Testing With Your Left Foot Forward

Jeremy Long

18 years information security experience

13 years software development experience

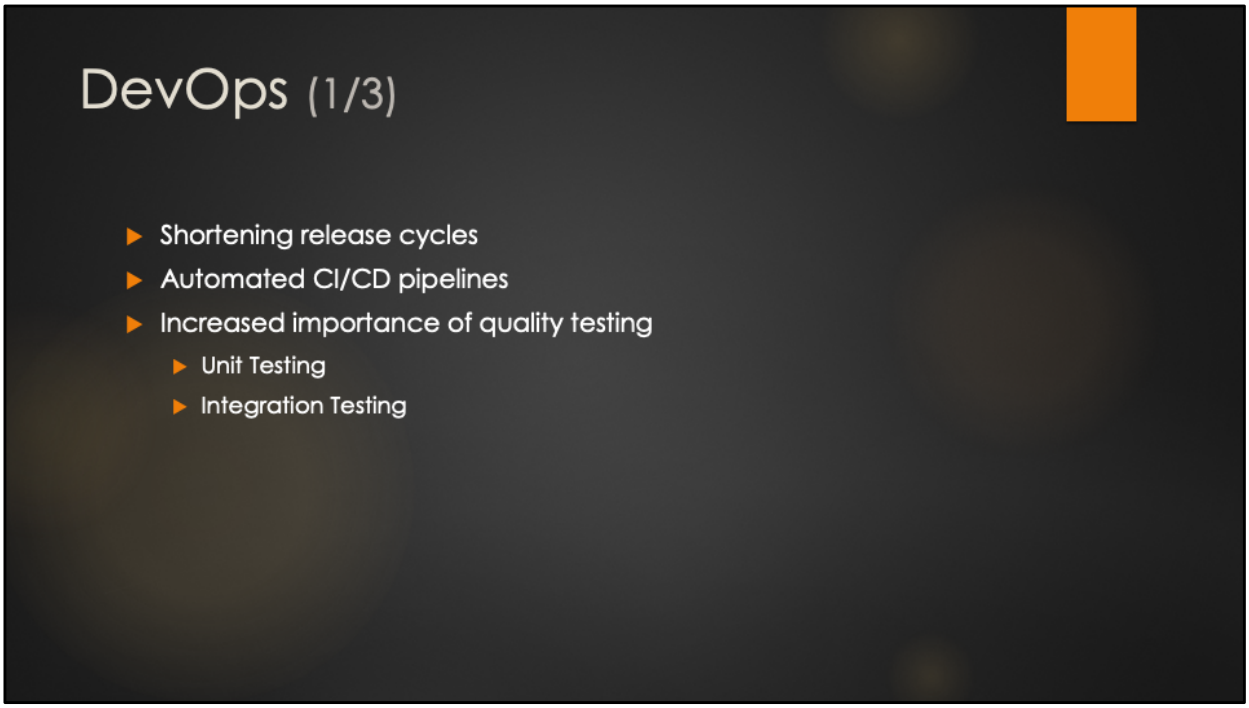
SAST enthusiast

OWASP dependency-check project lead

@ctxt / jeremy.long@gmail.com

What are we going to talk about?

- DevOps
- DAST
  - Defect Trackers
  - Pipeline integrations
- Manual Testing
  - Reporting Findings
- Traditional QA Testing Tools
  - REST Assured
  - Selenium
- Delivering Test Cases

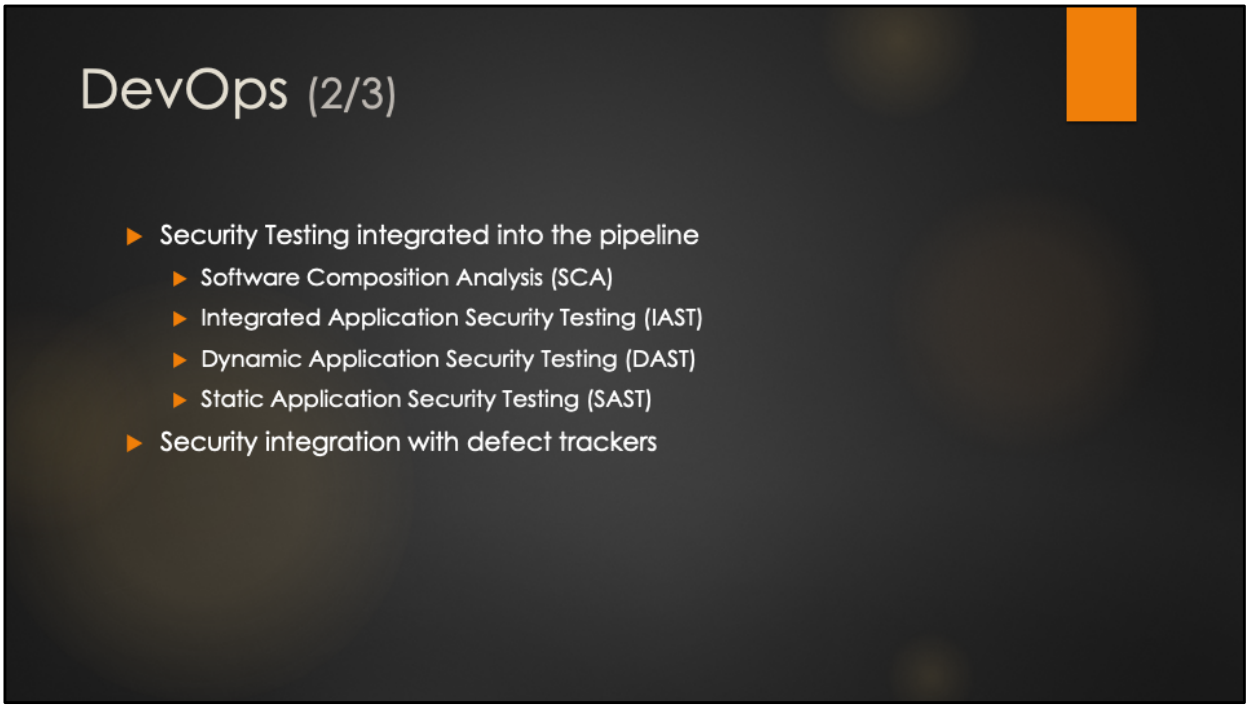


DevOps - I said this would be a brief overview as entire talks are devoted to this topic – so I'm just going to pick on a few items.

We are seeing shorter release cycles – teams are going from releasing every 3 to 18 months to release 3 to 18 times an hour.

Our build pipelines have become truly amazing in the amount and variety of things we are automating as part of our CI/CD. Advances with cloud and infrastructure as code allows an unparalleled opportunity to release fast.

The only way DevOps makes sense is with an increase in automated testing. If we are releasing often – we need to ensure we aren't breaking things. We have embraced an increased importance on unit and integration testing. I see industry standards looking for 80% code coverage – as I was putting this talk together I was talking to my buddy Steve and his team has set an 85% code coverage commitment.

