

Old-School IRC, New Victims: Inside the Newly Discovered SSHStalker Linux Botnet

flare.io/learn/resources/blog/old-school-irc-new-victims-inside-the-newly-discovered-sshstalker-linux-botnet

Research Team

February 9, 2026



Flare’s research team has uncovered a previously undocumented Linux botnet operation we’re calling SSHStalker. To the best of our knowledge, no other research team has reported on this threat actor. Our SSH honeypot captured multiple attacks over two months, revealing a sophisticated operation that blends 2009-era Internet Relay Chat (IRC) botnet tactics with modern mass-compromise automation.

Key Findings: SSHStalker’s Attack Infrastructure and Tactics

- **IRC is the operational backbone:** SSHStalker relies on classic, “old-school” IRC botnet mechanics (multiple C-based bot variants + Perl IRC bot + Tsunami malware + Keiten malware + multi-server/channel redundancy), indicating the threat actor prioritizes resilient, low-cost command-and-control over modern C2 sophistication.
- **Automated mass-compromise pipeline:** The campaign chains an SSH scanner (Golang “nmap” lookalike) with rapid staging (GCC install, compile-and-run workflow) as well as scanners developed by others and hands-off enrollment into IRC channels – consistent with a botnet operator optimizing for scale and repeatability.
- **Persistence and recovery are very noisy but effective:** The kit uses low-effort persistence (cron every minute) with a watchdog “update” relaunch model, meaning defenders can disrupt it, but must do so comprehensively, or the bot returns within ~60 seconds.
- **The toolset blends stealth helpers with legacy-era Linux exploitation:** Alongside log cleaners (utmp/wtmp/lastlog tampering) and rootkit-class artifacts, the actor keeps a large back-catalog of Linux 2.6.x-era exploits (2009–2010 CVEs). These are low value against modern stacks, but remain effective against “forgotten” infrastructure and long-tail legacy environments.
- **Strong ecosystem overlap with Outlaw/Maxlas-style playbooks, but no hard attribution:** File structure, execution chaining, IRC enrollment, and persistence patterns resemble known Romanian-linked Linux botnet operations; however, no direct Outlaw/Dota identifiers (names, hashes, or canonical artifacts) were found, suggesting a derivative operator, copycat, or adjacent crew rather than confirmed Outlaw itself.

Who is SSHStalker? A New Linux Botnet Operation

SSHStalker looks, at first glance, like a sophisticated threat actor who targets Linux systems, while executing fileless malware, rootkits, log cleaners, and a wide array of kernel exploits. As our analysis deepened, however, the narrative shifted. What we actually found was a loud, stitched-together botnet kit that mixes old-school IRC control, compiling binaries on hosts, mass SSH compromise, and cron-based persistence. In other words scale-first operation that favors reliability over stealth. In this blog, we report about the SSHStalker operation, how it stages and compiles multiple C-based IRC bots (in addition to Python, Perl, and Shell), how it keeps coming back within a minute, and why its Outlaw/Maxlas-like fingerprints suggest a derivative ecosystem threat actor rather than a new campaign of one of those. We are also giving defenders clear pointers to detect and mitigate the attack.



