Automating Configuration and Management of your GitHub Repositories With Minder



Evan Anderson

Software Engineer, Stacklok



Dania Valladares

Software Engineer Manager, Stacklok

stacklok

What We’re Going To Cover

Agenda

Repository security configuration and management.

What is it? Why does it matter to you?

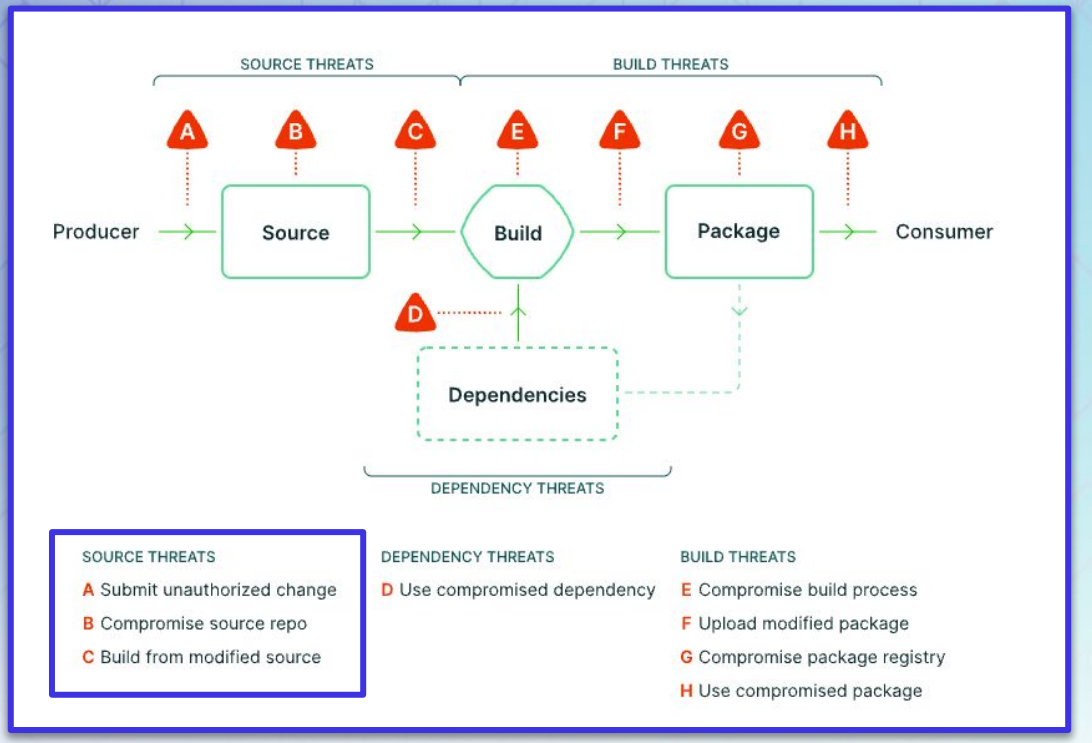
Automating and scaling security settings across your repositories.

What is Minder? How does it interact with your github repositories?

“You’ll be remembered by the rules you ~~break~~ make”

How to write your own rules for Minder using Rego.

Source code repos are targets for supply chain attacks



Source: slsa.dev

A: Sushiswap: Contractor with repository access pushed a malicious commit redirecting cryptocurrency to themself.

B: Linux Kernel: kernel.org hosts compromised, could have compromised Git repo.

C: Webmin: Attacker modified the build infrastructure to use source files not matching source control.

stacklok

A Lot of Tools!

Branch Protections

Required Checks

Github Actions

Secret Scanning

Dependabot / Renovate

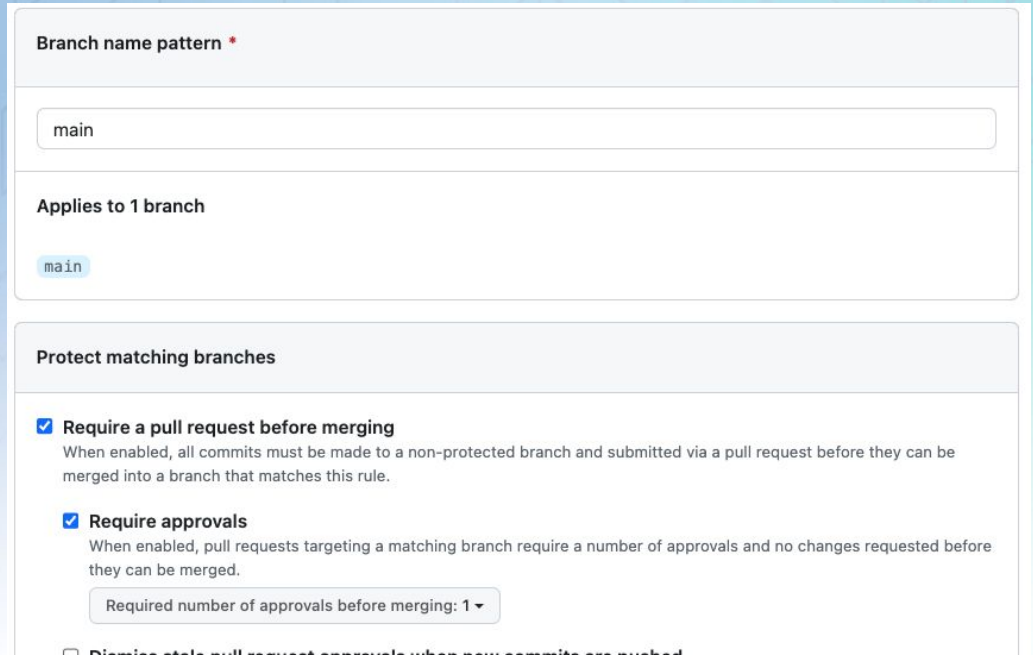
Code Scanners

Vulnerability Reports

Branch Protections

Require Code Review and other best practices.

First line of defense for changes – ensure at least two people review.



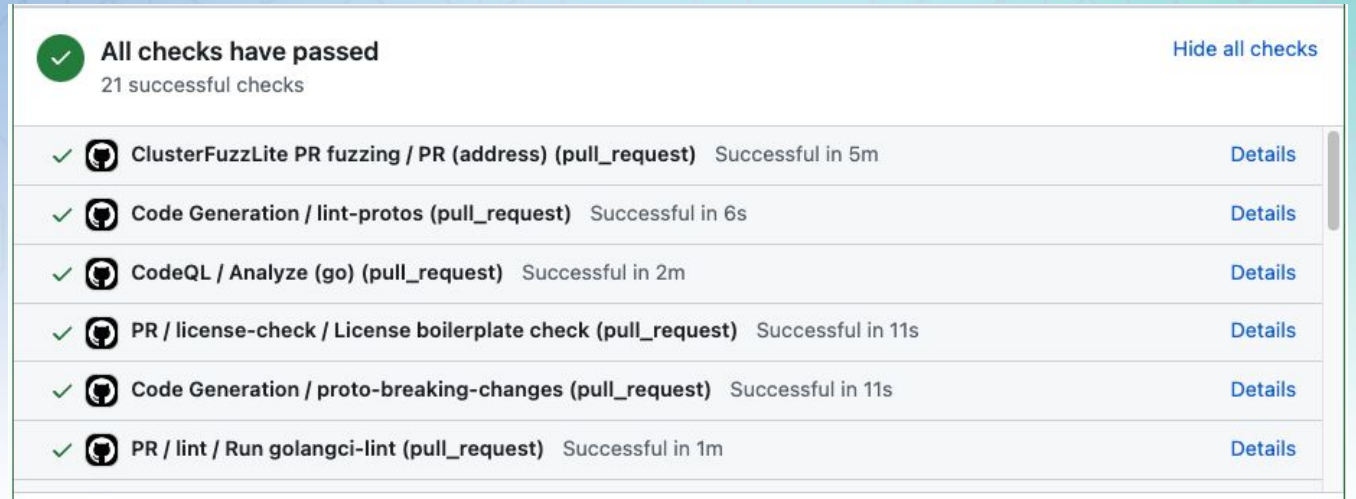
Beware of sock puppets!



Required Checks

Ensure that validation has been done before allowing code to be merged.

By default, checks are *optional*, but you can make some required!



Secret Scanning

Alert on PRs that include secrets matching various patterns.

GitHub can even block commits that include known patterns.

Free for public repos, but not on by default!

