ThoughtWorks®

Cabinet Office

Tin Tulip - Blue team

Showcase #9 - June 23

Agenda

What we achieved

Next scenarios

Summary

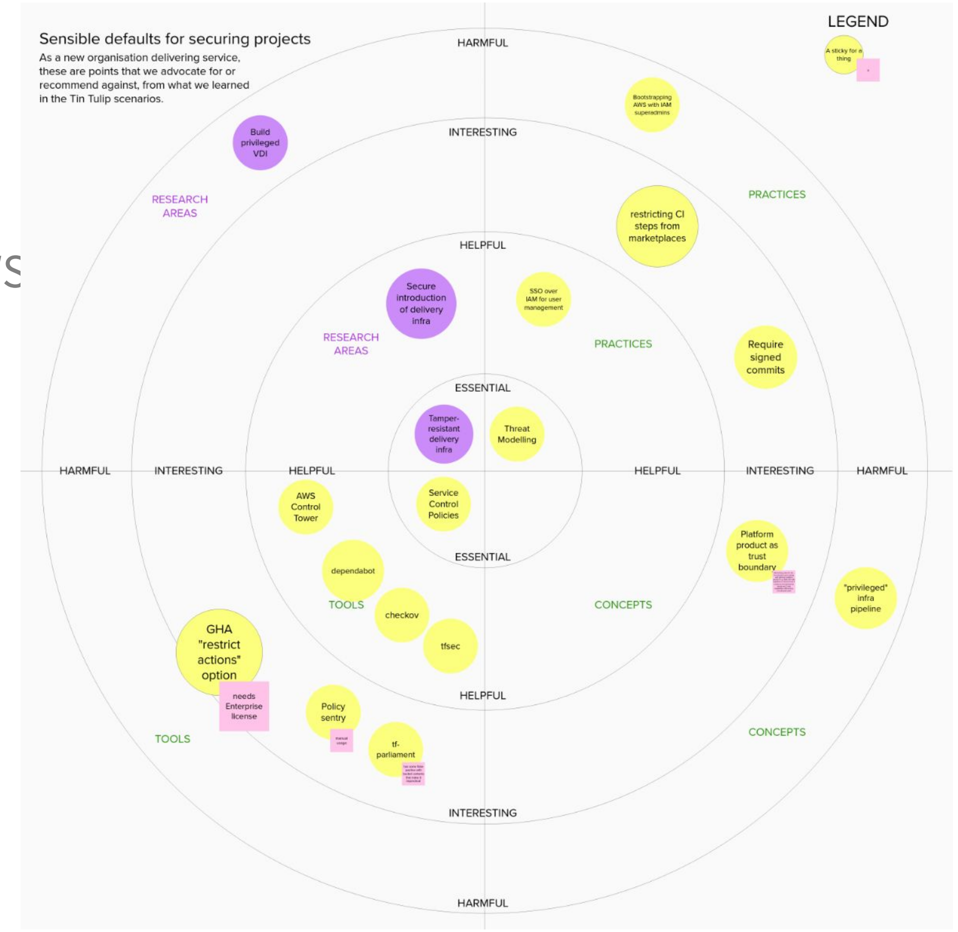
CLA's environment is undergoing testing of Scenario 1.

Blue and Red are planning the scenarios to cover in the next 5 weeks.

What we achieved

What we worked on

- Restricting Workload Egress
- Team Mozart (Web) can now access AWS
- Add RDS to the workload
- Apply for a creative license!
- Policy as code checks for Terraform


Sensible defaults for securing projects

As a new organisation delivering service, these are points that we advocate for or recommend against, from what we learned in the Tin Tulip scenarios.

Zoomable diagram available on Mural at this link (signup required)

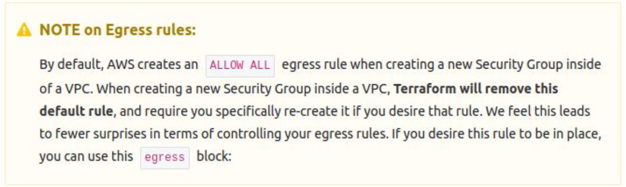
Restricting Workload Egress

What we built:
Removed internet egress rules for all SGs in Workloads accounts. (or so we thought!)