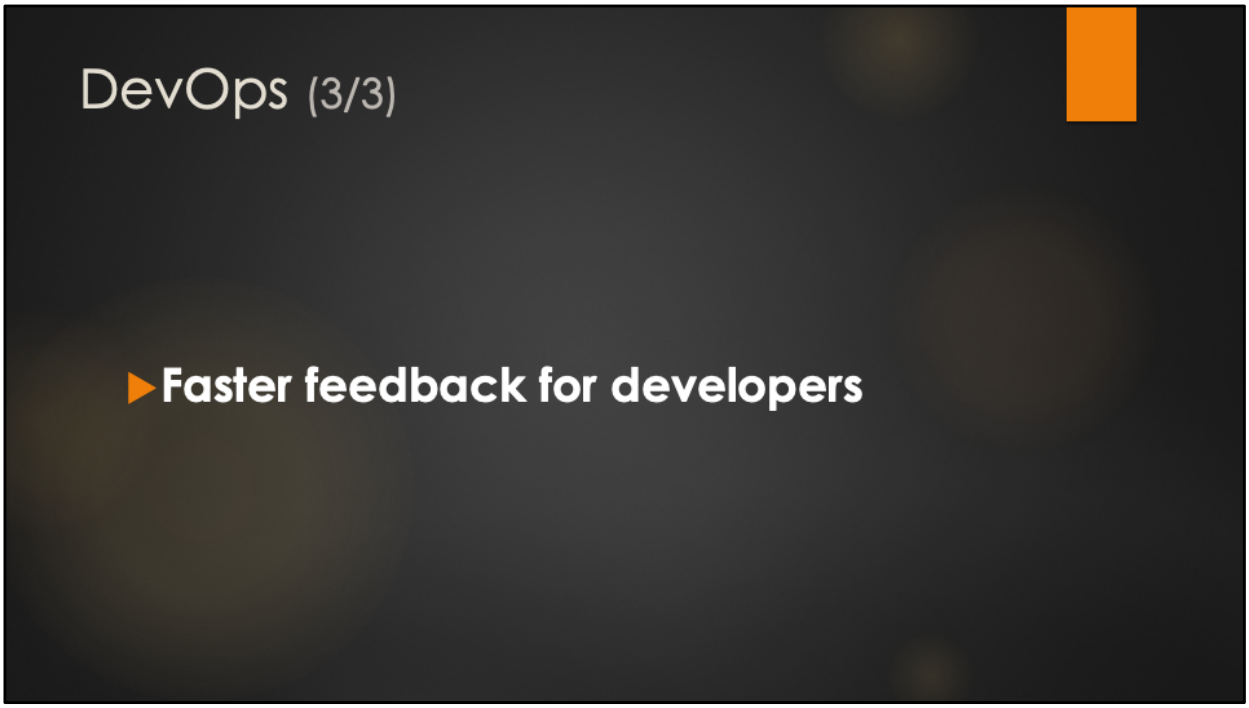


We've even seen an increase in automated security testing being integrated into the pipeline. As organizations read the data breach reports over the past few years and see that vulnerable 3rd party dependencies have been the root cause of several large breaches, like Equifax, SCA integrations have become quite common. Some of the newer IAST tools have been returning really good results to the development organization. We've also been seeing a rise in automated DAST being integrated into the pipeline – I'll touch more on this in a few minutes. When it comes to SAST – pipeline integrations are usually done by tuning the rules used to really focus on high confidence higher risk issues and presenting these results directly to the developers.

All of this allows the developers to own the security of their application. I mentioned my buddy Steve a few minutes ago - his team even asked for SAST IDE plugins so they could check their code before even making a commit – as security continues on our journey left we are seeing more and more developers embracing appsec they want the tools and knowledge to make their applications secure.

I'd also like to point out that in some cases – information security is still running full SAST scans with a much more complete ruleset – but this is done as an out-of-band process in the CI/CD pipeline. In those cases security should step-up and take on the

onus of false positive dispositioning – and deliver actionable results to the application teams using the teams standard defect trackers like Jira.



Moving to DevOps has been an amazing transformation for many of us – shortening release cycles from months or years to days or hourly – or even faster for some of the pariahs out there. The real benefit of this increased automation and increased pace is Faster Feedback for developers. We no longer have longer cycles between writing code, including infrastructure as code, and getting feedback on how the changes affected the app. Or worse, we write code, 5 months pass and security comes banging on our door saying we have to fix an issue. This takes a huge context switch and as a developer this can be hugely time consuming.

DAST in the pipeline is definitely not a new topic. There are numerous talks and blog posts about integrating DAST tools into the pipeline. From having the pipeline stand up infrastructure and run full DAST scanners against the target to using an applications Selenium test suite to drive ZAP or Burp to add security testing to your existing testing.

Automated DAST in the pipeline is, in my book, an absolute must. This is going to catch a lot of low hanging fruit. This is going to allow you to further refine your SAST rules – no longer need to try and have a SAST rule to check for appropriate header configuration. You get to use the right tool for the job. I've seen some crazy SAST rules trying to detect and understand cookie settings and security headers that might be set in code or via configuration – but many of these things might actually be being set at the F5. We might be creating FP with our SAST rules. If we can stand up the infrastructure, run some simple DAST checks – we can throw out all of those SAST rules because we have it covered.

The problem – false positives. When we setup these integrations again, we are generally tuning what the tool is actually looking for if we are feeding the results back to the development team at a DevOps pace. If we create too much noise for the

development organization – they will find a way around you. We have to be part of the solution to keep things going fast.

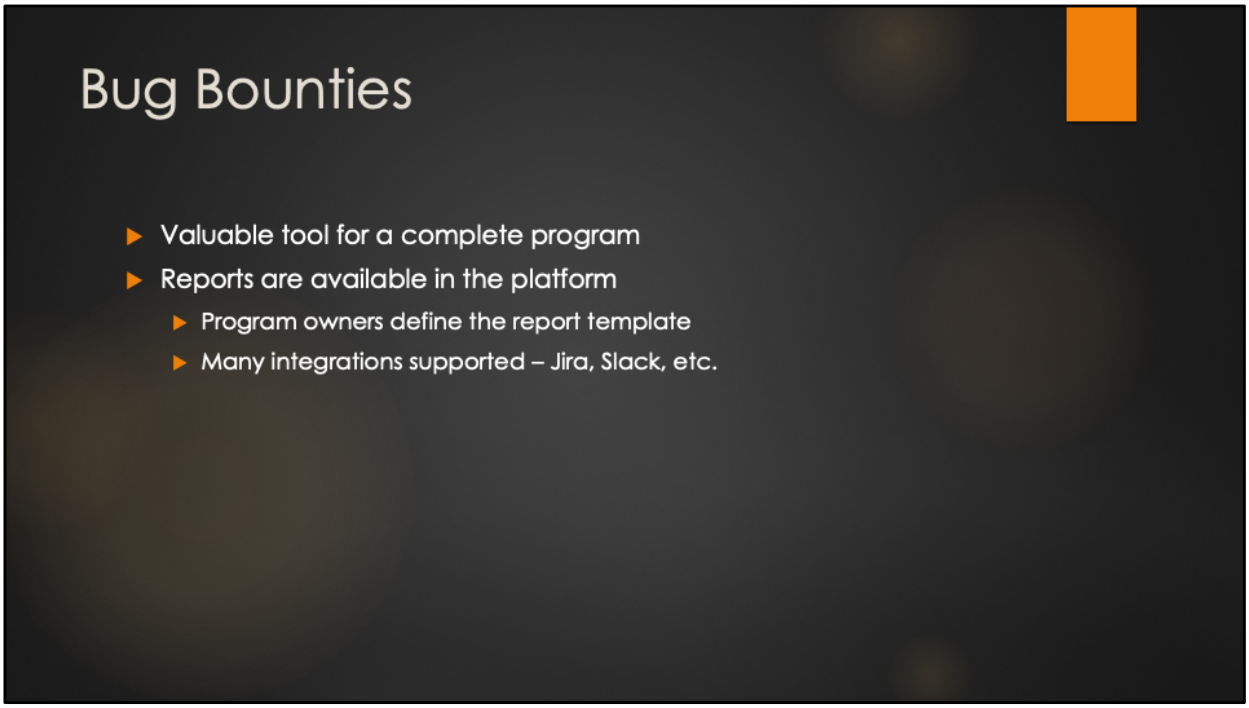
Manual testing must be incorporated into an application security program to ensure completeness. Manual testers have some of the most beautiful, devious minds, that can poke some really non-obvious holes into an application and show where all of the previous work was inadequate…. That just nauseating feeling as you stare gobsmacked at that P1 bug that was just reported.

