

Application Security Testing

How to find software vulnerabilities before you ship
or procure code

Anita D'Amico, Ph.D.

Hassan Radwan



Overview

- Why Care About Application Security?
- Quality vs Security
- Application Security Techniques
 - Manual Code Reviews
 - Static Application Security Testing
 - Dynamic Application Security Testing
- How to Incorporate Application Security into Your Organization

DHS goal: *Secure the software supply chain*

DHS funded **Secure Decisions** to develop innovative technologies to improve and expand security testing of software applications.

Led to new open source and commercial solutions; formation of new company Code Dx, Inc.

Co-organizer for OWASP Long Island chapter

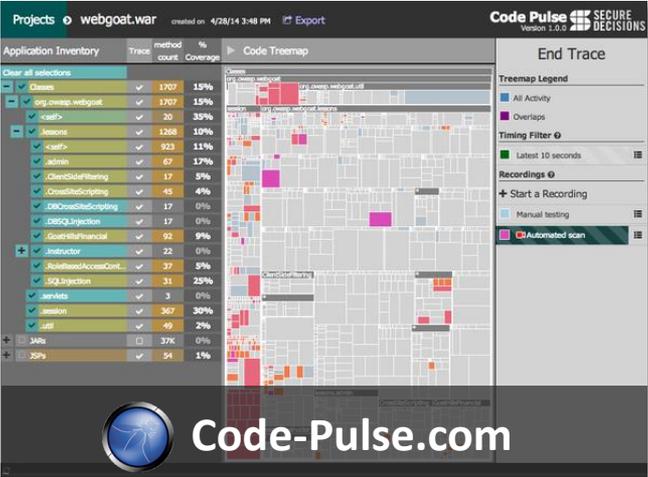


Based in
Northport &
Clifton Park,
NY



See <http://secoredecisions.com> for more cybersecurity innovations.

Developers of freely available community resources



<https://continuousassurance.org>

Why Care About Application Security?

Data Breaches Impact



at&t



350 million accounts impacted



JPMORGAN CHASE & Co.



Annual global cost of cybercrime is > \$400 b

Net Losses: Estimating the Global Cost of Cybercrime, Center for Strategic and International Studies, June 2014

