SIGPwny

FA2023 Week 15 • 2023-12-03

Cloud and Modern Infrastructure Security

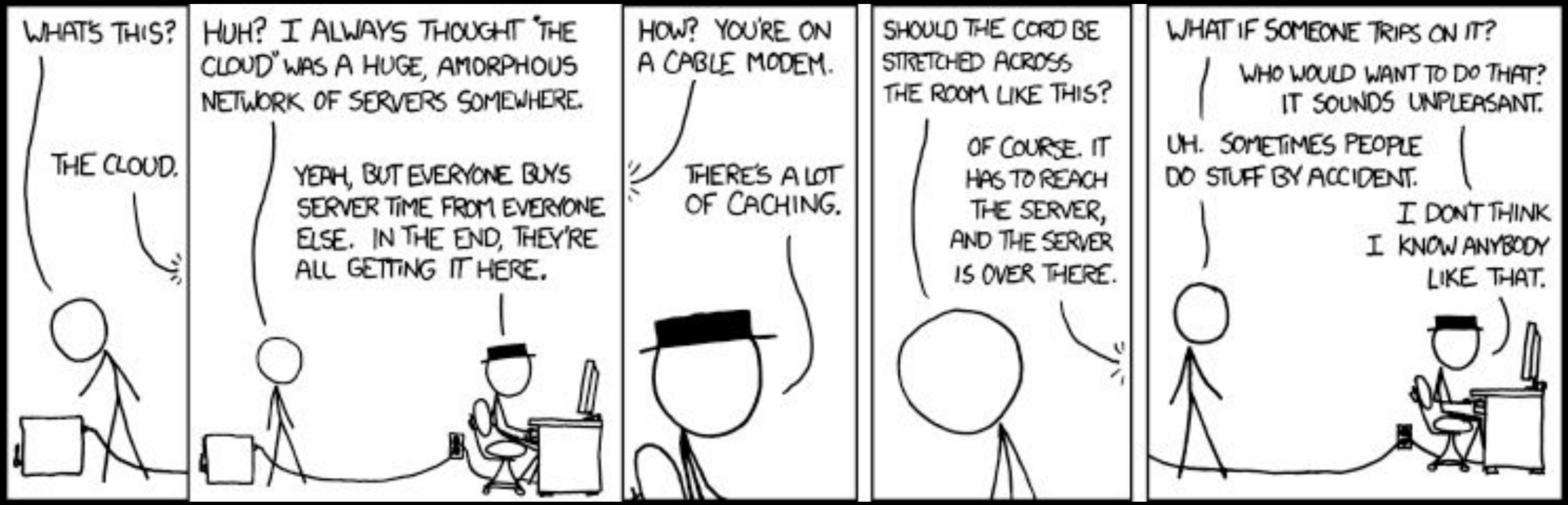
Sagnik Chakraborty

Announcements

- This Thursday, we might have a chill party or study session!

