Code Dx

Getting Adoption for a Secure SDLC

or

"What Am I Supposed to Do with This?"

Curtis Bragdon
curtis.bragdon@codedx.com
617-312-1466

Agenda

- About Me and Code Dx
- Basics of Application Security
- When to Fix Issues
- Specific Advice for AppSec in DevOps
- Q&A

Point of this Talk

Share ideas on building Application Security into DevOps

You may disagree with parts of this. Maybe strongly.

This is meant to start a conversation

now, when you return to work, and later on

Key Premises:

- Central security teams own security for applications
- DevOps teams own the applications
- Developers can actually do something about security

→ Adoption at the engineering level is critical

My Background

10+ Years in AppSec: Consultant, Trainer, SE, Sales Rep

10+ Years in App Dev: Engineer, Consultant, Trainer, SE

Got into AppSec almost by accident

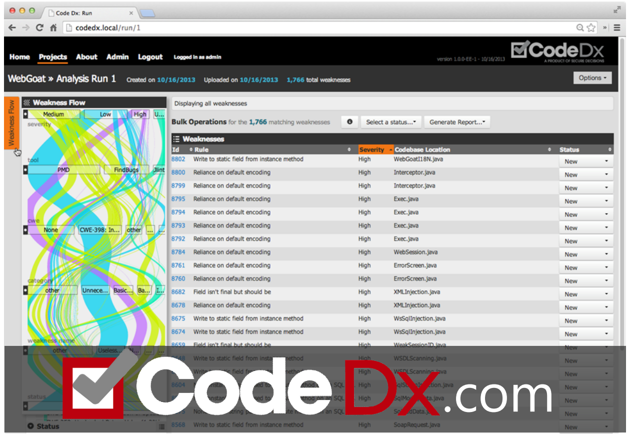
Interested in Process: What simple things can companies do to make it easier to write secure code?

Currently the Director of Sales for …

Code Dx

Code Dx, Inc.

Innovative commercial and open source
Application Security Testing solutions



WebGoat » Analysis Run 1 Created on 10/16/2013 Uploaded on 10/16/2013 1,766 total weaknesses

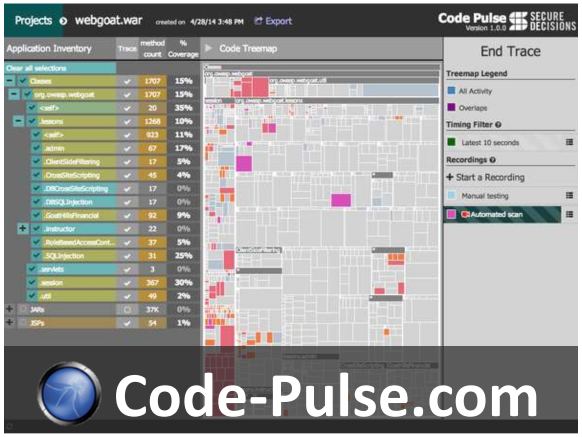
Displaying all weaknesses

Bulk Operations for the 1,766 matching weaknesses Select a status... Generate Report...

Weaknesses

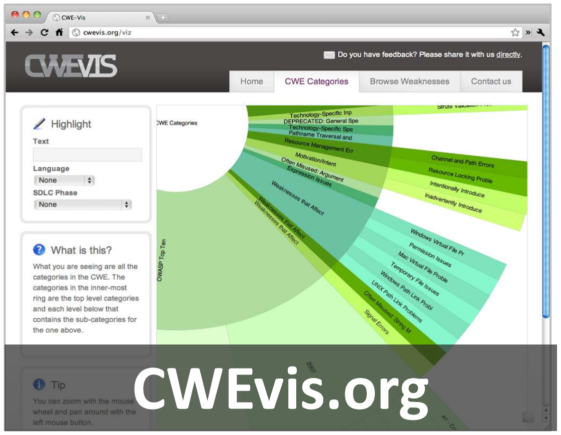
Id	Rule	Severity	Codebase Location	Status
8802	Write to static field from instance method	High	WebGoatI18N.java	New
8800	Reliance on default encoding	High	Interceptor.java	New
8799	Reliance on default encoding	High	Interceptor.java	New
8795	Reliance on default encoding	High	Exec.java	New
8794	Reliance on default encoding	High	Exec.java	New
8793	Reliance on default encoding	High	Exec.java	New
8792	Reliance on default encoding	High	Exec.java	New
8784	Reliance on default encoding	High	WebSession.java	New
8761	Reliance on default encoding	High	ErrorScreen.java	New
8760	Reliance on default encoding	High	ErrorScreen.java	New
8682	Field isn't final but should be	High	XMLInjection.java	New
8678	Reliance on default encoding	High	XMLInjection.java	New
8675	Write to static field from instance method	High	WsSqlInjection.java	New
8674	Write to static field from instance method	High	WsSqlInjection.java	New
8659	Field isn't final but should be	High	WeakSessionID.java	New
8568	Write to static field from instance method	High	SoapRequest.java	New

