
April 12th, 2019

# Static Code Analysis of Infrastructure

# Who am I?

- Mike Siegel - Wayfair - Red Team Lead (GPEN/GCIH/OSCP/CISA/CISSP)
  - Oscillate between Red and Blue teams every few years.
  - Have performed most security roles at one point or another:
    - Firewalls/Endpoint/AppSec/Pen Testing
    - Compliance/Architecture/Incident Response/Forensics
  - Joined Wayfair in June 2016, during the transition from ‘large startup’ to mature company.

@ml_siegel on Twitter

# A Brief Introduction to Infrastructure As Code

# Infrastructure as Code (IaC)

- Re-usable, re-deployable templates for instructure.
- Rise of IaC coincides with Cloud Computing and DevOps workflows.
- Allows for economy of scale when deploying infrastructure.
- Our infrastructure now lives in version control, and has a commit log.
- The code in question may be JSON, YAML, or domain specific languages such as HashiCorp Configuration Language (HCL)





# Infrastructure as Code (IaC) Continued ....

A Firewall rule Definition in Terraform

```
provider "google" {
  region      = "us-east1"
  project     = "my-project"
}

resource "google_compute_firewall" "default" {
  name    = "example_fw_rule"
  network = "default"

  allow {
    protocol = "tcp"
    ports    = ["443", "22"]
  }

  source_ranges = ["0.0.0.0/0"]
  target_tags   = ["test-node"]
}
```

- Define the Google Cloud Platform (GCP) Project
- Create a sample firewall rule.
- Open up ports 443 and 22 to the Internet for instances tagged ‘test-node

# Infrastructure as Code (IaC) Continued ....

## Setting File Permissions in Puppet

```
file { '/etc/squid/squid.conf':
  ensure  => present,
  owner   => 'root',
  group   => 'root',
  mode    => '0644',
  require => Package['squid'],
  content => template("squid/${squid::config}.erb"),
}

file {'/tmp/squid':
  ensure => 'directory',
  owner  => 'squid',
}

service { 'squid':
  ensure    => running,
  enable    => true,
  subscribe => File['/etc/squid/squid.conf'],
}
```

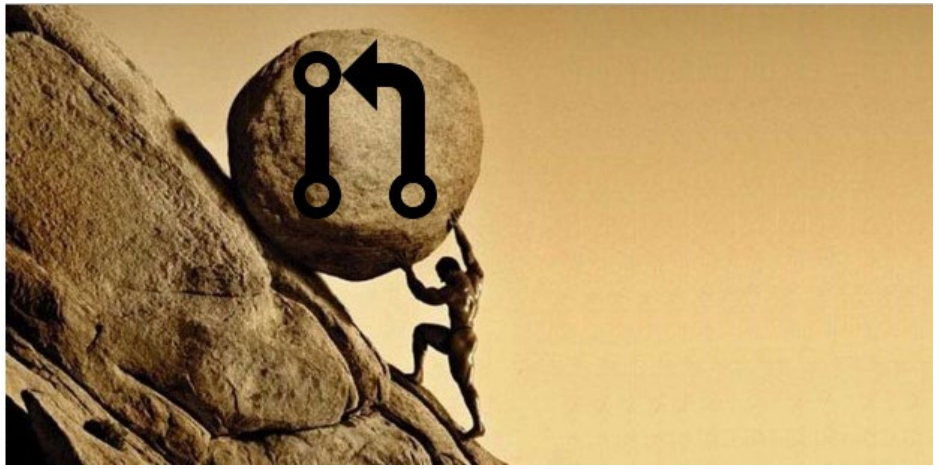
- Ensure the existence of our Squid configuration file
- Pull the base configuration from a template
- In this case, we're setting the mode to 0644
  - User: read-write
  - Group: read
  - Other (world): read

# IaC Benefits and Challenges to Security Organizations

- **Benefits for Security Organizations**
  - **Code review** for Infrastructure
  - **Auditable history** in version control
  - **Quickly rebuild** infrastructure
  - **Ensure consistency** and repeatability of security controls.
  - **Fewer ‘one offs’** and ‘pet’ servers (hopefully).

