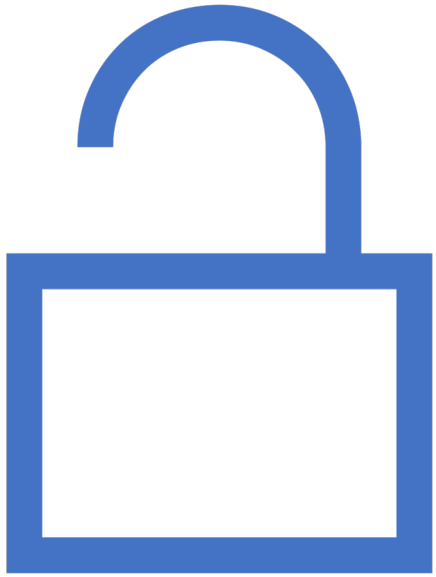


Secret Leakage

Control for Secret Leakage in an Age of Cloud

Liang Huang & Chandler Vaughn

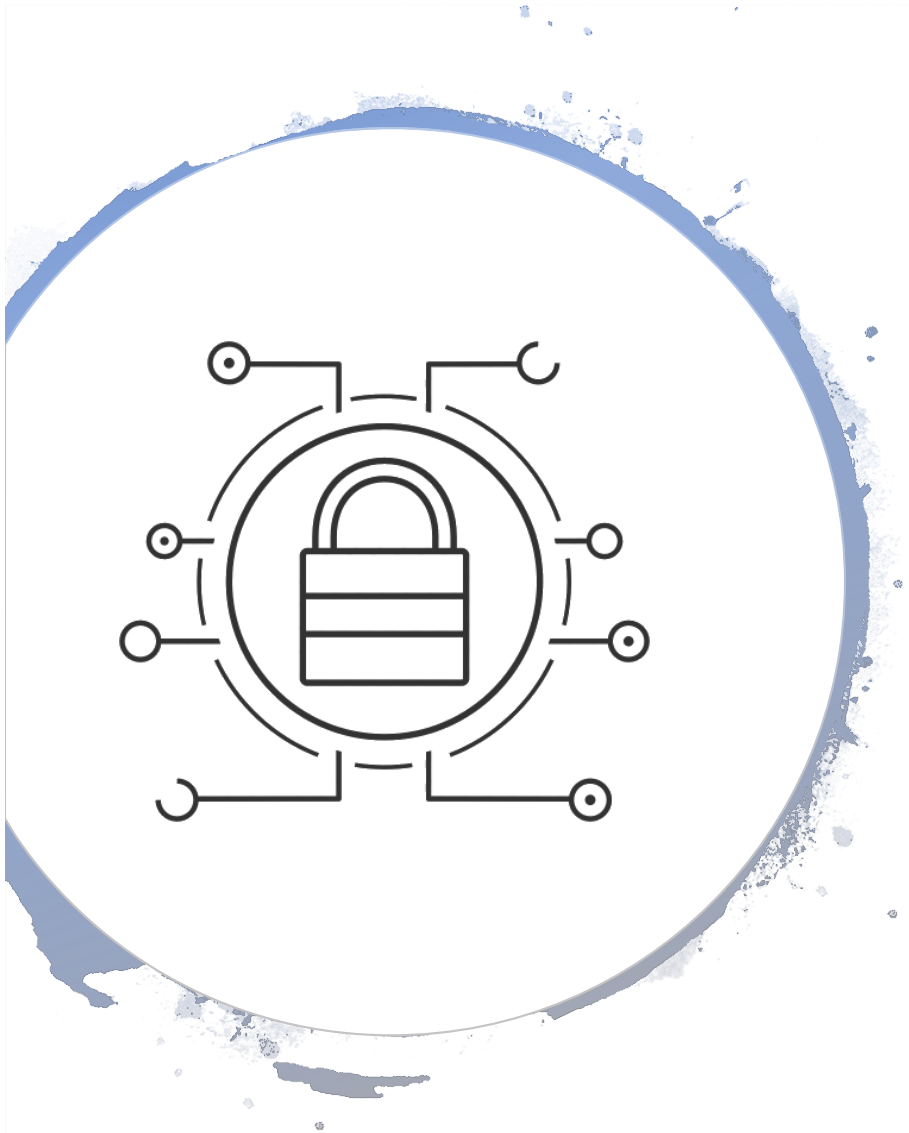
Executive Summary

We understand the extent by which internal corporate secrets and Cloud-related credentials are leaking out into the public domain via code commits into code repositories.

