Recon Village | DEF CON 34

The Breach Is Over.
The Exposure Is Not.

Anant Shrivastava
Cyfinoid Research

Kumar Ashwin
RedHuntLabs

Anant Shrivastava

Founder, Cyfinoid Research

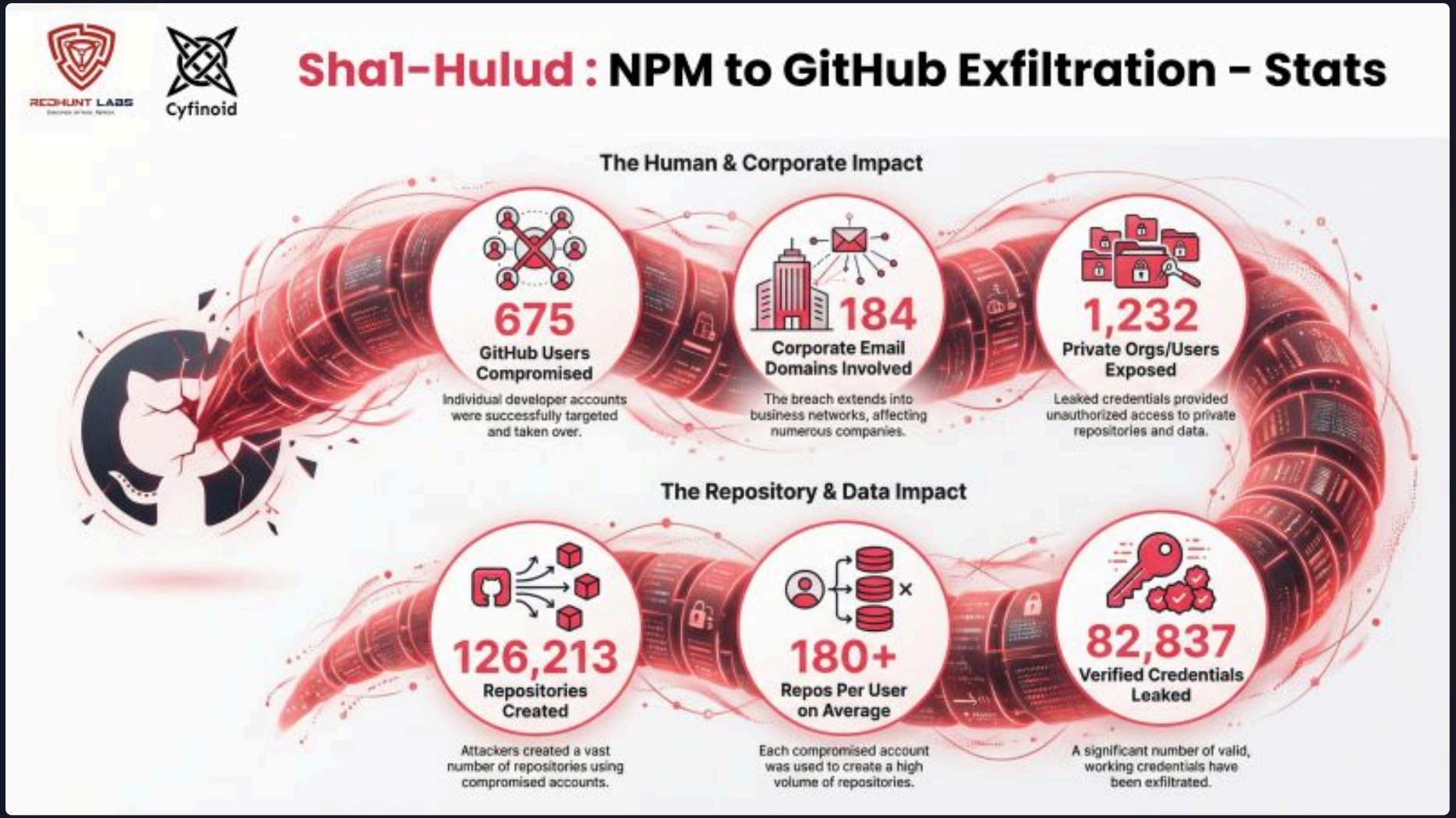
- Indian
- 18+ years in infosec
- Ops / Dev / Sec
- Open projects: CodeVigilant and Hacking Archives of India
- anantshri.info