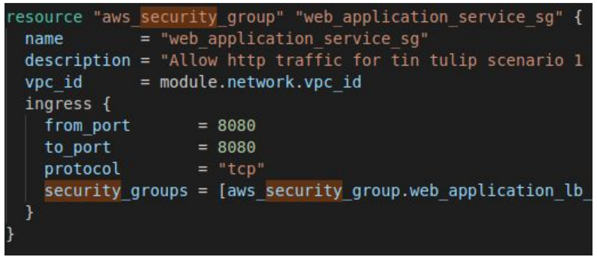
Why we built it:
CLA's service has no need to connect to the internet, enable us to implement this control.

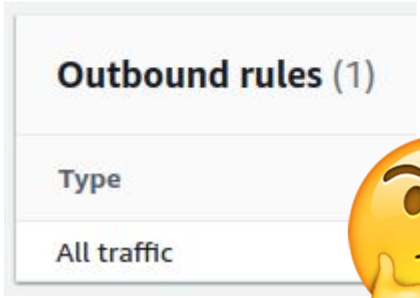
What we learned from it:

- Using SG-to-SG rules is easy
- Immutable infrastructure is an enabler of egress restrictions
- ~~tfsec/Checkov are a great way to not miss these opportunities~~
- There is value on checking both src and cloud config
- There's some weirdness around how TF executes this!


⚠ **NOTE on Egress rules:**

By default, AWS creates an `ALLOW ALL` egress rule when creating a new Security Group inside of a VPC. When creating a new Security Group inside a VPC, **Terraform will remove this default rule**, and require you specifically re-create it if you desire that rule. We feel this leads to fewer surprises in terms of controlling your egress rules. If you desire this rule to be in place, you can use this `egress` block:


```
resource "aws_security_group" "web_application_service_sg" {
  name        = "web_application_service_sg"
  description = "Allow http traffic for tin tulip scenario 1
  vpc_id      = module.network.vpc_id
  ingress {
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.web_application_lb_
  }
}
```


Outbound rules (1)

Type
All traffic

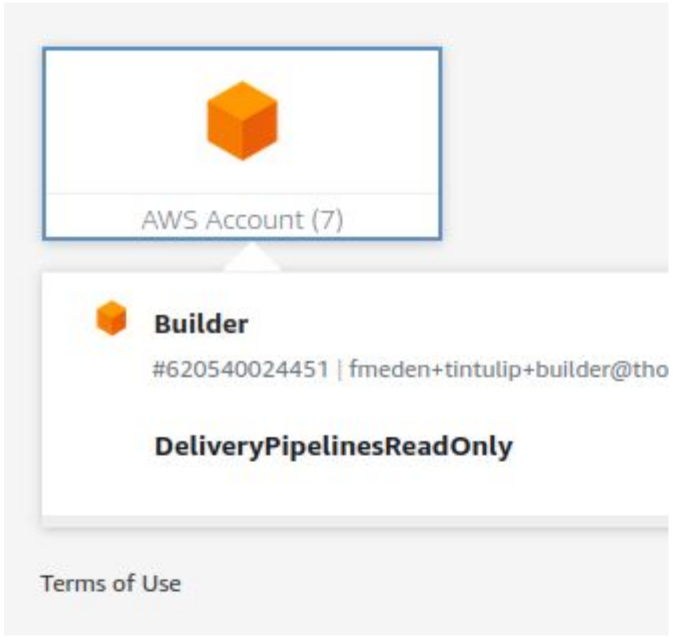
Team Mozart (Web) can now access AWS

What we built:
AWS SSO PermissionSet for team Mozart (Web)

Why we built it:
Webapp goes through a "trusted pipeline" to be deployed, developers now have a need to observe build logs.