Unfortunately, manual testing is not DevOps friendly.

But its not all doom and gloom - What advances has DevOps given our manual testing efforts?

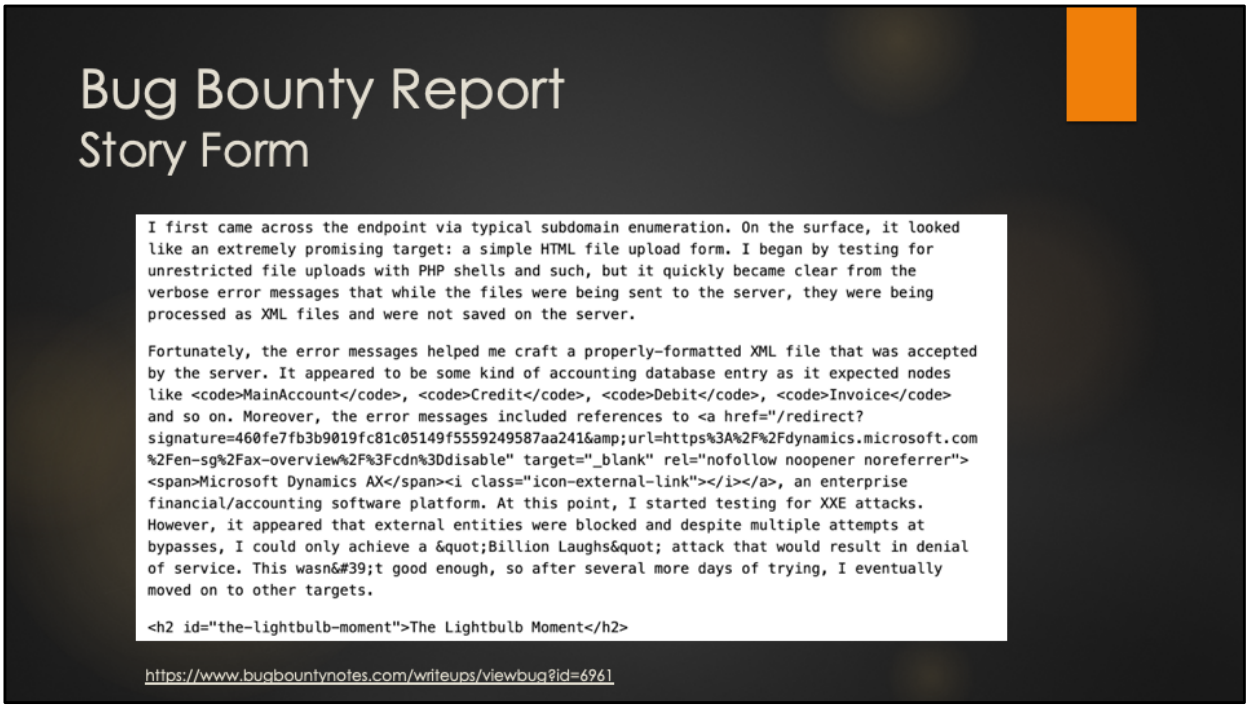
- Automated provisioning of test environments; while we absolutely should be testing in production, testing in lower environments we can have a higher confidence that the infrastructure and deployments are identical to production because we have clear repeatable automation creating these environments.
- Automated provisioning of credentials – for some organizations this is huge. I was working with Aaron, one of those really amazing testers that came from the development side of the house and as such he really deeply understands where developers may hide their dirty crusty bugs, but I digress, we had everything setup

and in place – the infrastructure and application was setup and ready for testing… But somehow the application team missed a step and never provisioned the credentials necessary for testing…. How many people here have ever walked into an engagement to find some critical requirement missing? DevOps can help…



I need to touch on bug bounty programs. This is just another way to engage a group of highly skilled testers to take a different look at your application. These are valuable tools to create a complete app sec program. The reports are available on the platform of choice, program owners can define the report template, you have the assistance of the bug bounty firm to assist in triaging the reports. You even several integration options available from Slack to Jira to complete custom access to the reports via APIs. These can be really great programs giving concise valuable reports.

Sometimes the reports are like cooking blogs [click]



I really don't want to belittle this report – as it was a really great find. However, the first three paragraphs of the report talk about his methodology, having found something interesting – but couldn't get a high enough pay-out with just insecure error reporting allowing him to figure out how a DoS via billion laughs – so he moved on. A month later he had the "lightbulb moment". Just like a cooking blog where we have to read 3 pages covering their found memories visiting the heritage home in a pine forest in Pennsylvania, when they would arrive their grandmother would have the most delicious smelling brownies for them when they arrived; you just can't beat smell… no the taste of that moist delectable brownies. But in reality – I'm standing in the kitchen with my ingredients and I just want the recipe so I know how to make the brownies.