Kumar Ashwin

Researcher, RedHuntLabs

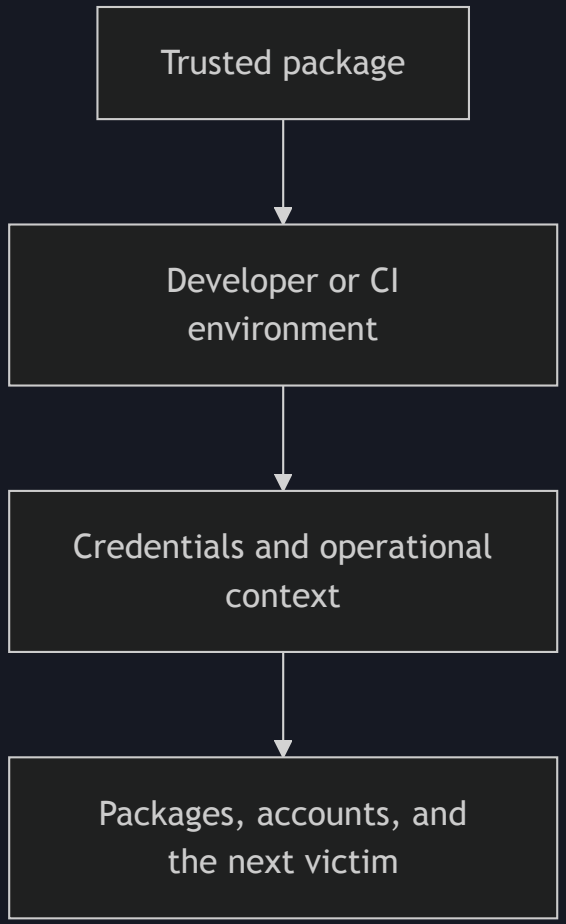
- Indian
- 5+ years in infosec
- AI, blockchain, and software security
- Open Projects : actsense, cot.run and more
- kumarashwin.com

Shai-Hulud: the fun begins



So, what do we know?

One breach fed the next



Shai Hulud 1: steal, publish, repeat

- Enter through compromised npm maintainer access
- Execute through a malicious package lifecycle script
- Harvest environment, cloud, GitHub, and npm credentials
- Publish stolen material to a public GitHub repository
- Use npm publishing rights to infect more packages