ctf.sigpwny.com

sigpwny{4_cl0udy_d4y_or_4_z3ro_d4y}



General Overview

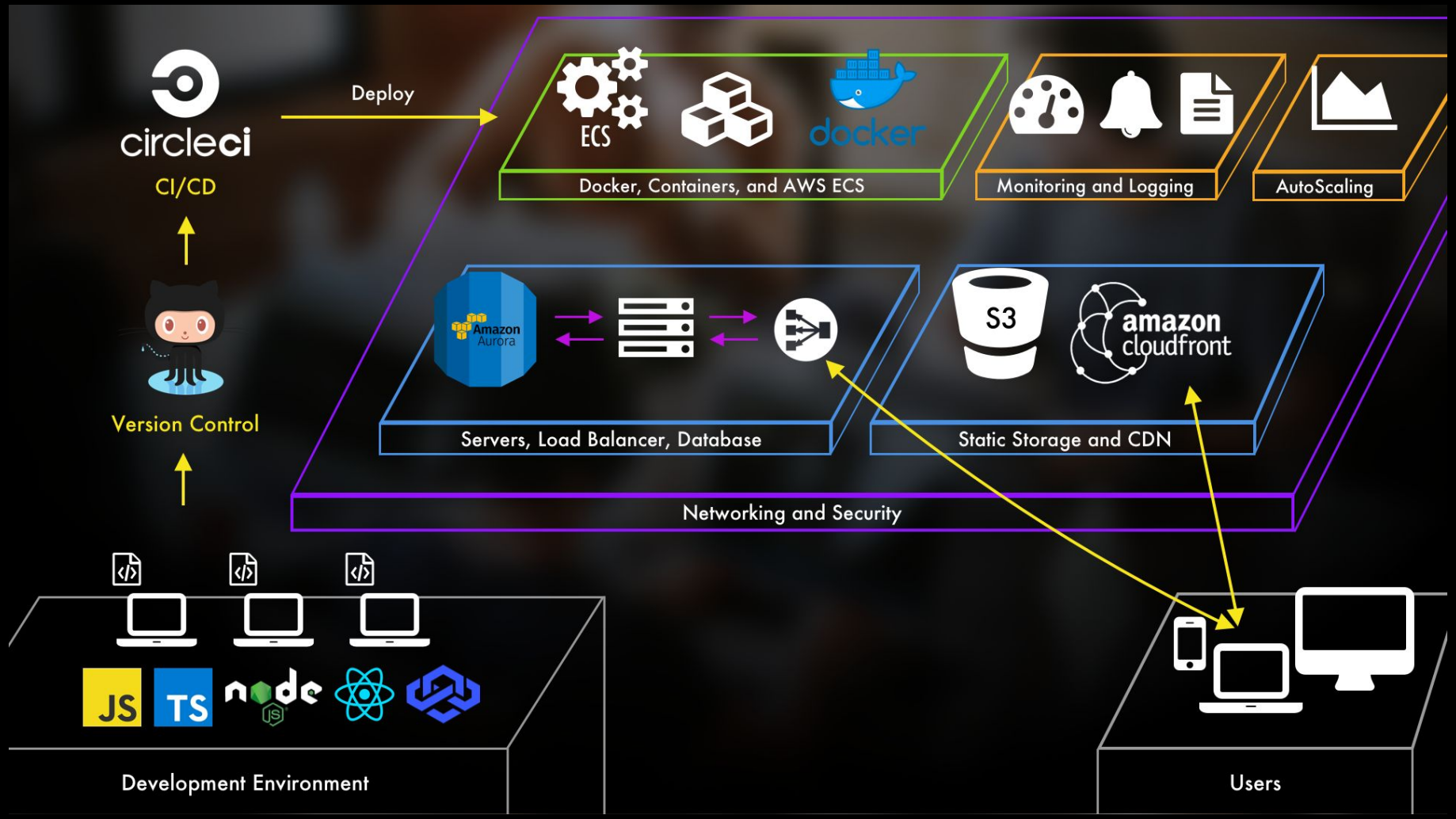
- Cloud-based Toolchains
  - Overview
  - Terraform
  - Uncovering Cloudflare
- CI/CD pipelines
  - Overview
  - Methodology
  - Artifact Poisoning

The Don’t Go To Jail Disclaimer

- This has been said in the beginning of the year, but beware that you cannot access information that isn’t allowed to be accessed by you without explicit permission from the distributors

- Even if it is in good faith, you may be held liable for any damages that result from poking around places you shouldn’t be in!

What exactly is the cloud

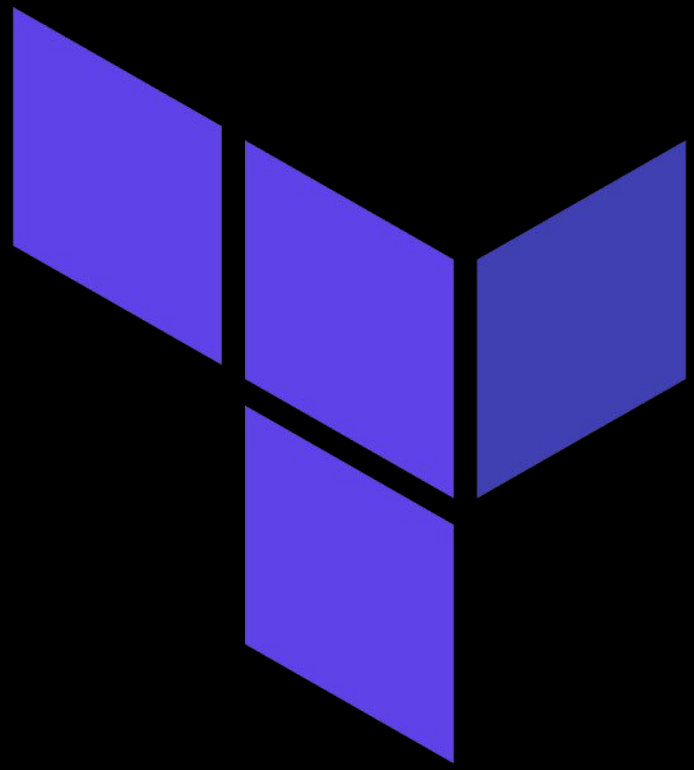


Modern Security toolchains

What do we use nowadays to automate/scan for bad software?
What do they miss?

