NEWS · AUGUST 11, 2026

Mein Server wurde kompromittiert — die Wiederherstellung kostete einen Befehl

Ein Angreifer erlangte Codeausführung auf meinem Server und ließ stundenlang einen Kryptominer laufen — und die Wiederherstellung bestand darin, einen Container wegzuwerfen und neu zu bauen, weil die Architektur den Explosionsradius längst festgelegt hatte und die Backups bereits getestet waren.

By Dr. Holger Flick



Die Website war offline, also tat ich, was jeder tut: Terminal auf und nachgesehen. Was ich fand, war ein Prozess namens `kmod-cache-update`, was klingt, als würde der Kernel ihn selbst starten, und in Wahrheit XMRig ist — ein quelloffener Monero-Miner unter falschem Namen. Er war auf drei Threads festgelegt, er sprach mit einem Mining-Pool, und fünf getrennte Watchdog-Skripte standen bereit, ihn im Sekundentakt neu zu starten, falls ihn etwas beenden sollte.

Jemand hatte Codeausführung auf meinem Server. Lassen Sie mich genau sein, was das bedeutete und was nicht, denn der Abstand zwischen diesen beiden Dingen ist der ganze Sinn dieses Beitrags.

Es bedeutete, dass ein Miner CPU-Zeit verbrannte, die ich nicht genehmigt hatte, die Maschine in Speichernot trieb und meine Website drei bis vier Stunden lahmlegte. Es bedeutete **nicht**, dass der Angreifer den Host erreichte. Es bedeutete nicht, dass er die Datenbank erreichte, oder den Mailserver, oder irgendeinen der anderen Dienste auf dieser Maschine. Alles, was der Angreifer berühren konnte, lag in einem einzigen Container — und als ich diesen Container wegwarf, verschwand das Problem mit ihm.

Diese Grenze war weder Glück noch Geistesgegenwart am Tag selbst. Sie wurde Monate zuvor gezogen, an einem ruhigen Nachmittag, als nichts brannte. Und dass ich das Problem in Ruhe statt hektisch bearbeiten konnte, liegt an etwas noch Unglamouröserem: Ich hatte Backups — und ich hatte daraus schon einmal tatsächlich wiederhergestellt.

Die Tür, die ich offen gelassen habe

Der Weg hinein war meine Gitea-Instanz — Gitea ist ein schlanker, selbst gehosteter Git-Server, also das, was man betreibt, wenn man sein eigenes privates GitHub möchte. Sie war sauber veröffentlicht, unter einem echten Hostnamen hinter Caddy mit gültigem Zertifikat, also genau so, wie man einen Git-Server exponieren sollte.

Der Fehler war kleiner und konkreter. **Die Registrierung war offen, und ein frisch angelegtes Konto durfte Git-Hooks erstellen.**

Giteas Einstellung `DISABLE_GIT_HOOKS` steht standardmäßig auf `false`, Benutzer dürfen also eigene Skripte hinterlegen, die der Server ausführt, wenn ein Repository-Ereignis eintritt. Das ist eine echte Funktion — so verdrahtet man Deployments —, aber zusammen mit einem Registrierungsformular, das jeder ausfüllen kann, schreibt sich der Ablauf von selbst: anmelden, Repository anlegen, Hook hinterlegen, pushen, und der Hook läuft auf meiner Maschine. Vier Schritte, kein Exploit, kein gestohlenes Passwort. Jeder einzelne davon eine Funktion, die genau wie vorgesehen arbeitet.

Die beiden Einstellungen, die das schließen:

```
; app.ini
[service]
DISABLE_REGISTRATION = true

[security]
DISABLE_GIT_HOOKS = true
```

Dahinter steckt eine allgemeinere Lehre, die ich jedem Kunden mitgeben würde, und sie hat wenig mit Gitea zu tun. **Ein Konto sollte nicht entstehen — und erst recht nicht aktiv werden —, ohne dass irgendetwas überprüft, dass ein Mensch dahintersteht.** Gitea vergibt ein funktionsfähiges Konto auf eine Formularübermittlung hin, ohne irgendetwas dazwischen: keine E-Mail-Bestätigung, keine Freigabe, an keiner Stelle ein zweiter Faktor. Auf einem System, dessen ganzer Zweck das Ausführen von Code ist, ist ein unverifiziertes Konto eine Erlaubnis zur Codeausführung — und sollte so ernst genommen werden, wie man es nähme, jemandem eine Shell zu geben.

