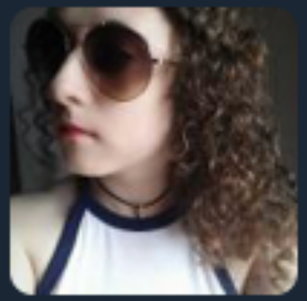
Domo arigato, Mr. Roboto: Security Robots a la Unit-Testing

Seth Law
seth@nvisium.com
Twitter: @sethlaw

Introduction

Who am I?



Jessica Ryan @Jhyp3 2/28/17

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

Who am I?

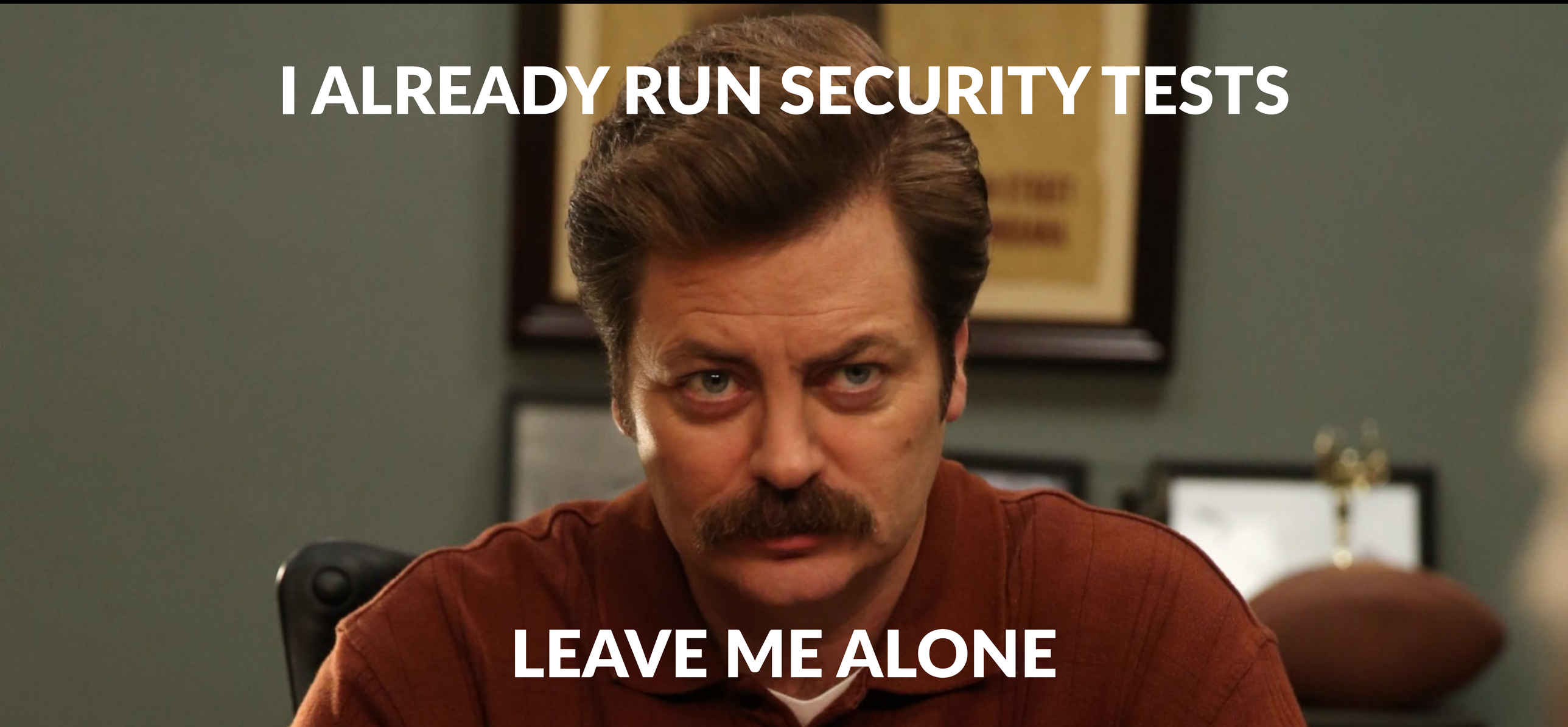
Who am I?

