Building a HashiCorp Vault provider

How to manage ultra-sensitive secrets and prevent leaks without too much stress

Why Vault?

Or any other secret management solution, for that matter?

That’s why

Dev put AWS keys on Github. Then BAD THINGS happened

Fertile fields for Bitcoin yields - with a nasty financial sting

Darren Pauli

Tue 6 Jan 2015 // 13:02 UTC

Bots are crawling all over GitHub seeking secret keys, a developer served with a $2,375 Bitcoin mining bill found.

DevFactor founder Andrew Hoffman said he used Figaro to secure Rails apps which published his Amazon S3 keys to his GitHub account.

He noticed the blunder and pulled the keys within five minutes, but that was enough for a bot to pounce and spin up instances for Bitcoin mining.

"When I woke up the next morning, I had four emails and a missed phone call from Amazon AWS - something about 140 servers running on my AWS account," Hoffman said.

"I only had S3 keys on my GitHub and they were gone within five minutes!"

"As it turns out, through the S3 API you can actually spin up EC2 instances, and my key had been spotted by a bot that continually searches GitHub for API keys."

Amazon (he said) told him such bot exploits were increasingly common with hackers running algorithms to perpetually search for GitHub for API keys.

"Once it finds one it spins up max instances of EC2 servers to farm itself Bitcoins," he said.

Amazon refunded his bill, as it had for others. In December 2013 hackers ran up a $3,493 Litecoin mining bill for developer Luke Chadwick.

The blunder was the latest of many and served as a reminder for administrators to adhere to Amazon's best practice guidelines. ®

Similar topics

Amazon

Bitcoin

GitHub

Awesome to have you here!

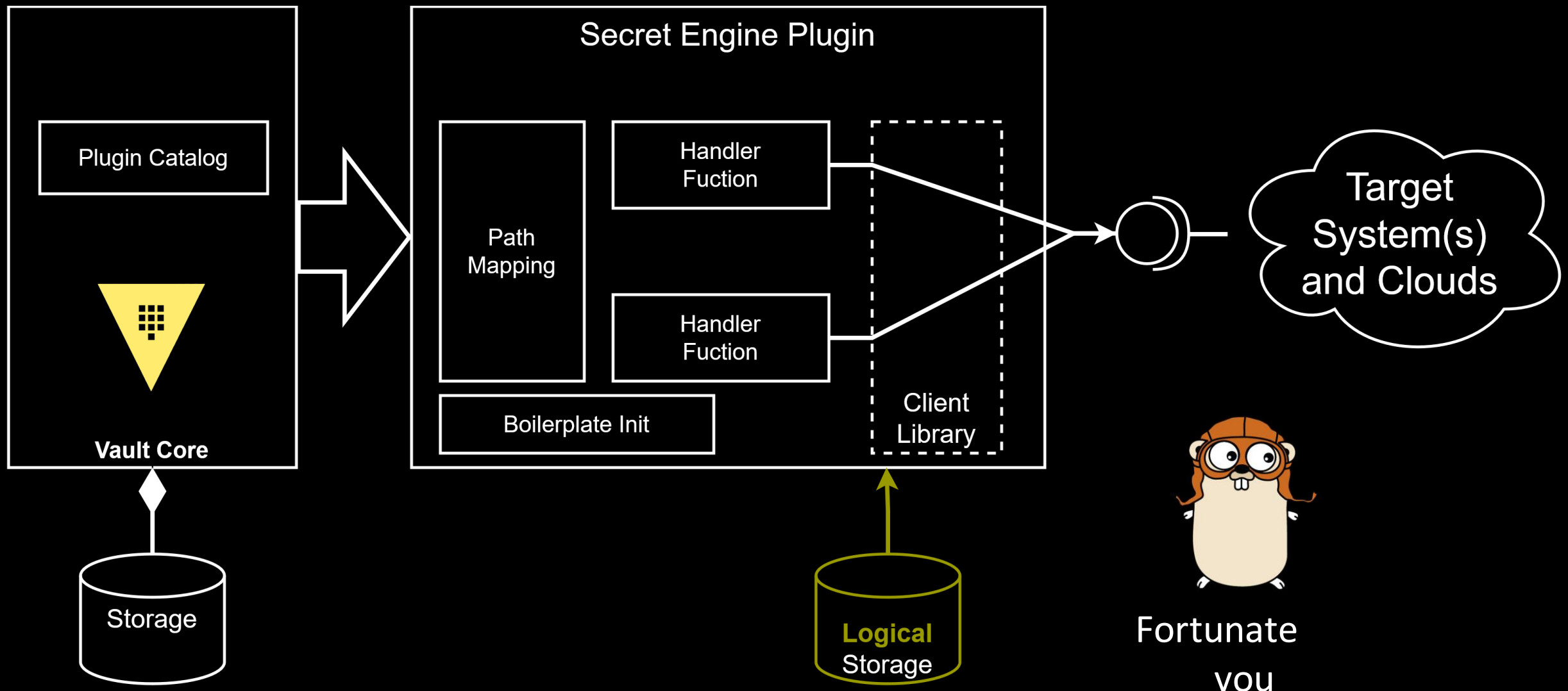
Building a HashiCorp Vault provider

… or rather what I wish I knew before I ever started

```go
type Post struct {  4 usages
    Title  string `json:"Title"`
    Author string `json:"Author"`
    Text   string `json:"Text"`
}

func PostsHandler(w http.ResponseWriter, r *http.Request) {  1 usage

    posts := []Post{
        Post{Title: "Post one", Author: "John", Text: "This is first post."},
        Post{Title: "Post two", Author: "Jane", Text: "This is second post."},
        Post{Title: "Post three", Author: "John", Text: "This is another post."},
    }

    json.NewEncoder(w).Encode(posts)
}

func main() {

    http.HandleFunc("/posts", PostsHandler)
    http.ListenAndServe(addr: ":5051", handler: nil)
}
```