Description: An ASCII art found in one of the files of the threat actor

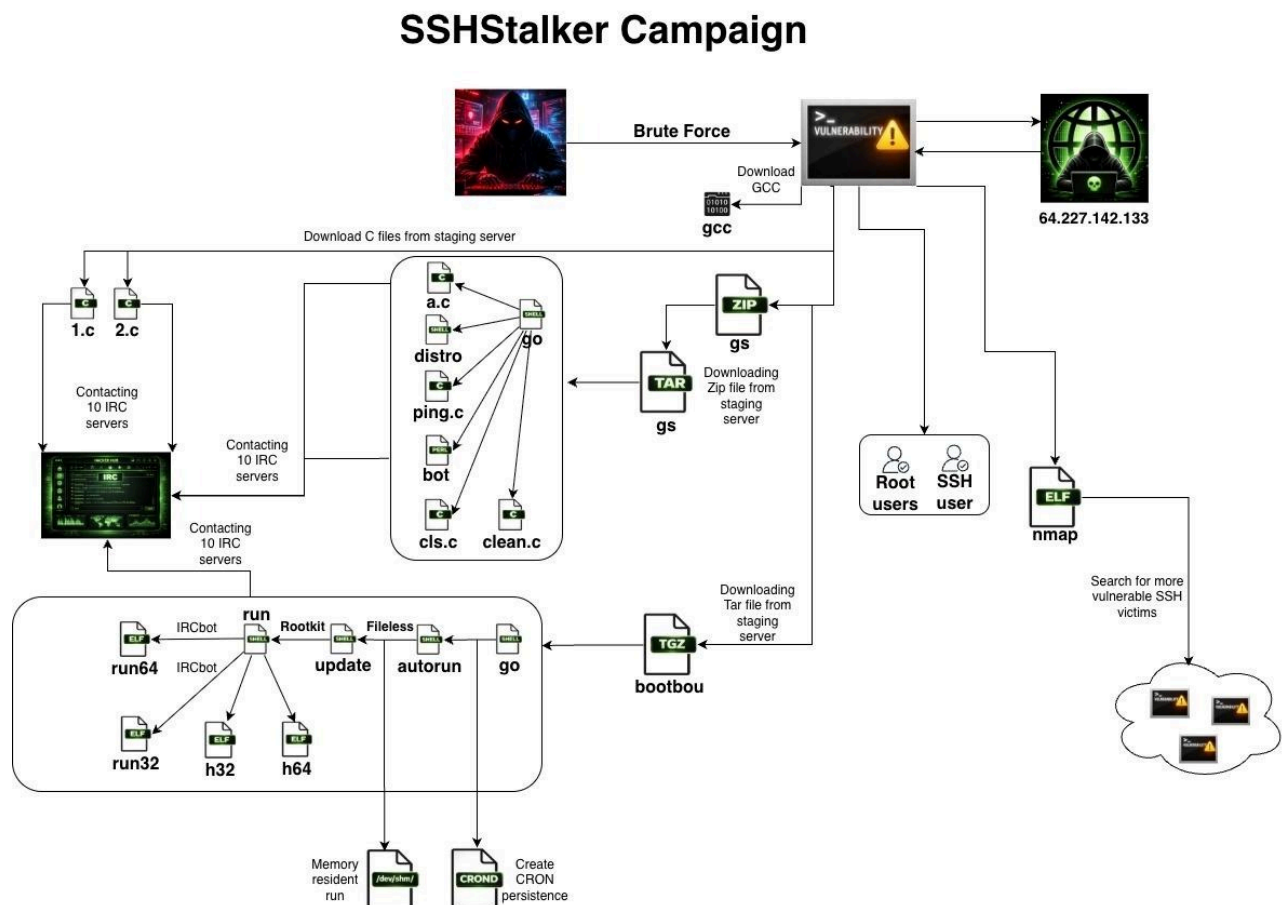
Technical Analysis: Attack Flow and Methodology

We deployed an SSH honeypot with intentionally weak credentials and monitored intrusion attempts since the beginning of 2026. While the majority of attacks originated from known threat actors already documented in public malware repositories and vendor reports, we identified a distinct cluster of intrusions with no prior media coverage or documented indicators of compromise.

To verify this was genuinely novel activity, we cross-referenced the attack patterns, infrastructure, and malware samples against existing threat intelligence databases, security vendor publications, and open-source malware collections. The toolset, execution flow, and command-and-control infrastructure showed no matches to previously reported campaigns.

We've designated this operation "SSHStalker" due to its distinctive behavior: the botnet maintained persistent access without executing any observable impact operations, despite having in its arsenal capabilities to launch DDoS attacks and conduct cryptomining. This "dormant persistence" pattern—infesting systems and establishing control without immediate monetization—differentiates it from typical opportunistic botnet operations and suggests either infrastructure staging, testing phases, or strategic access retention for future use.

The attack flow and our technical analysis follow below.



Description: SSHStalker's attack flow

Explaining the image above:

1. Dropping a binary named nmap, and renaming it to scan.
2. Download of GCC utility.
3. Dropping two C files 1.c and 2.c, compiling, and running them.

4. Dropping a compressed file (GS), that contains 7 files:
 - Four C files: a.c, clean.c, cls.c, and ping.c.
 - Two shell files: distro, and go.
 - One perl file: bot.
5. Dropping a compressed file (bootbou.tgz), that contains 8 files:
 - The binaries: h32, h64, run32, and run64.
 - The shell files: autorun, run, go, and update.

A worm capability:SSH Mass Scanning and Brute-Forcing

The binary is named nmap, but it is a Golang binary that scans for port 22 for servers with open SSH. It is a scanner that is a first part of a worm or a botnet to find new target servers from a newly infected server.

Stage 1: Deploying the IRC Botnet Framework (1.c, 2.c)

Next, two C files are dropped and executed. These are old school IRC bot malware, that are almost 100% identical, one is addressing gsm.ftp.sh and the room #auto and the other plm.ftp.sh and the room #xx. The logic behind it is to target a wide variety of hardware, as they also have small differences to run on ARM, X86, MISP etc.

For the script 1.c there were hard-coded C2 details: gsm.ftp.sh, channel #auto, key "gay".

For the script 2.c there were Hard-coded C2: plm.ftp.sh, channel #xx, key "gay".

Stage 2: Multi-Layer Payload Delivery (GS Archive)

The file GS is compressed with Zip and Tar, once inflated it contains 7 files.

The file go runs all the files one after the other.

Starting from the file a.c is yet another variant of the same IRC-controlled bot (1.c and 2.c), configured to connect to the gsm.ftp.sh network and join a specified keyed channel for command-and-control. Functionality and code layout strongly match the same toolkit. For the script a.c there were Hard-coded C2: gsm.ftp.sh, channel #xx, key "gay". It is compiled and executed, initiating C2 communication with the IRC channel.