- From Salt Lake City, UT
- Chief Security Officer at nVisium
- Focused on Application Security
- Developer/Security Engineer/Consultant/ Speaker
- Soccer Hooligan

nVISIUM

Security Unit-Testing

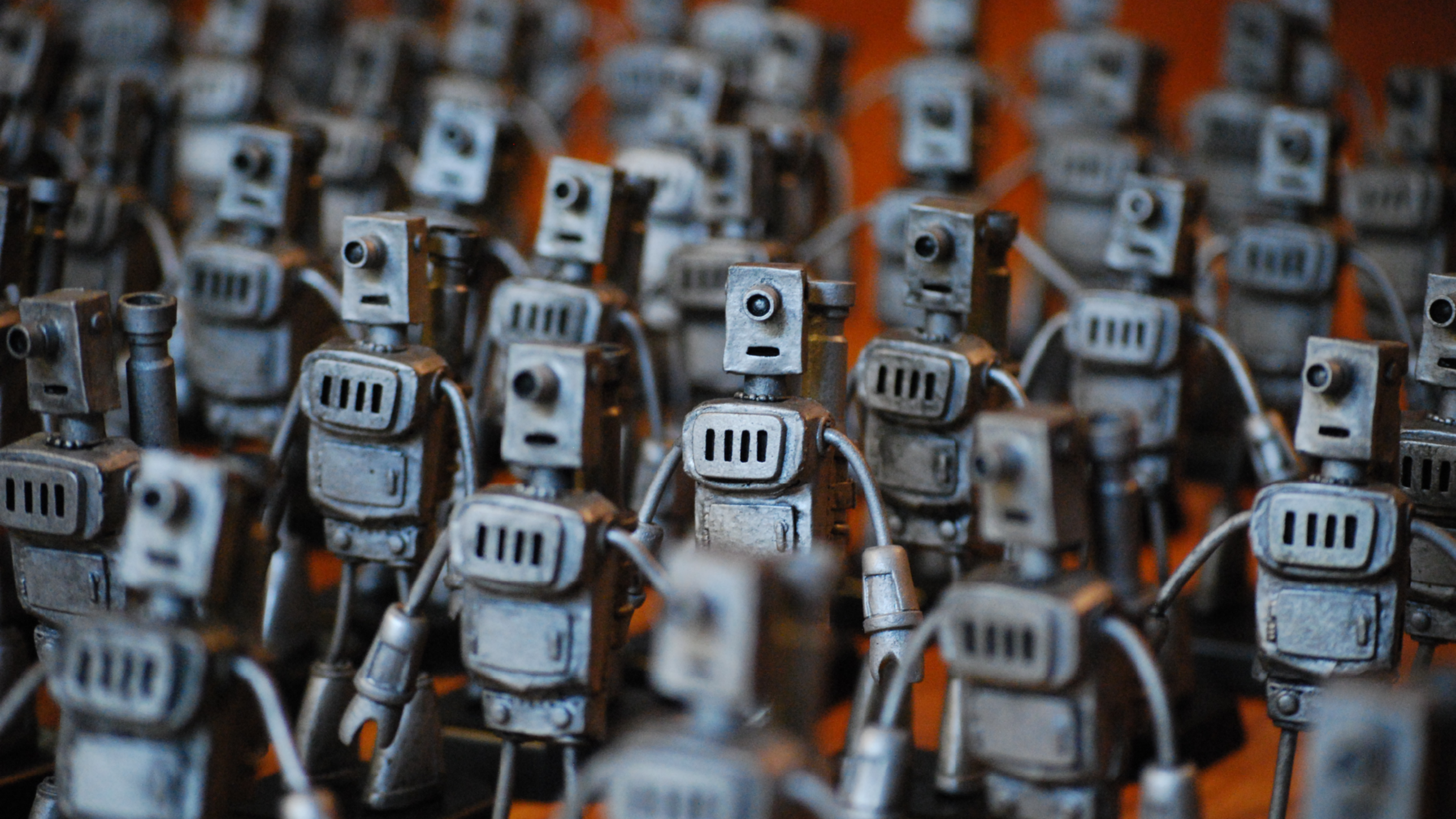
Why are we here?



Why are we here?

- Security goals != Development goals
- Existing 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.