Copied from here

Vault Secret Engine Deep Dive

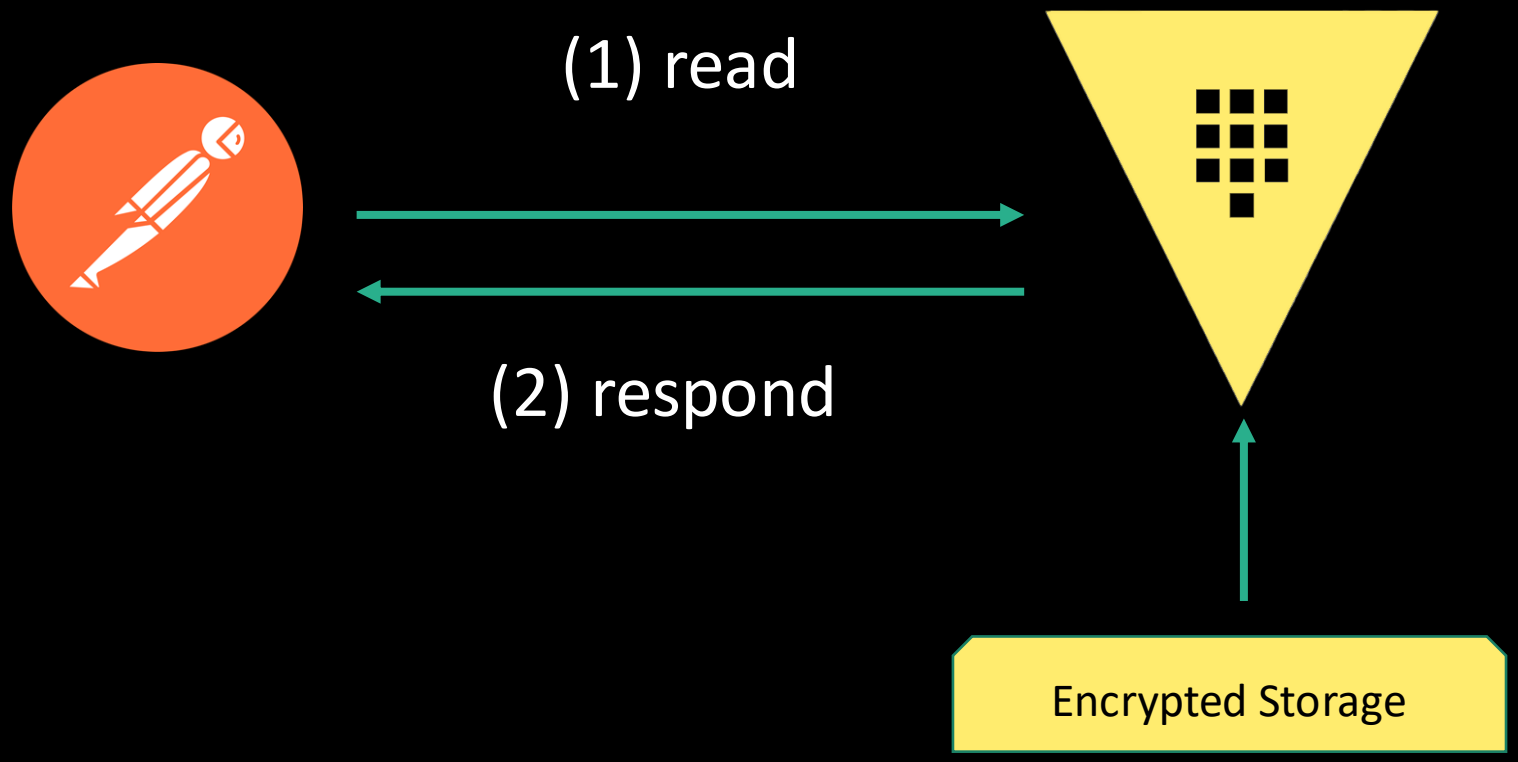


Fortunate you

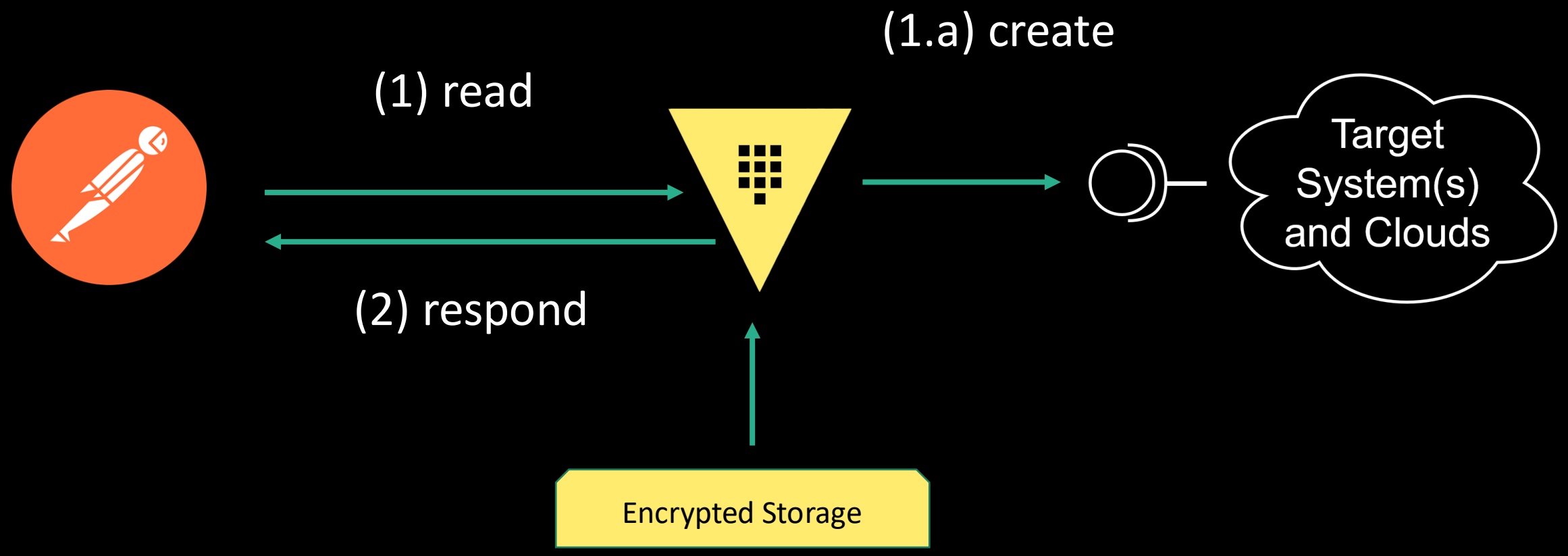
Vault and Vault Agent

Data and Lease Return Types

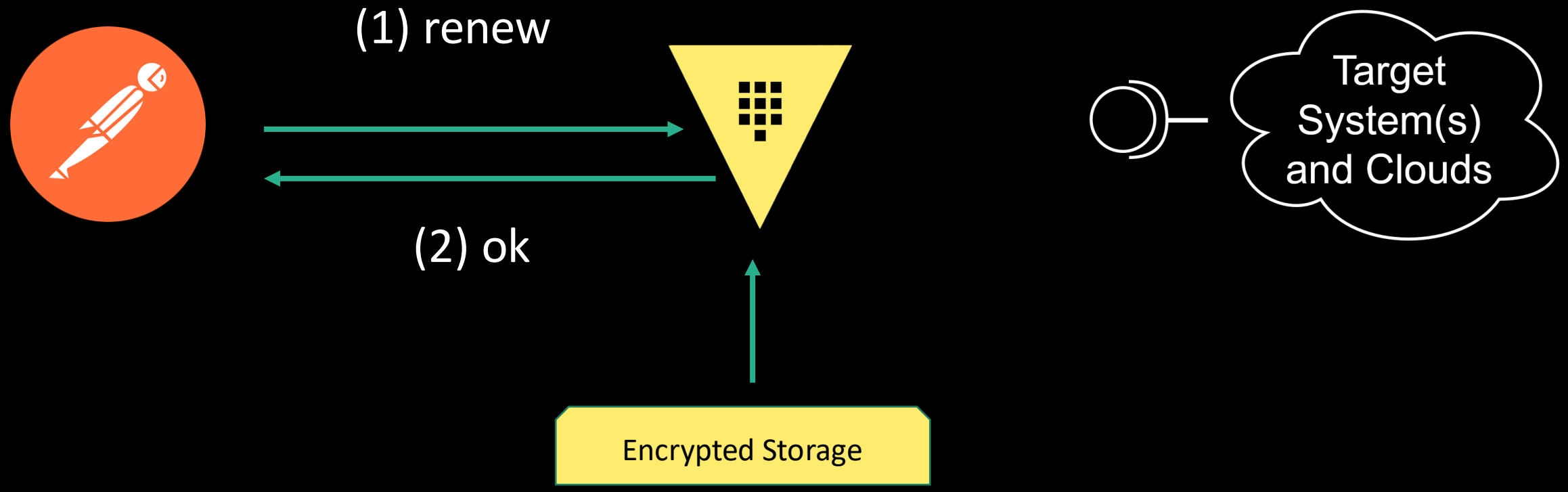
Data



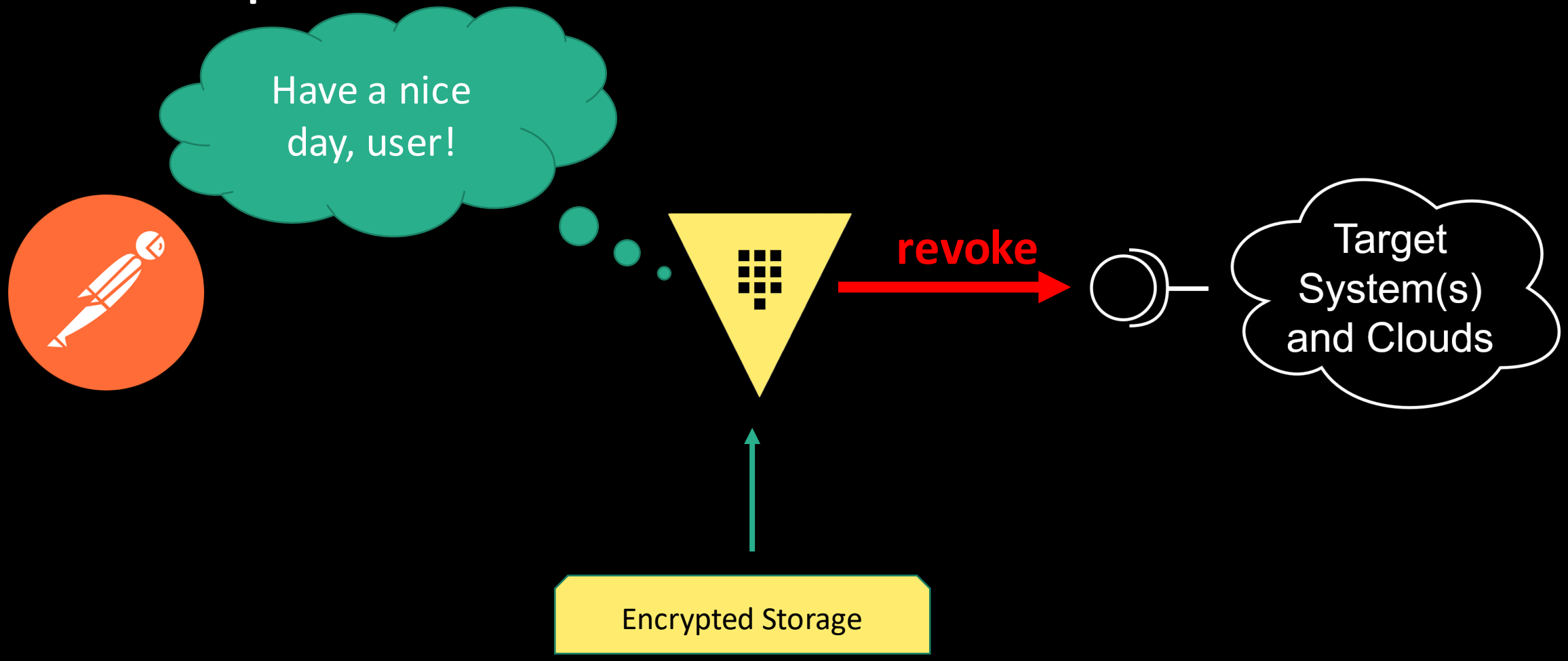
Lease | Create



Lease | Renew