- **Challenges for Security Organizations**
  - Existing Security tooling isn’t optimized for this: Typically it **validates AFTER deploy**
  - **Skill gap**: Security Engineering now needs to know puppet and terraform
  - **Resource constraints**: Reviewing code is time consuming! How many Infrastructure Security Engineers do you have per Site Reliability Infrastructure Engineers?
  - **Human Error** can now be magnified at scale!

# Wayfair Specific Challenges



- Limited Security Engineering staff: In 2017, **we had a ratio of one InfraSec Engineer to sixty Site Reliability Engineers.**
- **Peer reviewing** every change that might impact security **was growing impossible**.
- We wanted to move to Google Cloud rapidly, but **most security tooling didn't support GCP**!
- Previously, things like firewall rules, load balancer configurations, were in the hands of very few. **Now, a wider audience** could potentially contribute to the code base!
- Our legacy model of requesting firewall changes, **tickets with source and destination addresses**, wasn't suited to this new world.

## So what do we do now?

# A Brief Introduction to Static Code Analysis

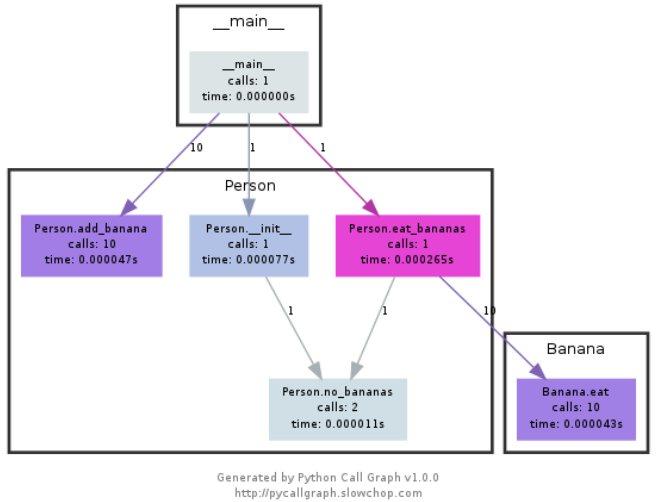
# Static Code Analysis

- Dating at least to the mid-eighties, a series of techniques used to analyze software for defects, including security defects.

- Analyzes the source code of applications using various methods:
  - String/pattern matching
  - Call graph analysis
  - Model checking using finite state machines.
  - Abstract syntax tree analysis.

- There are a huge number of available tools, both commercial and open source.

- While the intricacies of Static Analysis are the subject of numerous PhD thesis' and ACM publications, we can break this down into a simpler idea:
  - **Can we find security anti-patterns in code**?
  - **Yes, yes we can**.




```
sets.append(utils.build_conf_dict(
    'eval', 'B307', ['eval'],
    'Use of possibly insecure function - consider using safer '
    'ast.literal_eval.'
    ))
```

# Static Code Analysis Example: bandit

```
import MySQLdb

db = MySQLdb.connect(host="localhost",
user="martinbishop",
passwd="Hunter1",
db="setec")

cursor = db.cursor()

topping = raw_input("Select a topping: ")

cursor.execute("SELECT price FROM toppings WHERE name = '%s'" % topping)
for price in cursor.fetchall():
   print (price)

db.close()
```

Bandit is an open source static code analysis tool for Python. In this example, we write some terrible code using hardcoded database credentials and a vulnerable query string.

# Static Code Analysis Example: bandit

Bandit warns us of potential issues in our code.