Find flaws, not Exploits



Agenda

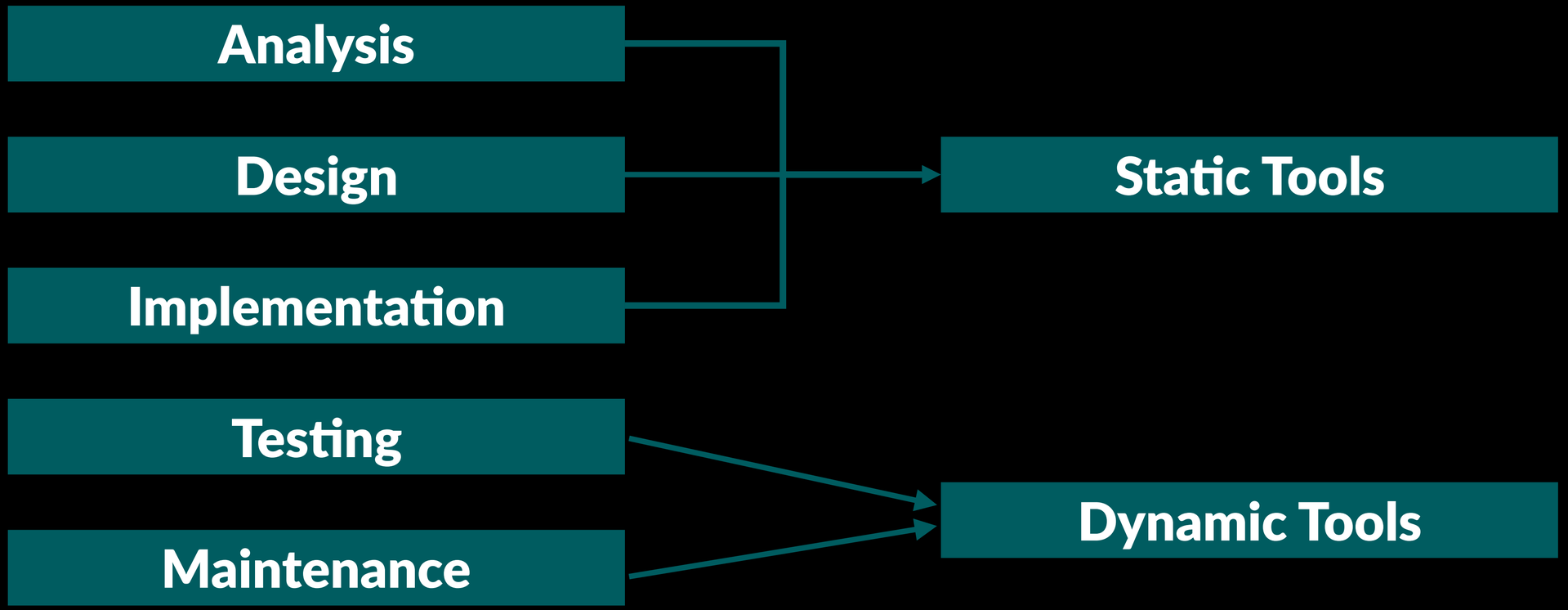
- Current Security Testing Tools
- Unit-Testing Frameworks
- Security Unit-Testing Requirements
- Security Unit-Testing Approach
- Security Payload Unit-Testing Repository/Runner (SPUTR)

Current Security Testing Tools

Current Security Testing Tools

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

Current Security Testing Tools



Dynamic Tools

- Interact with running application to identify vulnerabilities
- Usually implemented by security engineers
- Happen later in the SDLC after successful application builds
- Glorified QA 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/configuration
- Meet compliance needs
- Identify vulnerabilities with known exploits/ payloads
- Regular-expression engines with vulnerability-specific payloads