Lease | Revoke

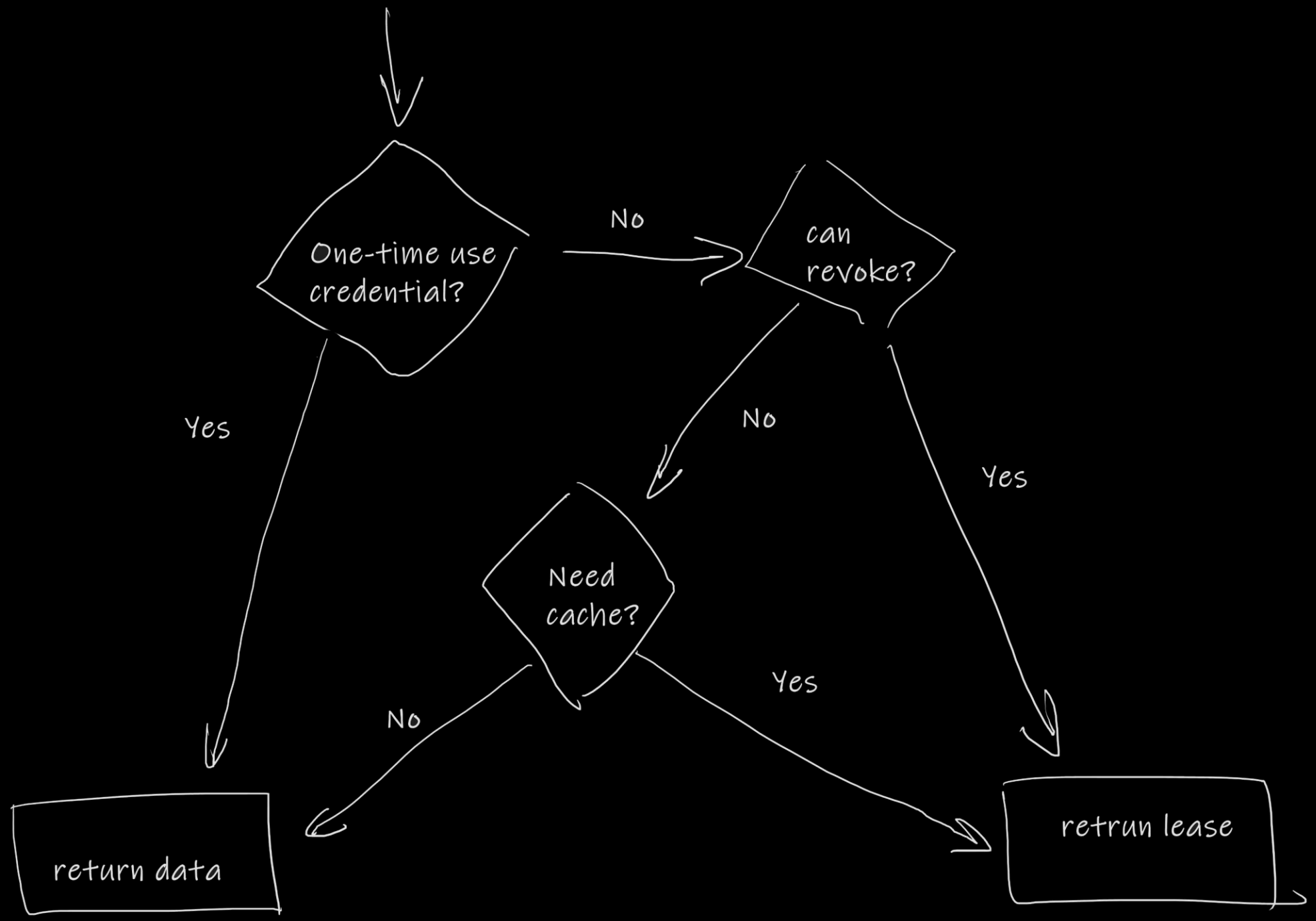


Granted Credentials Types

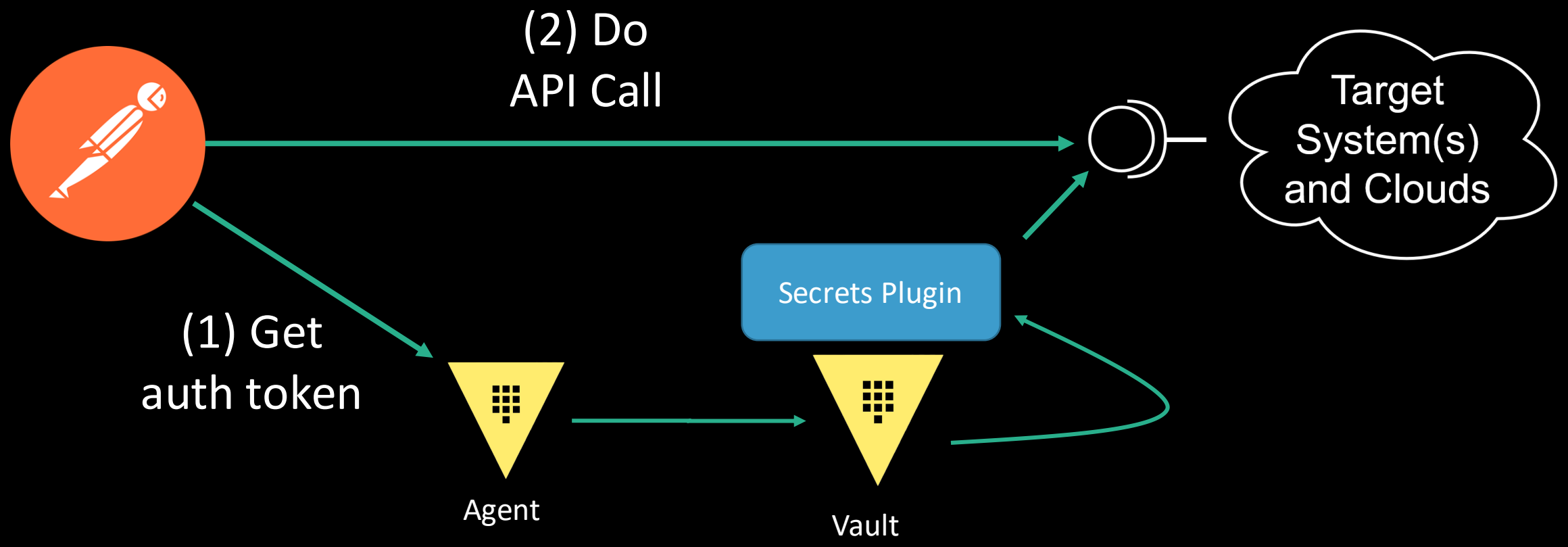
One-Time

Effectively Irrevocable

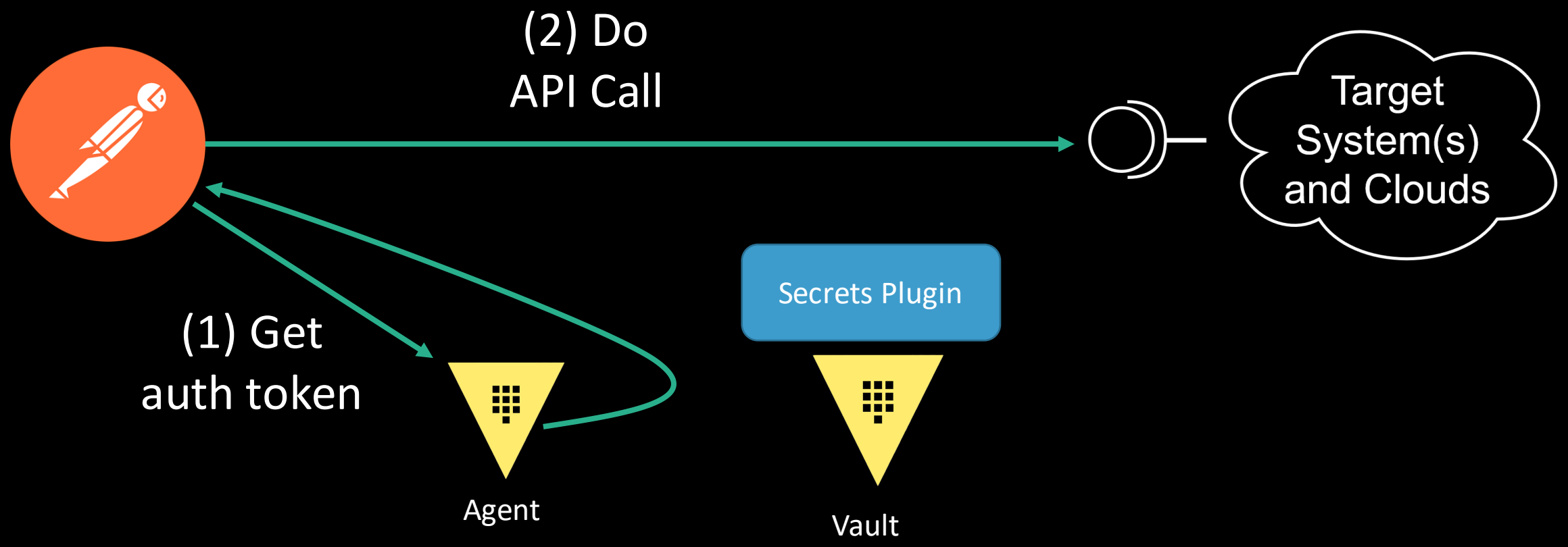
Revocable


One-time use credential?
No
can revoke?
Yes
No
Yes
Need cache?
No
Yes
return data
retrun lease