Vulnerability Correlation and Management



Application Inventory	Trace	method count	% Coverage
Clear all selections			
Classes	✓	1707	15%
org.owasp.webgoat	✓	1707	15%
<self>	✓	20	35%
.lessons	✓	1268	10%
<self>	✓	923	11%
.admin	✓	67	17%
.ClientSideFiltering	✓	17	5%
.CrossSiteScripting	✓	45	4%
.DBCrossSiteScripting	✓	17	0%
.DBSQLInjection	✓	17	0%
.GoatHillsFinancial	✓	92	9%
.instructor	✓	22	0%
.RoleBasedAccessCont...	✓	37	5%
.SQLInjection	✓	31	25%
.servlets	✓	3	0%
.session	✓	367	30%
.util	✓	49	2%
JARs	□	37K	0%
JSPs	✓	54	1%

Open source tool for tuning DAST tools



Do you have feedback? Please share it with us directly.

Highlight

Text:

Language: None

SDLC Phase: None

What is this?

What you are seeing are all the categories in the CWE. The categories in the inner-most ring are the top level categories and each level below that contains the sub-categories for the one above.

Tip

You can zoom with the mouse wheel and pan around with the left mouse button.

Free website for education about CWE

OWASP
Open Web Application Security Project

CodeDx

Public resources

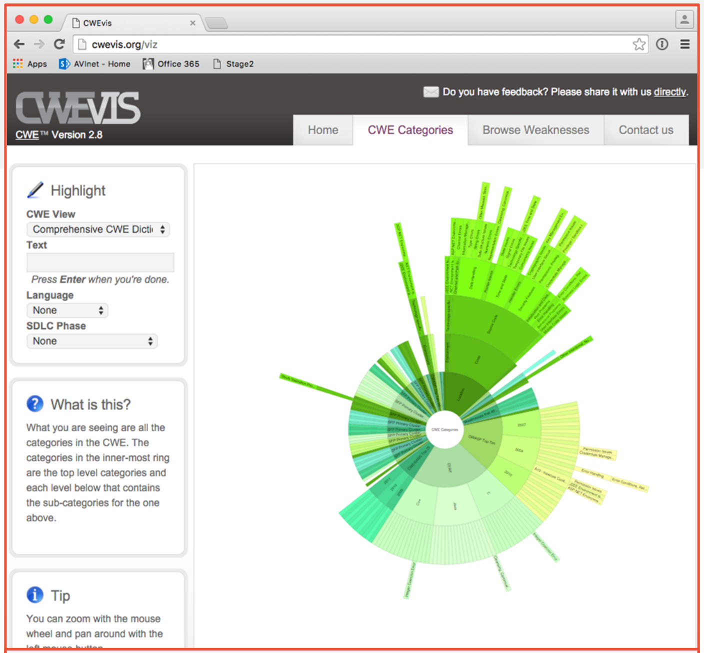
CWEvis - Interactive visualization to explore and learn the CWE
cwevis.org

Build Security In - DHS-sponsored education resource for software assurance
buildsecurityin.us-cert.gov

Software Assurance Marketplace (SWAMP) - DHS-sponsored software analysis and testing platform
continuousassurance.org

