Building AppSec Automation with python

Abhay Bhargav - CTO, we45

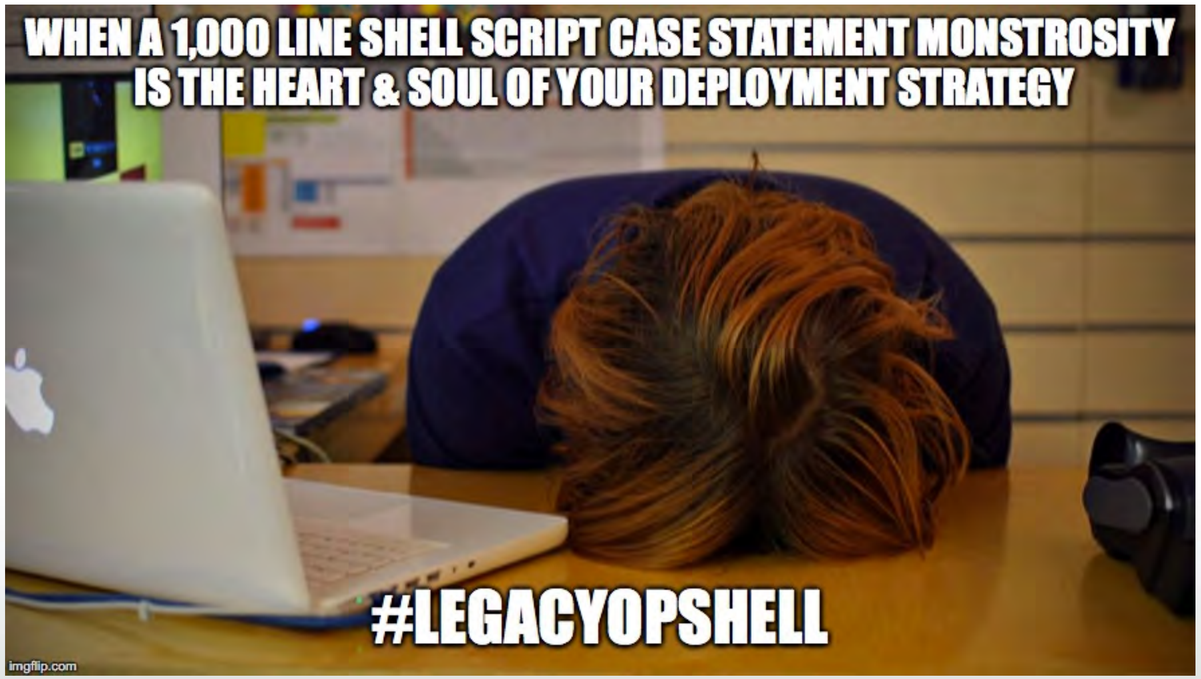
A Gentle Introduction to DevOps

we45

- What is DevOps?
- Where does Security fit in?

What is DevOps?

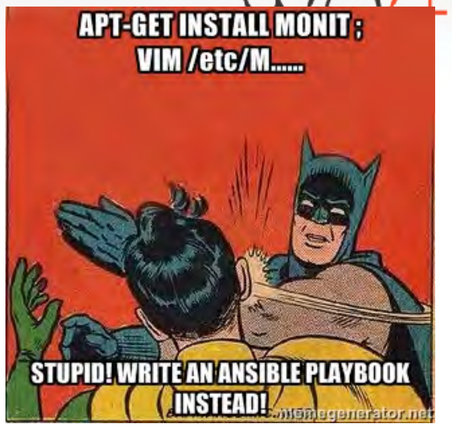
- Key Objective - Harmonize IT Operations by working with Developers and Ops seamlessly
- Rely on processes and automation to achieve higher throughput - Continuous Delivery



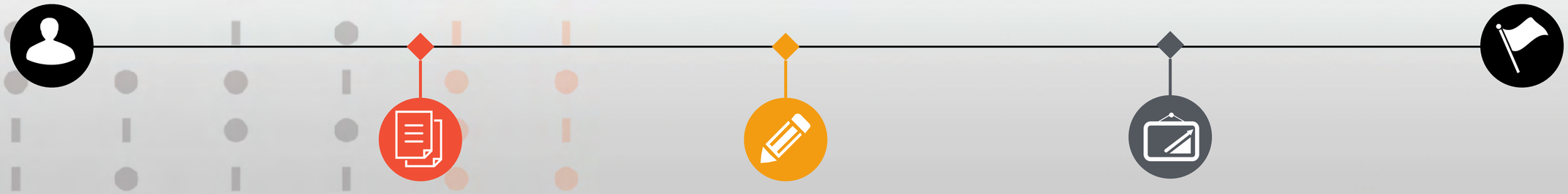
Without DevOps

Requirements — Design — Develop — Test — Deploy

With DevOps (hopefully…)