Terraform

- IaC tool to automate deployment of firewalls and network segmentation
- Organizations can define custom policies to match their business requirements
- Handles API keys and other sensitive information securely

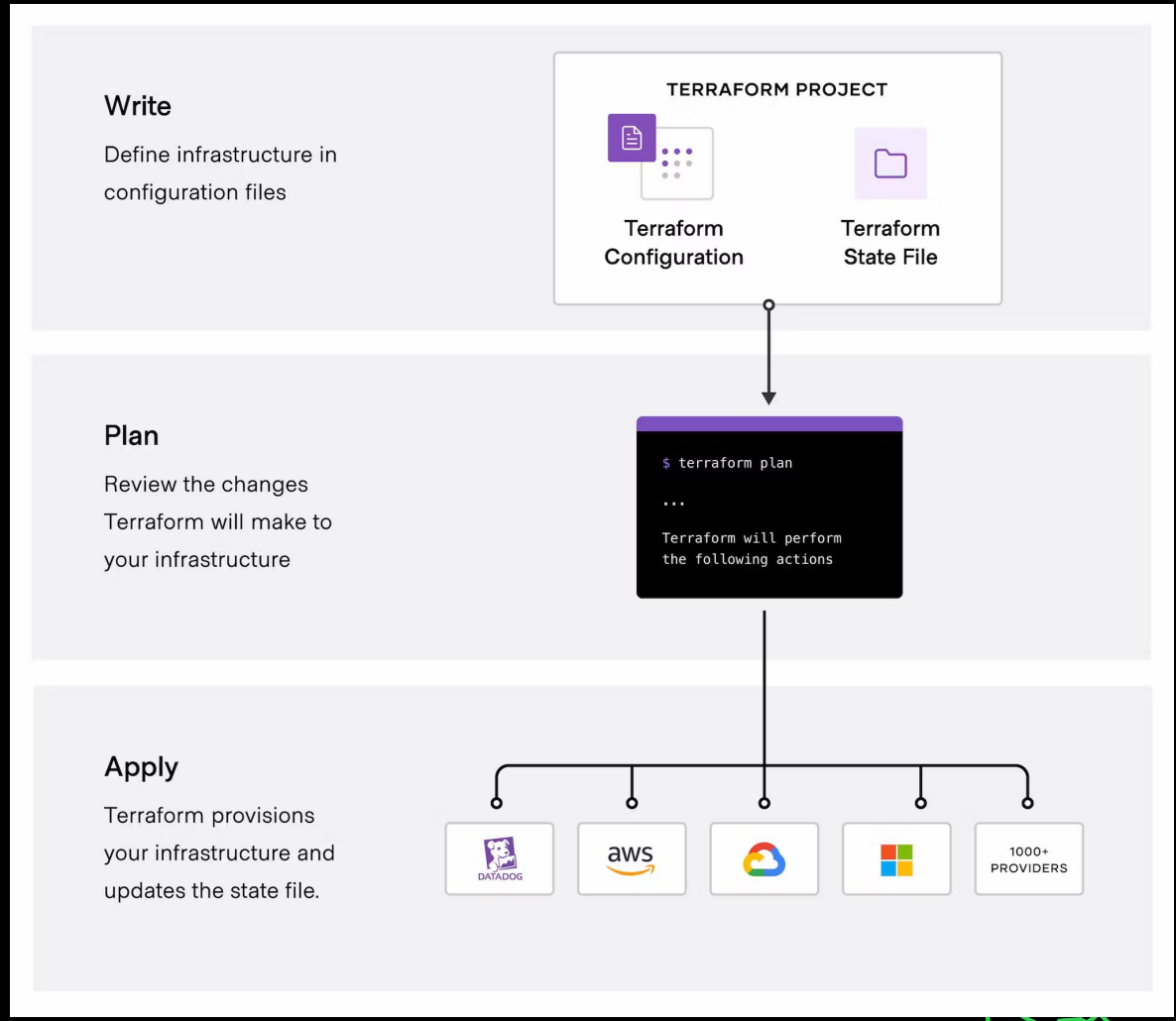


How to use Terraform?