One victim became the next distribution channel

Shai Hulud 2: now the spill stopped matching the victim

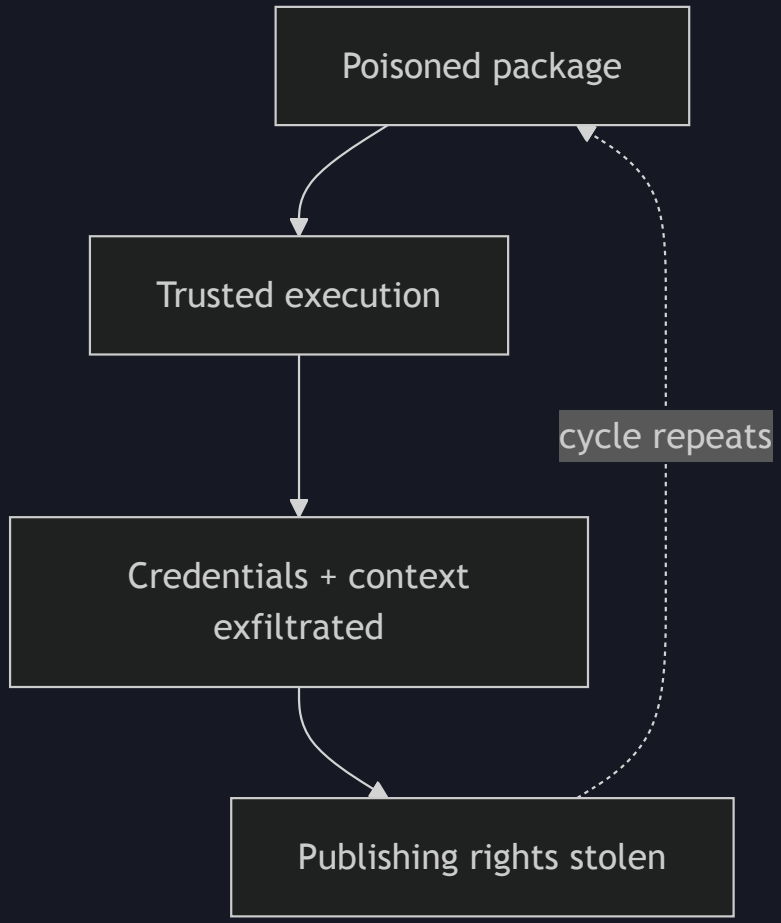
- Execute earlier during package installation
- Increase automation, propagation speed, and target breadth
- Harvest developer, CI/CD, and cloud-connected environments
- Reuse an available prior-victim account to host new stolen data
  - Mix one victim's secrets into another account's repositories

The host account stopped being a reliable victim label

Shai Hulud 1 vs Shai Hulud 2: what actually changed?

	Shai Hulud 1	Shai Hulud 2
Primary spread	Stolen npm publishing rights	Faster, broader automated propagation
Execution	Package lifecycle execution	Earlier preinstall execution
Exfiltration	Public repo under compromised account	Local or prior-victim account
Attribution	Difficult	Cross-account and chaotic
Aftermath	Leaked credentials	Mixed, cross-account exposure corpus

What can go wrong?



The worm got neutered

And everyone lived happily after

Why now?

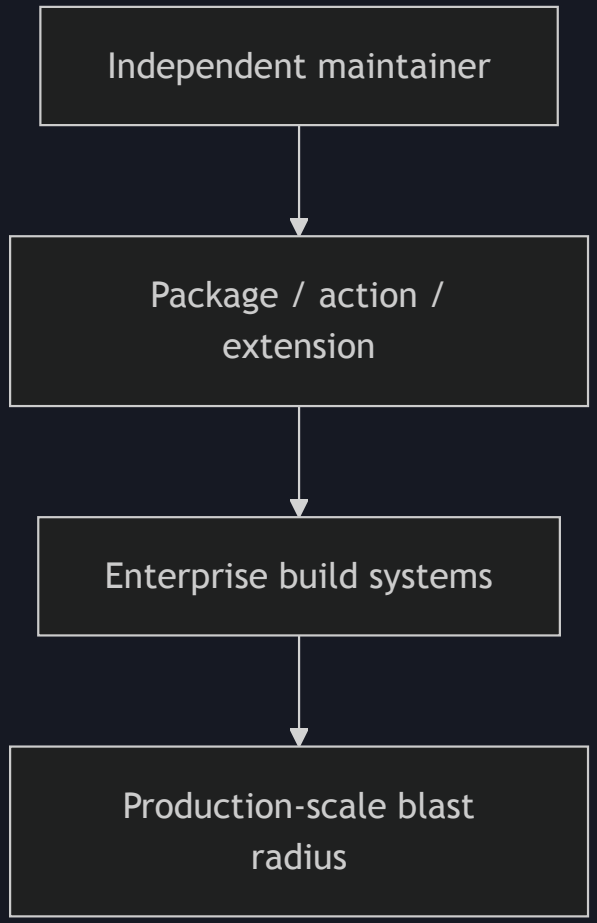
- Developer identity spans source, packages, CI/CD, cloud, and deployment
- Build systems sit at the intersection of multiple trust domains
- One publishing capability can affect thousands of downstream consumers
- Credentials remain reusable after the malicious package is removed
- Context makes unfamiliar secrets and future targets easier to identify

A leaked secret is a route into another trust system

Protection stops at the account boundary

- **Public repositories**
  - Automatic scanning for supported provider-pattern secrets
- **Generic private keys**
  - Detection requires an organization-owned repository with Team and Secret Protection enabled
- **Even when detected**
  - Generic private keys do not support validity checks

Your org chart is not your attack surface



The maintainer is outside your org chart,

not outside your attack surface.