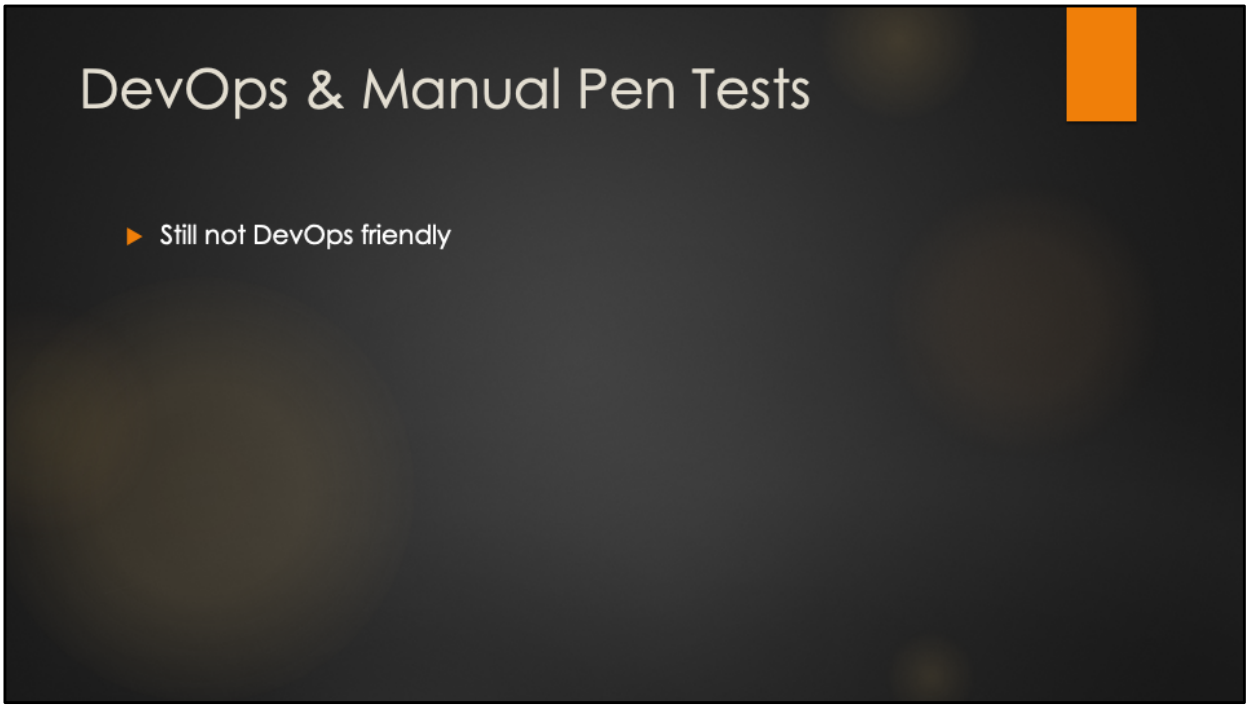
Of course I'm exaggerating – but how does knowing the process of discovery really aide me in remediating a vulnerability. In fact, this type of story form may make me miss points about the vulnerability reported. In this case the tester went on to describe how he figured out how to perform SQL Injection and extract the companies payroll information. I'm going to focus on the SQL Injection, but this report also mentioned insecure error reporting and XEE. The story form of this report should absolutely be written up in a blog after the vulnerability has been resolved – stories

like these help others learn how they might handle a scenario like this.

As with all types of testing – there are definitely varying quality of reports, and I don't' want to disparage the skill level of those that participate in bug bounties, but sometimes the reports leave a little to be desired.

How many people when the do an assessment are still delivering a PDF report? This didn't happen at my company – but I was talking with Susan, a newcomer to the InfoSec community, when she started out she was so excited to have gotten an internship to jumpstart her career.. She spent the summer copying and pasting from PDF documents into Jira. So that was a disappointing internship – at least it was paid.

Other companies I know, when engaging outside testing firms, require that a machine readable report be delivered. Most of the consulting companies use tools to collect assessment results and so generating a machine readable report is a trivial task. What I find interesting is that I haven't come across an industry standard for a machine readable pen test report.
The current state of the art is feeding results directly into defect trackers – this works well for internal testers, and is the reason we are asking for machine readable reports.



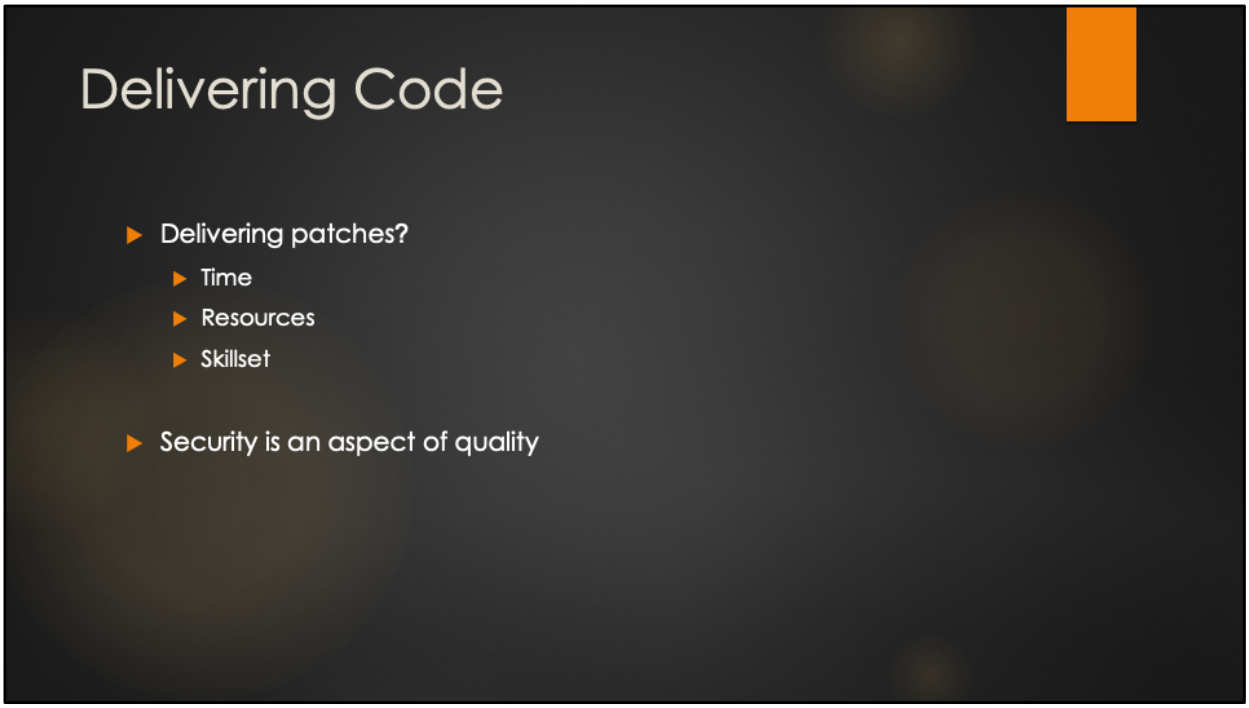
With the myriad of changes that have occurred with DevOps, we have increasing speed, our automated pipelines are able to get feedback to the developers faster so defects can be resolved earlier in the lifecycle. We have instrumentation in our apps so we can monitor how they are performing – and our development teams use this information to guide the ever increasing rate of change.

Unfortunately, I can't help with improving the speed that we can perform manual tests, we just have the improved, consistent environments that can make testing easier in many DevOps shops. However, we might be able to do something about the reports.

Let's talk about our current best practice.
Is integrating into with the development teams defect tracker that helpful? In the sense that we now have a trackable, assignable unit of work – absolutely. I know my friend Susan loves defect tracker integrations… But has this done anything to aid developers in understanding the identified issue or being able to understand a complicated scenario or payload? Not really..

I've worked with some development teams claiming they believe they fixed the issue, but they do not have access to Burp to be able to retest (organizational and training issue as opposed to a technical limitation) – the report explicitly called out using features in Burp to reproduce the issue. With other security defects the developers just did not fully understand the issue based on the report. Both of these issues might be solved by better technical writing and QA within the DAST team.