- Terraform files are written in HCL
- terraform init -> plan -> apply on example.hcl to create and manage state file

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```



State Configuration

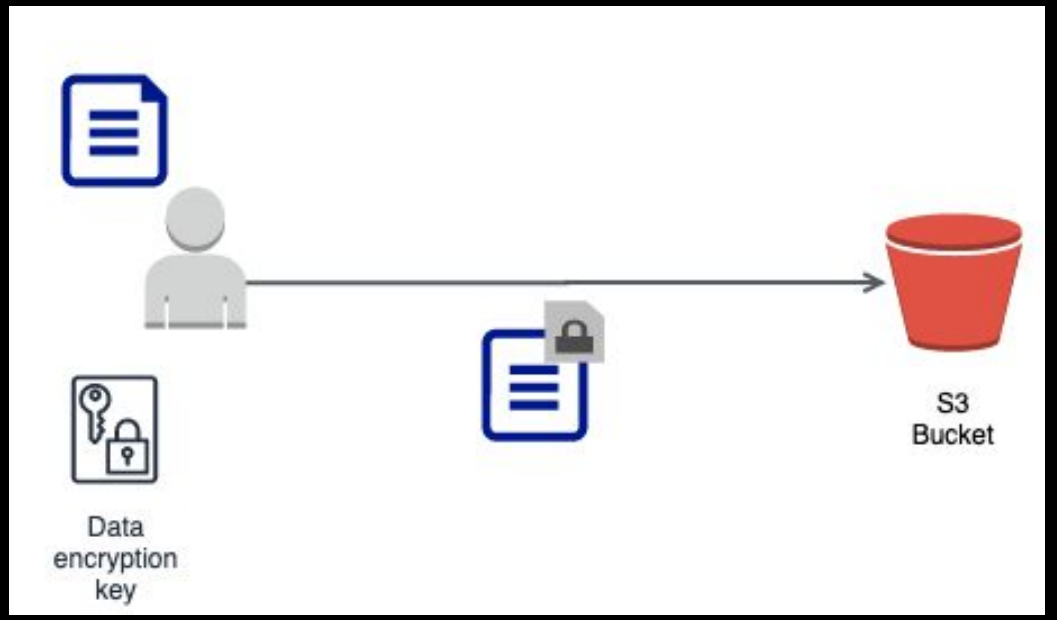
```
{
  "version": 4,
  "terraform_version": "0.12.29",
  "serial": 1,
  "lineage": "b7fa1c5f-3acc-42f6-8b2a-6a0f2a14b8d9",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider.aws",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "id": "i-1234567890abcdef0",
            "ami": "ami-0c55b159cbfafe1f0",
            "instance_type": "t2.micro",
            "tags": {
              "Name": "MyInstance"
            },
            // ... other attributes ...
          }
        }
      ]
    }
  ]
}
```

Terraform's Threat Model

- Some common threats in the Terraform threat model:
  - Unauthorized access to resources or sensitive information → this can stem from something like a faulty IAM role policy (can allow users to assume roles upon request may lead to privilege escalation)
  - Infrastructure tampering, leading to unapproved changes or unauthorized deployments → Not properly authorizing artifacts for deployment can lead to something serious… more on that on the next section
  - Mitigating such threats requires us to follow a strict set of protocols