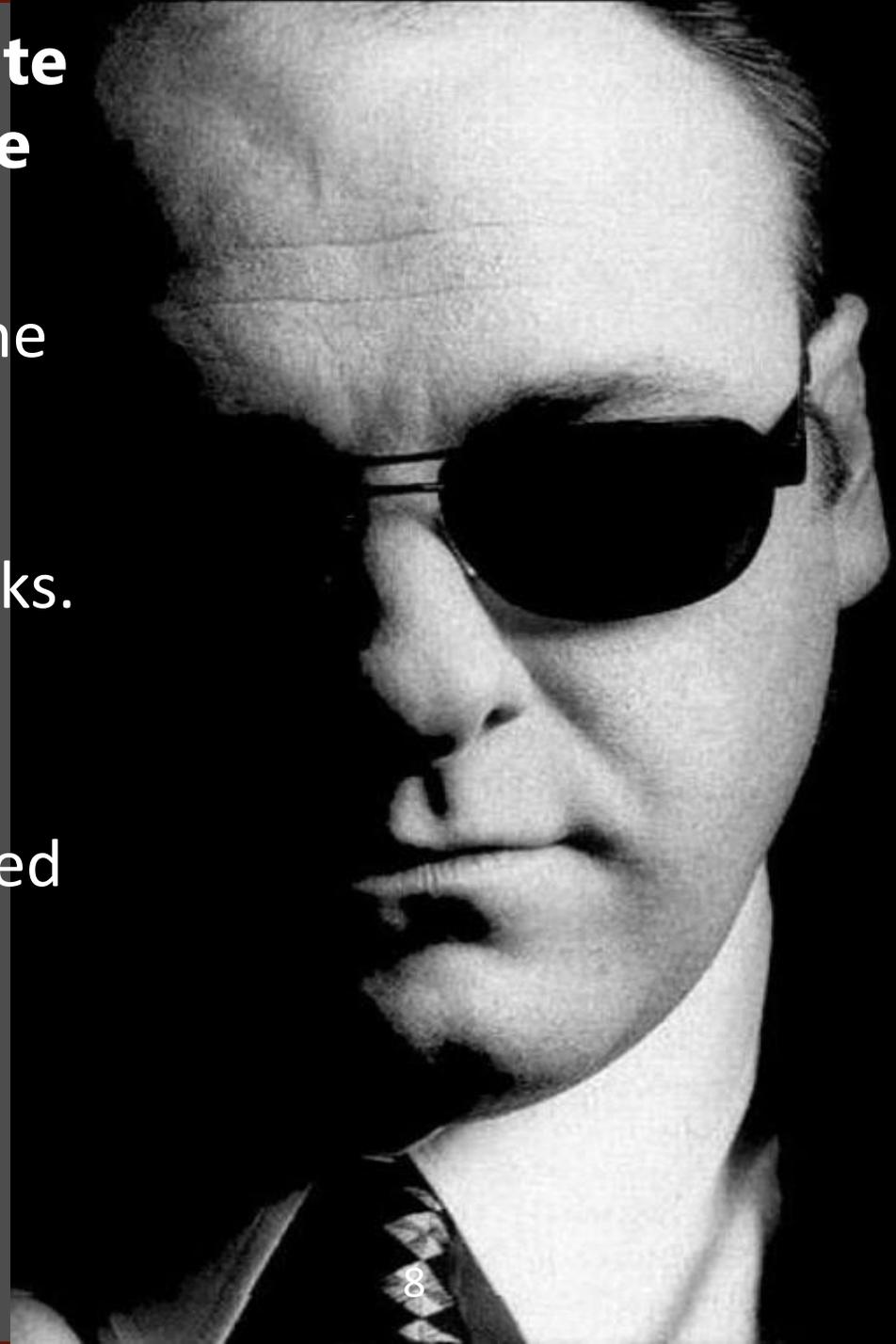


Web App Attacks: A favorite method of organized crime

“ This year, organized crime became the most frequently seen threat actor for Web App Attacks.

Verizon Data Breach Report 2015

Web Applications are used to perpetrate **31% of breaches into Financial Services**



Software flaws are at root of most cyber incidents

“ 90% of security incidents result from exploits against defects in software

Build Security In Website, DHS
<https://buildsecurityin.us-cert.gov/bsi/mission.html>



Bug Bounties

- Google pays “white hat” hackers up to \$20k to find vulnerabilities in its Web browser, before attackers do
- Microsoft offers as much as \$150k



YOU'RE NOT AS SECURE AS YOU MAY THINK!

Quality vs Security

Example Quality and Security Issues

Example Quality Issues

Confusing code

Performance issues

Concurrency issues

Memory leaks

Null pointer

Redundant & dead code

Example Security Issues

SQL Injection

Cross-site scripting (XSS)

Cross-site request forgery (CSRF)

Buffer overflows

Using hard coded passwords

Sensitive data exposure

Quality is Security

Quality and security are closely intertwined

2011 Firefox study found that 82% of vulnerable source files were also faulty files

Software Engineering Institute in 2014 study concluded that:

- Over half of security vulnerabilities are also quality defects
- There is a direct correlation between the number of quality defects in a system and the number of security vulnerabilities