Code Dx Knowledge Center
https://codedx.com/resources/

Code Dx

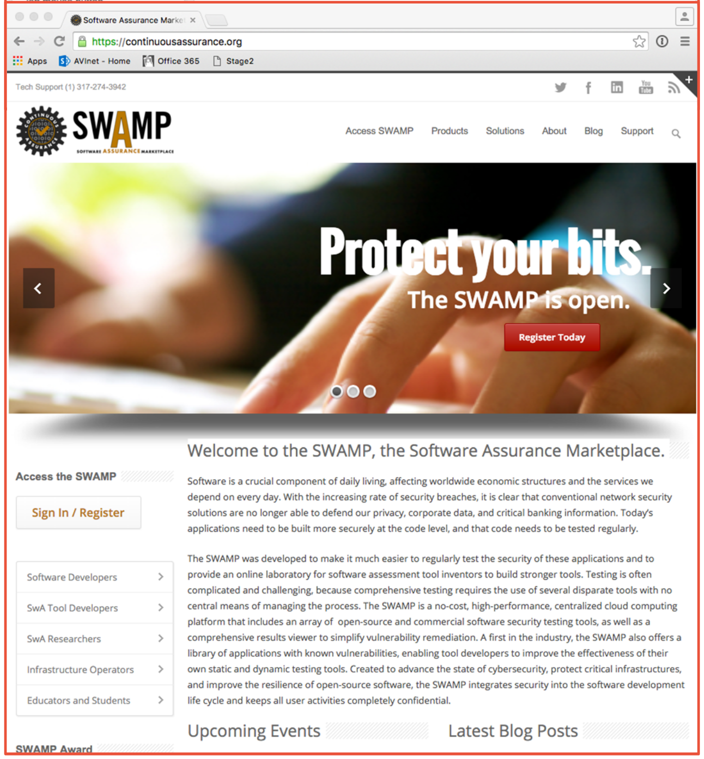


Do you have feedback? Please share it with us directly.

Press Enter when you're done.

What you are seeing are all the categories in the CWE. The categories in the inner-most ring are the top level categories and each level below that contains the sub-categories for the one above.

You can zoom with the mouse wheel and pan around with the left mouse button



Tech Support (1) 317-274-3942

Welcome to the SWAMP, the Software Assurance Marketplace.

Software is a crucial component of daily living, affecting worldwide economic structures and the services we depend on every day. With the increasing rate of security breaches, it is clear that conventional network security solutions are no longer able to defend our privacy, corporate data, and critical banking information. Today's applications need to be built more securely at the code level, and that code needs to be tested regularly.

The SWAMP was developed to make it much easier to regularly test the security of these applications and to provide an online laboratory for software assessment tool inventors to build stronger tools. Testing is often complicated and challenging, because comprehensive testing requires the use of several disparate tools with no central means of managing the process. The SWAMP is a no-cost, high-performance, centralized cloud computing platform that includes an array of open-source and commercial software security testing tools, as well as a comprehensive results viewer to simplify vulnerability remediation. A first in the industry, the SWAMP also offers a library of applications with known vulnerabilities, enabling tool developers to improve the effectiveness of their own static and dynamic testing tools. Created to advance the state of cybersecurity, protect critical infrastructures, and improve the resilience of open-source software, the SWAMP integrates security into the software development life cycle and keeps all user activities completely confidential.

Application Security (AppSec) protects software

AppSec: Measures taken to protect software applications from external threats

Security measures to prevent an attacker from

- Changing an application’s intended performance, by making it function in a different way or perform poorly
- Using the application as a vector for penetration into the enterprise on which it resides

Testing for vulnerabilities during and after development

Countermeasures to protect apps during an attack

Code Dx

AppSec applies to any software. Major focus is on…

- Public-facing web applications
  - Third leading method of perpetrating cyber crime
  - 75% of web apps contain vulnerabilities
  - 64% of companies experienced web-based attacks
  - 31% of Financial Services breaches are through web apps
- Mobile apps
  - 90% of most popular mobile health and finance apps have at least two critical vulnerabilities
  - Number of organizations testing their mobile apps for vulnerabilities nearly doubled from 2014 to 2015
- Cloud software
- Internally managed commercial software

Software weaknesses are at root of many 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

Finding and fixing these weaknesses *during development* reduces risk of attack and cost of fixing code

Bug Bounties are paid after code release

Google pays bounties up to $20k to find vulnerabilities in its Web browser

Microsoft offers as much as $150k

United Airlines pays bounties in *air miles*

Many factors make AppSec testing very resource-intensive

Existing commercial tools are **very expensive**.

Open-source tools are free, but are **hard to select and use.** And each tool works and reports differently.

*No single tool finds everything. You need to **run multiple tools** in order to find most vulnerabilities. Then **manually combine results**.*

Tools produce a **firehose of vulnerabilities**, that need to be prioritized so that the most important are fixed first.

Securing applications is hard, timely and costly
-- yet necessary.

Code Dx

DHS is investing heavily in AppSec testing

Goal: “Secure the software supply chain”

Prevent software-enabled attacks on infrastructure: stock exchange, power grid, water sources, transportation...