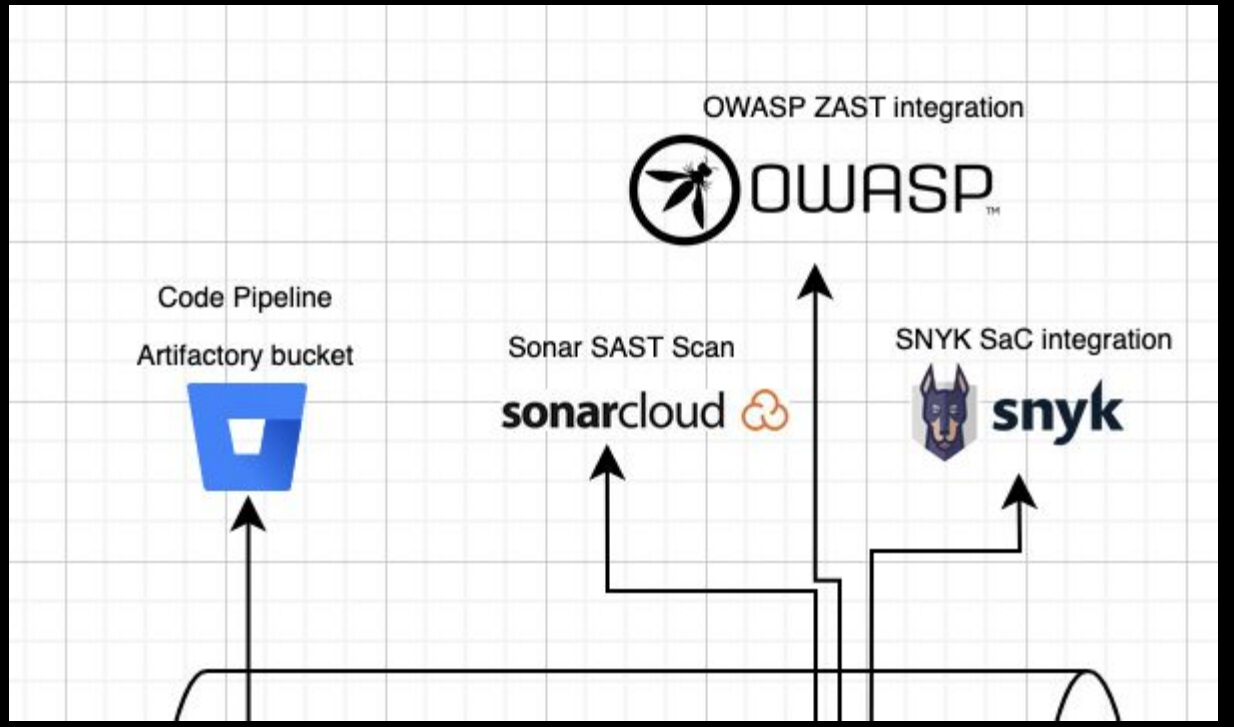
Security Considerations

- Enable server-side encryption for state config
- Enable state locking to disable concurrent modifications on the state file
- Ensure permissions are properly disbursed with IAM tools



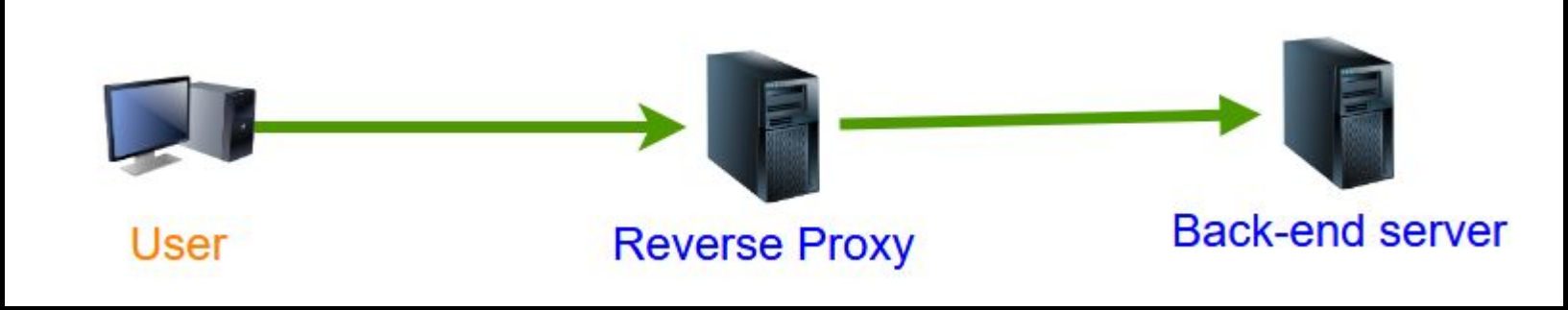
Code scanners

- Infrastructure that has been serviced on the cloud must always be subject to a continuous stream of checks to ensure that they are safe from any potential backdoors
- We can use SAST scans in deployment pipelines (e.g. SNYK) to continuously check software before integration into a workflow



Cloudflare

- Provides a CDN service on a large distributed network → used in a lot of web applications for fast and secure performance
- Cloudflare improves web app security by serving as a reverse proxy for your web app's traffic
  - This would be a server sitting in front of web servers and forwards client requests over to those web servers



Uncovering webservers locked behind cloudflare

- There are tools out there that can help you get the historical DNS records of a given domain (e.g. SecurityTrails) or checking historical SSL certificates that point to the origin IP address from a reverse proxy
  - From last week: If you find an **SSRF inside the web application** you can abuse it to obtain the IP address of the original server and get the files
- If you have a set of potential IPs where the web page, you can use the tool hakoriginfinder