What we learned from it:

- SSO is a good tool to allow fine-grained permissions.


AWS Account (7)

Builder

#620540024451 | fmeden+tintulip+builder@tho

DeliveryPipelinesReadOnly

Terms of Use

Add RDS to the workload

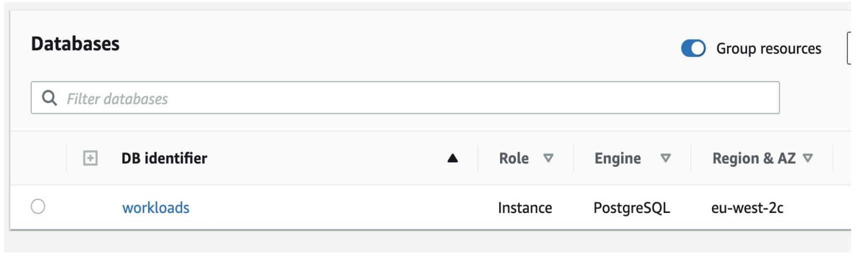
What we built:
A Postgres database instance that uses a secret manager to store the database credentials.

Why we built it:

To store application details from the creative license page.

What we learned from it:

- Easy to use with secrets manager and kms to encrypt the password for the database.
- Database instance can only be encrypted when created.



Databases

DB identifier	Role	Engine	Region & AZ
workloads	Instance	PostgreSQL	eu-west-2c

Resource available at this Link

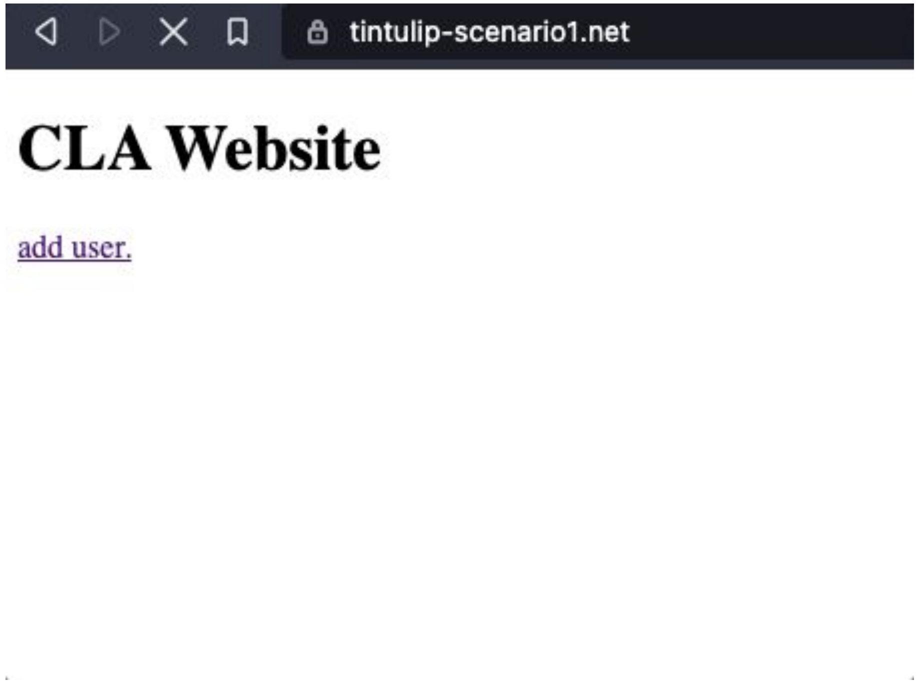
Applying for a creative license

What we built:
A sign up page that allows creatives to apply for a particular license.

Why we built it:
Provides the interface to store data in the database.

What we learned from it:

- End to end verification to ensure that the data input is stored after being processed by the microservice.


tintulip-scenario1.net

CLA Website

add user.