Trust is offloaded, protection is limited

- We moved trust outside the boundary
- We left protection inside it.
- Enterprise software increasingly trusts personal identities with production-scale blast radius.
- The contributor is not the weakest link.
- The mismatch between trust and protection is.

So, what do we know from Shai Hulud 1?

Finding	Count
Repositories captured	**35**
GitHub tokens identified	**30**
GitHub tokens still active	**23**

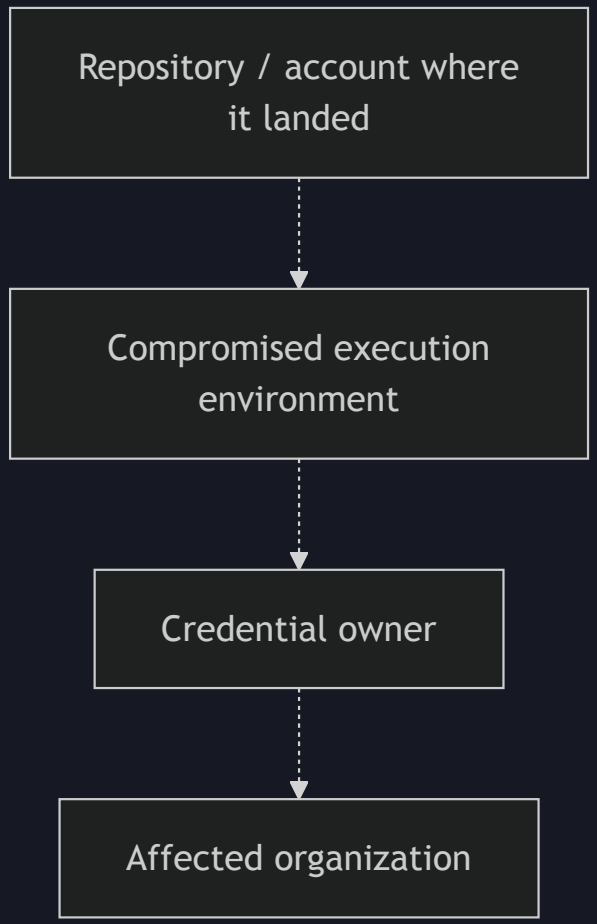
23 of 30 identified GitHub tokens remain active

Repository gone. Problem solved?

- Im almost all the cases there is still a private repo with Shai-hulud name
- Removing or hiding the public artifact reduces visibility.
- It does not revoke the credentials inside it.

Shai Hulud 2 made attribution messy

- General flow of reporting



- In Shai-hulud, The repository tells us where the secret landed, not where it came from.

Passwords and their strength

- The numbers (56,820 password occurrences → 2,516 unique)
- Strength (zxcvbn-scored):

Score	Count	Note
0 very-weak	32	
1 weak	71	4.1% weak/very-weak
2 fair	97	
3 good	118	
4 strong	2,198	~87% — mostly machine-generated

So why is the exposure still alive?

Detection gap

The scanner does not recognize the artifact.