- More exploits can be found on hacktricks

Hakoriginfinder:

```
# You can check if the tool is working with
prips 1.0.0.0/30 | hakoriginfinder -h one.one.one.one


# If you know the company is using AWS you could use the previous tool to search the
## web page inside the EC2 IPs
DOMAIN=something.com
WIDE_REGION=us
for ir in `curl https://ip-ranges.amazonaws.com/ip-ranges.json | jq -r '.prefixes[] | select(.service=="EC2") | select(.region|test("^us")) | .ip_prefix'`; do
    echo "Checking $ir"
    prips $ir | hakoriginfinder -h "$DOMAIN"
done
```

CI/CD pipeline security

How we secure pipelines in modern DevSecOps?

CI/CD Overview

- After development or completion of a task, normally we would like to be able to immediately integrate it and see the updates in live time

- CI (continuous integration) – workflow automation that allows us to automatically test code and verify correctness and robustness before merging it with whatever is existing

- CD (continuous deployment) – automatically push the completed products to the right parties

What could go wrong?

- Suppose that our code passed all the checks during CI that allows it to pass as a good build
  - Now the workflow will place everything into a neat little package and export it as an artifact
- What happens if I want to transmit this artifact across more than one workflow?

Artifact Poisoning

- Alteration or modification of software artifacts and packages by a malicious actor
- Any time you have an artifact that has been transmitted across multiple different workflows, make sure it is sanitized

Artifact Poisoning

```
...
    - name: Download artifact
      uses: dawidd6/action-download-artifact@v2
      with:
          workflow: main.yml
          name: ${{ matrix.libgccjit_version.gcc }}
          path: gcc-build
          repo: antoyo/gcc
          search_artifacts: true # Because, instead, the action only check the last job ran and that won't work since we want multiple artifacts.
    - name: Setup path to libgccjit
      run: |
          echo $(readlink -f gcc-build) > gcc_path
          # NOTE: the filename is still libgccjit.so even when the artifact name is different.
          ln gcc-build/libgccjit.so gcc-build/libgccjit.so.0
    - name: Set env
      run: |
        echo "LIBRARY_PATH=$(cat gcc_path)" >> $GITHUB_ENV
        echo "LD_LIBRARY_PATH=$(cat gcc_path)" >> $GITHUB_ENV
        echo "workspace="$GITHUB_WORKSPACE >> $GITHUB_ENV
    - name: Set RUST_COMPILER_RT_ROOT
      run: echo "RUST_COMPILER_RT_ROOT="${{ env.workspace }}/llvm/compiler-rt >> $GITHUB_ENV
...
```