Then the file distro is executed, which adjusts the execution based on the Linux distribution (CentOS, Ubuntu, and RedHat), to leave persistence execution. It drops/enables a service/init script named like network-man from a local stash (e.g., .reboot/...) and enables it via systemctl, chkconfig, or update-rc.d.

Next, the file ping.c is compiled and executed. It is set to elevate privileges. This script serves as a tiny privilege backdoor helper: calls setuid(0) / setgid(0) and execl("/bin/bash", ...) with intend to be compiled and installed as a setuid-root shell launcher (as go.sh does).

Now the Perl file bot is executed and it is also set to maintain IRC C2 communication. This is an IRC bot / DDoS bot that connects to an UnrealIRCd server, sets a fake process name (-bash), then joins a control channel and waits for commands (typical NICK/USER, JOIN, PRIVMSG flow). It includes attack routines (e.g., flood-style traffic) and general bot control logic, and is designed to be run long-term (option for startup persistence).

Lastly the files clean.c and cls.c are compiled and executed, they are responsible to clean and delete history and ssh connections logs. The script clean.c is a cross-platform "hide your tracks" utility: scans directories/files and removes evidence strings from logs, and can manipulate login/accounting artifacts (e.g., utmp/lastlog style areas). This is the kind of companion tool used after compromise to reduce forensic visibility.

The script cls.c is a dedicated login record cleaner targeting utmp/wtmp/lastlog-style records (Linux/*nix). It's designed to remove or rewrite entries associated with a username/session to hide interactive access.

Stage 3: Another Compressed File (bootbou)

This kit establishes persistence and “keep-alive” execution in a very classic, low-effort way: it records its current working directory (`pwd > dir`), then installs a cron job that runs every minute and silently executes an update script (`* * * * *`
`$dir/update >/dev/null 2>&1`).

That update script is essentially a watchdog: it checks for a `mech.pid` file, probes whether the PID inside still exists (using a signal check), and if not, it changes back into the kit’s directory and relaunched `./run` in the background with output suppressed. In other words, if a defender kills the main process, cron just brings it back within 60 seconds quietly and automatically.

These scripts run an IRC-botnet builder. It generates a `cfg` file with multiple IRC servers/ports (including UnderNET and several hard-coded IPs), then writes “ADMIN” tokens and randomized identity fields (IDENT/REALNAME/NICK) picked from a massive wordlist (Romanian slang, common names, pop culture, and generic commands).

This kind of randomized naming helps bots blend into noisy IRC channels and makes simple signature-based blocking harder. After creating the config, it runs `./autorun`, deletes a binary named `go`, and then selects an architecture-specific runner (`run32/run64`) optionally wrapped by a “hide” utility (masking as `crond`). Finally, it also shows a preference for operating out of memory-backed paths like `/dev/shm/...`, which reduces on-disk footprints and makes cleanup trickier.

Threat Intelligence: Infrastructure and Capabilities

On the staging server we were able to learn a lot about the operation and the threat actor. In addition to the above mentioned files it also contains many other files that provide further in depth about the threat actor’s targets, capabilities, and possible origin.

The threat actor blends mass compromise automation with a large back-catalog of Linux privilege-escalation exploits and rootkit-class tooling. In the toolset, we found 77 “IRC botnet files” and a smaller set of dedicated utilities that conduct ssh scanning that include initial access and IRC C2 enrollment and hands-off persistence.

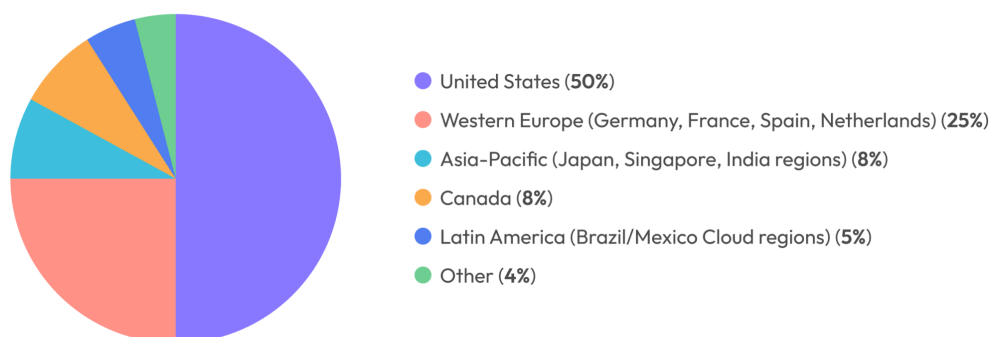
There’s also a heavy emphasis on running and managing the bot framework (e.g., an `autorun` component marked as an execution script) and operational helpers, which aligns with the actor doing scanning/banner work and then feeding results into their bot logic.

Fresh Scanning Results: 7,000 Compromised SSH Targets

We found a file that indicates almost 7,000 fresh results from an ssh scanner. These results were from January 2026 in a very close proximity to the attack against our honeypot.

The scan results are heavily dominated by cloud hosting providers, with strong indicators of Oracle Cloud infrastructure, which operates large ASN blocks such as AS31898 and related Oracle network ranges.