Tool Weaknesses

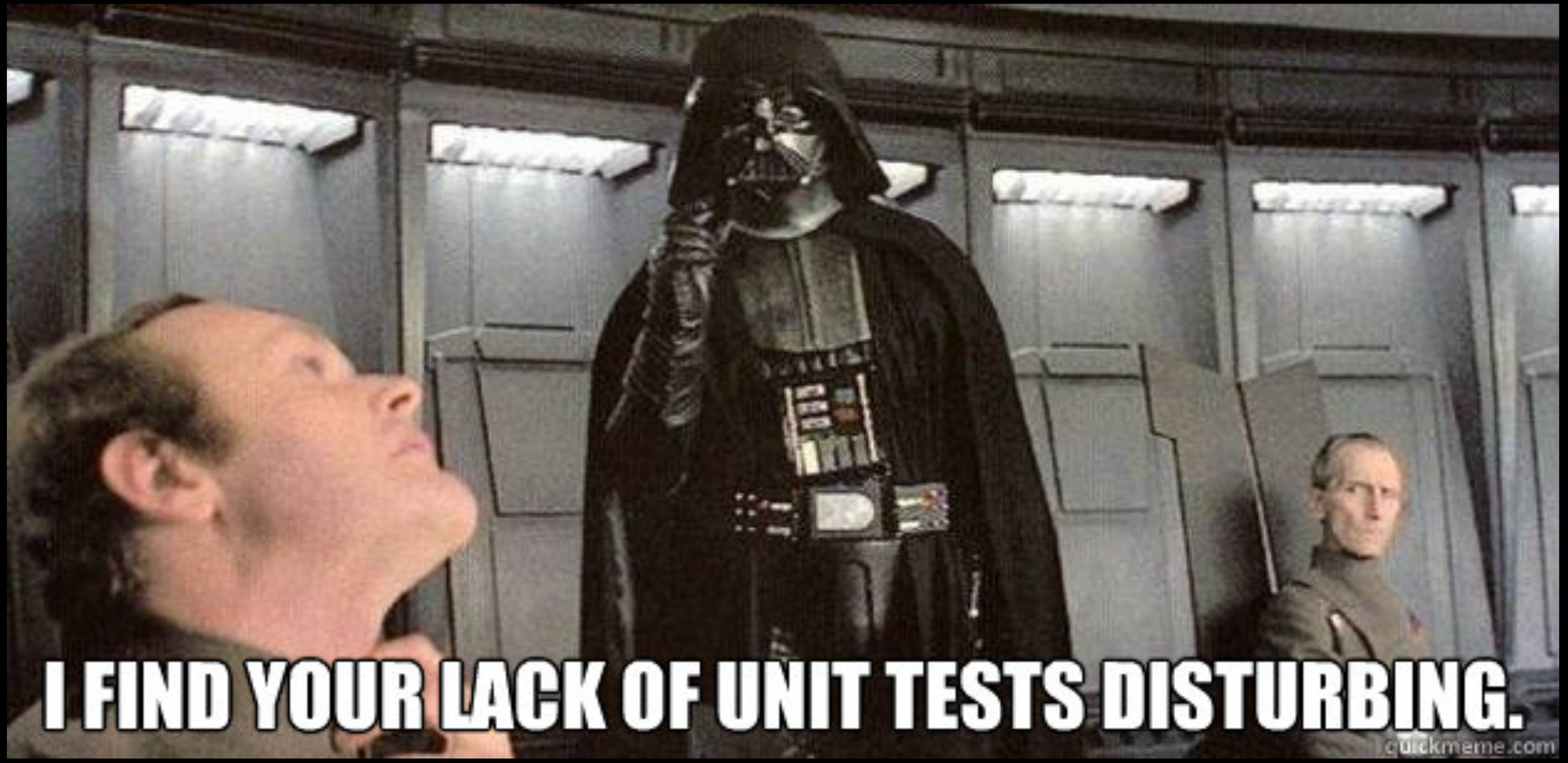
- False negatives due to generic identification of vulnerabilities through exploitation payloads
- Lack of human component means full classes of vulnerabilities are ignored (business logic, authorization, …)
- Edge cases are ignored because of timing needs.
- Cost

Unit-Testing Frameworks

15 lines (12 sloc) | 314 Bytes

Raw Blame

```
1  [redacted]
2
3  import org.junit.Test;
4
5  import static org.junit.Assert.*;
6
7  /**
8   * To work on unit tests, switch the Test Artifact in the Build Variants view.
9   */
10 public class ExampleUnitTest {
11     @Test
12     public void addition_isCorrect() throws Exception {
13         assertEquals(4, 2 + 2);
14     }
15 }
```


I FIND YOUR LACK OF UNIT TESTS DISTURBING.
quickmeme.com

Unit-Testing Frameworks

- Frameworks & languages have built-in scaffolding for testing
- Include mock controllers, third party libraries, and test runners
- Cover low-level unit testing to 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 application server

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

.NET

ASP.NET MVC Testing

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

.NET

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 provide more of a full-stack approach to testing components