We harvest source code files via GitHub harvesting and analyze source code for credential and login patterns for several popular subsystems and platforms.

We find tangible examples of credential leakages through our study, in essence over 10,000 distinct secrets in span of less than a month.

Finally, we present a framework and reference architecture to help mitigate this problem proactively, while enabling core tenants to good secret management.



Public Source Code Repositories Bring Both Convenience And A Host Of New Problems

- Proprietary source code leaks
- Repository hijacking for ransom
- Credentials from employees taken by 3rd party to supply malicious source code updates
- Implanted Trojans by bad actors
- ***Accidental secret leakages***

Secret Leaks As It Relates to Source Code

Questions of Secret Leakage

- How pervasive is secret leakage in terms of accidental leaks of potentially sensitive secrets to source code repositories like GitHub?
- How easy is it to locate secrets if one were to try?
- What, if anything, is there to do proactively to prevent secret leakage?

Questions of Control

- How is key rotation enabled in the company as employees leave the company?
- If there is a security breach of secret leakage, how long does it potentially take to mitigate the risk?
- Is there an ability to audit who has accessed a secret, and when it was accessed?
- Is there a methodology and structure to implement different levels of secrets that limit authorization to only what is required for an application, service, or employee? How is this accomplished with a distributed set of developers?
- Is the responsibility of keeping secrets secret in the hands of each individual employee (where its hard or impossible to control), or is it controlled and verified centrally?

Current Research

Problem Sizing

- V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani, "**Detecting and Mitigating Secret-Key Leaks in Source Code Repositories**," Mining Software Repositories, 2015.
- Meli, M., Mcniece, M. R., & Reaves, B. (2019). "**How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories**," Proceedings 2019 Network and Distributed System Security Symposium. doi:10.14722/ndss.2019.23418
- R. Shu, X. Gu, and W. Enck, "**A Study of Security Vulnerabilities on Docker Hub**," CODASPY, 2017.
- /@cdavis_. (2019, February 06). "**Discovering keys and secrets"** - CryptoMove Blog: Moving Target Data Protection. Retrieved July 16, 2019, from https://blog.cryptomove.com/ discovering-keys-and-secrets-855f6e857a37
- D. Bourke. (2017, Oct.) "**Breach Detection at Scale."** [Online]. Available: https://developer.atlassian.com/blog/2017/10/project-spacecrab-breach-detection/

Monitoring Tools

- GitRob
- TruffleHog
- Custom scripts

Developer Tools

- GitSecrets
- Numerous plugins
- Gitignore files

Secret Managers

- Hashicorp Vault
- Docker Secrets (Docker)
- Istio (Kubernetes)

How pervasive is secret leakage on GitHub?

Visible Secrets

Liang Huang

How Big Is The Problem?

- We approach the issue of sizing in phases:
  - Phase 1: A GitHub file harvesting phase
  - Phase 2: A secret harvesting phase
  - Phase 3: Results analysis

Phase 1: GitHub File Harvesting

- Phase 1 includes searching for any of 85 keyword identifiers
- Sorting results by the index so the most recent files are shown (1000 file limit)
- Deduplicating files, and downloading potential candidate files through the GitHub API
- Adjusting for API rate limits

GitHub API Search Keywords

objectrocket.com:	ACCESS_SECRET=
mongodb.net:	ACCESS_TOKEN=
access_token	ACCOUNT_SID=
access_secret	AWS-KEY=
api_key	AWS-SECRETS=
client_secret	AWS.config.accessKeyId=
consumer_secret	AWS.config.secretAccessKey=
customer_secret	AWSACCESSKEYID=
user_secret	AWSSECRETKEY=
secret_key	AWS_ACCESS=
-----BEGIN RSA PRIVATE KEY-----	AWS_ACCESS_KEY=
-----BEGIN EC PRIVATE KEY-----	AWS_ACCESS_KEY_ID=
-----BEGIN PRIVATE KEY-----	AWS_CF_DIST_ID=
-----BEGIN PGP PRIVATE KEY BLOCK-----	AWS_DEFAULT
-----BEGIN OPENSSH PRIVATE KEY-----	AWS_DEFAULT_REGION=