QUALITY CODE LEADS TO MORE SECURE CODE

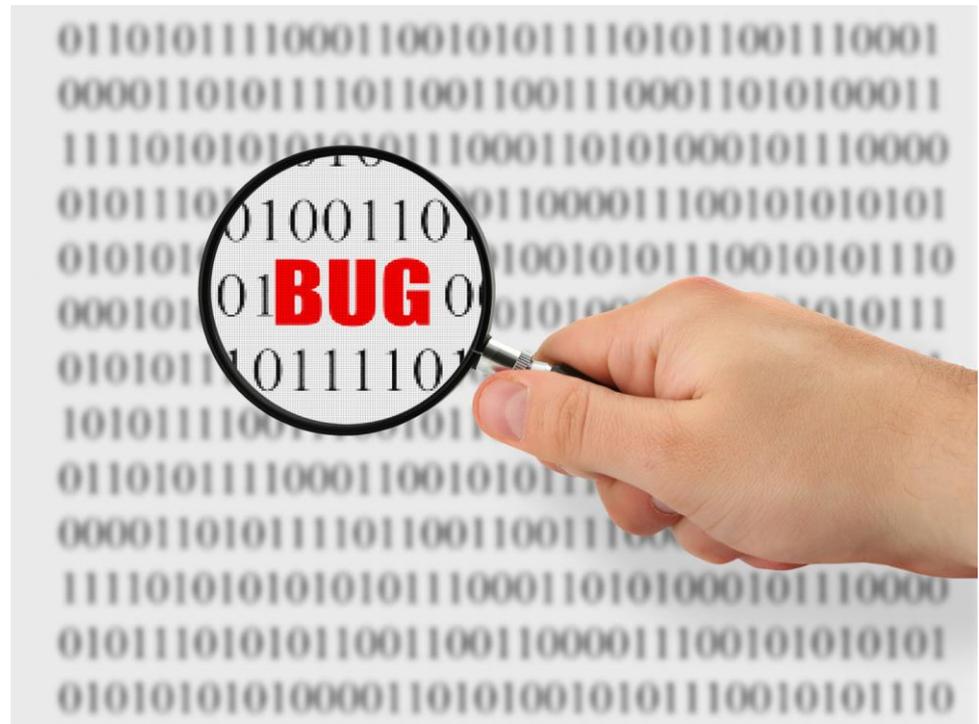
Application Security Testing Techniques

Application Security Testing Techniques

- Manual code reviews
- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)

Code Reviews

- Established practice for improved code quality
- Beneficial to both quality and security
- Useful to detect fundamental structural flaws
- Certain security issues can only be detected via manual reviews



Static Application Security Testing (SAST)

- Automated
- Statically scans source or binary files
- Detect potential vulnerabilities
- Potential vulnerabilities need manual verification
- Many SAST tools available, open source & commercial
- Dependency checking for known vulnerabilities



SAST Workflow

1. Input selection, source and/or binary files
2. SAST tools scan input files
3. SAST tools present list of potential vulnerabilities
4. Manual potential vulnerability triage
5. Remediation of confirmed vulnerabilities

WebGoat » Analysis Run 1

Created on 5/28/2015 Uploaded on 5/28/2015 1,332 total weaknesses View ▾

Weakness Flow

Filters clear all

Weakness count 254 / 1,332

Tool

- Checkstyle (0%)
- Dependency-Check (4.3%)
- FindBugs (2.4%)
- JSHint (0%)
- PMD (92.5%)
- Retire.js (0.8%)

Severity clear filter

- Info (0%)
- Low (0%)
- Medium (0%)
- High (100%)

Codebase Location

Tool Overlaps

CWE

Status

- New (100%)

Displaying weaknesses with a **High** severity

Bulk Operations for the 254 matching weaknesses

Change status... Generate report

Id	Tool	Rule	CWE	Codebase Location
1246	FindBugs	Empty database password	259	DatabaseUtilities.java:112
1073	FindBugs	Method ignores return value	252	SoapRequest.java:147
739	FindBugs	Method ignores return value	252	CommandInjection.java:180
678	FindBugs	Method ignores return value	252	BlindScript.java:230
613	FindBugs	Call to static DateFormat	662	HammerHead.java:248
608	FindBugs	Method call passes null for nonnull par...	476	WebSession.java:245-246
602	Dependency-...	CVE-2015-0254	-	standard-1.1.2.jar
601	Dependency-...	CVE-2013-6429	264	spring-core-3.2.4.RELEASE.jar
600	Dependency-...	CVE-2014-0054	352	spring-core-3.2.4.RELEASE.jar
599	Dependency-...	CVE-2014-1904	79	spring-core-3.2.4.RELEASE.jar
598	Dependency-...	CVE-2007-6059	399	mailapi-1.4.2.jar
597	Dependency-...	CVE-2007-6059	399	mail-1.4.2.jar
596	Dependency-...	CVE-2015-0254	-	jstl-1.2.jar
595	Dependency-...	CVE-2013-0248	264	commons-fileupload-1.2.2.jar
594	Dependency-...	CVE-2014-0050	264	commons-fileupload-1.2.2.jar
593	Dependency-...	CVE-2014-5701	20	axis-1.2.jar
592	Dependency-...	CVE-2014-3596	-	axis-1.2.jar
380	PMD	For-in loop variable not explicitly scoped	398	wysihtml5-0.3.0.js:9321

Typical SAST result listing with filtering workflow

WebGoat > Analysis Run 1 > Weakness 1111

Nonconstant string passed to execute method on an SQL statement detected by **FindBugs** [SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE]
First seen on **5/28/2015** 8 weaknesses in this file 24 similar weaknesses in this analysis run **Medium** severity
CWE **89** - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') [CWEVis | MITRE]

