
black hat®
ASIA 2017
MARCH 28-31, 2017
MARINA BAY SANDS / SINGAPORE

black hat ASIA 2017

Domo arigato, Mr. Roboto: Security Robots a la Unit-Testing

Seth Law

seth@nvisium.com

Twitter: @sethlaw

Introduction

Who am I?

- From Salt Lake City, UT
- Chief Security Officer at nVisium
- Focused on Application Security
- Previously presented at Black Hat on Mobile Application Security (SiRATool) and Response Analysis and Further Testing (RAFT)
- Soccer Hooligan

nVISIUM

amazon web services | Partner Network

CONSULTING PARTNER

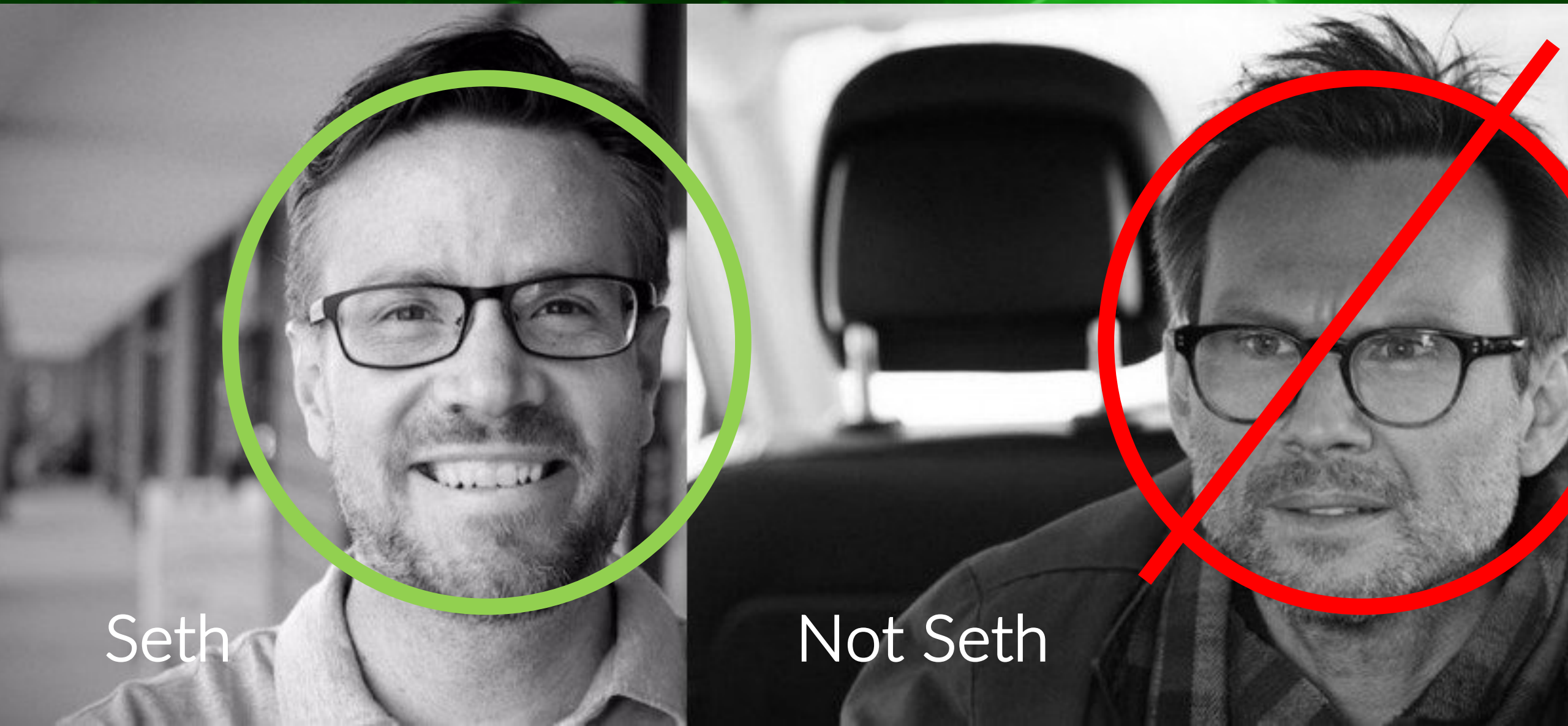
Who am I?