Meine Regel, und die verteidige ich: Wenn eine Anwendung Code auf Ihrer Infrastruktur ausführen kann, schließen Sie entweder die Selbstregistrierung vollständig und legen Konten selbst an, oder Sie koppeln die Aktivierung an eine echte Überprüfung. Und jedes Konto, das existiert, bekommt einen zweiten Faktor — TOTP oder einen WebAuthn-Passkey, nicht SMS, die NIST seit Jahren einschränkt, weil Rufnummern per SIM-Swapping umgeleitet werden können.

> **Zwei Maßnahmen, zwei verschiedene Aufgaben**
> Um ehrlich zu sein, was hier geholfen hätte: Zwei-Faktor-Authentifizierung allein hätte diesen Angriff *nicht* gestoppt, denn der Angreifer hat nie ein bestehendes Konto angefasst — er hat sich ein eigenes gemacht. Überprüfung bei der Registrierung schließt diese Tür; 2FA schützt die bereits existierenden Konten vor Diebstahl. Sie lösen verschiedene Probleme, und Sie wollen beides. Die beiden zu verwechseln führt zu einer gut verteidigten Anmeldeseite vor einem weit offenen Registrierungsformular.

Woraus ich schließe, dass das nicht persönlich war

Die Konfiguration des Miners kennzeichnete seinen Worker mit der ID meines Containers — Angreifer nutzen dieses Feld, um ihre Worker über tausende Opfer hinweg auseinanderzuhalten. Ich wurde also nicht ausgewählt, sondern aufgezählt. Die Sicherheitsfirma Wiz dokumentiert genau diese Kampagne als JINX-0132: automatisiertes Scannen nach im Internet exponierter DevOps-Infrastruktur, gefolgt von XM-

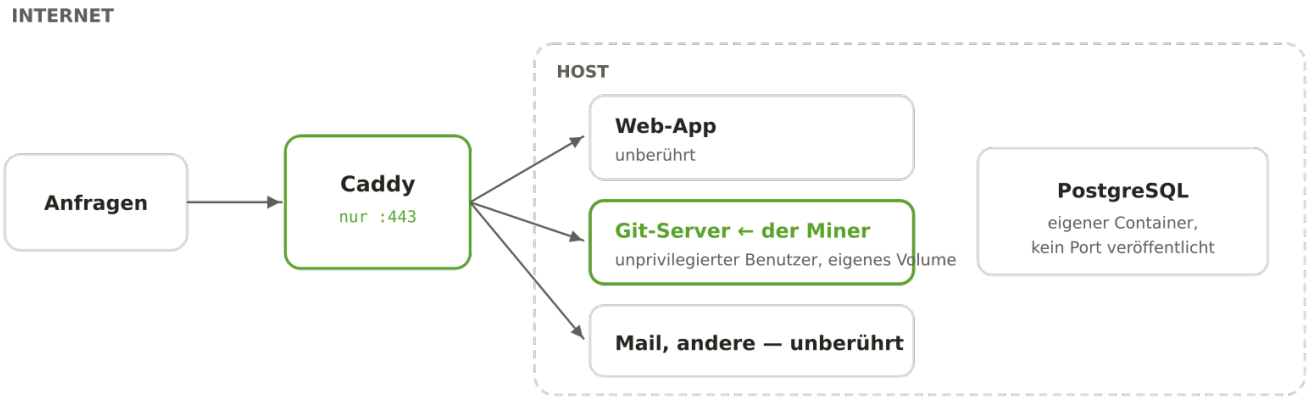
Rig, direkt aus dessen öffentlichen GitHub-Releases geladen. Ein Scanner fand ein Registrierungsformular und benutzte es, so wie bei allen anderen, die er findet.

Die Sandbox ist die eigentliche Geschichte

Jetzt der Teil, auf den es ankommt, denn er ist der Unterschied zwischen einem schlechten Nachmittag und einem schlechten Jahr.