Provider	Token	Partner	User	Push protection
Adafruit	adafruit_io_key	✓	✓	✓
Adobe	adobe_client_secret	✓	✓	✓
Adobe	adobe_device_token	✓	✓	✓
Adobe	adobe_pac_token	✓	✓	✓
Adobe	adobe_refresh_token	✓	✓	✓
Adobe	adobe_service_token	✓	✓	✓
Adobe	adobe_short_lived_access_token	✓	✓	✓
Aiven	aiven_auth_token	✓	✓	✓
Aiven	aiven_service_password	✓	✓	✓
Alibaba	alibaba_cloud_access_key_id alibaba_cloud_access_key_secret	✓	✓	✓
Amazon AWS	aws_access_key_id aws_secret_access_key	✓	✓	✓
Amazon AWS	aws_secret_access_key	✕	✓	✓

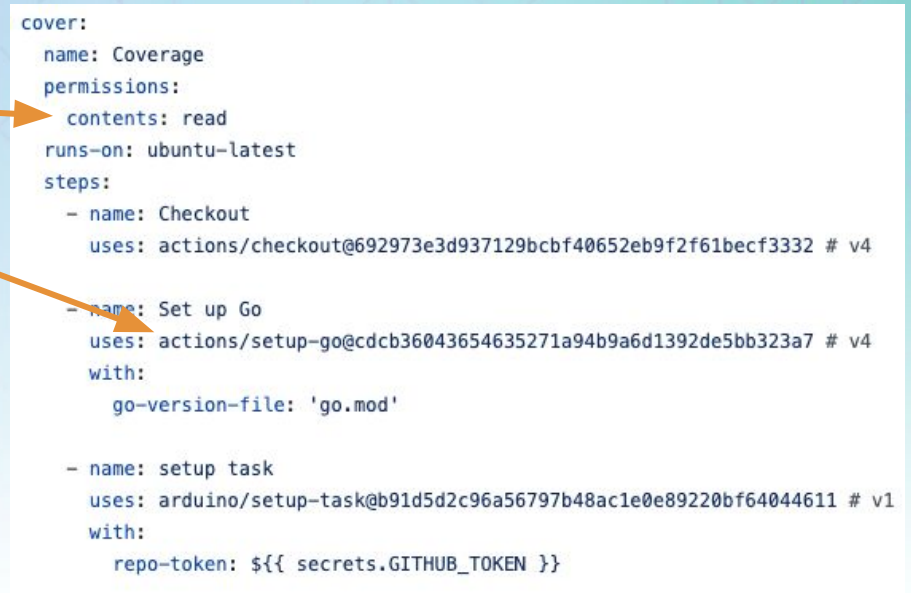
GitHub Actions

Actions are awesome!

Actions run with their own short-lived credentials, and are free and ephemeral.

There are still some risks!

- Permissions risk
- Dependency risk



```yaml
cover:
  name: Coverage
  permissions:
    contents: read
  runs-on: ubuntu-latest
  steps:
    - name: Checkout
      uses: actions/checkout@692973e3d937129bcbf40652eb9f2f61becf3332 # v4

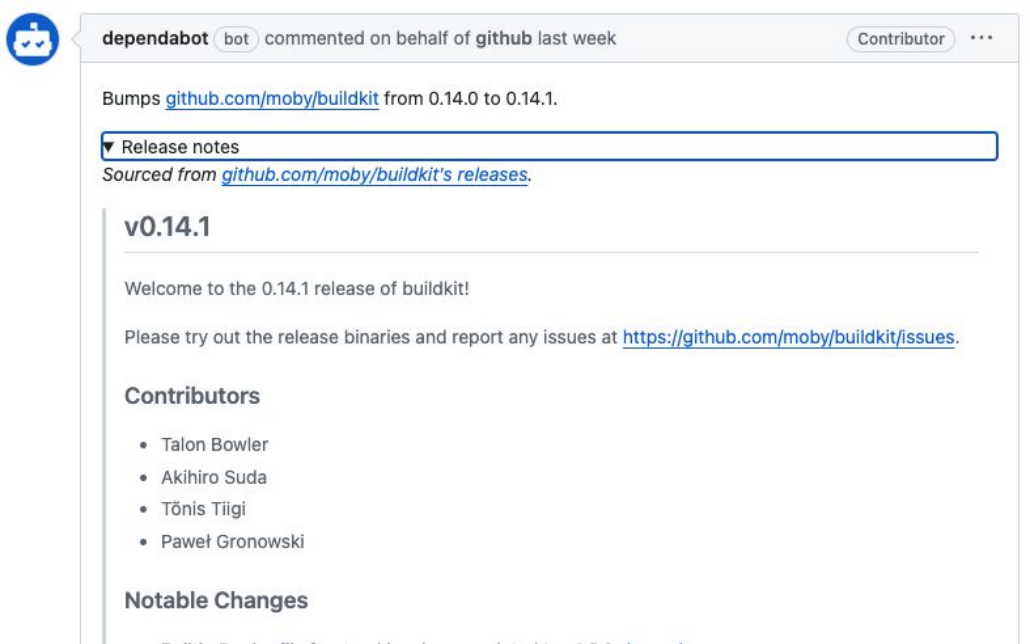
    - name: Set up Go
      uses: actions/setup-go@cdcb36043654635271a94b9a6d1392de5bb323a7 # v4
      with:
        go-version-file: 'go.mod'

    - name: setup task
      uses: arduino/setup-task@b91d5d2c96a56797b48ac1e0e89220bf64044611 # v1
      with:
        repo-token: ${{ secrets.GITHUB_TOKEN }}
```

Dependabot / Renovate

Not just for source code!

Can update Dockerfiles, GitHub Actions, etc...



dependabot bot commented on behalf of **github** last week — Contributor

Bumps github.com/moby/buildkit from 0.14.0 to 0.14.1.

▼ Release notes

Sourced from github.com/moby/buildkit's releases.

v0.14.1

Welcome to the 0.14.1 release of buildkit!

Please try out the release binaries and report any issues at https://github.com/moby/buildkit/issues.

Contributors

- Talon Bowler
- Akihiro Suda
- Tõnis Tiigi
- Paweł Gronowski

Notable Changes

Code Scanners

Tools like CodeQL or `govulncheck` can detect risky code patterns.

The best time to fix a CVE is at PR review time!

Dozens of contributed rules for common weaknesses:

CWE-20	JavaScript/TypeScript	js/count-untrusted-data-external-api	Frequency counts for external APIs that are used with untrusted data
CWE-20	JavaScript/TypeScript	js/incomplete-hostname-regexp	Incomplete regular expression for hostnames
CWE-20	JavaScript/TypeScript	js/incomplete-url-scheme-check	Incomplete URL scheme check
CWE-20	JavaScript/TypeScript	js/incomplete-url-substring-sanitization	Incomplete URL substring sanitization
CWE-20	JavaScript/TypeScript	js/incorrect-suffix-check	Incorrect suffix check
CWE-20	JavaScript/TypeScript	js/missing-origin-check	Missing origin verification in `postMessage` handler
CWE-20	JavaScript/TypeScript	js/regex/missing-regexp-anchor	Missing regular expression anchor
CWE-20	JavaScript/TypeScript	js/overly-large-range	Overly permissive regular expression range

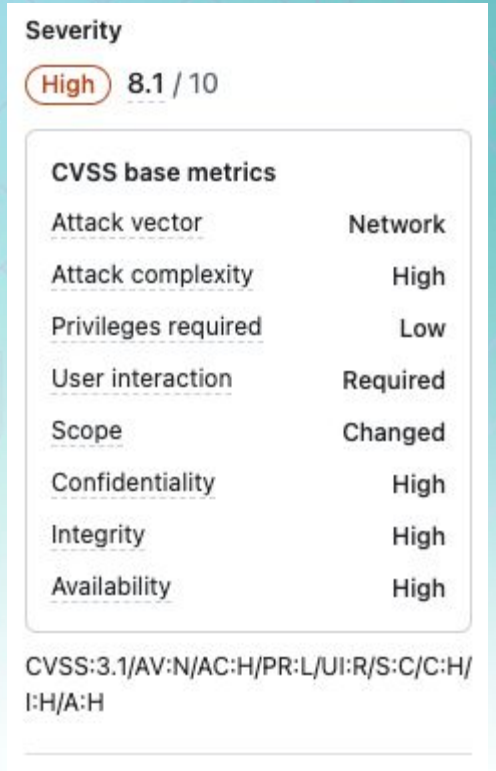
Vulnerability Reports

Private reporting of discovered vulnerabilities!

Collaborate on a fix in a private branch (even for public repos).

