Detection Engineering Demystified: Building Custom Detections for GitHub Enterprise

David French

Staff Adoption Engineer, Google Cloud

@threatpunter

BSides Las Vegas 2024

David French > About Me

- 18+ years in IT and cybersecurity
  - Blue Team life: Detection Engineer, Threat Hunter, SOC Analyst
  - Vendor life: Threat Research, Detection Engineering, building SIEMs & EDRs
- Currently at Google Cloud (Google Security Operations)
- Formerly Twilio, Elastic, Endgame, Capital Group
- Enjoys sharing knowledge & research:
  - Speaker at Black Hat, BSides, FIRST
  - Blog, community contributions, Detection-as-Code, Dorothy
- When I'm not working, I'm hiking, fishing, cycling, etc

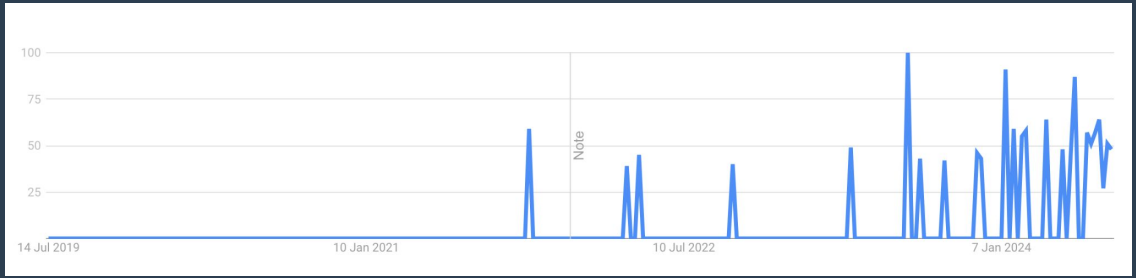
Agenda

1. A Brief Introduction to Detection Engineering
2. Case Study: Intelligence-Driven Detection Engineering
3. Developing a New Detection
4. Monitoring Your Data Pipeline
5. Testing Detections
6. Wrap Up
    a. Key Takeaways
    b. Useful Resources
    c. Q&A

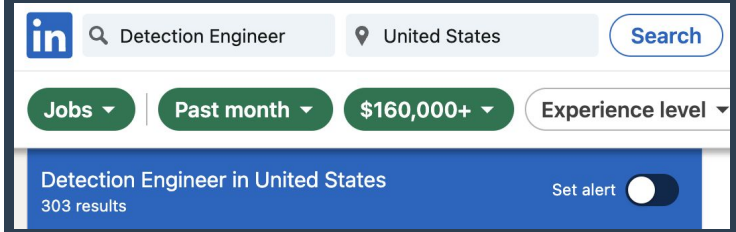
A Brief Introduction to Detection Engineering

What is Detection Engineering?

- Specialization focused on implementing detective security controls that identify malicious or unauthorized activity
- Aims to detect potential incidents before they can cause significant damage
- Continuous process for developing, testing, and refining detections
- Complements preventative controls
- Emphasis on detecting behavior vs. indicators of Compromise (IOCs)



Interest over time (Source: Google Trends)



Benefits of Detection Engineering (1)

- Reduced risk
  - Proactive detection identifies attacks early on
  - Gives defenders a chance to respond before significant damage occurs
  - Can save $ and (sometimes) lives…
- Faster response times
  - Actionable detections provide the info needed to understand and respond to threats quickly
  - Helps maintain company's reputation & trust with customers

From Risky Biz News:

Ransomware attacks increase hospital mortality rates:

A whitepaper published last year by academics from the University of Minnesota's medical school has looked at the aftermath of ransomware attacks on US hospitals and found evidence to suggest that mortality rates typically increase by around 20%.

Hacked to Pieces? The Effects of Ransomware Attacks on Hospitals and Patients

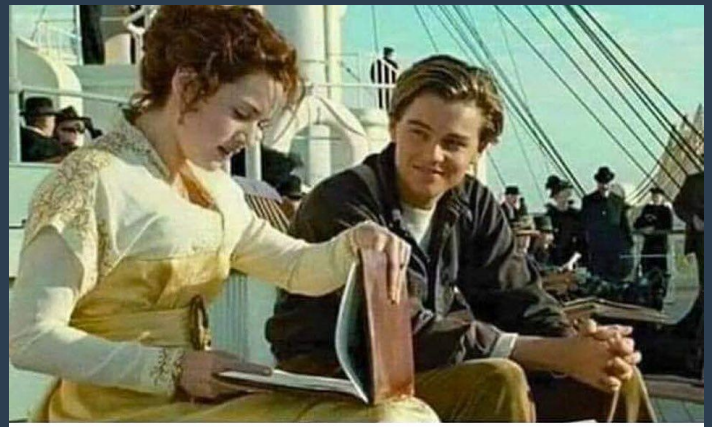
USD 4.45 million

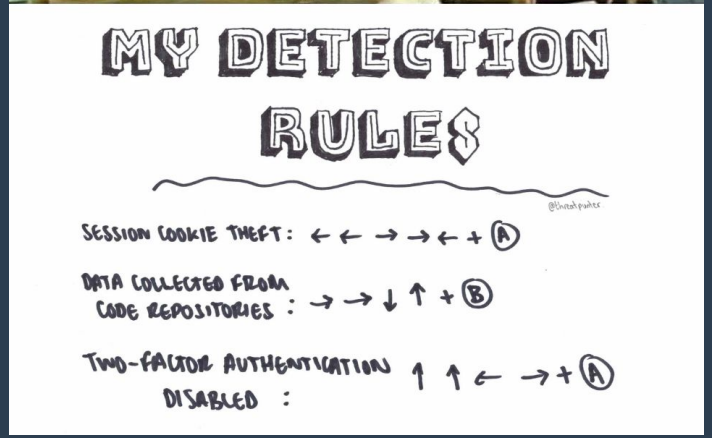
The global average cost of a data breach in 2023 was USD 4.45 million, a 15% increase over 3 years.

IBM Cost of a Data Breach Report 2023

Benefits of Detection Engineering (2)

- Increased visibility
  - Engineers proactively identify and integrate relevant data sources
  - Detection coverage is assessed against new threats/tactics
- Can help satisfy compliance/regulatory requirements
  - e.g. Detections related to data loss prevention or SWIFT compliance

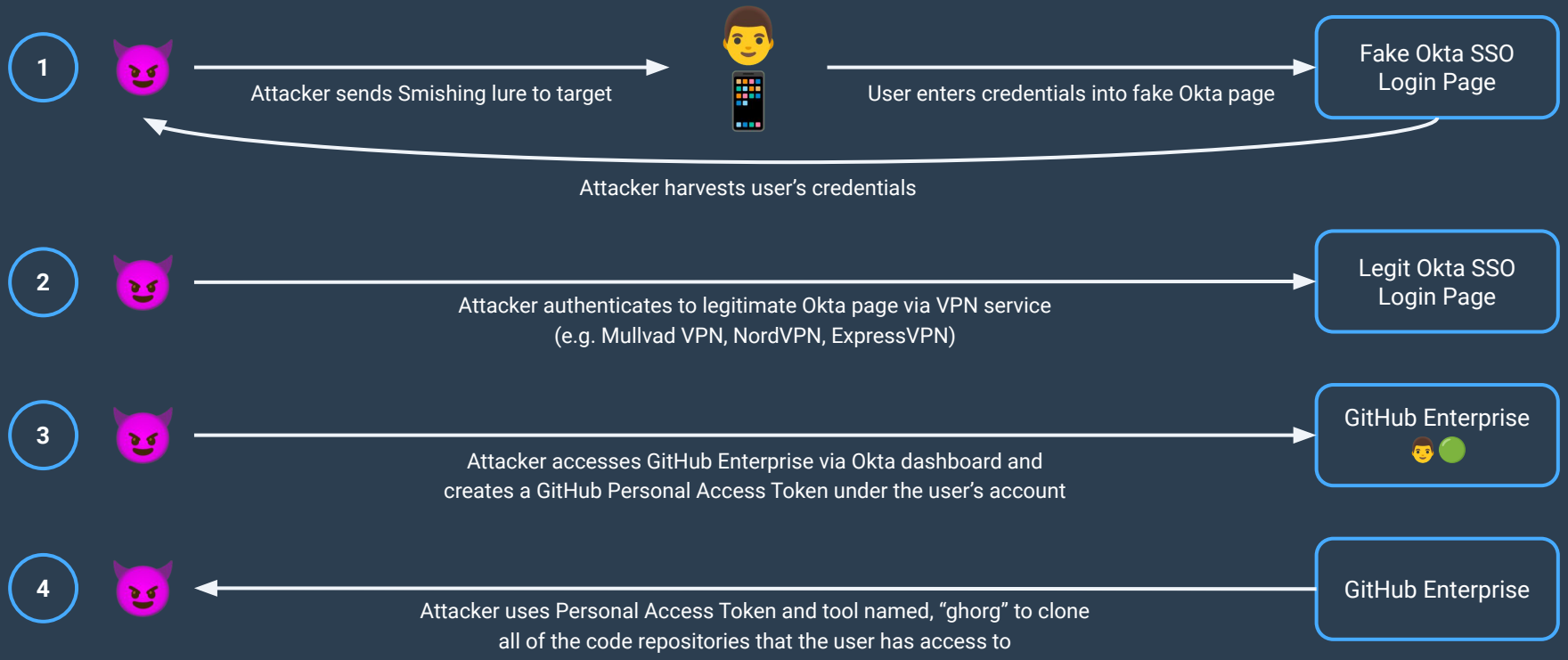




Case Study: Intelligence-Driven Detection Engineering

Case Study: Intelligence-Driven Detection Engineering

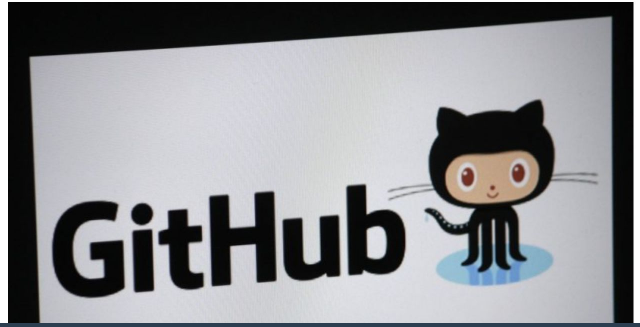
Intelligence received from fellow practitioner working in the same industry



Why GitHub is Targeted by Attackers

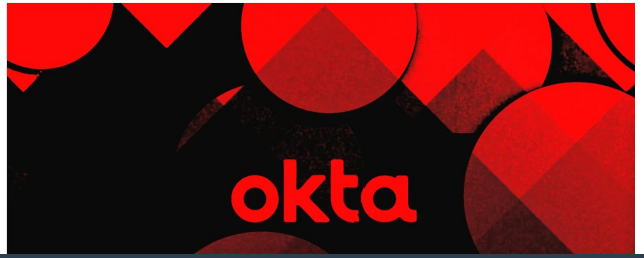
- Code repositories may contain intellectual property
- Source code can be examined for vulnerabilities that can be exploited in subsequent attacks
- Harvested secrets can be used in follow up attacks
- Software supply chain attacks