This type of validation can be built into an IDE, or a
Continuous Integration/Deployment pipeline.

```
--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector
through
string-based query construction.
   Severity: Medium   Confidence: Medium
   Location: badcode.py:12
11
12      cursor.execute("SELECT price FROM toppings WHERE name = '%s'" %
toppping
)
13      for row in cursor.fetchall():

--------------------------------------------------
```

# Static Code Analysis of Infrastructure

# Static Code Analysis of Infrastructure

- Wayfair Security Engineering wanted to ensure a consistent approach to security configurations in GCP, reducing manual effort. A search began for solutions.
- At the time, in early 2017, there was no solution (as of 2019, Sentinel and tflint are other options)
- So we built our own, called ‘terrafirma’.
- Terrafirma takes the result of terraform plans (essentially a state file) and analyzes them for security defects. These defects are classified by simple YAML rule files.
- This was then baked into the deployment process using Jenkins: a ‘FATAL’ level error would fail the build and fail to deploy.

# Static Code Analysis of Infrastructure: terrafirma

We define some simple rules in YAML for firewalls

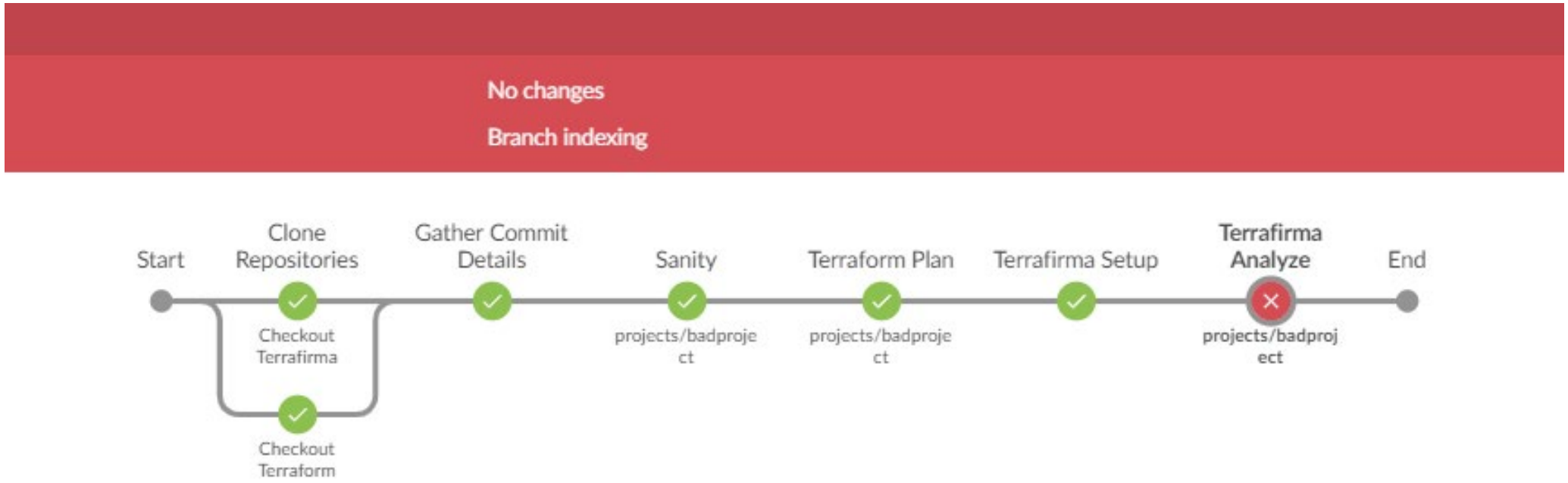
```yaml
google_compute_firewall:
  - issue: FW_1
    name: Source range open to Internet
    fields:
      source_ranges:
        - 0.0.0.0/0
    level: WARN

  - issue: FW_2
    name: SSH Open
    fields:
      allow.ports:
       - "22"
      allow.protocol:
        - tcp
    level: INFO

  - issue: FW_3
    checktype: dynamic
    name: Overly permissive rule, slash 8
    function: cidr_comparison
    test: GE
    fields:
      source_ranges:
        - "8"
    level: WARN
```