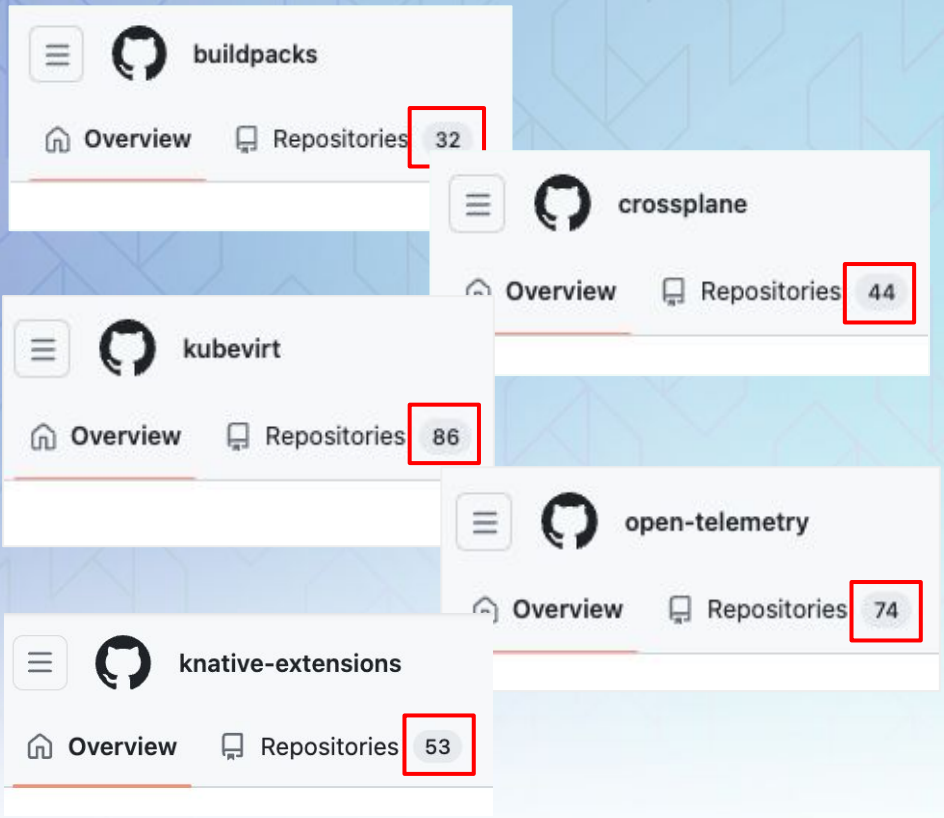
Easy to score the vulnerability and request a CVE.

Publish when the issue is fixed.



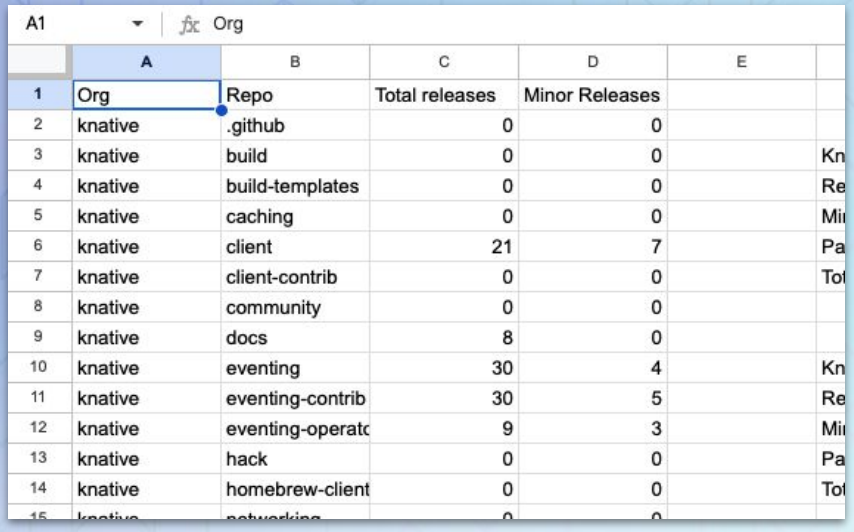
What’s the challenge?

Most CNCF projects have (or will have) lots of repos



- Repos tend to have wildly variable configuration
- Hard to keep track of what you *should* be doing
- Need to continuously manage as project grows and settings and best practices change

Manually configuring project repos is … not ideal



A1 | fx Org

	A	B	C	D	E
1	Org	Repo	Total releases	Minor Releases	
2	knative	.github	0	0	
3	knative	build	0	0	Kn
4	knative	build-templates	0	0	Re
5	knative	caching	0	0	Mi
6	knative	client	21	7	Pa
7	knative	client-contrib	0	0	Tot
8	knative	community	0	0	
9	knative	docs	8	0	
10	knative	eventing	30	4	Kn
11	knative	eventing-contrib	30	5	Re
12	knative	eventing-operatc	9	3	Mi
13	knative	hack	0	0	Pa
14	knative	homebrew-client	0	0	Tot
15	knative	networking	0	0	

Spreadsheets

```
for x in $(
     gh repo list knative |
       cut -d\  -f 1); do
  gh repo fork --clone knative/$x;
  (cd $x; cp /tmp/CODE_OF_CONDUCT.md
.;
    git commit -m “Update COC”
CODE_OF_CONDUCT.md;
    gh pr create -F /tmp/coc-body -f
  )
done
```

Bash scripts

stacklok

Agenda

Repository security configuration and management.

What is it? Why does it matter?

Automating and scaling security settings across your repositories.

What is Minder? How it interacts with my github repositories?

“You’ll be remembered by the rules you ~~break~~ make”

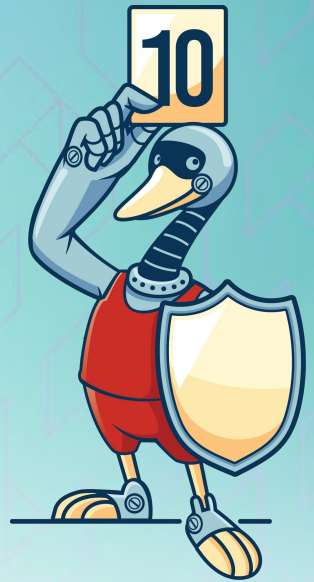
Learn to write your own rules for Minder using Rego.

You Don’t Have To Do It All At Once

Get a little bit better each month!

Use tools like OpenSSF Scorecard to assess your progress.

Tools like Minder can help make the process more uniform!





minder

Guardrails, Not Burden



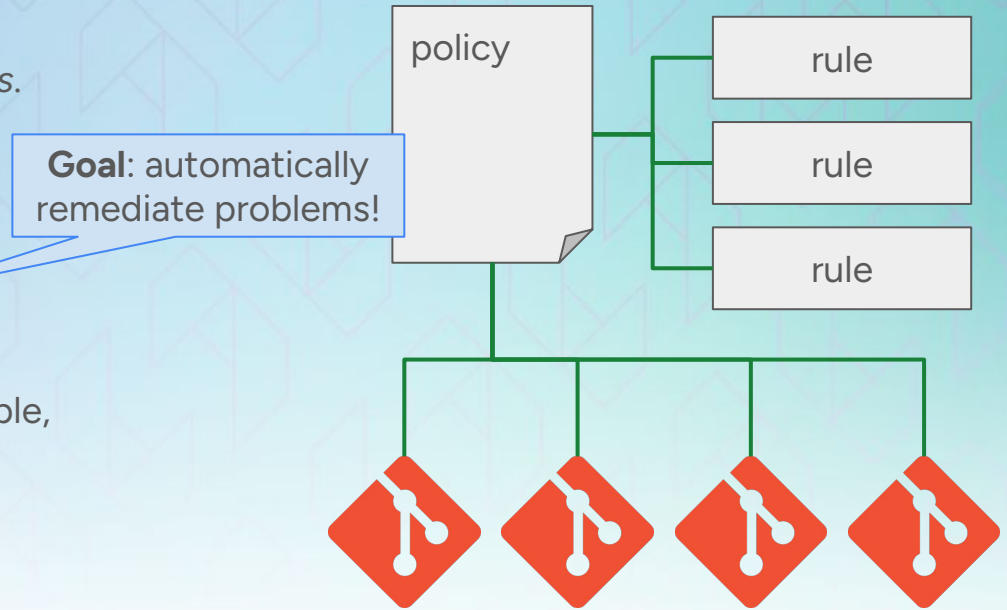
Minder is a tool to apply consistent security settings (*Profiles*) to multiple GitHub repos.

A Profile consists of zero or more *Rules*.