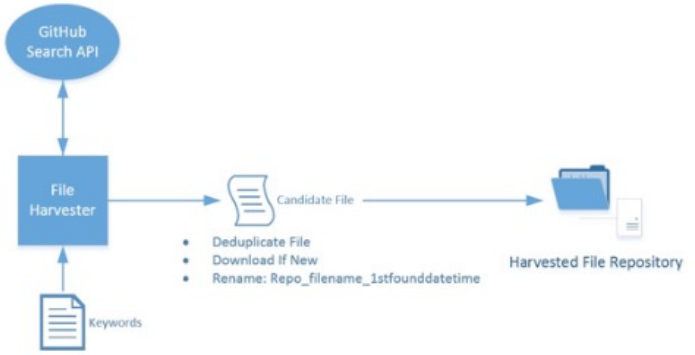


Figure 1: Phase One: Candidate File Harvesting

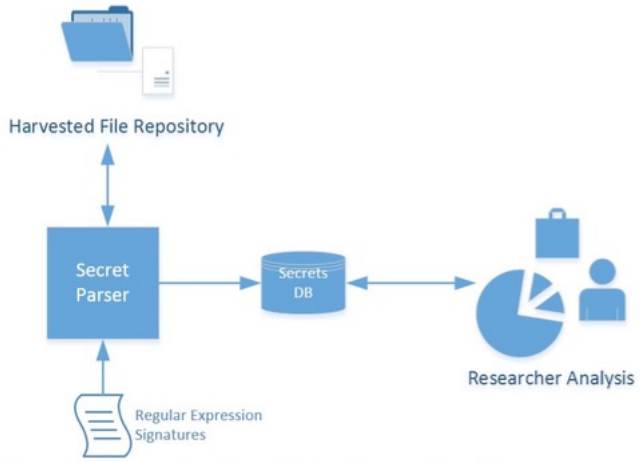


Figure 2: Phase Two: Secret Extraction and Analysis

Regular Expression Signatures

Slack Token:	(xox[p|b|o|a]-[0-9]{12}-[0-9]{12}-[0-9]{12}-[a-z0-9]{32}),
RSA private key:	-----BEGIN RSA PRIVATE KEY-----,
SSH (OPENSSH) private key:	-----BEGIN OPENSSH PRIVATE KEY-----,
SSH (DSA) private key:	-----BEGIN DSA PRIVATE KEY-----,
SSH (EC) private key:	-----BEGIN EC PRIVATE KEY-----,
PGP private key block:	-----BEGIN PGP PRIVATE KEY BLOCK-----,
Facebook Access Token:	EAACEdEose0cBA[0-9A-Za-z]+,
Facebook Oauth:	[f|F][a|A][c|C][e|E][b|B][o|O][o|O][k|K].*['|\][0-9a-f]{32}['|\]",
Twitter Oauth:	[t|T][w|W][i|I][t|T][t|T][e|E][r|R].*['|\][0-9a-zA-Z]{35,44}['|\]",
Generic Secret:	[s|S][e|E][c|C][r|R][e|E][t|T].*['|\][0-9a-zA-Z]{32,45}['|\]",
Generic API Key:	9]\ 5 5y,

.
.
.