Jessica Ryan @Jhyp3 · Feb 28

@sethlaw @miketweaver @BsidesSLC I'm so excited but I can't unsee you as anyone other than the dad from Mr Robot

black hat ASIA 2017

Who am I?

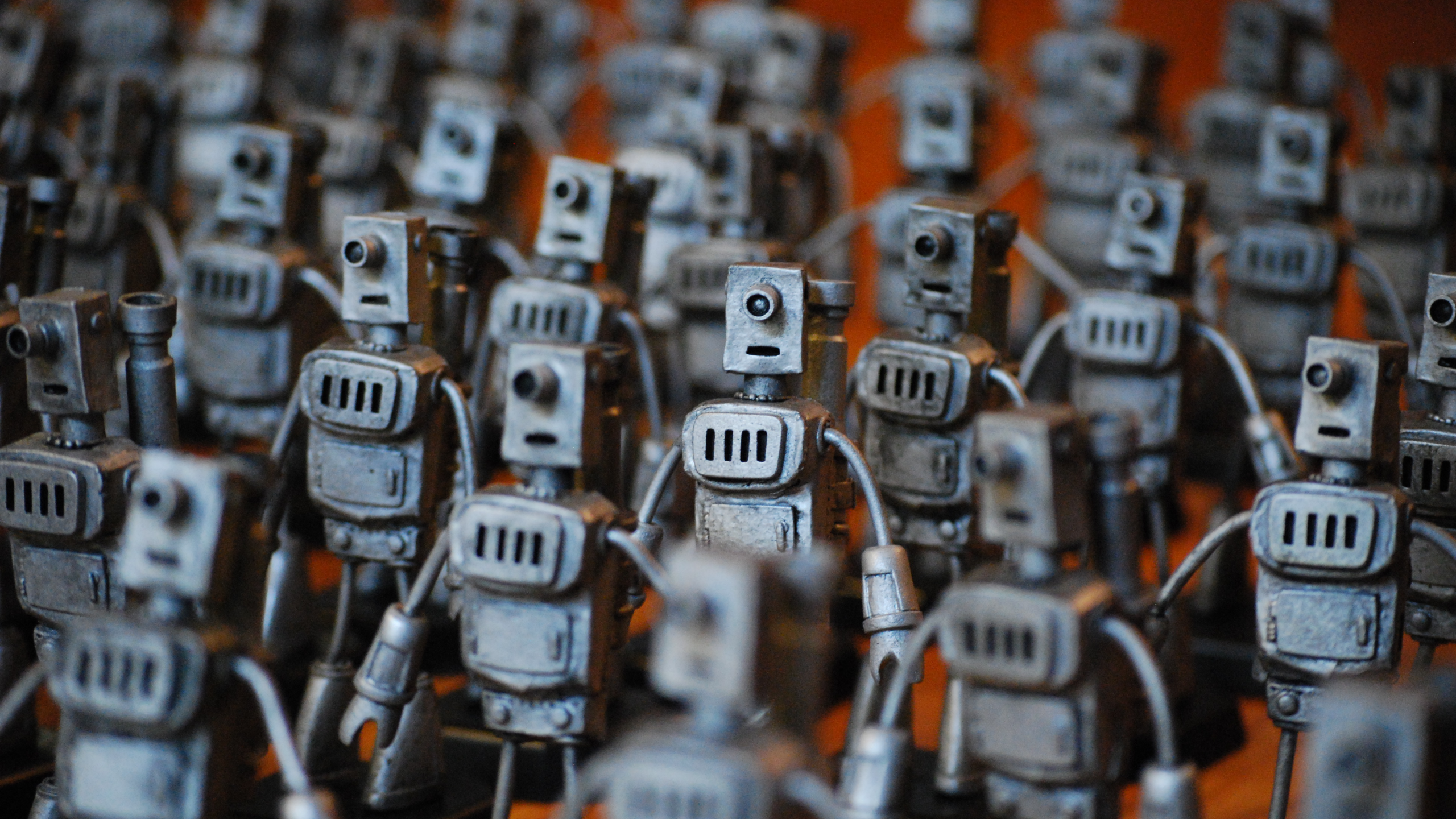


Security Unit-Testing

Why are we here?

Why are we here?

- Security goals != Development goals
- Security tools don't always fit into the development pipeline
- Business goals are at odds with full coverage security testing
- Solve these problems with Test Driven Development (TDD) tools.