Status

New

Activity Stream

Write comments with Markdown

Status set to **New** during Analysis Run 1 by **admin**
4 days ago

The weakness occurs in **WebGoat-6.0.1-stripped.war/WEB-INF/classes/org/owasp/webgoat/lessons/SqlStringInjection.java** on line **101**

```
90     Connection connection = DatabaseUtilities.getConnection(s);
91
92     ec.addElement(makeAccountLine(s));
93
94     String query = "SELECT * FROM user_data WHERE last_name = '" + accountName + "'";
95     ec.addElement(new PRE(query));
96
97     try
98     {
99         Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
100                                                         ResultSet.CONCUR_READ_ONLY);
101         ResultSet results = statement.executeQuery(query);
102
103         if ((results != null) && (results.first() == true))
104         {
105             ResultSetMetaData resultsMetaData = results.getMetaData();
106             ec.addElement(DatabaseUtilities.writeTable(results, resultsMetaData));
107             results.last();
108
109             // If they get back more than one user they succeeded
110             if (results.getRow() >= 6)
111             {
112                 makeSuccess(s);
113                 getLessonTracker(s).setStage(2);
114
115                 StringBuffer msg = new StringBuffer();
116
117                 msg.append(WebGoatI18N.get("StringSqlInjectionSecondStage"));
118
119                 s.setMessage(msg.toString());
120             }
121         }
122         else
123         {
124             ec.addElement(WebGoatI18N.get("NoResultsMatched"));
125         }
126     } catch (SQLException sqlE)
127     {
128         ec.addElement(new P() { addElement(s, e.getMessage()); });
129         sqlE.printStackTrace();
130     }
131 } catch (Exception e)
132 {
133     s.setMessage(WebGoatI18N.get("ErrorGenerating") + this.getClass().getName());
134     e.printStackTrace();
135 }
```

Typical detailed view with source listing and remediation guidance

Dynamic Application Security Testing (DAST) – Application penetration testing

- Manual and/or Automated
- Dynamically scan application at runtime
- Attempt to penetrate an application by detecting and exploiting vulnerabilities
- Typically performed in the run up to releases
- Many DAST tools available, open source & commercial



DAST Workflow

1. Application staging
2. DAST tools manually/automatically tuned to identify the *attack surface*
3. DAST tools perform active probing for vulnerabilities
4. Identified vulnerabilities reported
5. Remediation of vulnerabilities

Burp Suite Professional

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts

Site map Scope

Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

Host	Method	URL	Params	Sta...	Length	MIME type	Title
http://0b7bd624bab7...	GET	/addressbook/32/	<input type="checkbox"/>	200	2765	HTML	Contacts
http://0b7bd624bab7...	POST	/addressbook/32/De...	<input checked="" type="checkbox"/>	200	3100	HTML	Contacts
http://0b7bd624bab7.mdseclabs.net/addressbook		/addressbook/32/De...	<input checked="" type="checkbox"/>	200	4914	HTML	Contacts
		/addressbook/32/De...	<input checked="" type="checkbox"/>	200	2835	HTML	Contacts
		/addressbook/32/De...	<input type="checkbox"/>	200	2765	HTML	Contacts

Context menu for `http://0b7bd624bab7.mdseclabs.net/addressbook`:

- Add to scope
- Spider this branch
- Actively scan this branch
- Passively scan this branch
- Engagement tools
- Compare site maps
- Expand branch
- Expand requested items
- Delete branch
- Copy URLs in this branch
- Copy links in this branch
- Save selected items
- Site map help

Request/Response details:

```

1.1
abs.net
Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101
...
application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
q=0.5
accept-encoding: gzip, deflate
Proxy-Connection: keep-alive
Referer: http://0b7bd624bab7.mdseclabs.net/labs/lab.ashx?lab=7

```

0 matches

Typical DAST scanning screen

The screenshot displays the Burp Suite Professional interface. The top menu bar includes 'Burp Intruder Repeater Window Help'. Below it are tabs for 'Target', 'Proxy', 'Spider', 'Scanner', 'Intruder', 'Repeater', 'Sequencer', 'Decoder', 'Comparer', 'Extender', 'Options', and 'Alerts'. The 'Scanner' tab is active, showing a tree view of scanned URLs on the left and a list of detected issues on the right. The selected issue, 'Cross-site scripting (reflected)', is shown in detail in the bottom right pane. This pane includes a red warning icon, the issue title, and a table of metadata: Issue (Cross-site scripting (reflected)), Severity (High), Confidence (Certain), Host (http://0b7bd624bab7.mdseclabs.net), and Path (/search/11/Default.aspx). Below the metadata is an 'Issue detail' section explaining that the SearchTerm parameter was echoed unmodified in the response.