Geographically, these IPs are typically distributed across global cloud regions (US, EU, APAC). This pattern is consistent with opportunistic automation or disposable attack infrastructure rather than dedicated nation-state or boutique hosting operations.



Description: Geo-location distribution of the SSH IP addresses found in the results file

Legacy Exploit Arsenal: 16 CVEs Targeting Linux Kernel 2.6.x

Where the kit really stands out is the exploit inventory: 81 exploit-related artifacts covering 16 distinct CVEs (so this isn't a single exploit, but rather a collection of Linux kernel exploits). Several are explicitly labeled kernel privesc / kernel exploit in the dataset, including:

- CVE-2009-2692 (sock_sendpage NULL pointer deref) present as both linux-sendpage.c ("Linux kernel exploit") and simple.c ("Linux exploit").
- CVE-2009-2698 (udp_sendmsg) present as udp_sendmsg.c ("Linux kernel exploit").
- CVE-2010-3849 present as full-nelson.c, described as "Local root for Linux <= 2.6.37 (x86/x64)".
- CVE-2010-1173 present as sctp-boom.py, explicitly a remote DoS against Linux Kernel <= 2.6.33.3 (SCTP INIT crash).
- Additional CVEs appear inside the exploit module set (e.g., CVE-2009-2267, CVE-2009-2908, CVE-2009-3547, CVE-2010-2959, CVE-2010-3437, and others), with many tracked as "Linux exploit" modules even when the sheet doesn't include a full one-line description for each.

Kernel targeting in the dataset clusters around older enterprise-era Linux:

- Explicit kernel/version references and/or builds for 2.6.18, 2.6.18-164, 2.6.31, and 2.6.37.
- The kit includes exploit modules named for 2.6.31 (e.g., exp_2.6.31-2010.c, plus exp_ingom0wnar.c and exp_powerglove.c as "Linux exploit"),
- and a clearly scoped "works up to" boundary via full-nelson.c (<= 2.6.37).

Operationally, there are multiple helper scripts to compile/launch exploit modules at scale the kind of glue code you ship when you expect to land on varied, inconsistent systems and want the exploit path to be opportunistic.

Finally, the kit contains rootkit-labeled malware rather than only post-exploitation scripts. We found five distinct rootkit-class artifacts by name:

- brk and brk2 (both labeled Linux.Trojan.Rootkit.15)
- prt and ptrace (both labeled LINUX/Rootkit-S),
- and x (explicitly labeled rootkit).

There's also an additional bundle note marked "strutum crypto runner + rootkit" ("Various files"), suggesting the actor couples stealth/persistence (rootkit behavior) with monetization tooling (crypto runner) as part of the same deployment package.

Concerning the kernel exploits and rootkits used by the threat actors. These findings indicate a toolkit ecosystem built around 2009-2010 era Linux kernel vulnerabilities, primarily targeting the 2.6.x generation that dominated legacy enterprise servers and embedded appliances.

In today's environment (2026), their direct relevance is low for fully maintained infrastructure, but not zero:

- A realistic threat-intel estimate would place exposure at roughly 1–3% of internet-facing Linux servers.
- This is rising to 5–10% in long-tail environments (legacy hosting providers, abandoned VPS images, outdated appliances, industrial/OT gear, or niche embedded deployments).

The exploit kits themselves being open source, widely mirrored, and operationally simple lowers the skill barrier. Namely they are often used by low-to-mid tier actors, botnet operators, or opportunistic access brokers rather than top-tier APTs. However, the presence of a curated kit with multiple kernel targets, version-specific modules, and naming tied to working exploit chains still suggests moderate operational maturity: the actor likely understands kernel version fingerprinting, privilege escalation chaining, and mass exploitation workflows, even if they are not developing novel exploits.

Website Reconnaissance and AWS Credential Harvesting

We found a kit that is aimed to find exposed secrets on websites. It included a heavily obfuscated Python script that is aimed to generate IP addresses and execute a binary called http grabber. This binary appears to be a multi-threaded HTTP/HTTPS scanning and content-matching tool. The binary loads target paths (a list of over 34,000 file paths on websites) and search patterns (AWS keys), performs web requests using browser-like headers, and scans responses using regex or keyword matching before saving results. The presence of proxy support, modern TLS/HTTP protocol stacks, and DNS-over-HTTPS capabilities indicates it was designed to operate reliably at scale and potentially bypass basic network restrictions.

It runs a binary "website grabber" that gets a configuration that contains a list of over 33,000 website paths and a search goal that looks for AWS keys.

```
regex_first_key:AKIA regex:([A-Z0-9]{4})(AKIA|A3T|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{12,}
```

IRC Infrastructure: EnergyMech and Behavioral Camouflage

The recovered files strongly indicate the presence of an EnergyMech IRC bot deployment, a historically legitimate but frequently repurposed automation framework widely used in early IRC networks for channel management and automation.

EnergyMech provides a full command-and-control style interface through IRC itself, including user privilege hierarchies (partyline users, channel masters, bot masters, and owners), remote command execution, multi-bot linking, and automated channel enforcement features such as mass-kick protection, clone detection, and automatic re-opping. In malicious or attacker-controlled infrastructure these same features can be repurposed for covert coordination, persistence, and distributed bot orchestration.