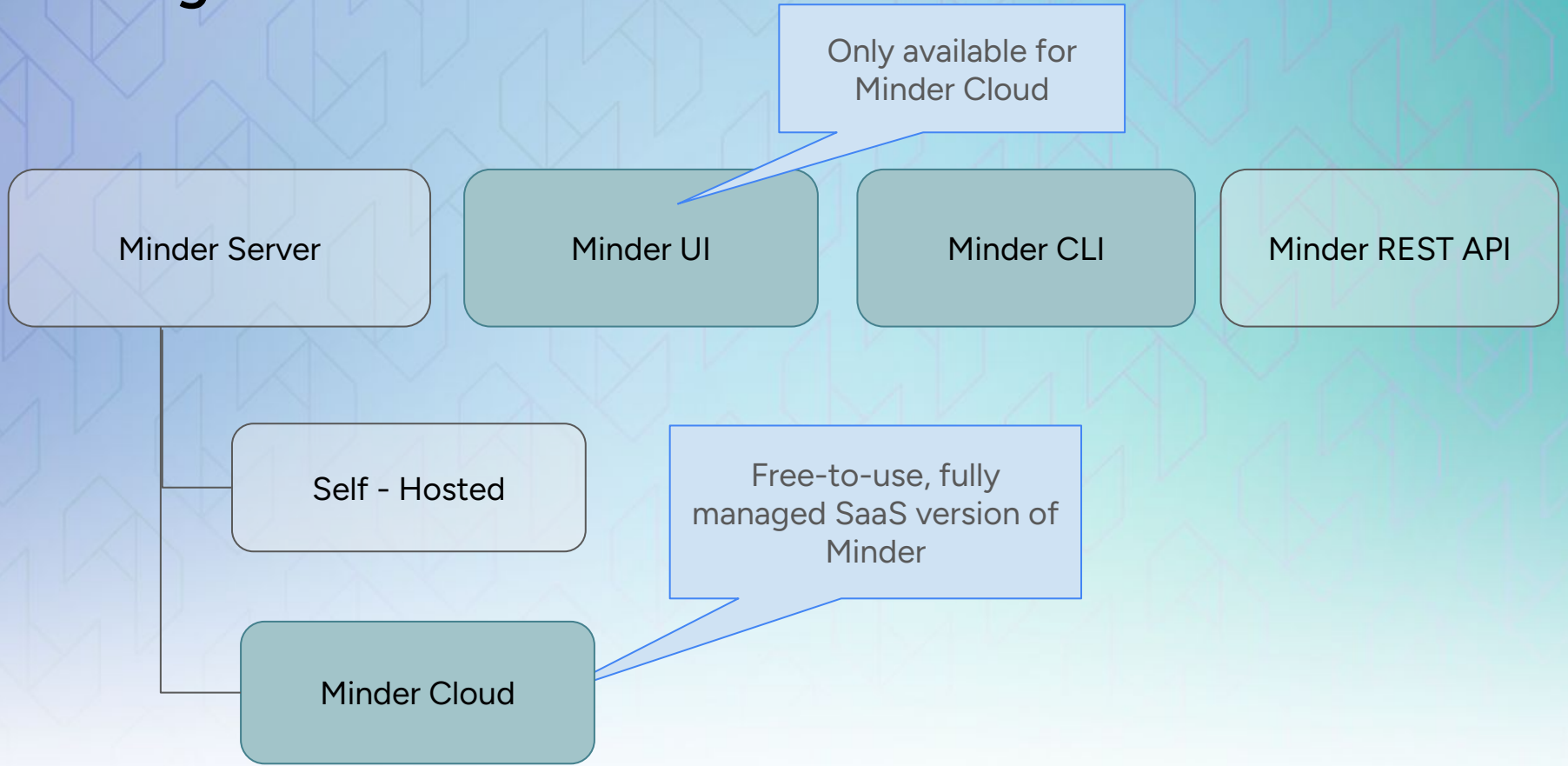
A Rule defines what an API field or GitHub file should match.

A Rule can also define a *remediation* to fix a mismatch.

Rules can have parameters – for example, the name of a dependency file or a language ecosystem.



Getting started with Minder



First Exercise

In this section you will

- Create Your Minder Cloud Account
- Setup your Minder Organization
- Enroll your Github Repositories with Minder
- Create your first Minder Profile
- Add managed rules to your Profile
- Enable auto-remediation for your Profile.

Setup: Select a GitHub Repository

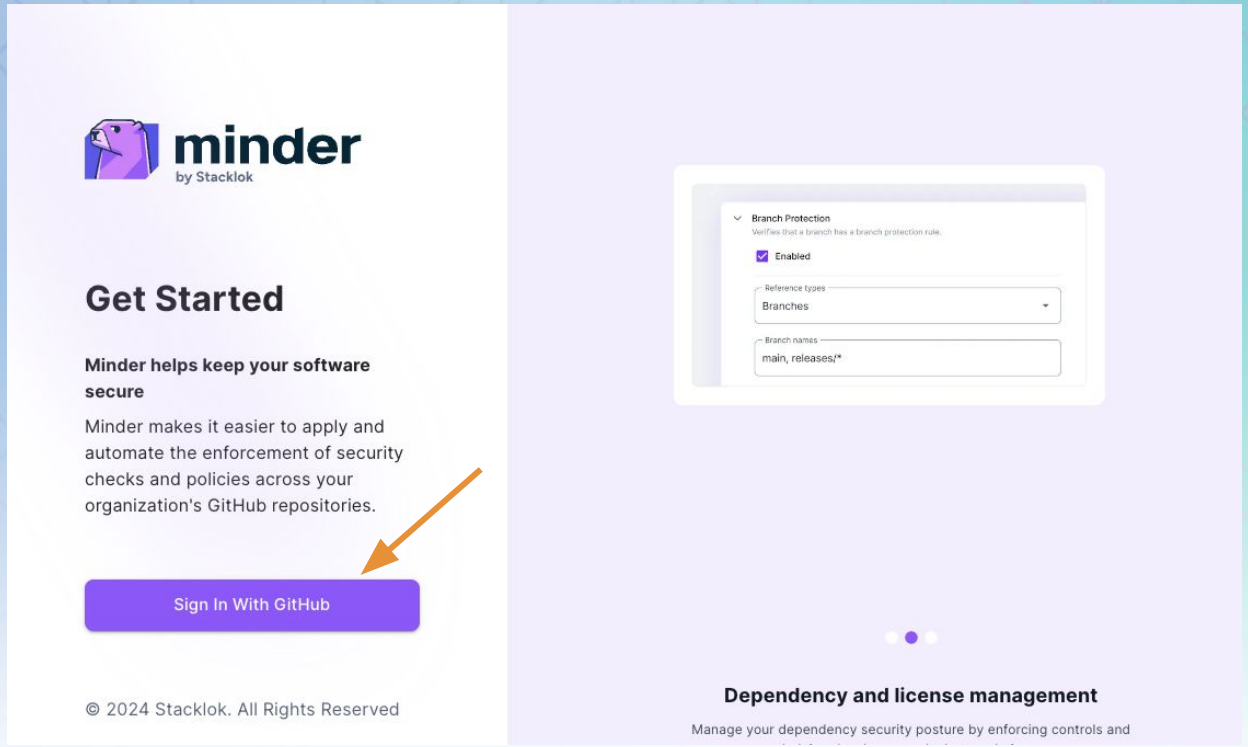
Choose a repo with at least one or two GitHub Actions (for the second part) to manage with Minder.

If you don’t have a repo in mind, consider forking

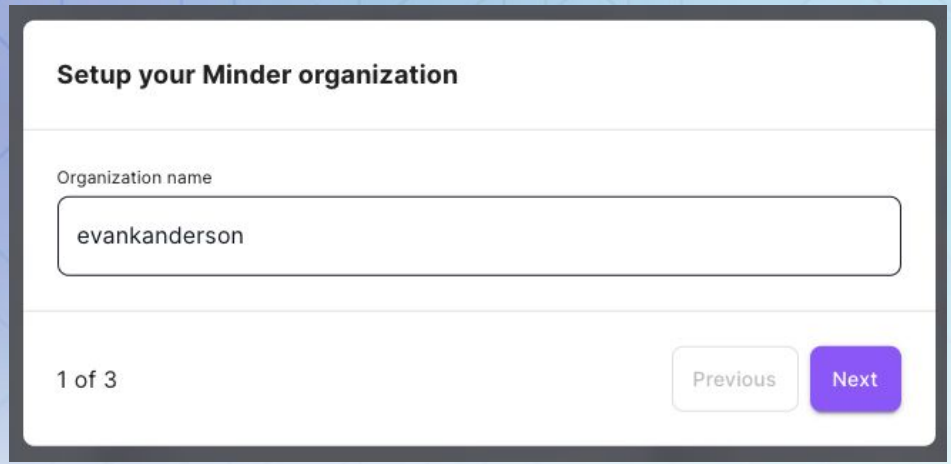
https://github.com/Stacklok-Demo-Org/frizbee

