ODIN

TLP:CLEAR

ODIN RESEARCH · MONOGRAPH 14 / 30 · RECON WORKBENCHES

VShell Staging and Recon Workbenches

Hospital Mapping, Relay Kits and Scanner Fleets

How two unlinked attacker servers and 2,049 scanner and suspected VShell endpoints cover reconnaissance, portal testing and covert access

Two-week observation window ending 18 November 2026 · Prepared November 2026
Full Report · Public edition · Version 1.1

719
Files across 25 extensions on workbench 189.24.105[.]131:8899

12
File paths on Workbench 2 (port 8088), with 16 secret and command-and-control matches

769
VShell endpoints on 685 IPs: 0 confirmed, 282 suspected, 487 candidates

425
ARL (Asset Reconnaissance Lighthouse) endpoints on 413 IPs, led by port 5003

What Is in This Report. 11 chapters and two appendices.

How two unlinked attacker servers and 2,049 scanner and suspected VShell endpoints cover reconnaissance, portal testing and covert access

01	Executive Summary	04
02	How the Pieces Fit: A Four-Role Model	07
03	The Toolchain and Where Defenders See It	10
04	Workbench 2: Portal Bypass and a Split Global Socket Kit	13
05	Workbench 1: Hospital Recon and VShell Staging	16
06	Scanner Fleets and the Dual-Use Problem	20
07	What Defenders Should Do	23
08	Detection Rules	25
09	Indicators	28
10	Scope and Limits	31
11	Sources	33
A	Indicators of compromise (4)	35
B	Detection rules (4 files)	36

IN SHORT

In a two-week observation window ending 18 November 2026, Odin by Brandefense studied two unlinked attacker workbenches and 2,049 scanner and suspected VShell endpoints. One server held 719 files: hospital portal maps, internal console login tests and VShell implant stagers. The other listed 12 file paths for portal bypass, password-hash edits and split Global Socket relays. No VShell endpoint is confirmed. Block the first workbench, hunt for the second one's file markers, keep dual-use scanners on watchlists, and alert on Global Socket relay lookups and VShell WebSocket traffic.

01

CHAPTER 01

Executive Summary

IN SHORT

In a two-week observation window ending 18 November 2026, Odin by Brandefense studied two unlinked attacker workbenches and 2,049 scanner and suspected VShell endpoints. One server held 719 files: hospital portal maps, internal console login tests and VShell implant stagers. The other listed 12 file paths for portal bypass, password-hash edits and split Global Socket relays. No VShell endpoint is confirmed. Block the first workbench, hunt for the second one's file markers, keep dual-use scanners on watchlists, and alert on Global Socket relay lookups and VShell WebSocket traffic.

INSIDE THIS CHAPTER

Three Decisions for Leaders 04

The Study in Numbers 05

01 Executive Summary

This report looks at the tools that sit between a first scan and lasting access. It covers internet-wide asset scanners, web portal attack scripts, and two covert access tools: VShell, a WebSocket-based remote access implant, and Global Socket (`gsocket`), a relay that tunnels shells through public servers. Nothing in the data links the two workbenches or the scanner fleets to one actor.

1. **A 719-file workbench mixes hospital portal mapping, console login tests and VShell stagers.**
   The server 189.24.105[.]131 exposed an open directory on port 8899. It held port scans, cached maps of hospital web portals and scripts scraped from a hospital information system. Its own access log records two outside hosts downloading a Windows VShell agent.

2. **File names on a second workbench point to portal bypass and relay kits.**
   On a second server (port 8088, address withheld), 12 file paths and 16 secret and command-and-control matches suggest tools for bypassing content management system (CMS) logins and editing password hashes. Other paths point to a Global Socket binary split into chunks. Script bodies were not available, so these capabilities are inferred.

3. **No VShell endpoint is confirmed, and most scanner consoles are dual-use.**
   We count 769 VShell endpoints: 282 suspected and 487 candidates. Alongside them sit 425 ARL, 422 Nessus, 397 Acunetix and 36 reNgine consoles. Legitimate security teams use these scanners too, so they belong on watchlists, not block lists.

Three Decisions for Leaders

1. **Watch for covert relay and WebSocket channels.** Alert on outbound WebSocket upgrades to ports 9999, 8084 and 9097, and on DNS lookups for `*.gs.thc[.]org`. Tie each alert to the process that made it.
2. **Protect portals against hash edits and split-binary uploads.** Audit CMS user tables for unexpected password-hash changes. Remove web file-manager plugins, and alert when a web server process starts shell scripts or `mount` over `/proc`.
3. **Separate active scanning from exposed scanner consoles.** Rate-limit inbound ARL, reNgine and Acunetix bursts at the perimeter. Put exposed scanner consoles on watchlists instead of block lists.

The Study in Numbers

Number	What it is
719	Files across 25 extensions on workbench 189.24.105[.]131:8899
12	File paths on Workbench 2 (port 8088), with 16 secret and command-and-control matches
769	VShell endpoints on 685 IPs: 0 confirmed, 282 suspected, 487 candidates
425	ARL (Asset Reconnaissance Lighthouse) endpoints on 413 IPs, led by port 5003
819	Exposed Acunetix (397) and Nessus (422) scanner consoles, owners unknown
2,049	Scanner and VShell endpoints in all, on 1,882 IPv4 addresses
4	Published indicators: 1 workbench host to block and 3 SHA-256 hashes

How to use this report.

Security leaders can read Chapter 1, Chapter 6 and Chapter 7. Healthcare IT and portal administrators should read Chapters 4 and 5. SOC and detection engineers should start with Chapters 3, 8 and 9.

02

CHAPTER 02

How the Pieces Fit: A Four-Role Model

IN SHORT

The two workbenches and the scanner fleets are separate observations. We group them into four roles, from surface mapping to covert access, to show where defenders can see each step. This grouping is an analytical model, not a known operation.

INSIDE THIS CHAPTER

What we studied 08

How the stages could connect 08

02 How the Pieces Fit: A Four-Role Model

No shared operator identifier, target domain or network handoff links 189.24.105[.]131, Workbench 2 and the scanner fleets. We place their tools side by side because each covers a different stage of an intrusion. Nothing here shows that any target was breached.

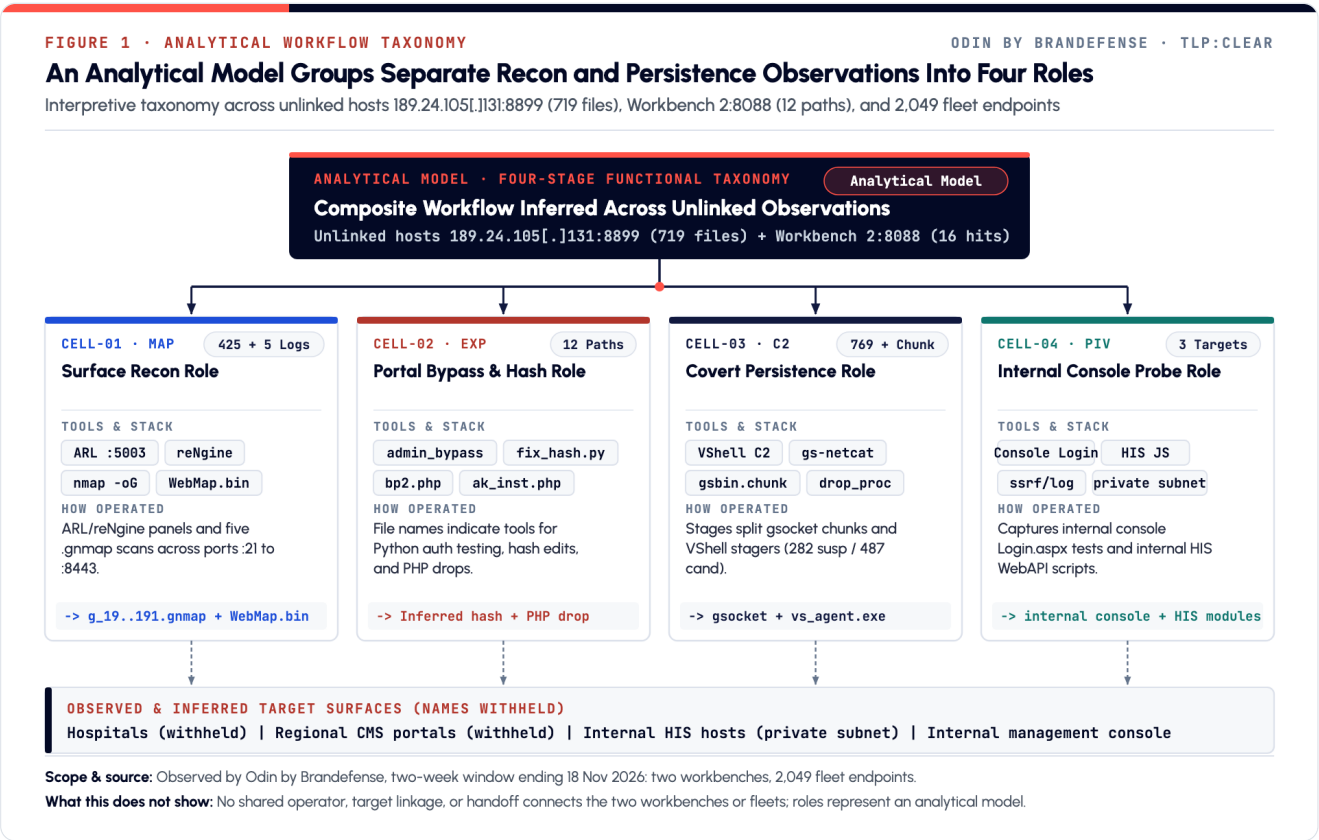


Figure 1: An analytical model groups separate recon and persistence observations into four roles

The four roles in the model are listed below.