Resource available at this Link

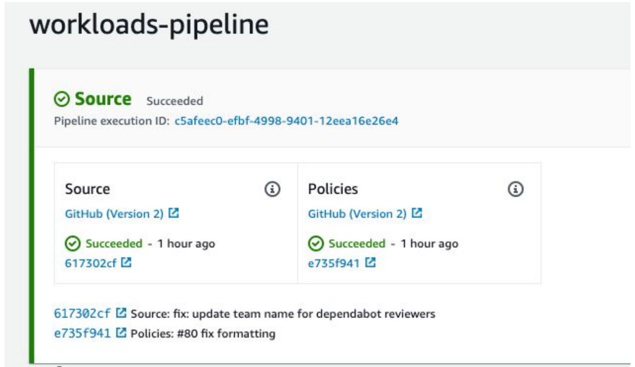
Policy-as-Code Terraform Checks

What we built:
Added additional checks that prevent local-exec from being used and ensure roles can only be assumed from the builder account

Why we built it:
Introduced a set of rules that check within a trusted environment to detect misconfigurations within Terraform files

What we learned from it:

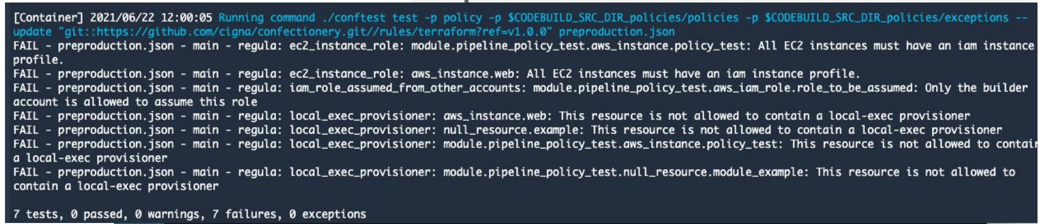
- Introduced rego - interesting syntax as well as learning curve
- Tests run against the (JSON) plan and include all things in state
- Policies live in a separate repository to source code
- Using Confectionery (library of standardized rules) can help speed up catching misconfigurations


workloads-pipeline

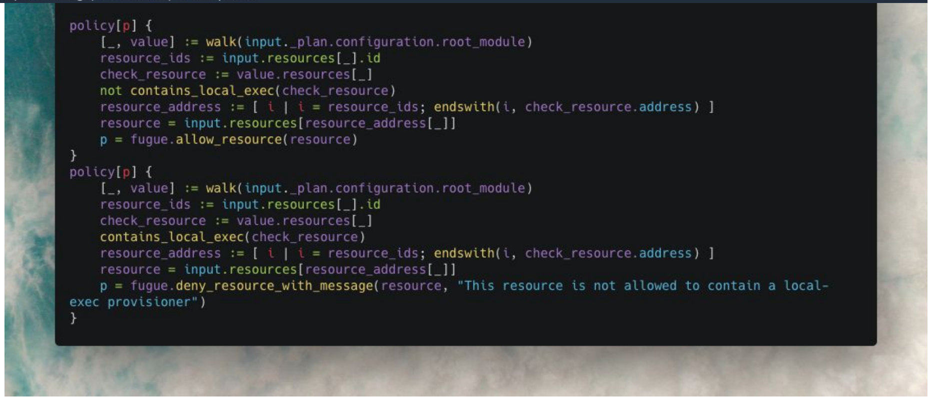
Source Succeeded
Pipeline execution ID: c5afeec0-efbf-4998-9401-12eea16e26e4

Source	Policies
GitHub (Version 2)	GitHub (Version 2)
Succeeded - 1 hour ago	Succeeded - 1 hour ago
617302cf	e735f941