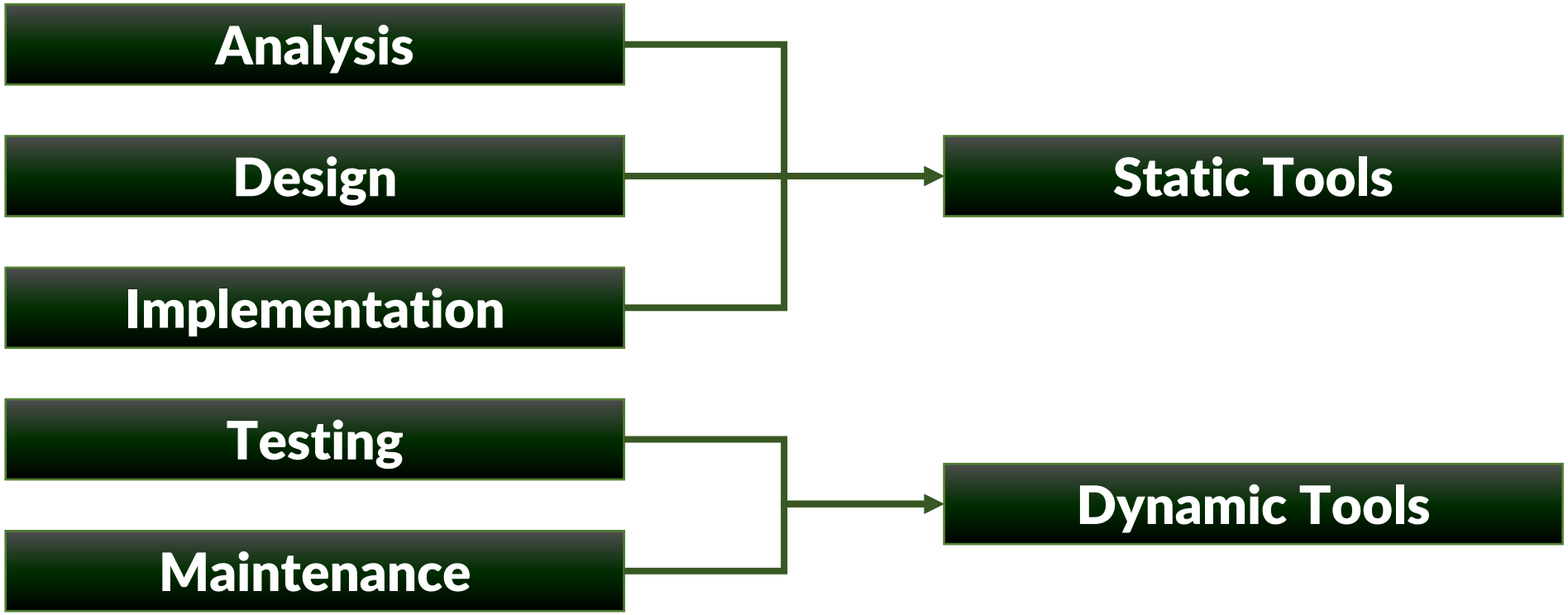


Agenda

- Current Security Testing Tools
- Unit Testing Frameworks
- Security Unit Testing
- Requirements for Security Unit Testing
- Security Payload Unit Testing Repository/Runner

Current Security Testing Tools

Current Security Testing Tools



Current Security Testing Tools

- Target specific needs in the SDLC
- Vulnerability identification and false positive reduction
- Easy to use, hard to absorb
- Typically driven by compliance needs
- Divided into static and dynamic tools

Dynamic Tools

- Interact with running application to identify vulnerabilities
- Implemented into SDLC by build or DevOps engineers
- Happen later in the SDLC after successful application builds
- Glorified quality assurance integration test

Static Tools

- Inspects and instruments application source to identify vulnerabilities
- Implemented into SDLC by developers or build engineers
- Introduced early in the SDLC during development with developer IDE integration
- Cross between functional and integration test

Tool Strengths

- Speed of setup
- Ability to meet compliance needs
- Proficient at identifying generic vulnerabilities with known vulnerabilities
- Usually start out as regular-expression engines with vulnerability-specific payloads

Tool Weaknesses

- False negatives due to generic identification of vulnerabilities through exploitation payloads
- Lack of human component means full classes of vulnerabilities are ignored (business logic, authorization, …)
- Edge cases are ignored because of timing needs.

Unit Testing Frameworks

Unit Testing Frameworks

- Frameworks & languages have built-in scaffolding for testing
- Can include mock controllers, third party libraries, and test runners
- Try to fill need for low-level unit testing along with complete integration testing.