Requirements Design Develop Test Deploy



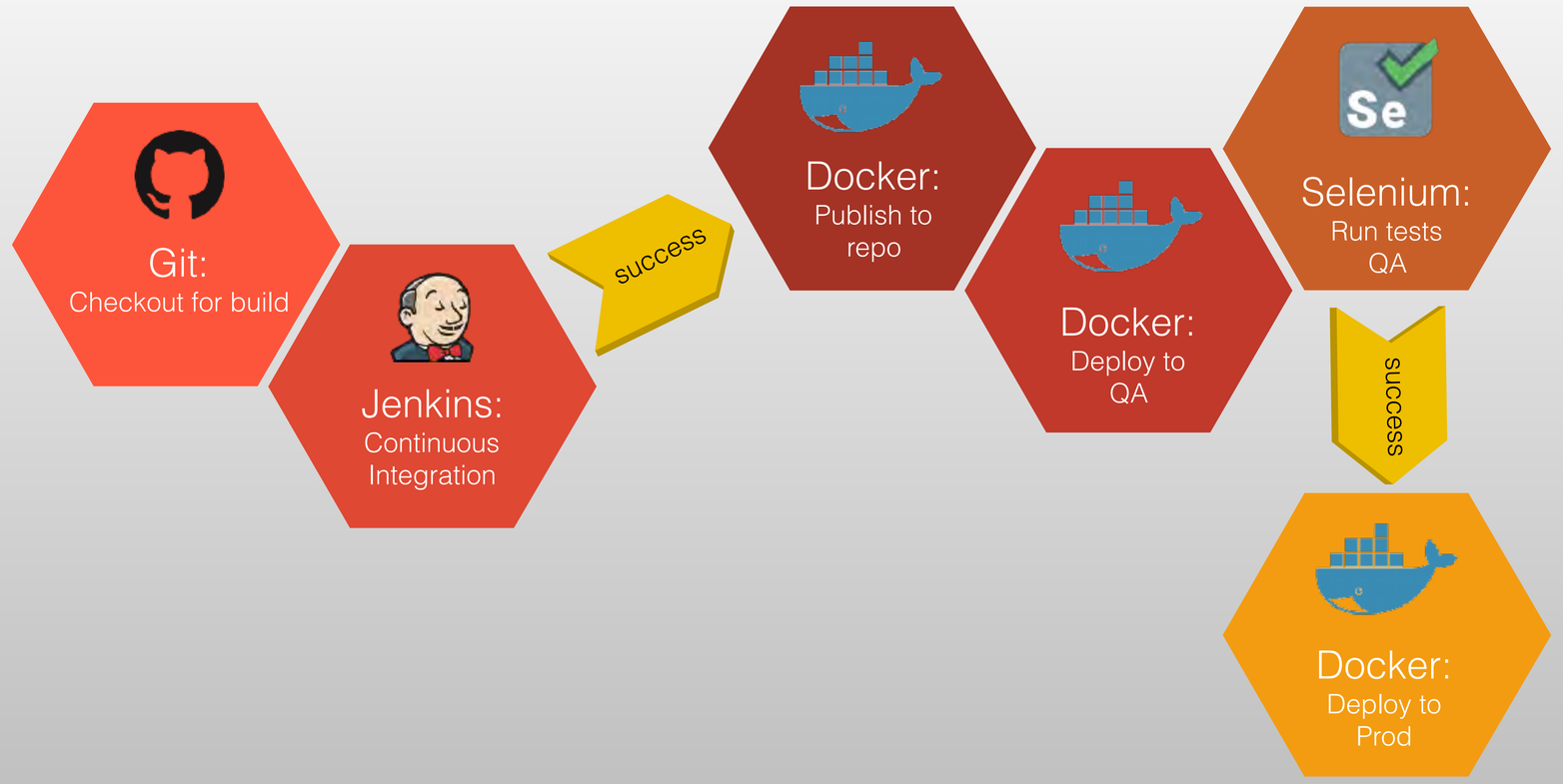
Example pipeline

we45

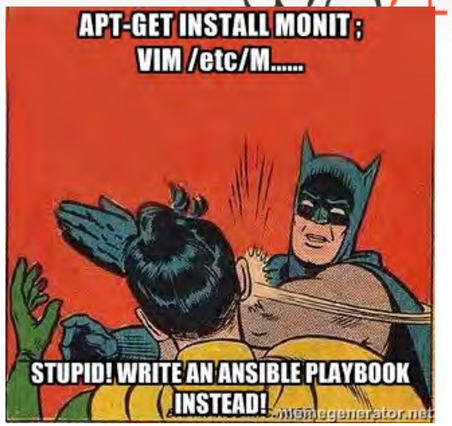
Developer

Coding -
Modify and commit

Orchestration engine

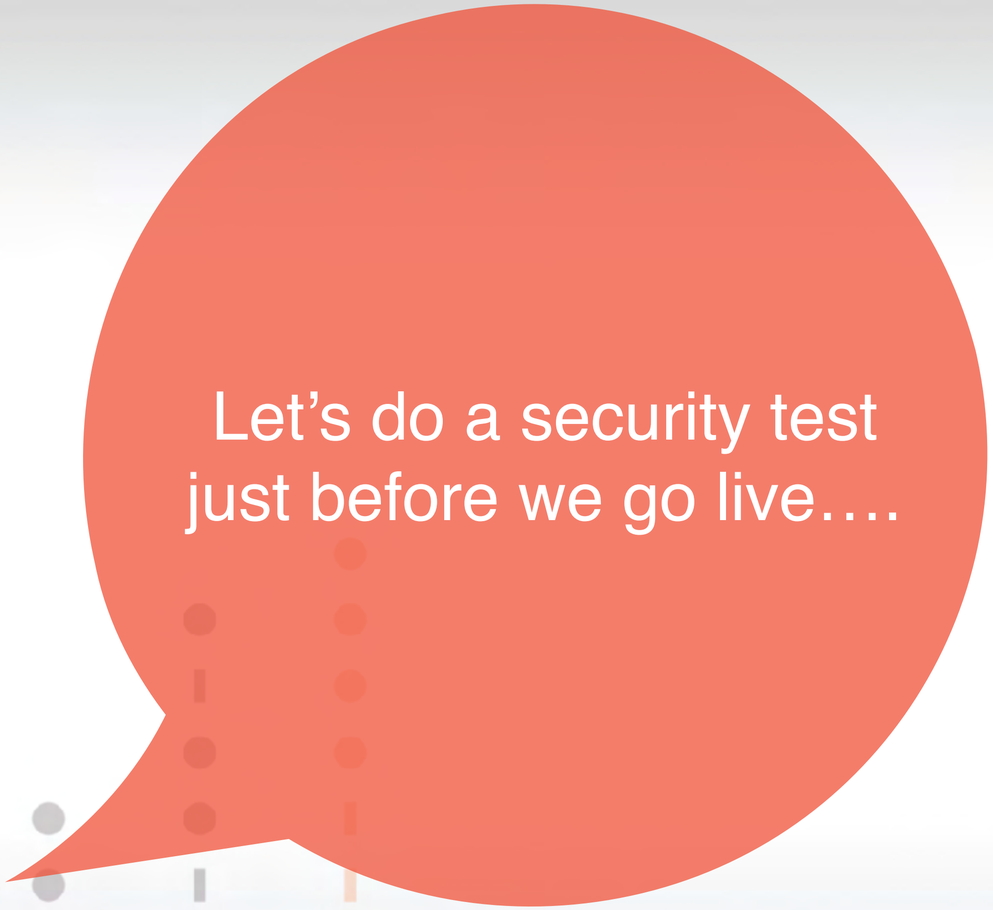


But....