Ownership gap

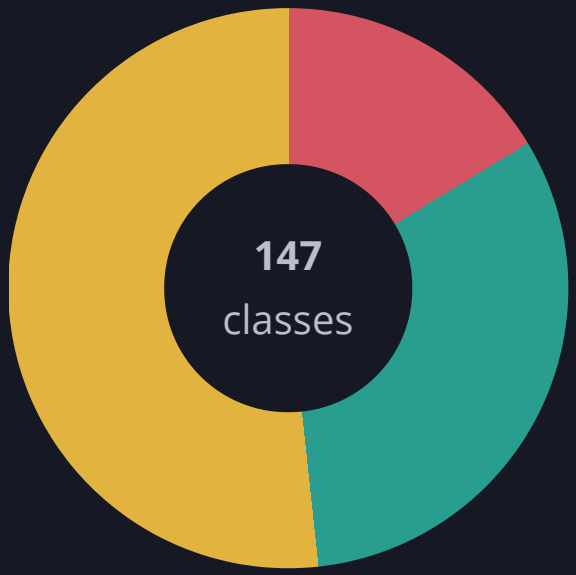
Nobody can route the credential to the team that can revoke it.

Provenance gap

The visible repository points to the wrong organization.

Most secret classes live in the long tail

Combined Shai Hulud 1 + Shai Hulud 2 corpus | August 2026



51.7% of classes have 99 or fewer findings

The long tail contributes only **1.39%** of findings

The Joy of Research and the flaws

Shai Hulud 2 only | 35,000 repositories

- We started looking at SSH Keys

Raw extractor output — **538,009**
files in `ssh_keys/`

Unique by exact bytes — **5,240**
532,769 identical copies removed

Unique by fingerprint — **5,171**
distinct SSH keys

- We were working in a folder with multiple copies of same data

The corrected SSH-key corpus mapped to real accounts

Shai Hulud 2 only | quick KeyChecker pass

5,171
fingerprint-unique SSH keys checked

0
password-protected keys

48
GitHub deployment keys

110
known unique accounts mapped

GitHub accounts
96

GitLab accounts
12

Hugging Face accounts
2

Bitbucket instances
23

Bitbucket acceptances do not disclose username, so account uniqueness is unknown.

One of the 35,000 repositories was ours

Controlled Shai Hulud 2 experiment

1 of 35,000
Our repository

→

Harmless AWS credential
Canarytoken

→

Every attempted call
Phones home

378
API interactions

59
source IPs

20
infrastructure countries

137
days observed

Courtesy: Canarytokens.org.

So, what happened first?

2025-11-26 | UTC

07:44:46	`GetCallerIdentity`	Does it work, and who am I?
+23 sec	`DescribeInstances`	What compute exists?
+24 sec	`ListFunctions20150331`	What can execute?
+102 min	`ListBuckets`	What data is visible?

41 / 59

source IPs opened with `GetCallerIdentity`

127 of 378 interactions asked the same question

> **First validate identity. Then inventory the account.**

The questions changed

Day 0

Who am I?

`GetCallerIdentity`

Identity and inventory

Day 3.6

Which models exist?

`ListFoundationModels`

Bedrock discovery

Day 17.9

Where are more secrets?

`ListSecrets`

Credential chaining

Day 62.1

Can I use a model?

`InvokeModel`

First invocation wave

Day 136.8

What else unlocks?

`DescribeParameters`

Secrets + SSM sweep

Different sources, same objectives

31 model discovery / invocation calls

37 Secrets Manager / SSM discovery calls

> Test the identity. Map what it can reach. Try what it can unlock.

> **Caveat:** Different sources converged on these objectives. This is not one actor changing strategy.

What did they try first?

47 AWS API operations | 378 interactions

Identity, IAM & secrets	
GetCallerIdentity	127
GetAccount	17
ListSecrets	19
DescribeParameters	18
ListUsers	4
ListRoles	4
ListUserPolicies	3
GetUser	2
ListGroups	2
GetAccountSummary	2
ListAttachedUserPolicies	2
DescribeOrganization	1
ListAccessKeys	1
ListPolicies	1

Compute & images	
DescribeInstances	26
DescribeImages	9
DescribeSnapshots	8
DescribeLaunchTemplates	3
DescribeHosts	3
DescribeInstanceStatus	1
RunInstances	1
TerminateInstances	1

What did they map next?