- Stock exchange software IS vulnerable.
  - Russians cracked the Nasdaq. Breached online portal that executives used to communicate
- Water supply software IS vulnerable
  - Iranians breached software controlling flood gates of NY dam

Make AppSec technologies better at finding “true positive” vulnerabilities, easier to use, and more accessible to everyone

Major types of AppSec testing

Manual

- Code review
- Manual pen-testing
- Low false positives
- Finds things that automated tools miss
- Hard to scale

SAST – Static Analysis

- Finds theoretical issues
- High false positives
- Results need to be triaged
- Automated, scalable

DAST – Dynamic Analysis

- Requires a running application
- Often used later in development cycle
- Automated, scalable

AppSec testing: Source code vs running app

Review the source code

- There are known coding patterns that present potential weaknesses
  - Common Weakness Enumeration (CWE) [MITRE – www.cwevis.org]
- Manual code review reveals some
- **Static Application Security Testing** (SAST) tools find them automatically

Simulated attack

- Probe a running application looking for ways in – *black-box testing*
- Apply known attack patterns looking for response *Pen-Testing*
- **Dynamic Application Security Testing** (DAST) tools act as "robot hackers"

When to Find Issues

The earlier in the Software Development Lifecycle that you find the vulnerabilities, the easier and cheaper they are to fix.

But you do have a choice. You can:

1. Find most vulnerabilities before the software is released, or
2. Find a whole lot of vulnerabilities after the software is released.

Choice 1: Find the bugs before the code is released

Developers and QA can find vulnerabilities during coding stage

- Source code analyzers (SAST) can be integrated into development environment, and with issue trackers.

Security analysts can find vulnerabilities at key points in the software development process.

- Use Manual code reviews, Static Application Security Testing (SAST) tools, and some dynamic testing
- Send results back to the developers for remediation

Choice 2: Find the bugs after the code is released

Application security analysts can conduct regular penetration testing

- Pay **white hat hackers** to test software from the outside
- Use automated pen testing -- Dynamic Application Security Testing (DAST) – tools on your web sites.

Rely on **strangers** to find the bugs. Offer bug bounties

- Google and Yahoo both pay up to $20k
- Microsoft offers as much as $150k for a really big bug
- United Airlines pays bounties in air miles
- WordPress pays up to $1K per bug

How to Do This

So it’s clear that we want AppSec built into the process and that we need agreement from the engineering teams to make this happen.

Here are the simple ways to make that happen.

Keys to Adoption of Secure DevOps

- Train Your Development Teams
- Run Tools Centrally - Tune Them for a Small Set of Highly Actionable Results
- Get Results to Development Teams - Available on the Desktop, Preferably the IDE
- Development Teams Should Look at Results About Twice per Week
- Above All: Eliminate the False Positives

Train Your Development Teams

Follow up regularly with short quizzes or other quick and simple tasks to make sure the knowledge is still there.

Code Reviews

Automatic scans

Other reinforcements

Run Tools Centrally

Tune Them for a Small Set of Highly Actionable Results

Central groups or specific people on a team should be in charge of running automated tools and reviewing the results initially

They should set policy for what absolutely needs to be fixed and make sure that development teams see results that most need to be fixed

Development Teams: Desktop, IDE

Get rid of the detailed reports that get mailed out

They are ignored

With documents / mail, or if responses are stored in multiple places (or not at all), there is a chance for information leak

Make results available directly from a central database

available on desktop

comments about them are reflected in one place

Get results to the IDE – that is where people work

Once again, small, actionable set of results

Look at Results About Twice per Week

Some people will take great exception to this

“people should scan their code often, even continually”

In practice, this varies widely

multiple times per day

every few months

Twice per week makes strikes a balance

Make sure that major issues don’t fester

AppSec does not take over the schedule

Developers should be thinking about a problem to be solved, designing, coding and writing their own little tests

Never slow down the process

Above All: Eliminate the False Positives

Nothing will kill an application security program faster than developers wasting their time chasing noise

everything should be clear, real and significant

otherwise: they will find an excuse

Small set of real issues = Any work in fixing them is worthwhile

Don’t worry about fixing every issue

Another sprint or another release right behind this one

Deeper dive at the end

Never slow down the process

Code Dx

Another Way of Looking At This

If you are a security analyst: Provide excellent training and send a small number of issues to be fixed.

If you run a security team: Have your team set simple policies aligned with training. Put in place a central database of security issues from which everyone works.

If you are a development manager: Demand that the security team provides your team a small number of actionable results or specific guidance on what to fix.

If you are a staff engineer: Pay attention to the training. Focus on a small set of real issues and make sure they are fixed.

Never slow down the process

Code Dx

Curtis Bragdon

Director of Sales

✉ curtis.bragdon@codedx.com

(617) 312-1466