Requirements Design Develop Test Deploy

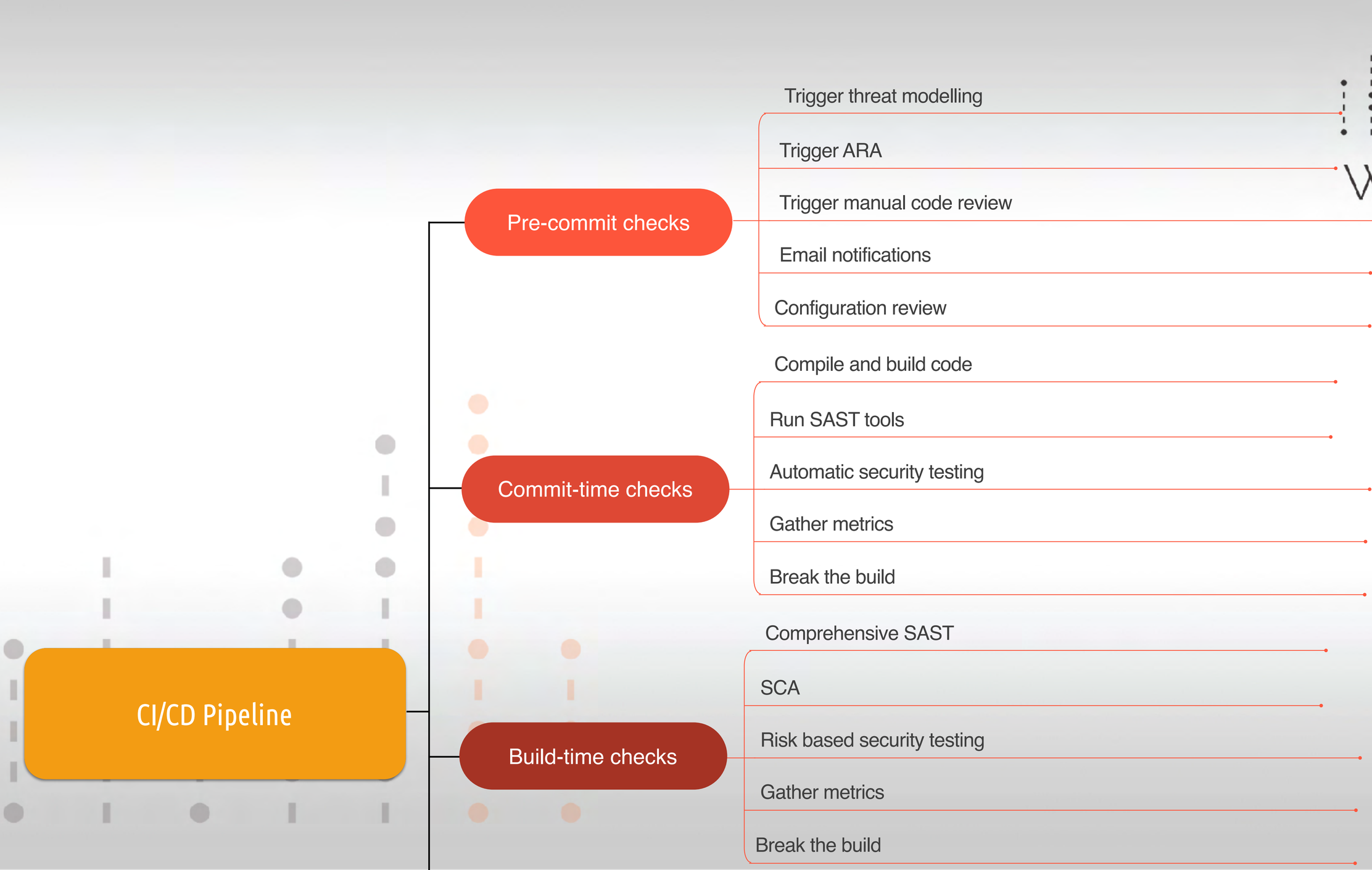
security?

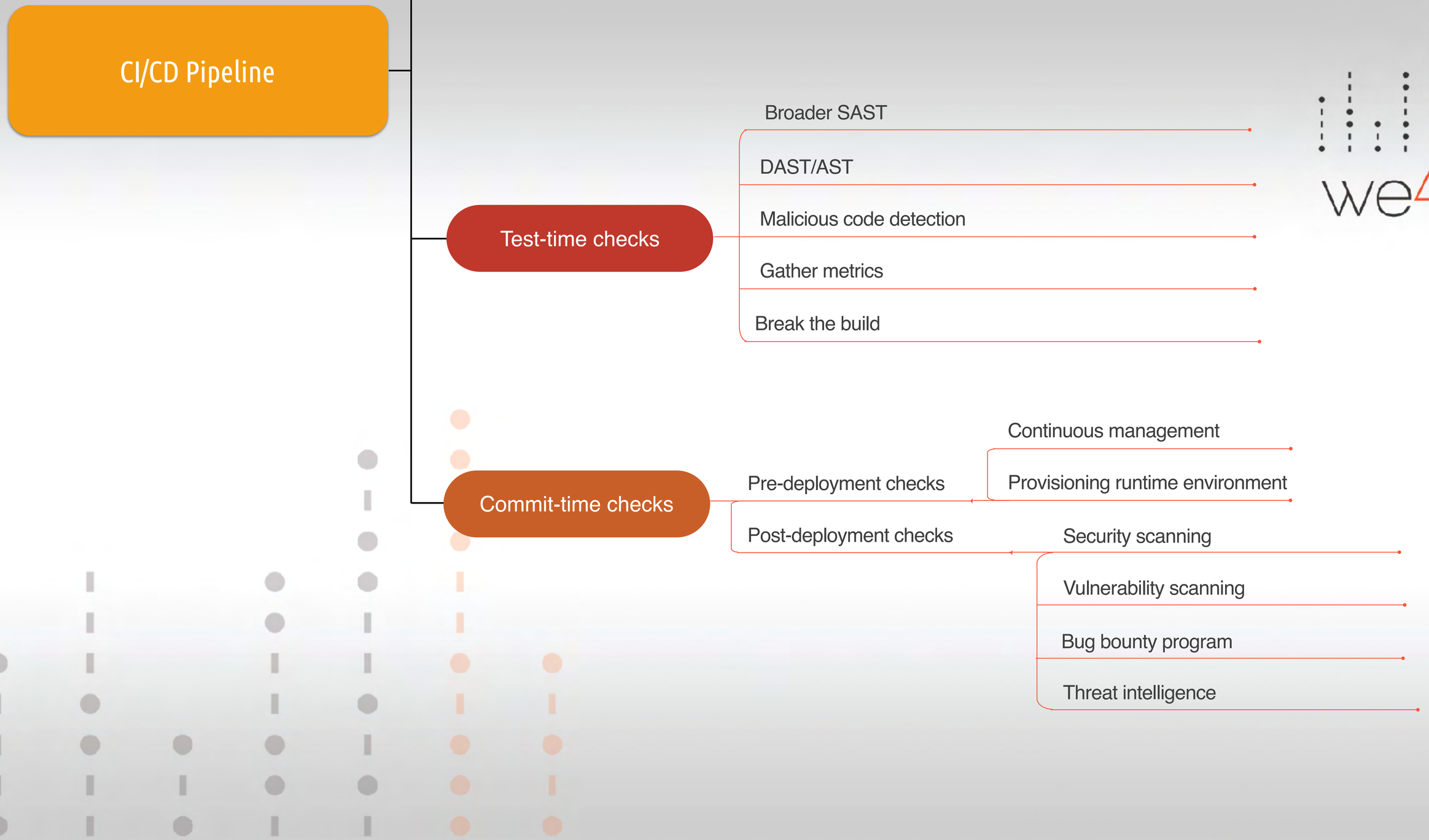


we45

The line that has ruined
Application Security for all of us.

In Short….