Java Spring Unit Testing

- Allows testing without full Spring or other containers
- Framework provides mock objects for environment, jndi, servlets, and portlets
- Also includes basic reflection test objects and MVC to access Model and View objects.

Java Spring Integration Testing

- Allows testing with full Spring environment, data access via JDBC or ORM
- Provides context and transaction management, dependency injection, and support classes
- Means you can interact with any piece of the application without using the Spring app.

ASP.NET MVC Testing

- Allows testing of an MVC application
- Built-in unit test framework directly calls MVC controllers methods
- Not available in all versions of Visual Studio ($$$)
- Ability to mock different components using built-in and 3rd party frameworks

ASP.Net MVC Testing

- No access to HTML
- Limited access to full HTTP Request/Response

Django Testing

- Uses python standard unit test library
- Hybrid of unit/integration test framework
- Auto-creates model database for tests
- Test client acts as dummy web browser with low-level access to HTTP Request/Response

Testing Frameworks Summary

- Unit test frameworks focus on low level functionality (ASP.NET, Java Spring Unit Tests, etc)
- Integration test framework is needed for security testing

Security Unit Testing

Security Unit Testing Requirements

- Functional application
- Maintain authentication state
- Consistent responses

Functional Application

- Application should run in a production-like state, including:
  - Mock and/or test data
  - Full HTTP Request/Response
  - Rendered HTML

Maintain Authentication State

- Application should perform authentication and authorization functions
  - Goes back to functional application
  - Full vulnerability classes depend on this functionality.
  - Include login, logout, and registration functions

Consistent Responses

- Application should maintain state during the duration of a test
  - Still part of a functional application
  - Allow for multiple calls in one test

Java Spring Example

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes =
        {MvcConfig.class,MoneyxApplication.class},
        webEnvironment =
        SprintBootTest.WebEnvironment.RANDOM_PORT)
public class InjectionTest extends MoneyXTestTemplate {
    @LocalServerPort
    private int port;
```

ASP.NET MVC Example

```
private void StartIIS() {
  var appPath = GetApplicationPath(_appName);
  var pf = Environment.GetFolderPath(
      Environment.SpecialFolder.ProgramFiles
  );
  _iis = new Process();
  _iis.StartInfo.FileName = pf +
                            @"\IIS Express\iisexpress.exe"
  _iis.StartInfo.Arguments = string.Format("/path:\"{0}\" /port:{1}",
                                           appPath, 2020);
  _iis.Start();
}
```

Python Django Example

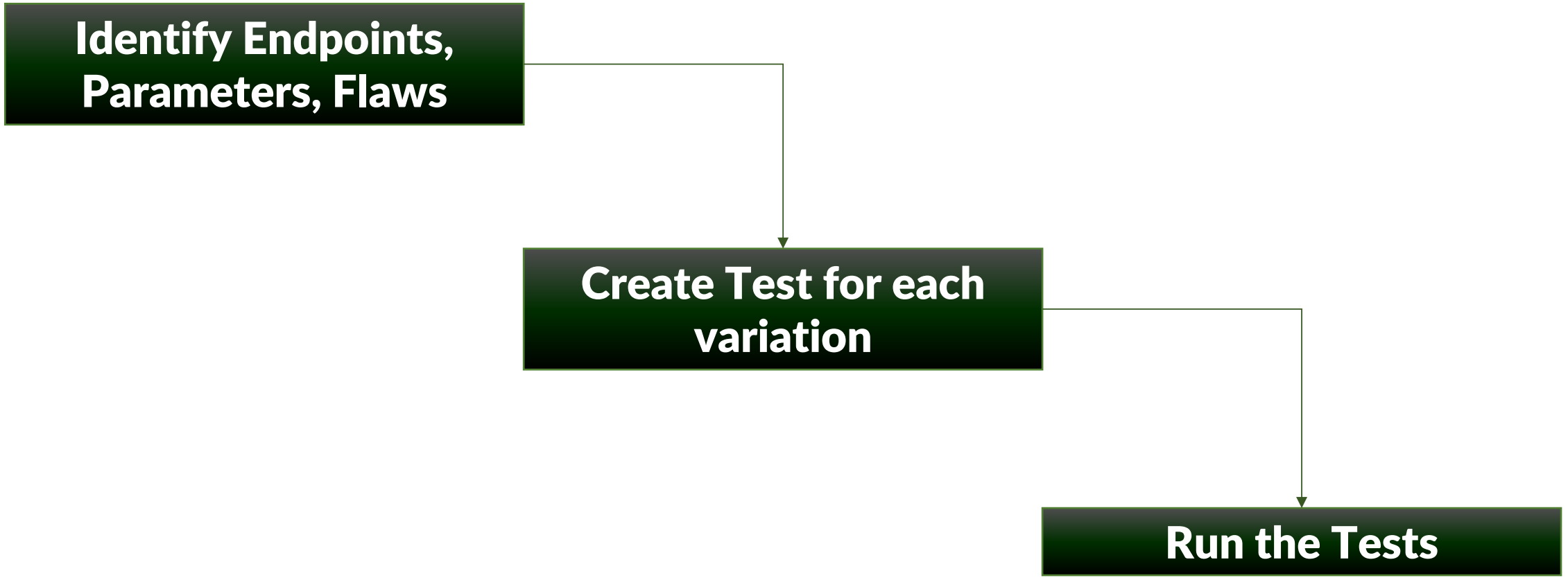
```
class TestSecurity(TestCase):
      “Security Tests”
      fixtures = [‘users’,’userProfiles’,’groups’]