Role	What it does	Main evidence
Surface recon	Finds subdomains, open ports and login pages	Five `nmap -oG` scan files and hospital web-map caches on port 8899; ARL panels on 5003 and reNgine panels on 8082
Portal bypass and hash edits	Tests CMS logins, edits password hashes, drops PHP files	12 file paths and 16 matches on port 8088
Covert persistence	Keeps access through relays and implants	Split Global Socket paths on port 8088; VShell agent downloads on port 8899; 769 suspected or candidate VShell endpoints
Internal console probing	Tests logins on internal management consoles	Login test captures and scraped hospital system scripts from a private 10.x subnet, stored on port 8899

What we studied

Cluster	Scope	Endpoints or files	Status	Role
Workbench 1: hospital recon and VShell	189.24.105[.]131 port 8899, plus suspected 8082 and 8086 and candidate 8091 and 8093	719 files	Block, strong evidence	Port scans, hospital web maps, internal console tests, VShell stagers
Workbench 2: portal and Global Socket kit	Port 8088, address withheld	12 paths	Hunt file markers	CMS bypass, hash edits, PHP drops, Global Socket chunks
VShell fleet	685 IPs	769	Hunt and watch	WebSocket listeners on 9999 (227), 8084 (71), 9097 (68)
ARL fleet	413 IPs	425	Watchlist	Subdomain, port and web fingerprint discovery, mostly on 5003 (282)
Nessus fleet	394 IPs	422	Watchlist	Vulnerability assessment consoles, mostly on 8834 (316)
Acunetix fleet	363 IPs	397	Watchlist	Web vulnerability scanners on 3443 (205), 7880 (58), 13443 (27)
reNgine fleet	27 IPs	36	Watchlist	Container recon pipelines (`nuclei`, `httpx`, `subfinder`), mostly on 8082 (18)

How the stages could connect

- **Surface mapping.** On port 8899, five greppable Nmap scans list hospital portals, VPN gateways on 1723 and admin web ports such as 8080 and 8443. ARL and reNgine automate the same kind of discovery.
- **Portal testing.** On port 8088, file names point to Python login-bypass and hash-edit scripts (`fix_hash`). They sit next to PHP uploaders (`bp2.php`, `ak_inst.php`).
- **Persistent access.** Port 8088 also lists a chunked Global Socket binary (`gsbin.chunk.ac`). On port 8899, the access log records downloads of `vs_agent.exe`, and Linux VShell stagers sit in the same directory.

03

CHAPTER 03

The Toolchain and Where Defenders See It

IN SHORT

Five tool families span asset discovery, portal script staging and covert relay delivery. Each leaves file names or network markers that defenders can hunt.

INSIDE THIS CHAPTER

The five tool families 10

What each stage leaves behind 11

03 The Toolchain and Where Defenders See It

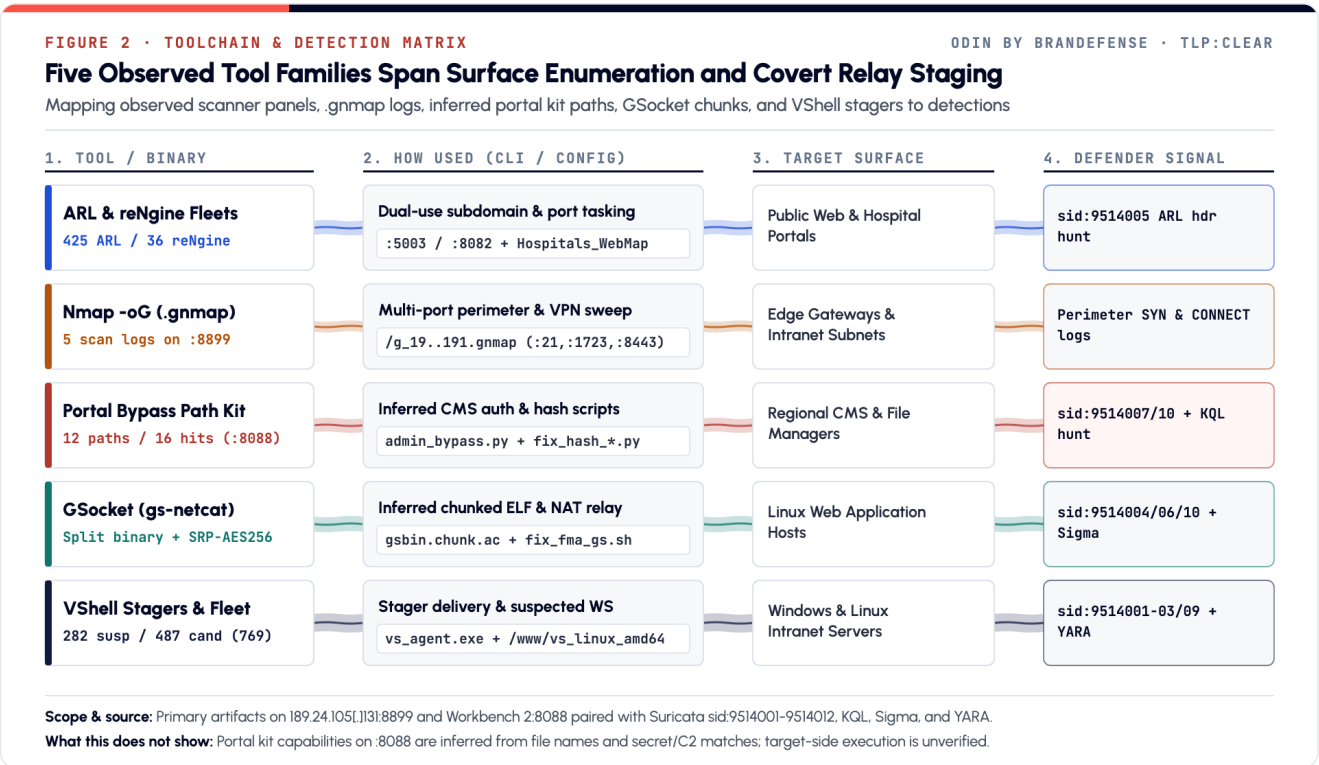


Figure 2: Five observed tool families span surface enumeration and covert relay staging

The five tool families

Tool family	How it is used	Where it shows up	Defender signal
ARL and reNgine	Dual-use subdomain and port discovery	Panels on 5003 and 8082	ARL HTTP header markers, inbound scan bursts
Nmap greppable output	Multi-port sweeps of the perimeter and VPN gateways	Five `.gnmap` files on port 8899	Perimeter SYN and CONNECT logs
Portal bypass kit	Inferred CMS login and hash scripts	`admin_bypass.py`, `fix_hash_*.py` on port 8088	Script-name fetches, web workers starting shells
Global Socket (`gs-netcat`)	Inferred chunked binary and NAT-traversal relay	`gsbin.chunk.ac`, `fix_fma_gs.sh` on port 8088	`*.gs.thc[.]org` DNS lookups, chunk writes on disk
VShell	Stager delivery and suspected WebSocket listeners	`vs_agent.exe`, `/www/vs_linux_amd64` on port 8899	Stager downloads, WebSocket upgrades on high ports

ARL is an open-source asset discovery platform. Its console listens on HTTPS port 5003 by default. reNgine is an open-source recon framework that chains tools such as `subfinder`, `httpx` and `nuclei` in containers.