CI/CD Pipeline
Pre-commit checks
Trigger threat modelling
Trigger ARA
Trigger manual code review
Email notifications
Configuration review
Commit-time checks
Compile and build code
Run SAST tools
Automatic security testing
Gather metrics
Break the build
Build-time checks
Comprehensive SAST
SCA
Risk based security testing
Gather metrics
Break the build


CI/CD Pipeline
Test-time checks
Broader SAST
DAST/AST
Malicious code detection
Gather metrics
Break the build
Commit-time checks
Pre-deployment checks
Continuous management
Provisioning runtime environment
Post-deployment checks
Security scanning
Vulnerability scanning
Bug bounty program
Threat intelligence
we45

The Need of the Hour….

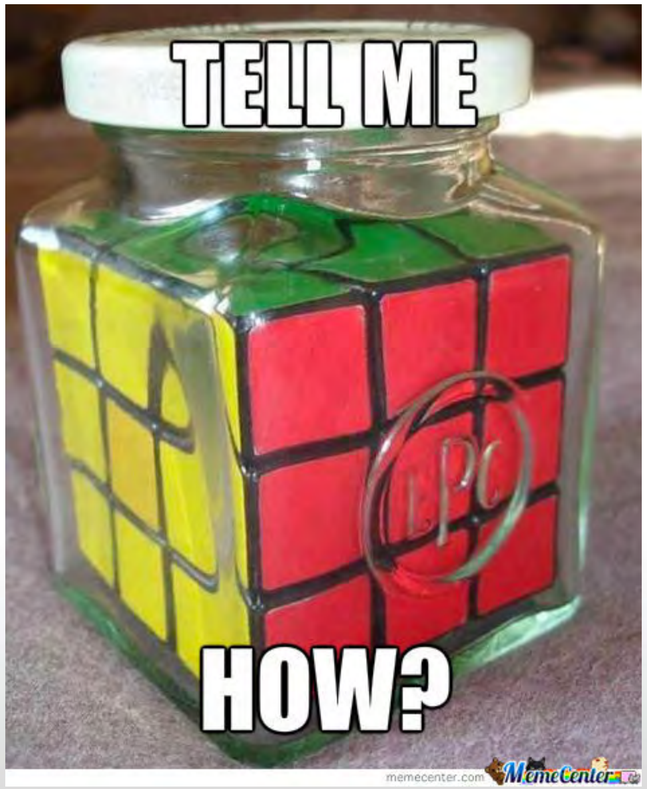
- Continuous Application Security Practices to keep pace with Continuous Delivery
  - Dynamic Application Security Testing in the Pipeline
  - Static Application Security Testing in the Pipeline

Security in DevOps

Threat modeling	SAST Security - Composition	DAST IAST	Security in IaC	Security monitoring & attack detection

The Application Security Engineer's Story

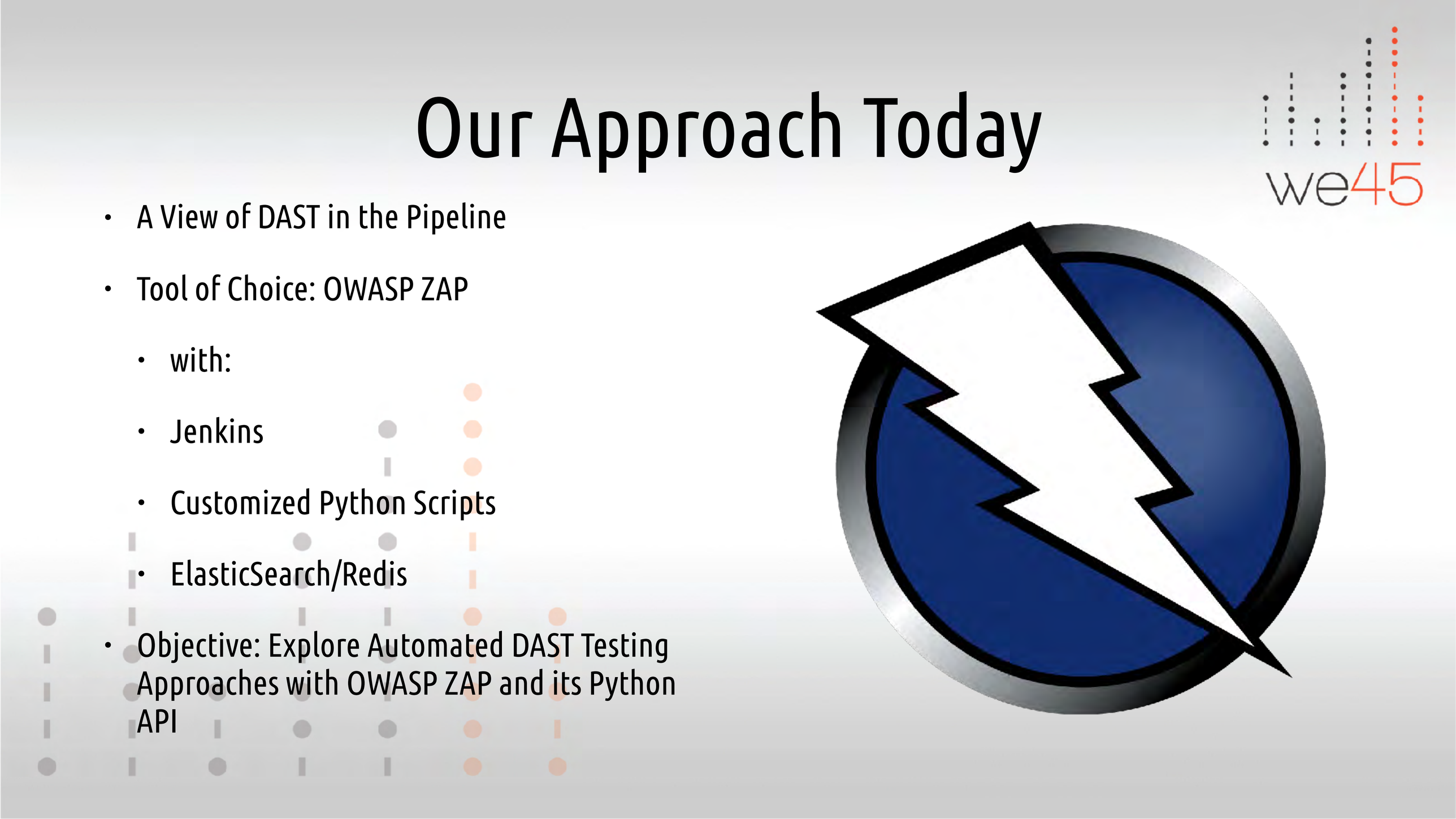
- How?
  - Run DAST in the Pipeline?
  - Correlate Results from DAST
  - Compare Results from scans in time?



The Need of the Hour….