617302cf Source: fix: update team name for dependabot reviewers
e735f941 Policies: #80 fix formatting


```
[Container] 2021/06/22 12:00:05 Running command ./conftest test -p policy -p $CODEBUILD_SRC_DIR_policies/policies -p $CODEBUILD_SRC_DIR_policies/exceptions --
update "git::https://github.com/cigna/confectionery.git//rules/terraform?ref=v1.0.0" preproduction.json
FAIL - preproduction.json - main - regula: ec2_instance_role: module.pipeline_policy_test.aws_instance.policy_test: All EC2 instances must have an iam instance
profile.
FAIL - preproduction.json - main - regula: ec2_instance_role: aws_instance.web: All EC2 instances must have an iam instance profile.
FAIL - preproduction.json - main - regula: iam_role_assumed_from_other_accounts: module.pipeline_policy_test.aws_iam_role.role_to_be_assumed: Only the builder
account is allowed to assume this role
FAIL - preproduction.json - main - regula: local_exec_provisioner: aws_instance.web: This resource is not allowed to contain a local-exec provisioner
FAIL - preproduction.json - main - regula: local_exec_provisioner: null_resource.example: This resource is not allowed to contain a local-exec provisioner
FAIL - preproduction.json - main - regula: local_exec_provisioner: module.pipeline_policy_test.aws_instance.policy_test: This resource is not allowed to contai
a local-exec provisioner
FAIL - preproduction.json - main - regula: local_exec_provisioner: module.pipeline_policy_test.null_resource.module_example: This resource is not allowed to
contain a local-exec provisioner

7 tests, 0 passed, 0 warnings, 7 failures, 0 exceptions
```


```
policy[p] {
    [_, value] := walk(input._plan.configuration.root_module)
    resource_ids := input.resources[_].id
    check_resource := value.resources[_]
    not contains_local_exec(check_resource)
    resource_address := [ i | i = resource_ids; endswith(i, check_resource.address) ]
    resource = input.resources[resource_address[_]]
    p = fugue.allow_resource(resource)
}
policy[p] {
    [_, value] := walk(input._plan.configuration.root_module)
    resource_ids := input.resources[_].id
    check_resource := value.resources[_]
    contains_local_exec(check_resource)
    resource_address := [ i | i = resource_ids; endswith(i, check_resource.address) ]
    resource = input.resources[resource_address[_]]
    p = fugue.deny_resource_with_message(resource, "This resource is not allowed to contain a local-
exec provisioner")
}
```

Resource available at this Link

Next scenarios

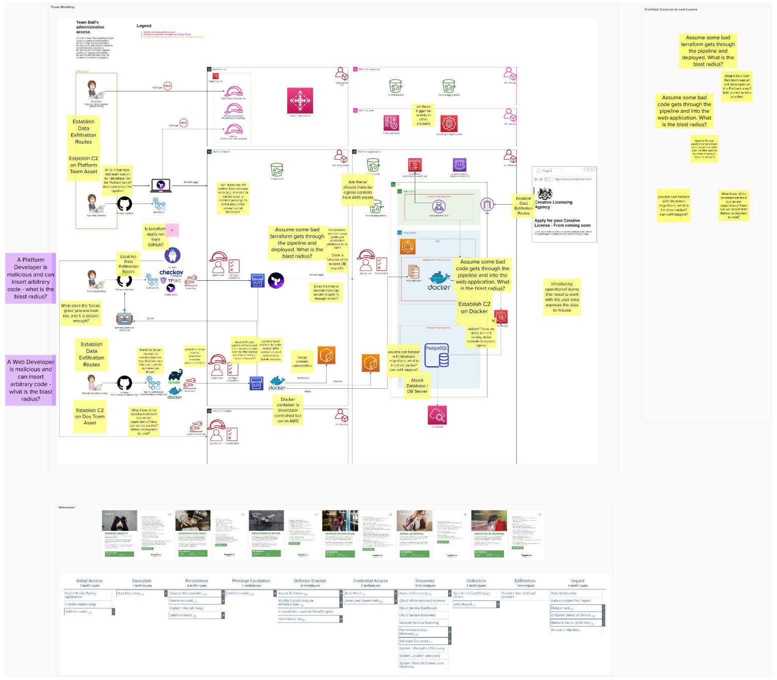
Prioritised scenario list

To cover in next 5 weeks:

- □Assume some bad terraform gets through the pipeline and deployed. What is the blast radius?
- What if Rob from Red team was an evil developer on the Platform team? Add control to infra pipeline
- Assume some bad code gets through the pipeline and into the web-application. What is the blast radius?

