Identity, Access and Secret Management

Ryan Lane - Jun 13, 2016

Example service

Example service

Example service

Example service Multiple environments

SaaS

production services

- fastly
- twilio
- digicert
- stripe
- markmonitor

productivity

- google
- dropbox
- jira
- confluence
- slack

monitoring

- splunk
- thousandeyes
- datadog
- pagerduty

helpdesk

- zendesk

analytics

- kiss
- looker

dev/ci

- github
- travisci

blog/cms

- medium
- squarespace

IaaS

infrastructural

- ec2
- s3
- iam
- ecr
- vpc

data

- rds
- redshift
- dynamodb
- elasticache
- elasticsearch
- datapipeline

... 47 more various

Internal Services

infrastructural

- kubernetes
- ssh bastion
- etcd
- vpn
- stackstorm
- airflow
- confidant
- hubot
- zipkin
- sentry

production

- so
- many
- services...
- ...
- ...

Identity

- SaaS/Web Services
  - SSO with API
  - SSO without API
  - No SSO with API
  - No SSO, without API
- EC2 user identity

SSO: SAML

Service Provider
User Agent
Identity Provider
1
Request target resource
(Discover the IdP)
2
Respond with XHTML form
3
Request SSO Service
(Identify the user)
4
Respond with XHTML form
5
Request Assertion Consumer Service
6
Redirect to target resource
7
Request target resource
8
Respond with requested resource

SSO: SAML

Providers:

- Google
- Onelogin
- Okta
- Ping Identity
- ...

SSO: OpenID

Dead end.

SSO: OpenID Connect (OAuth2 + OpenID)

Providers:

- Google
- Facebook
- Okta
- Ping Identity
- ...

No SSO...

No SSO with API

```
{% for name, user in pillar.users.items() %}
{% if user.get("disabled", False) %}
{% set action = "absent" %}
{% else %}
{% set action = "present" %}
{% endif %}
Ensure pagerduty user {{ name }} is {{ action }}:
  pagerduty_user.{{ action }}:
    - name: "{{ user.full_name }}"
    - email: {{ user.email }}

Ensure github user {{ name }} is {{ action }}:
  github.{{ action }}:
    - name: "{{ user.github.name }}"
    - email: {{ user.email }}
    - fullname: "{{ user.full_name }}"
{% endfor %}
```

SSO with API

```
{% for name in pillar['iam_groups'] %}
Ensure iam group {{name}} is present:
  boto_iam.group_present:
    - name: {{name}}
    - users:
{% for username, user in pillar['users'].items() %}
        - {{username}}
{% endfor %}
    - policies_from_pillars:
        - 'iam_groups:{{ name }}'
{% endfor %}
```

No SSO, No API

Secrets Files

```
Twilio:
  User: own
  Password: me
  Security questions:
    What's your pet's name? fluffles

Stripe:
  User: lol
  Password: what?
  Security questions:
    Why do we even? We don't even.
...
```

credentials.txt

No SSO, No API

Password Management.

What about LDAP?

Die In a Fire, LDAP

nsscache

Die In a Fire, LDAP

Access controls

Ensuring least privilege

Access Control

Access Control

Access Control
Docker

Access Control
Docker

SSH management

SSH management

SSH management

Secrets

Limiting footprint, scope, and access.

EC2 instances
repo a
repo b
repo c
service-a
service-b
service-c
twilio
stripe
zendesk

Hardcoded Secrets

```
# Shady looking variables, with high entropy values
ACCESS_TOKEN = 'AKIAlWxxtmzKD78tmCP8tIcQ'
ACCESS_TOKEN_SECRET = 'l6q3ztS5pMz5i62WvovPyrAAjRfczKIHtmv'

# Innocent looking variable, but shady looking dict key with a high
# entropy value.
INNOCENT_DICT = {'secret1': 'O80NhqsbKahKnqioQdv7'}

# Safely defined variable, that's a false positive
API_TOKEN_PASSWORD_KEY = os.getenv('SECRETS_API_TOKEN_PASSWORD_KEY')  # nosec


def check_password(password):
    # Comparison with a high entropy string
    if password == 'FMwoP5Bg0FZiIkLRgoaj':
        return True
    # List with high entropy strings
    elif password in ['ZzjpTWRe9SnidDb7d2MR', 'pmHFdYrNYvOVQWqYjDgL']:
        return True
    return False


# Function definition with a shady looking kwarg and a high
# entropy value.
def call_api(api_key='o7fopdjCQ8uoXbpaVEax'):
    # function call with an argument with high entropy.
    expiration = api.call(api_key, 'Mqgk65osbzRYESUD47hd')
    # function call with a high entropy string, but a safe
    # looking caller and a safe looking variable target
    expiration = expiration.strftime('%Y%m%dT%H%M%SZ')
```