- Continuous Application Security Practices to keep pace with Continuous Delivery
  - Dynamic Application Security Testing in the Pipeline
  - Static Application Security Testing in the Pipeline

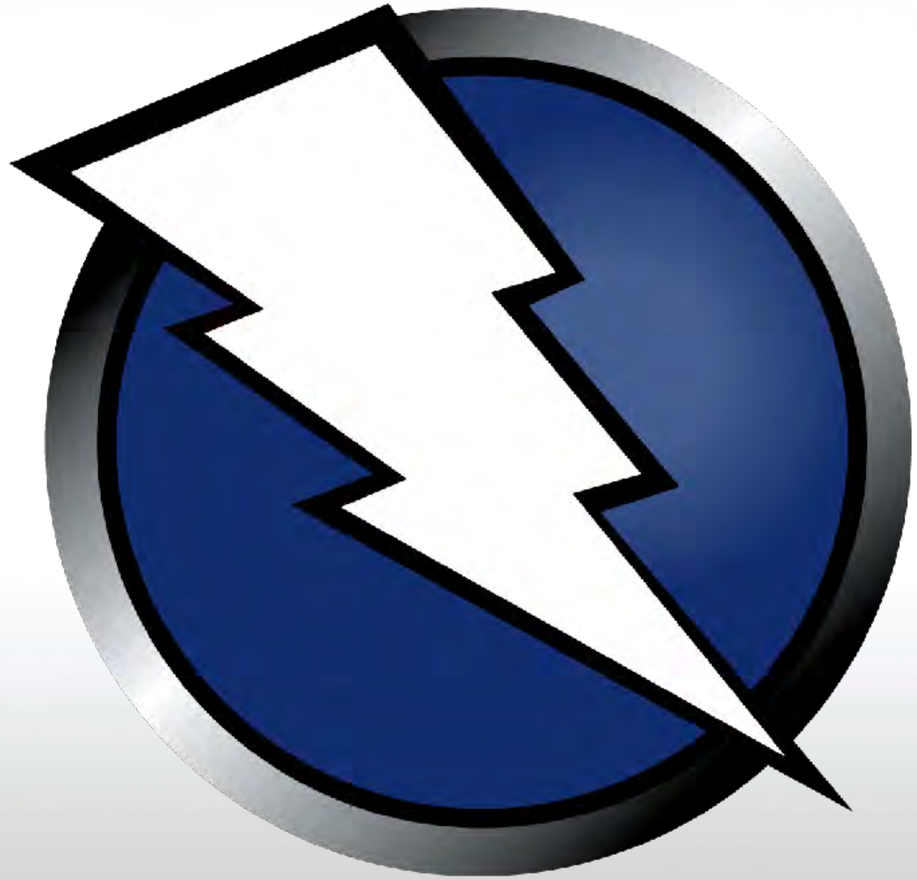
Our Approach Today



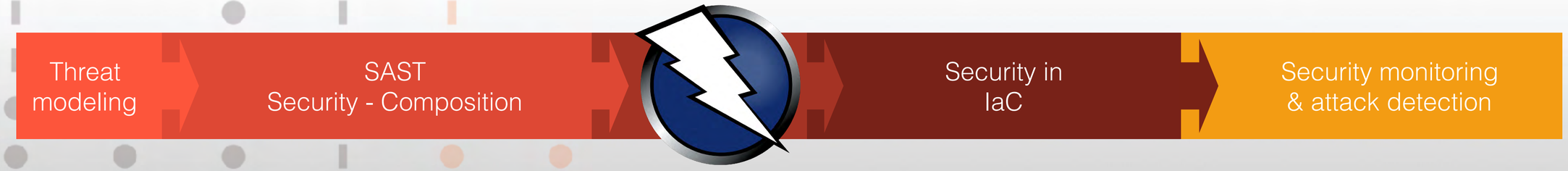
- A View of DAST in the Pipeline
- Tool of Choice: OWASP ZAP
  - with:
  - Jenkins
  - Customized Python Scripts
  - ElasticSearch/Redis
- Objective: Explore Automated DAST Testing Approaches with OWASP ZAP and its Python API

Why OWASP ZAP?

- Free and Open Source Web Application Vulnerability Scanner
- Feature-Rich, well supported, with several contributors
- Community Support - Plugins, Add-ons, etc.
- Documentation - Better than most scanners out there
- Great API and Scriptable Scanner



Security in DevOps



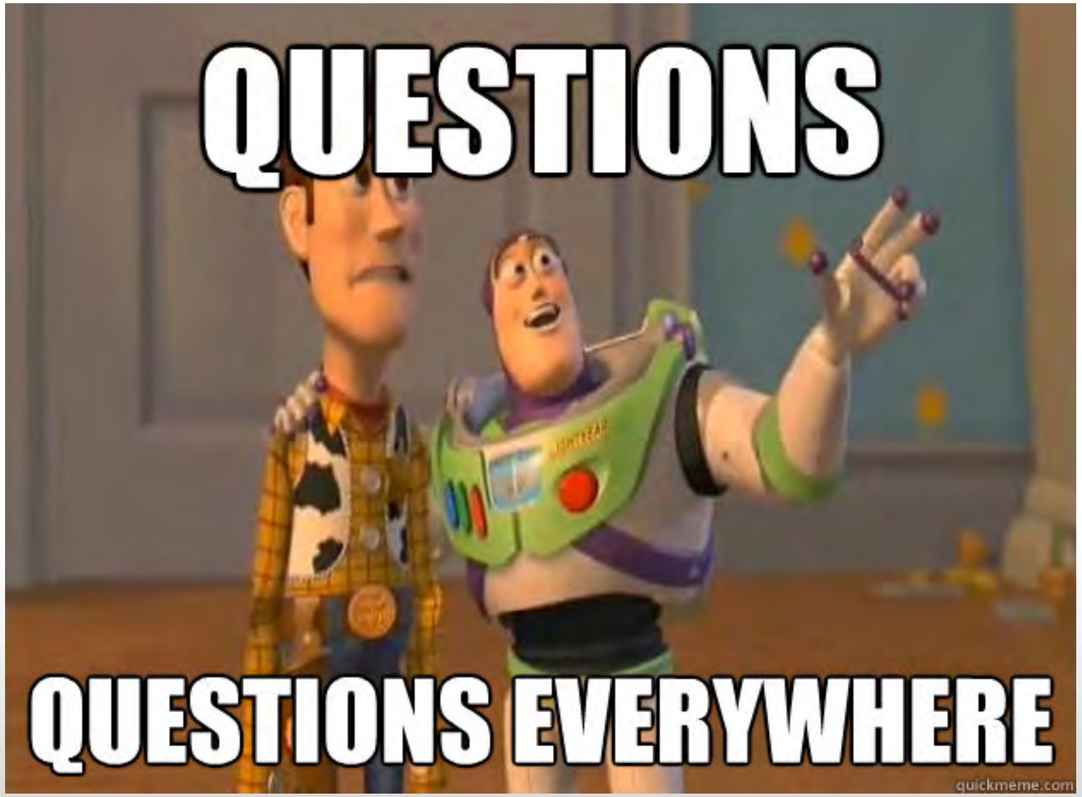
Stories for today….