Based on our ‘ANY’ SSH rule from earlier, terrafirma warns us of multiple issues, for example the source range being larger than a /8 network.

```
---
ISSUE FW_1
- Source range open to Internet
- SEVERITY WARN
- RESOURCE example_fw_rule.google_compute_firewall
---
ISSUE FW_2
- SSH Open
- SEVERITY INFO
- RESOURCE example_fw_rule.google_compute_firewall
---
ISSUE FW_3
- Overly permissive rule, slash 8
- SEVERITY WARN
- RESOURCE example_fw_rule.google_compute_firewall
---
ISSUE MI_1
- SSH open to the Internet
- SEVERITY FATAL
- RESOURCE example_fw_rule.meta_policy
---
```

# Static Code Analysis of Infrastructure: Pipeline integration

Terraform plans which violate our YAML rules will cause the deploy process to fail. If this happens, the engineer will either have to correct their mistakes, or work with security to submit a merge request to adjust the policy.



# Static Code Analysis of Infrastructure: Pipeline integration

Terrafirma Analyze / projects/badproject - <1s

✓ › Running Security Analysis of the plan with Terrafirma — Print Message

✓ › plan projects-badproject — Restore files previously stashed

✗ ⌄ set -x source terrafirma_venv/bin/activate /bin/tfjson projects/badproject/terraform_plan.out | python terrafirma_venv/bin/terrafirma — Shell Script

```
[terrafirma] Running shell script
    ++ REDACTED
    + /bin/tfjson projects/badproject/terraform_plan.out
    + python terrafirma_venv/bin/terrafirma
    ISSUE: SB_2
    - DESCRIPTION: Bucket writable by all users.
    - SEVERITY: FATAL
    - RESOURCE: google_storage_bucket_acl.bad-store-acl.role_entity.0

    google_storage_bucket_acl.bad-store-acl:
    {
     role_entity.0: WRITER:allAuthenticatedUsers
    }
    ---
    ISSUE: SB_4
    - DESCRIPTION: Bucket writable by all users.
    - SEVERITY: FATAL
    - RESOURCE: google_storage_bucket_acl.bad-store-acl.role_entity.0

    google_storage_bucket_acl.bad-store-acl:
    {
     role_entity.0: WRITER:allAuthenticatedUsers
    }
    ---
```

✓ › Security issues of FATAL level have been found in the plan — Print Message

✗ ⌄ Update the commit status in GitLab

```
    script returned exit code 3
```

# Static Code Analysis of Infrastructure

- terrafirma's rulesets can also be applied to:
  - Google Cloud Storage (similar to S3 in Amazon Web Services)
  - Identity and Access Management permissions
  - Load balancers
  - Validating compute image versions (the equivalent of Amazon Machine Images)
  - Pretty much any terraform resource you can think of.

- Internally, it's using Terraform plans converted to JSON and doing pattern matching/Python function calls.

- Other tools that could accomplish this for terraform: tflint and HashiCorp's Sentinel

- But this concept can be beyond terraform: to nginx, puppet, chef, ansible and other platforms.

# Other examples...

# Static Code Analysis of Infrastructure: gixy for nginx

- Yandex's gixy is a Python tool that inspects nginx configurations for security issues.

- Some of these include
  - SSRF
  - HTTP Splitting
  - Path traversal

For example, the lack of a trailing slash can lead to the nginx 'off by slash' configuration issue

```
location /foo {
    alias /data/bar/;
}
```

```
$ wget /foo../app/config.py
```

Which lets an attacker perform an LFI/traversal attack.