Report: Microsoft's GitHub Account Gets Hacked



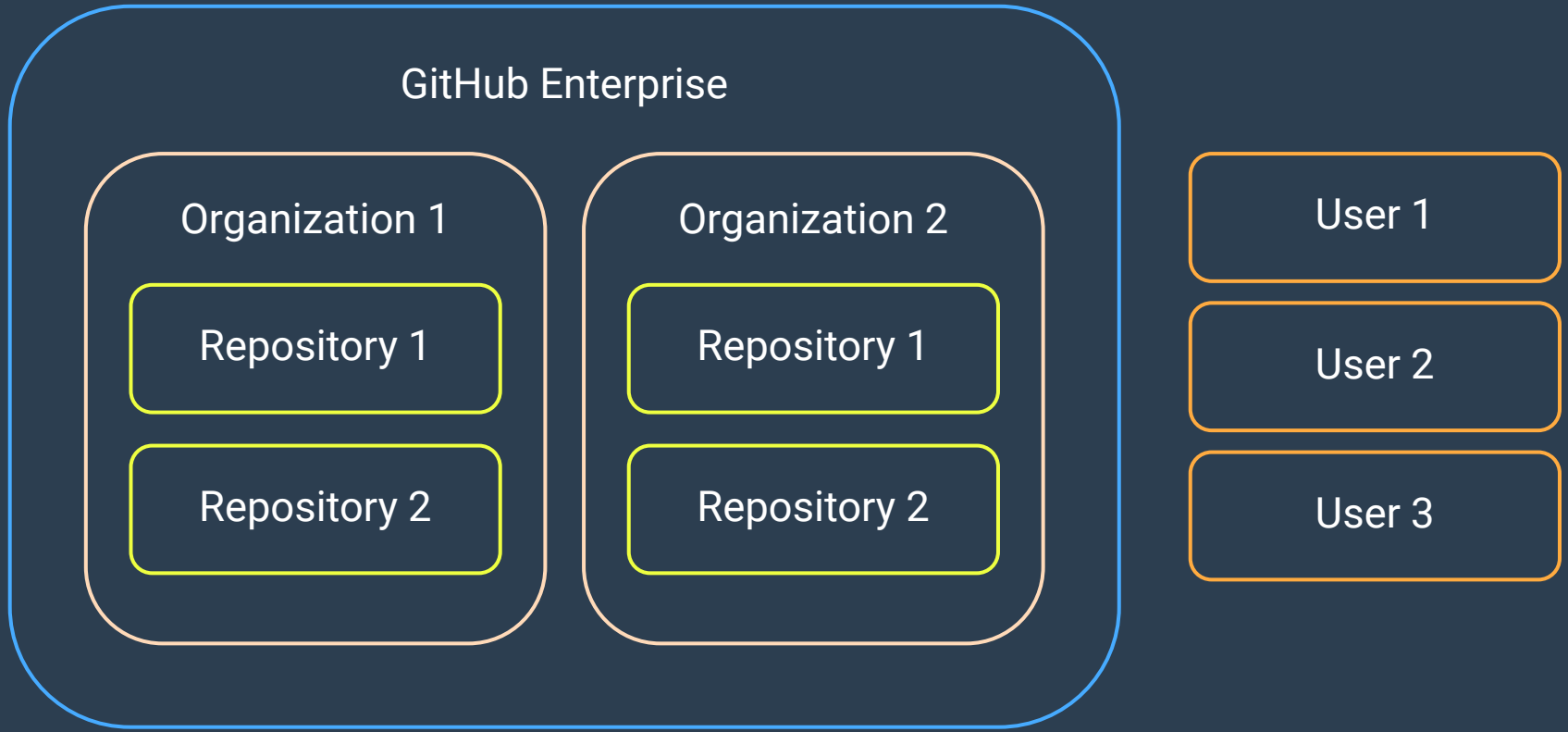
Okta's source code stolen after GitHub repositories hacked

By Ax Sharma December 21, 2022 01:15 AM 1

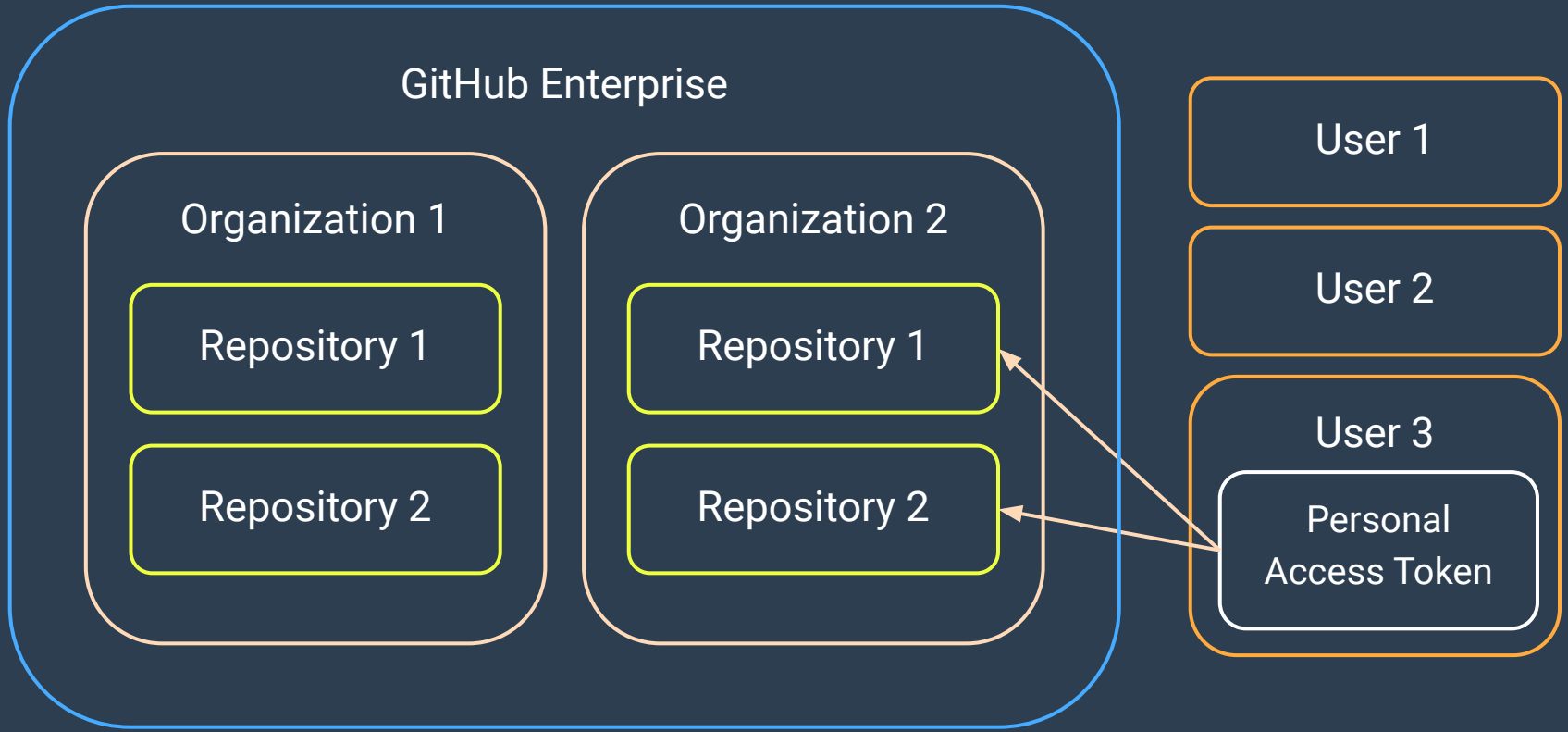


Why Attackers Target GitHub, and How You Can Secure It - Matt Caulfield

GitHub: Key Terms & Concepts



GitHub: Key Terms & Concepts



Reviewing the Attacker's Tool of Choice

README | Code of conduct | Apache-2.0 license | Security

ghorg

go report A+ | reference | mentioned in awesome | License Apache 2.0 | WakeMeOps v1.9.12

Pronounced [gore-guh]; similar to gorge. You can use ghorg to gorge on orgs.

Use ghorg to quickly clone all of an orgs, or users repos into a single directory. This can be useful in many situations including

1. Searching an orgs/users codebase with ack, silver searcher, grep etc..
2. Bash scripting
3. Creating backups
4. Onboarding new team members (cloning all team repos)
5. Performing Audits

3. Clone a **users** repos

```
ghorg clone <github_username> --clone-type=user --base-url=https://<your-hosted-github>.com --token=XXXXXX
```

Indicator vs. Behavior-Based Detection

Indicator-Based Detection

Analyze tool and write signatures to alert on specific indicators (e.g. user agent)

Tools are often easily modified to evade this type of detection

What if the attacker uses a different tool?

Behavior-Based Detection

Focus on detecting underlying actions that are required to achieve the desired outcome

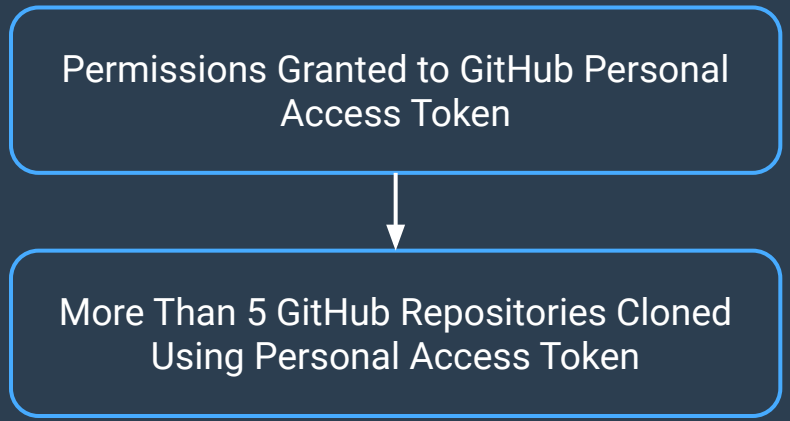
These detections often have a longer shelf life than indicator-based detections