- The Application Security Engineer/ DevSecOps Engineering Perspective
- The Automation-focused Pentester Perspective

Key Questions - AppSec Engineering/DevSecOps

we45

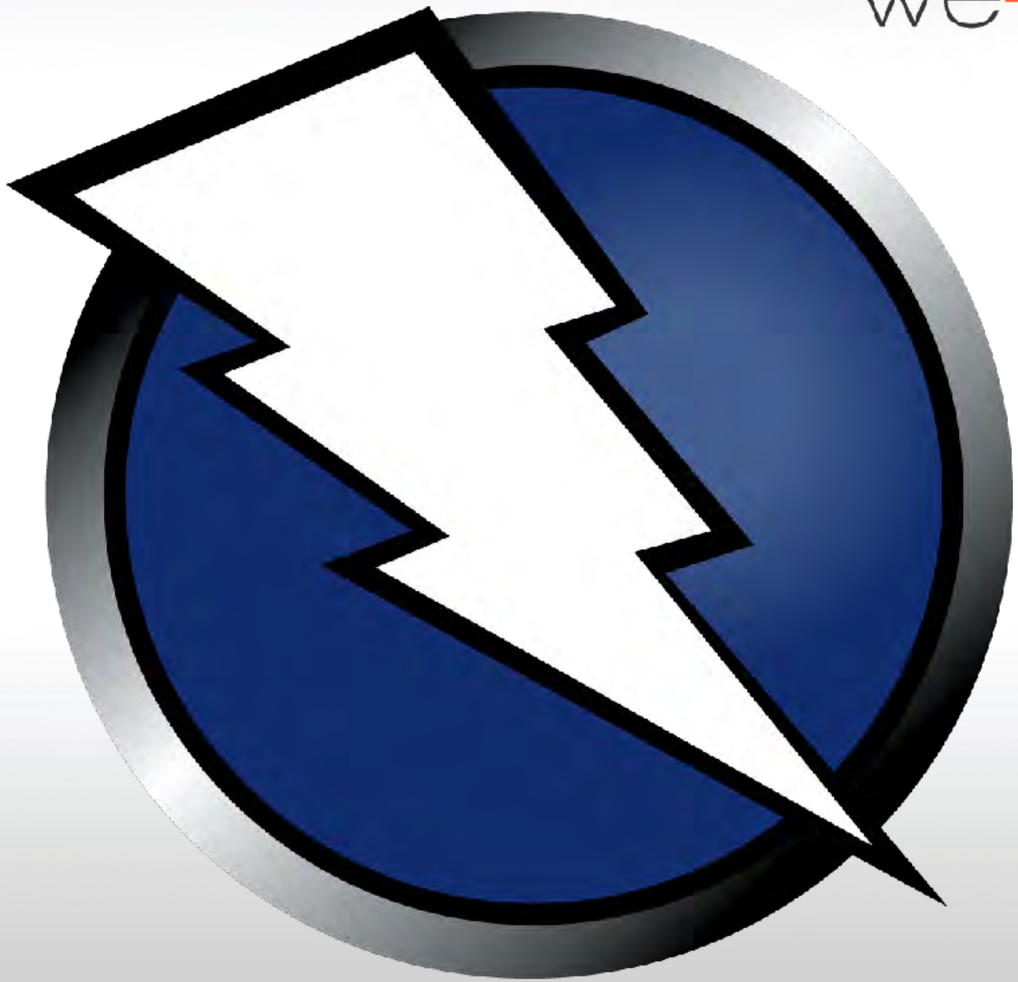
- How do we roll out Automated Security Testing in the pipeline?
  - Authenticated Scanning in the Pipeline - for Apps/API, etc
  - Account for changes in Attack Surface



Introduction to the OWASP ZAP API

we45

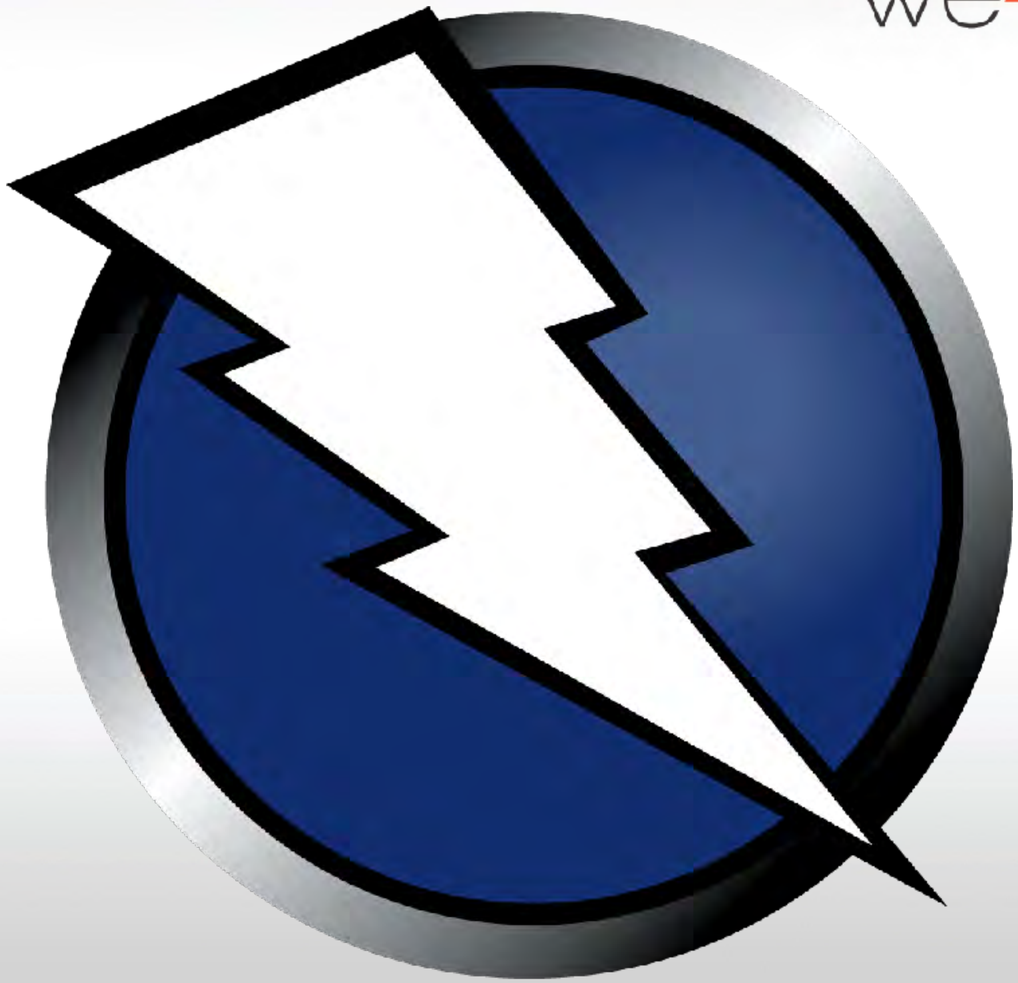
- OWASP ZAP - Automation
  - Concept Overview
  - Useful Concepts and API
- OWASP ZAP Python API Deep-Dive
- Workshop Exercises