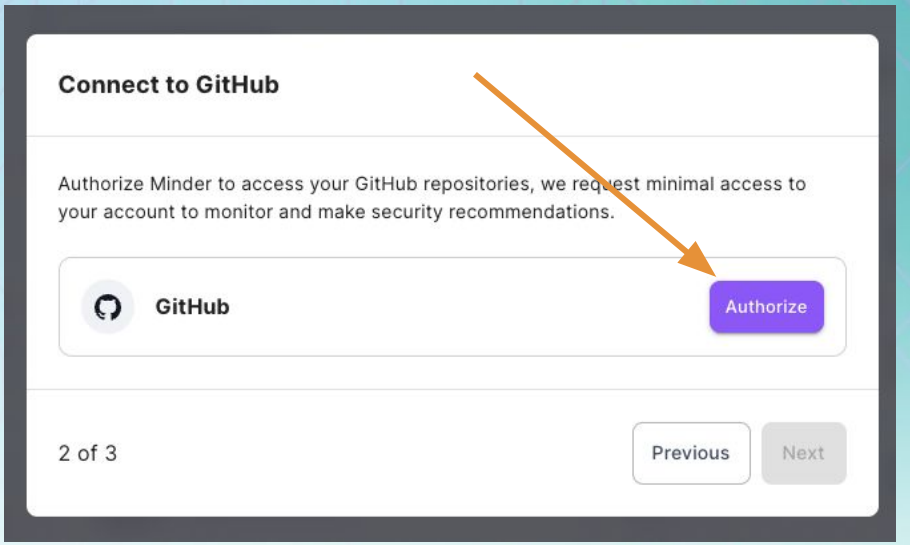
Create your Minder Cloud Account

Go to https://cloud.stacklok.com/

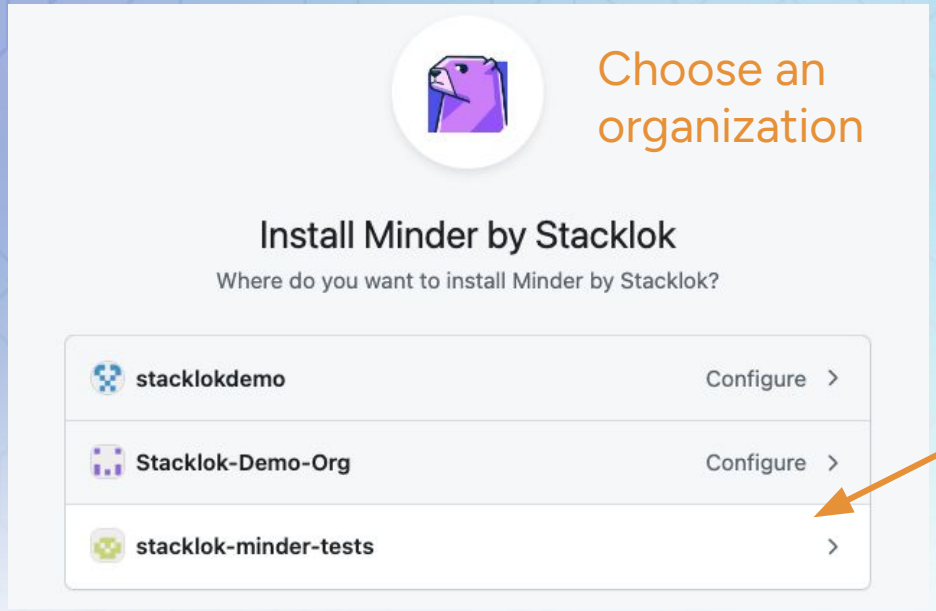


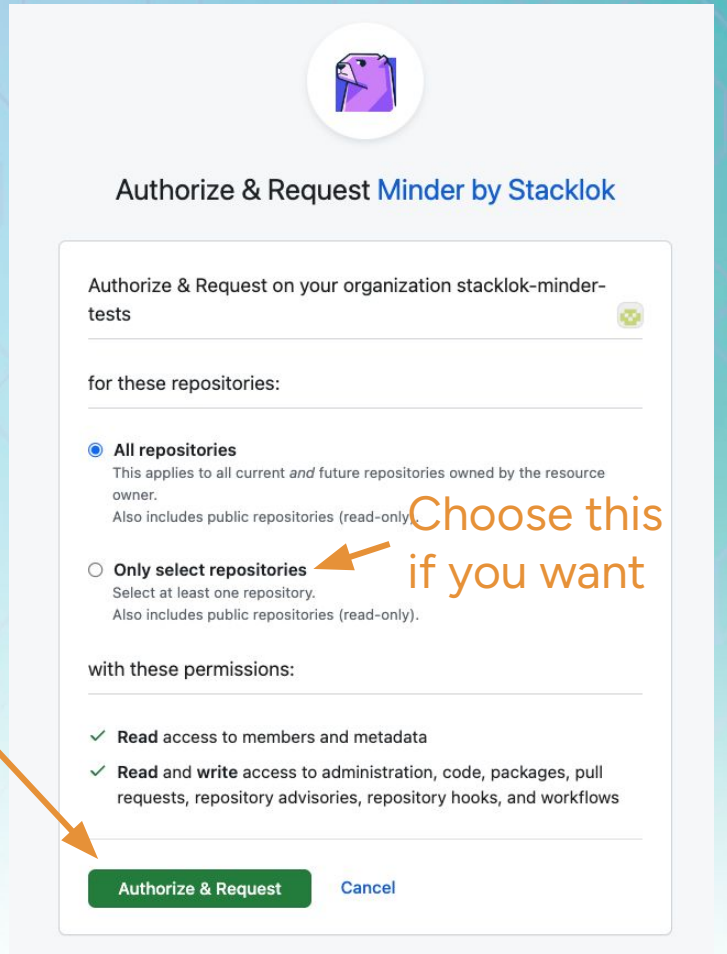
Create an Organization and Connect to Github