Incorporates context and sequences/patterns of behavior

Developing a New Detection

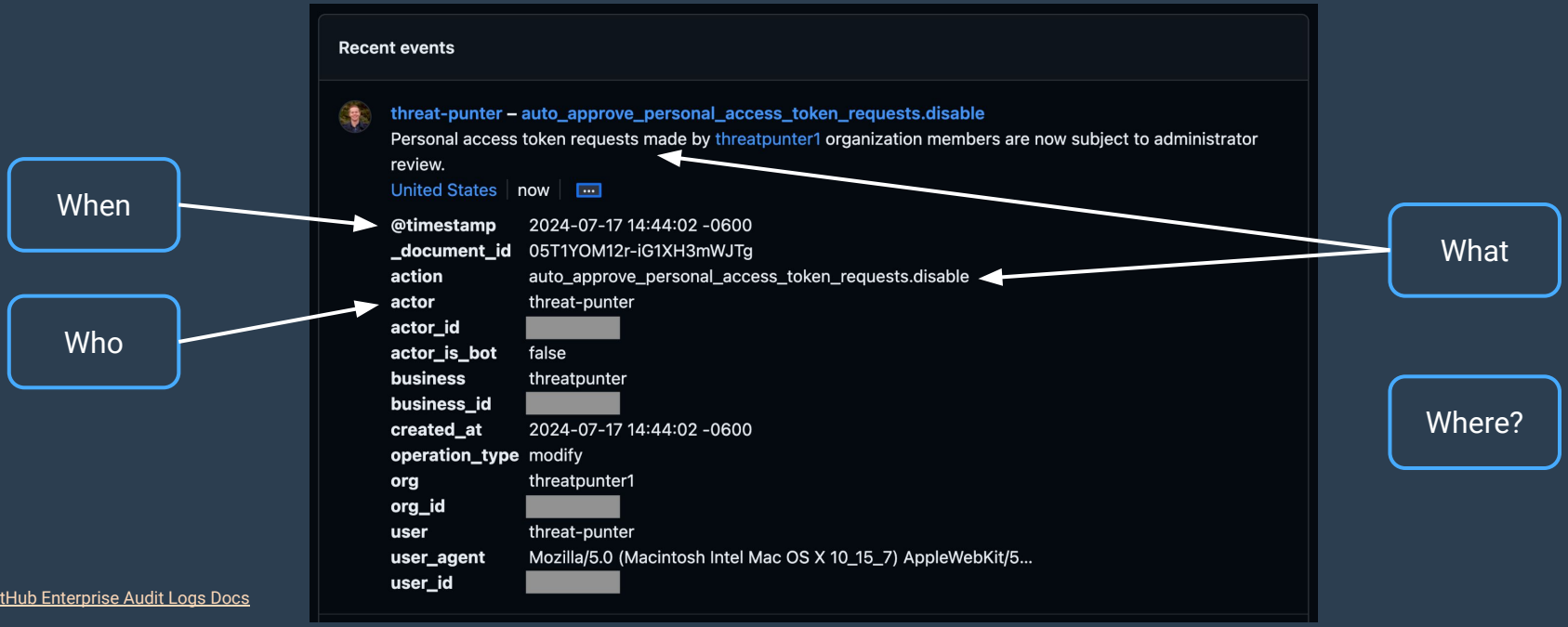
Designing a New Detection

- Let’s develop a detection that alerts us when a GitHub user performs the following sequence of behaviors
- We’ll build a simple detection and refine it from there



Understanding the GitHub Enterprise Audit Log

“The audit log lists events triggered by activities that affect your enterprise within the last 180 days. The audit log retains Git events for seven days.”

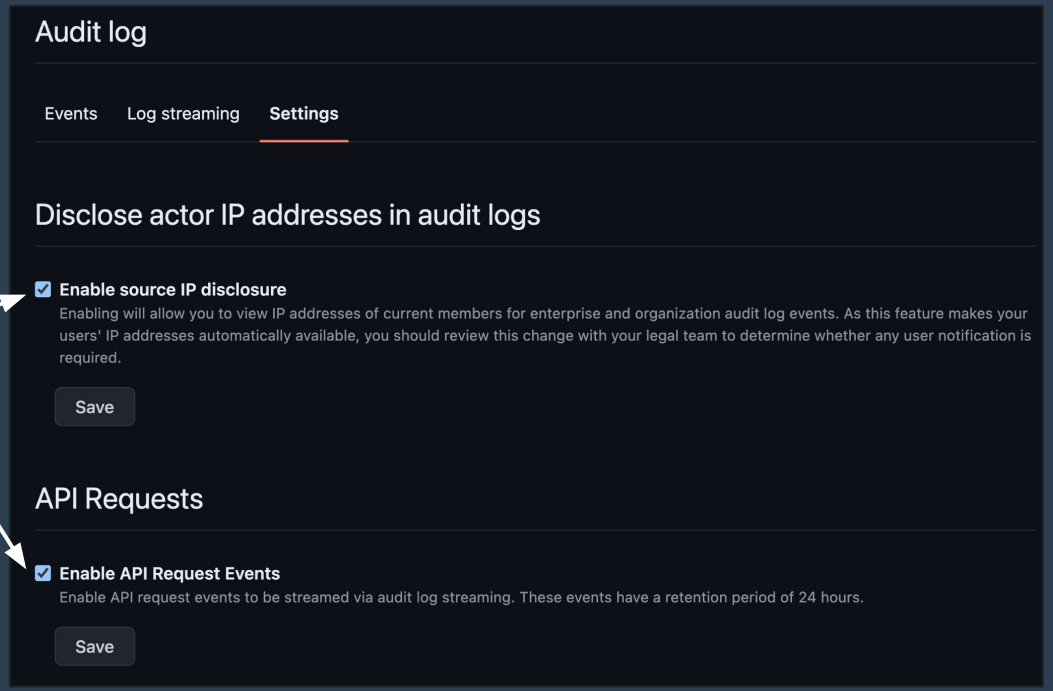


GitHub Enterprise Audit Logs Docs

Configuring the GitHub Audit Log

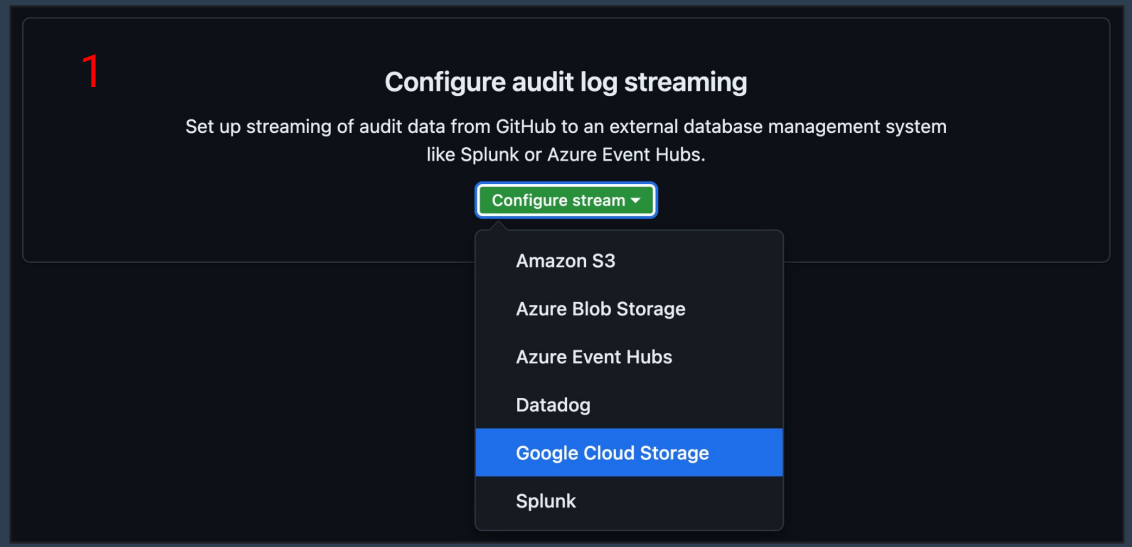
This data source must be configured before it can be used for many detection use cases

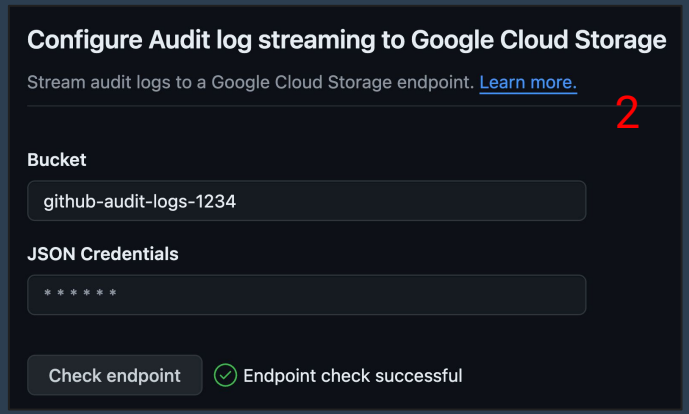
These options are not enabled by default

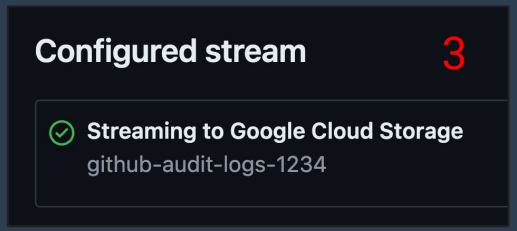


Configuring GitHub Audit Log Streaming

The audit log can be streamed to an external system for storage and further analysis







Streaming the Audit Log for Your Enterprise

Ingesting GitHub Audit Logs into a SIEM

Enables log analysis, detection use cases, correlation with other security-relevant events, long term retention, etc

Simulating the Behavior to Detect

1. Create a new GitHub Personal Access Token (PAT)
2. Grant the token access to at least six GitHub repositories
3. Clone the six GitHub repositories using the PAT
4. Explore the events that were generated

EVENTS | PIVOT | Search events...

TIMESTAMP	EVENT	PRINCIPAL.USER.USERID	METADATA.PRODUCT_EVENT_TYPE	PRINCIPAL.IP	EXTRACTED.FIELDS["REPO"]	EXTRACTED.FIELDS["PROGRAMMATIC_ACCESS_TYPE"]
2024-07-17T04:34:23.925	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo6	threat-punter	git.clone		threatpunter1/repo6	Fine-grained personal access token
2024-07-17T04:34:17.486	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo5	threat-punter	git.clone		threatpunter1/repo5	Fine-grained personal access token
2024-07-17T04:34:11.092	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo4	threat-punter	git.clone		threatpunter1/repo4	Fine-grained personal access token
2024-07-17T04:34:00.361	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo3	threat-punter	git.clone		threatpunter1/repo3	Fine-grained personal access token
2024-07-17T04:33:51.889	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo2	threat-punter	git.clone		threatpunter1/repo2	Fine-grained personal access token
2024-07-17T04:33:39.099	USER_RESOURCE_UPDATE_CONTENT threat-punter - threatpunter1/repo1	threat-punter	git.clone		threatpunter1/repo1	Fine-grained personal access token
2024-07-17T04:33:22.947	USER_RESOURCE_UPDATE_CONTENT threat-punter - unknown resource	threat-punter	personal_access_token.access_granted		[Unknown]	[Unknown]

GitHub Audit Log Events Documentation

Developing the Detection Logic (1)

```
18  events:
19    // GitHub Personal Access Token (PAT) access granted event
20    $github_pat.metadata.product_name = "GITHUB"
21    $github_pat.metadata.product_event_type = "personal_access_token.access_granted"
22
23    // GitHub repository clone event
24    $github_clone.metadata.product_name = "GITHUB"
25    $github_clone.metadata.product_event_type = "git.clone"
26    $github_clone.additional.fields["repository_public"] = "false"
27    $github_clone.additional.fields["programmatic_access_type"] = "Fine-grained personal access token"
28    $github_clone.target.resource.name = $github_repo_name
29
30    // Join GitHub PAT access granted event to GitHub repository clone event
31    $github_pat.principal.user.userid = $github_clone.principal.user.userid
32
33    // Placeholder for match section
34    $github_pat.principal.user.userid = $user_id
35
36    // Ensure PAT access granted event occurred before repository clone events
37    $github_pat.metadata.event_timestamp.seconds < $github_clone.metadata.event_timestamp.seconds
```