What each stage leaves behind

Stage	Artifacts on the workbenches	How it works	Why defenders miss it
Surface mapping	`/g_19.gnmap`, `/g_179.gnmap`, `/g_184.gnmap`, `/g_187.gnmap`, `/g_191.gnmap`	Greppable sweeps across FTP, SSH, HTTP and HTTPS, PPTP (1723) and alt-HTTPS (8443)	Firewalls log routine SYN noise without linking it to later login probes
Sector web caching	`/pg_Hospitals_Main_WebMap.bin`, `/pg_Interactions_HospitalMails.bin`, `/logon.html`	Stored copies of hospital login pages and mail endpoints (112 to 132 KB caches)	Page scraping looks like normal browser GET traffic
Portal login and hash scripts	Portal-named `_admin_bypass.py` and `fix_hash_*.py`	File names suggest admin bypass and CMS password-hash edits	Endpoint tools rarely watch SQL updates in CMS user tables
Split-binary relay kit	`/gsbin.chunk.ac`, `/fix_fma_gs.sh`, `/drop_proc.sh`	File names suggest rebuilding a chunked `gs-netcat` binary and hiding processes under `/proc`	Native `gs-netcat` resolves `*.gs.thc[.]org` and sends no TLS SNI on port 443
WebSocket stager delivery	`/http8899.log`, `/vs_agent.exe`, `/www/vs_linux_amd64`, `/st_5_linux_amd64`	Python `http.server` hands out a Windows agent and 9,800-byte Linux ELF stagers	A long-lived WebSocket tunnel looks like normal web app streaming

CHAPTER 04

Workbench 2: Portal Bypass and a Split Global Socket Kit

IN SHORT

Workbench 2 lists 12 file paths on port 8088 and carries 16 secret and command-and-control matches. The names point to CMS login tests, password-hash edits and a Global Socket relay split into chunks.

INSIDE THIS CHAPTER

What the 16 matches are 13

The 12 paths in three groups 14

Why split the relay binary 14

04 Workbench 2: Portal Bypass and a Split Global Socket Kit

All portal names on this server are withheld. Script bodies were not available, so everything in this chapter rests on file paths and the secrets and endpoints found in them. We did not see any script run against a target, and we did not see any database change.

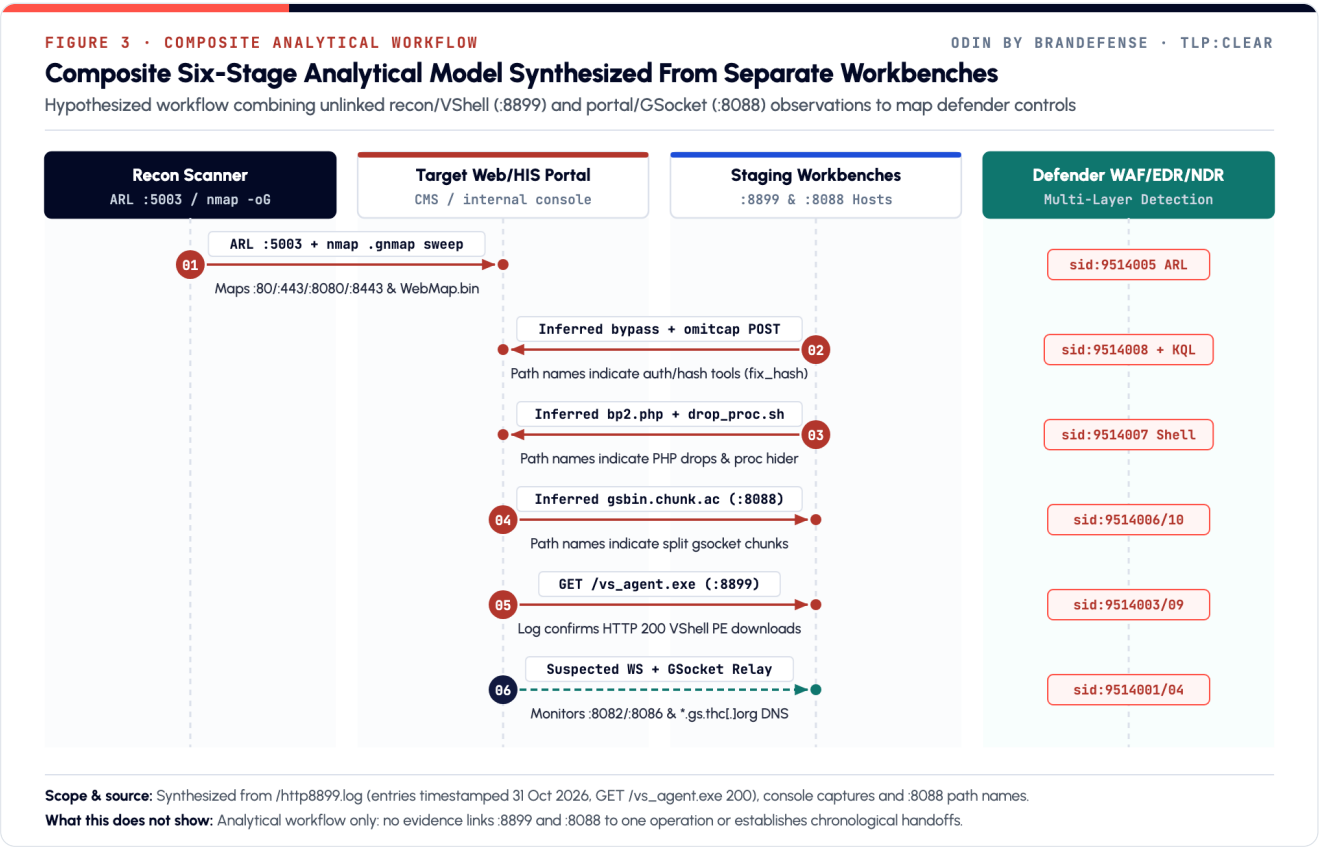


Figure 3: Composite six-stage analytical model synthesized from separate workbenches

Figure 3 combines both workbenches into one six-stage model. It shows where perimeter and host controls can catch each step. It is a model, not a timeline: nothing links port 8899 and port 8088 to one operation.

What the 16 matches are

The files on port 8088 contain 16 secret and command-and-control matches.

Match type	Count
Command-and-control endpoint references	9
Generic passwords	6
Square payment API token	1

The 12 paths in three groups

Inferred group	Paths (portal names withheld)	What the names suggest
1. Portal bypass and hash scripts	Four portal-named Python scripts: an admin-bypass script, a hash-fix script (`fix_hash_*.py`) and two exploit helpers	Testing portal login flaws and editing or restoring CMS password hashes
2. Webshell and process helpers	`/ak_inst.php` , `/bp2.php` , `/drop_shells.sh` , `/drop_proc.sh`	Secondary PHP webshell droppers and a Linux process-hiding script
3. Split Global Socket relay kit	A portal-named `_gs_deploy.php` , `/fix_fma_gs.sh` , a portal-named `gs_*.sh` , `/gsbin.chunk.ac`	A `gs-netcat` binary split into `gsbin.chunk.*` parts, and file manager (`fma`) helpers for an encrypted relay shell

Six of the file names carry a specific portal's name. Together with the live secrets, this weighs against a generic open-source test fixture. It does not rule out a research lab, a malware sandbox or a decoy server.

Why split the relay binary

Global Socket lets two machines connect through public relay servers, even behind NAT. It encrypts sessions with SRP and AES-256. Normally an attacker installs it with a bootstrap script from `gsocket[.]io` .

Splitting the `gs-netcat` binary into small chunks would avoid that download. Small chunks can also slip under CMS file-manager upload limits (`upload_max_filesize`) and static ELF signature scans. A helper script (`fix_fma_gs.sh`) would then join the chunks on the server.

Once running, native `gs-netcat` looks up relay hosts under `*.gs.thc[.]org` . It starts its encrypted session without a TLS SNI value. Both traits help defenders: the DNS lookup is easy to log, and a missing SNI on port 443 is unusual.

The name `drop_proc.sh` suggests a process-hiding helper. A common Linux trick binds an empty directory over `/proc/<pid>` with `mount --bind` , so tools such as `ps` no longer list the process.

CHAPTER 05

Workbench 1: Hospital Recon and VShell Staging

IN SHORT

Workbench 189.24.105[.]131:8899 in Germany exposes 719 files on a Python 3.12 web server. It holds hospital reconnaissance files, login tests against an internal console and VShell implant downloads.

INSIDE THIS CHAPTER

What the 719 files are 17

Hospital portals and internal systems 17

Login tests against an internal console 17

What the delivery log records 18

The co-hosted VShell ports 18

05 Workbench 1: Hospital Recon and VShell Staging

The server sits on NetLab Global (DE). It runs an open Python 3.12 `http.server` on port 8899. The same address also shows two suspected VShell listeners (8082, 8086) and two candidate ones (8091, 8093).

All hospital names, portal addresses and the addresses of hosts that downloaded files are withheld. How the downloading hosts relate to the hospital targets is unknown.