Phase 2: Secret Harvesting

- Phase 2 includes searching for fixed form secrets, classifying each, and deduplication
- 34 regular expression searches performed on each file

Secret Harvesting Results

Days run = 25
(06/12/2019 – 07/14/2019)

Candidate files harvested = 131,491

Repositories represented = 22,700

Source code size = 4.2 Gigabytes

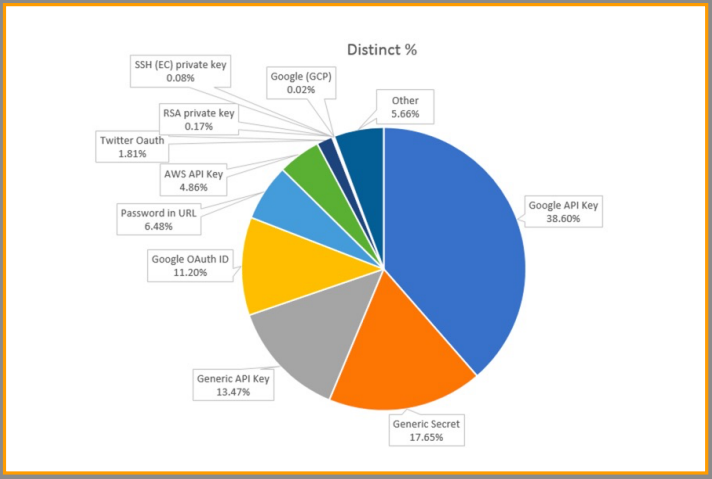
Located

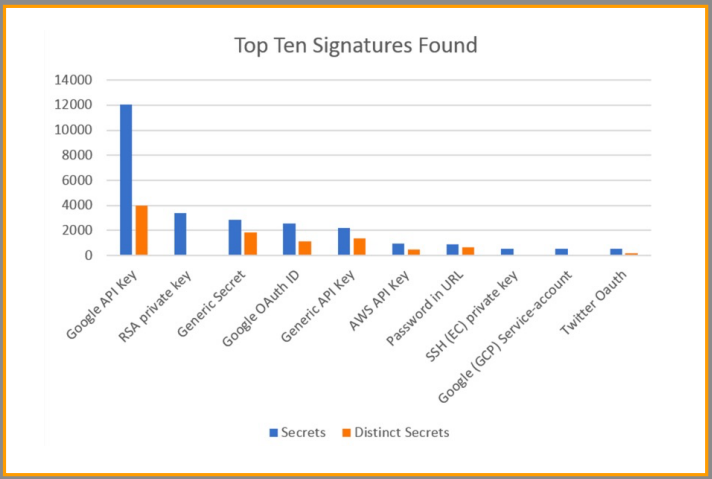
Secrets	= 27,847
Distinct Secrets	= 10,388