Developing the Detection Logic (2)

```
40    match:
41      $user_id over 30m
42
43    outcome:
44      $github_repo_name_distinct_count = count_distinct($github_repo_name)
45
46    condition:
47      // Customize GitHub repo count to fit your environment
48      $github_pat and $github_clone and $github_repo_name_distinct_count > 5
49  }
```

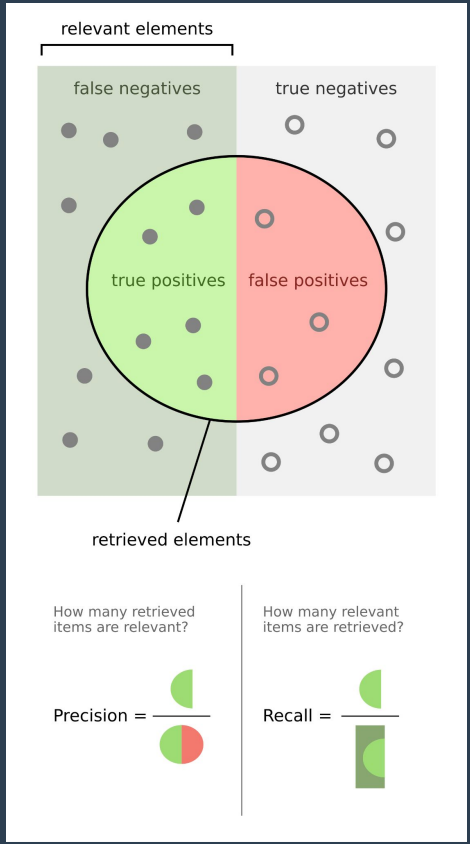
Validating the Detection

- Simulate the behavior again and validate that the detection generates an alert
- Or run the detection logic over historical events and validate that it matches the intended behavior

	TIMESTAMP ↓	DETECTION	GITHUB_REPO_NAME_DISTINCT_COUNT…	USER_ID (DETECTION)	METADATA.PRODUCT_EVENT_TYPE
⌄	2024-07-17T21:30:00.000	DETECTION user_id:threat-punter	6	threat-punter	personal_access_token.access_granted, …
EVENTS (7)					
	2024-07-17T21:03:23.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - unknown resource	[Unknown]	[Unknown]	personal_access_token.access_granted
	2024-07-17T21:03:28.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo1	[Unknown]	[Unknown]	git.clone
	2024-07-17T21:03:33.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo2	[Unknown]	[Unknown]	git.clone
	2024-07-17T21:03:38.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo3	[Unknown]	[Unknown]	git.clone
	2024-07-17T21:03:43.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo4	[Unknown]	[Unknown]	git.clone
	2024-07-17T21:03:48.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo5	[Unknown]	[Unknown]	git.clone
	2024-07-17T21:03:53.000	USER_RESOURCE_UPDATE_CONTENT G threat-punter - threatpunter1/repo6	[Unknown]	[Unknown]	git.clone

False Positive Report from the SOC

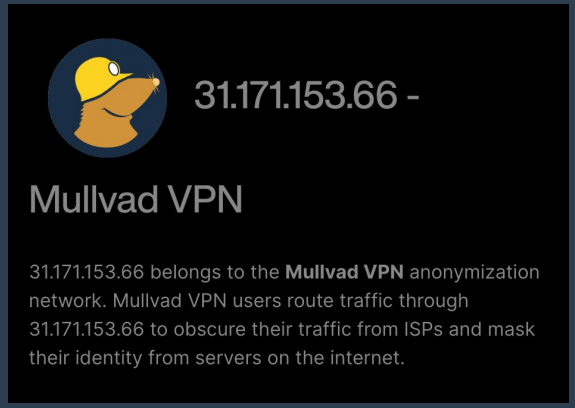
- When a software engineer gets a new laptop, it's common for them to:
  - Create a new GitHub Personal Access Token
  - Clone the private code repos that they work on
- This behavior causes pesky false positives
- Let's look for a way to tune the detection to increase its precision

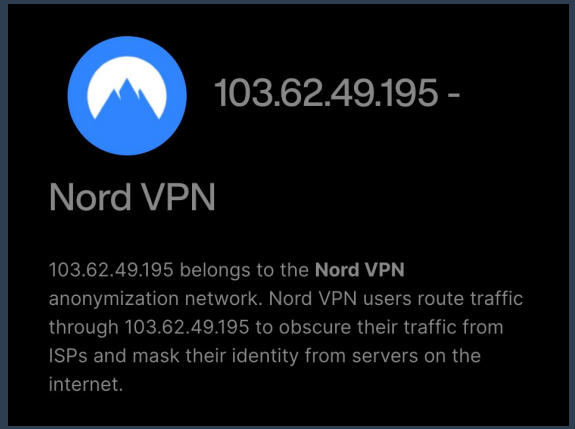


Precision and recall – Wikipedia

Detection Engineering and Third-Party Data Feeds

- Modify the detection logic to alert when the behavior originates from an IP address that's attributed to a VPN service
  - Might introduce FPs if engineers are using VPN services
  - Users should be using the company-provided VPN
  - Create another detection to alert on non-approved VPN use
- Spur is a service that provides data on VPNs, residential proxies, and bots
  - Feeds can be ingested into SIEM for correlation or enrichment





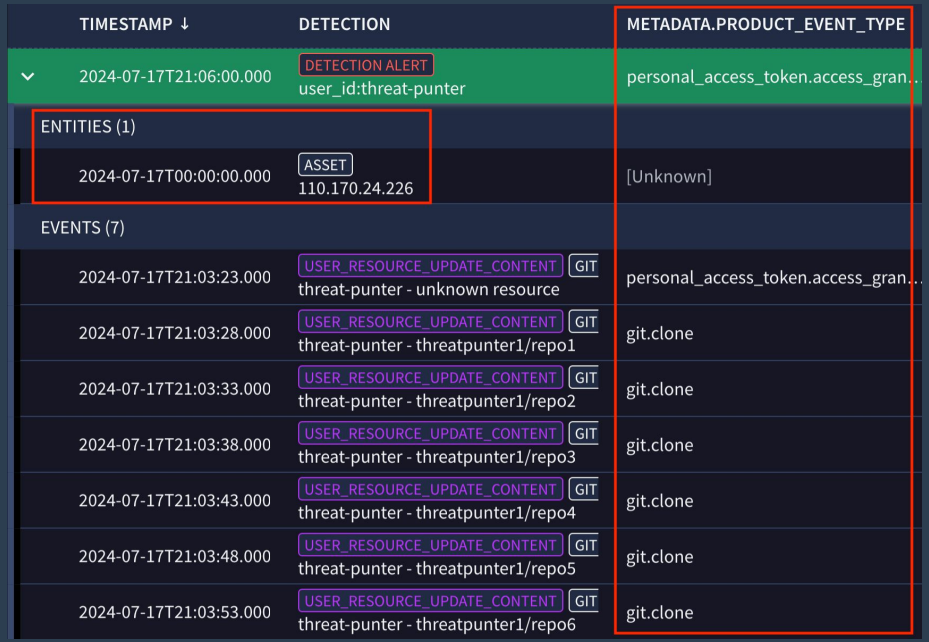
Modifying the Detection Logic

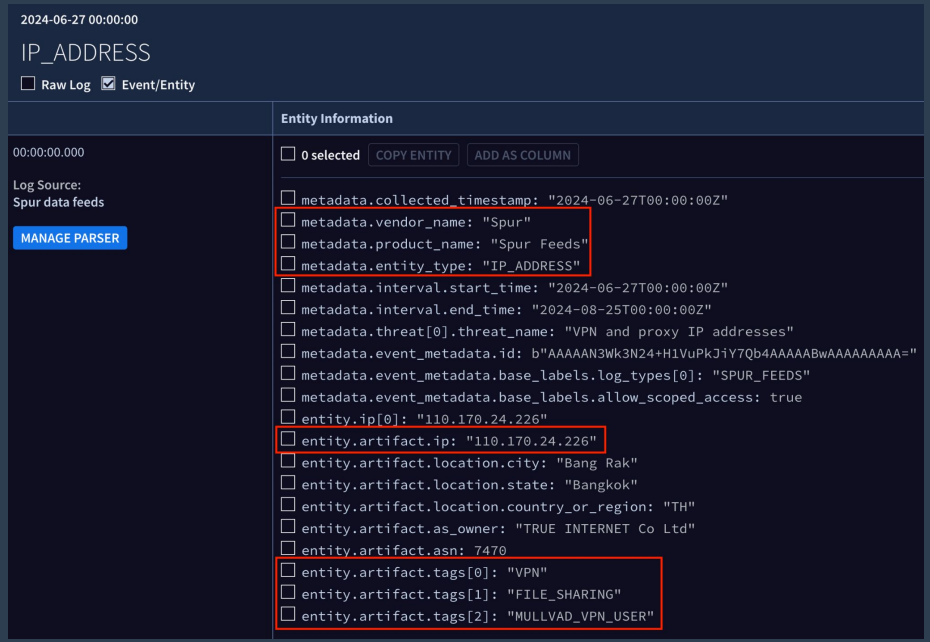
Modify the detection logic to match on events that originated from an IP address that Spur attributes to a VPN service

```
18  events:
19    // GitHub Personal Access Token (PAT) access granted event
20    $github_pat.metadata.vendor_name = "GITHUB"
21    $github_pat.metadata.product_name = "GITHUB"
22    $github_pat.metadata.product_event_type = "personal_access_token.access_granted"
23
24    // GitHub repository clone event
25    $github_pat.metadata.vendor_name = "GITHUB"
26    $github_clone.metadata.product_name = "GITHUB"
27    $github_clone.metadata.product_event_type = "git.clone"
28    $github_clone.additional.fields["repository_public"] = "false"
29    $github_clone.target.resource.name = $github_repo_name
30    $github_clone.principal.ip = $ip
31
32    // IP address is in Spur's VPN IP address feed
33    $spur.graph.metadata.vendor_name = "Spur"
34    $spur.graph.metadata.product_name = "Spur Feeds"
35    $spur.graph.metadata.entity_type = "IP_ADDRESS"
36    $spur.graph.entity.artifact.tags = "VPN"
37    // Join GitHub events with Spur data
38    $spur.graph.entity.artifact.ip = $ip
```