Concept Overview - OWASP ZAP

we45

- Context
- Session
- Active Scan
- Passive Scan
- Scan Policy
- Alert



Workshop Exercise - Basic ZAP Functionality

- Concept overview:
  - Context
  - Sites
  - Scan Policy
  - Scripts
  - Script Console

ZAP - Useful API Operations


```
from zapv2 import ZAPv2 as ZAP

zap.spider
    #spider operations

zap.core
    #App-wide operations

zap.ascan
    #Active Scan

zap.pscan
    #Passive Scan

zap.script
    #Operations with ZAP Scripts

zap.context
    #Context related operations
```

ZAP API Quicksearch operations


```
zap.spider.scan()
    #initiate ZAP Spider Scan against target

zap.ascan.scan()
    #initiate ZAP Active Scan against Target

zap.core.alerts()
    #all alerts (scan results) from the ZAP Scanner

zap.core.urls()
    #list of URLs from ZAP

zap.ascan.status(), zap.spider.status()
    #real time status of the spider or ascan

zap.ascan.scan_progress()
    #List of Vulnerabilities being tested for with number of payloads
```

Workshop Exercise - ZAP API Walkthrough

1. `ipython` walkthrough
2. Walkthrough ZAP API Code - Please refer to Instructions in the HTML

Running Authenticated Scans in OWASP ZAP

we45

- Approaches:
  - Selenium-driven Scan Process
  - Leveraging canned ZAP Sessions
  - Zest Scripting

Selenium-Authenticated Scan

Run Selenium and ZAP in Headless Mode

Leverage Functional Scripts

Beats Spidering the app! :)

ZAP Session-Authenticated Scan

Programmatically invoked with ZAP API

Maintains state with Sessions/Tokens, etc

ZestScript Authenticated Scan

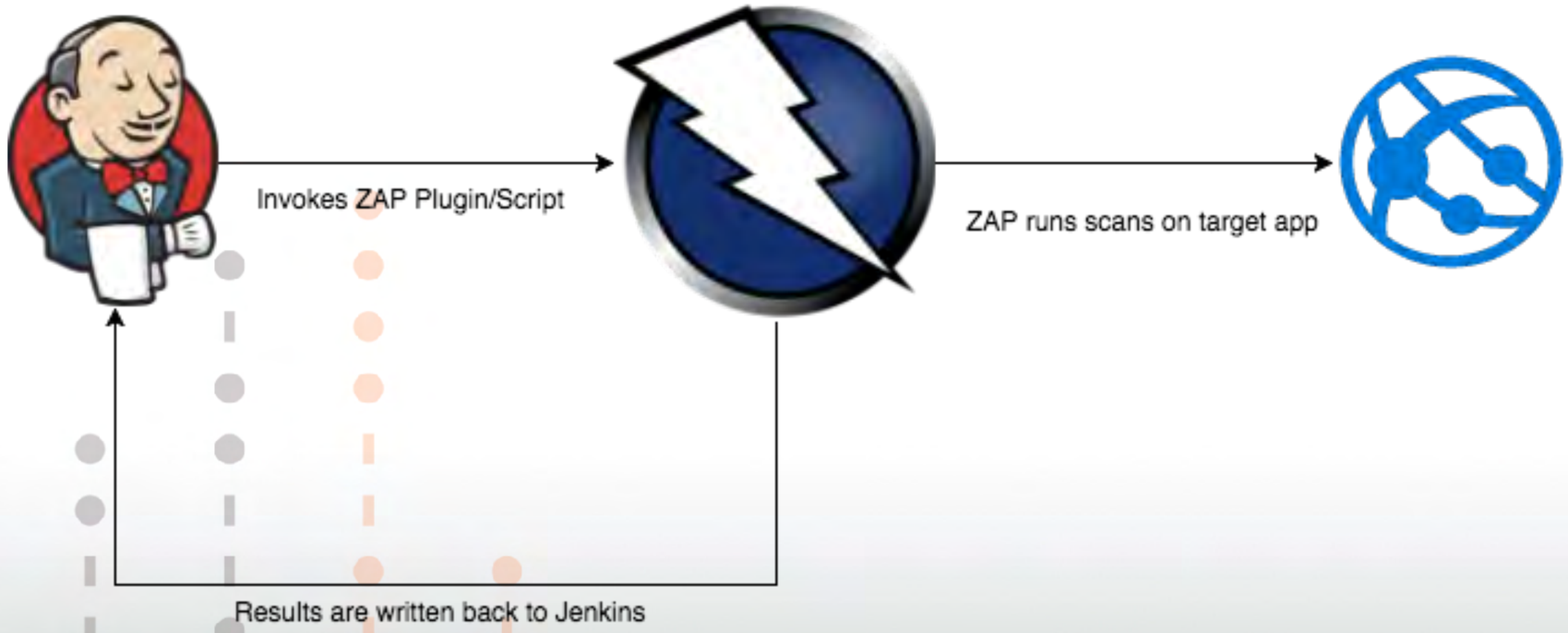
Programmatically invoked with ZAP API

Easily Customizable

Workshop Exercise - Automated, Authenticated ZAP Scans

1. Selenium-ZAP Scan - Follow the HTML Instructions
2. ZAP Session Scans - Follow the HTML Instructions
3. Zest ZAP Scans - Follow the HTML Instructions

ZAP in the Continuous Delivery Pipeline

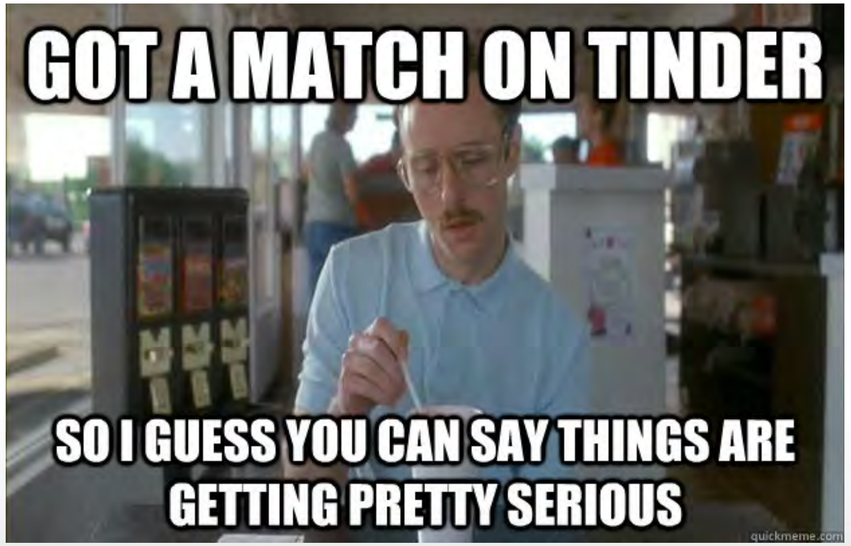


Workshop Exercise - Automated, Authenticated ZAP Scans

- Authenticated ZAP Scans - Jenkins Integration - Follow HTML Instructions

Correlating DAST Results

- The Common Weakness Enumeration (CWE) system is the best we have for correlation right now
- Problems:
  - Several tools don⁹t give any/accurate CWEs
  - Multiple CWE values tend to be difficult to handle and correlate with - BurpSuite, etc