Artifact Poisoning

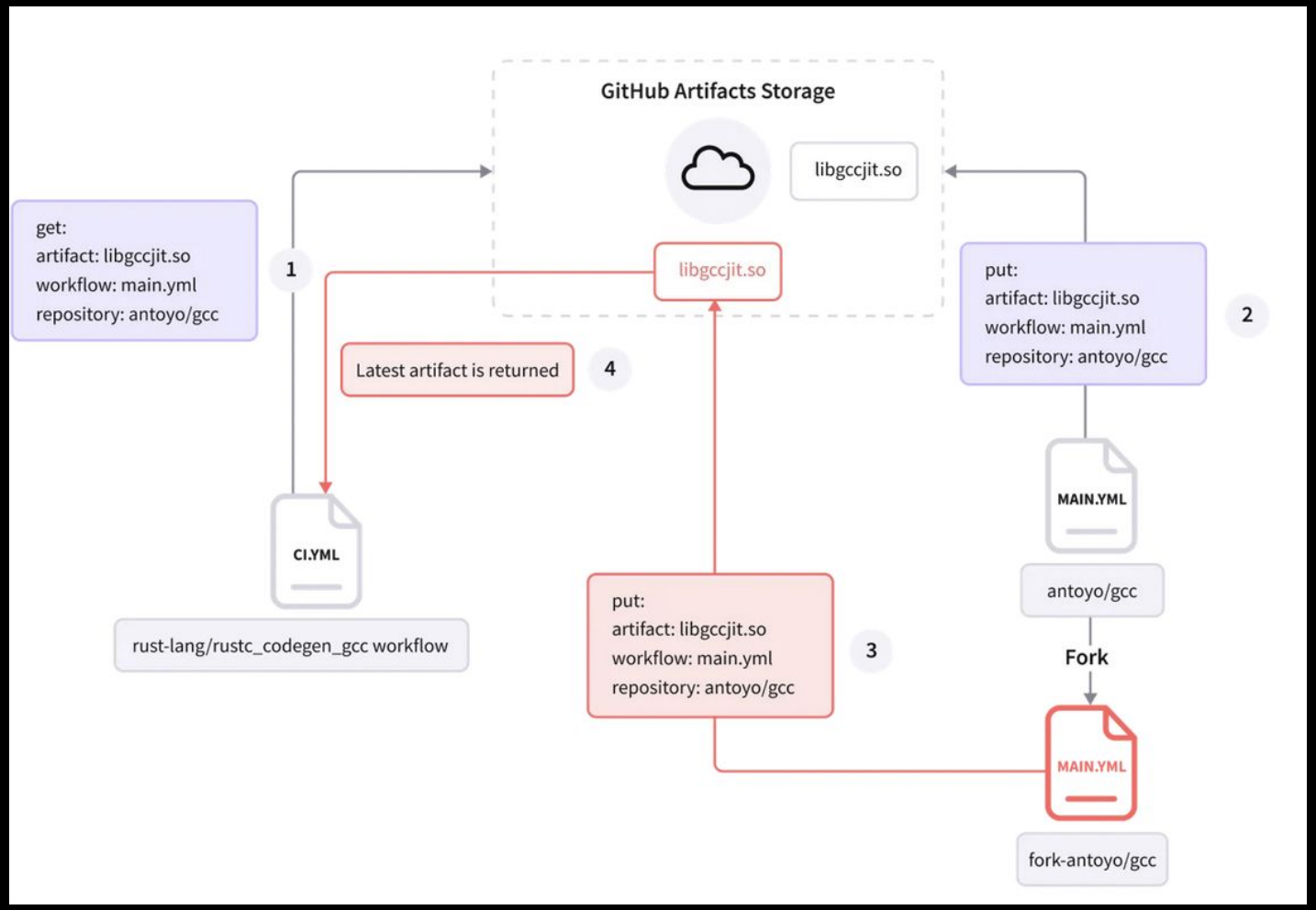
```
...
    - name: Download artifact
      uses: dawidd6/action-download-artifact@v2
      with:
          workflow: main.yml
          name: ${{ matrix.libgccjit_version.gcc }}
          path: gcc-build
          repo: antoyo/gcc
          search_artifacts: true # Because, instead, the action only check the last job ran and that won't work since we want multiple artifacts.
    - name: Setup path to libgccjit
      run: |
          echo $(readlink -f gcc-build) > gcc_path
          # NOTE: the filename is still libgccjit.so even when the artifact name is different.
          ln gcc-build/libgccjit.so gcc-build/libgccjit.so.0
    - name: Set env
      run: |
        echo "LIBRARY_PATH=$(cat gcc_path)" >> $GITHUB_ENV
        echo "LD_LIBRARY_PATH=$(cat gcc_path)" >> $GITHUB_ENV
        echo "workspace="$GITHUB_WORKSPACE >> $GITHUB_ENV
    - name: Set RUST_COMPILER_RT_ROOT
      run: echo "RUST_COMPILER_RT_ROOT="${{ env.workspace }}/llvm/compiler-rt >> $GITHUB_ENV
...
```