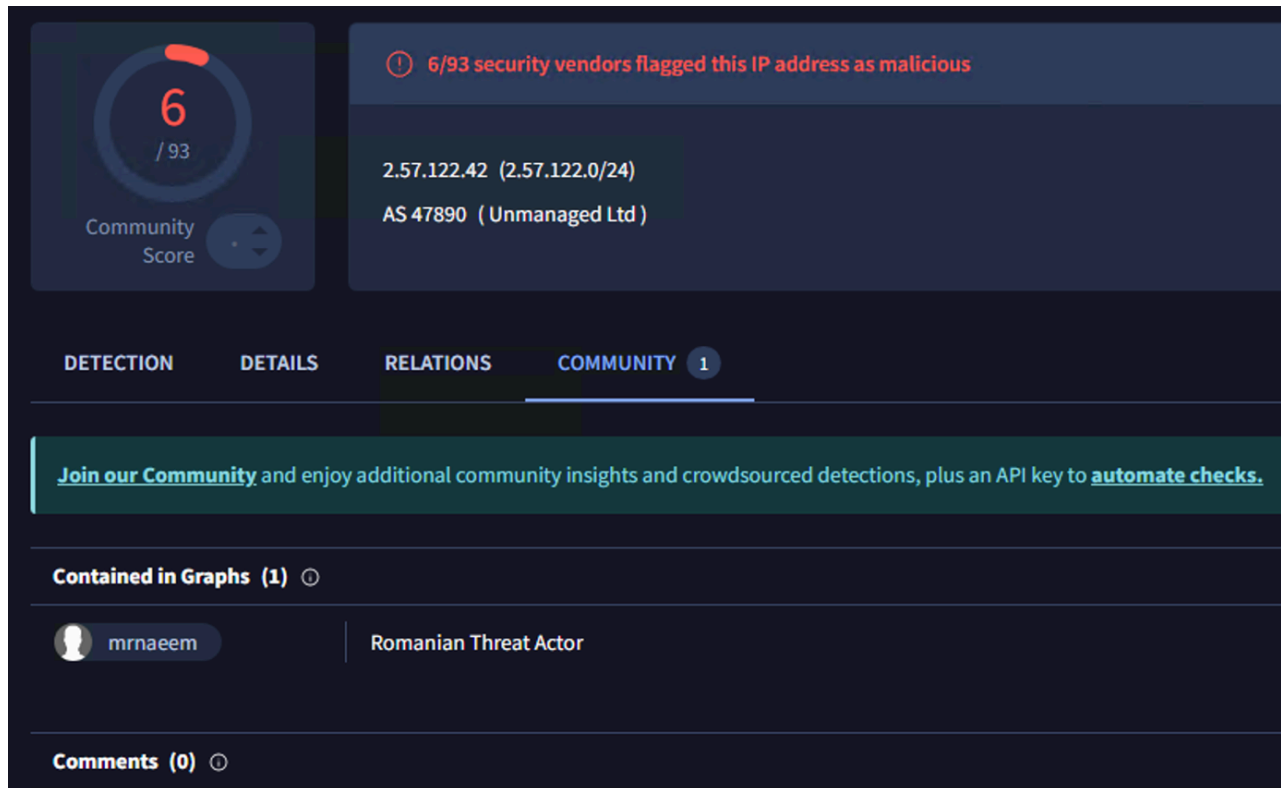
What makes this case particularly notable is the presence of companion text banks (insults, pickup lines, random chat phrases, nickname dictionaries), which map directly to built-in EnergyMech commands such as INSULT, PICKUP, SAY, and automated topic or message rotation. This strongly suggests the bot was configured not only for control, but also for behavioral camouflage. Namely, generating human-like noise in IRC channels to obscure real operator activity or make automated presence appear organic. This tactic is consistent with legacy botnet operational tradecraft where blending into public channels reduced suspicion while still allowing operators to issue commands via private messages, DCC sessions, or linked bot networks.

Cryptocurrency Mining Operations and Financial Motivation

We also found a few Cryptomining kits. One of them had this configuration:

POOL: 2.57.122.42:6960, WALLET: 0xCFCe0805aEF20326bA1A89bEC2447D6237D75D2.z1.0,

This indicates a cryptocurrency mining configuration pointing to a private pool and using an Ethereum-format wallet, most likely for mining Ethereum Classic (ETC) or a similar ETH-hash coin. In the screenshot below you can see the private IP address of the cryptomining pool has a comment that it is linked to a Romanian threat actor.