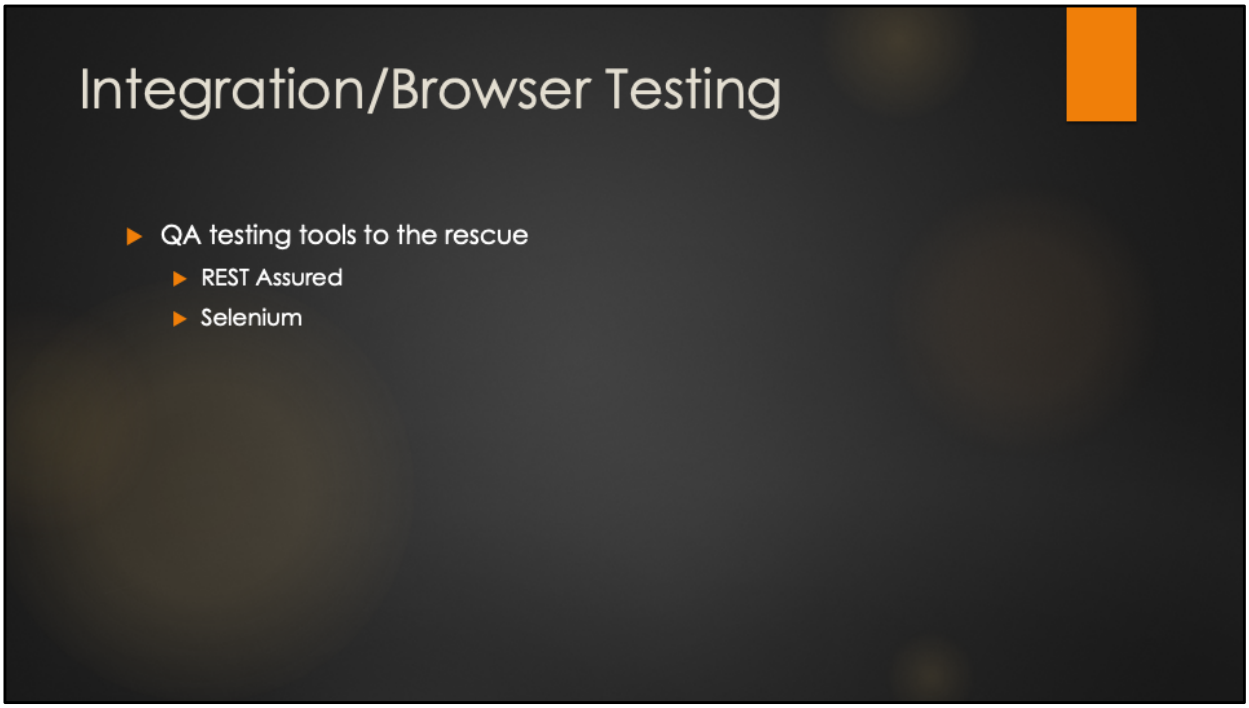
Alternatively, we could begin speaking the language of developers. When we were producing PDFs people told us we needed to use the same tools as the developers to communicate bugs – so we started pushing findings into the defect trackers. If we really wanted to speak the language of developers and drive remediation – we would be delivering code.



When I've talked to individuals about this topic – you'll see some tester's eye's bulge out. In most cases the DAST testing team is extremely time-boxed. We barely have enough time to find the defects and write a coherent report. Now you are suggesting we write the fix too?

In some orgs – this is the case, you find it – you fix it. If you are lucky enough to be in an org like that – I'm envious. But in reality – this is not an opportunity for most of us. Be it time, resources, or even skill set – not all of the DAST team has a developer background.

But what is security really?
Isn't it just an aspect of quality? We already know that with DevOps and agile development we have seen a huge emphasis on testing.

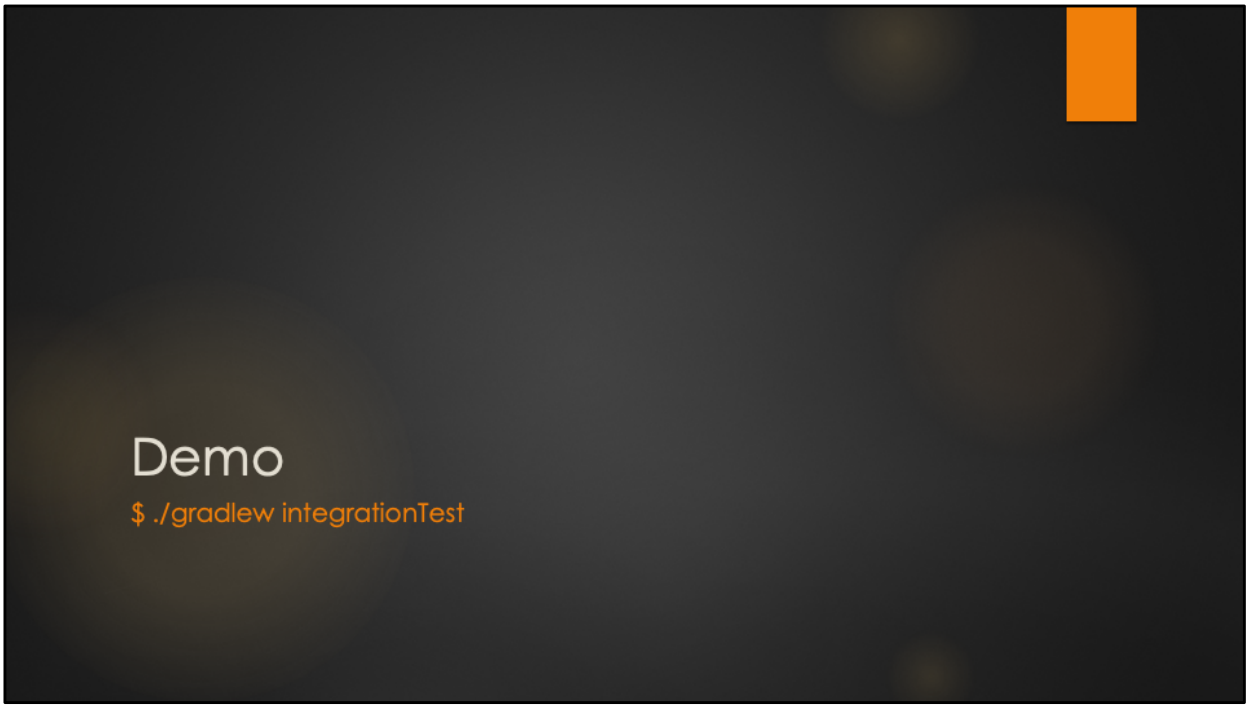


Many of our development teams are using tools like Selenium or REST Assured (there are others). If you are not familiar with these types of testing tools I'm going to give a brief overview. I'm covering two tools because we still see a mix of traditional web applications and single page applications – depending on the scenario one or the other may be more appropriate for testing.

REST Assured is a framework for testing RESTful services. It integrates nicely into other testing frameworks like jUnit or testNG. You can perform a GET, POST, PUT, etc. with given parameters, get the results back, and perform assertions against the results. For instance– if I put a single tic into the last name parameter I do not expect to see a detailed error message containing a SQL Query with invalid syntax.

Selenium – automates web browsers and is most used to perform automated testing of applications. Teams using Selenium likely have invested a lot of time and energy into this tool – it is truly an amazing tool that has advanced over the years to allow an unbelievable level of testing against your running application. Basically, you can script any interaction a user would have with the UI of the web application – and for our purposes the user is malicious! Now when you first look at Selenium you will see that we are constrained to what is available in the UI in terms of finding fields, buttons,

links, etc. Yes, this means JavaScript will likely be running and performing any client side validation that the developer has put in place. Luckily, using Selenium you can inject, just like a normal user with the developer tools, any JavaScript you want. You can even hook up a proxy, in code, and manipulate the request or response to your hearts nefarious content.