Vault & Vault Agent

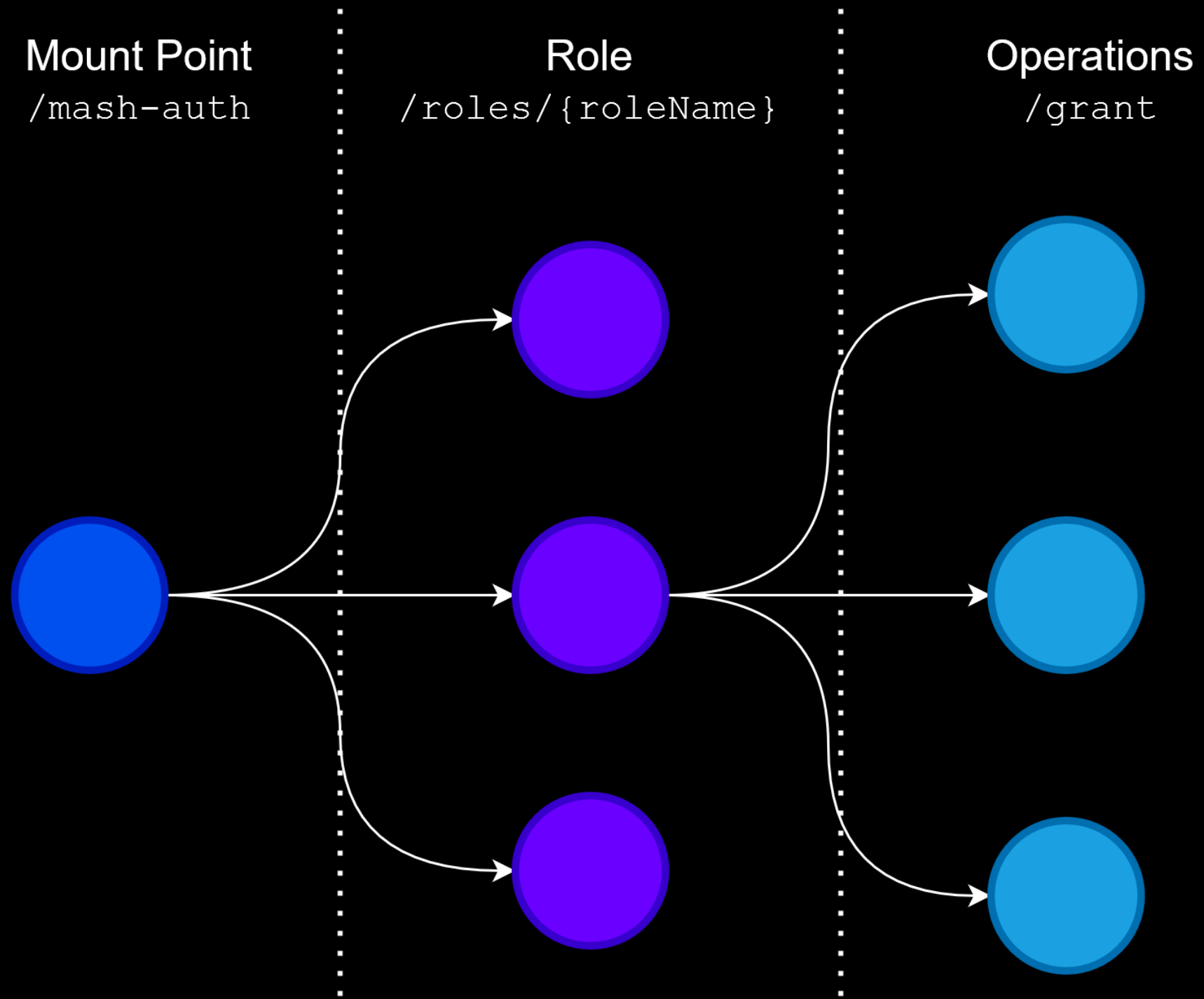


Vault & Vault Agent



Role

How to store organize your credentials


Mount Point
/mash-auth
Role
/roles/{roleName}
Operations
/grant

Storage

How to store anything in Vault

```go
                Callback: b.handleDeleteRoleData,
                Summary:  "Delete area authentication data",
            },
        },
        HelpSynopsis:    pathAreasHelpSyn,
        HelpDescription: pathAreasHelpDesc,
    }
}

func (b *AuthPlugin) roleExistenceCheck(ctx context.Context,    6 usages    lspwd2 *
    req *logical.Request,
    data *framework.FieldData) (bool, error) {

    return b.checkObjectExistsInStorage(ctx,
        req,
        b.storagePathForRole(data)+storedRoleKeyPathSuffix)
}

func (b *AuthPlugin) handleWriteRoleKeys(ctx context.Context, req *logical.Request, data *framework.FieldData) (*logical.Response
    chain := SimpleChain(
        readRole[RoleContext](mustBePresent: false),
        blockOperationOnImportedRole[RoleContext],
        updateRoleKeysFromRequest,
        saveRoleKeys[RoleContext],
        setInitialRoleUsage[RoleContext],
        saveRoleUsage[RoleContext],
```

*AuthPlugin.roleExistenceCheck(ctx context.Context, req *logical.Request, data *framework.FieldData) (bool, error)

```go
func (b *AuthPlugin) roleName(data *framework.FieldData) string { 2 usages  lspwd2
    return data.Get(roleName).(string)
}

func (b *AuthPlugin) rolesStorageRoot() string { 2 usages  lspwd2
    return b.backendUUID + "/role/"
}

func (b *AuthPlugin) checkObjectExistsInStorage(ctx context.Context,  1 usage  lspwd2 *
    req *logical.Request,
    rolePath string) (bool, error) {

    if out, err := req.Storage.Get(ctx, rolePath); err != nil {
        return false, errwrap.Wrapf( format: "existence check failed: {{err}}", err)
    } else : out != nil, nil
}

// noopRenewRevoke revocation of the secret that was issued
func noopRenewRevoke(context.Context, *logical.Request, *framework.FieldData) (*logical.Response, error) { 5 usages  lspwd2
    return nil, nil
}

func (b *AuthPlugin) configPath() string { 5 usages  lspwd2
    return b.backendUUID + "/config"
}
```

```go
}

type VaultStorageImpl struct {  13 usages  lspwd2
    VaultStorage
}

func (vsi *VaultStorageImpl) Persist(ctx context.Context, storage logical.Storage,  4 usages  lspwd2 *
    path string, json interface{}) error {

    if se, err := logical.StorageEntryJSON(path, json); err != nil {
        return errwrap.Wrapf("failed to create config storage entry: {{err}}", err)
    } else {
        if e := storage.Put(ctx, se); e != nil {
            return errwrap.Wrapf(fmt.Sprintf("failed to persist object: #{e.Error()}"), e)
        } else {
            return nil
        }
    }
}

func (vsi *VaultStorageImpl) PersistBinary(ctx context.Context, storage logical.Storage, path string, data []byte) error {  1 usag
    se := logical.StorageEntry{
        Key:   path,
        Value: data,
    }
```

*VaultStorageImpl.Persist(ctx context.Context, storage logical.Storage, path string, json interface{}) error

```go
        } else {
            if e := storage.Put(ctx, se); e != nil {
                return errwrap.Wrapf(fmt.Sprintf("failed to persist object: #{e.Error()}"), e)
            } else {
                return nil
            }
        }
    }

func (vsi *VaultStorageImpl) PersistBinary(ctx context.Context, storage logical.Storage,    1 usage    lspwd2 *
    path string, data []byte) error {

    se := logical.StorageEntry{
        Key:   path,
        Value: data,
    }
    return storage.Put(ctx, &se)
}

func (vsi *VaultStorageImpl) Read(ctx context.Context, storage logical.Storage, path string, receiver interface{}) (bool, error)
    if entry, err := storage.Get(ctx, path); err != nil {
        return false, errwrap.Wrapf(format: "failed to read object from storage: {{err}}", err)
    } else if entry == nil : false, nil , else {
        if err := entry.DecodeJSON(receiver); err != nil {
            return true, errwrap.Wrapf(format: "cannot unmarshal data structure ({{err}})", err)
        } else : true, nil ,
```

*VaultStorageImpl.PersistBinary(ctx context.Context, storage logical.Storage, path string, data []byte) error

```go
}

func (vsi *VaultStorageImpl) Read(ctx context.Context, storage logical.Storage, path string, receiver interface{}) (bool, error)
    if entry, err := storage.Get(ctx, path); err != nil {
        return false, errwrap.Wrapf("failed to read object from storage: {{err}}", err)
    } else if entry == nil { false, nil } else {
        if err := entry.DecodeJSON(receiver); err != nil {
            return true, errwrap.Wrapf("cannot unmarshal data structure ({{err}})", err)
        } else { true, nil }
    }
}

func (vsi *VaultStorageImpl) ReadBinary(ctx context.Context, storage logical.Storage,    1 usage    lspwd2 *
    path string) (bool, []byte, error) {
    if entry, err := storage.Get(ctx, path); err != nil {
        return false, []byte{}, errwrap.Wrapf("failed to read data from storage: {{err}}", err)
    } else if entry == nil {
        return false, []byte{}, nil
    } else {
        return true, entry.Value, nil
    }
}
```

Couldn’t read it

Do data found at this path

There you go!

*VaultStorageImpl.ReadBinary(ctx context.Context, storage logical.Storage, path string) (bool, []byte, error)

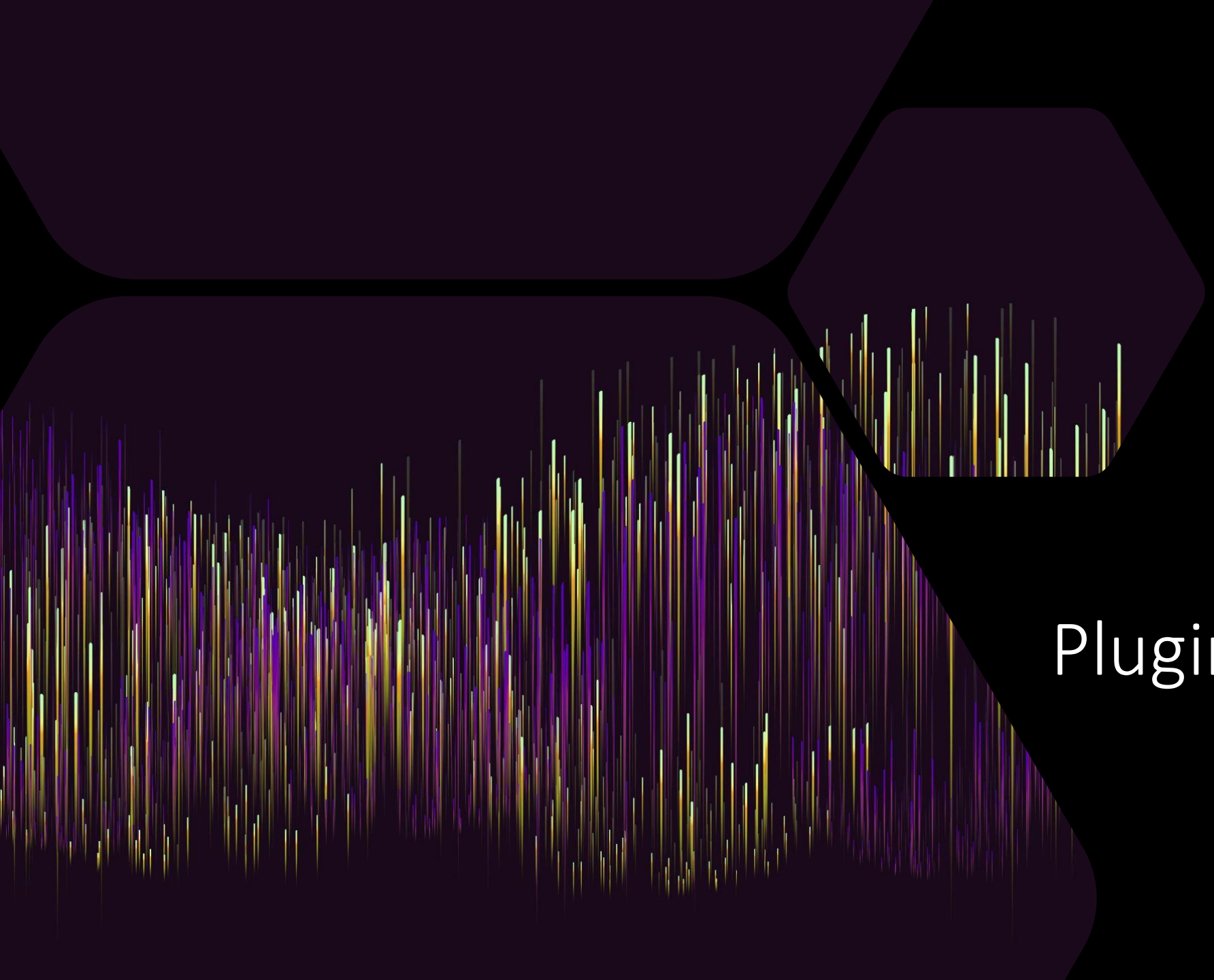
```go
    se := logical.StorageEntry{
        Key:   path,
        Value: data,
    }
    return storage.Put(ctx, &se)
}

func (vsi *VaultStorageImpl) Read(ctx context.Context, storage logical.Storage,
    path string, receiver interface{}) (bool, error) {

    if entry, err := storage.Get(ctx, path); err != nil {
        return false, errwrap.Wrapf("failed to read object from storage: {{err}}", err)
    } else if entry == nil {
        return false, nil
    } else {
        if err := entry.DecodeJSON(receiver); err != nil {
            return true, errwrap.Wrapf("cannot unmarshal data structure ({{err}})", err)
        } else {
            return true, nil
        }
    }
}

func (vsi *VaultStorageImpl) ReadBinary(ctx context.Context, storage logical.Storage,
    path string) (bool, []byte, error) {
```