Security Unit-Testing Requirements

Security Unit Testing Requirements

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

- Unit-Test framework must perform authentication and authorization functions
  - Working client AND 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();
}
```

.NET

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 Lessons Learned

- Requires unique setup for each language and framework
- Spend as much time meeting requirements as writing tests
- Combination of dynamic and static security testing

Security Unit-Testing Approach

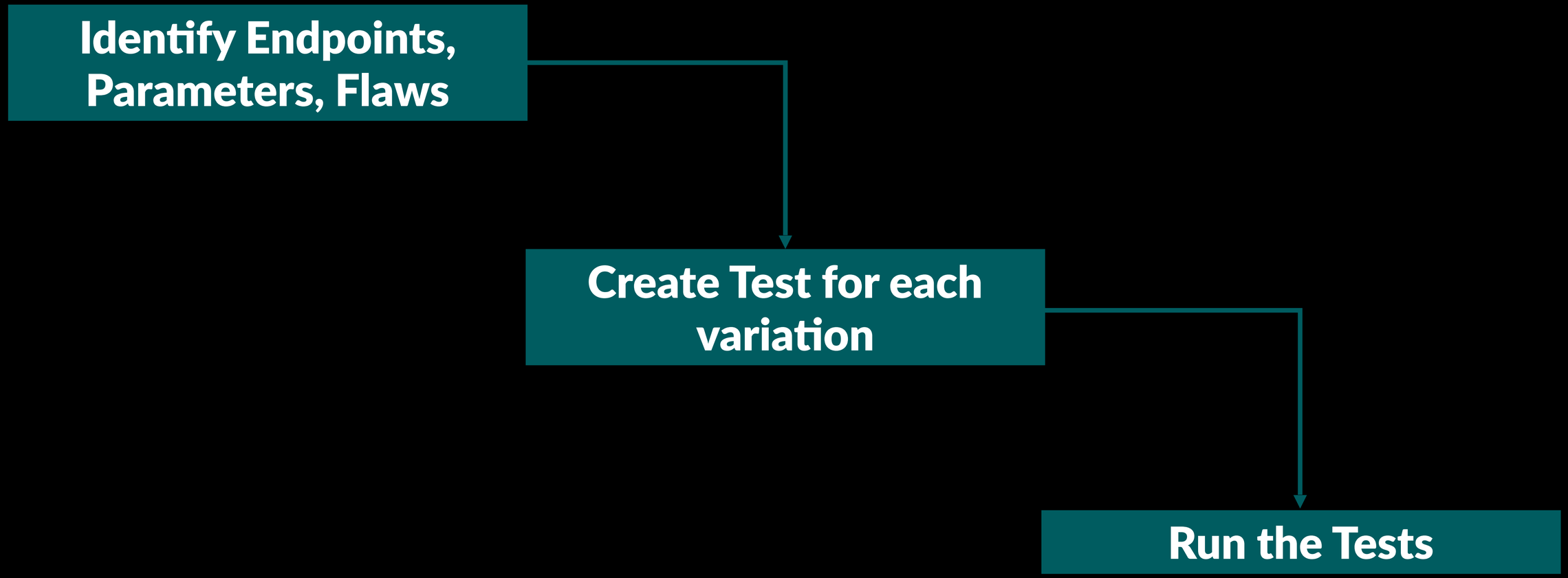
Security Unit-Testing Approach

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

Math is hard

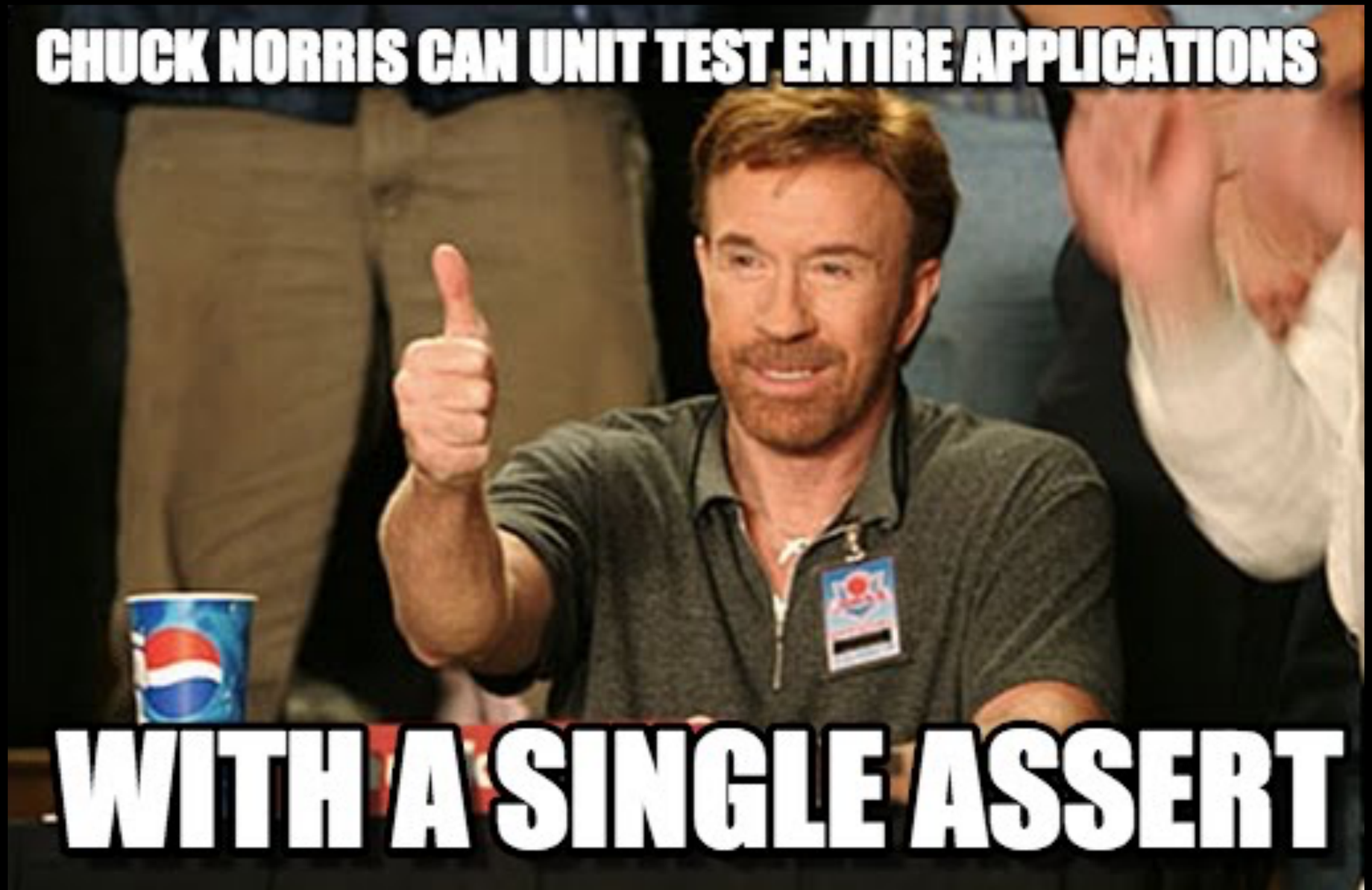
- 10 endpoints
- 10 parameters on each endpoint
- 10 vulnerabilities for each parameter
- 5 payloads per vulnerability
- 10x10x10x5 = 5000 tests