Demo
$ ./gradlew integrationTest

REST Assured Example

```
@Test
@Tag("security")
@Tag("integrationTest")
public void doNotReturnDetailedErrorMessages() {
    JSONObject requestParams = new JSONObject();
    requestParams.put("email", "'");
    requestParams.put("password", "AnyPa$$wordWillD0");

    RestAssured
        .given()
        .contentType(ContentType.JSON)
        .accept(ContentType.JSON)
        .body(requestParams.toString())
        .when()
        .post("/rest/user/login")
        .then()
        .statusCode(500)
        .body("error.original.code", not(containsString("SQLITE_ERROR")))
        .body("error.sql", not(containsString("SELECT ")))
        .body("error.parent.sql", not(containsString("SELECT ")))
        .body("error.original.sql", not(containsString("SELECT ")));
```

Selenium Example

```
WebDriver driver = get("http://localhost:3000/");
WebDriverWait wait = new WebDriverWait(driver, 5);

driver.findElement(By.id("userMenuButton")).click();
WebElement element = wait.until(
        presenceOfElementLocated(ByHelper.buttonText("Track Orders")));
element.click();

element = wait.until(presenceOfElementLocated(By.id("orderId")));
element.sendKeys("<iframe src=\"javascript:alert(`xss`)\">");

driver.findElement(By.id("trackButton")).click();

Alert alert = wait.until(alertIsPresent());
assertThat("XSS payload should be neutralized",
    alert, nullValue());
```

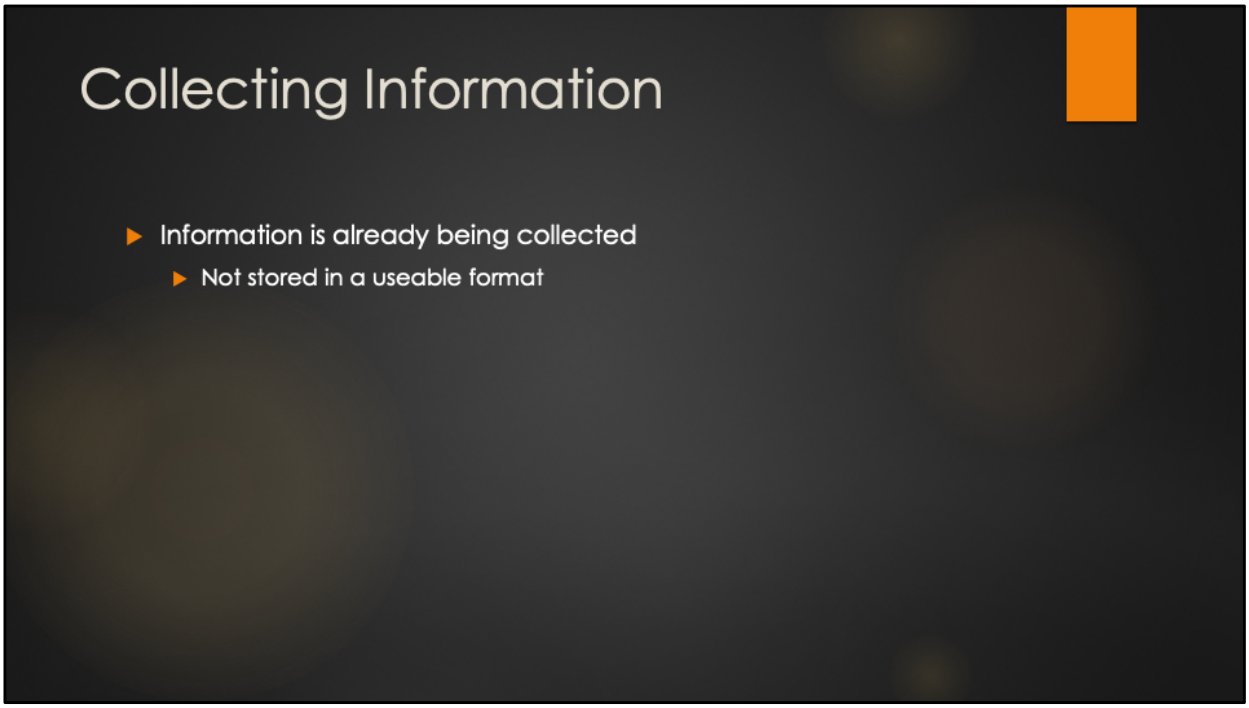
- Mention the Selnium IDE
- Discuss implicit wait vs explicit wait (WebDriverWait)
- Not a great example of a templatized test case – but useful to understand the capabilities

Selenium w/ Proxy Example

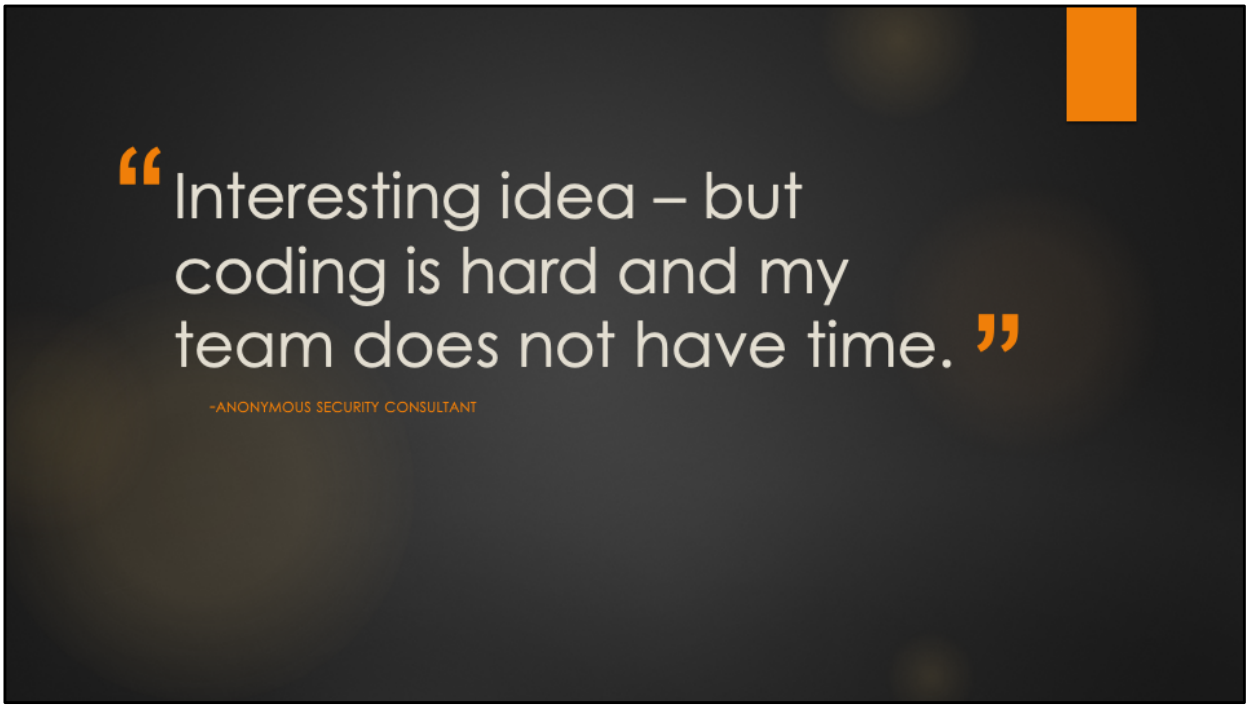
```
protected BrowserMobProxy getProxyServer() {
    BrowserMobProxy mobProxy = new BrowserMobProxyServer();
    // trust applications with invalid certificates
    mobProxy.setTrustAllServers(true);
    mobProxy.start();
    return mobProxy;
}
```

```
getProxy().addRequestFilter((request, contents, messageInfo) -> {
    if (request.getMethod() == HttpMethod.POST
            && messageInfo.getUrl().contains(("login"))) {
        String original = contents.getTextContents();
        String updated = original.replace("name=root", "name=root' OR '1'='1");
        contents.setTextContents(updated);
    }
    return null;
});
```