# Static Code Analysis of Infrastructure: gixy for nginx

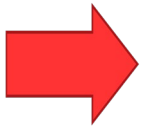
```
==================== Results ===================

>> Problem: [alias_traversal] Path traversal via misconfigured alias.
Description: Using alias in a prefixed location that doesn't ends with
directory separator could lead to path traversal vulnerability.
Additional info:
https://github.com/yandex/gixy/blob/master/docs/en/plugins/aliastraversal.md
Pseudo config:

location /foo {
        alias /dat/bar;
}

==================== Summary ===================
Total issues:
    Unspecified: 0
    Low: 0
    Medium: 1
    High: 0
```

# Static Code Analysis of Infrastructure: foodcritic for Chef

- Foodcritic is a Chef cookbook analyzer
- Not focused on security per se, but easy enough to write rules that detect bad things:
- Given bad file permissions on an important file:

We can detect and fail deploy based on Cucumber definitions of what's acceptable.


```
require "spec_helper"

describe "FC999" do
  context "with a cookbook with a recipe that sets mode" do
    recipe_file <<-EOH
    file '/etc/shadow' do
      mode '0777'
    end
    EOH
  it { is_expected.to violate_rule }
  end
end
```

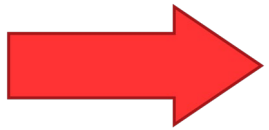


```
FC999: Setting incorrect mode on shadow: badcookbook/badcook:2
```

# Static Code Analysis of Infrastructure: puppet-lint for Puppet

- Puppet is another configuration management software.
- puppet-lint is a program for linting (or validating) puppet manifests.
- Puppet is a great area to focus security efforts, since it often controls file permissions and service setup.
- A github user named floek wrote a series of puppet-lint rules to scan puppet manifests for potentially bad items.

```
 file { '/wayfair/foo/bar':
  ensure => directory,
  mode   => '0777',
}

file { '/var/run/foo':
  ensure  => directory,
}
```



```
puppet/modules/bar/manifests/files.pp - ERROR: File or directory definition
with world permissions detected (security!) on line 120
puppet/sdk/bar/manifests/blah/server.pp - ERROR: Possible password variable
in exec used (security!) on line 288
puppet/servicestest/manifests/init.pp - WARNING: Vhost without ssl detected
(security!) on line 77
puppet/sdk/foo/manifests/server.pp - ERROR: Class or defined_type parameter
in exec used (security!) on line 288
puppet/modules/bar/manifests/files.pp - ERROR: File or directory definition
with world permissions detected (security!) on line 120
puppet/services/foo/dev.pp - ERROR: File or directory definition with world
permissions detected (security!) on line 101
```

# Under the hood

- Many static analysis tools tools rely on ‘Abstract Syntax Tree’ analysis of code, this is the approach foodcritic takes.
- Simpler checks can use regex, string and pattern matching.
- Tools used by programmers for ‘behavior driven development’ (BDD) and unit testing can be harnessed for infrastructure!
- puppet-lint in particular, relies in ‘rspec’ a behavior driven testing tool. You state what you expect, and the results are compared.
- foodcritic rules are written Cucumber, another BDD framework which uses a more ‘friendly’ syntax.

# Benefits of Static Code Analysis of Infrastructure

- Reduce manual review bottlenecks
- Security/Hardening requirements live in source control:
  - Easily audit changes
  - Follow an already defined merge request review process
- Allow DevOps teams/SRE to maintain the ability to rapidly deploy new infrastructure.
- Ensure hardened infrastructure PRIOR to deploy
- Autoscaling infrastructure might not exist long enough to do a CIS benchmark the ‘traditional’ way.

# References and Further Reading

- The Awesome Static Analysis List https://github.com/mre/awesome-static-analysis

Analysis Tools for Infrastructure

- terrafirma https://github.com/wayfair/terrafirma
- tflint https://github.com/wata727/tflint
- puppet-lint-security-plugins https://github.com/floek/puppet-lint-security-plugins
- foodcritic http://www.foodcritic.io/
- gixy https://github.com/yandex/gixy


wayfair®