Secret Harvesting Results

Signature	Signature:Filename	Secret
AWS API Key	AWS API Key:03sarath_AW[redacted]19-07-14 21:12:23	['AKIA5N6LFCOKAA[redacted]']
Generic Secret	Generic Secret:08zhangyi_[redacted]t_sina_version.py_2019-07-01 02:28:08	["SECRET_KEY = 'E[redacted]"]
Google API Key	Google API Key:090233847[redacted]wing.java_2019-06-16 01:06:23	['AIzaSyB-ccRdQ8_[redacted]vQcePo60NMHLLktX0pg']
SSH (OPENSSH) private key	SSH (OPENSSH) private key[redacted]%96-----BEGIN%20OPENSSH%20PRIVATE%20KEY-----_2019-06-12 16:56:45	['-----BEGIN OPEN[redacted]']
RSA private key	RSA private key:0cherry_20[redacted]25:14	['-----BEGIN RSA P[redacted]']
Google API Key	Google API Key:0dbtech_co[redacted]Container.js_2019-06-27 21:26:50	['AIzaSyBI_k_ZRPY[redacted]']
Generic Secret	Generic Secret:0v3rWatch_[redacted]2018.html_2019-06-16 08:48:15	["secret_key = <spa[redacted]"secret_key = '9
RSA private key	RSA private key:0xRick_0xF[redacted].md_2019-06-15 18:15:38	['-----BEGIN RSA P[redacted]']
Google API Key	Google API Key:0xced_XCD[redacted]UnratedVideo.json_2019-06-21 21:05:03	['AIzaSyAO_FJ2Slq[redacted]']
Google API Key	Google API Key:0xced_XCD[redacted]Video.json_2019-06-21 21:04:59	['AIzaSyAO_FJ2Slq[redacted]']
RSA private key	RSA private key:0xdead4ea[redacted]7-01 02:41:04	['-----BEGIN RSA P[redacted]']
Google OAuth ID	Google OAuth ID:0xkag_alp[redacted]07:27	['624395471329-0c[redacted]com']
PGP private key block	PGP private key block:100p[redacted]l_2019-06-12 16:52:04	['-----BEGIN PGP P[redacted]']
Generic Secret	Generic Secret:113169503[redacted].java_2019-06-17 04:21:13	['SECRET = 260f81[redacted]0d807dd3efa0378208b4 ']
RSA private key	RSA private key:116378184[redacted].pem_2019-07-04 01:00:39	['-----BEGIN RSA P[redacted]']
RSA private key	RSA private key:116378184[redacted]y.pem_2019-07-04 01:02:06	['-----BEGIN RSA P[redacted]']
Generic Secret	Generic Secret:117534172[redacted]019-06-20 02:22:22	['secretKey = S4xF[redacted]']
AWS API Key	AWS API Key:11wicketsfan[redacted]troller-25-04-2019.php_2019-06-21 12:04:10	['AKIAIMRMGR2LP[redacted]']
Generic Secret	Generic Secret:11wicketsfa[redacted]ontroller-25-04-2019.php_2019-06-21 12:04:10	['secret = 59a5ef4[redacted]4cb8d3080c2d2a10cd348a25119171
AWS API Key	AWS API Key:11wicketsfan[redacted]troller.php-old_2019-06-21 12:04:07	['AKIAIMRMGR2LP[redacted]']
Generic Secret	Generic Secret:11wicketsfa[redacted]ontroller.php-old_2019-06-21 12:04:07	['secret = 59a5ef4[redacted]4cb8d3080c2d2a10cd348a25119171
RSA private key	RSA private key:123songfe[redacted].client.key_2019-07-15 16:15:32	['-----BEGIN RSA P[redacted]']
RSA private key	RSA private key:123songfe[redacted].server.key_2019-07-15 16:15:29	['-----BEGIN RSA P[redacted]']
Google API Key	Google API Key:1298se_Co[redacted]6-15 18:26:24	['AIzaSyB9tiUIhlC_[redacted]']
Google API Key	Google API Key:1298se_Co[redacted]06-29 01:46:43	['AIzaSyB9tiUIhlC_[redacted]']
RSA private key	RSA private key:131248468[redacted]2019-06-19 04:25:10	['-----BEGIN RSA P[redacted]']
SSH (OPENSSH) private key	SSH (OPENSSH) private key[redacted]9-06-12 17:06:35	['-----BEGIN OPEN[redacted]']
Generic Secret	Generic Secret:138172061[redacted]_2019-06-18 08:58:35	["secret_key = 'Dq[redacted]"]
Google API Key	Google API Key:13RedFox_[redacted]er.ejs_2019-06-27 16:06:12	['AIzaSyBs-s7-go0[redacted]']

Secret Harvesting Results (rollup)





Signatures	Secrets	Distinct Secrets
Google API Key	12081	4010
RSA private key	3406	18
Generic Secret	2848	1834
Google OAuth ID	2540	1163
Generic API Key	2207	1399
AWS API Key	937	505
Password in URL	925	673
SSH (EC) private key	536	8
Google (GCP) Service-account	522	2
Twitter Oauth	514	188
SSH (OPENSSH) private key	311	9
GitHub	191	122
PGP private key block	155	8
SSH (DSA) private key	149	5
Facebook Oauth	147	122
Google OAuth Access Token	140	115
Google Oauth	79	76
Slack Webhook	54	46
Twitter Access Token	18	18
Twilio API Key	18	12
MailGun API Key	13	13
Stripe Standard API Key	12	7
Square OAuth Secret	11	11
Square Access Token	7	7
Amazon MWS Auth Token	7	4
PayPal Braintree Access Token	6	4
MailChimp API Key	5	5
Facebook Access Token	4	4
Slack Token	3	3
Heroku API Key	1	1
Grand Total	27847	10388

Mitigations

Keeping Secrets Secret

Chandler Vaughn

Mitigations:
Keeping Secrets Secret

PROBLEM SCOPE: We are really scoping the discussion to managing a diverse set of sensitive credentials and managing it such that a diverse and potentially distributed set of individual developers and users can potentially leverage those sensitive credentials to build and operate applications.