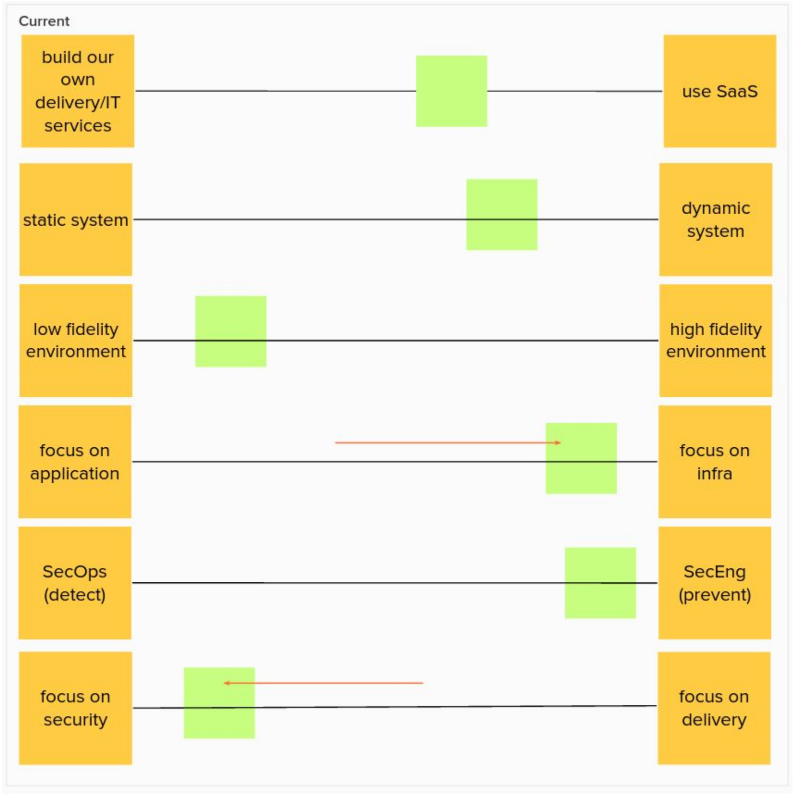
Stretch goals:

- Assume the app pipeline compromised, can it escalate to take over the infra pipeline by virtue of being in the same account?
- assume can tamper with database migrations, what is the blast radius? can exfil happen?
- What if one of the development team add an evil dependency? How can we detect that? Before deployment to prod?



A Platform Developer is malicious and can insert arbitrary code - what is the blast radius?
A Web Developer is malicious and can insert arbitrary code - what is the blast radius?
Establish Data Exfiltration Routes
Establish C2 on Platform Team Asset
Establish C2 on Dev Team Asset
Assume some bad terraform gets through the pipeline and deployed. What is the blast radius?
Assume some bad code gets through the pipeline and into the web-application. What is the blast radius?
Establish C2 on Docker
Attack Database / DB Server
Docker container is developer controlled but run in AWS
checkov
docker
Creative Licensing Agency

Tradeoff Sliders review



- Focus on infrastructure pipelines
- Focus on controls rather than building out more infrastructure

Sliders tracker (link requires access):

https://app.mural.co/t/thoughtworksclientprojects1205/m/thoughtworksclientprojects1205/1620729955822

Appendix: Guiding Principles

Guiding principle for the project

Does this teach us something new about a security control, or how to defeat it?

Guiding principle for platform implementation

In order to research the known security boundaries, the blue team will implement a test platform based on published best practices, including those published by the NCSC

Guiding principle for communicating learnings

The key audience for learnings are government departments, who want to empower their local technology teams to deliver secure systems