7** (semi-automated vetting)
- Stringer avg. inputs: **308** (x44) (manual reverse engineering)

Conclusion

Key takeaways

Contributions:

ROSA (1st **fuzzer-based** generic backdoor detector) + *ROSARUM* (first standardized backdoor benchmark)

github.com/binsec/rosa archived repository

github.com/binsec/rosarum archived repository

- **All** ROSARUM backdoors detected (**8h** fuzzing campaigns)
- Avg. detection time: **1 hour 30 minutes**
- Avg. manual effort: **7** suspicious runtime behaviors **to vet**
- **44 times** fewer false positives than Stringer
- **No reverse engineering** needed
- **No source code** needed

Dimitri Kokkonis
PhD student
CEA List, IP Paris

Mastodon: @plumtrie@mastodon.social
Twitter: @plumtrie
Homepage: kokkonisd.github.io
BINSEC team: binsec.github.io

We are hiring! secubic-ptcc.github.io/currentjobs

Dr. Michaël Marcozzi
marcozzi.net

Emilien Decoux

Pr. Stefano Zacchiroli
upsilon.cc/~zack/

AWARD WINNER BEST ARTIFACT

preprint 👇