For an IT department to maintain proper control over these assets, it leads to one unifying conclusion:

Secrets must be centralized if they are to be controlled

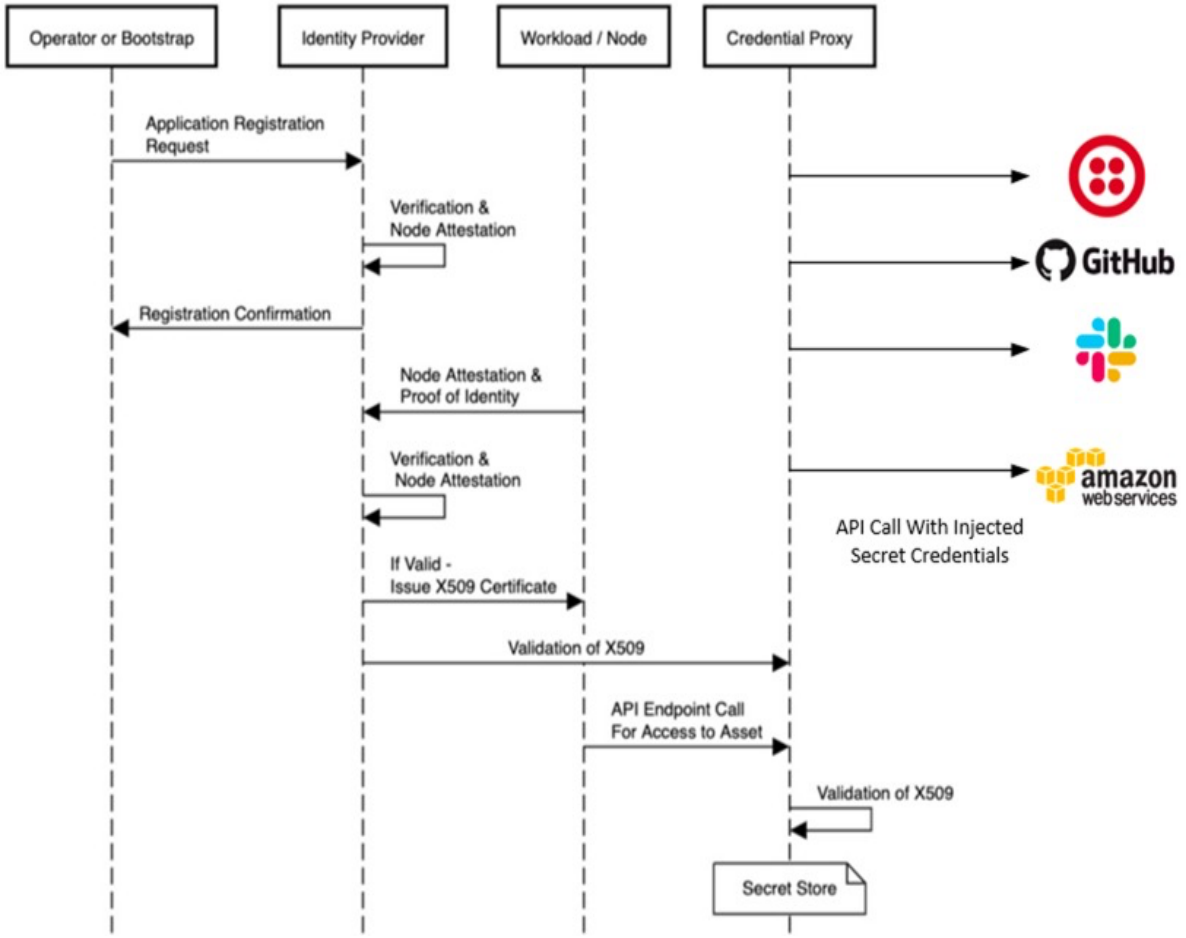
Mitigations: Keeping Secrets Secret

We would want a system that provides:

- A simple way to execute **key and password rotation**
- An **audit trail** of who could potentially access a secret, and when they actually do access a secret
- **Minimization to secret sprawl and leakage**
- **Encryption** at rest and in transit of sensitive data
- **Access control**
- **Strong guarantees** for identity, access control, audit logging, and non-repudiation