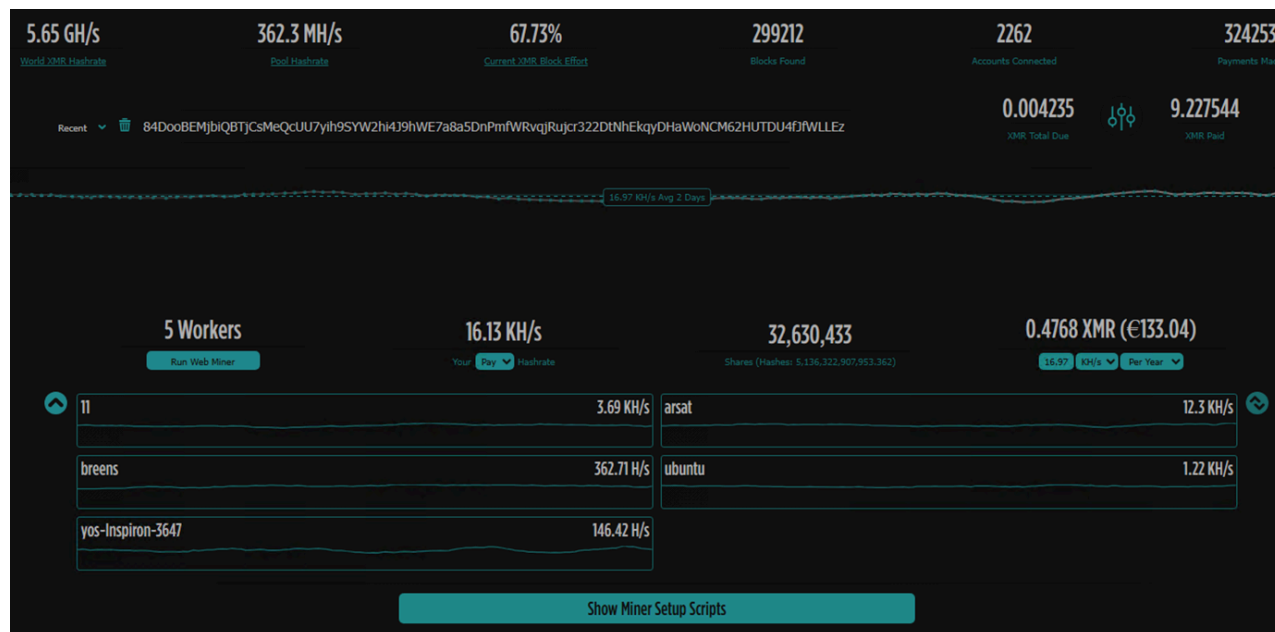


Virus Total marks the IP address as malicious with connection to a Romanian threat actor

We also found PhoenixMiner (MD5: ba8ecb3b2739e4846e939698d682f73f), which is a high-performance cryptocurrency mining program historically used for mining Ethereum and Ethereum-family coins, along with a rootkit – prochider (MD5: c644c04bce21dacdeb1e6c14c081e359).

We also found another Cryptomining kit with Linux.Cryptominer.Camelot which mine through a pool (auto.c3pool.org:19999) with this wallet:

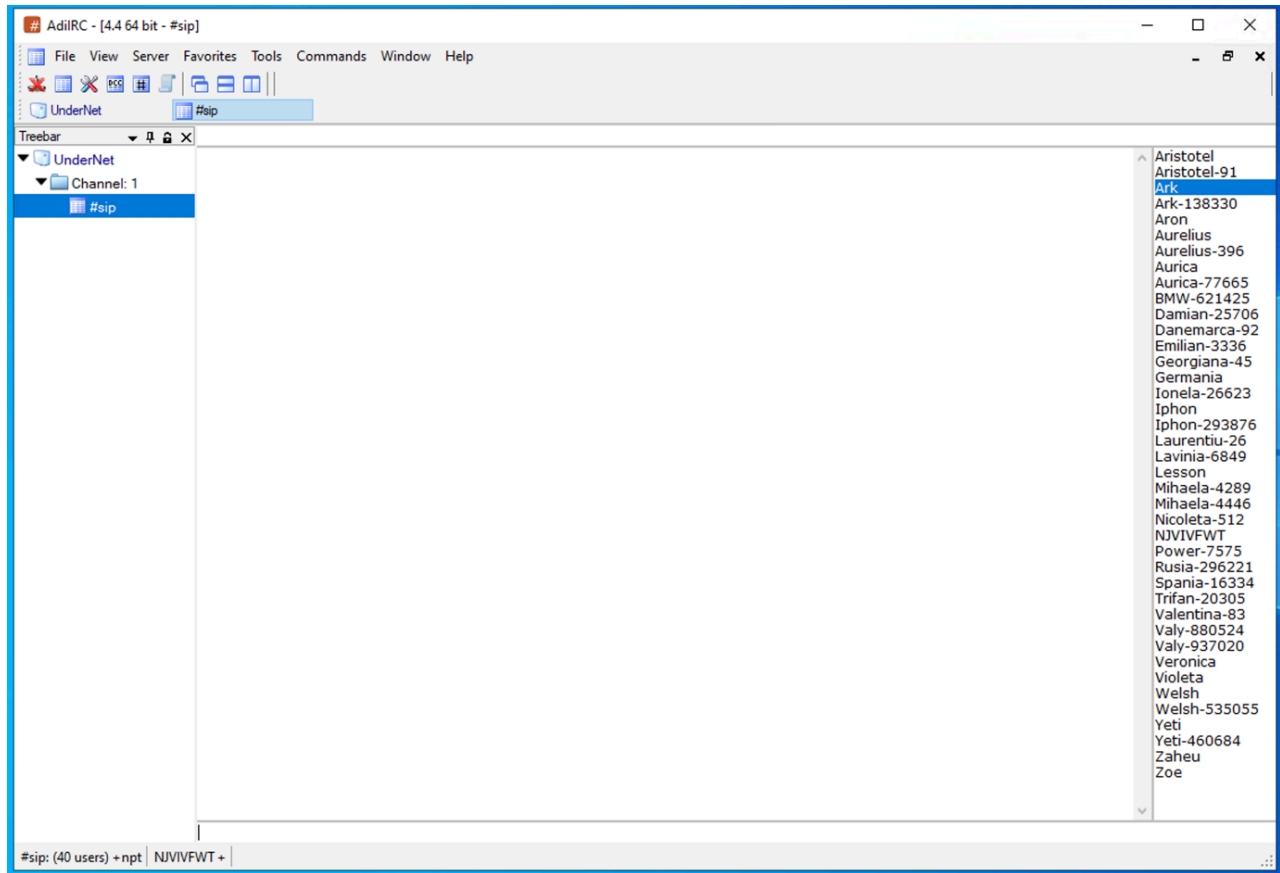
84DooBEMjbiQBTjCsMeQcUU7yih9SYW2hi4J9hWE7a8a5DnPmfWRvqjRujcr322DtNhEkqyDHaWoNCM62HUTDU4fJfWLLEz.