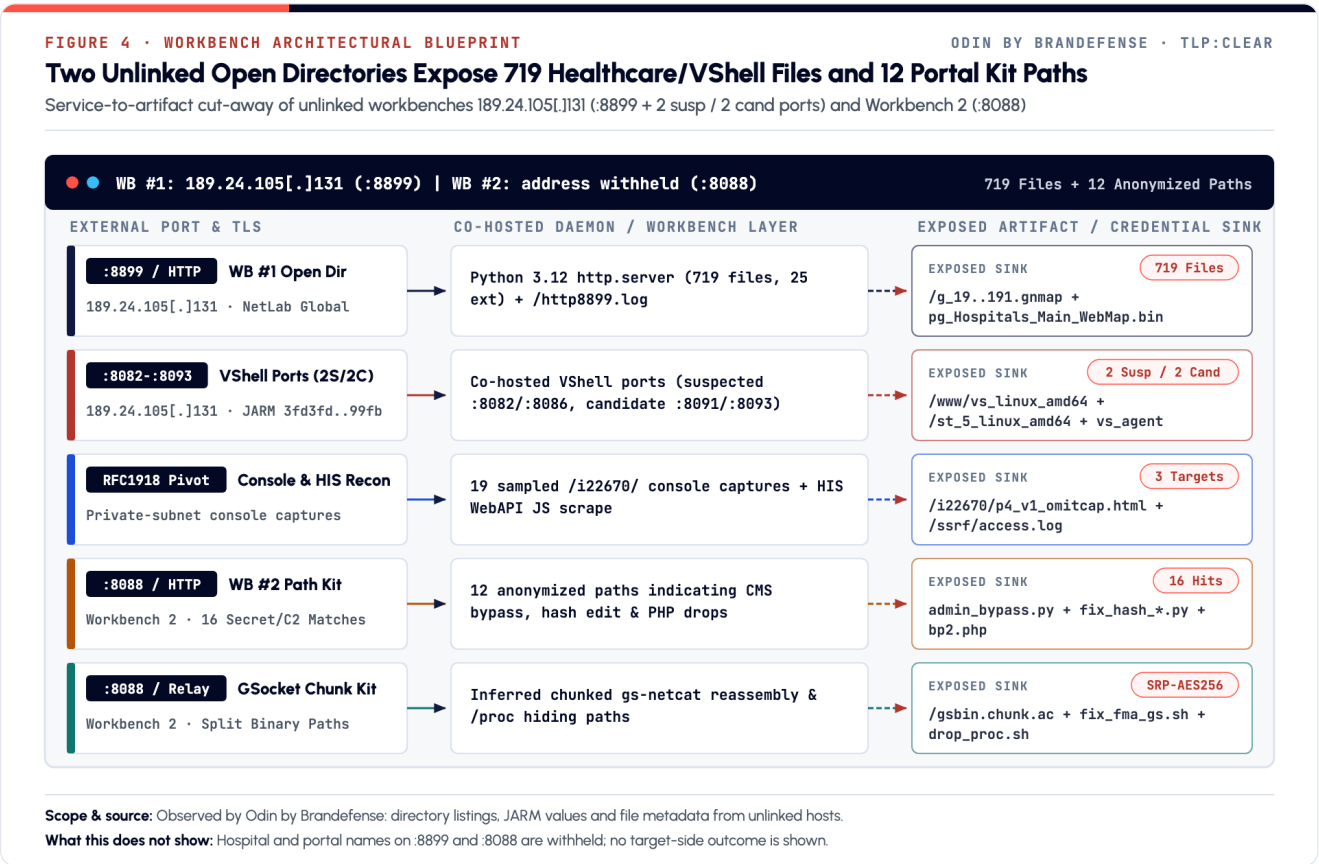


Figure 4: Two unlinked open directories expose 719 healthcare and VShell files and 12 portal kit paths

What the 719 files are

The 719 files span 25 extensions. JavaScript scraped from target management interfaces makes up 39.6% of them.

Extension group	Files	Share	Example	Role
`.js`	285	39.6%	Scraped HIS module scripts (names withheld)	Hospital information system and app-management UI modules
`.txt` and `.out`	102	14.2%	`/i22670/resp_post_dummy.txt`, `/i21706_scan.out`	Captured HTTP POST responses (58 `.txt`) and scan outputs (44 `.out`)
`.py` and `.sh`	91	12.7%	Python recon and replay scripts	Login replay scripts (56 `.py`) and Unix staging wrappers (35 `.sh`)
`.html` and `.body`	53	7.4%	`/i22670/p4_v1_omitcap.html`, `/logon.html`	Hospital login captures and console CAPTCHA-omission test pages (39 `.html`, 14 `.body`)
`.bin` and no extension	68	9.5%	`/pg_Hospitals_Main_WebMap.bin`, `/www/vs_linux_amd64`	Hospital web maps (34 `.bin`) and 9,800-byte Linux ELF stagers (34 without extension)
`.hdr`, `.http`, `.https`	39	5.4%	HTTP header captures	Raw request templates and response header dumps
`.log`, `.json`, `.err`	41	5.7%	`/http8899.log`, `/ssrf/access.log`, `/kharon_deps.log`	Delivery logs (15 `.log`), payload JSON (16) and scan error logs (10)
`.exe`, `.gnmap` and 8 others	40	5.6%	`/vs_agent.exe`, `/g_19.gnmap`	13 Windows PE files, 5 `.gnmap` scans, 2 `.ps1`, 2 `.pem`, plus `.tmp`, `.png` and `.gz`

Hospital portals and internal systems

The server stores two compiled web-map caches of hospital portals and a copy of one hospital login page (`/logon.html`). Subfolders `/i21931/` and `/i22001/` hold scripts scraped from a hospital information system (HIS). They include a password input module and an SSH host management module.

Some of these scripts came from internal web APIs on a private 10.x subnet, on ports 8099 and 8081. That means the operator could reach, or had copies from, systems that should not face the internet.

The folder names, such as `/i21931/` and `/i22670/`, look like engagement or ticket numbers. We do not treat that as proof of an authorized test, for reasons set out in Chapter 6.

Login tests against an internal console

Subfolder `/i22670/` holds 19 files from an internal ASP.NET management console. Its login page sits on the same private subnet; the path is withheld.

File type	Count	Detail
`.html` login captures	17	16 `Login.aspx` variants of 89,656 to 89,670 bytes, plus `post_dummy_saved.html` (88,424 bytes)
`.txt` response	1	`resp_post_dummy.txt`
`.body` capture	1	`r_masterjs.body`

The file names `p4_v1_omitcap.html`, `p4_v2_capempty.html` and `p4_v3_evttarget.html` point to three tests. They drop the CAPTCHA field, send it empty, and replay the ASP.NET `__EVENTTARGET` postback. A server-side request forgery (SSRF) callback log, `/ssrf/access.log` (60,550 bytes), sits next to them.

What the delivery log records

The file `/http8899.log` (120,784 bytes) is the server's own access log. It mixes implant downloads with internet proxy noise.

- **Windows agent downloads.** Five `GET /vs_agent.exe HTTP/1.1` entries carry timestamps on 31 October 2026, in server time with no recorded time zone. Two outside hosts received HTTP 200 responses. Their addresses are withheld.
- **Proxy noise.** Four minutes later, opportunistic proxy scanners received HTTP 501 `Unsupported method ('CONNECT')`.
- **Linux stagers.** Two 9,800-byte Linux AMD64 binaries sit on the server, `/www/vs_linux_amd64` and `/st_5_linux_amd64`. Cross-compilation logs (`/mingw_install.log`, `/kharon_deps.log`) sit beside them.

The log shows that files were served. It does not show what the downloading hosts did with them.

The co-hosted VShell ports

Port	Status	Detail
8082	Suspected VShell	Seen 13 November 2026
8086	Suspected VShell	Same host
8091	Candidate VShell	JARM `3fd3fd00000000000043d43d00043dc3b2afa8a5ec09b510a8559aff7899fb`
8093	Candidate VShell	Same JARM as 8091

JARM is a fingerprint of how a server answers TLS handshakes. A shared JARM across two ports suggests the same software stack.

CHAPTER 06

Scanner Fleets and the Dual-Use Problem

IN SHORT

Of 2,049 endpoints on 1,882 IPv4 addresses, 769 are suspected or candidate VShell nodes. The rest are ARL, Nessus, Acunetix and reNgine consoles that legitimate teams also run, so they need watchlists, not blocks.