Proposed Solution: Centralized Secret Store w/ Credential Proxy

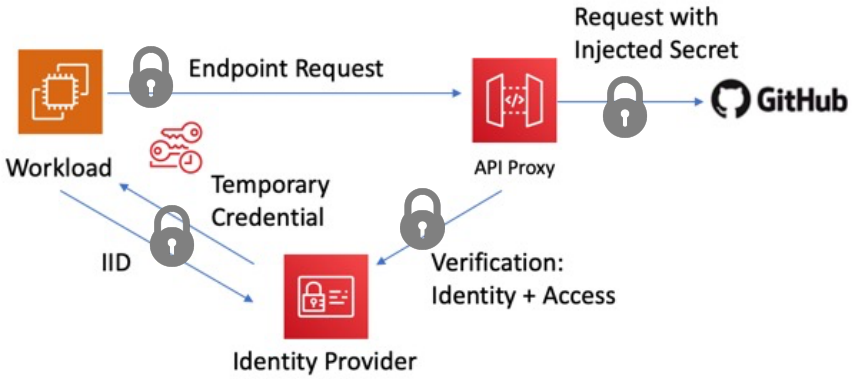
- **Application Registration:** A workload or application is registered with the identity provider and receives an X.509 certificate of identity if approved.
- **Authentication:** A workload, node, or application authenticates to an identity provider either utilizing a cryptographic key pair or in the case of specific infrastructure providers like Amazon Web Services (AWS) an Instance Identification Document (IID) or similar. Once authenticated, the workload asks 'who am I?'
- **Identification Documentation:** Based on access controls, a X.509 certificate of identity is then forwarded not only to the workload, but also to the proxy system so that it can do validation.
- **Proxy Request:** If a workload needs access to resources that potentially need sensitive secrets, the workload sends a request to the proxy, for a specific API endpoint of the proxy, along with the X.509 certificate proving its identity.
- **Proxy Intermediation:** The proxy receives the request, validates the certificate of the requestor, and then if valid, locates the necessary credential secret, and makes an external request to the external system with secrets injected into the request. This process validates both the validity and integrity of the request.
- **Return Passthrough:** Assuming success, the proxy returns the payload back to the calling workload.



Test Implementation (AWS --> GitHub)

Our test system provides:

- A complete seal on potential secret leakage - Secured and hidden API keys
- Full encryption of data at rest and in transit, provided by TLS and AWS provided AES 256
- A simple way to rotate the API key without impacting production code
- Ample logging of access to resources
- Strong guarantees for node identification, access control, audit logging, and non-repudiation for the requesting node or workload.



- AWS utilizes PKCS7 signatures and the Cryptographic Message Syntax (CMS) standard. This is equally valid for this reference architecture as the outlined usage of X.509 for signatures earlier. PKCS7 acts as a container to X.509 certificates.

PKCS7: Cryptographic Message Syntax Standard — public keys with proof of identity for signed and/or encrypted message for PKI

```yaml
ProxyMethod:
  Type: AWS::ApiGateway::Method
  Properties:
    ResourceId: !Ref ProxyResource
    RestApiId: !Ref RestApi
    # Enforce that requests made to this endpoint need to be signed
    AuthorizationType: AWS_IAM
    HttpMethod: ANY
    RequestParameters:
      method.request.path.proxy: true
    Integration:
      IntegrationHttpMethod: ANY
      Type: HTTP_PROXY
      Uri: !Sub https://api.github.com/{proxy}
      RequestParameters:
        integration.request.path.proxy: method.request.path.proxy
        # Credential Injection
        integration.request.header.Authorization: !Sub "'token ${GitHubAccessKey}'"
        integration.request.header.Accept-Encoding: "'application/json'"
```

- Proxy enforces signature verification
- Performs the heavy lifting for secret injection
- Only an administrator has access to the secret in this implementation

Proxy Gateway

Application Code

- **Notice the generic endpoint – no credentials required in code that could potentially leak**
- Security hinges on node and workload attestation and the identity framework, along with digital signatures