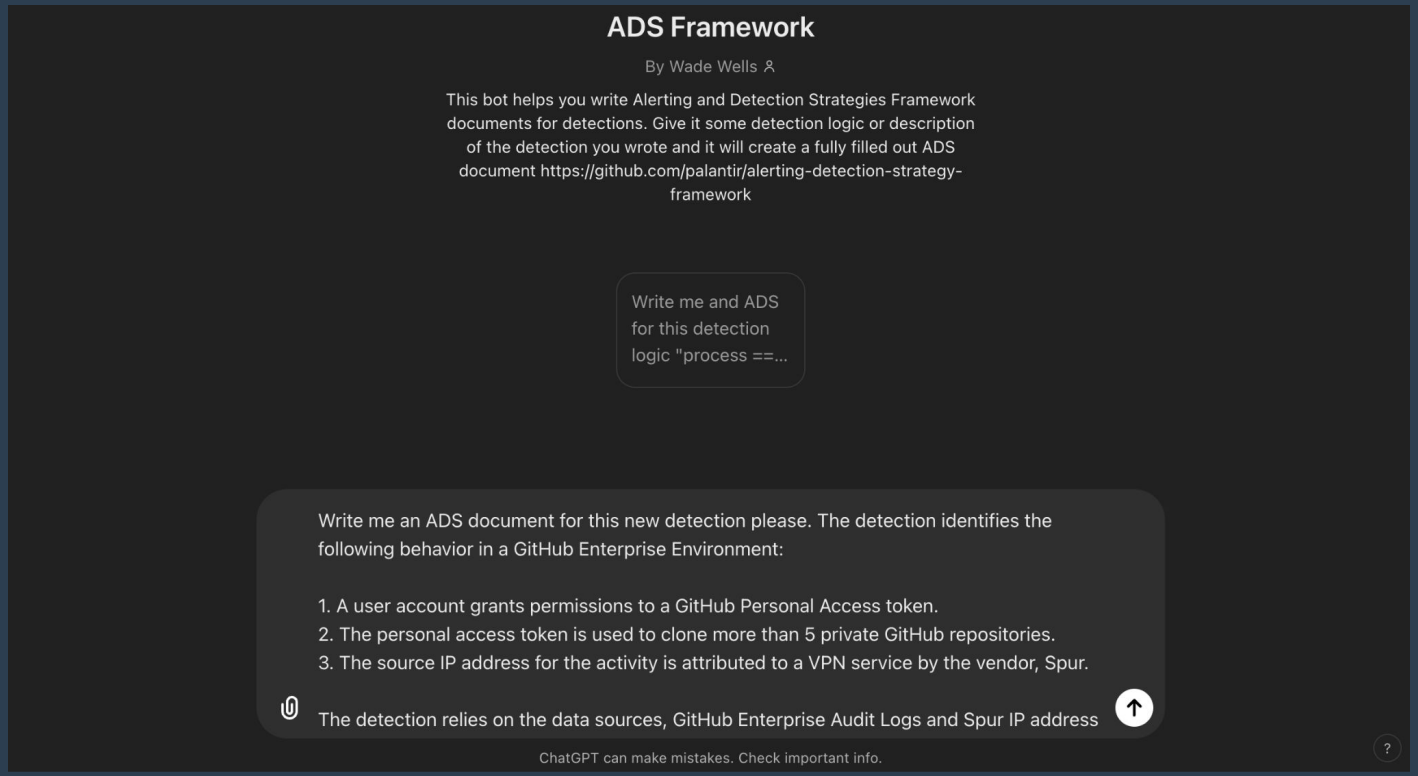
Validating the Detection

Simulate the attacker behavior again and validate that the detection alerts on the intended behavior



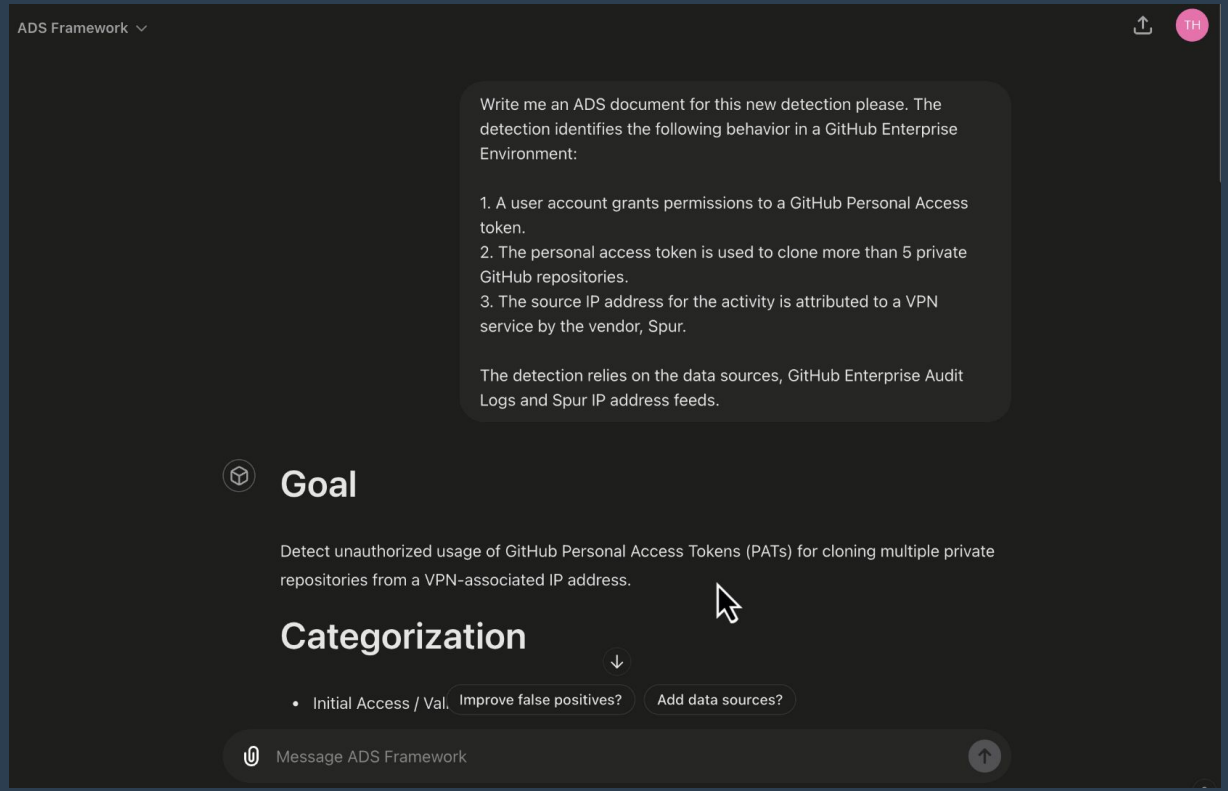


Documenting the Detection (1)



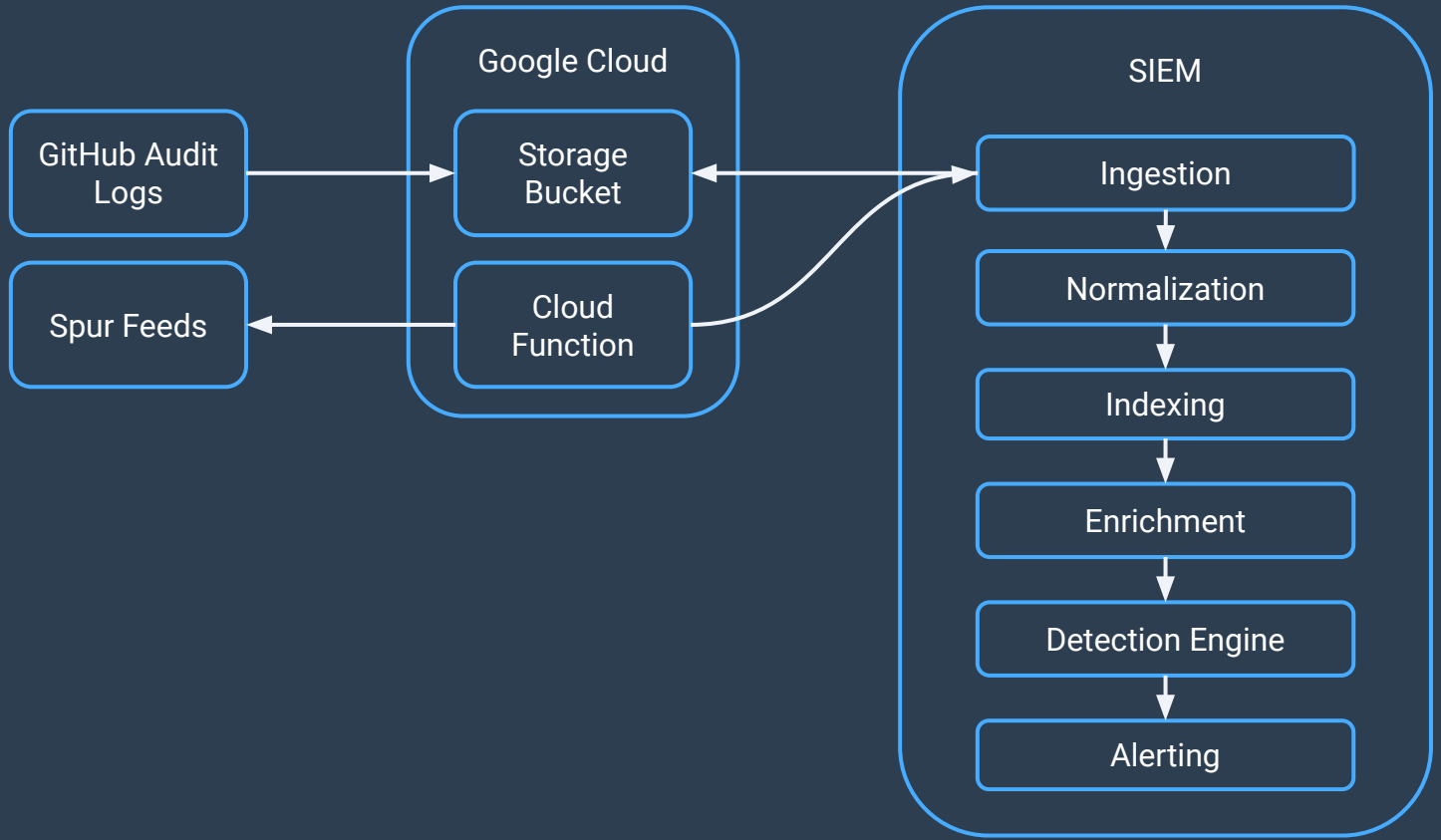
ADS Framework AI Assistant – Wade Wells
Alerting and Detection Strategy Framework - Palantir

Documenting the Detection (2)



Monitoring Your Data Pipeline

Simple Data Pipeline Overview

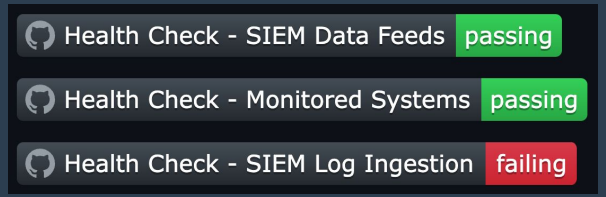


Reasons to Monitor Your Data Pipeline

- Environments drift over time
- Logging interruptions or spikes
- Parsing errors
  - Vendors change their logging schema
- Latency affecting log ingestion, normalization, indexing, or detection engine
- Highly recommend Josh Liburdi's Building Better Hunt Data presentation
  - Deep dive on monitoring & improving data quality