Security Unit-Testing Approach

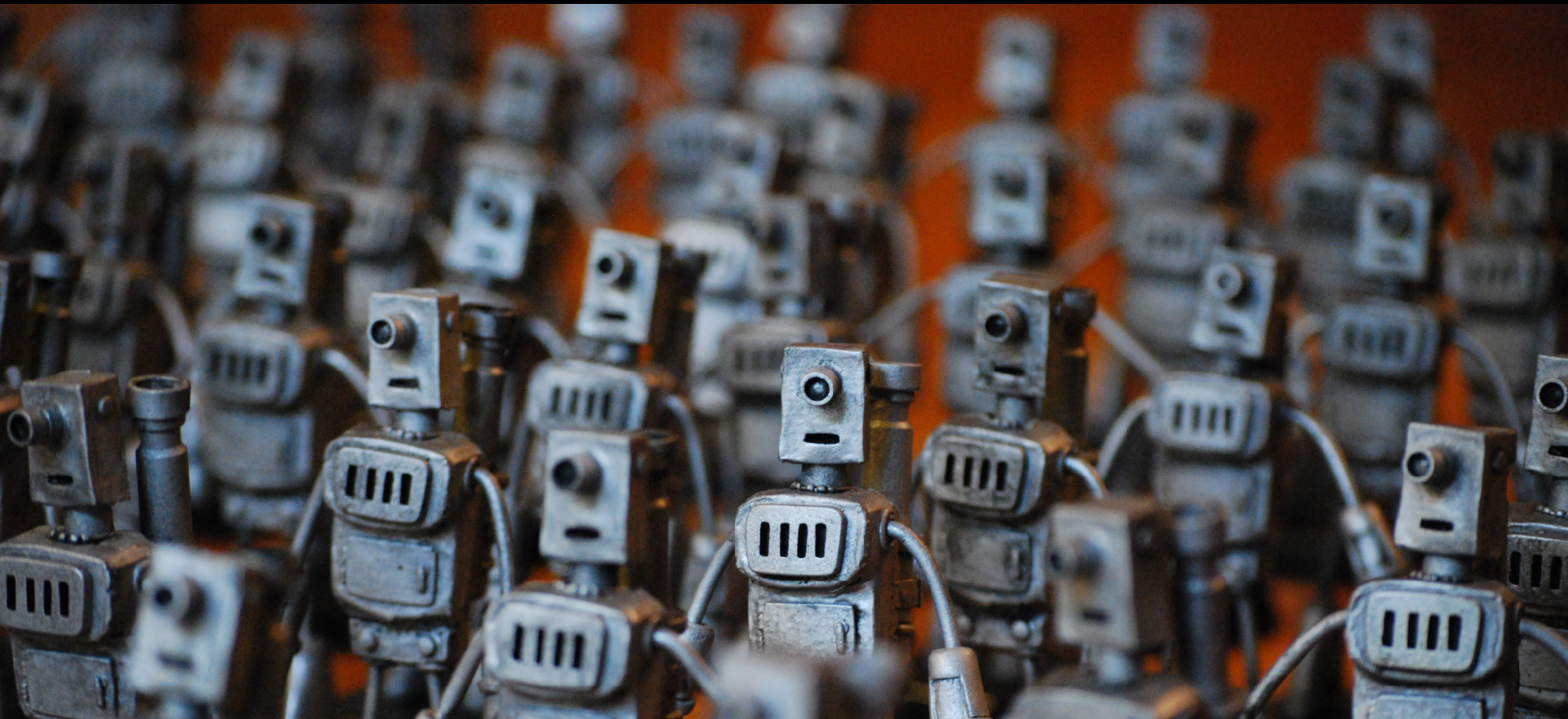


Identify

Create


CHUCK NORRIS CAN UNIT TEST ENTIRE APPLICATIONS
WITH A SINGLE ASSERT

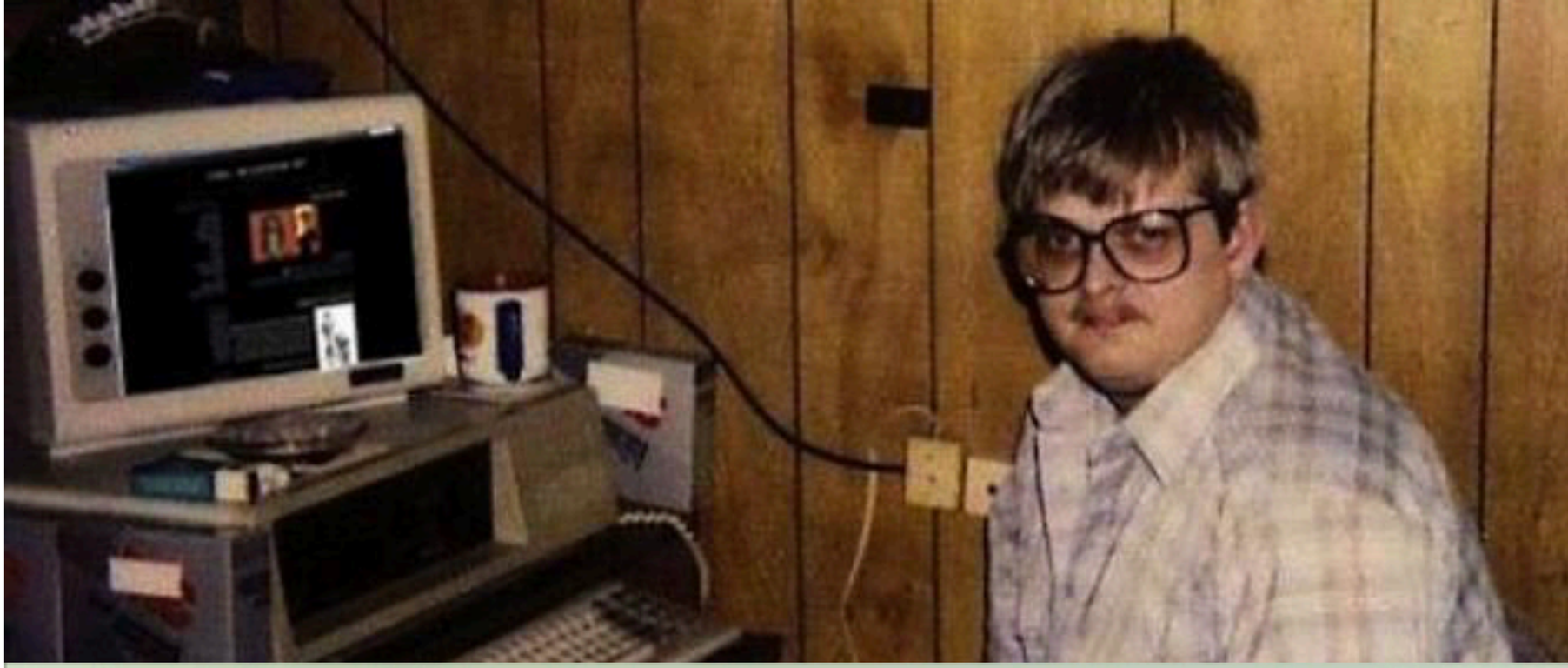
Test



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



~~0046~~ 0096
~~0049~~ 0097
~~0050~~ 0098

Current Security Payloads

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

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&>
```