Typical detailed view with remediation guidance

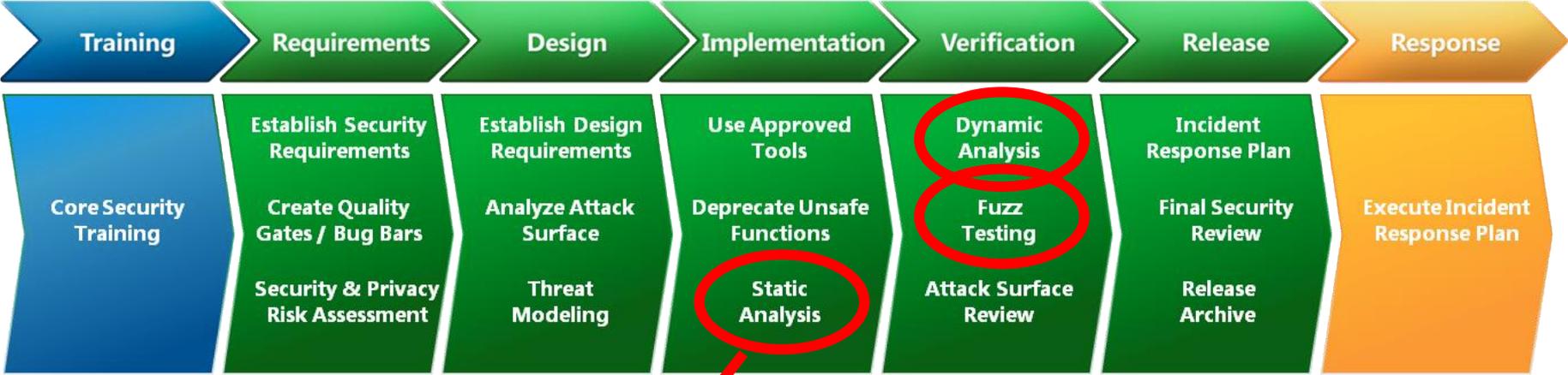
Issue detail

The value of the **SearchTerm** request parameter is copied into the HTML document as plain text between tags. The payload **1d329<script>alert(1)</script>27a3a1b60c71d9423** was submitted in the SearchTerm parameter. This input was echoed unmodified in the application's response.