Couldn’t read it

Do data found at this path

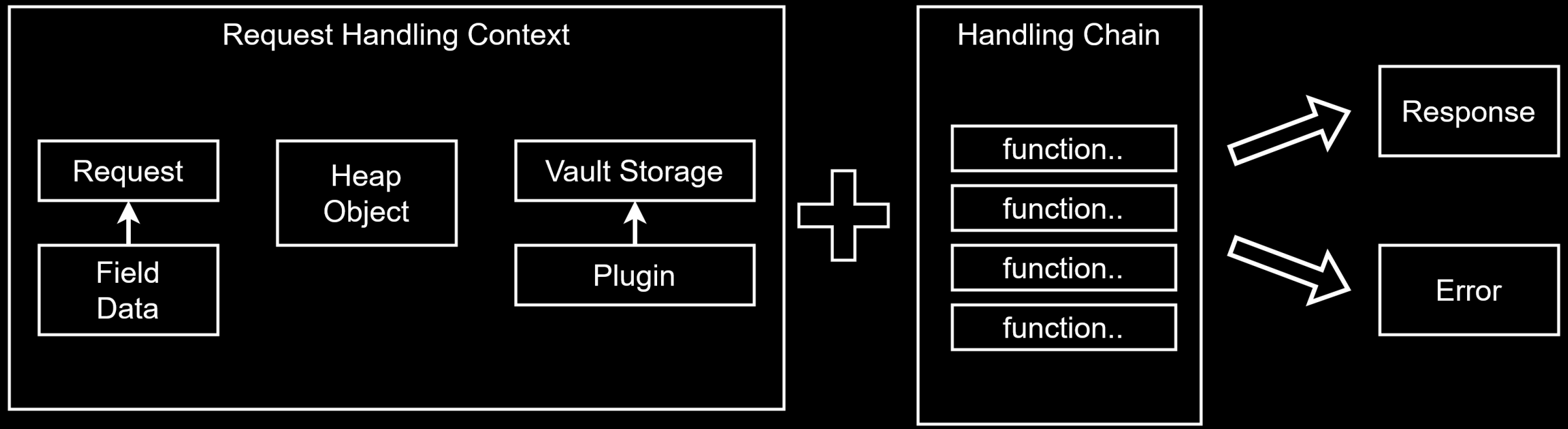
There you go!

*VaultStorageImpl.Read(ctx context.Context, storage logical.Storage, path string, receiver interface{}) (bool, error)



Plugin Architecture

Building Elements

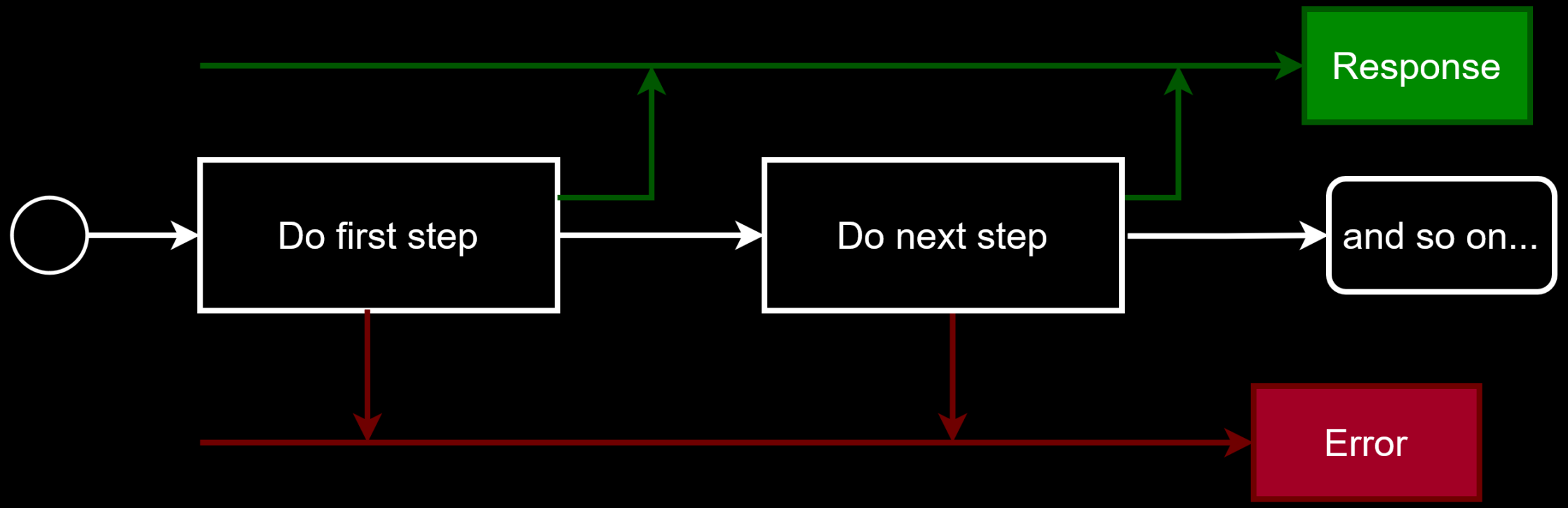


```go
func (b *AuthPlugin) roleExistenceCheck(ctx context.Context,  6 usages  lspwd2 *
    req *logical.Request,
    data *framework.FieldData) (bool, error) {

    return b.checkObjectExistsInStorage(ctx,
        req,
        b.storagePathForRole(data)+storedRoleKeyPathSuffix)
}

func (b *AuthPlugin) handleWriteRoleKeys(ctx context.Context, req *logical.Request,  1 usage  lspwd2 *
    data *framework.FieldData) (*logical.Response, error) {
    chain := SimpleChain(
        readRole[RoleContext](mustBePresent: false),
        blockOperationOnImportedRole[RoleContext],
        updateRoleKeysFromRequest,
        saveRoleKeys[RoleContext],
        setInitialRoleUsage[RoleContext],
        saveRoleUsage[RoleContext],
    )

    return handleRoleBoundOperation(ctx, b, req, data, chain)
}

func (b *AuthPlugin) handleReadRoleData(ctx context.Context, req *logical.Request, data *framework.FieldData) (*logical.Response,
    chain := SimpleChain(
```

*AuthPlugin.handleWriteRoleKeys(ctx context.Context, req *logical.Request, data *framework.FieldData) (*logical.Response, error)

```go
        HelpSynopsis:    helpSynRoleImport,
        HelpDescription: helpDescRoleImport,
    }
}

// pathRoleExport Exports settings for secure exchange
func (b *AuthPlugin) pathRoleExport(ctx context.Context, req *logical.Request,    2 usages    lspwd2 *
    d *framework.FieldData) (*logical.Response, error) {

    // Read the certificate and role
    readChain := SimpleChain(
        readRole[RoleExportContext](mustBePresent: true),
        blockNonExportableRole,
        readRecipientCertificate,
        renderEncryptedRoleData,
    )

    var container RoleExportContext = &RoleExportContainer{}
    return handleOperationWithContainer(ctx, b, req, d, container, b.storagePathForRole(d), readChain)
}

func (b *AuthPlugin) pathRoleImport(ctx context.Context, req *logical.Request, d *framework.FieldData) (*logical.Response, error
    if pemBlock, pemErr := retrieveImportPEMBlockFromRequest(d); pemErr != nil {
        return logical.ErrorResponse(text: "input does not contain a valid PEM block (%s)", pemErr.Error()), nil
```

Handling Pipeline



```go
}

func (reqCtx *RequestHandlerContext[T]) WriteBinaryPath(ctx context.Context, path string, data []byte) error {
    if len(reqCtx.storagePath) == 0 { errors.New("storage path to read/write to/from hasn't been initialized") }

    return reqCtx.plugin.vaultStorage.PersistBinary(ctx, reqCtx.request.Storage, path, data)
}

// TransformerFunc function performing a transformation in the chain of responsibility. The function
// returns either response (and no further processing would be needed), or a data element for the next-in-chain
type TransformerFunc[T any] func(context.Context, *RequestHandlerContext[T]) (*logical.Response, error)

func SimpleChain[T any](subsequent ...TransformerFunc[T]) TransformerFunc[T] {
    return func(ctx context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error) {
        for _, f := range subsequent {
            response, err := f(ctx, reqCtx)
            if err != nil || response != nil {
                return response, err
            }
        }
        return nil, nil
    }
}

func Tail[T any](f1 TransformerFunc[T],
```

*RequestHandlerContext[T].WriteBinaryPath(ctx context.Context, path string, data []byte) error

```go
    if reqCtx.heap.GetRole().Keys.Imported { return logical.ErrorResponse("operation is not permitted on an imported role"), nil }
}

func blockOperationOnForceProxyRole[T RoleContext](_ context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error
    if reqCtx.heap.GetRole().Keys.ForceProxyMode {
        return logical.ErrorResponse(text: "operation is not permitted as this role requires proxy mode"), nil
    } else { return nil, nil }
}

func blockUsageExceedingLimits[T RoleContext](_ context.Context,  8 usages  lspwd2 *
    reqCtx *RequestHandlerContext[T]) (*logical.Response, error) {

    role := reqCtx.heap.GetRole()
    if role.Usage.Expired() {
        return logical.ErrorResponse(text: "this role has expired (granted until %s)", role.Usage.ExpiryTimeString()), nil
    } else if role.Usage.Depleted() {
        return logical.ErrorResponse(text: "this role has depleted its usage quota"), nil
    }

    return nil, nil
}

func decreaseRemainingUsageQuota[T RoleContext](ctx context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error)
    role := reqCtx.heap.GetRole()
```