Private Repo: https://github.com/vaughnch/security-proxy

```
1  #this proxy will obfusate credentials for the target service
2  #credentials are stored centrally to the proxy
3
4  import re
5  import urlparse
6  import requests
7  from aws_requests_auth.aws_auth import AWSRequestsAuth
8  from aws_requests_auth import boto_utils
9
10
11 #Proxy URL
12 APP_PROXY_URL = 'https://ynxga79izl.execute-api.us-east-1.amazonaws.com/dev/repos/vaughnch/security-proxy'
13
14
15 def get_aws_auth(url):
16     # The following signs the request
17     api_gateway_netloc = urlparse.urlparse(url).netloc
18     api_gateway_region = re.match(
19         r"[a-z0-9]+\.execute-api\.(.+)\.amazonaws\.com",
20         api_gateway_netloc
21     ).group(1)
22
23     return AWSRequestsAuth(
24         aws_host=api_gateway_netloc,
25         aws_region=api_gateway_region,
26         aws_service='execute-api',
27         **boto_utils.get_credentials()
28     )
29
30
31 list_issues_response = requests.get(
32     url=APP_PROXY_URL,
33     auth=get_aws_auth(APP_PROXY_URL)
34 )
35
36 print list_issues_response.json()
```

```
{u'issues_url': u'https://api.github.com/repos/vaughnch/security-proxy/issues{/number}',
u'deployments_url': u'https://api.github.com/repos/vaughnch/security-proxy/deployments',
u'stargazers_count': 0, u'forks_url': u'https://api.github.com/repos/vaughnch/security-proxy/forks',
'mirror_url': None, u'subscription_url': u'https://api.github.com/repos/vaughnch/security-proxy/subscript
u'notifications_url': u'https://api.github.com/repos/vaughnch/security-proxy/notifications{?since,all,part
u'collaborators_url': u'https://api.github.com/repos/vaughnch/security-proxy/collaborators{/collaborator}
u'updated_at': u'2019-07-22T22:11:27Z',
u'private': False,
u'pulls_url': u'https://api.github.com/repos/vaughnch/security-proxy/pulls{/number}',
u'disabled': False,
u'issue_comment_url': u'https://api.github.com/repos/vaughnch/security-proxy/issues/comments{/number}',
u'labels_url': u'https://api.github.com/repos/vaughnch/security-proxy/labels{/name}',
u'has_wiki': True,
u'full_name': u'vaughnch/security-proxy',
u'owner': {u'following_url': u'https://api.github.com/users/vaughnch/following{/other_user}',
u'events_url': u'https://api.github.com/users/vaughnch/events{/privacy}',
u'avatar_url': u'https://avatars0.githubusercontent.com/u/13335453?v=4',
u'url': u'https://api.github.com/users/vaughnch',
u'gists_url':
u'https://api.github.com/users/vaughnch/gists{/gist_id}',
u'html_url': u'https://github.com/vaughnch',
u'subscriptions_url': u'https://api.github.com/users/vaughnch/subscriptions',
u'node_id': u'MDQ6VXNlcjEzMzM1NDUz',
u'repos_url': u'https://api.github.com/users/vaughnch/repos',
u'received_events_url': u'https://api.github.com/users/vaughnch/received_events',
u'gravatar_id': u'',
u'starred_url': u'https://api.github.com/users/vaughnch/starred{/owner}{/repo}',
u'site_admin': False,
u'login': u'vaughnch',
u'type': u'User',
u'id': 13335453
```

Proxy Response

- The result is a proxied GitHub response utilizing hidden credentials that are injected at runtime
- Secrets are injected only after verification of identity and access rights
- Secrets are:
  - Never exposed
  - Easily rotated or expired
  - Audited
  - Encrypted
  - Supplied with strong guarantees

Conclusion

The issues around secret leakage are staggering for todays capabilities.

However, with a centralize secret management approach, and through proper identity and verification frameworks leveraging modern technologies like PKI, IT departments can effectively achieve security control on secrets.

Future research should explore the newer emerging standards of **Secure Production Identity Framework for Everyone** (SPIFFE), the **SPIRE server** which is a production implementation of SPIFFE, and **X.509-SVID** as a way to ensure secure, yet lightweight identity for workloads and applications.

Questions

Thank You