Same 47-operation inventory | no filtering to a top ten

Network	
DescribeSecurityGroups	4
DescribeAddresses	3
DescribeCustomerGateways	3
DescribeNetworkAcls	3
DescribeNatGateways	3
DescribeRegions	3
DescribeVpcEndpoints	2
DescribeNetworkInterfaces	2
DescribeRouteTables	2
DescribeSubnets	2
DescribeVpcs	2

Data, platform & spend	
ListBuckets	29
InvokeModel	21
ListFunctions20150331	16
ListFoundationModels	9
ListObjects	5
GetSendQuota	3
DescribeDBInstances	2
DescribeRepositories	2
ListClusters	2
GetBucketVersioning	1
DescribeDBSecurityGroups	1
DescribeDBSnapshots	1
ConverseStream	1
ListStacks	1

Machines got there first

92.3% **349 of 378 interactions carried non-browser tooling user agents**

128 Boto3	**90** TruffleHog	**42** Python urllib
31 AWS CLI	**29** Browser / Chrome	**24** Python aiohttp
13 JavaScript / Node SDK	**9** Boto v2	**8** Go SDK
2 Other	**1** Python requests	**1** Java SDK

Caveat: The user agent identifies the client. It does not identify the operator or prove intent.

Where did the traffic come from?

Source infrastructure, not attacker nationality

76.2% **hosting / cloud**
288 interactions

US	148	NL	82
GB	27	HK	11
IL	3	KW	3
ID	2	PL	2
FR	1	KR	1

23.8% **ISP networks**
90 interactions

DE	40	IN	32
SG	10	CA	8
CH	2	JP	2
AU	1	NZ	1
MA	1	VN	1

This maps rented execution capacity. It does not tell us where the human was.

Which sources were busiest?

Source IP	Observed location	Network	Client	Hits
68.235.46.90	Chicago, US	tzulo	urllib	42
45.77.123.177	Los Angeles, US	Constant / Vultr	Boto3	37
185.65.134.196	Amsterdam, NL	31173 Services	Boto3	34
223.226.96.36	Raipur, IN	Bharti Airtel	Chrome	29
3.230.174.76	Ashburn, US	AWS	Boto3	22
2.219.188.125	London, GB	Sky UK	TruffleHog	21
34.220.125.154	Boardman, US	AWS	Boto v2	17
54.173.140.231	Ashburn, US	AWS	Boto3	14
163.116.173.75	Düsseldorf, DE	Netskope	TruffleHog	11
92.60.40.230	Amsterdam, NL	Owl	Boto3	11

One breach. Two clocks.

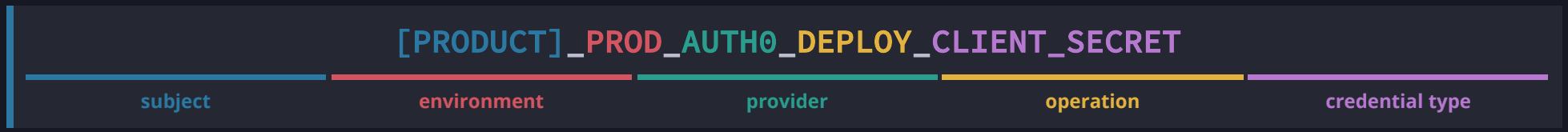
Credential clock

How long does the exposed capability remain usable?

Context clock

How long do the leaked names, paths, services, and relationships remain useful?

Can a variable name become reconnaissance?



The value is the capability. The name is the routing label.

Common Actions names already expose the route

Selected from 8,391 distinct GitHub Actions names

Observed name	Systems	What the name reveals
AWS_SECRET_ACCESS_KEY	6.13%	Cloud credential class
JENKINS_URL	5.20%	CI endpoint
NPM_TOKEN	2.89%	Package-publishing capability
GH_TOKEN	2.45%	Source-control capability
SONAR_TOKEN	1.90%	Code-analysis service
SLACK_TOKEN	1.61%	Messaging integration

The name routes the test before the value is checked.