How to Incorporate Application Security into Your Organization

Before you Test: SDLC

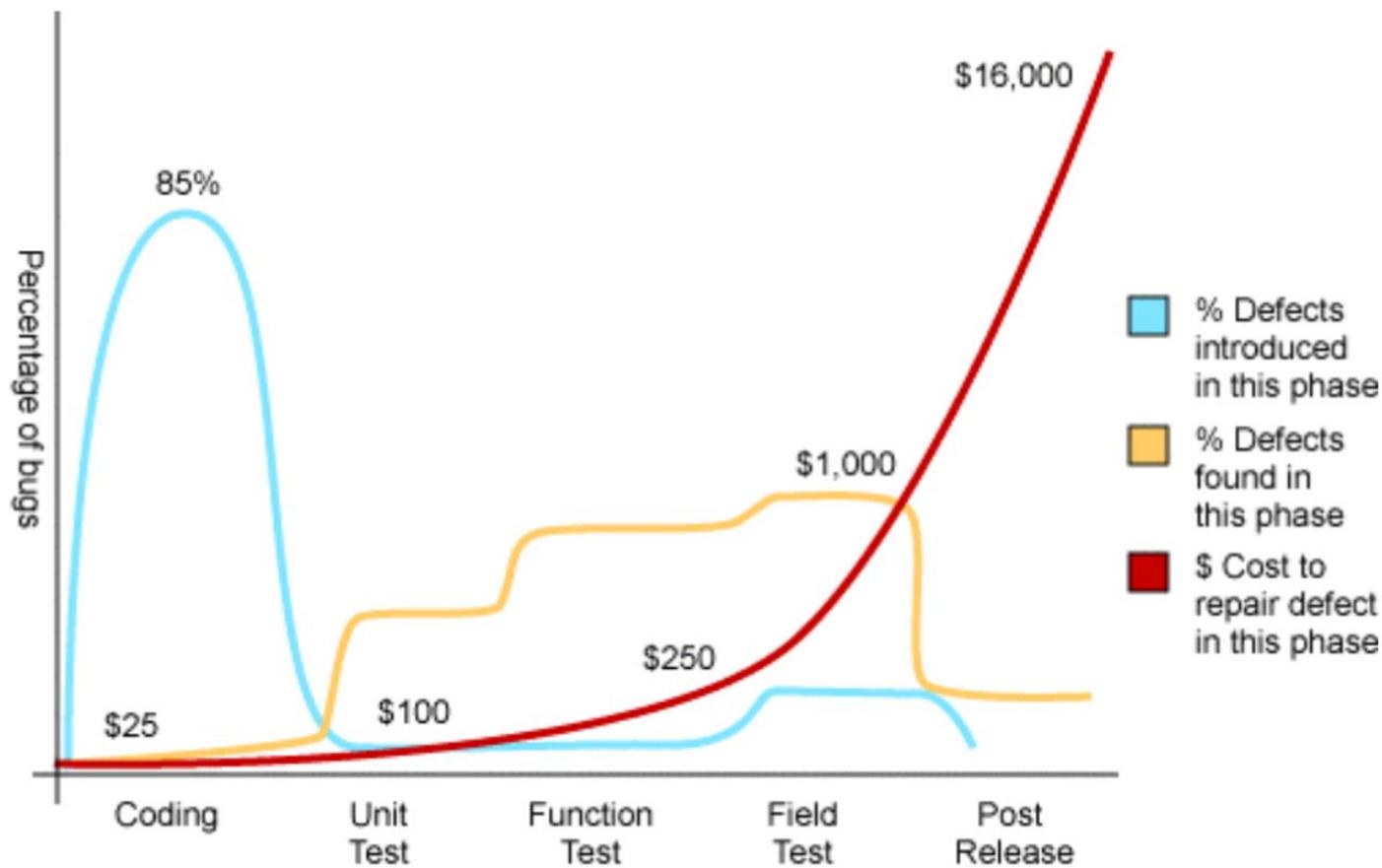
- Application security cannot be a final after-thought
- Incorporate security into all aspects of the SDLC
- Learn from the processes that are out there
 - OpenSAMM, BSIMM, Microsoft Security DLC



Source: Microsoft Security Development Lifecycle

SAST + Code Reviews

Test Early, Fix Early



Source: *Applied Software Measurement, Capers Jones, 1996*

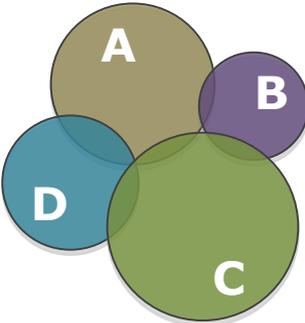
Before you Test: Educate

- Stakeholders and development teams need to see value in application security: *Pay now or pay more later*
- Integrate application security tools into the developers' IDEs
- Many resources available to offer guidance