Secret Management

GPG

gpg
repo a
repo b
repo c
EC2 instances
service-a
service-b
service-c
twilio
stripe
zendesk

Sneaker

kms
S3
EC2 instances
repo a
repo b
repo c
service-a
service-b
service-c
twilio
stripe
zendesk

Vault

vault
db
EC2 instances
repo a
repo b
repo c
service-a
service-b
service-c
twilio
stripe
zendesk

Confidant

dynamodb
confidant
KMS
EC2 instances
repo a
repo b
repo c
service-a
service-b
service-c
twilio
stripe
zendesk

KMS Auth Token

```
token = kms_client.encrypt(
    KeyId='alias/authnz',
    Plaintext=(
        '{"not_before":"20160613T100000Z",'
        '"not_after":"20160613T110000Z"}'
    ),
    EncryptionContext={
        'from': 'service-a',
        'to': 'service-b',
        'user_type': 'service'
    }
)['CiphertextBlob']
```

IAM policy

```
‘confidant-authnz-encrypt’:
  Version: ‘2012-10-17’
  Statement:
    - Action:
        - ‘kms:Encrypt’
      Effect: ‘Allow’
      Resource: ‘arn:aws:kms:us-east-1:1234:key/...’
      Condition:
        StringEquals:
          ‘kms:EncryptionContext:from’: ‘service-a‘
          ‘kms:EncryptionContext:user_type’: ‘service’
       StringLike:
          ‘kms:EncryptionContext:to’: ‘*’
```

IAM policy

```
'confidant-authnz-decrypt':
  Version: '2012-10-17'
  Statement:
    - Action:
        - 'kms:Decrypt'
      Effect: 'Allow'
      Resource: 'arn:aws:kms:us-east-1:1234:key/...'
      Condition:
        StringEquals:
          'kms:EncryptionContext:to': 'confidant-production'
          'kms:EncryptionContext:user_type': '*'
       StringLike:
          'kms:EncryptionContext:from': '*'
```

Service to service auth

User to service auth

Bandit

```
# Shady looking variables, with high entropy values
ACCESS_TOKEN = 'AKIAlWxxtmzKD78tmCP8tIcQ'
ACCESS_TOKEN_SECRET = 'l6q3ztS5pMz5i62WvovPyrAAjRfczKIHtmv'

# Innocent looking variable, but shady looking dict key with a high
# entropy value.
INNOCENT_DICT = {'secret1': 'O80NhqsbKahKnqioQdv7'}

# Safely defined variable, that's a false positive
API_TOKEN_PASSWORD_KEY = os.getenv('SECRETS_API_TOKEN_PASSWORD_KEY')  # nosec


def check_password(password):
    # Comparison with a high entropy string
    if password == 'FMwoP5Bg0FZiIkLRgoaj':
        return True
    # List with high entropy strings
    elif password in ['ZzjpTWRe9SnidDb7d2MR', 'pmHFdYrNYvOVQWqYjDgL']:
        return True
    return False


# Function definition with a shady looking kwarg and a high
# entropy value.
def call_api(api_key='o7fopdjCQ8uoXbpaVEax'):
    # function call with an argument with high entropy.
    expiration = api.call(api_key, 'Mqgk65osbzRYESUD47hd')
    # function call with a high entropy string, but a safe
    # looking caller and a safe looking variable target
    expiration = expiration.strftime('%Y%m%dT%H%M%SZ')
```

Bandit

```
Run metrics:
        Total lines of code: 35
        Total lines skipped (#nosec): 1
        Total issues (by severity):
                Undefined: 0
                Low: 4
                Medium: 0
                High: 4
        Total issues (by confidence):
                Undefined: 0
                Low: 3
                Medium: 2
                High: 3

Test results:

>> Issue: [high_entropy_assign] Possible hardcoded secret(s) 'AKIA...tIcQ'.
   Severity: High   Confidence: High
   Location: examples/evil.py:6
6       ACCESS_TOKEN = 'AKIAlWxxtmzKD78tmCP8tIcQ'
--------------------------------------------------
>> Issue: [high_entropy_assign] Possible hardcoded secret(s) 'l6q3...Htmv'.
   Severity: High   Confidence: High
   Location: examples/evil.py:7
7       ACCESS_TOKEN_SECRET = 'l6q3ztS5pMz5i62WvovPyrAAjRfczKIHtmv'
--------------------------------------------------
>> Issue: [high_entropy_assign] Possible hardcoded secret(s) 'FMwo...goaj'.
   Severity: High   Confidence: High
   Location: examples/evil.py:19
19          if password == 'FMwoP5Bg0FZiIkLRgoaj':
--------------------------------------------------
```

Thank You.

lyft

@squiddlane

rlane@lyft.com