One name can disclose four coordinates

Naming pattern	What it reveals	Observed examples
`SERVICE_(URL|HOST|ADDR)`	Target	`JENKINS_URL`, `VAULT_ADDR`, `OKTA_ISSUER`
`SERVICE_(TOKEN|API_KEY)`	Capability	`ARTIFACTORY_TOKEN`, `OPENAI_API_KEY`
`SERVICE_(USER|CLIENT_ID|ROLE)`	Identity	`ARTIFACTORY_USER`, `AWS_ROLE_ARN`
`SERVICE_(ORG|PROJECT|BUCKET)`	Scope	`VERCEL_PROJECT_ID`, `S3_BUCKET_NAME`

> **Target + capability + identity + scope. The value is unknown; the route is not.**

The namespace is already a service inventory

72 services consolidated from 3,664 of 8,391 names

Service	Distinct Actions names	Most prevalent matching name
AWS	735	`AWS_REGION` (10.14%)
GitHub	395	`GH_TOKEN` (2.45%)
Slack	267	`SLACK_TOKEN` (1.61%)
Artifactory / JFrog	246	`ARTIFACTORY_USER` (1.45%)
Microsoft Azure	218	`AZURE_TENANT_ID` (0.45%)
SonarQube	169	`SONAR_TOKEN` (1.90%)
Firebase	119	`FIREBASE_API_KEY` (0.17%)
Okta	115	`OKTA_CLIENT_ID` (0.74%)

> **This list did not come from a CMDB. It came from variable names.**

The private service graph becomes public

Internal relationship	Services exposed by names
Delivery	GitHub, Jenkins, SonarQube, Artifactory / JFrog, npm, Docker, Argo CD, Snyk
Identity and secrets	Okta, Auth0, Microsoft Azure AD, HashiCorp Vault
Cloud and data	AWS, Azure, Google Cloud, Firebase, Snowflake, PostgreSQL, Kafka
Observability and comms	Sentry, Datadog, PostHog, Slack
AI and SaaS	OpenAI, Anthropic, Gemini, Vercel, Jira, Stripe

> **Where code ships. Where identities live. Where data flows. Where defenders watch.**

The names become a hypothesis queue

Sanitized archetypes:

```
[PRODUCT]_PROD_AUTH0_DEPLOY_CLIENT_SECRET
[SERVICE]_STAGING_ARTIFACTORY_CI_READER_TOKEN
[APP]_S3_IAM_USER_AWS_SECRET_ACCESS_KEY
```

> **Caveat: 63 names** encode environment + provider + credential type + role. Names only; keyword groups overlap.

So, should we use worse names?

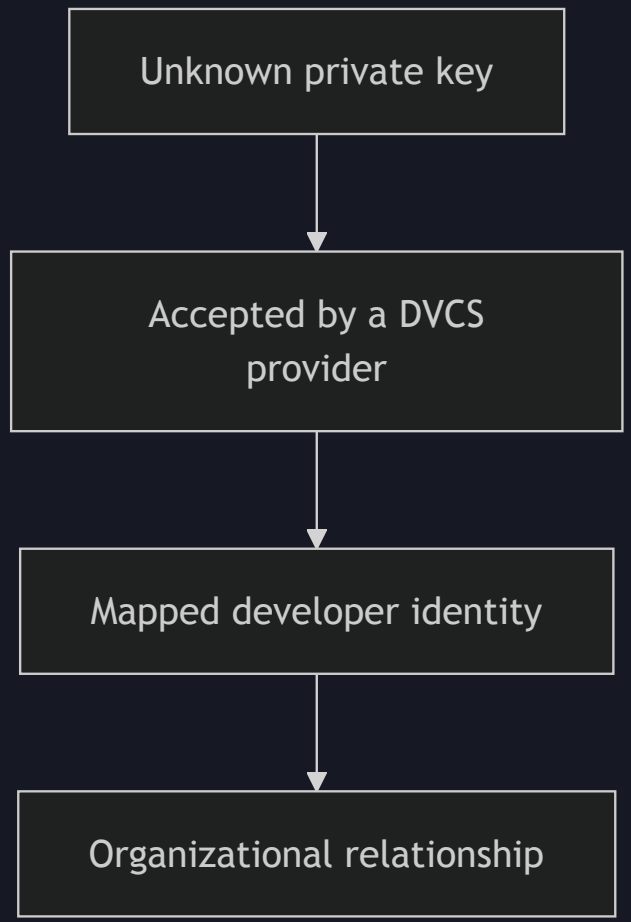
No.