INSIDE THIS CHAPTER

The five fleets 20
Could these be legitimate? 21
How to tell scanners from implants 21

06 Scanner Fleets and the Dual-Use Problem

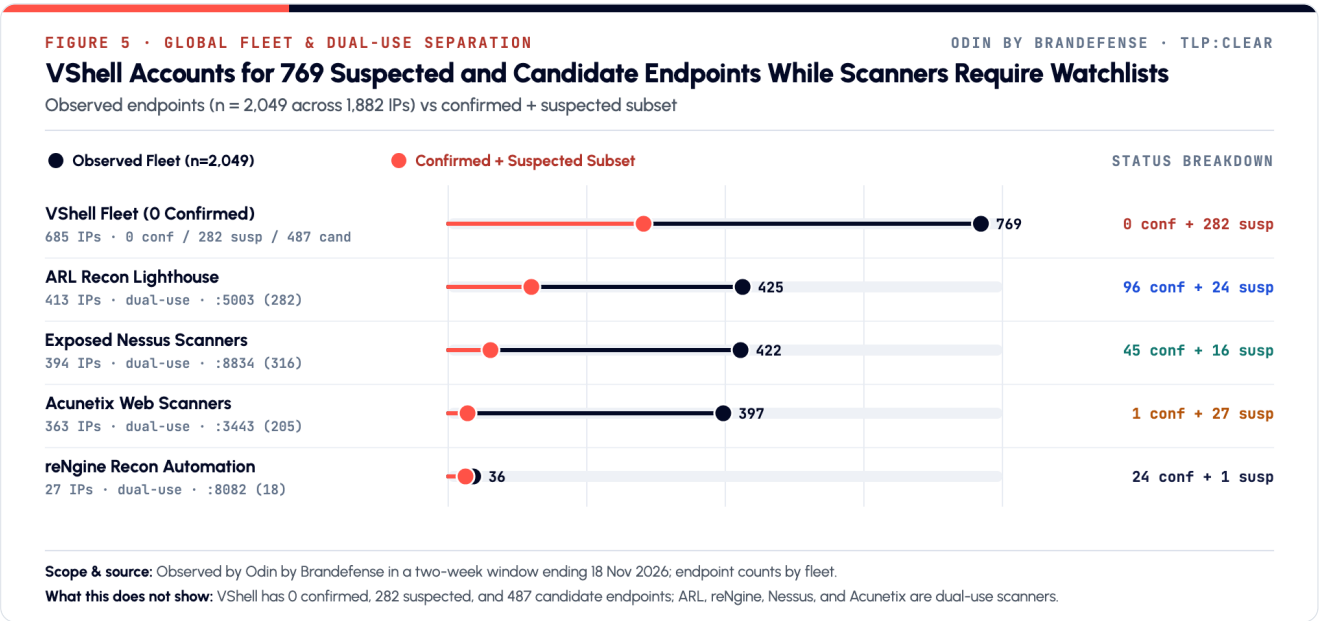


Figure 5: VShell accounts for 769 suspected and candidate endpoints while scanners require watchlists

The five fleets

Fleet	Endpoints (IPs)	Confirmed / suspected / candidate	Top ports	Top countries
VShell	769 (685)	0 / 282 / 487	9999 (227), 8084 (71), 9097 (68), 80 (21)	HK (270), CN (218), US (187)
ARL	425 (413)	96 / 24 / 305	5003 (282), 443 (95), 3389 (19)	CN (235), US (127), HK (46)
Nessus	422 (394)	45 / 16 / 361	8834 (316), 443 (42), 80 (19), 8000 (17)	US (136), CN (33), DE (31)
Acunetix	397 (363)	1 / 27 / 369	3443 (205), 7880 (58), 13443 (27)	US (83), FR (63), CN (50)
reNgine	36 (27)	24 / 1 / 11	8082 (18), 443 (9), 80 (5)	US (9), DE (8), FR (5)

For the scanner fleets, "confirmed" means the product is confirmed. It does not mean hostile use. Who runs these consoles, and whether they are authorized, cannot be told from a banner.

Two watchlist samples sit at one hosting provider, SonderCloud Limited: a suspected VShell node on 9999 (HK) and an ARL console on 443 (US). One sample each does not show a regional pattern or a shared operator. Their addresses are withheld.

Could these be legitimate?

We tested four other explanations before recommending any block.

Explanation	Workbench 1 (port 8899)	Workbench 2 (port 8088)
Authorized red team	Folder names look like engagement IDs, but the server is unauthenticated and serves implants to outside hosts	No sign of an engagement; portal-specific names and live secrets
Honeypot or decoy	Unlikely: real HTTP 200 implant downloads and co-hosted suspected VShell ports	Cannot be ruled out without script bodies or access logs
Research lab or sandbox	Unlikely for the same reasons	Cannot be ruled out
Open-source test fixture	Does not fit hospital-specific web maps	Six portal-named files weigh against it

The verdict differs by server. Workbench 1 has strong evidence and belongs on a block list. Workbench 2 rests on file names only, and a research lab or honeypot cannot be ruled out. We withhold its address and recommend hunting for its file markers with the YARA rule. Declaring an active intrusion would need host or portal evidence.

How to tell scanners from implants

1. **Watch exposed scanner consoles.** Nessus (8834), reNgine (8082), ARL (5003) and Acunetix (3443) consoles go on a watchlist. Rate-limit a source only when it also sends an inbound burst of 10 or more requests per second, or shares a host with an implant.
2. **Allowlist your own scanners narrowly.** Exempt the egress addresses of your own attack surface tools and contracted testers, for scanning traffic only. Never let that allowlist hide a contact with a workbench, a split binary, a webshell or a hash edit.
3. **Do not act on one network signal.** A single `*.gs.thc[.]org` lookup or `gsocket[.]io` connection calls for a DNS and host investigation, not containment. An inbound scan or postback burst calls for a temporary WAF limit.
4. **Contain a host only on two signals.** A single stager file, chunk or `/proc` bind mount calls for an EDR investigation. A webshell drop calls for quarantine and a web root audit. A confirmed CMS hash change calls for isolating the portal and rotating hashes. Automatic containment needs at least two signals, one of them a file or process on the host.

07

CHAPTER 07

What Defenders Should Do

IN SHORT

This chapter lists the actions defenders should take, in priority order.

07 What Defenders Should Do

Within 24 hours

- [ ] Block 189.24.105[.]131 (ports 8899, 8082, 8086). Hunt for the Workbench 2 file markers with the YARA rule.
- [ ] Alert on DNS lookups for `*.gs.thc[.]org` and on TLS connections to `gsocket[.]io`, and tie each to the host process.
- [ ] Put exposed ARL, reNgine, Nessus and Acunetix consoles on watchlists, not block lists.

Within 72 hours

- [ ] Bind CAPTCHA checks to the server-side session on ASP.NET and hospital login pages. A postback with a missing or empty CAPTCHA field must fail.
- [ ] Audit CMS user tables for unexpected password-hash changes. Remove web file-manager plugins and mount upload folders `noexec`.
- [ ] Inspect outbound WebSocket upgrades to ports 8082, 8084, 9097 and 9999.

Within 7 to 30 days

- [ ] Restrict internal HIS (8081, 8099) and management console interfaces to jump-host VLANs. Block SSRF egress from the web tier to private ranges and cloud metadata.
- [ ] Watch Linux web servers for `gsbin.chunk.*` files, `gs-netcat` processes and unexpected `/proc/<pid>` bind mounts (check `/proc/mounts`).
- [ ] Deploy the Suricata, KQL, Sigma and YARA rules in Chapter 8 after tuning allowlists.

CHAPTER 08

Detection Rules

IN SHORT

This chapter describes each published rule and where to deploy it.

INSIDE THIS CHAPTER

How the Suricata signatures work 25
How the other rules work 25
How to sort an event 26

08 Detection Rules

The full rules are in Appendix B. Each file covers a different layer: network, cloud SIEM, Linux host and files.

Rule file	What it catches
`odin_vshell_gsocket_arl_recon.rules`	Suricata 7: 12 signatures for VShell WebSocket and stager traffic, Global Socket relay DNS, ARL headers, chunk transfers and console postback bursts
`sentinel_gsocket_vshell_and_cms_hash_tamper.kql`	Microsoft Sentinel and Defender XDR: host process, file and network events, correlated per device in 1-hour windows
`sigma_linux_gsocket_and_process_hiding_persistence.yml`	Sigma (2 rules): Global Socket chunk rebuilds and relays, `/proc` bind-mount hiding and VShell stagers
`odin_vshell_gsocket_and_portal_staging_kit.yar`	YARA (2 rules): VShell delivery logs and portal kit markers

How the Suricata signatures work

SID	Looks for	Why
9514001	Outbound `Upgrade: websocket` with `Sec-WebSocket-Key` to ports 8082, 8084, 8086, 8091, 8093, 9097 or 9999	Low-severity hunt; pair with 9514002, 9514011 or 9514012
9514002	Outbound URI `/ws_agent` or `/ws_client`	VShell WebSocket agent endpoint (T1071)
9514003	`GET` for `vs_linux_amd64` or `st_5_linux_amd64`	Linux VShell stager download
9514004	DNS query ending `.gs.thc[.]org`	Global Socket relay pool lookup
9514005	Inbound `X-ARL-Token`, `ARL-Finger` or the ARL product name in headers	Hunt for ARL-marked scanning (TA0043)
9514006	`gsbin.chunk.` in a URI	Split Global Socket chunk transfer (T1190)
9514007	URI ending `/drop_shells.sh` or `/drop_proc.sh`	Shell dropper or process-hider fetch
9514008	Five or more `POST` requests to a `Login.aspx` page with `__EVENTTARGET` from one source in 60 seconds	Console postback replay burst
9514009	`GET` URI ending `/vs_agent.exe`	Windows VShell agent download
9514010	URI ending `/fix_fma_gs.sh`	Chunk reassembly helper fetch
9514011	Outbound URI containing `v_ver=`	VShell agent version beacon
9514012	Outbound URI containing `vshell`	VShell agent identifier

The inbound portal rules (9514005 to 9514008 and 9514010) need Suricata behind TLS offload to see the HTTP layer. A plain WebSocket handshake carries no VShell banner, so allowlist internal WebSocket services on the hunted ports. Native `gs-netcat` sends no SNI, while `gsocket[.]io` serves only HTTPS installer scripts.