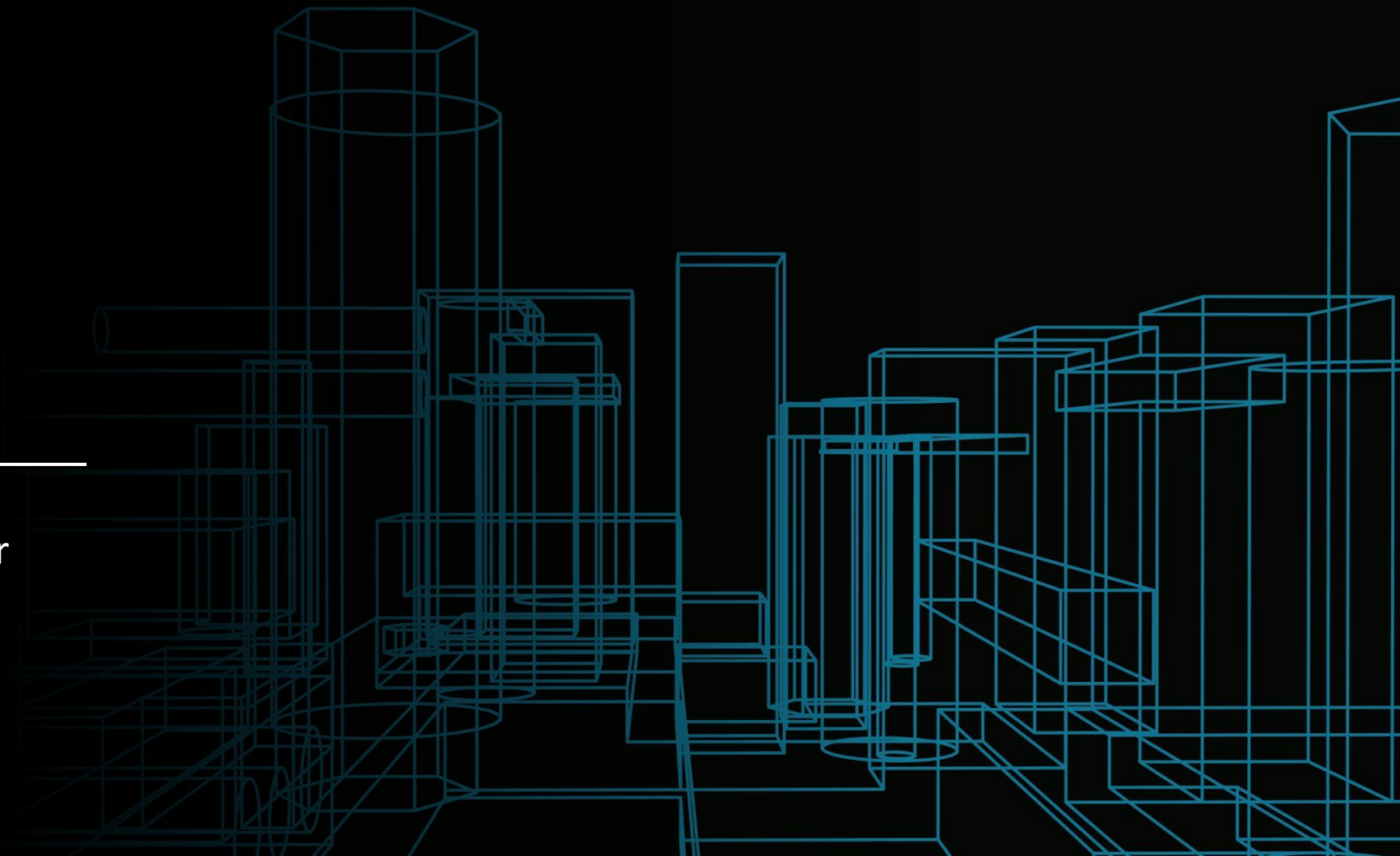
blockOperationOnForceProxyRole[T RoleContext](_ context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error)

```go
    return nil, nil
}

func blockRoleIncapableOf[T RoleContext](apiLevel int) func(ctx context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Resp
    return func(_ context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error) {
        return doBlockRoleIncapableOf(reqCtx.heap.GetRole(), apiLevel)
    }
}

func blockNonExportableRole(_ context.Context,    3 usages    lspwd2 *
    reqCtx *RequestHandlerContext[RoleExportContext]) (*logical.Response, error) {

    if !reqCtx.heap.GetRole().Keys.Exportable {
        return logical.ErrorResponse(text: "this role is not exportable"), nil
    } else {
        return nil, nil
    }
}

func doBlockRoleIncapableOf(role *StoredRole, apiLevel int) (*logical.Response, error) {    1 usage    lspwd2
    capable := false
    switch apiLevel {
    case 2:
        capable = role.Keys.IsV2Capable()
    case 3:
        capable = role.Keys.IsV3Capable()
```

blockRoleIncapableOf[T RoleContext](apiLevel int) func(ctx context.Context, reqCtx *RequestHandlerContext[T]) (*logical.Response, error) > func(_ context.Context, reqCtx *Reque

The fun part

Explain what you've done to your colleagues



I’m supposed to say something positive…

Your GO
code is utterly
beautiful!

Fearing new security tooling

99 reasons to avoid security, reason #7

Prevention is always cheaper.

Do onto security *before* the (lack of) security will do onto you!

My flight plan to Vault

We use cloud, so we must be secure!

Home Workspaces Explore

Search Postman

Sign In Create Account

Working locally in Scratch Pad. **Switch to a Workspace**

Scratch Pad New Import

Overview

No Environment

Collections

APIs

Environments

Mock Servers

Monitors

History

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create Collection

Scratch Pad

The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

In your Scratch Pad

0	0	0
Requests	Collections	Environments

Get started

- Create a request
- Create a collection
- Create an API
- Create an environment
- View More

Find and Replace Console

Runner Trash

9°C Bewolkt

Search

15:05 19/02/2023

Home Workspaces Explore

Search Postman

Sign In Create Account

Working locally in Scratch Pad. **Switch to a Workspace**

Scratch Pad New Import

Overview

No Environment

Collections

APIs

Environments

Mock Servers

Monitors

History

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create Collection

Scratch Pad

The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

In your Scratch Pad

0 Requests

0 Collections

0 Environments

Get started

Create a request

Create a collection

Create an API

Create an environment

View More

Working locally in Scratch Pad. **Switch to a Workspace**

Find and Replace Console

Runner Trash

9°C Bewolkt

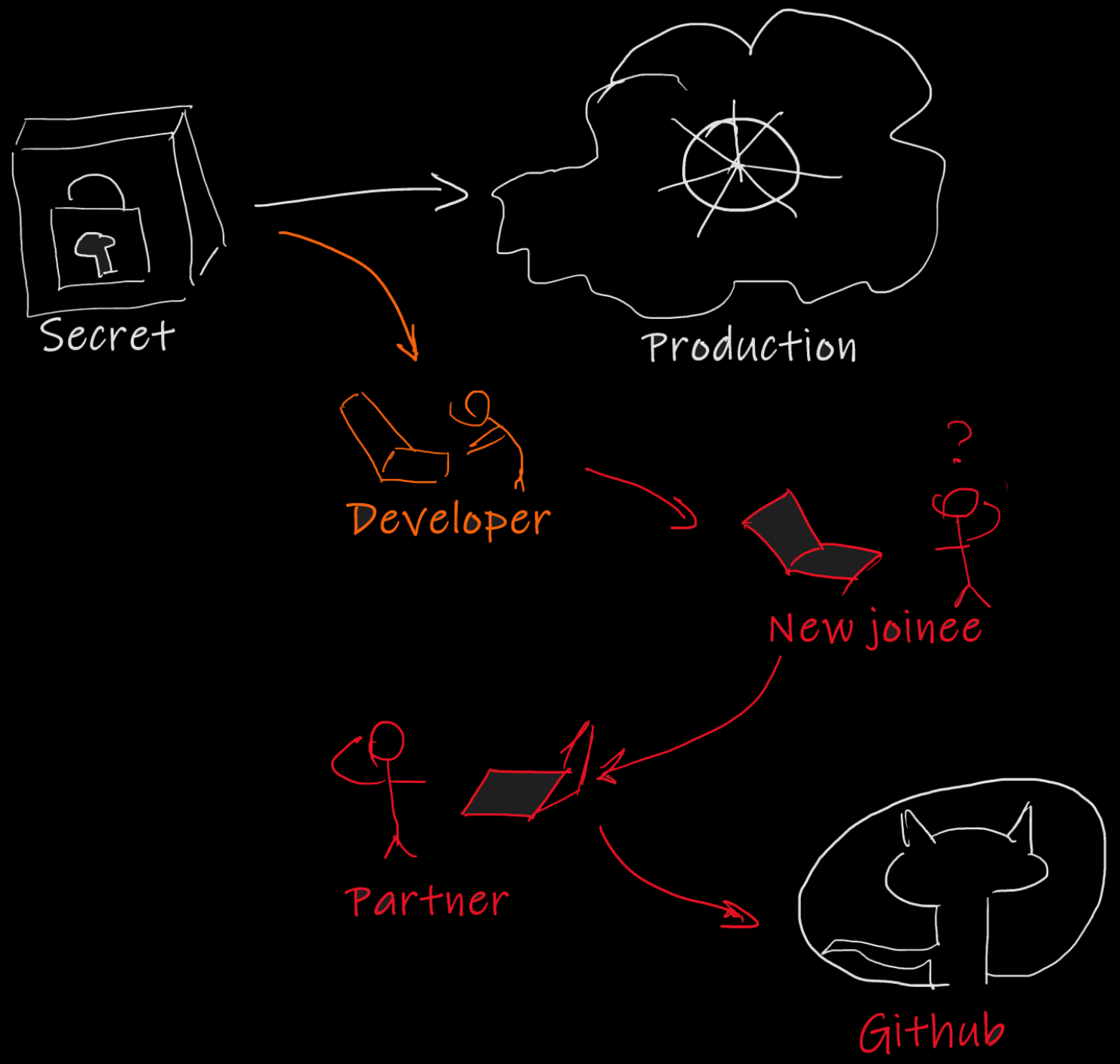
Search

14:34 19/02/2023

What’s a big deal here?