Crypto wallet activity

Inside the Command-and-Control: Observing IRC Channels

During our investigation, we were able to successfully authenticate to the IRC server associated with the infrastructure; however, we did not observe active command-and-control style tasking, bot commands, or operator chatter. The channel behavior appeared limited to users connecting and disconnecting, with no visible operational coordination at the time of observation. Notably, the server and room structure were hosted on what appears to be a legitimate, public IRC network, and the environment itself looked authentic and maintained – suggesting either dormant infrastructure, staging infrastructure, or the use of real IRC ecosystems to blend [malicious operations](#) into normal platform traffic.



A screenshot from the “infected machines IRC channel”

SSHStalker’s Operational Fingerprint and Romanian Connections

SSHStalker does not appear to focus on novel exploit development but instead demonstrates operational control through mature implementation and orchestration, by primarily using C for core bot and low-level components, shell for orchestration and persistence, and limited Python and Perl usage mainly for utility or supporting automation tasks inside the attack chain and running the IRCbot.

Additional threat intelligence collected from staging infrastructure indicates the threat actor maintains an extensive malware and script repository combining both open-source offensive tooling and previously published malware samples. The dataset includes scripts and artifacts written or commented in Russian, Chinese, German, and French, but the volume is sparse and limited and these were consistently traceable to already known public malware families or open repositories.

In contrast, Romanian-language artifacts and comments were not observed elsewhere, suggesting either original development or localized modification layers. This linguistic signal is further reinforced by Romanian-style nicknames, slang patterns, and naming conventions inside IRC channels and configuration wordlists, which together provide a strong (though not definitive) geographic attribution indicator.

From an operational perspective, SSHStalker shows strong similarities to historically documented Linux botnet operations such as Outlaw-style or Maxlas-style activity:

- Heavy SSH brute-force exposure,
- automated post-compromise staging,
- multi-layer persistence,
- and IRC-based command coordination.

Structurally, the toolkit shows clear lineage overlap with Outlaw-like operational flow and files structure:

- Multi-stage SSH compromise, staging, and bot enrollment.
- Large exploit back-catalog targeting legacy Linux kernels.
- Cron-based persistence and watchdog execution models.
- IRC multi-server redundancy and channel failover.
- Noise-generation and identity randomization for channel camouflage.

File hierarchy, naming conventions, and execution chaining logic strongly resemble legacy botnet kits where operators aggregate multiple public kits and glue them together with orchestration scripts. The presence of duplicated IRC bot variants, multiple packaging layers (ZIP → TAR → payload bundles), and redundant persistence mechanisms suggests an actor optimizing for reliability and spread, rather than stealth or precision targeting.

Having said that, none of the files we found had any link or mention to outlaw (a.k.a Dota or Dota3) any of its indications of compromise, files or hash values. We took a long length to find a distinct connection or link to Outlaw and failed, thus we decided that this could either be a copycat, a follower, or mere coincidence and named the threat actor SSHStalker.

Overall, SSHStalker appears to sit in the mid-tier operator category. The threat actor is not developing zero-days or novel rootkits, but demonstrating strong operational discipline in mass compromise workflows, infrastructure recycling, and long-tail persistence across heterogeneous Linux environments. The combination of recycled global tooling with localized Romanian-language artifacts and IRC ecosystem behavioral patterns provides one of the strongest clustering signals observed across the campaign.