Jeder Dienst auf dieser Maschine läuft in seinem eigenen Docker-Container, und von außen ist nichts erreichbar

außer über Caddy, einen Reverse Proxy, der TLS terminiert und Anfragen nach innen weiterreicht. Die Anwendungen brauchen nie ein eigenes öffentliches Gesicht.



Ein öffentlicher Eingang, jeder Dienst in seiner eigenen Box — der Miner landete in einer davon und blieb dort

Lesen Sie dieses Diagramm als eine Reihe von Mauern und dann den Vorfall dagegen. Der Code des Angreifers lief in der hervorgehobenen Box. Er lief als unprivilegierter Benutzer dieses Containers, nicht als root und nicht als ich. Er schrieb in das eigene Datenvolume dieses Containers. Er erschien in dessen eigener Control Group. Die Web-Anwendung in der Box darüber lieferte weiter Seiten aus, bis der Speicherdruck die ganze Maschine lahmlegte. Die Datenbank, in ihrem eigenen Container ganz ohne veröffentlichten Port, war von dort, wo der Angreifer stand, nie erreichbar.

Das ist gemeint, wenn von *Explosionsradius* die Rede ist, und die entscheidende Eigenschaft: **Sie legen ihn**

vorab fest. Nicht während des Vorfalls, wenn Sie müde sind und raten — sondern Monate vorher, wenn

Sie entscheiden, ob die Datenbank einen veröffentlichten Port bekommt und ob jeder Dienst seinen eigenen Container erhält. Diese Entscheidungen kosten zum Zeitpunkt nichts. Sie zahlen sich vollständig an einem Nachmittag aus, den Sie nicht geplant haben.

Genau das machte auch die Bereinigung trivial. Ich musste nichts desinfizieren und nicht über das Dateisystem hinweg suchen, was der Angreifer sonst noch angefasst hatte. Der Explosionsradius war ein Container, also war das Heilmittel ein Container: stoppen, Zustand wegwerfen, aus einer Definition neu bauen, der ich vertraue. Dieser letzte Punkt zählt — meine Compose-Dateien, meine Caddy-Konfiguration und meine Deployment-Skripte liegen sämtlich in Git, weshalb „aus etwas neu bauen, worauf der Angreifer nie Zugriff hatte" ein Befehl ist und kein Projekt.

> **Dafür braucht es nicht exakt meinen Stack**
>
> Podman bietet dieselbe Isolation mit rootlosen Containern als Voreinstellung, was sogar noch stärker ist. Traefik und nginx erledigen die Reverse-Proxy-Aufgabe ebenso gut, und Forgejo — ein community-geführter Fork von Gitea — ist ein ausgezeichneter Git-Server mit denselben prüfenswerten Einstellungen. Managed Hosting nimmt Ihnen einen Großteil der Isolation ab, und die verbleibende Frage lautet, wer sich auf dem registrieren darf, was Sie exponieren. Das Prinzip trägt überall: ein öffentlicher Eingang, geringstmögliche Rechte im Inneren, und eine Grenze, die Sie bewusst gewählt haben.

Warum ich nie nervös wurde

Ich möchte etwas beschreiben, das nicht technisch ist, denn es ist der Teil, den ich einem Kunden am ehesten über meine Arbeitsweise vermitteln möchte.

Als ich den Miner fand, empfand ich Ärger. Ich empfand keine Angst. Dieser Unterschied war weder Gelassenheit noch Erfahrung — er war Monate zuvor bezahlt worden, durch ein Backup-Regime, das ich bereits getestet hatte. Weil eine Wiederherstellung verfügbar und nachweislich funktionsfähig war, blieben mir alle Optionen offen:

- Ich konnte den Dienst sofort stoppen, ohne abzuwägen, was das Abschalten kosten würde.
- Ich konnte die Beweise sichern, bevor ich aufräumte, statt die Spuren in der Eile zu vernichten.
- Ich konnte es ablehnen, eine Produktionsdatenbank über eine Verbindung von einem Host mit einem Angreifer darauf zu bewegen, und einfach warten, bis das gefahrlos war.
- Ich konnte mich für Neubauen statt Reparieren entscheiden, weil ich wusste, woraus ich neu bauen würde.