How the other rules work

The KQL query looks back 14 days. It collects four kinds of events. The first is host processes such as `gs-netcat`, VShell stagers or `mount` binding over `/proc`. The second is web server workers (`php-fpm`, `apache2`, `nginx`, `w3wp.exe` and others) running hash-edit, bypass or dropper commands.

The third kind is file writes of stagers, PHP droppers or `gsbin.chunk.*` parts. The fourth is network connections to the two workbenches, `*.gs.thc[.]org`, `gsocket[.]io`, or suspected VShell ports from web workers. The query then groups events per device per hour.

It labels each group by strength of evidence. Host plus network signals, or two or more host signals, should be escalated at once. A single host signal goes to EDR review, and a single network signal goes to perimeter hunting. Do not suppress by host name; exclude only verified container runtimes (`runc`, `containerd-shim`).

The first Sigma rule fires on a process named `gs-netcat` or `gsocket`, or on command lines with `gsbin.chunk.`, `GSOCKET_ARGS=`, `fix_fma_gs.sh` or `_gs_deploy.php`. The second fires on `mount` with `/proc/` and `bind`, or on `drop_proc.sh`, `drop_shells.sh` and the two VShell stager names. Container runtime start-up is the main false positive.

YARA rule	Fires when	Status
`ODIN_VShell_Staging_Log_And_Agent_Delivery`	A Python HTTP log shows a successful `vs_agent.exe` download, or both Linux stager names appear	Checked against log excerpts
`ODIN_Portal_Hash_Tamper_And_GSocket_Chunk_Kit`	A chunk marker appears with a process-hiding or hash-edit marker, or both hash-edit and bypass markers appear	Heuristic; not checked against script bodies

How to sort an event

The triage logic never blocks on uncorroborated evidence.

What the event shows	Action
Your own documented scanner, with no implant signal	Allowlist
An exposed scanner console with no hostile traffic	Watchlist
An inbound scan burst or console postback burst	Temporary WAF rate limit and hunt
A single relay DNS or SNI signal	DNS and host investigation
Contact with a workbench address but no host artifact	Perimeter block and network hunt
A single stager, chunk or `/proc` bind-mount signal on a host	EDR investigation
A portal webshell drop	Quarantine the file and audit the web root
A CMS password-hash change	Isolate the portal and rotate hashes
Two or more signals, at least one on the host	Block and contain the host

09

CHAPTER 09

Indicators

IN SHORT

This chapter lists the published indicators and explains what stays withheld.

INSIDE THIS CHAPTER

Endpoints to block 28

File hashes for hunting 28

MITRE ATT&CK mapping 29

09 Indicators

Appendix A lists the published set of 4 indicators.

Type	Count
Workbench host to block (189.24.105[.]131, three ports)	1
File hashes (SHA-256) from workbench 1	3
Withheld: Workbench 2 address and three watchlist nodes	4
Withheld: hashes of scraped hospital, portal and console pages	11

Endpoints to block

Endpoint	Role	Hosting	Status
189.24.105[.]131:8899	Open directory, 719 files, VShell delivery log	NetLab Global (DE)	Block, strong evidence
189.24.105[.]131:8082	Co-hosted suspected VShell listener, seen 13 November 2026	NetLab Global (DE)	Block, strong evidence
189.24.105[.]131:8086	Co-hosted suspected VShell listener	NetLab Global (DE)	Block, strong evidence

Check current port banners and the JARM value before making a block permanent. The three withheld watchlist nodes are a suspected VShell node on 9999 (SonderCloud, HK), an ARL console on 443 (SonderCloud, US) and a reNgine console on 8082 (Hetzner, DE).

File hashes for hunting

All three published hashes come from workbench 1. None was seen on a target system. The two stager binaries are the ones to hunt on endpoints; the build log describes the staging server itself. The delivery log, the SSRF listener log and the scan output are withheld, because they hold third-party addresses.

File	Size (bytes)	SHA-256
`/www/vs_linux_amd64` (Linux ELF stager)	9,800	`5372c43e83132fad627f4bdab1a98c4a669053252404882afff40ed9cbae1bf4`
`/st_5_linux_amd64` (Linux ELF stager)	9,800	`162a74648d48a685c679f8ba5ede4bdd2fb1c690bc8446694a98521d1262add0`
`/kharon_deps.log` (implant build log)	20,919	`e8dfabda9665936e9f21f69d139117c5e65c9db754d9ebf42200fced8afb8cc0`

Eleven more hashes cover scraped hospital, portal and console pages. They are withheld, because they derive from victim content. Also withheld: the Workbench 2 address, the hospital and portal names, the two hosts that downloaded `vs_agent.exe`, the three watchlist addresses and all other suspected or candidate addresses.

MITRE ATT&CK mapping

Tactic	Technique	How it appears here	Detection
Reconnaissance	TA0043	ARL and reNgine discovery, Nmap sweeps, hospital web maps	SID 9514005; burst check
Initial Access	T1190 Exploit Public-Facing Application	Inferred CMS bypass and hash scripts; console CAPTCHA-omission tests	SIDs 9514006 to 9514008, 9514010; KQL web worker branch
Command and Control	T1071 Application Layer Protocol	VShell WebSocket agents and Global Socket relays	SIDs 9514001 to 9514004, 9514009, 9514011, 9514012; Sigma rules

10

CHAPTER 10

Scope and Limits

IN SHORT

This chapter sets out what the evidence can show and what it cannot.

10 Scope and Limits

- **Window.** The findings cover a two-week observation window ending 18 November 2026. The 31 October 2026 date is a timestamp inside the attacker's own access log, not an observation date.
- **Two servers, no link.** Nothing ties workbench 1, workbench 2 and the scanner fleets to one actor. The four-role grouping is an analytical model.
- **File names are not actions.** Workbench 2 capabilities come from file paths and the secrets in them. Script bodies and access logs were not available, and we saw no target-side change.
- **Served is not run.** The delivery log shows `vs_agent.exe` was served to two hosts. It does not show the file ran, or that those hosts belong to the hospital targets.
- **No confirmed VShell nodes.** All 769 VShell endpoints are suspected or candidate. Use them for hunting, not blocking.
- **Confirmed scanners are not hostile scanners.** A confirmed ARL, reNgine, Nessus or Acunetix console runs that product. Its owner may be an authorized tester.
- **Rule testing.** Every published rule file loads without errors in its engine (Suricata 7.0.12, YARA-X, pySigma and Microsoft's Kusto.Language analyzer, checked on 20 November 2026). The rules were not replayed against live traffic, so test and tune them in your own environment before production use.

11

CHAPTER 11

Sources

IN SHORT

This chapter lists the public references used in the report.

11 Sources

1. MITRE ATT&CK. Reconnaissance (TA0043). https://attack.mitre.org/tactics/TA0043/ (verified 18 November 2026)
2. MITRE ATT&CK. Exploit Public-Facing Application (T1190). https://attack.mitre.org/techniques/T1190/ (verified 18 November 2026)
3. MITRE ATT&CK. Application Layer Protocol (T1071). https://attack.mitre.org/techniques/T1071/ (verified 18 November 2026)
4. The Hacker's Choice. Global Socket (gsocket, gs-netcat). https://github.com/hackerschoice/gsocket (verified 18 November 2026)
5. Tophant. ARL system installation guide. https://github.com/tophant-ai/ARL-doc/blob/master/docs/system_install.md (verified 19 November 2026)
6. reNgine project. yogeshojha/rengine. https://github.com/yogeshojha/rengine (verified 18 November 2026)
7. Nmap Project. Greppable output (-oG). https://nmap.org/book/output-formats-grepable-output.html (verified 18 November 2026)

APPENDICES

Indicators and Detection Rules

FOR DEFENDERS

Appendix A lists the indicators published for this study. Appendix B carries every published detection rule in full.

INSIDE

A. Indicators of compromise (4) 35

B. Detection rules (4 files) 36

A Indicators of Compromise

Values are defanged for display. The CSV, JSON and STIX files on the online edition carry the raw values. Values the report marks as withheld are not listed.

ID	Type	Indicator	Port	Context	Action
R14-IOC-01	IPv4	189.24.105[.]131	8899	Healthcare/SMC Recon & VShell Staging Workbench	BLOCK
R14-IOC-02	SHA-256	5372c43e83132fad627f4bdab1a98c4a669053252404882afff40ed9cbae1bf4	8899	Staged VShell Linux AMD64 Binary	HUNT
R14-IOC-03	SHA-256	162a74648d48a685c679f8ba5ede4bdd2fb1c690bc8446694a98521d1262add0	8899	Staged Secondary Linux AMD64 Binary	HUNT
R14-IOC-04	SHA-256	e8dfabda9665936e9f21f69d139117c5e65c9db754d9ebf42200fced8afb8cc0	8899	Implant Cross-Compile Dependency Log	HUNT

B Detection Rules

Each rule file loads in its own engine (YARA-X, Suricata 7, Kusto, pySigma). Tune thresholds and allow-lists to your environment before alerting or blocking. The same files can be downloaded from the online edition.

odin_vshell_gsocket_and_portal_staging_kit.yar

YARA

```
/*
   Odin by Brandefense - Report #14 YARA Toolkit & Staging Artifact Suite
   Classification: TLP:CLEAR
   Scope: Two rules: one for VShell staging logs/ELFs and one heuristic rule for :8088
          portal/GSocket paths (raw :8088 script bodies withheld under TLP:CLEAR).
   Validation: Loads in YARA-X; not replayed against live samples.
*/

rule ODIN_VShell_Staging_Log_And_Agent_Delivery {
    meta:
        author = "Odin by Brandefense"
        classification = "TLP:CLEAR"
        report_id = "Odin-Monograph-14"
        severity = "high"
        validation_status = "loads_in_yara_x_checked_against_8899_log_excerpt"
        description = "Detects Python HTTP logs serving vs_agent.exe or VShell ELF stager names"
    strings:
        $s1 = "GET /vs_agent.exe HTTP/1.1\" 200" ascii
        $s2 = "/python3.12/http/server.py" ascii
        $s3 = "vs_linux_amd64" ascii
        $s4 = "st_5_linux_amd64" ascii
    condition:
        filesize < 2MB and (($s1 and $s2) or ($s3 and $s4))
}

rule ODIN_Portal_Hash_Tamper_And_GSocket_Chunk_Kit {
    meta:
        author = "Odin by Brandefense"
        classification = "TLP:CLEAR"
        report_id = "Odin-Monograph-14"
        severity = "medium"
        validation_status = "loads_in_yara_x_heuristic_raw_script_bodies_withheld"
        description = "Heuristic matching Workbench 2 (port 8088) path and env markers"
    strings:
        $g1 = "gsbin.chunk." ascii
        $g2 = "fix_fma_gs.sh" ascii
        $g3 = "GSOCKET_ARGS" ascii
        $p1 = "drop_proc.sh" ascii
        $p2 = "drop_shells.sh" ascii
        $t1 = "fix_hash_" ascii
        $t2 = "_admin_bypass.py" ascii
    condition:
        filesize < 500KB and ((1 of ($g*) and 1 of ($p*)) or (1 of ($g*) and 1 of ($t*)) or ($t1 and $t2))
}
```

odin_vshell_gsocket_arl_recon.rules

SURICATA

```
# Odin by Brandefense - Report #14 Suricata 7.x VShell, GSocket & ARL Recon Detection Suite
# Classification: TLP:CLEAR
# Scope: 12 network signatures covering generic non-standard WebSocket upgrades, VShell agent URIs and stager
downloads,
#        GSocket relay DNS lookups (*.gs.thc.org), ARL-marked HTTP headers, chunked gsbin staging, and console
postback bursts.
# 1. Generic Outbound WebSocket Upgrade Hunt on Non-Standard High Ports (8082-9999)
alert http $HOME_NET any -> $EXTERNAL_NET [8082,8084,8086,8091,8093,9097,9999] (msg:"ODIN-HUNT Outbound WebSocket
Upgrade on Non-Standard High Port"; flow:established,to_server; http.connection; content:"upgrade"; nocase;
```

```
http.header; content:"Upgrade: websocket"; nocase; fast_pattern; http.header_names; content:"Sec-WebSocket-Key";
nocase; metadata:created_at 2026_11_18, mitre_technique_id T1071, hunt_scope generic_websocket_port_hunt;
classtype:misc-activity; sid:9514001; rev:3;)
# 2. VShell WebSocket Agent Endpoint URI (/ws_agent or /ws_client)
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ODIN-VSHELL Outbound WebSocket Agent URI (/ws_agent or /
ws_client)"; flow:established,to_server; http.uri; content:"/ws_"; nocase; fast_pattern; pcre:"/\/ws_(?:agent|
client)(?:[\/?#]|$)/i"; metadata:created_at 2026_11_18, mitre_technique_id T1071; classtype:trojan-activity; sid:
9514002; rev:2;)
# 3. VShell Linux ELF Stager Binary Download (vs_linux_amd64 / st_5_linux_amd64)
alert http any any -> any any (msg:"ODIN-VSHELL Linux ELF Stager Binary HTTP Download";
flow:established,to_server; http.method; content:"GET"; http.uri; content:"_linux_amd64"; nocase; fast_pattern;
pcre:"/\/(vs_linux_amd64|st_5_linux_amd64)$/i"; metadata:created_at 2026_11_18, mitre_technique_id T1071;
classtype:trojan-activity; sid:9514003; rev:2;)
# 4. Global Socket (GSocket / gs-netcat) Default Relay Pool DNS Lookup (*.gs.thc.org)
alert dns $HOME_NET any -> any any (msg:"ODIN-GSOCKET Outbound DNS Query for Global Socket Relay Pool
(*.gs.thc.org)"; dns.query; content:".gs.thc.org"; nocase; endswith; fast_pattern; metadata:created_at
2026_11_18, mitre_technique_id T1071; classtype:trojan-activity; sid:9514004; rev:2;)
# 5. ARL (Asset Reconnaissance Lighthouse) Console API Token or Finger Header Hunt
alert http any any -> $HOME_NET any (msg:"ODIN-HUNT ARL Asset Reconnaissance Lighthouse HTTP Header Marker";
flow:established,to_server; http.header; content:"ARL"; fast_pattern; pcre:"/(X-ARL-Token|ARL-Finger|Asset
Reconnaissance Lighthouse)/i"; metadata:created_at 2026_11_18, mitre_tactic TA0043, hunt_scope
arl_marked_header_hunt; classtype:misc-activity; sid:9514005; rev:2;)
# 6. Split GSocket Binary Chunk (gsbin.chunk.*) HTTP Transfer
alert http any any -> any any (msg:"ODIN-GSOCKET Chunked Binary Stager HTTP Transfer"; flow:established;
http.uri; content:"gsbin.chunk."; nocase; fast_pattern; metadata:created_at 2026_11_18, mitre_technique_id T1190;
classtype:web-application-attack; sid:9514006; rev:1;)
# 7. Portal Shell or Process-Hiding Dropper Script Transfer (drop_shells.sh / drop_proc.sh)
alert http any any -> any any (msg:"ODIN-PORTAL-KIT Portal Shell Dropper or Process-Hider Script Fetch";
flow:established; http.uri; content:"/drop_"; nocase; fast_pattern; pcre:"/\/(drop_shells\.sh|drop_proc\.sh)$/i";
metadata:created_at 2026_11_18, mitre_technique_id T1190; classtype:web-application-attack; sid:9514007; rev:3;)
# 8. Internal ASP.NET Management Console Repeated Postback Replay Burst Hunt
alert http any any -> $HOME_NET any (msg:"ODIN-HUNT ASP.NET Login.aspx Repeated Postback Replay Burst";
flow:established,to_server; http.method; content:"POST"; http.uri; content:"/Login.aspx"; nocase; fast_pattern;
http.request_body; content:"__EVENTTARGET"; nocase; threshold:type both, track by_src, count 5, seconds 60;
metadata:created_at 2026_11_18, mitre_technique_id T1190; classtype:misc-activity; sid:9514008; rev:3;)
# 9. VShell Windows PE Agent Binary Download (/vs_agent.exe)
alert http any any -> any any (msg:"ODIN-VSHELL Windows PE Agent Binary HTTP Download (vs_agent.exe)";
flow:established,to_server; http.method; content:"GET"; http.uri; content:"/vs_agent.exe"; nocase; endswith;
fast_pattern; metadata:created_at 2026_11_18, mitre_technique_id T1071; classtype:trojan-activity; sid:9514009;
rev:1;)
# 10. GSocket File-Manager Chunk Reassembly Helper Script Fetch (/fix_fma_gs.sh)
alert http any any -> any any (msg:"ODIN-GSOCKET File-Manager Chunk Reassembly Script Fetch (fix_fma_gs.sh)";
flow:established; http.uri; content:"/fix_fma_gs.sh"; nocase; endswith; fast_pattern; metadata:created_at
2026_11_18, mitre_technique_id T1190; classtype:web-application-attack; sid:9514010; rev:1;)
# 11. VShell Agent Version Query Parameter Marker (v_ver=)
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ODIN-VSHELL Agent Version Query Parameter Beacon (v_ver=)";
flow:established,to_server; http.uri; content:"v_ver="; nocase; fast_pattern; metadata:created_at 2026_11_18,
mitre_technique_id T1071; classtype:trojan-activity; sid:9514011; rev:1;)
# 12. VShell Agent Identifier URI Marker (vshell)
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ODIN-VSHELL Agent Identifier URI Marker (vshell)";
flow:established,to_server; http.uri; content:"vshell"; nocase; fast_pattern; metadata:created_at 2026_11_18,
mitre_technique_id T1071; classtype:trojan-activity; sid:9514012; rev:1;)
```

sentinel_gsocket_vshell_and_cms_hash_tamper.kql

KQL (SENTINEL / DEFENDER XDR)

```
// Odin by Brandefense - Report #14 Microsoft Sentinel & Defender XDR Hunting Query
// Classification: TLP:CLEAR
// Collects host process, file-write, and network egress signals and correlates them per device
// across 1-hour windows to separate corroborated host+network activity from single-signal hunts.
let Lookback = 14d;
let CorrelationWindow = 1h;
// Workbench 1 matches on IP; workbench 2 is a precautionary finding on port 8088 only.
let IsWorkbench = (ip:string, port:int) { ip == "189.24.105.131" };
let SuspectedVShellPorts = dynamic([8082, 8084, 8086, 8091, 8093, 9097, 9999]);
let WebServerWorkers = dynamic(["php-fpm", "php", "apache2", "httpd", "nginx", "w3wp.exe"]);
union isfuzzy=true
(
    DeviceProcessEvents
```

```
    | where Timestamp >= ago(Lookback)
    | where FileName in~ ("gs-netcat", "vs_agent.exe", "vs_linux_amd64", "st_5_linux_amd64")
        or ProcessCommandLine matches regex @"gsbin\.chunk\.|GSOCKET_ARGS|fix_fma_gs\.sh|drop_proc\.sh|
drop_shells\.sh"
        or (FileName =~ "mount" and ProcessCommandLine contains "/proc/" and ProcessCommandLine contains "bind")
    | extend SignalDomain = "HostProcess",
             HuntSignal = "GSocket_VShell_Or_ProcHide_Execution",
             ArtifactDetail = ProcessCommandLine
    | project Timestamp, DeviceName, SignalDomain, HuntSignal, InitiatingProcessFileName, FileName,
ArtifactDetail
),
(
    DeviceProcessEvents
    | where Timestamp >= ago(Lookback)
    | where InitiatingProcessFileName in~ (WebServerWorkers)
    | where ProcessCommandLine matches regex @"fix_hash_|admin_bypass|_gs_deploy\.php|ak_inst\.php|bp2\.php|gs-
netcat"
    | extend SignalDomain = "HostProcess",
             HuntSignal = iff(ProcessCommandLine matches regex @"fix_hash_|admin_bypass",
"WebWorker_Portal_HashTamper_Or_AuthBypass", "WebWorker_Portal_Webshell_Or_Relay_Drop"),
             ArtifactDetail = ProcessCommandLine
    | project Timestamp, DeviceName, SignalDomain, HuntSignal, InitiatingProcessFileName, FileName,
ArtifactDetail
),
(
    DeviceFileEvents
    | where Timestamp >= ago(Lookback)
    | where FileName in~ ("vs_agent.exe", "vs_linux_amd64", "st_5_linux_amd64", "ak_inst.php", "bp2.php")
        or FileName startswith "gsbin.chunk."
    | extend SignalDomain = "HostFile",
             HuntSignal = "Staged_Implant_Or_Split_GSocket_Chunk_Write",
             ArtifactDetail = strcat(FolderPath, "/", FileName)
    | project Timestamp, DeviceName, SignalDomain, HuntSignal, InitiatingProcessFileName, FileName,
ArtifactDetail
),
(
    DeviceNetworkEvents
    | where Timestamp >= ago(Lookback) and ActionType == "ConnectionSuccess"
    | where IsWorkbench(RemoteIP, RemotePort)
        or RemoteUrl endswith ".gs.thc.org"
        or InitiatingProcessFileName in~ ("gs-netcat", "gsocket")
        or RemoteUrl =~ "gsocket.io"
        or RemoteUrl endswith ".gsocket.io"
        or (RemoteIPType == "Public" and RemotePort in (SuspectedVShellPorts) and InitiatingProcessFileName in~
(WebServerWorkers))
    | extend SignalDomain = "NetworkEgress",
             HuntSignal = case(
                 IsWorkbench(RemoteIP, RemotePort), "Verified_Staging_Workbench_Egress",
                 RemoteUrl endswith ".gs.thc.org" or InitiatingProcessFileName in~ ("gs-netcat", "gsocket"),
"GSocket_Relay_Pool_Egress",
                 RemoteUrl =~ "gsocket.io" or RemoteUrl endswith ".gsocket.io",
"GSocket_Bootstrap_Download_Egress",
                 "WebWorker_Suspected_VShell_Port_Egress"
             ),
             ArtifactDetail = strcat(RemoteIP, ":", tostring(RemotePort), " ", RemoteUrl)
    | project Timestamp, DeviceName, SignalDomain, HuntSignal, InitiatingProcessFileName, FileName =
InitiatingProcessFileName, ArtifactDetail
)
| summarize
    FirstSeen = min(Timestamp),
    LastSeen = max(Timestamp),
    EventCount = count(),
    DistinctSignals = dcount(HuntSignal),
    HasHostSignal = countif(SignalDomain in ("HostProcess", "HostFile")) > 0,
    HasNetworkSignal = countif(SignalDomain == "NetworkEgress") > 0,
    HuntSignals = make_set(HuntSignal, 10),
    Processes = make_set(FileName, 10),
    SampleDetails = make_set(ArtifactDetail, 5)
    by DeviceName, WindowStart = bin(Timestamp, CorrelationWindow)
| extend CorroborationTier = case(
    HasHostSignal and HasNetworkSignal, "MultiDomain_Host_And_Network_Corroborated",
```

```
    DistinctSignals >= 2 and HasHostSignal, "MultiSignal_Host_Corroborated",
    HasHostSignal, "SingleSignal_Host_Investigation",
    "SingleSignal_Network_Hunt"
)
| order by HasHostSignal desc, DistinctSignals desc, EventCount desc
```

sigma_linux_gsocket_and_process_hiding_persistence.yml

SIGMA

```
title: Linux GSocket Split-Binary Reassembly and Covert Relay Persistence
id: 7d41c9a2-58f3-4e19-9b62-14a095140001
status: experimental
description: Detects chunked Global Socket (gsbin.chunk.*) reassembly or gs-netcat relay execution.
author: Odin by Brandefense
date: 2026-11-18
tags:
  - attack.persistence
  - attack.command_and_control
logsource:
  category: process_creation
  product: linux
detection:
  selection_gs_bin:
    Image|endswith:
      - '/gs-netcat'
      - '/gsocket'
  selection_gs_cli:
    CommandLine|contains:
      - 'gsbin.chunk.'
      - 'GSOCKET_ARGS='
      - 'fix_fma_gs.sh'
      - '_gs_deploy.php'
  condition: selection_gs_bin or selection_gs_cli
falsepositives:
  - Documented administrative relays; confirm each case manually and do not suppress by host
level: high
---
title: Linux Procfs Bind-Mount Process Hiding and VShell ELF Stager Execution
id: 7d41c9a2-58f3-4e19-9b62-14a095140002
status: experimental
description: Detects userland PID hiding via mount --bind over /proc/<pid> and Linux VShell stagers.
author: Odin by Brandefense
date: 2026-11-18
tags:
  - attack.persistence
  - attack.command_and_control
logsource:
  category: process_creation
  product: linux
detection:
  selection_proc_bind:
    Image|endswith: '/mount'
    CommandLine|contains|all:
      - '/proc/'
      - 'bind'
  selection_stager_names:
    CommandLine|contains:
      - 'drop_proc.sh'
      - 'drop_shells.sh'
      - '/vs_linux_amd64'
      - '/st_5_linux_amd64'
  condition: selection_proc_bind or selection_stager_names
falsepositives:
  - Container runtime initialization namespaces (exclude verified containerd or runc daemons)
level: critical
```

ODIN

About Odin Research

Odin Research is the public research series of Odin by Brandefense. It documents adversary infrastructure seen on the internet and turns each finding into indicators and detection rules that defenders can use. Victims and third parties stay anonymous, addresses appear only for confirmed adversary infrastructure, and no working attack code is published.

Odin is the agentic analyst inside the Brandefense threat intelligence platform, which unifies Cyber Threat Intelligence, Digital Risk Protection and External Attack Surface Management.

Online edition, indicators and rules · brandefense.io/odin-research/reports/vshell-staging-and-recon-workbenches/
Short Read · brandefense.io/odin-research/blog/attacker-workbenches-map-hospitals-and-stage-vshell-implants/
Defense Kit · brandefense.io/odin-research/defense-kit/