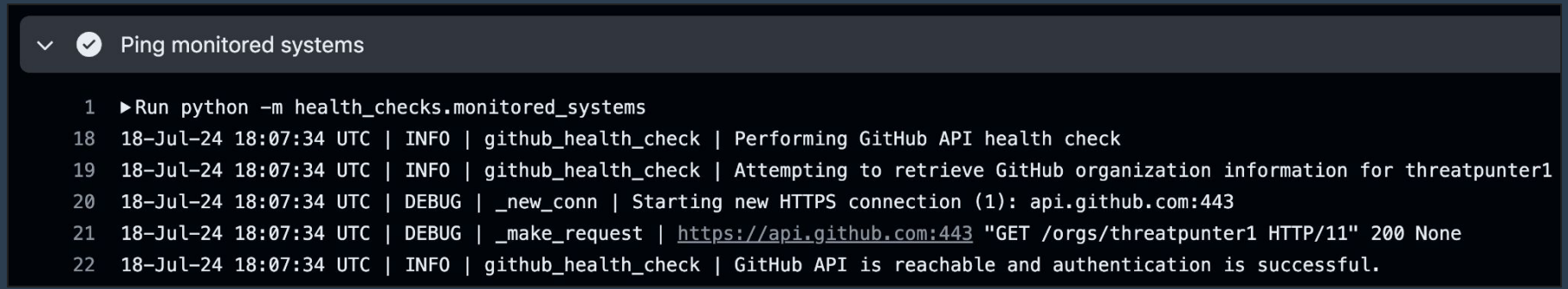
Example Technique for Proactive Data Pipeline Monitoring

- We need to detect issues with our data pipeline
- “Silent asset” detections can generate false positives
- One option is to implement health checks for systems that are important to us
  - Carry out a basic read operation against monitored systems
  - Validate that the SIEM ingested, normalized, and indexed the event(s)
- Health check jobs can run in your SOAR, automation tool, or CI/CD pipeline



Example: GitHub Audit Log Health Check (1)

- Health check consists of two GitHub Actions jobs
- First job is scheduled to run daily
- Simple API call to retrieve info for the GitHub Enterprise we're monitoring



Example: GitHub Audit Log Health Check (2)

Second job validates that the events from GitHub were ingested, normalized, indexed, and are searchable in the SIEM

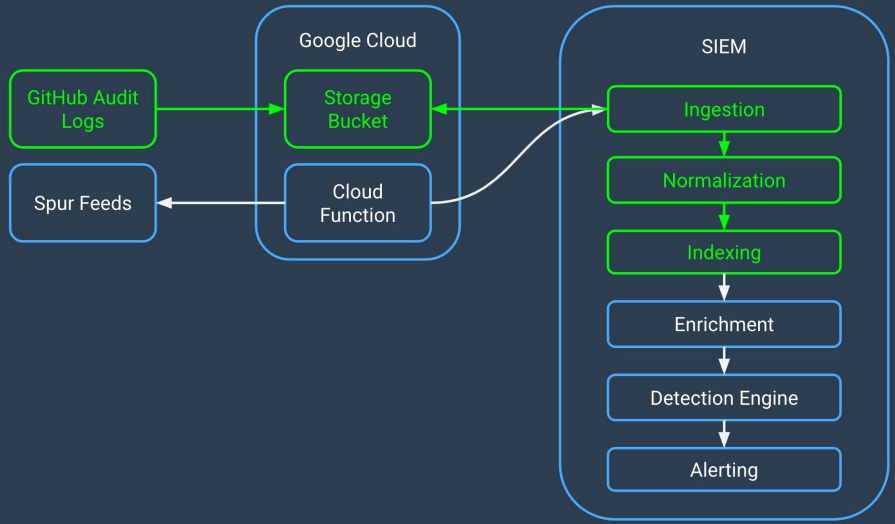
Validate SIEM log ingestion from monitored systems

```
1  ▶Run python -m health_checks.log_ingestion
17 18-Jul-24 18:24:53 UTC | INFO | validate_github_log_ingestion | Running UDM search to validate GitHub log ingestion:
18 {
19     "query": "metadata.event_type = \"USER_RESOURCE_UPDATE_CONTENT\" AND metadata.log_type = \"GITHUB\" AND metadata.product_name =
   \"GITHUB\" AND metadata.product_event_type = \"api.request\" AND extracted.fields[\"url_path\"] = \"/organizations/166444767\" AND
   network.http.method = \"GET\" AND extracted.fields[\"user_programmatic_access_name\"] = \"github-api-health-check\"",
20     "start_time": "2024-07-18T15:24:53Z",
21     "end_time": "2024-07-18T18:24:53Z"
22 }
23 18-Jul-24 18:24:57 UTC | INFO | validate_github_log_ingestion | 1 events returned from UDM search:
24 {
25     "query": "metadata.event_type = \"USER_RESOURCE_UPDATE_CONTENT\" AND metadata.log_type = \"GITHUB\" AND metadata.product_name =
```

Health Check - SIEM Log Ingestion passing

Example: GitHub Audit Log Health Check (3)

- This isn't a comprehensive solution for monitoring *all* components of your data pipeline
- Intended to be a practical example to spark ideas
- With a bit of code, we have some confidence that security events from our GitHub environment are being ingested into our SIEM and are searchable



Testing Detections

Testing Your Detections: Don’t Skip This Step!

- Validates that your detection & alerting capabilities are working
- Common issues impacting detections:
  - Logging interruptions or parsing errors
  - Misconfigured data sources
  - Vendors changing their logging schema
- Testing helps you identify & fix issues before malicious activity goes unnoticed

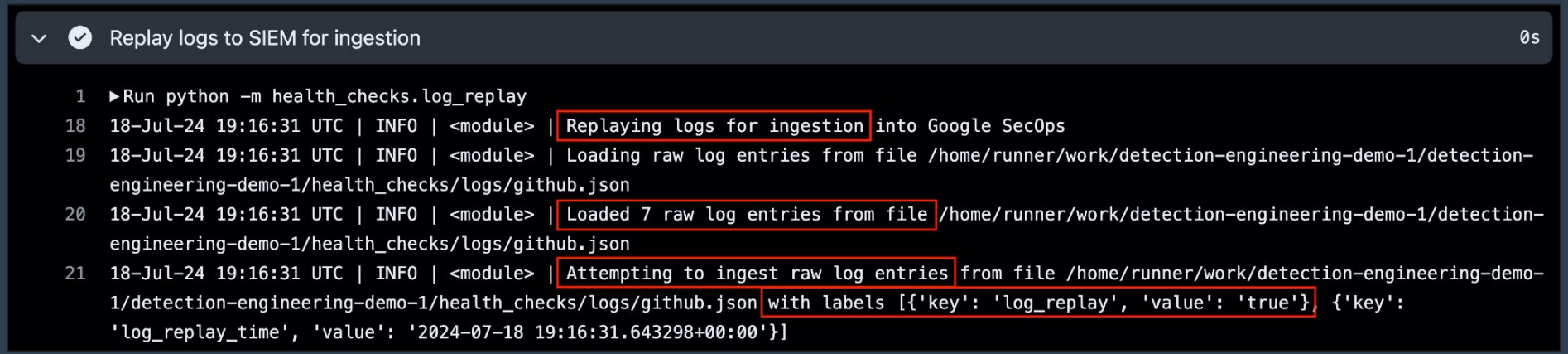
State of SIEM Detection Risk (2024) – CardinalOps

Example: Testing the GitHub Detection (1)

- How can we automate the regular testing of the new GitHub detection?
- Perfect world: Create a test that simulates the attacker behavior end-to-end and validates that the detection generated an alert
- Real world:
  - Not possible to create & configure a new Personal Access Token via GitHub's API
    - Do we want to do this anyway? Probably not…
  - We don't want to carry out sensitive actions from a VPN service
  - Developing complex tests is expensive

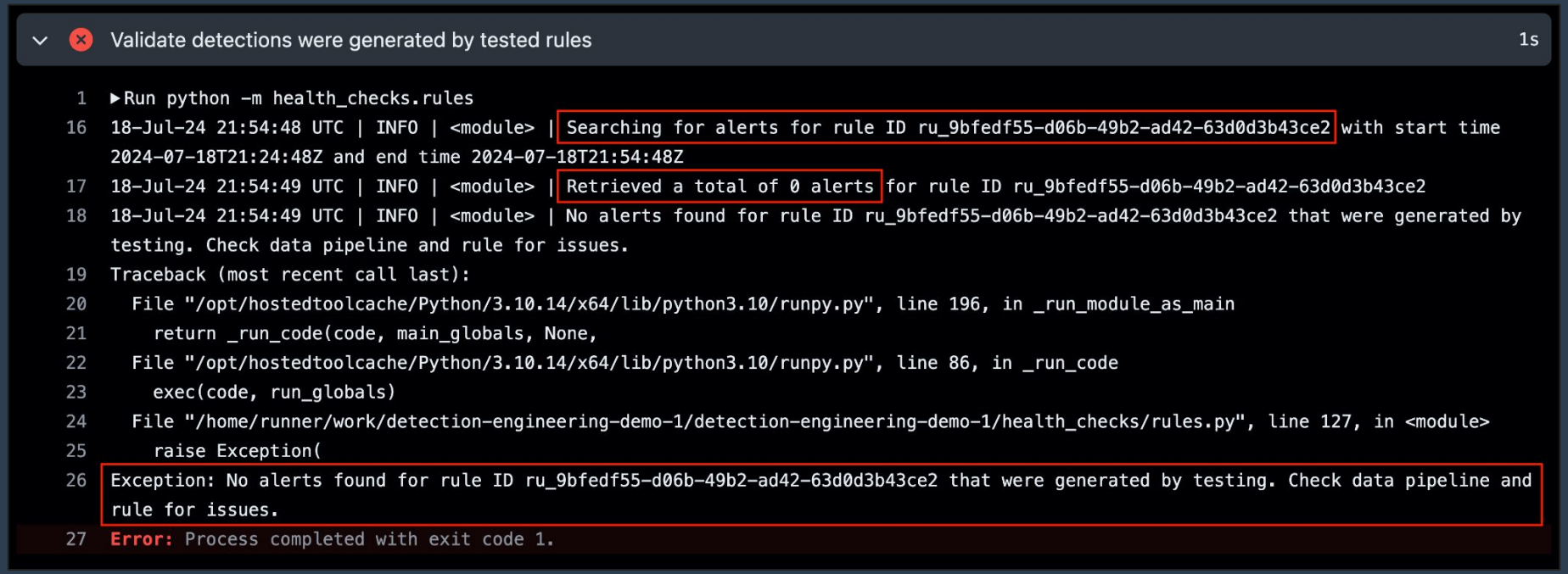
Example: Testing the GitHub Detection (2)

- Two GitHub Actions jobs scheduled to run daily
- First job replays historical GitHub audit log events to the SIEM for ingestion
  - Modify the timestamps in the logs prior to ingestion
  - Label the events so it's clear they're related to testing
- Second job validates that the detection generated an alert



Example: Testing the GitHub Detection (3)