Secrets sprawl

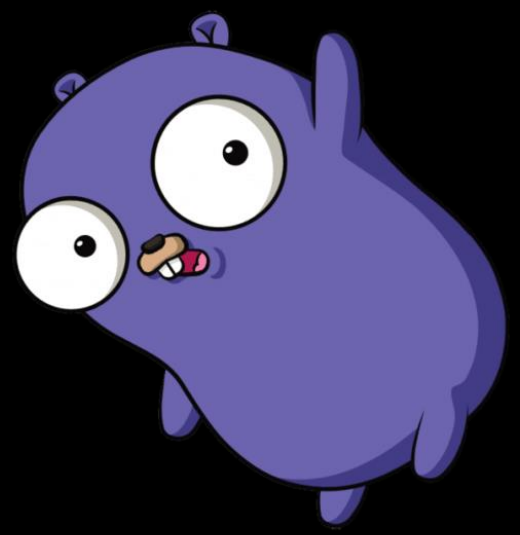
Undesired, unintended or uncontrollable copying and **retaining** long-term confidential data



I’m not
allowed to use

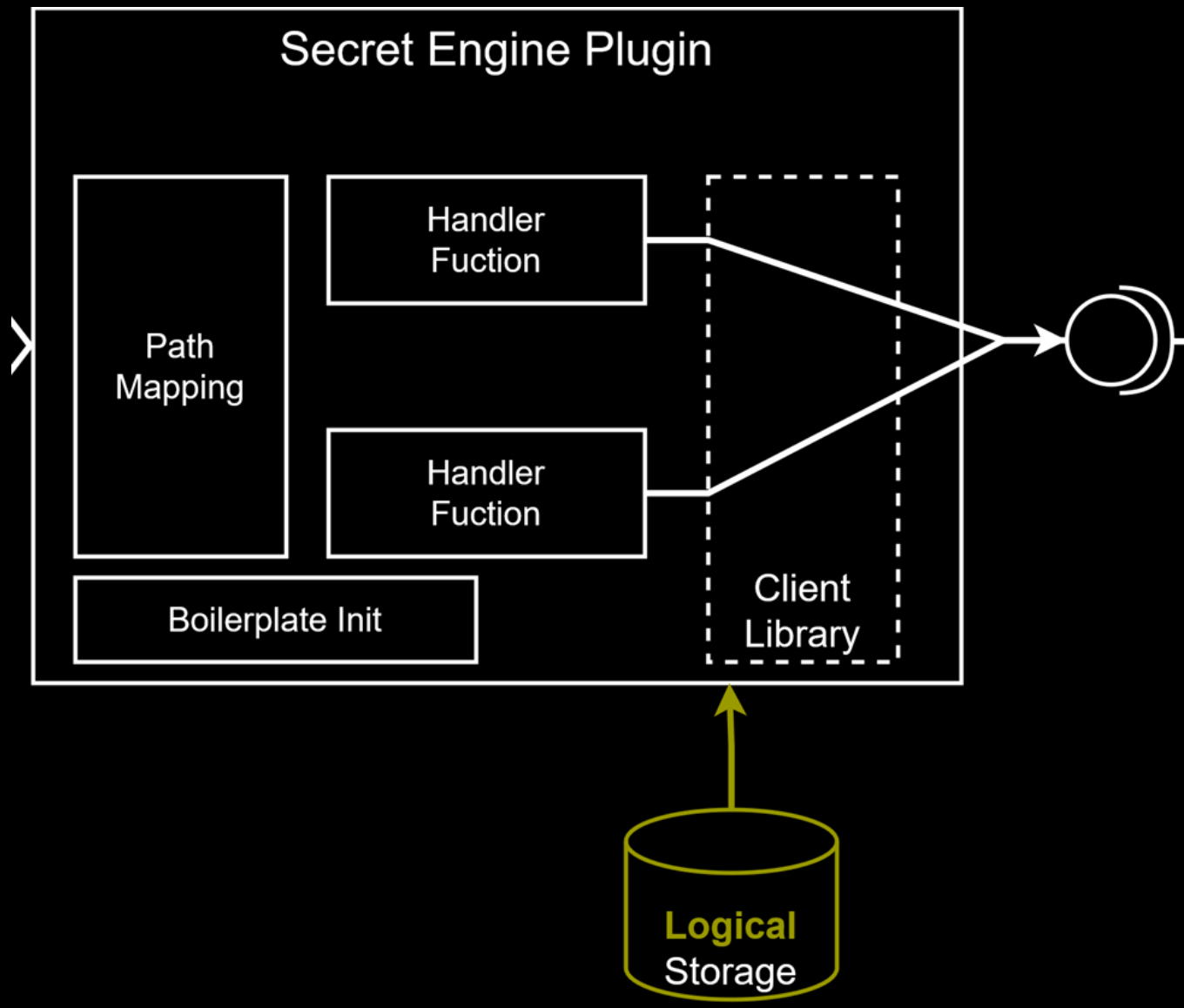
Yeah, we should be safe!...

Let’s make safe again!



Me

Basically, a hero to the rescue



Home Workspaces API Network Explore Search Postman

My Workspace New Import

Overview Notion

Collections
APIs
Environments
Mock Servers
Monitors
Flows
History

- Mashery Vault provider
- Notion
  - GET List databases
  - GET Search database
  - GET Get page
  - POST Search
- VaultIntegration

Notion

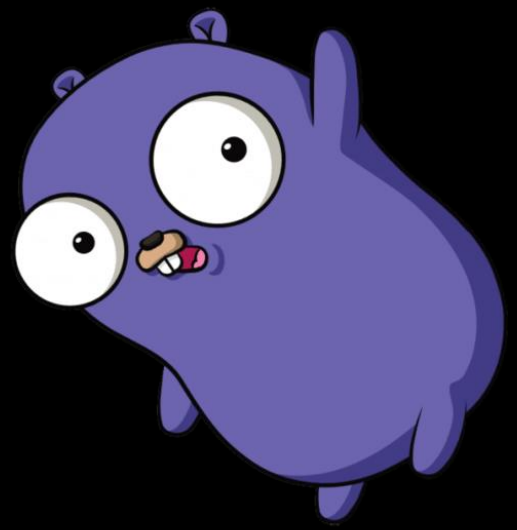
Share

Authorization Pre-request Script Tests Variables Runs

This script will execute before every request in this collection. Learn more about Postman's execution order

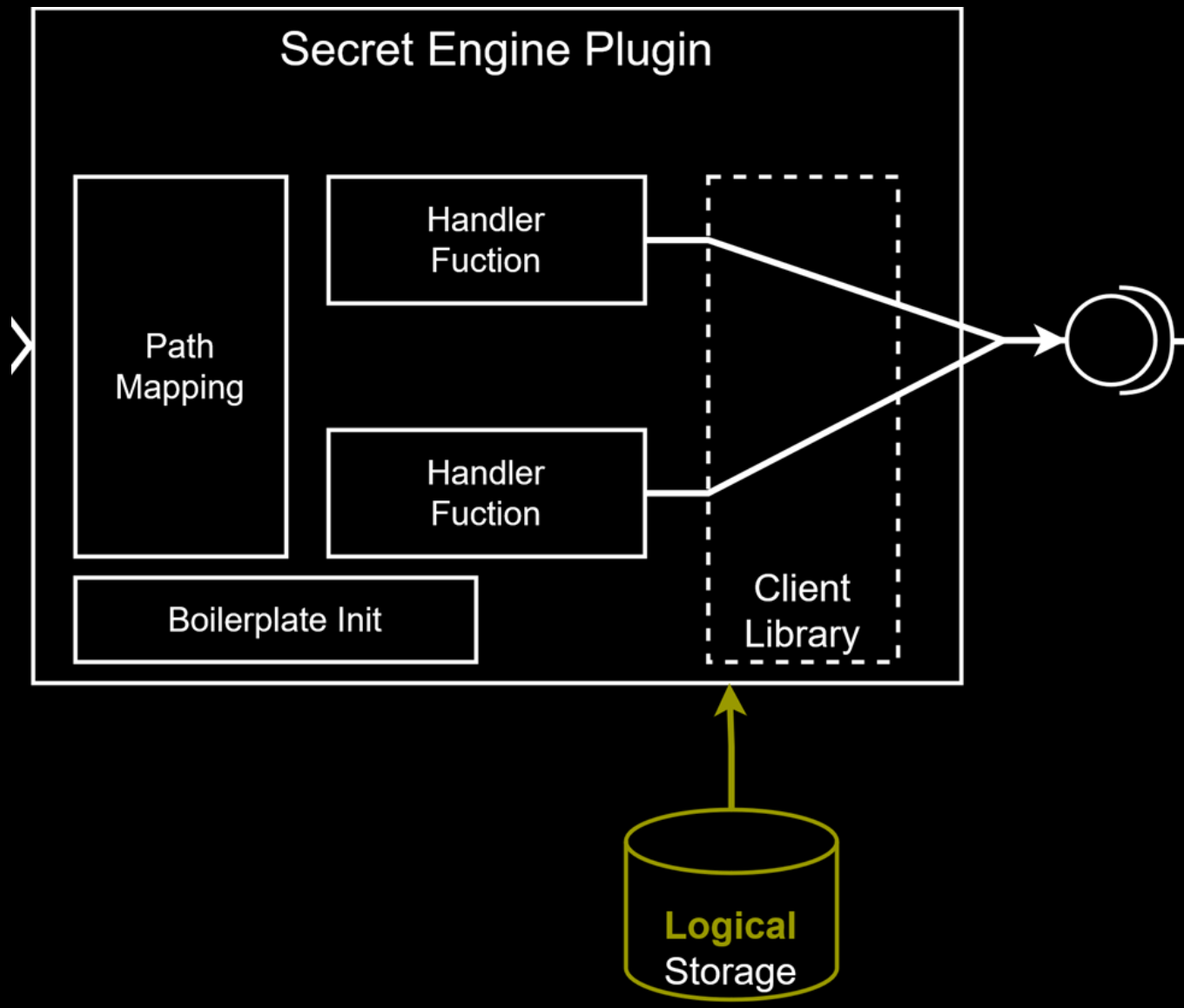
```
pm.sendRequest("http://127.0.0.1:8300/v1/passwords/data/notions.so/bike-mnt-tracker",
    function (err, response) {
        const tkn = response.json().data.data.t
        pm.variables.set("notion_token", tkn)
    }
);
```