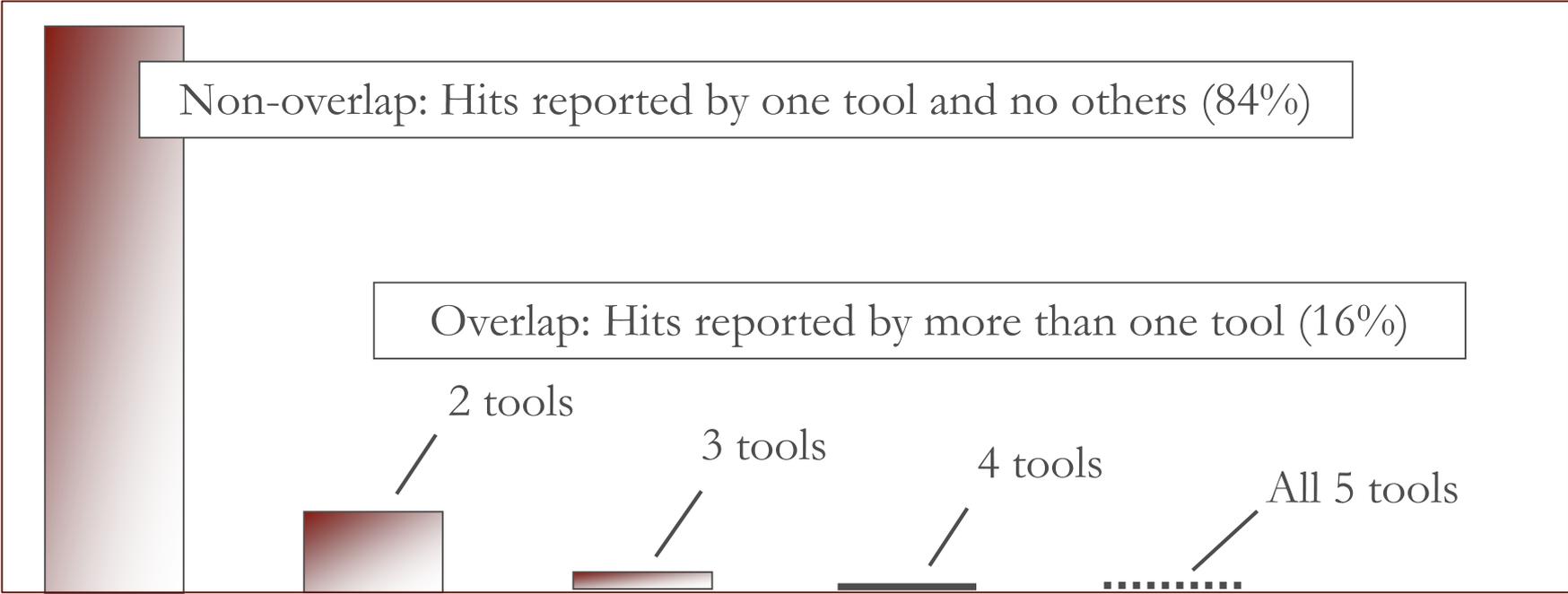
Best practice: Use multiple static analysis tools and combine results

Different tools identify different problems...



One tool on average detects 14% weaknesses

Kris Britton and Chuck Willis, “Sticking to the Facts: Scientific Study of Static Analysis Tools”, Sept 2011: <http://vimeo.com/32421617>

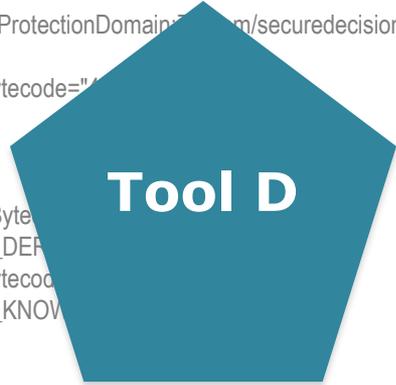
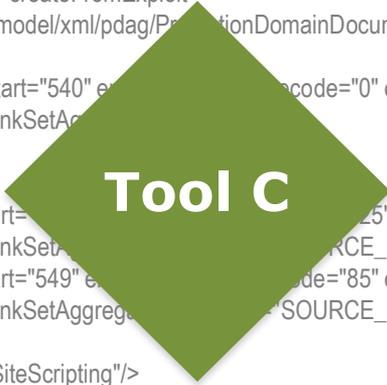
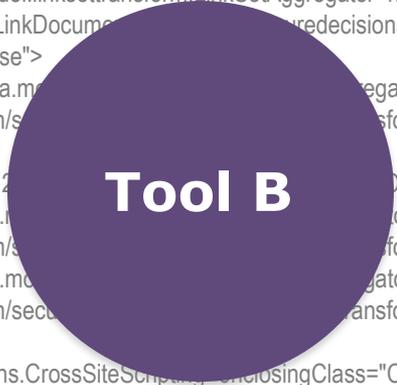


Paul E. Black, “Evaluating Static Analysis Tools”, 8 July 2009: http://samate.nist.gov/docs/eval_SA_tools_MIT_LL_July_2009.ppt

Use software vulnerability management system to combine/normalize multiple results

```
<BugInstance type="NP_NULL_ON_SOME_PATH" priority="1" abbrev="NP" category="CORRECTNESS">  
<Class classname="com.securedesigns.tva.model.linksettransform.LinkSetAggregator">  
  <SourceLine classname="com.securedesigns.tva.model.linksettransform.LinkSetAggregator" start="58" end="58" sourcepath="com/securedesigns/tva/model/linksettransform/LinkSetAggregator.java"/>  
</Class>  
<Method classname="com.securedesigns.tva.model.linksettransform.LinkSetAggregator" name="createFromExploit" signature="void" start="59" end="59" sourcepath="com/securedesigns/tva/model/linksettransform/LinkSetAggregator.java"/>  
<Location path="com/securedesigns/tva/model/linksettransform/LinkSetAggregator.java" start="59" end="59" sourcepath="com/securedesigns/tva/model/linksettransform/LinkSetAggregator.java"/>  
<SourceLine classname="com.securedesigns.tva.model.linksettransform.LinkSetAggregator" start="59" end="59" sourcepath="com/securedesigns/tva/model/linksettransform/LinkSetAggregator.java"/>  
</BugInstance>
```