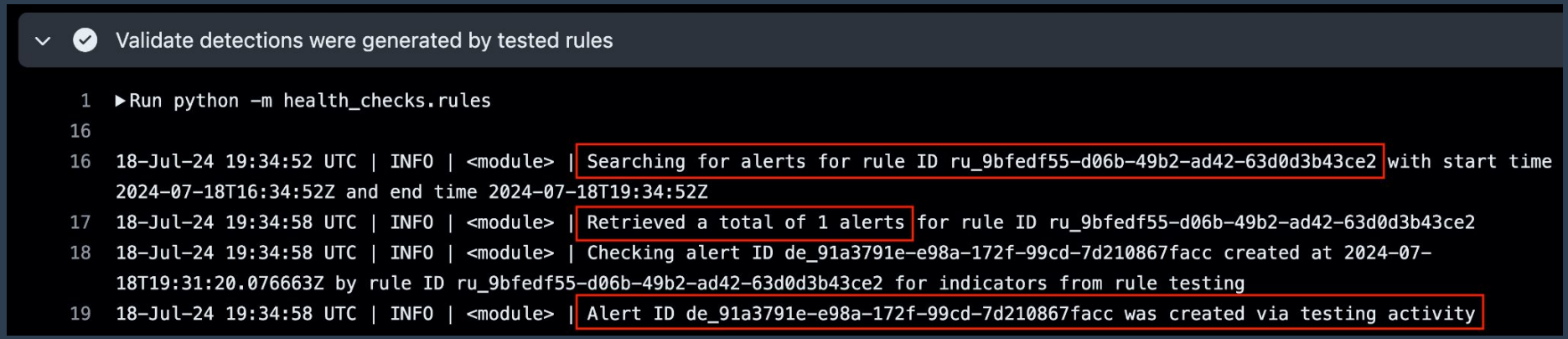
Errors should be routed to the security team to investigate



Health Check - Rule-Based Detection Validation failing

Example: Testing the GitHub Detection (4)

- Check if any of the alerts from the detection were generated by testing
  - e.g. `log_replay: true`
- Automatically close alerts generated by testing
- This approach can be applied to other detections where end-to-end, automated attack simulation isn't possible/needed



Key Takeaways

Key Takeaways

- Detection Engineering is a proactive approach for identifying attacker behavior
  - Goes beyond generic, out-of-the-box detections/signatures
  - Detect potential security incidents before they can cause significant damage
- Monitor your data pipeline to ensure that your data is available when you need it
- Test your detections to validate that you'll be alerted to malicious activity
- Teams require a diverse skill set
  - Combination of technical expertise, engineering mindset, curiosity, data analysis skills, etc

Useful Resources

Monitoring for Suspicious GitHub Activity with Google Security Operations – David French

A Recipe for Improving SecOps Detections – John Stoner

ADS Framework AI Assistant – Wade Wells

Implementing a Modern Detection Engineering Workflow – Dan Lussier

SpecterOps Training Course – Adversary Tactics: Detection

Practical Threat Detection Engineering — Megan Roddie

Thank you

David French

Staff Adoption Engineer, Google Cloud

@threatpunter

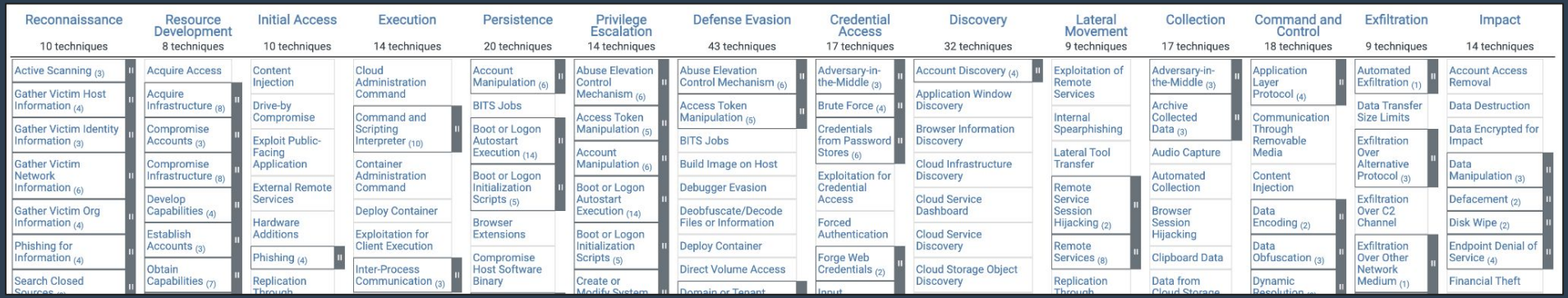
Q&A

Appendix

Tactics, Techniques, and Procedures (TTPs) Explained

What is MITRE ATT&CK?

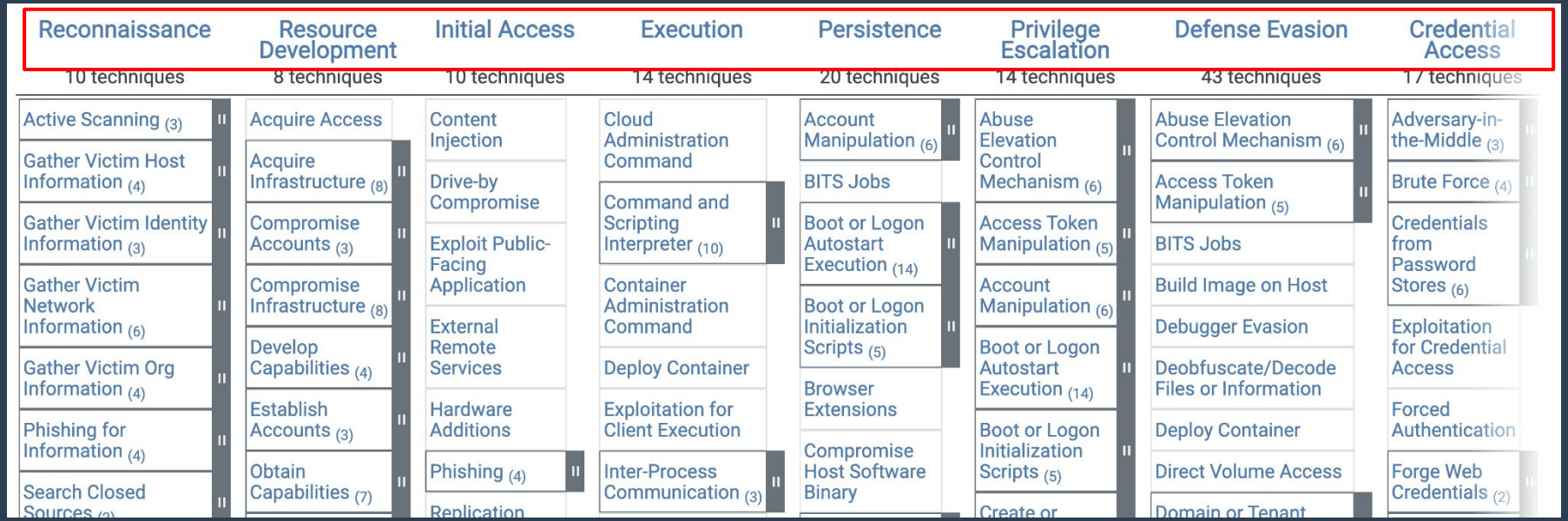
- Attack lifecycle framework – Helps us break down the phases of an attack and the techniques used in each phase
- A knowledge base of attacker behavior based on real-world observations
- Provides us with a common language to describe an attack and attacker TTPs



MITRE ATT&CK Matrix for Enterprise

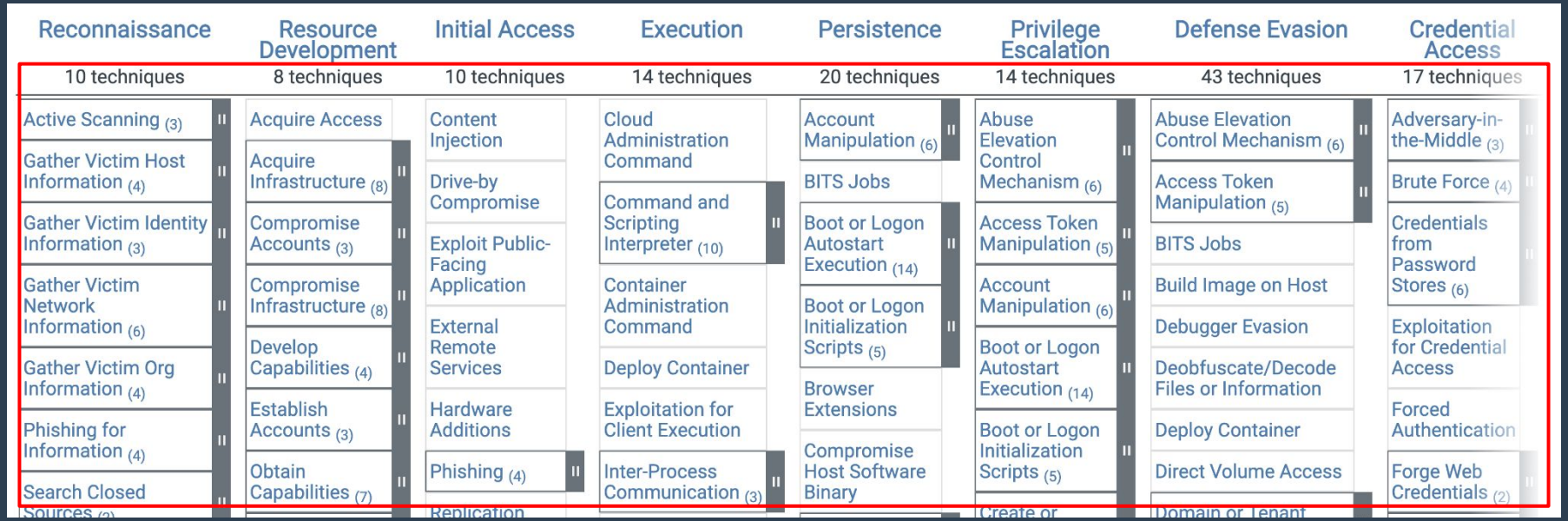
TTPs Explained: Tactics

- **Tactics** represent the **why** behind an attacker's actions
  - The goal the attacker is trying to achieve during a particular stage of an attack