Actions Download libgccjit.so file from the antoyo/gcc repo

This doesn't distinguish the master repo from forked versions. 💀

Set appropriate env paths for the library for the user's system

Rust Artifact Poisoning Workflow



Mitigations for artifact poisoning

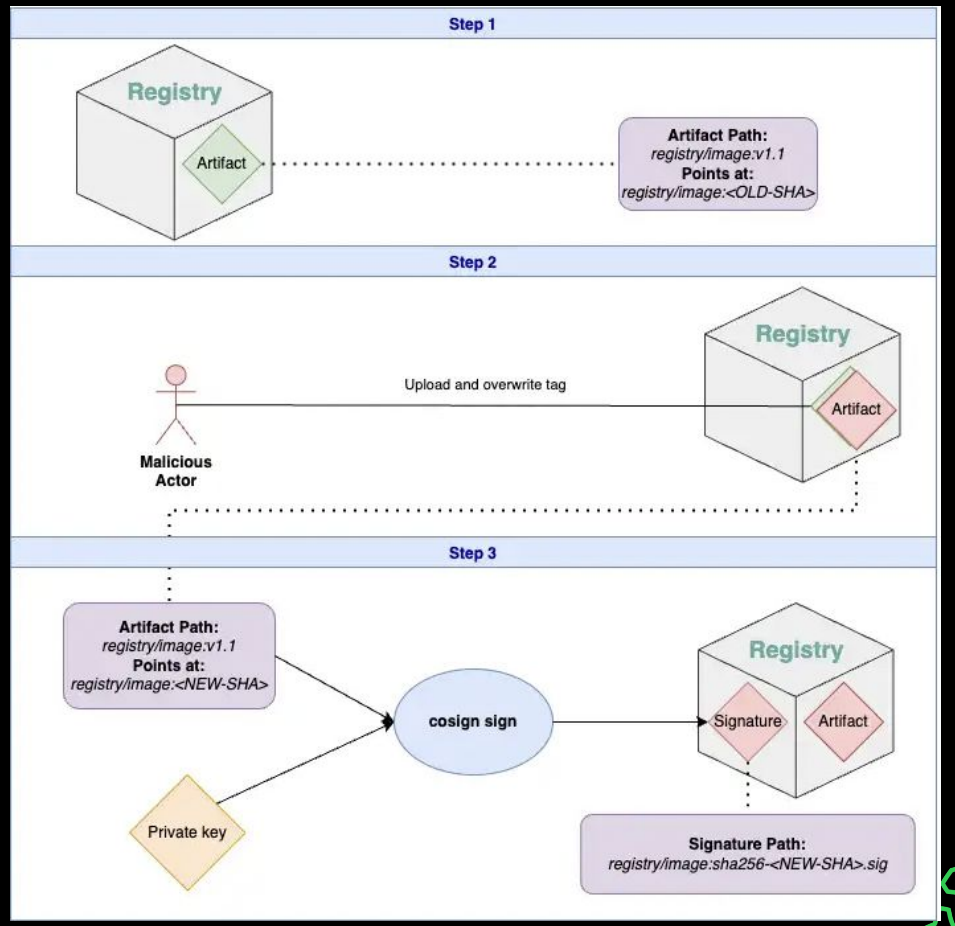
- Cosign – a toolchain and standard for signing, verifying and assuring software integrity through key pairs
  - Given a path to the artifact registry and a private key, cosign generates and uploads a signature to the registry. The path the looks like this:
    `registry/image:sha256-4fb53f12d2ec18199f16d7c305a12c54cda68cc622484bfc3b7346a44d5024ac.sig`
- By signing artifacts before uploading them to the registry, you guarantee that the artifacts were not tampered with after they have been uploaded…right?

It’s not joever.

- Along with cosigning, you can also tag artifacts for release mapped to a specific version
- This could be dangerous!
  - So, you have your package ready for uploading. You add the `v1.1` tag, upload it to the registry, then invoke cosign on `image:v1.1` to sign it. OOPS!
- A malicious actor could still add a malicious artifact to the registry and make you sign it

How to do that?

- An artifact with `v1.1` stored in registry
- A malicious actor with access to the registry uploads a fake artifact, namely to the path `image:v1.1`
- When the CI/CD uses `cosign sign` to verify the artifacts with a privkey, it reads the artifact from the path, but the path points to a bad artifact!



A fix: Use digests

- Digests work because they map directly to the image
- Verifying the image from before will fail because the digest generated from the signature will not match the digest from the fake artifact

```
# get the digest of the image before pushing it
docker tag image:latest real-registry/image:v1.1

digest=$(docker push real-registry/image:v1.1 | cut -d ' ' -f3)
# OUTPUT: v1.1: digest: sha256:4fb53f12d2ec18199f16d7c305a12c54cda68cc622484bfc3b7346a44d5024ac
size: 528
# using the cut command to get only the digest

# sign the image
cosign sign --key private.key "real-registry/image@${digest}"
```

Resources for cloudsec testing

- Hacktricks cloud – a collection of overviews between different exploits, including the ones we talked about. There are also a couple of ones not cover, such as IAM role escalation

- Blog post 1 and Blog post 2 from LegitSecurity were used for the artifact poisoning part of this slide. LegitSecurity covers a multitude of many other cloudsec vulns from the past

- Exploit DB

Next Meetings

Good luck on finals and happy winter break!

ctf.sigpwny.com

`sigpwny{4_cl0udy_d4y_or_4_z3ro_d4y}`

Meeting content can be found at sigpwny.com/meetings.

SIGPwny