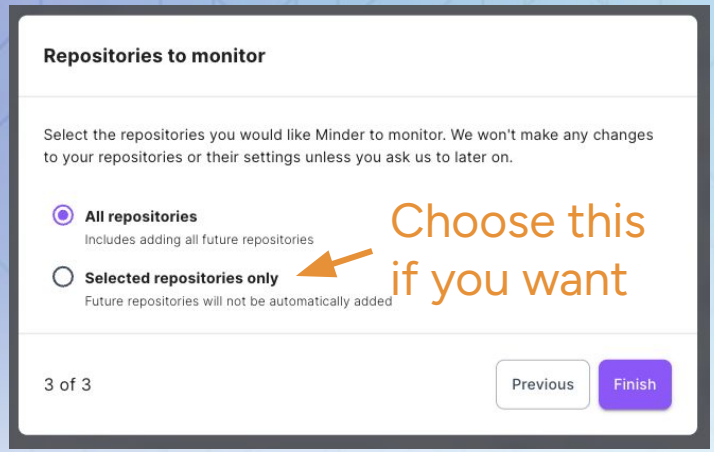


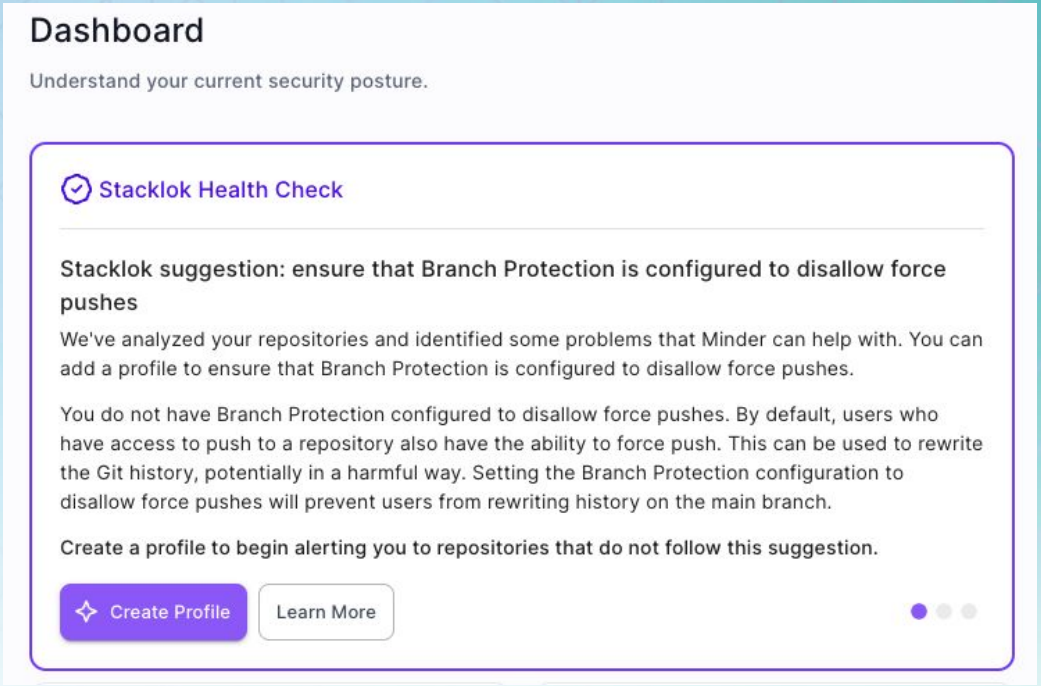
Authorize GitHub



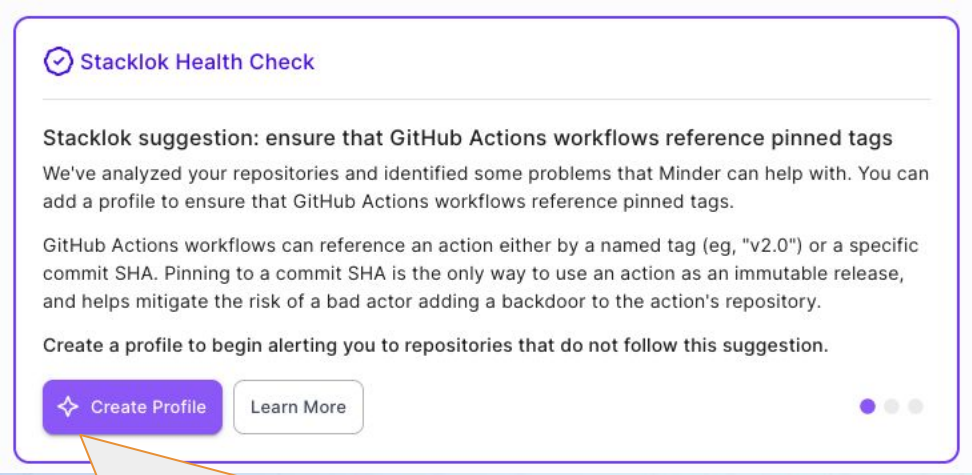


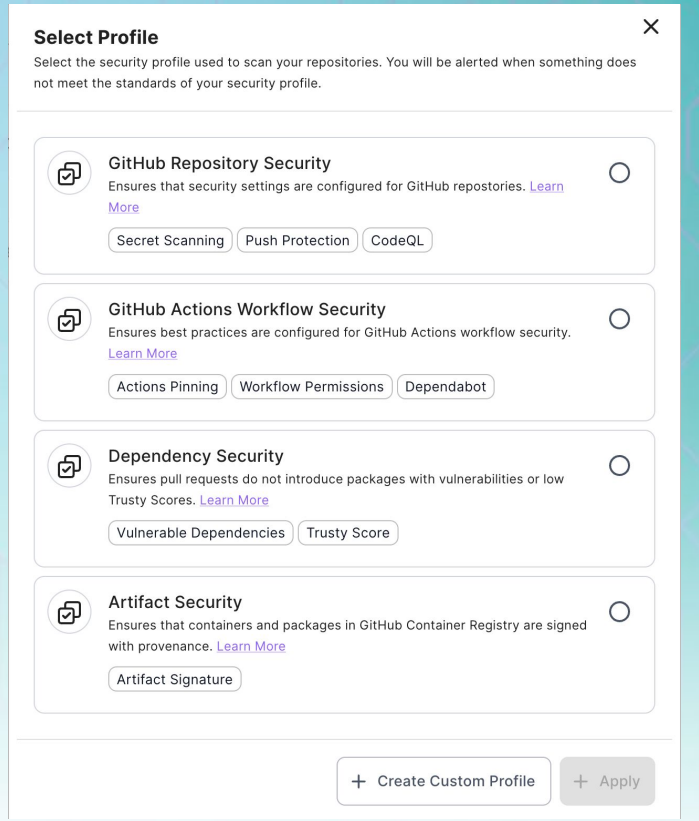
Enroll your Github Repository





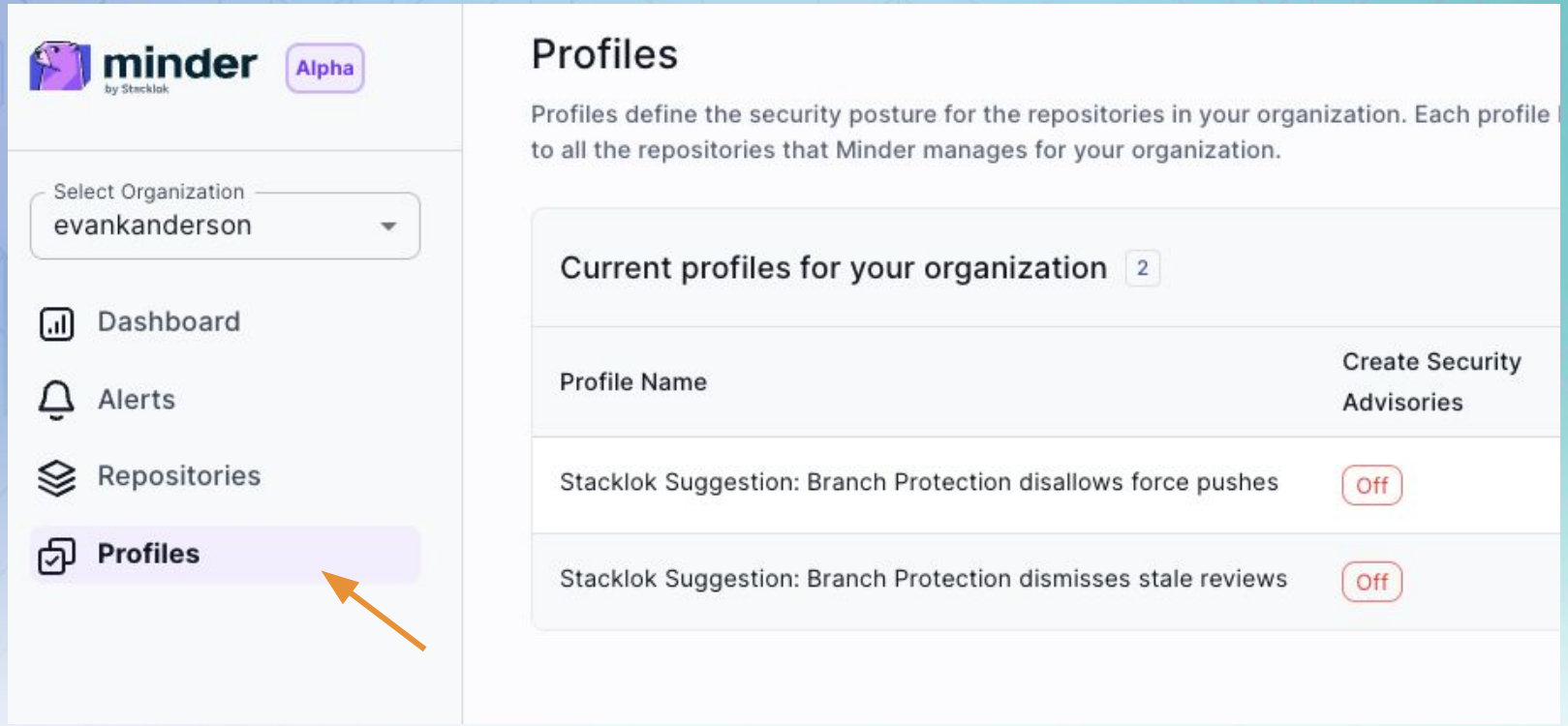
Creating Minder Profiles: Managed vs Custom





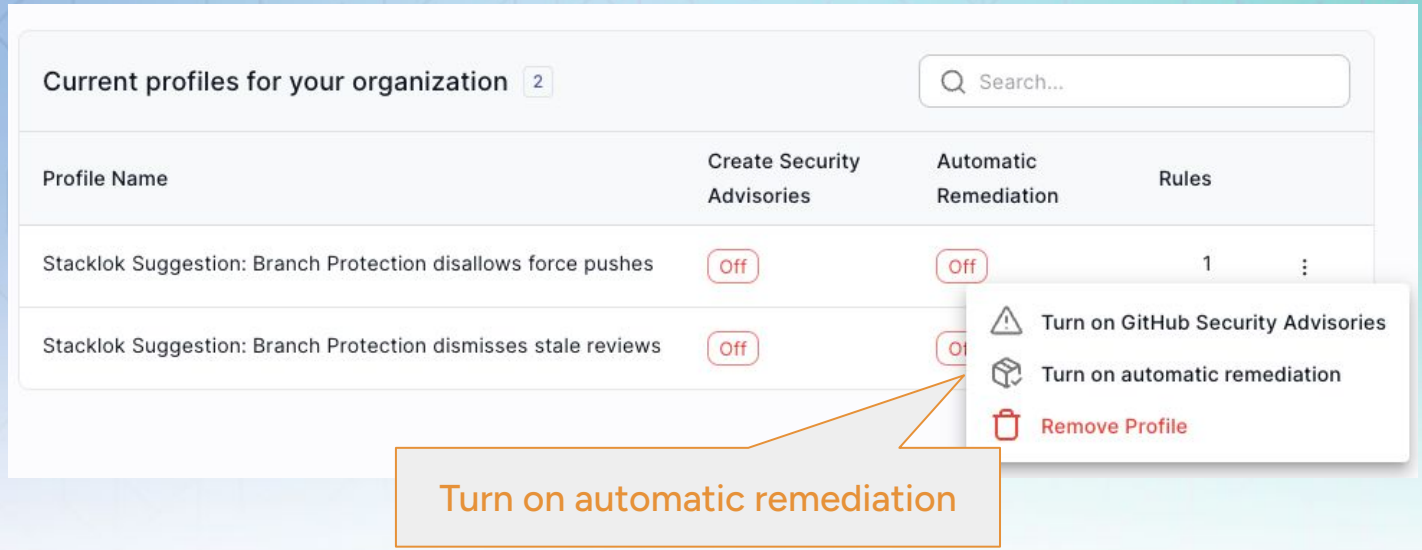
Feel free to accept some of these recommended profiles – don't worry, Minder won't make changes yet!