FindBugs Output



```
<Vulnerability>  
<ClassInfo>  
  <ClassID>FE4EA...055-4C36-863E-5A01C4A0E1A4</ClassID>  
<Enclosure>...</Enclosure>  
<System>...</System>  
<Severity>...</Severity>  
</ClassInfo>  
<InstanceInfo>  
  <InstanceID>0010C1C949B6B1146790E9BA5186616D</InstanceID>  
  <InstanceSeverity>...</InstanceSeverity>  
  <Confidence>5.0</Confidence>  
</InstanceInfo>  
<AnalysisInfo>  
  <Unified>  
    <Context>  
      <Function name="handleRequest" namespace="org.owasp.webgoat.lessons.CrossSiteScripting" enclosingClass="CrossSiteScripting"/>  
      <FunctionDeclarationSourceLocation path="java/org/owasp/webgoat/lessons/CrossSiteScripting/CrossSiteScripting.java" line="234" lineEnd="296" colStart="2" colEnd="0"/>  
    </Context>  
  </Unified>  
</AnalysisInfo>
```

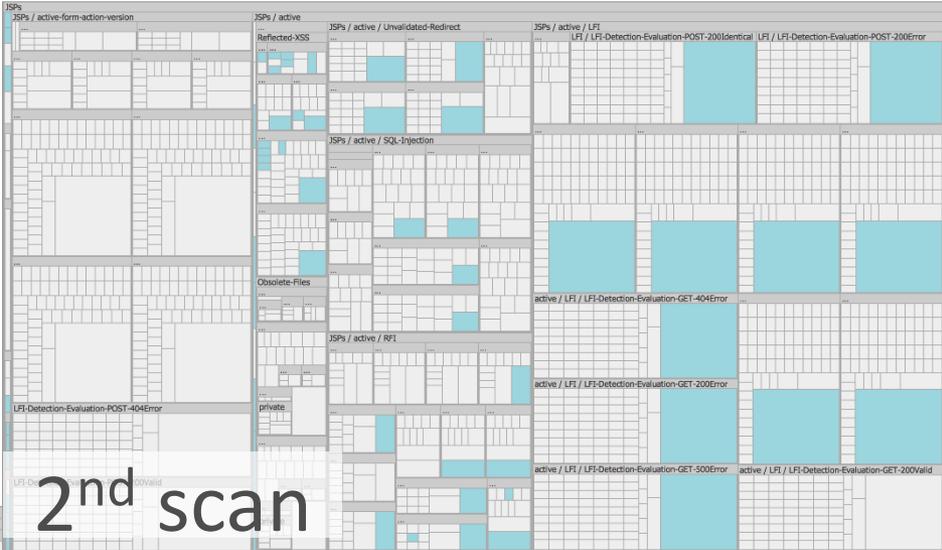
Fortify Output

working with different tool vendors is a challenging and time-consuming process: the engines work differently, which is good since they catch different types of problems...

Jim Bird, *Building Real Software*

http://swreflections.blogspot.com/2009_04_01_archive.html

Best practice: Tune DAST pen testing tools to maximize code coverage



Tuning DAST tools improves amount of code covered when penetration testing applications

Code coverage illustration prepared using OWASP Code Pulse



Recommendations

- Adopt all three techniques in limited doses initially
 - Manual
 - Static Application Security Testing (SAST)
 - Dynamic Application Security Testing (DAST)
- Use a vulnerability management system to combine and normalize results of different techniques
 - Examples: Code Dx, Thread Fix, Risk I/O
- Don't get overwhelmed, focus on a subset of the initial findings – Example: Filter weaknesses to focus only on OWASP Top Ten (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013)
- Integrate the tooling into your SDLC

 Anita D'Amico

 anita.damico@codedx.com

 @AnitaDamico

 Hassan Radwan

 hassan.radwan@codedx.com

 @leRadwan