TTPs Explained: Techniques

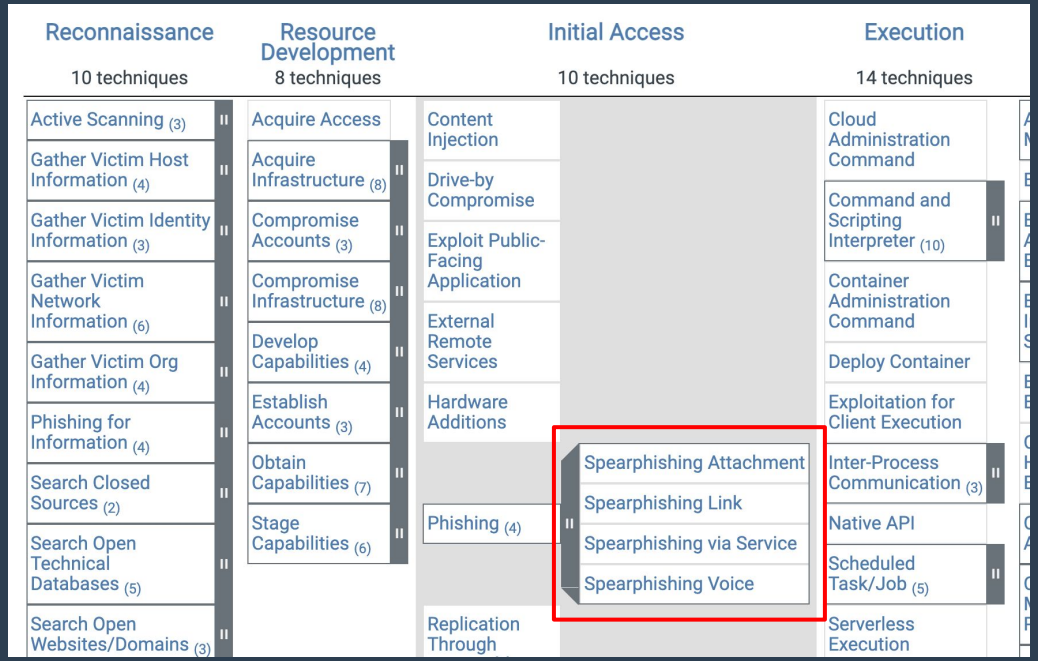
Techniques represent **how** an attacker achieves a tactical objective by performing an action



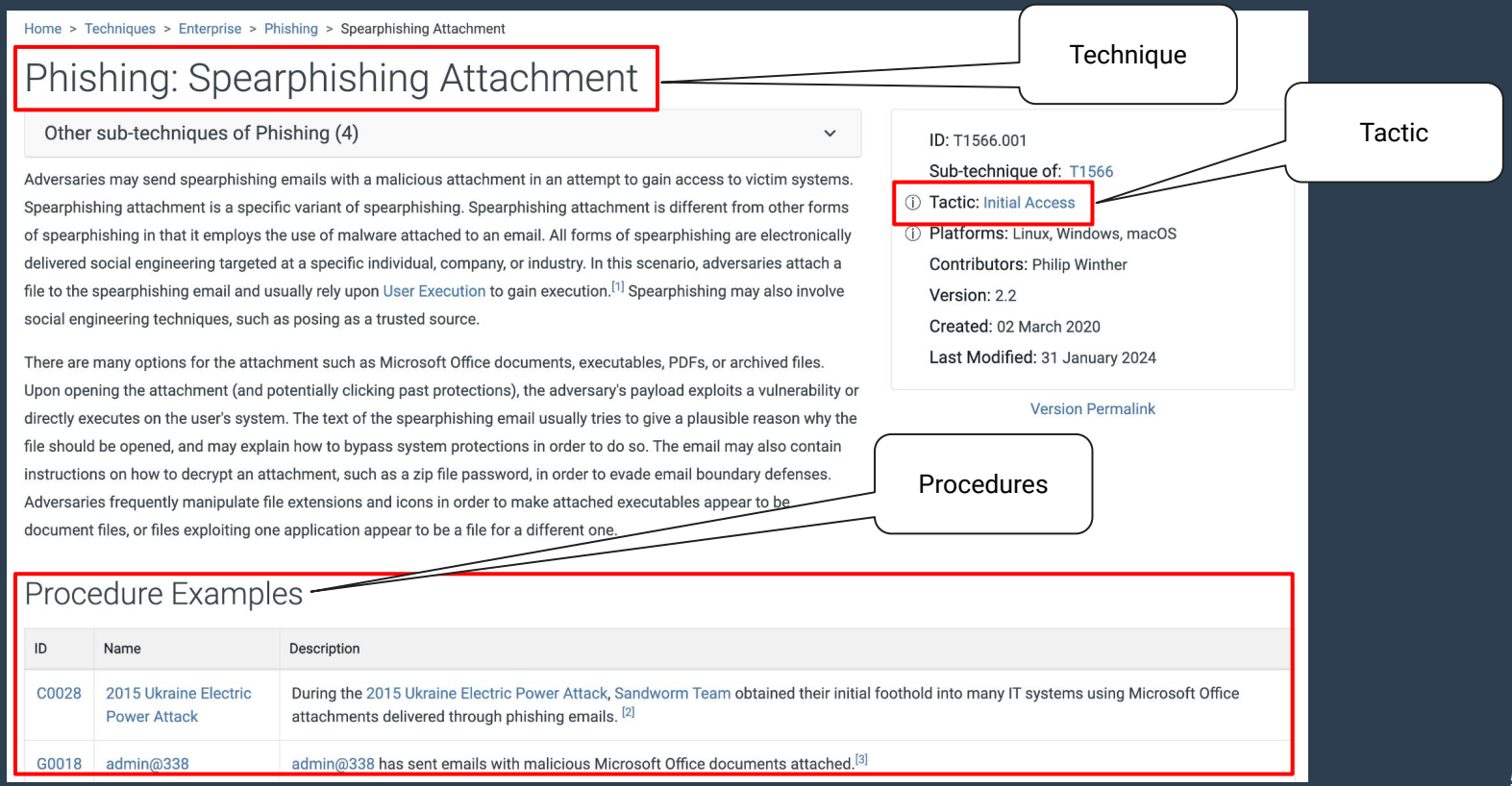
Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access
10 techniques	8 techniques	10 techniques	14 techniques	20 techniques	14 techniques	43 techniques	17 techniques
Active Scanning (3)	Acquire Access	Content Injection	Cloud Administration Command	Account Manipulation (6)	Abuse Elevation Control Mechanism (6)	Abuse Elevation Control Mechanism (6)	Adversary-in-the-Middle (3)
Gather Victim Host Information (4)	Acquire Infrastructure (8)	Drive-by Compromise	Command and Scripting Interpreter (10)	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Brute Force (4)
Gather Victim Identity Information (3)	Compromise Accounts (3)	Exploit Public-Facing Application	Container Administration Command	Boot or Logon Autostart Execution (14)	Account Manipulation (6)	BITS Jobs	Credentials from Password Stores (6)
Gather Victim Network Information (6)	Compromise Infrastructure (8)	External Remote Services	Deploy Container	Boot or Logon Initialization Scripts (5)	Boot or Logon Autostart Execution (14)	Build Image on Host	Exploitation for Credential Access
Gather Victim Org Information (4)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Browser Extensions	Boot or Logon Initialization Scripts (5)	Debugger Evasion	Forced Authentication
Phishing for Information (4)	Establish Accounts (3)	Phishing (4)	Inter-Process Communication (3)	Compromise Host Software Binary	Create or	Deobfuscate/Decode Files or Information	Forge Web Credentials (2)
Search Closed Sources (2)	Obtain Capabilities (7)	Replication				Deploy Container	
						Direct Volume Access	
						Domain or Tenant	

TTPs Explained: Sub-Techniques

Sub-Techniques provide greater detail and granularity on how a Technique might be implemented



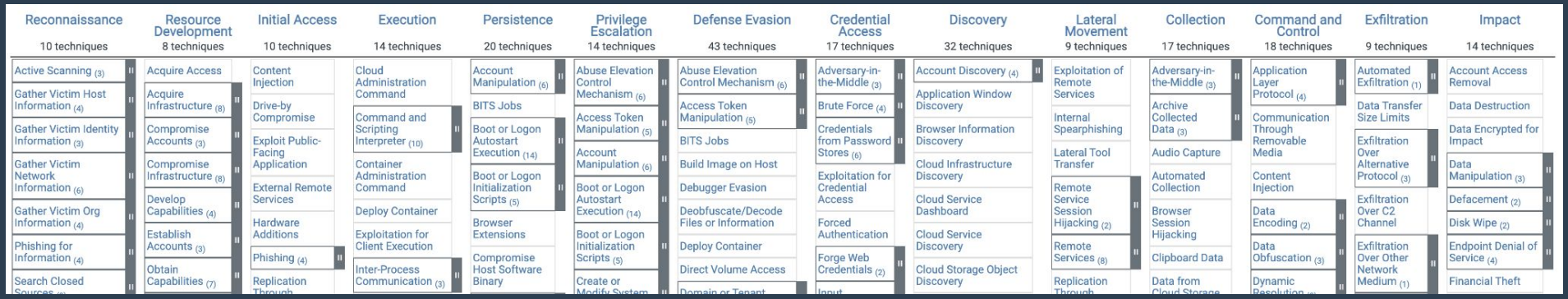
Reviewing a MITRE ATT&CK Technique



Mapping Threat Intelligence to TTPs

Mapping Threat Intel to TTPs: Why bother?

- Helps break down the phases of an attack and understand the techniques used in each phase
- Provides us with a common language to categorize observed behaviors
- Focus detection efforts on specific & relevant behaviors
- Makes it easier to understand current coverage against attacker behavior



SpecterOps Adversary Tactics: Detection Training Course
MITRE ATT&CK Matrix for Enterprise

Mapping Threat Intel to TTPs (1)

- The threat group compromises a software developer’s Okta user account via a Smishing campaign
- They’re using VPN services like Mullvad VPN, NordVPN, and ExpressVPN to mask their IP address and geolocation
- They create a GitHub Personal Access Token under the victim’s account and proceed to clone all of the GitHub repositories that the account has access to
- They’re using an open-source tool to steal the contents of code repositories in an automated fashion

Mapping Threat Intel to TTPs (2)

- Tactic: Initial Access (TA0001)
- Technique: Phishing (T1566)
- Procedures:
  - Register a domain that looks similar to the target organization's Okta SSO subdomain
    - e.g. `company-sso[.]com` or `company-okta[.]net`
  - Host a fake Okta SSO login portal using the new domain
  - Send an SMS message to the target user asking them to sign in to the fake Okta portal
  - Harvest the user's username, password, and one-time passcode (OTP)
  - Using a VPN/anonymization service, login to the target organization's Okta portal as the target user
  - Access the target web application(s) as the authenticated user

Mapping Threat Intel to TTPs (3)

- The threat group compromises a software developer's Okta user account via a Smishing campaign
- They're using VPN services like Mullvad VPN, NordVPN, and ExpressVPN to mask their IP address and geolocation
- They create a GitHub Personal Access Token under the victim's account and proceed to clone all of the GitHub repositories that the account has access to
- They're using an open-source tool to steal the contents of code repositories in an automated fashion

Mapping Threat Intel to TTPs (4)

- Tactic: Persistence (TA0003)
- Technique: Create Account (T1136) – Not always a perfect fit, but that's ok
- Procedures:
  - Create a new GitHub Personal Access Token (PAT) under the compromised user's account
  - Grant the PAT access to all of the code repositories that the user has access to

---

- Tactic: Collection (TA0009)
- Technique: Data from Information Repositories: Code Repositories (T1213.003)
- Procedures:
  - Use the open source tool, "ghorg" to clone all of the GitHub code repositories that the PAT has access to

Preventative Security Controls

Preventative Security Controls

- Configure a Personal Access Token (PAT) policy for your GitHub organization(s)
  - Enforce an approval policy
- Encourage users to create short-lived tokens for specific tasks
- Limit permissions granted to tokens based on principle of least privilege
- Configure an IP allow list for your GitHub organization (if possible)
- Require security keys for GitHub two-factor-authentication
- Monitor for secrets in code, CI/CD scripts, wikis, etc
  - Attackers will find them and use them against you