Exploring your Profiles

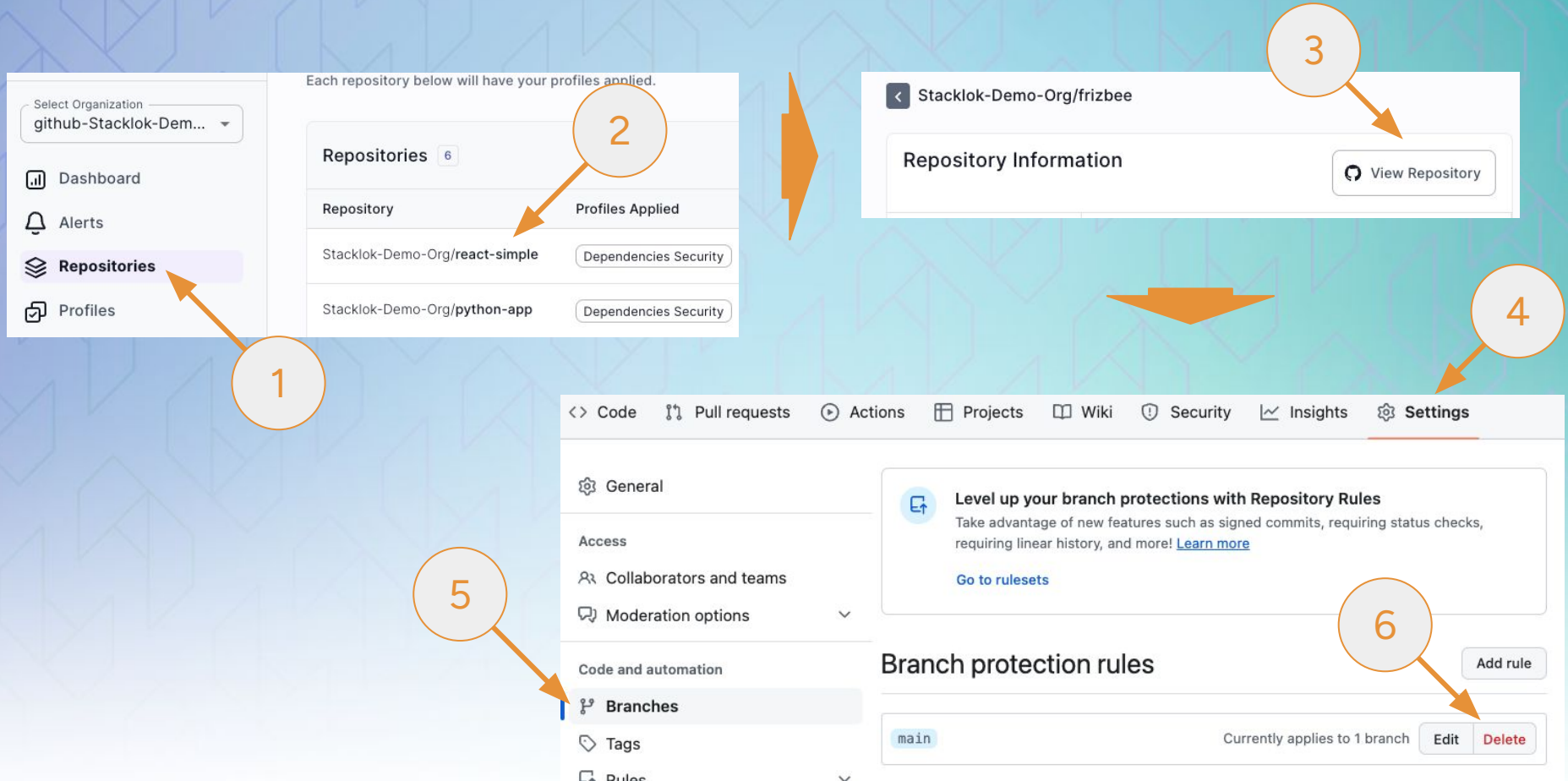


Enabling Autoremediation

Now, let's have Minder start enforcing branch protections on `main`:



Let's Test Auto-Remediation



Agenda

Repository security configuration and management.

What is it? Why does it matter?

Automating and scaling security settings across your repositories.

What is Minder? How it interacts with my github repositories?

“You’ll be remembered by the rules you ~~break~~ make”

Learn to write your own rules for Minder using Rego.

Extending Minder with your own custom rules

Rules are just instantiations of a Minder object known as **Rule Type**. Rule types definitions are checked in as a YAML file.

The rule type includes:

- Metadata (`version`, `name`, `severity`, etc)
- Input data (`ingest`, `in_entity`, `rule_schema`)
- Evaluation rules (`eval` – written in Rego, the same language as OPA)
- Correction actions (`alert`, `remediate`)



Anatomy of a Rule Type

Yaml

Copy Code

```yaml
---
version: v1
type: rule-type
name: my_cool_new_rule
context:
  provider: github
description: // Description goes here
guidance: // Guidance goes here
def:
  in_entity: repository  // what are we evaluating?
  param_schema: // parameters go here
  rule_schema: // rule definition schema goes here
  # Defines the configuration for ingesting data relevant for the rule
  ingest: // observed state gets fetched here
  eval: // evaluation goes here
  remediation: // fixing the issue goes here
  alert: // alerting goes here
```

Attacking GitHub Actions

Actions are a popular platform for CI (we talked about them for defense, earlier).

Stored in `.github/workflows/*.yaml` in your repository:

easy to code review, branch protect, etc!

But… they can also be a vulnerability vector!

https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions

Securing GitHub Actions

Threat: Dependency Attacks

Anyone who can change this git tag, can change your CI

```
steps:
 - name: Checkout
   uses: actions/checkout@v4
 - name: Security Scan
   uses: aquasecurity/trivy-action@0.23.0
```

Defense: *pin actions to commit SHA*

(Minder's frizbee integration does this)

Threat: Actions Mis-Use

Default permissions allow this read/write on deployments, etc.

```
steps:
 - name: Checkout
   uses: actions/checkout@v4
 - name: Security Scan
   uses: aquasecurity/trivy-action@0.23.0
```

Defense: *specify permissions explicitly*

(We're going to build this!)

GitHub Actions Pinning

First: make sure you trust the action *as it exists now*.

Second: replace the tag with a commit SHA.

Third: enable Dependabot to send update PRs.

```
actions/checkout@v4
⇒
actions/checkout@692973e3d937129bcbf40652eb9f2f61becf3332 # v4
```

You can help mitigate this risk by following these good practices:

- **Pin actions to a full length commit SHA**

  Pinning an action to a full length commit SHA is currently the only way to use an action as an immutable release. Pinning to a particular SHA helps mitigate the risk of a bad actor adding a backdoor to the action's repository, as they would need to generate a SHA-1 collision for a valid Git object payload. When selecting a SHA, you should verify it is from the action's repository and not a repository fork.

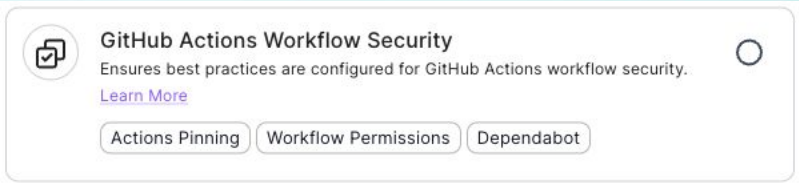
- **Audit the source code of the action**

  Ensure that the action is handling the content of your repository and secrets as expected. For example, check that secrets are not sent to unintended hosts, or are not inadvertently logged.

- **Pin actions to a tag only if you trust the creator**

  Although pinning to a commit SHA is the most secure option, specifying a tag is more convenient and is widely used. If you'd like to specify a tag, then be sure that you trust the action's creators. The 'Verified creator' badge on GitHub Marketplace is a useful signal, as it indicates that the action was written by a team whose identity has been verified by GitHub. Note that there is risk to this approach even if you trust the author, because a tag can be moved or deleted if a bad actor gains access to the repository storing the action.





GitHub Actions Permissions

By default, GitHub Actions have read/write permissions for most GitHub scopes!