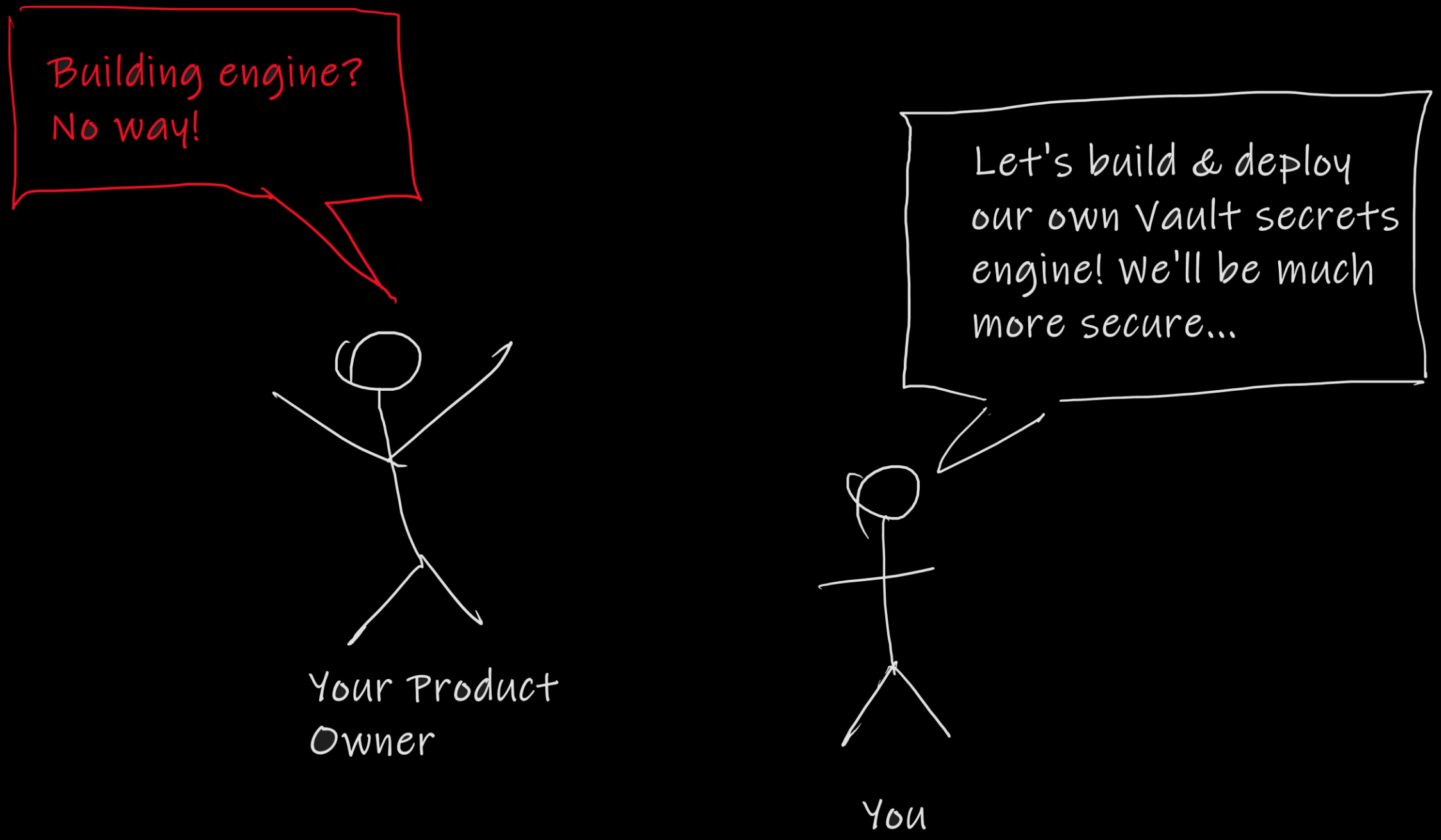
Let’s make safe again!



Me

Gonna code this

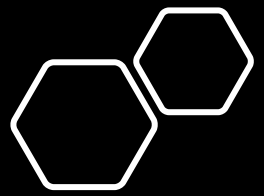


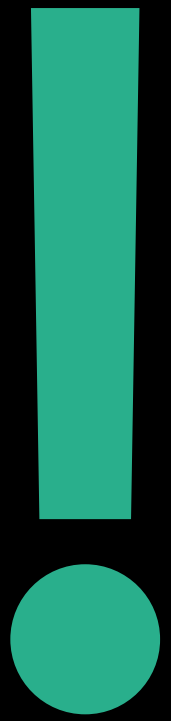

Building engine?
No way!
Your Product Owner
Let's build & deploy our own Vault secrets engine! We'll be much more secure...
You

We don’t do security, because we can’t

99 reasons to avoid security, reason #1

Do you *really* need your own secret engine?



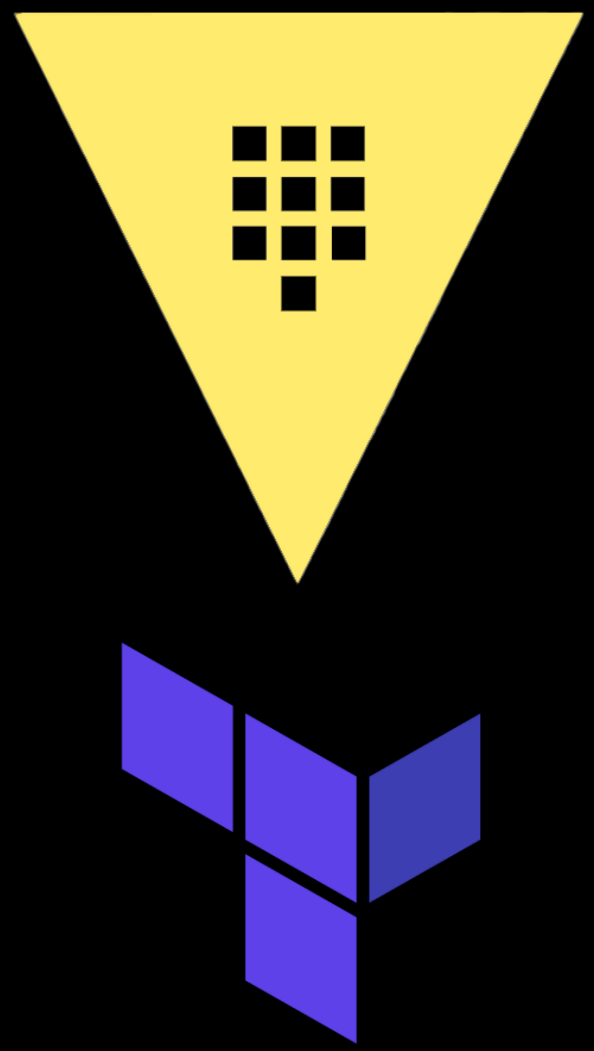


You **do** if you don’t know why you **don’t**

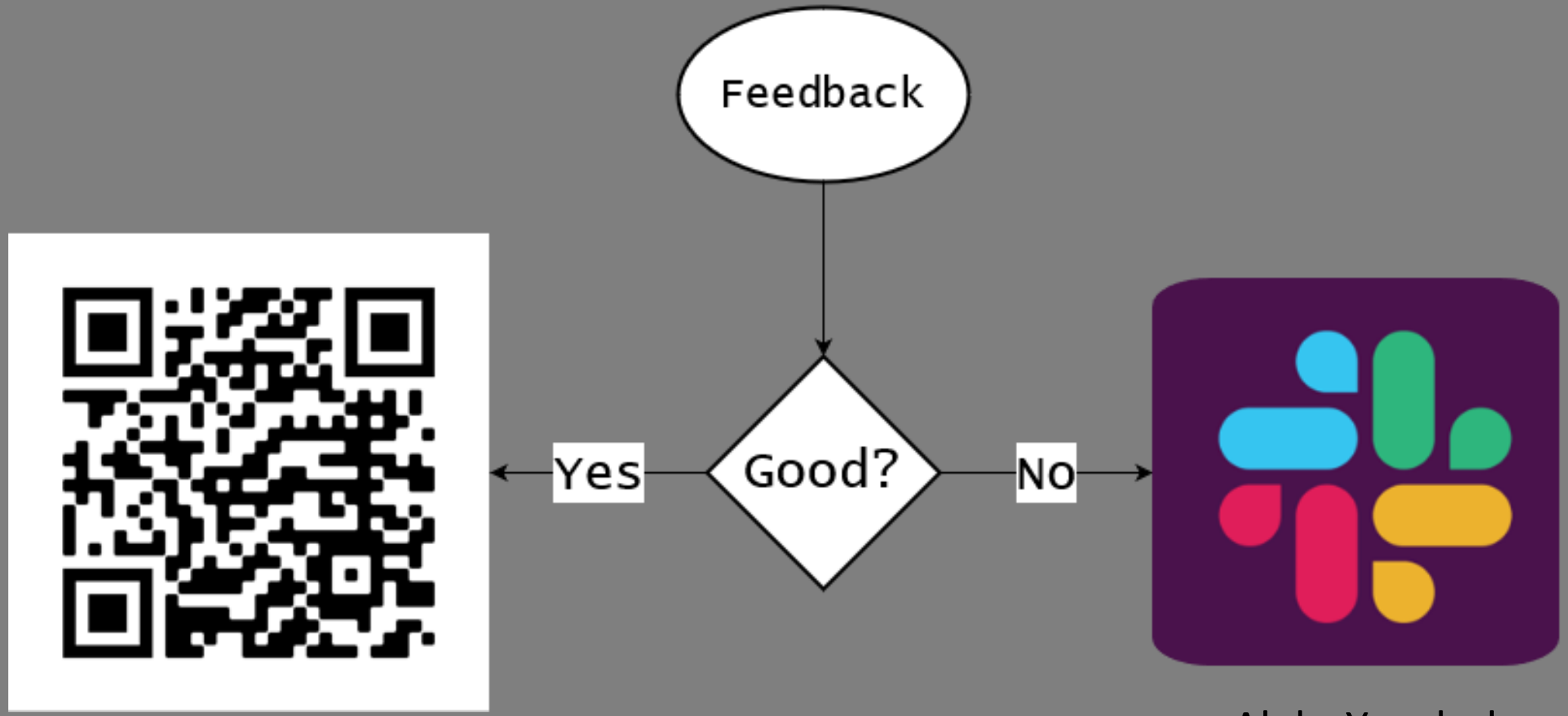
99 reasons to avoid security, reason #75

Q&A

(trust me, I'm a solution architect)



Happy Vault Hacking!



Aleks Yanchuk

How to ask more questions

References

- My LinkedIn
- Vault Plugin Development landing page
- Building plug-in tutorial
- My own work: A secret engine for TIBCO Mashery
- Terraform plugin talk, Golang Rotterdam meetup