Das ist es, was Backups tatsächlich kaufen, und es sind keine Dateien. Es sind **Entscheidungen**. Wer ohne Wiederherstellungsweg administriert, verhandelt am Ende im eigenen Kopf mit dem Angreifer — *vielleicht, wenn ich das schnell wegräume, vielleicht, wenn ich nicht zu genau hinsehe, vielleicht ist es in Ordnung* — und jeder dieser Drücke schiebt in Richtung der schlechtesten verfügbaren Wahl. Wer eine getestete Wiederherstellung hat, spürt nichts davon und trifft unmittelbar deshalb bessere Entscheidungen.

Die Disziplin dahinter ist die altbekannte, langweilige:

1. **Drei Kopien der Daten, auf zwei Medienarten, eine davon außer Haus.** Die 3-2-1-Regel hat so lange überlebt, weil sie immer wieder überlebt.
2. **Mindestens eine Kopie, die der laufende Server nicht erreichen kann.** Genau dieser Punkt schlägt Ransomware. Hält Ihr Server gültige Schreibrechte auf das Backup-Ziel, dann besitzt derjenige, der Ihren Server besitzt, auch Ihre Backups. Append-only-Repositories oder Pull-basierte Sicherungen lösen das.
3. **Eine Wiederherstellung, die Sie tatsächlich durchgeführt haben.** restic und BorgBackup prüfen die Integrität eines Repositorys auf Wunsch, und diese Prüfungen sollten Sie laufen lassen — aber Integrität ist nicht Wiederherstellbarkeit. Ein Backup, das Sie nie zurückgespielt haben, ist eine Hypothese.
4. **Konfiguration in der Versionsverwaltung, nicht nur Daten im Backup.** Datensicherungen holen Ihre Inhalte zurück. Konfiguration in Git holt die ganze Maschine zurück.

Punkt drei ist der, den alle überspringen, und er ist derjenige, der ein Backup von einem Gefühl in einen Vermögenswert verwandelt. Der Grund für meine Ruhe war nicht, dass Backups existierten. Es war, dass ich eines hatte funktionieren sehen.

Was das tatsächlich beweist

Eine Kompromittierung ist kein binäres Ereignis, bei dem man entweder in Ordnung oder ruiniert ist. Sie hat einen Radius, und diesen Radius entwerfen Sie.

- **Die Sandbox hat ihre Aufgabe erfüllt.** Jemand hatte stundenlang Codeausführung auf meiner Maschine und kam aus einem Container nicht heraus. Diese Eindämmung wurde lange vor dem Vorfall entschieden und machte aus einer möglichen Katastrophe ein paar Stunden Ausfall.
- **Lassen Sie keine Konten ohne Überprüfung entstehen.** Auf jedem System, das Code ausführt, ist ein aus einer unverifizierten Formularübermittlung entstandenes Konto eine Erlaubnis zur Codeausführung. Schließen Sie die Selbstregistrierung oder koppeln Sie die Aktivierung an eine echte Überprüfung — und geben Sie den verbleibenden Konten einen zweiten Faktor, denn die beiden Maßnahmen bewachen verschiedene Türen.
- **Backups kaufen Entscheidungen, keine Dateien.** Die getestete Wiederherstellung ist der Grund, warum ich überlegt statt verzweifelt handeln konnte. Diese Ruhe ist das eigentliche Ergebnis.
- **Mein Alarm war ein Ausfall, kein Sicherheitswerkzeug.** Das gehört zugegeben: Die Eindämmung begrenzte den Schaden, sagte mir aber nicht, dass etwas nicht stimmt. Das tat ein Leistungssymptom. Ressourcen-Alarme auf anhaltende CPU-Last sind die günstige Korrektur, denn ein leiserer Miner liefe noch immer.

> Entwerfen Sie den Explosionsradius, solange nichts kaputt ist, und testen Sie die Wiederherstellung, bevor Sie sie brauchen. Wer beides tut, für den wird ein Einbruch zur Unannehmlichkeit statt zum Notfall.

Wenn Sie eigene Infrastruktur betreiben und ein zweites Paar Augen darauf möchten, wo Ihr Explosionsradius derzeit endet, sprechen wir darüber.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —