ODIN

TLP:CLEAR

ODIN RESEARCH · MONOGRAPH 38 / 30 · CREDENTIAL THEFT

Secret Harvesting Pipelines
Git, JavaScript and Cloud Keys

How five exposed directories show Git and JavaScript scraping, bulk GitHub and Stripe token staging, and AWS IAM key checks

Two-week observation window ending 20 November 2026 · Prepared November 2026
Full Report · Public edition · Version 1.1

4,656
Unique secret findings observed in the two-week window ending 20 November 2026

1,001
Secret records in the latest arrival set across 35 secret types

5
Open-directory harvester and key-testing servers reviewed (0 shared hosts; all addresses and ports withheld)

193
Secret records across 15 files on the five servers (4.15% of the two-week total)

What Is in This Report. 10 chapters and two appendices.

How five exposed directories show Git and JavaScript scraping, bulk GitHub and Stripe token staging, and AWS IAM key checks

01	Executive Summary	04
02	Five Workbenches and the Five-Stage Model	07
03	Git and JavaScript Scrapers, TruffleHog Scans, and Callback Logs	10
04	Bulk GitHub and Stripe Token Dumps and AWS IAM Validators	13
05	What Secrets Harvesters Keep, and Other Explanations	17
06	What Defenders Should Do	20
07	Detection Rules	22
08	Indicators	25
09	Scope and Limits	27
10	Sources	29
A	Indicators of compromise (0)	31
B	Detection rules (4 files)	32

IN SHORT

In a two-week observation window ending 20 November 2026, Odin by Brandefense observed five open web directories holding 193 secret records within 4,656 secret findings and 1,001 arrivals. No network links connect the five servers, so we group them as an analytical five-stage model. Together they show exposed Git and JavaScript scrapers, TruffleHog repository scans, plain-text GitHub and Stripe dumps, AWS IAM check scripts, and Telegram alert bots. Block web access to `.git` folders, keep cloud keys out of frontend bundles, and alert on burst AWS identity checks.

CHAPTER 01

Executive Summary

IN SHORT

In a two-week observation window ending 20 November 2026, Odin by Brandefense observed five open web directories holding 193 secret records within 4,656 secret findings and 1,001 arrivals. No network links connect the five servers, so we group them as an analytical five-stage model. Together they show exposed Git and JavaScript scrapers, TruffleHog repository scans, plain-text GitHub and Stripe dumps, AWS IAM check scripts, and Telegram alert bots. Block web access to `.git` folders, keep cloud keys out of frontend bundles, and alert on burst AWS identity checks.

INSIDE THIS CHAPTER

Three Decisions for Leaders 04

The Study in Numbers 05

01 Executive Summary

Cloud intrusions often start with leaked keys rather than software exploits, a risk teams can monitor through cyber threat intelligence. Automated tools scan websites for exposed `.git` folders and client-side JavaScript files, extract tokens, sort them by provider, and test which ones still work.

When operators or security researchers leave their own working directories open over HTTP, external attack surface management can help defenders find those exposed files. We reviewed the files statically, tested zero credentials, and withheld all five host addresses, ports, credential-file paths, and file hashes.

1. **Five separate open directories show each step of automated secret collection and cloud key testing.** Nothing links the five hosts to one operator, so we present their combined workflow as an analytical model. Separate directories hold scraped `.git/config` files, cached JavaScript keys, TruffleHog scan outputs, plain-text GitHub and Stripe lists, shell scripts that check AWS IAM keys, and Telegram bot configs.

2. **One United States server concentrates 140 GitHub and Stripe tokens in two plain-text files.** An open aggregator directory holds 106 GitHub tokens in one flat file and 34 Stripe API keys in a second text dump. Across all five hosts, version-control tokens make up 117 of the 193 secret records (60.6%), followed by 34 Stripe keys (17.6%) and 29 AWS IAM keys (15.0%).

3. **Cloud key test scripts and callback folders leave clear defensive signals.** A Singapore directory pairs AWS CLI version 2 files with shell scripts holding 19 AWS access keys and 2 secret keys. A United States directory holds TruffleHog outputs, a Python alerting bot, key-check scripts, and 199 HTTP callback logs from 119 source addresses.

Three Decisions for Leaders

1. **Keep secrets out of web roots and browser code.** Block HTTP requests to `/.git/` and `.env` paths at the reverse proxy, verify that frontend builds never embed AWS or bot keys, and turn on repository push protection.
2. **Replace long-lived cloud and payment keys with scoped tokens.** Move CI/CD jobs from static AWS `AKIA` keys to short-lived OpenID Connect (OIDC) roles, use fine-grained GitHub tokens, and replace broad Stripe secret keys with restricted keys.
3. **Watch cloud identity checks and server egress.** Alert in AWS CloudTrail when `GetCallerIdentity` calls spike from unfamiliar networks, flag server connections to `api.telegram[.]org`, and check your own cloud subnets for open directory listings.

The Study in Numbers

Number	What it is
4,656	Unique secret findings observed in the two-week window ending 20 November 2026
1,001	Secret records in the latest arrival set across 35 secret types
5	Open-directory harvester and key-testing servers reviewed (0 shared hosts; all addresses and ports withheld)
193	Secret records across 15 files on the five servers (4.15% of the two-week total)
140	Tokens in two plain-text files on one US aggregator host (106 GitHub tokens and 34 Stripe keys)
29	AWS IAM key records across three hosts (27 access keys and 2 secret keys)
199	Sampled HTTP callback log files from 119 source addresses on the US TruffleHog and callback host

How to use this report.

Security leaders can start with Chapters 1, 5 and 6. Cloud IAM, AppSec and web platform teams should read Chapters 3, 4 and 6. SOC analysts and detection engineers should go to Chapters 7 and 8, along with Appendix B. CTI analysts will find the workspace breakdown and alternative explanations in Chapters 2 through 5.

02

CHAPTER 02

Five Workbenches and the Five-Stage Model

IN SHORT

Five open web directories held 193 secret records across 15 files within 4,656 two-week findings. Because the five hosts share no infrastructure links, we treat their tools as a five-stage analytical model for defenders rather than one actor group. Review each workbench separately before blocking.

INSIDE THIS CHAPTER

The wider two-week secret baseline 07

Why we model five stages instead of one actor 08

02 Five Workbenches and the Five-Stage Model

Public repositories are only one place where credentials leak, which is why dark web monitoring looks well beyond code hosts. Deployed websites also expose `.git/config` files when teams pull code straight into a web root, and single-page web apps often bake API keys into compiled JavaScript bundles. Once a script collects those strings, a second set of tools filters out dead keys by calling cloud and SaaS identity endpoints.

When someone leaves an unauthenticated HTTP file listing open over a working folder, the directory layout shows how the workflow is organized. We read file metadata and sanitized secret counts only. We did not run any of the scripts, test any of the keys, or contact any system named in them.



Figure 1: Five independent workbenches map the secret-harvesting lifecycle

The wider two-week secret baseline

In the two-week observation window ending 20 November 2026, Odin by Brandefense observed 4,656 unique secret findings. Within that set, the latest arrival slice holds 1,001 records across 35 secret types. Five types make up 675 of the 1,001 latest arrivals (67.4%). They are database connection strings (255), GitHub tokens (143), AWS access keys (106), BIP-39 wallet seed phrases (92), and Telegram bot tokens (79).

Enterprise cloud, source-control and payment credentials fill out the next tier of arrivals. Private keys account for 69 records, Stripe API keys for 40, Google Cloud service accounts for 39, Binance keys for 21, and Square tokens for 19. Twenty-five other secret types account for the remaining 138 records, including 16 Google OAuth secrets, 15 AWS secret keys, 15 GitHub personal access tokens, and 11 GitLab tokens.

Secret type	Latest arrivals (1,001)	Share	Typical source
`db_connection`	255	25.47%	Exposed `.env` files, container configs and TruffleHog JSON scans
`github_token`	143	14.29%	Plain-text token dumps, `.git/config` files and bot scripts
`aws_access_key`	106	10.59%	Shell IAM check scripts, JavaScript bundle caches and cloud configs
`bip39_mnemonic`	92	9.19%	Wallet backup files and stealer logs
`telegram_bot_token`	79	7.89%	Alerting bot configs, `.env` files and web controllers
`private_key`	69	6.89%	Exposed TLS, SSH and service PEM private keys
`stripe_key`	40	4.00%	Payment key flat files and backend `.env` configs
`gcp_service_account`	39	3.90%	Google Cloud service-account JSON key files
`binance_key`	21	2.10%	Exchange bot configs and `.env` files
`square_token`	19	1.90%	Merchant checkout configs and backend scripts
Remaining 25 types	138	13.79%	Includes 16 GCP OAuth secrets, 15 AWS secret keys, 15 GitHub PATs and 11 GitLab tokens

Why we model five stages instead of one actor

Five open-directory servers in our dataset focus specifically on harvesting, sorting or testing third-party credentials. Together they hold 193 secret records across 15 secret-bearing files (4.15% of the 4,656 two-week total). Exactly 192 of the 193 records sit inside the 1,001-record arrival slice, plus one Telegram bot token in a bot environment file on the US TruffleHog workbench.

Nothing ties these five servers to each other. They share 0 IP addresses, 0 TLS certificates, 0 SSH keys and 0 bot tokens. Grouping them as an analytical five-stage model maps each defensive control point, from `.git` and JavaScript scraping through TruffleHog scanning, flat-file aggregation, AWS IAM testing, and bot alerting.

Workbench role	Country	Exposed / sampled files	Secret records (193)	Main artifacts on the server
Exposed Git collector	Unattributed	Secret index only	8	Consolidated Git config text dump (7 GitLab, 1 GitHub)
JavaScript key scraper	Unattributed	Secret index only	9	Cached JS key dump (7 AWS), Telegram bot configs (2)
TruffleHog and alerting host	US	10,000 / 293 (220 rows)	15	TruffleHog scan JSONs, Python alerting bot, 199 callback logs
Bulk GitHub and Stripe dump	US	46 / 46	140	GitHub token dump (106), Stripe key dump (34), 17 archives
AWS IAM validator	SG	287 / 150	21	Two AWS IAM check scripts (17 and 4 records), AWS CLI v2

CHAPTER 03

Git and JavaScript Scrapers, TruffleHog Scans, and Callback Logs

IN SHORT

Three open directories show how harvesters pull credentials out of web roots, frontend JavaScript bundles and Git repositories while logging HTTP callbacks and Telegram alerts. Block web requests for `/.git/config` and strip cloud keys out of client builds.

INSIDE THIS CHAPTER

The US TruffleHog, alerting bot and callback workbench 10

JavaScript bundle scraping and Git metadata harvesting 11

03 Git and JavaScript Scrapers, TruffleHog Scans, and Callback Logs

Upstream collection targets mistakes made during web deployment and source-code management. One mistake is deploying an application with `git clone` inside the public web root, which leaves `/.git/config` readable over HTTP. Another is passing cloud keys into frontend build variables that compile straight into public `.js` files (T1552.001).

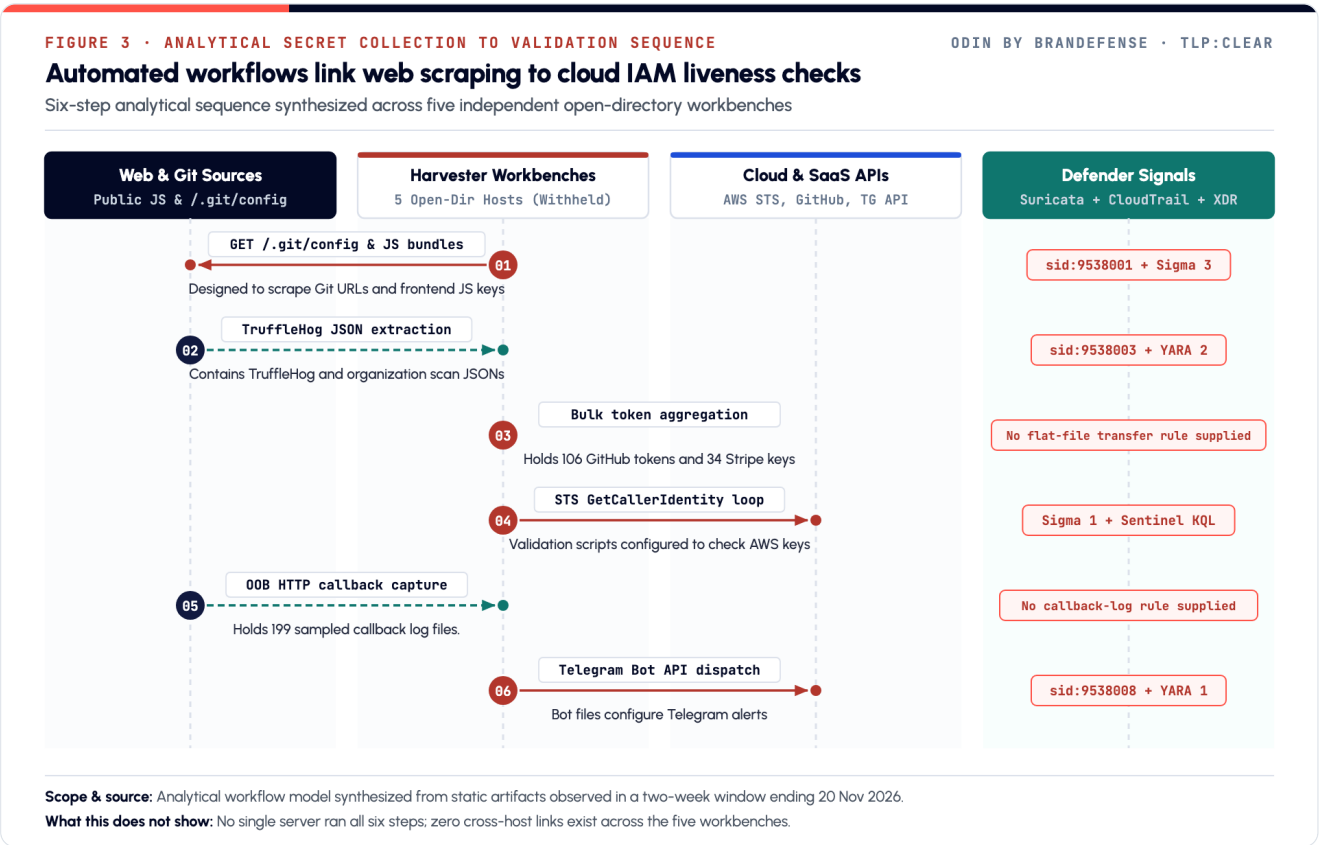


Figure 3: Automated workflows link web scraping to cloud IAM liveness checks

The US TruffleHog, alerting bot and callback workbench

In the United States, an open directory reports 10,000 exposed files, with 293 sampled entries (220 detailed rows in our review). Fifteen secret records sit across seven files on this host. They comprise 10 database connection strings, 2 GitHub personal access tokens, 1 GitHub token, 1 AWS access key, and 1 Telegram bot token.

In its scan directory, the host keeps a filtered TruffleHog JSON summary file (2 database connection strings) next to two full organization scan files. All target organization names and file paths are withheld. The first organization scan JSON is 55,916 bytes and holds 6 database connection strings; the second is 3,807 bytes and holds 1 database connection string. One more database connection string sits in a 584-byte application template file (name and path withheld).

A companion bot folder holds seven files that make up a custom Python alerting and key-testing setup:

- **Alerting bot script and two versioned backups.** The main Python script is 78,125 bytes and holds 1 GitHub token. Beside it sit two numbered backup copies (72,010 bytes and 78,125 bytes).

- **Configuration and environment files.** A TOML configuration file (37,656 bytes) defines the bot's rules and targets, while an environment file (415 bytes) holds 2 GitHub personal access tokens and 1 Telegram bot token.
- **Key and token verifiers.** Two modular Python verifier scripts (3,435 bytes, holding 1 AWS access key, and 2,934 bytes) separate key checks from the main bot loop.

The same workbench holds TLS material, three cloud blob scripts, a small configuration file, and callback logs. The sampled callback folder contains 199 files associated with 119 source addresses; 118 are external and withheld. Of those files, 173 have recorded sizes, while 26 directory entries have no size.

The callback log filenames use date prefixes in June and July 2026 (20260616, 20260622, 20260718 to 20260721, and 20260724). Those dates are timestamps inside filenames stored on the server, not Odin's observation period.

Artifact on the US TruffleHog workbench	Size (bytes)	Secret records	Purpose in the workspace
Organization TruffleHog scan A JSON	55,916	6 `db_connection`	Full TruffleHog scan output for an anonymized target organization
TruffleHog summary JSON	Unsized	2 `db_connection`	Filtered TruffleHog JSON summary output
Organization TruffleHog scan B JSON	3,807	1 `db_connection`	Secondary organization TruffleHog JSON scan output
Application template file (name withheld)	584	1 `db_connection`	Template file holding one database connection string
Custom Python alerting bot (+ two backups)	78,125 (72,010 / 78,125)	1 `github_token`	Custom secret-alerting bot across versioned backups
Bot TOML configuration file	37,656	0	Configuration file for the alerting bot
Bot environment file	415	2 `github_pat`, 1 `telegram_bot_token`	Bot environment credentials
Python key and token verifier scripts	3,435 / 2,934	1 `aws_access_key`	Modular cloud-key and token verification scripts
Callback TLS certificate and key, blob scripts, C2 `.env`	1,123 / 1,704; 2,953 to 4,476; 25	0	Co-hosted listener TLS material, three blob scripts, and a 25-byte config
Sampled HTTP callback log files	12 to 24,965	0 (199 log files)	Out-of-band HTTP callback logs across 119 source addresses (118 external + `127.0.0.1`)

JavaScript bundle scraping and Git metadata harvesting

Two smaller workbenches show specialized web scraping before any repository clone or cloud login takes place:

- **JavaScript bundle key scraper (address withheld).** Stores 7 AWS access keys in a cached JavaScript file beside a Telegram bot config and a PHP controller (2 bot tokens; 9 records total).
- **Exposed Git collector (address withheld).** Stores 8 secret records in a text dump (7 GitLab tokens and 1 GitHub token) pulled from public `/.git/config` URLs such as `hxxps://oauth2:<token>@gitlab[.]com/...` (T1552.001, T1528).

04

CHAPTER 04

Bulk GitHub and Stripe Token Dumps and AWS IAM Validators

IN SHORT

The staged files may contain example, expired or active keys; their status was not tested. One US server holds 140 GitHub and Stripe token records in two text files. A Singapore server holds 21 AWS key records in two shell scripts. Review your own exposed directories and rotate affected credentials.

INSIDE THIS CHAPTER

Bulk GitHub and Stripe aggregation (US) 13

Automated AWS IAM key validation (Singapore) 14

04 Bulk GitHub and Stripe Token Dumps and AWS IAM Validators

Raw scraping produces noise: example keys from documentation, expired tokens, and internal database URLs that do not resolve from the internet. Two open directories show how operators turn raw text hits into sorted provider lists and AWS IAM credential-check scripts.

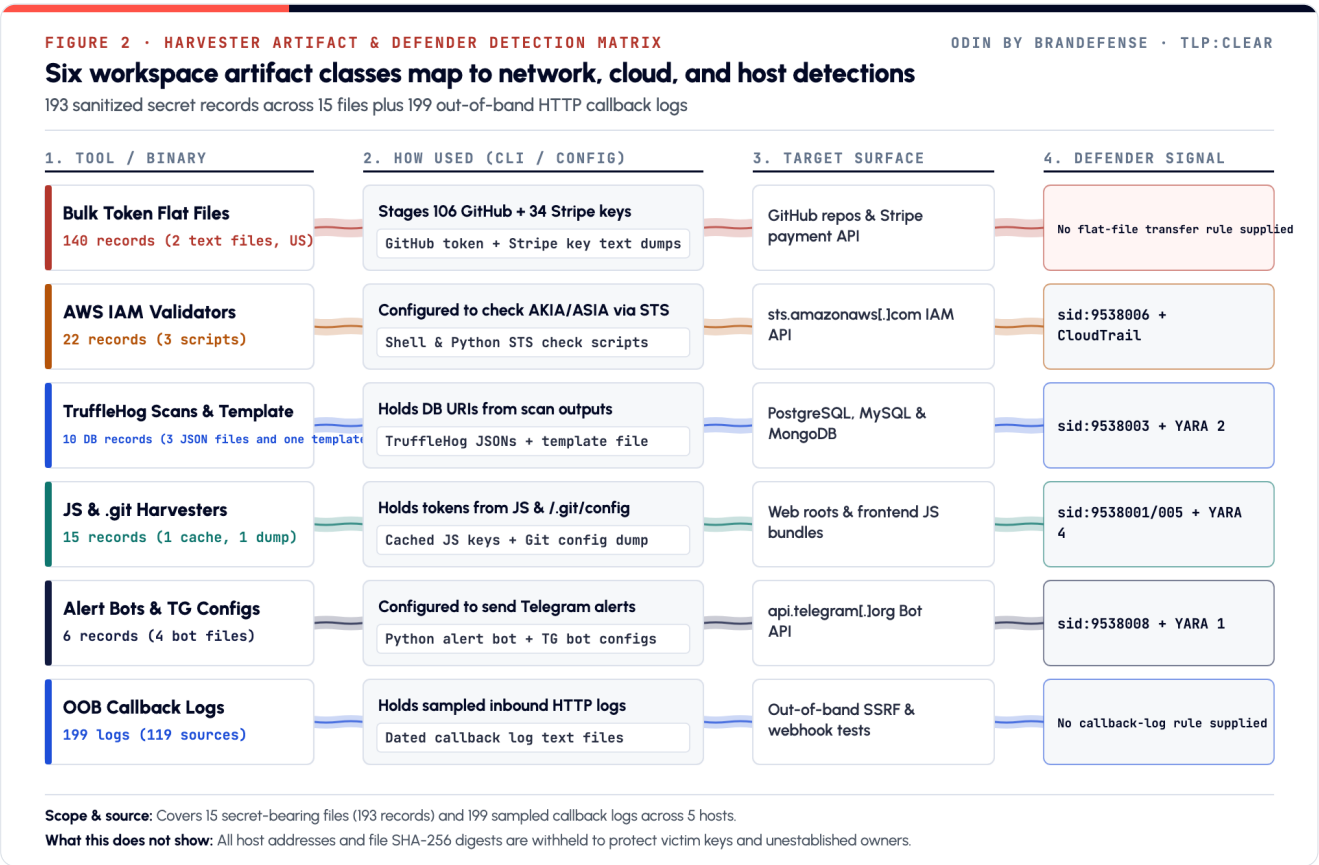


Figure 2: Six workspace artifact classes map to network, cloud, and host detections

Bulk GitHub and Stripe aggregation (US)

In the United States, an open aggregator directory exposes 46 files in its root folder. Those files comprise 18 text and log files, 17 archives, 8 scripts, and 3 key or PID files. Two plain-text files hold 140 secret records, which is 72.5% of all 193 secrets across the five workbenches:

- **GitHub token flat file (106 `github_token` records).** Collects 106 GitHub tokens in one newline-separated text dump for repository cloning or CI inspection (T1528).
- **Stripe key flat file (34 `stripe_key` records).** Collects 34 Stripe API keys next to unindexed sibling key lists for cloud, AI and email providers. Unrestricted Stripe secret keys allow balance checks, customer reads, or refund calls unless scoped with restricted keys and IP allowlists.

The other 44 files suggest a receiver and packaging setup. They include two Python listeners, several logs, six shell scripts, five host summary files, and 17 archives. Their names and locations are withheld because the exposed server may still serve credential-bearing files.



Figure 4: Open HTTP directories expose harvester outputs alongside validator scripts

Automated AWS IAM key validation (Singapore)

In Singapore, an open directory reports 287 exposed files, with 150 sampled entries in our review. Those 150 files include 61 text/HTML/JSON files, 47 PDFs, 25 cookie/runtime files, 9 scripts, 4 shared libraries, and 4 archives. Twenty-one AWS IAM credential records sit inside two shell validation scripts: one holding 17 `aws_access_key` records and a second holding 2 `aws_access_key` and 2 `aws_secret_key` records.

Beside those two files sit unpacked AWS CLI v2 installation files, four cloud check scripts, and three SMTP test scripts. Batch AWS validators call the AWS Security Token Service (`STS`) `GetCallerIdentity` action at `sts.amazonaws[.]com` (T1078.004, T1580). An IAM permission policy cannot deny `sts:GetCallerIdentity`. Any active `AKIA` or `ASIA` key therefore returns its Account ID and ARN while writing a CloudTrail event inside the owning account.

Workbench	Artifact group	Files	Secret records	Role in the workspace
US aggregator	GitHub token flat file	1	106 `github_token`	Bulk GitHub token list (54.9% of all 193 workbench records)
US aggregator	Stripe key flat file	1	34 `stripe_key`	Bulk Stripe API key list (85.0% of the 40 latest Stripe arrivals)
US aggregator	Packaged archives (names withheld)	17	Unindexed	Credential and honeypot archives withheld to protect exposed data
US aggregator	Collection listeners and shell scripts	8	0	Two Python collection listeners and six host-processing shell scripts
SG validator	Two AWS IAM validation shell scripts	2	19 access, 2 secret	Shell scripts staging 21 AWS IAM key records for testing
SG validator	AWS CLI v2 and S3 discovery scripts	4	0	Unpacked AWS CLI v2 key-check and S3 enumeration scripts
SG validator	SMTP credential test scripts	3	0	Python SMTP credential and mail-relay test scripts
SG validator	Web-fuzzing JSONs (3) and report PDFs (47)	50	0	Web-fuzzing outputs and report PDFs (consistent with research use)

05

CHAPTER 05

What Secrets Harvesters Keep, and Other Explanations

IN SHORT

Version-control tokens, Stripe keys and AWS IAM keys make up 93.3% of the 193 workbench records, even though database URLs lead raw leaks. Dual-use bug-bounty and honeypot explanations remain plausible across these five hosts, which is why all host addresses are withheld. Hunt for exposed keys in your own web roots.

INSIDE THIS CHAPTER

Other ways to read the five workbenches 18

05 What Secrets Harvesters Keep, and Other Explanations

Comparing the 193 secret records inside the five workbenches against the 1,001 latest arrivals across all exposed files shows a sharp filter. Database connection strings lead raw arrivals (255 of 1,001, or 25.5%), yet they make up only 10 of the 193 workbench records (5.2%). Many database URLs inside leaked `.env` files point to `localhost` or private container networks that cannot be reached from outside.

Harvester dumps and validation scripts instead keep credentials intended for internet-facing cloud, code and payment APIs. Three categories account for 180 of the 193 workbench records (93.3%). They are version-control tokens (117, or 60.6%), Stripe payment keys (34, or 17.6%), and AWS IAM keys (29, or 15.0%).

The 108 `github_token` records on these workbenches represent 75.5% of all 143 `github_token` arrivals in the latest slice. The 34 `stripe_key` records represent 85.0% of all 40 `stripe_key` arrivals.

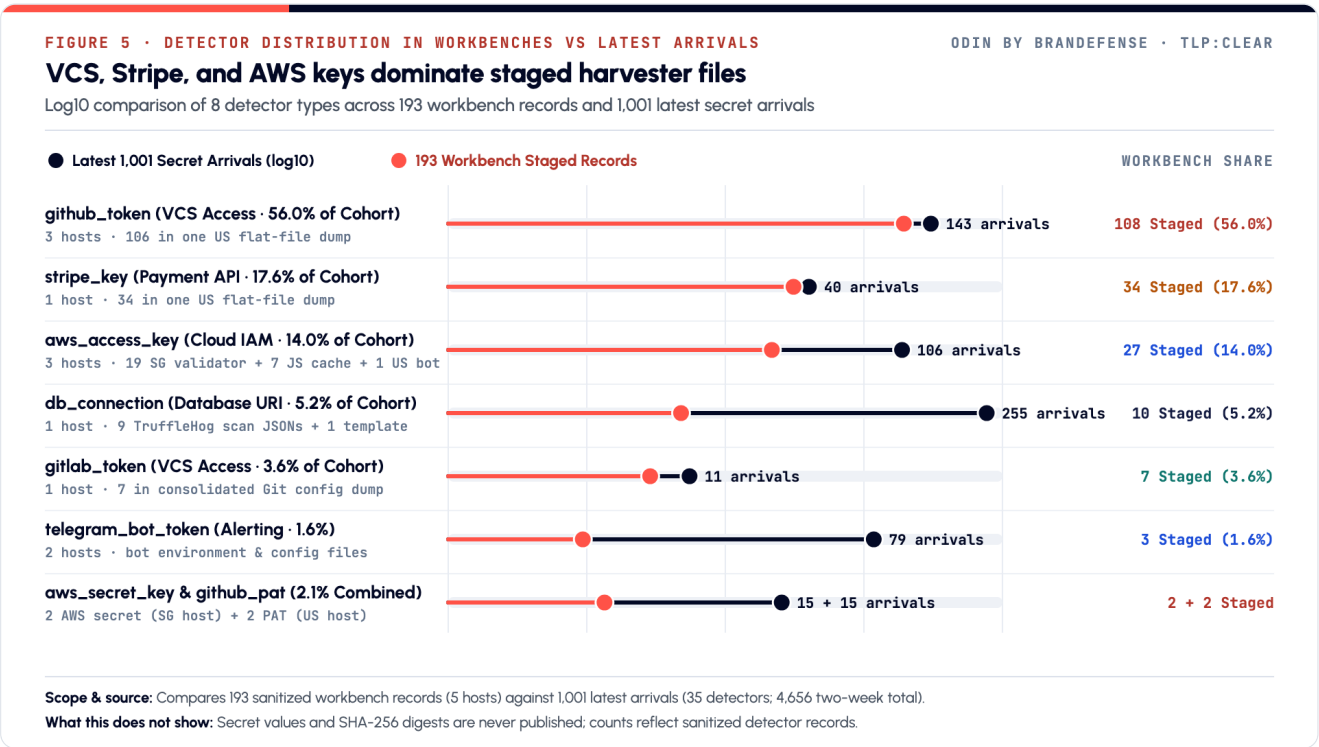


Figure 5: VCS, Stripe, and AWS keys dominate staged harvester files

Secret type	Category	Workbench records (193)	Workbench share	Hosts	Files	Latest arrivals (1,001)
`github_token`	Source control	108	55.96%	3	3	143
`stripe_key`	Payments	34	17.62%	1	1	40
`aws_access_key`	Cloud IAM	27	13.99%	3	4	106
`db_connection`	Database	10	5.18%	1	4	255
`gitlab_token`	Source control	7	3.63%	1	1	11
`telegram_bot_token`	Messaging / alert	3	1.55%	2	3	79
`aws_secret_key`	Cloud IAM	2	1.04%	1	1	15
`github_pat`	Source control	2	1.04%	1	1	15

Other ways to read the five workbenches

Open directories that hold scanners and harvested keys are not automatically criminal infrastructure. Three alternative readings matter before any SOC writes detection or blocking policy:

1. **Bug-bounty hunters or red-team operators.** On the Singapore validator server, the AWS check scripts sit alongside three `ffuf` web-fuzzing JSON files, redirect lists, and 47 vulnerability report PDFs. On the US TruffleHog server, organization scan JSONs sit near a red-team C2 configuration file. Either server could belong to a researcher who left an HTTP folder open, though exposed victim keys remain at risk.

1. **Honeypot or threat-research collectors.** On the US aggregator server, the GitHub and Stripe token files share the root folder with honeypot deployment archives, collection scripts, and host summary files. That mix fits either an attacker reusing honeypot scripts or a deception researcher exposing captured material.

1. **Unverified key liveness.** Because Odin replayed 0 credentials against AWS, GitHub, GitLab, Stripe or Telegram, we do not know how many of the 193 staged records were active versus already revoked.

06

CHAPTER 06

What Defenders Should Do

IN SHORT

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

06 What Defenders Should Do

Web and frontend engineering

- [ ] Block inbound HTTP access to `/.git/`, `.env*` and `.bak*` paths at reverse proxies, CDNs and WAFs. Deploy containers from clean build artifacts rather than `git clone` folders inside web roots.
- [ ] Audit frontend build variables (`NEXT_PUBLIC_*`, `VITE_*`, `REACT_APP_*`) so AWS IAM keys, database URLs and Telegram bot tokens never compile into public JavaScript bundles.

Source control and CI/CD

- [ ] Turn on GitHub and GitLab secret scanning and push protection across all repositories. Remove embedded `oauth2:` or PAT credentials from `.git/config` remote URLs on build runners.
- [ ] Replace classic long-lived GitHub personal access tokens with short-lived, fine-grained tokens scoped to named repositories, and require SAML SSO authorization.

Cloud IAM and payment APIs

- [ ] Retire static AWS IAM user keys (`AKIA*`) in CI/CD and developer workflows in favor of short-lived OIDC federated roles (`AssumeRoleWithWebIdentity`).
- [ ] Replace unrestricted Stripe secret keys with restricted keys limited to the exact API endpoints an application needs, and enforce source IP allowlists where supported.

SOC and attack-surface monitoring

- [ ] Alert on AWS CloudTrail `GetCallerIdentity` calls coming from AWS CLI, Boto3 or curl user agents on unfamiliar networks, and deploy the rules in Chapter 7.
- [ ] Scan your own cloud VPS and lab ranges for unauthenticated `Index of /` HTTP listeners (`python3 -m http.server`) so internal test workbenches never expose scan outputs or keys.

07

CHAPTER 07

Detection Rules

IN SHORT

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

INSIDE THIS CHAPTER

How the Suricata signatures work 22

How the KQL, Sigma and YARA rules work 23

MITRE ATT&CK mapping 23

07 Detection Rules

The full rules are in Appendix B. Static syntax checks passed, including `lint_suricata.py` with zero problems. Suricata, YARA, Sigma, and Kusto engines were unavailable locally, and the rules were not replayed against live traffic. Test and tune them in your own environment before production enforcement.

Rule file	What it catches
`odin_automated_secret_harvesters_and_cloud_validators.rules`	Suricata 7: eight signatures for `/.git/config` and `/.env` probes, exposed harvester directory listings, TruffleHog JSON transfers, `.git` metadata requests, batch AWS validator scripts, STS HTTP queries, and server TLS sessions to `api.telegram[.]org`
`sentinel_secret_harvester_and_cloud_key_validation_hunt.kql`	Microsoft Sentinel and Defender XDR: unapproved `trufflehog`, `/.git/config` or `get-caller-identity` processes joined by initiating process identity to connections in the next 15 minutes for AWS STS, GitHub, GitLab, Stripe or Telegram API egress
`sigma_trufflehog_harvester_and_aws_iam_key_validation.yml`	Sigma (3 rules): CloudTrail `GetCallerIdentity` calls from CLI or SDK user agents, Linux harvester or validator command lines, and web requests for `/.git/config`, `/.git/credentials` or `/.env`
`odin_secret_harvester_and_cloud_validator_artifacts.yar`	YARA (4 rules): secret-scanning and Telegram alert bots, TruffleHog JSON scan outputs, batch AWS IAM validator scripts, and consolidated `.git/config` or token dumps

How the Suricata signatures work

SID	Looks for	Why
9538001	Inbound `GET` with `/.git/config` in the URI	Catches automated `.git/config` token harvesters probing web roots (block candidate)
9538002	Outbound 200 response with `Index of /` and harvester markers	Flags an internal server accidentally exposing TruffleHog, AWS CLI or `.git/config` artifacts
9538003	HTTP `GET` request for TruffleHog `.json` scan outputs	Catches retrieval of exposed TruffleHog repository scan JSON files
9538004	Inbound `GET` request for `/.env` environment files	Catches automated harvesters probing web roots for environment secrets
9538005	Inbound `GET` request for `/.git/HEAD`, `/.git/credentials` or `/.git/index`	Catches repository-dump tools pulling exposed Git metadata
9538006	HTTP file transfer containing `AWS_SECRET_ACCESS_KEY` and `GetCallerIdentity`	Catches retrieval of staged cloud IAM validation scripts
9538007	HTTP request URI containing `Action=GetCallerIdentity`	Flags unencrypted or proxied AWS STS identity check requests
9538008	Outbound TLS SNI `api.telegram[.]org` from server segments	Hunt only: flags server-tier processes contacting the Telegram Bot API

Enforce SID 9538001 in inline block mode at web reverse proxies. Keep SID 9538008 scoped to production server and CI subnets rather than staff workstation networks.

How the KQL, Sigma and YARA rules work

The Sentinel KQL query looks back seven days across `DeviceProcessEvents` and `DeviceNetworkEvents` for unapproved secret-scanning or cloud-validation command lines. It joins each process to outbound API connections using the initiating process identity. The corrected query tests connections in the next 15 minutes to AWS STS, GitHub, GitLab, Stripe or Telegram.