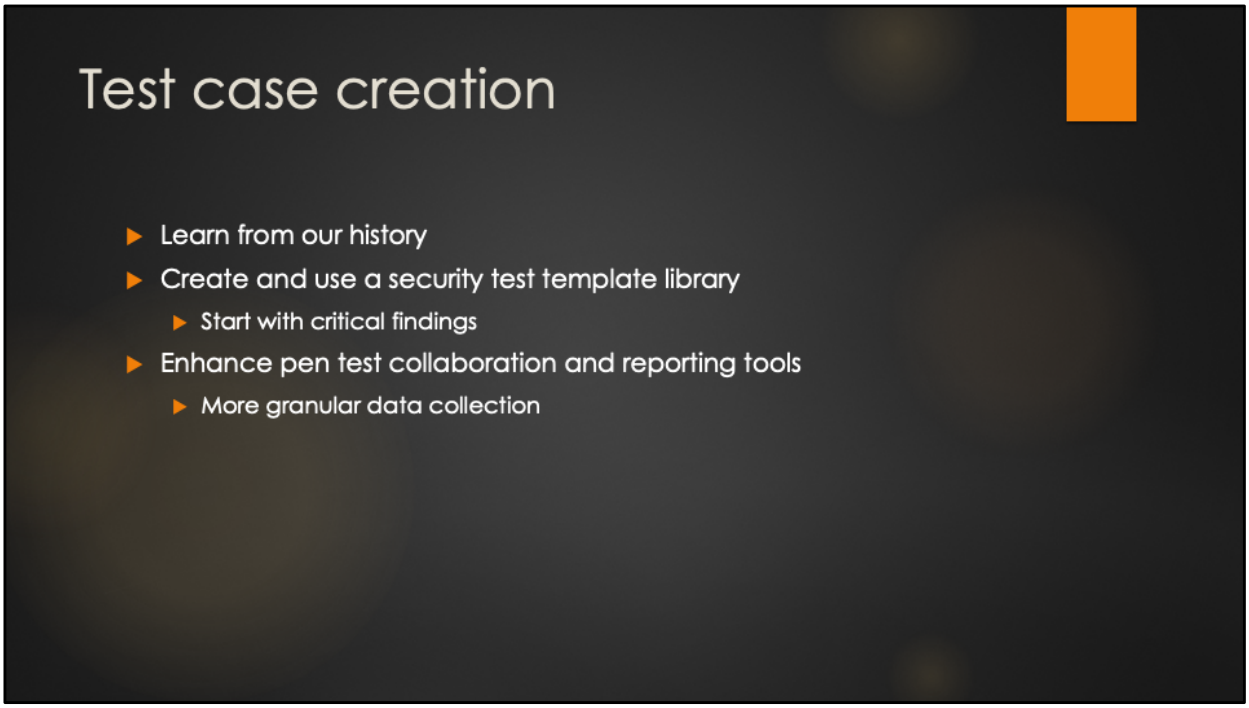
Sometimes you need to use a proxy – here we have an example where we might need to bypass client side input validation. We are using BrowserMob Proxy.



The title of the talk is testing with your left foot forward – by thinking about test case creation, shifting our findings left, when we are capturing the data for the report – if we are more precise in how we capture the data we would be able to automate the test case creation. For the most part – we are already capturing a majority of the data we would need; it is just in the write-up as opposed to stored as fields in a database. We'll talk more about this later. Let's see what this looks like.

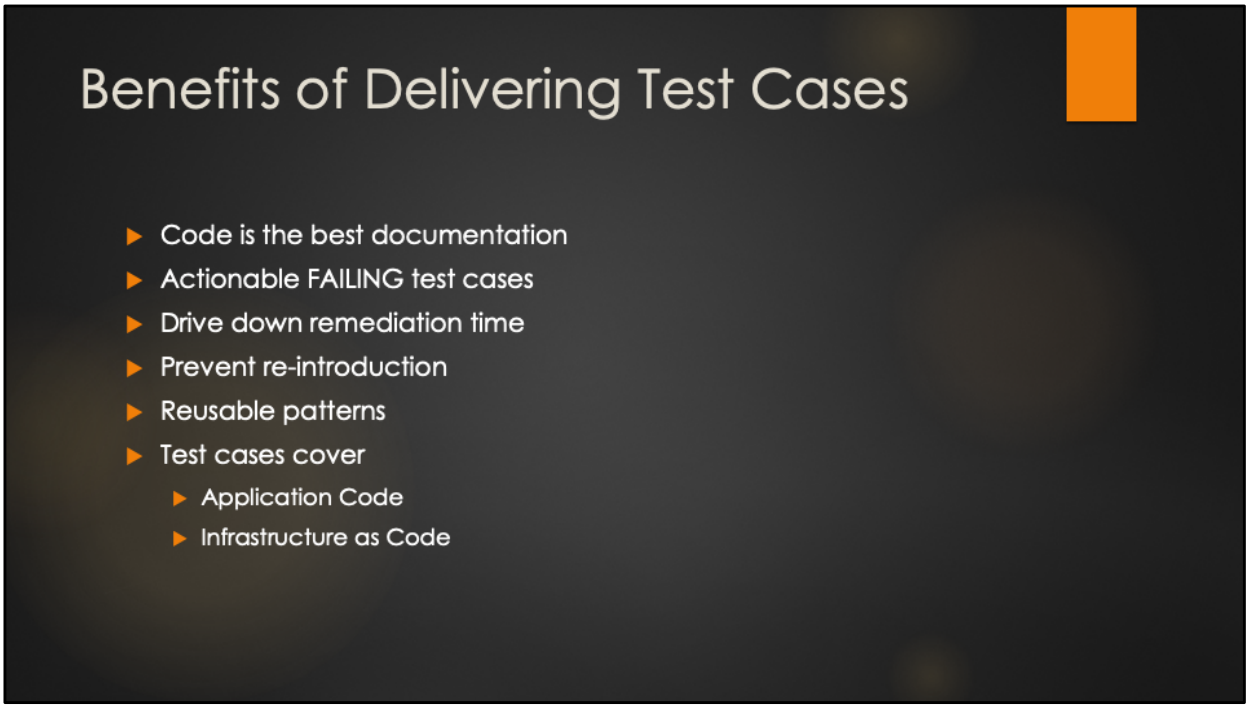


When I’ve discussed the idea about delivering test cases as part of a standard engagement one of the first things I heard was “Interesting idea – but coding is hard and my team doesn’t have time.” Now I’m not going to call the guy out that said this to me and I am not going to get into the political debate as to whether or not AppSec professionals need to be able to code – whatever, we have some very talented folks that came from the networking side of the house. But is the type of coding we are discussing here actually hard? Not in the slightest.