Detection and Mitigation

Detection Strategies: Monitoring for SSHStalker Activity

- There were many binaries that had virus total detections, you can use an AV to scan for malicious files in your environments.
- Monitor for gcc, make, or build tool execution on production servers.
- Alert when compilers run from user directories, /tmp, or /dev/shm.
- Investigate package installs of development toolchains outside normal build pipelines.
- Alert when newly compiled binaries execute within seconds or minutes of creation.
- Monitor for short-lived source files followed by new executable artifacts.
- Monitor outbound TCP connections that remain open for long periods.
- Detect IRC handshake behavior (client registration, channel join patterns).
- Alert on servers communicating with unknown external chat or relay infrastructure.
- Alert on cron jobs executing every minute.
- Alert on creation of new root/privileged users and SSH keys.
- Monitor for cron entries created outside configuration management tools.
- Investigate cron jobs executing scripts from unusual directories.
- Monitor or block execution from /dev/shm and tmpfs locations.
- Alert on hidden directories or binaries executing from memory-backed storage.
- Monitor suspicious access to utmp, wtmp, and lastlog files.
- Alert when non-system processes attempt to modify login accounting records.

Mitigation Recommendations: Hardening SSH and Removing Attack Vectors

- Disable SSH password authentication.
- Enforce SSH key-based authentication only.
- Implement SSH brute-force rate limiting and monitoring.
- Limit access to SSH outside your network.
- Remove compilers from production images if possible.
- Allow toolchain execution only in controlled build environments.
- Enforce egress filtering based on business need.

- Block unnecessary outbound TCP connections by default.
- Disable execution from /dev/shm if possible.
- Monitor tmpfs execution events in EDR or audit tooling.
- Monitor cron for new or modified entries.
- Alert on high-frequency execution persistence patterns.

Indicators of Compromise

#	Type	Value	Hash	Comments
1	elf file	h32	MD5: 0d01bd11d1d3e7676613aacb109de55f	Prochider rootkit
2	elf file	h64	MD5: 1e288bb6920d9cf07d0e5dbc8614469d	Prochider rootkit
3	elf file	run32	MD5: 32ee52b2918e06e3925eaccb0bea2d66	IRCbot
4	elf file	run64	MD5: 88a31724d376ba7ac8ce5c10f97da83d	IRCbot
5	elf file	nmap	MD5: f8f76d8772f07b716913ba85f3af8380	No detections was scanned under cr4myx.exe and classified as payload
6	shell file	autorun	MD5: 5b9d4ff6a89da88dcf1d7d04b6d1e976	Creates cron persistence and runs "run"
7	shell file	go	MD5: 4c3d248b1fc8d4963ebdded4aecfcb8e	Win.Trojan.Tsunami-5
8	shell file	run	MD5: 70677ce8be9ebc5f81c299f753b98d66	Runs the rootkits
9	shell file	update	MD5: 26ad93d703a565a2642c422b2434fc78	
10	Tar file	bootbou.tgz	MD5 98f1ac9c9baf2562eb00b7d4f89dc0dc	Contains malware and scripts
11	IP address	64.227.142.133	—	Marked malicious no specific detections
12	C file	1.c	MD5: 6ca73134ee02fb373ebaf9321b9840c8	Win.Trojan.Tsunami-5
13	C file	2.c	MD5: a24cabef282713b6c0e3f9c3efdabd91	Win.Trojan.Tsunami-5
14	zip file	gs.zip	MD5: 3b64c2ecd7ea152f9d4af9d0461db265	Contains malware and scripts
15	tar file	gs.tar	MD5: fb2ddb699bed59ff420b43f5640e7e0c	Contains malware and scripts
16	C file	a.c	MD5: a8d19e08aa022bacd8a76777874fad8a	Win.Trojan.Tsunami-5
17	perl file	bot	MD5: 1d4c9039ca7e0b3e93c708f5d02f92a0	Win.Trojan.Tsunami-5
18	C file	clean.c	MD5: 077cdcbe6c1bf4a0f4bc81feaf283be3	Clean access records and logs
19	C file	cls.c	MD5: 2b31ba929f3e4f8e8c84b3815b0e4909	Cleans login logs
20	shell file	distro	MD5: 94e513b01f26399ae16ac91b50fde268	

21	shell file	go	MD5: fd55f0754084ba041539bb469f06a83d	
22	C file	ping.c	MD5: 4777d24c864c04a6bfabb836811edf2d	Privileged backdoor
23	IP address	185.243.218.59	—	IRC channels #xx and #SIP
24	IP address	154.35.175.201	—	IRC channels #xx and #SIP
25	IP address	94.125.182.255	—	IRC channels #xx and #SIP
26	IP address	23.228.66.219	—	IRC channels #xx and #SIP
27	IP address	199.71.214.87	—	IRC channels #xx and #SIP
28	IP address	172.83.156.122	—	IRC channels #xx and #SIP
29	Domain	irc.undernet.org	—	IRC channels #xx and #SIP
30	Domain	plm.ftp.sh	—	IRC channels #xx and #SIP
31	Domain	gsm.ftp.sh	—	IRC channels #xx and #SIP