      def setUp(self):
            self.client = Client()

      def test_caching(self):
            vuln = False
            req = self.client.login(username=‘test’,
                                    password=‘pass’)
      …
```

Security Unit Testing Approach

Approach

- Building one security unit test != impenetrable application
- Must test each endpoint
- AND each parameter

Approach


Identify Endpoints, Parameters, Flaws
Create Test for each variation
Run the Tests

Identify

- Not for the faint of heart
- Each endpoint (and variation)
- Each parameter wherever it may be
- Each security flaw

Create

- Build your security unit test bots
- Hundreds, maybe thousands.
- Each test == 1 flaw, 1 endpoint, 1 parameter

Test

- Turn them loose
- Timing becomes important
- Parallel execution for faster completion
- Environment startup for each test, class of vuln, endpoint
- Database resets between tests

Security Payload Unit Testing Repository/Runner

SPUTR

- Building intentionally-vulnerable applications
- Test known vulnerable endpoints and parameters
- Security payloads are exploit focused, redundant and produce false-positives
- Speed up security integration into SDLC

Current Security Payloads

- Developed to uncover exploitable flaws for false positive reducion
- Use generic escape sequences and payloads
- Focused on application output more than input

SPUTR Payloads

- Focused on payloads that cause application errors, not exploitation
- Eliminates redundant testing of the same escape sequences

XSS Payloads from fuzzdb

```
'
 <font style='color:expression(alert('XSS'))'>
' onmouseover=alert(/Black.Spook/)
' or 2=2
"
" or 202
";eval(unescape(location))//#  %0Aalert(0)
"><BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>
"><iframe%20src="http://google.com"%%203E
"><img src=x onerror=prompt(1);>
"><img src=x onerror=window.open('https://www.google.com/');>
'%22--%3E%3C/style%3E%3C/script%3E%3Cscript%3Eshadowlabs(0x000045)%3C/script%3E
%27%22--%3E%3C%2Fstyle%3E%3C%2Fscript%3E%3Cscript%3ERWAR%280x00010E%29%3C%2Fscript%3E
%3Cscript%3Exhr=new%20ActiveXObject%28%22Msxml2.XMLHTTP%22%29;xhr.open%28%22GET%22,%22/xssme2%
&#x61;l&#x65;rt&#40;1)
&<script&S1&TS&1>alert&A7&(1)&R&UA;&&<&A9&11/script&X&>
```

XSS Payload from SPUTR

```
4j0kh"4j0kh
```

Payload Generation

Demo

SPUTR Generation

- Needed to identify endpoints of different frameworks
- Starting point for unit-test creation
- Still need to map which tests apply to which endpoints

Generation

Demo

SPUTR Testing

- Needed to test multiple application built using different frameworks
- Speed up building of tests

Testing

Demo

SPUTRing the future

- Payloads
  - Further payload options + refinement
  - Additional vulnerabilities (IDOR/Redirects/etc)
- Testing
  - Speed
- Generation
  - More languages and frameworks
  - Burp Suite Pro plugin

Summary

- Current security testing tools are great at finding some vulnerabilities, but not all
- Creation of simple security bots for unit testing specific functionality reveal additional flaws.
- Use SPUTR (https://github.com/sethlaw/sputr) in a DevOps pipeline to speed up security bot creation.