- **Keep code legible:** Descriptive names help maintainers review and operate systems.
- **Stop leaking the namespace:** Allowlist what reaches jobs, subprocesses, logs, and support bundles.
- **Shorten the capability:** Prefer expiring workload identity over standing credentials.
- **Separate the scopes:** Do not let one process inherit production, development, read, and publish access together.

> **The fix is not naming it `X7`. The fix is not leaking the namespace.**

The key did not become more dangerous

It became legible.



Let's assume prevention fails

Assume a trusted package, action, or extension eventually executes malicious code in a developer or build environment.

Then ask:

- What can it read?
- Which identities can it inherit?
- What can those identities publish or modify?
- Where can the data leave?
- Which capabilities survive incident containment?

How do we limit the blast radius?

- Replace standing publishing tokens with short-lived workload identity
- Narrow credential scope, audience, and lifetime
- Separate read, build, and publish identities
- Isolate jobs so unrelated secrets are never present together
- Constrain repository creation and visibility changes
- Restrict unnecessary outbound paths from build environments

Do not let one compromised process inherit the organization

What should the attacker inherit?

On compromise, the attacker should inherit a capability that is:

- Narrow
- Expiring
- Non-propagating

So, are we remediated?

- The malicious package was removed?
- The public repository disappeared?
- The obvious tokens were rotated?
- Every exposed capability was invalidated?
- The leaked operational map is no longer accurate?

Recovery is the disappearance of usable exposure

What we are still measuring

For each artifact, we ask:

- What was exposed?
- Was it recognized and verified?
- Can the actual owner be attributed?
- Is it still accepted?
- What context makes it useful?
- How long does that usefulness survive?

Tools of the trade

Tools used include:

01
TruffleHog
`trufflesecurity/trufflehog`
Credential discovery, classification, and verification

02
gimmePATz
`6mile/gimmepatz`
GitHub and npm PAT validation, scope, and access analysis

03
GH Navigator
`cyfinoid/ghnavigator`
Client-side GitHub token analysis and repository navigation

04
KeyChecker
`cyfinoid/keychecker`
SSH private-key fingerprinting and DVCS account mapping

05
Canarytokens
`canarytokens.org`
Controlled, functionless AWS credential for observing attempted use

In short

Incident closure is not exposure closure.

- **Look beyond scanner hits:** Include unverified secrets, private keys, provenance, and operational context.
- **Revoke capabilities, not artifacts:** A hidden repository is not a revoked token or an invalidated relationship.
- **Make protection follow trust:** Supply-chain contributors need defenses that match their downstream blast radius.

> **Assume breach. Make inherited capabilities narrow, expiring, and non-propagating.**

Incident references

- GitHub: securing the npm supply chain after Shai Hulud 1
- Aikido: Shai Hulud 1 technical behavior
- RedHuntLabs: GitHub patterns in the second-wave npm attack
- Microsoft: Shai Hulud 2 technical analysis
- Wiz: Shai Hulud 2 spread, cross-account exfiltration, and victimology
- GitHub: generic private-key scanning availability
- GitHub: supported secret-scanning patterns and capabilities
- Canarytokens: controlled canary credentials

Questions or concerns?

Let's keep the conversation going.

Anant Shrivastava

Cyfinoid Research

anant@cyfinoid.com

Kumar Ashwin

RedHuntLabs

hacker@redhuntlabs.com