EXCEPT `pull_request` has read-only

`pull_request_target` has read/write

This reduces the risk of a script injection compromising your repo, but removing permissions is *even safer*.

Scope	Default access (permissive)	Default access (restricted)	Maximum access for pull requests from public forked repositories
actions	read/write	none	read
attestations	read/write	none	read
checks	read/write	none	read
contents	read/write	read	read
deployments	read/write	none	read
discussions	read/write	none	read
id-token	none	none	read
	write	none	read
		read	read
	write	read	read
pages	read/write	none	read
pull-requests	read/write	none	read
repository-projects	read/write	none	read
security-events	read/write	none	read
statuses	read/write	none	read

Don't use this, it's dangerous

Detecting Rules with Default Permissions

GitHub Actions can have permissions at two different levels:

Top-Level: `permissions` in the YAML file

- Applies to all jobs in the file that don't have their own permissions.

Job-Level: `jobs.<job_id>.permissions`

- Applies to one specific job.

⇒ Find workflows without `permissions` where at least one job doesn't have `jobs.<job_id>.permissions`

Planning our Rule Type

Input data: `ingest`

- `type: git` (load files from repo)
- `type: rest` (load data from an API call)
- `type: diff` (for PRs)

Rule evaluation: `eval`

- `type: jq` (select data with JQ expressions)
- `type: rego` (evaluate data with Rego)

Second Exercise

In this section you will

- Install and use the Minder CLI.
- Define a custom Minder Rule.
- Apply the custom rule to your profile.

Setup: Installing the Minder CLI

https://minder-docs.stacklok.dev/getting_started/install_cli

Mac: `brew install stacklok/tap/minder`

Windows: `winget install stacklok.minder`

Try the following commands:

```
$ minder auth login

$ minder profile list

$ minder profile status list --detailed \
    -n repository-github-profile
```

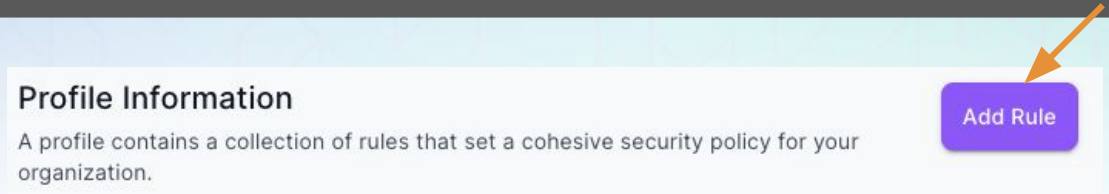
Writing a Rule Type

Rule Types need to be created through the CLI:

```
$ minder ruletype apply -f rule_file.yaml
```

And then applied by a profile with either the CLI or a custom profile in the GUI:

```
$ minder profile apply -f profile.yaml
```



Building a Custom Rule

Documentation:
https://minder-docs.stacklok.dev/how-to/custom-rules

Examples:
https://github.com/stacklok/minder-rules-and-profiles/

Hint:
`dependabot.yaml` and `trivy_action_enabled.yaml` parse yaml from git.

Rego Hints!

```
# You can use helper function for “or”:

permissions_ok(workflow) if {
      has_permissions(workflow)
}

permissions_ok(workflow) if {
      every job in workflow.jobs {
            has_permissions(job)
      }
}

# Check if a map exists:
count(foo) >= 0
```

https://play.openpolicyagent.org/

Remediating Issues

Defining the actions that fix the issues

Minder has the ability to automatically fix the issues it encounters when applying our custom rule. To enable this capability we need to specify the remediation steps as part of our custom rule definition.

Remediations also have different types, like pull-requests or rest for API Calls

```
---
def:
  remediate:
    type: rest
    rest:
      method: PATCH
      endpoint: "/repos/{{.Entity.Owner}}/{{.Entity.Name}}"
      body: |
        { "delete_branch_on_merge": {{ .Profile.enabled }} }
```

Current Limits of Remediations

Right now, minder can do the following remediations:

- `rest`: Make an HTTP request to an `endpoint` with a `body` and `headers`.
  Current limits: can only make a single request
- `pull_request`: Send a pull request to the repo for a human to review
  Current limits: can only replace the contents of a single file
- `gh_branch_protection`: Patch the set of branch protections (legacy)

Most of the fields in remediations support go templates to include parameters.

https://minder-docs.stacklok.dev/ref/proto#ruletypedefinitionremediate

Third Exercise

In this section you will

- Add autoremediation to a new Custom Rule Type.
- Test auto-remediation.

Let's Add a SECURITY.md!

Since the previous rule scans a number of actions, we'd need to PR a change to several files, which is not currently supported.

Instead, let's define a baseline `SECURITY.md` file to indicate how to file vulnerability reports.

dependabot_configured.yaml has an example remediation for creating a file.

Configuring File Remediation

```
remediate:
  type: pull_request
  pull_request:
    title: "Add Security reporting”
    body: |
      This is a Minder automated pull request.

      This pull request adds a SECURITY.md file to the repository.  For more information, see
https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repos
itory
    contents:
    - path: .github/dependabot.yml
      action: replace
      content: |
        # Security Policy

        Fill in some content
```

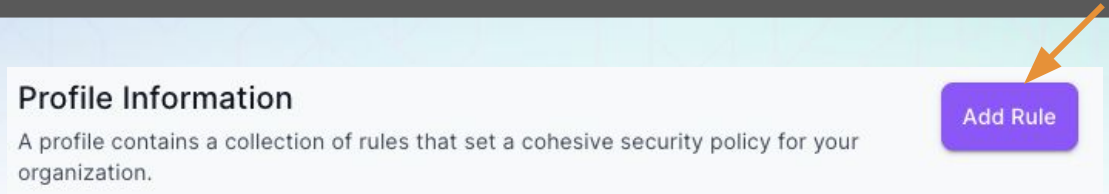
Loading Your Rule Type

Rule Types need to be created through the CLI:

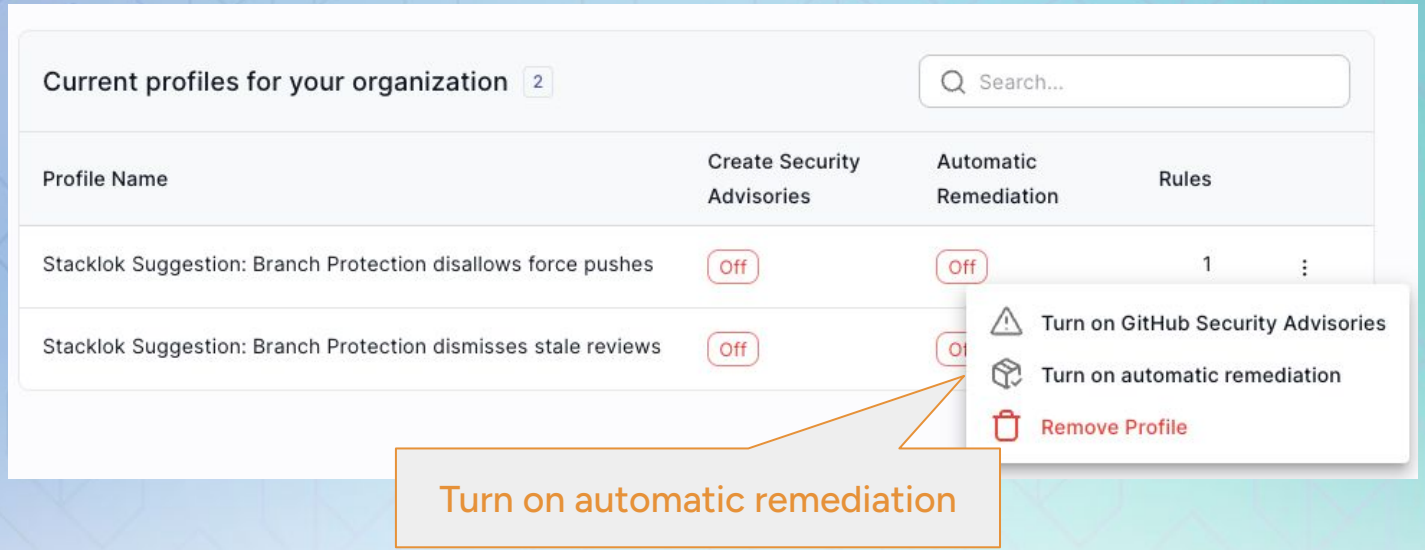
```
$ minder ruletype apply -f security_file.yaml
```

And then applied by a profile with either the CLI or a custom profile in the GUI:

```
$ minder profile apply -f profile.yaml
```



Turn on Automatic Remediation



Thank you!

For questions, issues or to follow along our community conversations

- stacklok.com
- github.com/stacklok
- discord.gg/stacklok
- youtube.com/@stacklok
- meetup.com/stacklok-user-group

Q&A