Workshop Exercise

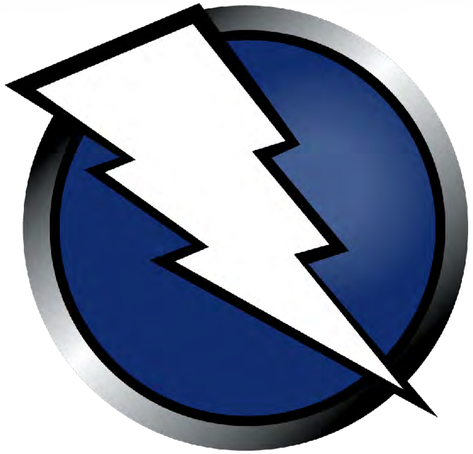
1. Correlation of Application Vulnerabilities based on CWE - Follow HTML Instructions
2. Diff Scans with ZAP - Follow HTML Instructions

AppSec Automation - A Pentester's perspective

we45

- How do we go beyond traditional DAST?
  - Scale Custom/Business Logic Security Flaws
  - Create Custom Application Exploits for non-standard/esoteric flaws
  - Create a Library of attacks extending/ complementing DAST Scanners

Tools we will use

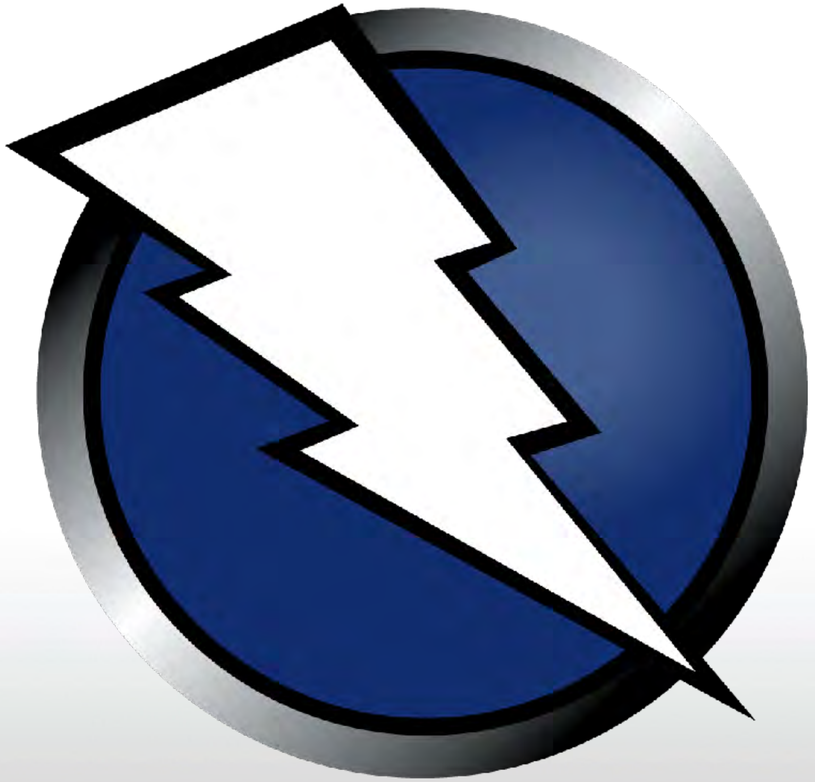


- OWASP ZAP 2.6.0
- mitmproxy 0.17

OWASP ZAP - Scripting Framework

we45

- Active Rules => Scripts invoked during Active Scan
- Authentication Scripts => Scripts invoked to facilitate authentication for a Context
- Fuzzer Processors => Scripts invoked after Fuzzers are run with ZAP
- HTTPSender => Scripts invoked against every request/ response received by ZAP
- Proxy => Runs inline and acts on all requests and responses
- Targeted Rules => Invoked on specific urls or on manual start only
- Standalone => Invoked manually
- Passive Rules => Passive Scanning Rules



Configuring ZAP to run with Python

- ZAP supports scripts written in Jython
  - Python on Java JVM
  - Not fully compatible with python libraries
  - limitations on networking and i/o libraries in python
- Works when [8]Python Scripting[9] add-on is installed in OWASP ZAP.
- Third Party Python Libs can be linked when refer to the jython `site-packages` directory

mitmproxy

- Primarily used as an extensible, interception proxy.
- Powerful Inline scripting framework
- Pure Python implementation :) - Highly extensible and scriptable
- Current version is 2.x on python 3 only

ZAP Scripting QuickSearch


```
msg
    #the message object that is acted upon to parse/manipulate

msg.getRequestHeader()
    #Request Header Object

msg.getRequestHeader().getURI()
    #fetches the URI from the request header

msg.getRequestBody()
    #Fetches the request body from the request

msg.getResponseBody()
    #Fetches the request body from the request

msg.setRequestBody()
    #Sets a different request body from the one in the original request
```

ZAP Active Rules Template


```
"""
The scanNode function will typically be called once for every page
The scan function will typically be called for every parameter in every
URL and Form for every page
"""
def scanNode(sas, msg):
    #Invoke something for every page here

def scan(sas, msg, param, value):
    #invoke something for every param here.

    sas.raiseAlert(1, 1, 'Active Vulnerability title', 'Full description',
                   msg.getRequestHeader().getURI().toString(),
                   param, 'Your attack', 'Any other info', 'The solution
', '', 0, 0, msg);
```

mitmproxy inline scripting


```
def request(context, flow):
    flow.request.headers
        #request headers object

    flow.request.host
        #host in the request

    flow.request.path
        #request path

    flow.request.content
        #request body


def response(context, flow):
    flow.response.headers
    # request headers object

    flow.response.host
    # host in the request

    flow.response.path
    # request path

    flow.response.content
    # request body
```

Workshop Exercises

1. ZAP POST Request Insecure Direct Object Reference Active Script
2. ZAP JSON Insecure Direct Object Reference Active Script
3. ZAP Standalone Script
4. mitmproxy JWT Bruteforce Script
5. mitmproxy JWT Attribute check script