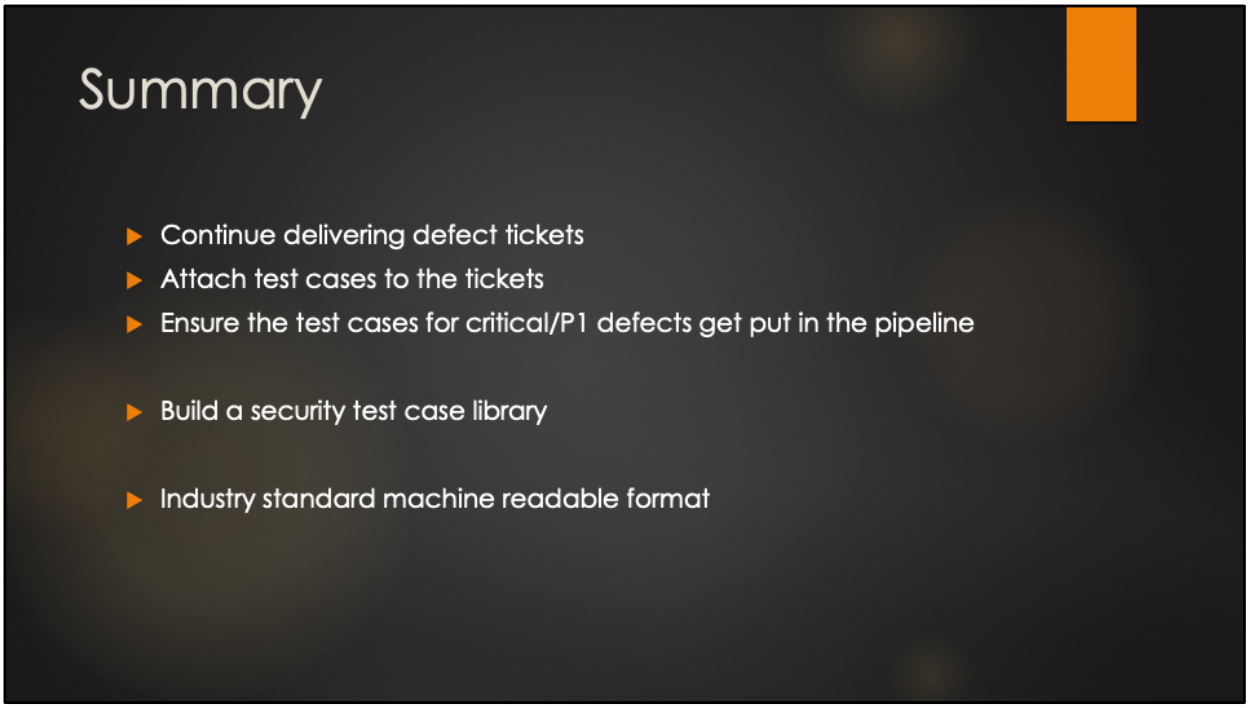
When I first started doing application assessments full time, circa 2005, one of the hardest things was writing the report. If you found an issue in an application you had to go find a previous report that cited something similar copy, paste, and tweak to meet the specific nuances of what you found this time. Not very scalable, it was a pain in the ass – every report was a unique and beautiful snow flake. I hate doing things manually – thus we ended up templatizing our write-ups and built automation around report writing. This was a fairly standard evolution of reporting across the industry. We can learn from our history and do the exact same thing here. We can use a security test case template library to aide in the creation of test cases we can delivery as part of an assessment.

Another opportunity is our pen test collaboration and reporting tools – some of these have nearly everything we need to generate a test case. However, the information may not currently be captured in a way that we can use the test case templates to automatically generate the test case. This is going to be an area of research and development.

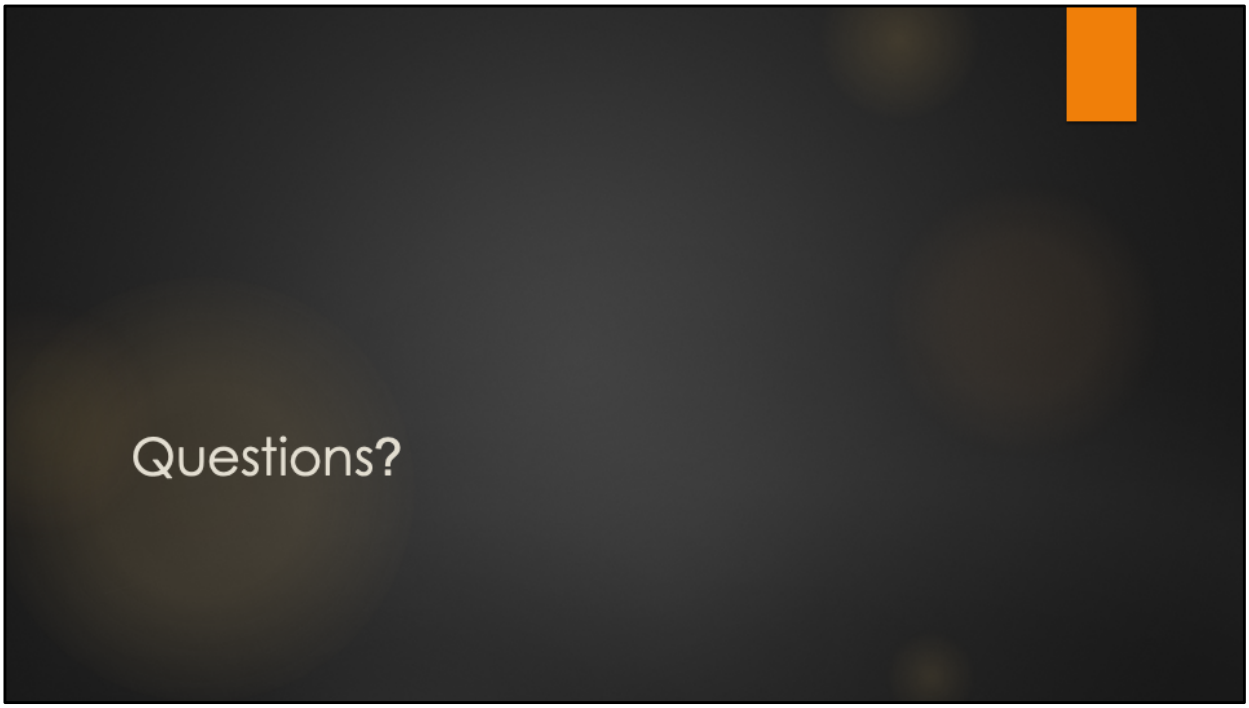


What are the benefits of delivering test cases…

For most developers the BEST documentation is code.



We have to keep delivering defect tickets (or machine readable reports so we can automate the generation of jira tickets).


Questions?