“ ‘ & %

SPUTR Payloads

- Focus on characters and strings that expose application errors, not exploitation
- Eliminate redundant testing of the same escape sequences

XSS Payload from SPUTR

```
4j0kh"4j0kh
```

Payload Generation

Demo

SPUTR Test Generation

- Identify as many endpoints as possible from the code of different frameworks
- Starting point for unit-test creation
- Map which parameters and tests apply to the endpoints

Generation

Demo

SPUTR Testing

- Consistent way to test multiple application built on different languages and frameworks
- Callable from AWS CodePipeline or Jenkins
- Decrease cost of building unit tests

Testing

Demo

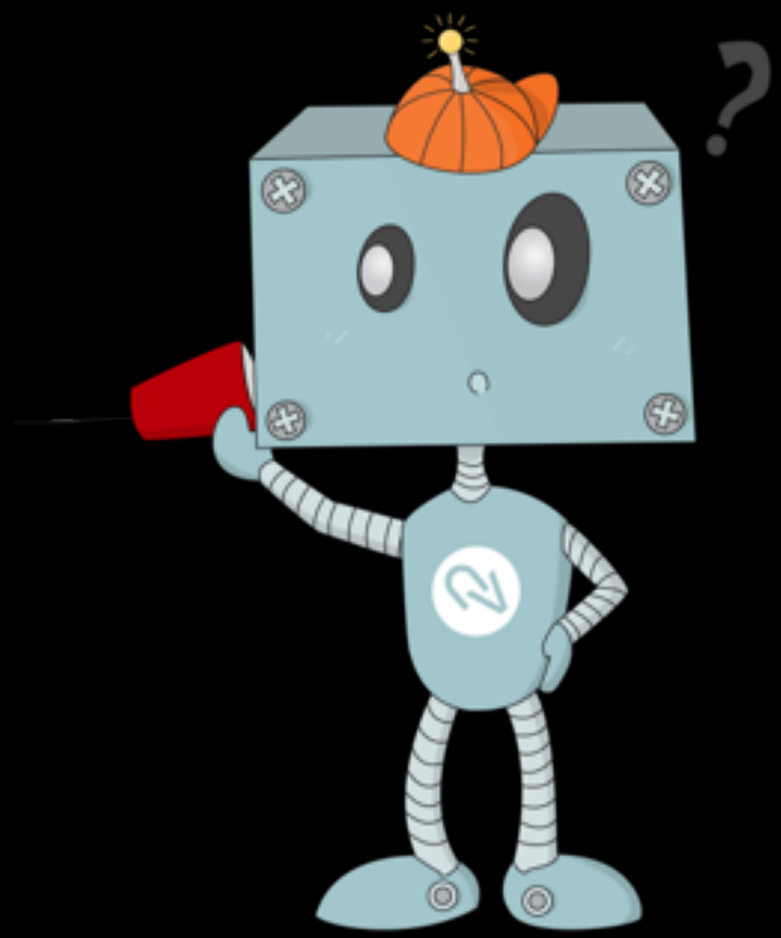
SPUTRing the future

- Payloads
  - Further payload options + refinement
  - Additional vulnerabilities (IDOR/Redirects/etc)
- Testing
  - Speed
- Generation
  - Automated analysis
  - 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.

Questions



Seth Law
seth@nvisium.com
Twitter: @sethlaw