Correlated matches receive `REVIEW_UNAPPROVED_HARVESTER` or `INVESTIGATE_BULK_KEY_VALIDATION` outside two exact allowlisted scanner hosts. Process-only matches receive `HUNT_UNAPPROVED_SECRET_SCANNER` without qualifying API egress.

Sigma rule 1 reads AWS CloudTrail logs and flags `sts.amazonaws[.]com` `GetCallerIdentity` calls from `aws-cli/`, `Boto3/`, `botocore/`, `curl/` or `python-requests/` user agents outside two exact CI role ARNs. Sigma rule 2 flags Linux process creation for secret-scanning and cloud-validation command lines outside the two exact scanner hostnames. Sigma rule 3 flags web-server access logs requesting `/.git/config`, `/.git/HEAD`, `/.git/credentials` or `/.env`.

YARA rule	Fires when
`ODIN_Keyleak_Secret_Scanner_And_Alert_Bot`	Under 512 KB, two secret-scanner or token markers appear alongside `api.telegram[.]org/bot`, `api.github[.]com` or `GetCallerIdentity`
`ODIN_TruffleHog_Org_Secret_Scan_Output`	Under 10 MB, three TruffleHog JSON schema keys (`SourceMetadata`, `DetectorName`, `DetectorType`, `Verified`) appear with a database or cloud detector name
`ODIN_AWS_IAM_Batch_Credential_Validator_Script`	Under 256 KB, `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` appear with `get-caller-identity` or `aws s3 ls` and a shell or Boto3 loop
`ODIN_JS_Bundle_And_GitConfig_Token_Aggregator`	Under 5 MB, `[remote "origin"]` and a Git remote `url =` line appear with GitHub or GitLab domains, or two token-aggregator and Telegram Bot API markers appear

MITRE ATT&CK mapping

Tactic	Technique	How the pipeline uses it	Detection
Credential Access	T1552.001 Unsecured Credentials: Credentials In Files	Scraping `/.git/config`, JavaScript bundles, `.env` files and TruffleHog JSON outputs	SIDs 9538001 to 9538004; Sigma rules 2 and 3; YARA rules 2 and 4
Credential Access	T1528 Steal Application Access Token	Aggregating 108 GitHub tokens, 2 GitHub PATs, 7 GitLab tokens and 34 Stripe keys	No flat-file transfer rule supplied; SID 9538005 covers Git metadata probes, while KQL and Sigma rule 2 provide process context
Defense Evasion, Persistence	T1078.004 Valid Accounts: Cloud Accounts	Staging 27 AWS access keys and 2 secret keys inside shell validation scripts	SIDs 9538006 and 9538007; KQL; Sigma rule 1; YARA rule 3
Discovery	T1580 Cloud Infrastructure Discovery	Batch `GetCallerIdentity` and S3 enumeration wrappers	KQL; Sigma rule 1; YARA rule 3
Exfiltration	T1567 Exfiltration Over Web Service	Potential Telegram-based exfiltration configured in bot files; no transmission observed	SID 9538008; KQL; YARA rule 1

CHAPTER 08

Indicators

IN SHORT

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

INSIDE THIS CHAPTER

Behavioral indicators in this report 25

08 Indicators

Appendix A holds 0 published indicators: 0 network addresses and 0 file hashes. All five open-directory host addresses, ports, server credential-file paths, and file digests are withheld to protect victim credentials and account for dual-use security-research hosts. Two behavioral URI and cloud API hunt indicators appear in this report only.

Type	Count	Where to use it
Network addresses and ports (Appendix A)	0	All five host addresses and ports are withheld
File hashes and server file paths (Appendix A)	0	All credential-bearing paths, archives and file hashes are withheld
Inbound web probe URI (in text)	1	Block at the WAF and reverse proxy
Cloud API action pattern (in text)	1	Hunt caller context in AWS CloudTrail; never block the domain

Behavioral indicators in this report

Indicator	Type	Pipeline stage	Use
`/.git/config`	Web probe URI	Stage 1: Git collection	Block inbound HTTP requests at the WAF
`sts.amazonaws[.]com/?Action=GetCallerIdentity`	Cloud API pattern	Stage 4: Cloud check	Watch unusual callers in CloudTrail; do not block

`sts.amazonaws[.]com` and `api.telegram[.]org` are legitimate public services. Never block `sts.amazonaws[.]com` at the firewall; inspect CloudTrail caller identities, source ASNs and user-agent strings instead.

Withheld values include all five server IP addresses and ports, all server credential-file and archive paths, and all 118 external callback source IPs. Target organization names, third-party names in co-hosted files, and all SHA-256 digests of secret-bearing or unestablished files are also withheld.

09

CHAPTER 09

Scope and Limits

IN SHORT

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

09 Scope and Limits

- **Window.** All observations come from a two-week observation window ending 20 November 2026.
- **Static review, no key testing.** File metadata and detector counts were reviewed without running scripts, replaying any of the 193 secret records, or contacting target systems. Live-key ratios are unmeasured.
- **Five stages are an analytical model.** The five servers share 0 IP addresses, certificates, keys or bot tokens. Grouping them into five stages maps controls across the credential lifecycle rather than naming one actor.
- **Dual-use research and honeypots remain plausible.** Fuzzing files and report PDFs on the Singapore host and honeypot archives on the US aggregator fit research use, so all identifiers are withheld.
- **Server file dates vs observation window.** June and July 2026 date strings (20260616 to 20260724) inside callback log filenames on the US TruffleHog host are server file timestamps, not Odin's monitoring window.
- **Rule testing.** Static checks (`lint_suricata.py` with 0 problems, plus syntax and fixture checks) passed on 21 November 2026. Suricata, YARA, Sigma and Kusto engines were not installed locally, so test and tune rules before production use.

10

CHAPTER 10

Sources

IN SHORT

This chapter lists the public references used in the report.

10 Sources

1. MITRE ATT&CK. T1552.001 Unsecured Credentials: Credentials In Files. https://attack.mitre.org/techniques/T1552/001/ (verified 20 November 2026)
2. MITRE ATT&CK. T1528 Steal Application Access Token. https://attack.mitre.org/techniques/T1528/ (verified 20 November 2026)
3. MITRE ATT&CK. T1078.004 Valid Accounts: Cloud Accounts. https://attack.mitre.org/techniques/T1078/004/ (verified 20 November 2026)
4. MITRE ATT&CK. T1580 Cloud Infrastructure Discovery. https://attack.mitre.org/techniques/T1580/ (verified 20 November 2026)
5. Amazon Web Services. AWS Security Token Service API Reference: GetCallerIdentity. https://docs.aws.amazon.com/STS/latest/APIReference/API_GetCallerIdentity.html (verified 20 November 2026)
6. Amazon Web Services. AWS Identity and Access Management User Guide: Security best practices in IAM. https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html (verified 20 November 2026)
7. GitHub Docs. Code Security: About secret scanning. https://docs.github.com/en/code-security/secret-scanning/introduction/about-secret-scanning (verified 20 November 2026)
8. Stripe Documentation. API keys. https://docs.stripe.com/keys (verified 20 November 2026)

APPENDICES

Indicators and Detection Rules

FOR DEFENDERS

No individual indicators are published for this study; Appendix A explains why. Appendix B carries every published detection rule in full.

INSIDE

A. Indicators of compromise (0) 31

B. Detection rules (4 files) 32

A Indicators of Compromise

No individual addresses or file hashes are published for this study. The hosts in this report are exposed or unattributed systems rather than confirmed adversary infrastructure, so they stay anonymous. Use the detection rules in Appendix B.

B Detection Rules

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

odin_automated_secret_harvesters_and_cloud_validators.rules

SURICATA

```
# Odin by Brandefense - Monograph 38 / 46 Suricata 7.x Detection Suite
# Scope: Automated Git/JS secret harvesters, TruffleHog scan transfers, environment probes, and AWS IAM
validators
# Validation: Test and tune in staging before inline enforcement.

alert http any any -> $HOME_NET any (msg:"ODIN R38 Exposed Git Repository Config Probe (/.git/config)";
flow:established,to_server; http.method; content:"GET"; bsize:3; http.uri; content:"/.git/config"; bsize:>=12;
fast_pattern; classtype:web-application-attack; sid:9538001; rev:1; metadata:policy security-ips,
mitre_technique_id T1552.001;)
alert http $HOME_NET any -> any any (msg:"ODIN R38 Open Directory Listing Exposing Secret Harvester Artifacts";
flow:established,to_client; http.stat_code; content:"200"; bsize:3; file.data; content:"Index of /";
fast_pattern; pcre:"/(trufflehog[a-z0-9_.-]*\.json|awscliv2\.zip|GetCallerIdentity|\.git\/config)/i";
classtype:policy-violation; sid:9538002; rev:1; metadata:policy behavioral_hunt, mitre_technique_id T1552.001;)
alert http any any -> any any (msg:"ODIN R38 HTTP Retrieval of TruffleHog JSON Scan Output";
flow:established,to_server; http.method; content:"GET"; bsize:3; http.uri; content:"trufflehog"; nocase;
fast_pattern; content:".json"; nocase; classtype:credential-theft; sid:9538003; rev:1; metadata:policy
behavioral_hunt, mitre_technique_id T1552.001;)
alert http any any -> $HOME_NET any (msg:"ODIN R38 Inbound HTTP Environment Secret File Probe (/.env)";
flow:established,to_server; http.method; content:"GET"; bsize:3; http.uri; content:"/.env"; bsize:>=5;
fast_pattern; classtype:web-application-attack; sid:9538004; rev:1; metadata:policy behavioral_hunt,
mitre_technique_id T1552.001;)
alert http any any -> $HOME_NET any (msg:"ODIN R38 Inbound HTTP Git Metadata Or Credential Store Probe";
flow:established,to_server; http.method; content:"GET"; bsize:3; http.uri; content:"/.git/"; bsize:>=10;
fast_pattern; pcre:"/\/\.git\/(HEAD|credentials|index|logs\/HEAD)$/i"; classtype:credential-theft; sid:9538005;
rev:1; metadata:policy behavioral_hunt, mitre_technique_id T1528;)
alert http any any -> any any (msg:"ODIN R38 HTTP Transfer of Batch AWS IAM Credential Validation Script";
flow:established,to_client; http.stat_code; content:"200"; bsize:3; file.data; content:"AWS_SECRET_ACCESS_KEY";
fast_pattern; content:"GetCallerIdentity"; nocase; classtype:credential-theft; sid:9538006; rev:1;
metadata:policy behavioral_hunt, mitre_technique_id T1078.004;)
alert http any any -> any any (msg:"ODIN R38 HTTP Request Carrying AWS STS GetCallerIdentity Action";
flow:established,to_server; http.uri; content:"Action=GetCallerIdentity"; bsize:>=24; fast_pattern;
classtype:trojan-activity; sid:9538007; rev:1; metadata:policy behavioral_hunt, mitre_technique_id T1078.004;)
alert tls $HOME_NET any -> $EXTERNAL_NET 443 (msg:"ODIN R38 HUNT Outbound Telegram API TLS Session From Server
Segment"; flow:established,to_server; tls.sni; content:"api.telegram.org"; bsize:16; fast_pattern;
classtype:policy-violation; sid:9538008; rev:1; metadata:policy behavioral_hunt, mitre_technique_id T1567;)
```

odin_secret_harvester_and_cloud_validator_artifacts.yar

YARA

```
/*
   Odin by Brandefense - Monograph 38 / 46 Static YARA Detection Rules
   Scope: Automated secret-harvesting bots, TruffleHog scan outputs, AWS IAM batch validators, and Git/JS token
aggregators
*/

rule ODIN_Keyleak_Secret_Scanner_And_Alert_Bot
{
    meta:
        author = "Odin by Brandefense"
        monograph = "38 / 46"
        date = "2026-11-20"
        mitre_attack = "T1552.001, T1528, T1567"
        description = "Matches Python/TOML secret-scanning and alerting bot artifacts paired with cloud or
messaging API endpoints"
    strings:
        $k1 = "DetectorName" ascii
        $k2 = "AWS_ACCESS_KEY_ID" ascii
```

```
        $k3 = "github_pat_" ascii
        $k4 = "sk_live_" ascii
        $api1 = "api.telegram.org/bot" ascii
        $api2 = "api.github.com" ascii
        $api3 = "GetCallerIdentity" ascii
    condition:
        filesize < 512KB and (2 of ($k*)) and (1 of ($api*))
}

rule ODIN_TruffleHog_Org_Secret_Scan_Output
{
    meta:
        author = "Odin by Brandefense"
        monograph = "38 / 46"
        date = "2026-11-20"
        mitre_attack = "T1552.001"
        description = "Matches structured TruffleHog JSON scan outputs containing verified or unverified
credential records"
    strings:
        $th1 = "\"SourceMetadata\"" ascii
        $th2 = "\"DetectorName\"" ascii
        $th3 = "\"DetectorType\"" ascii
        $th4 = "\"Verified\"" ascii
        $det1 = "Postgres" ascii nocase
        $det2 = "MySQL" ascii nocase
        $det3 = "MongoDB" ascii nocase
        $det4 = "Github" ascii nocase
        $det5 = "AWS" ascii nocase
    condition:
        filesize < 10MB and (3 of ($th*)) and (1 of ($det*))
}

rule ODIN_AWS_IAM_Batch_Credential_Validator_Script
{
    meta:
        author = "Odin by Brandefense"
        monograph = "38 / 46"
        date = "2026-11-20"
        mitre_attack = "T1078.004, T1580"
        description = "Matches shell or Python scripts iterating over harvested AWS IAM access keys to test
liveness via STS or S3"
    strings:
        $env1 = "AWS_ACCESS_KEY_ID" ascii
        $env2 = "AWS_SECRET_ACCESS_KEY" ascii
        $call1 = "get-caller-identity" ascii
        $call2 = "GetCallerIdentity" ascii
        $call3 = "aws s3 ls" ascii
        $loop1 = "while IFS=" ascii
        $loop2 = "for key in" ascii
        $loop3 = "boto3.client" ascii
    condition:
        filesize < 256KB and $env1 and $env2 and (1 of ($call*)) and (1 of ($loop*))
}

rule ODIN_JS_Bundle_And_GitConfig_Token_Aggregator
{
    meta:
        author = "Odin by Brandefense"
        monograph = "38 / 46"
        date = "2026-11-20"
        mitre_attack = "T1552.001, T1528"
        description = "Matches consolidated .git/config remote credential dumps or JavaScript bundle scraper
caches"
    strings:
        $git1 = "[remote \"origin\"]" ascii
        $git2 = "url = https://" ascii
        $git3 = "gitlab.com" ascii nocase
        $git4 = "github.com" ascii nocase
        $js1 = "oauth2:" ascii
```

```
        $js2 = "AWS_ACCESS_KEY_ID" ascii
        $js3 = "api.telegram.org/bot" ascii
    condition:
        filesize < 5MB and (($git1 and $git2 and ($git3 or $git4)) or (2 of ($js*)))
}
```

sentinel_secret_harvester_and_cloud_key_validation_hunt.kql

KQL (SENTINEL / DEFENDER XDR)

```
// Odin by Brandefense - Monograph 38 / 46 Microsoft Sentinel & Defender XDR Hunt Query
// Purpose: Correlate unapproved secret-harvesting or batch cloud key validation processes with API/Telegram
egress
let Lookback = 7d;
let ApprovedScannerHosts = dynamic([
    "sec-trufflehog-runner-01.corp.example",
    "ci-secret-scan-01.corp.example"
]);
let HarvesterProcesses =
    DeviceProcessEvents
    | where Timestamp >= ago(Lookback)
    | where not(DeviceName in~ (ApprovedScannerHosts))
    | where FileName in~ ("python", "python3", "bash", "sh", "trufflehog", "aws", "curl")
    | where ProcessCommandLine has_any (
        "trufflehog",
        "/.git/config",
        "get-caller-identity",
        "GetCallerIdentity",
        "AWS_SECRET_ACCESS_KEY",
        "sts.amazonaws.com",
        "api.stripe.com",
        "api.telegram.org/bot"
    )
    | extend ProcSignal = case(
        ProcessCommandLine has_any ("/.git/config", "api.telegram.org/bot", "api.stripe.com"),
"HarvesterOrAggregator",
        ProcessCommandLine has_any ("get-caller-identity", "GetCallerIdentity", "AWS_SECRET_ACCESS_KEY",
"sts.amazonaws.com"), "CredentialValidator",
        "SecretScannerRun"
    )
    | project ProcTime = Timestamp, DeviceId, DeviceName, AccountName, ProcessId, FileName, ProcessCommandLine,
ProcSignal;
let ValidationEgress =
    DeviceNetworkEvents
    | where Timestamp >= ago(Lookback)
    | where not(DeviceName in~ (ApprovedScannerHosts))
    | where RemoteUrl has_any (
        "sts.amazonaws.com",
        "api.github.com",
        "gitlab.com",
        "api.stripe.com",
        "api.telegram.org"
    )
    | project
        NetTime = Timestamp,
        DeviceId,
        DeviceName,
        ProcessId = InitiatingProcessId,
        FileName = InitiatingProcessFileName,
        RemoteUrl,
        InitiatingProcessFileName;
let QualifyingEgress =
    HarvesterProcesses
    | join kind=inner ValidationEgress on DeviceId, DeviceName, ProcessId, FileName
    | where NetTime between (ProcTime .. ProcTime + 15m)
    | summarize
        EgressConnections = count(),
        DistinctEndpoints = make_set(RemoteUrl, 10),
        InitiatingBins = make_set(InitiatingProcessFileName, 10),
        LastEgressTime = max(NetTime)
```

```
      by ProcTime, DeviceId, DeviceName, ProcessId, FileName, ProcessCommandLine;
HarvesterProcesses
| join kind=leftouter QualifyingEgress on ProcTime, DeviceId, DeviceName, ProcessId, FileName, ProcessCommandLine
| extend EgressConnections = coalesce(EgressConnections, 0)
| extend ResponsePriority = case(
    ProcSignal == "HarvesterOrAggregator" and EgressConnections > 0, "REVIEW_UNAPPROVED_HARVESTER",
    ProcSignal == "CredentialValidator" and EgressConnections >= 5, "INVESTIGATE_BULK_KEY_VALIDATION",
    "HUNT_UNAPPROVED_SECRET_SCANNER"
)
| project
    ProcTime,
    LastEgressTime,
    DeviceName,
    AccountName,
    ProcessId,
    FileName,
    ProcSignal,
    EgressConnections,
    DistinctEndpoints,
    InitiatingBins,
    ResponsePriority,
    ProcessCommandLine
| sort by EgressConnections desc, ProcTime desc
```

sigma_trufflehog_harvester_and_aws_iam_key_validation.yml

SIGMA

```
title: AWS CloudTrail STS GetCallerIdentity Validation From CLI Or SDK Client
id: 6d41c289-42b0-48f3-891c-38a01b953801
status: experimental
description: Detects AWS STS GetCallerIdentity calls generated by AWS CLI, Python Boto3, or curl clients outside
approved CI/CD principals, matching batch IAM credential liveness checks used by automated shell and Python
credential validation scripts.
author: Odin by Brandefense
date: 2026-11-20
logsource:
    product: aws
    service: cloudtrail
detection:
    selection_sts:
        eventSource: 'sts.amazonaws.com'
        eventName: 'GetCallerIdentity'
    selection_ua:
        userAgent|contains:
            - 'aws-cli/'
            - 'Boto3/'
            - 'botocore/'
            - 'curl/'
            - 'python-requests/'
    filter_approved_ci:
        userIdentity.arn:
            - 'arn:aws:iam::000000000000:role/ci-deployment-runner'
            - 'arn:aws:iam::000000000000:role/security-audit-scanner'
    condition: selection_sts and selection_ua and not 1 of filter_*
falsepositives:
    - Interactive developer workstation CLI sessions outside federated SSO roles
level: medium
tags:
    - attack.credential-access
    - attack.t1552.001
    - attack.defense-evasion
    - attack.t1078.004
    - attack.discovery
    - attack.t1580
---
title: Linux Secret-Harvesting Bot And Credential Validator Script Execution
id: 9b17e432-5f18-4d61-9a20-38a02b953802
status: experimental
description: Detects Linux command lines invoking secret-harvesting bots, TruffleHog repository scans, or AWS IAM
```

```
batch validation scripts outside approved scanner hosts.
author: Odin by Brandefense
date: 2026-11-20
logsource:
    product: linux
    category: process_creation
detection:
    selection_bin:
        Image|endswith:
            - '/python'
            - '/python3'
            - '/bash'
            - '/sh'
            - '/trufflehog'
    selection_args:
        CommandLine|contains:
            - 'trufflehog '
            - '/.git/config'
            - 'get-caller-identity'
            - 'GetCallerIdentity'
            - 'AWS_SECRET_ACCESS_KEY='
            - 'api.telegram.org/bot'
            - 'api.stripe.com/v1/'
    filter_approved_hosts:
        ComputerName:
            - 'sec-trufflehog-runner-01.corp.example'
            - 'ci-secret-scan-01.corp.example'
    condition: selection_bin and selection_args and not 1 of filter_*
falsepositives:
    - Authorized internal secret-scanning appliances not listed in filter_approved_hosts
level: high
tags:
    - attack.credential-access
    - attack.t1552.001
    - attack.t1528
    - attack.exfiltration
    - attack.t1567
---
title: Web Server Access To Exposed Git Config Or Environment Credential Files
id: 3f82a904-7c4d-41e9-b811-38a03b953803
status: experimental
description: Detects inbound HTTP requests targeting exposed .git/config, .git/credentials, or .env credential
files on web servers.
author: Odin by Brandefense
date: 2026-11-20
logsource:
    category: webserver
detection:
    selection_uri:
        cs-uri-stem|contains:
            - '/.git/config'
            - '/.git/HEAD'
            - '/.git/credentials'
            - '/.env'
    condition: selection_uri
falsepositives:
    - Authorized vulnerability scanners testing web roots for exposed VCS directories
level: high
tags:
    - attack.credential-access
    - attack.t1552.001
    - attack.t1528
```

ODIN

About Odin Research

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

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

Online edition, indicators and rules · brandefense.io/odin-research/reports/secret-harvesting-pipelines/
Short Read · brandefense.io/odin-research/blog/inside-automated-secret-harvesting-and-cloud-key-validation-pipelines/
Defense Kit · brandefense.io/odin-research/defense-kit/