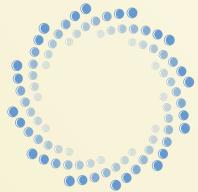
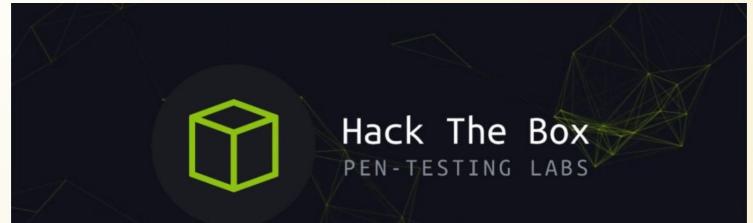


# Making Application Security Part of Your SDLC!

Joshua Dow  
Joshua.Dow@owasp.org  
@0xJDow

# `:~# whoami`

- Lead AppSec Engineer at ClearCompany
- Bug Bounty Hunter / CTF competitor
- Father, (soon to be) Husband, Gamer



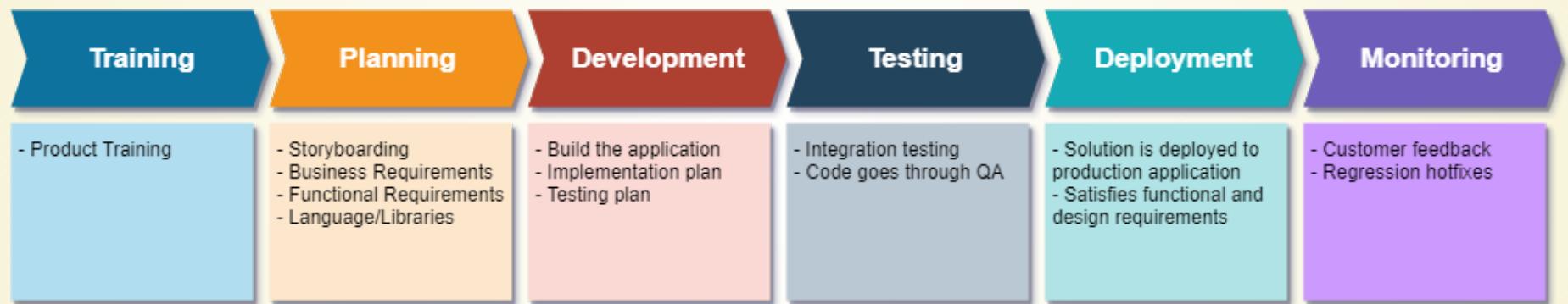
ClearCompany

**bugcrowd**

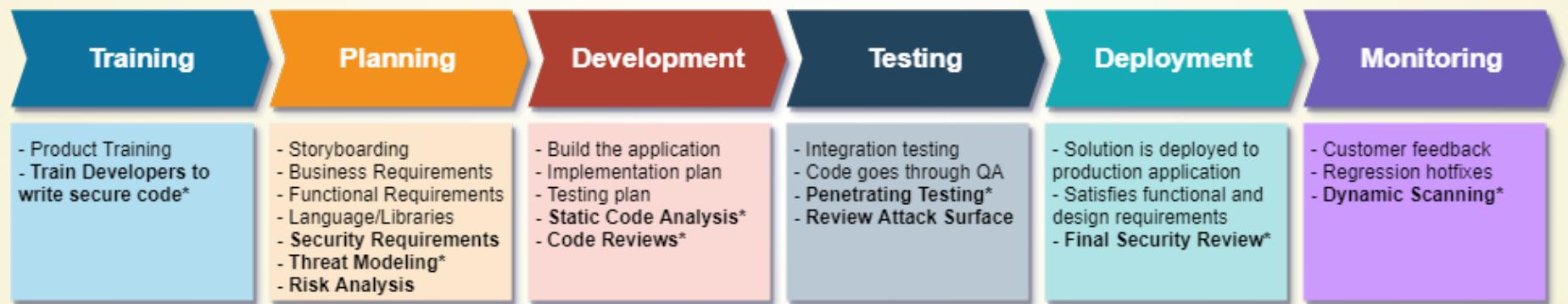
# So what's this all about anyway?

- Development teams are shipping code faster than ever before!
- Data breaches and cyber attacks are happening faster than ever before...
- To mitigate these risks security considerations need to be made at every part of the SDLC

# A Typical SDLC



# A Secure SDLC



# Training

# Security Training for Developers

- Developers should be accountable for the code they produce, but they cannot do that unless they know what to look out for.
- **Much** less costly to identify and fix bugs during development than after code has already made it further along the SDLC.

# Focus On

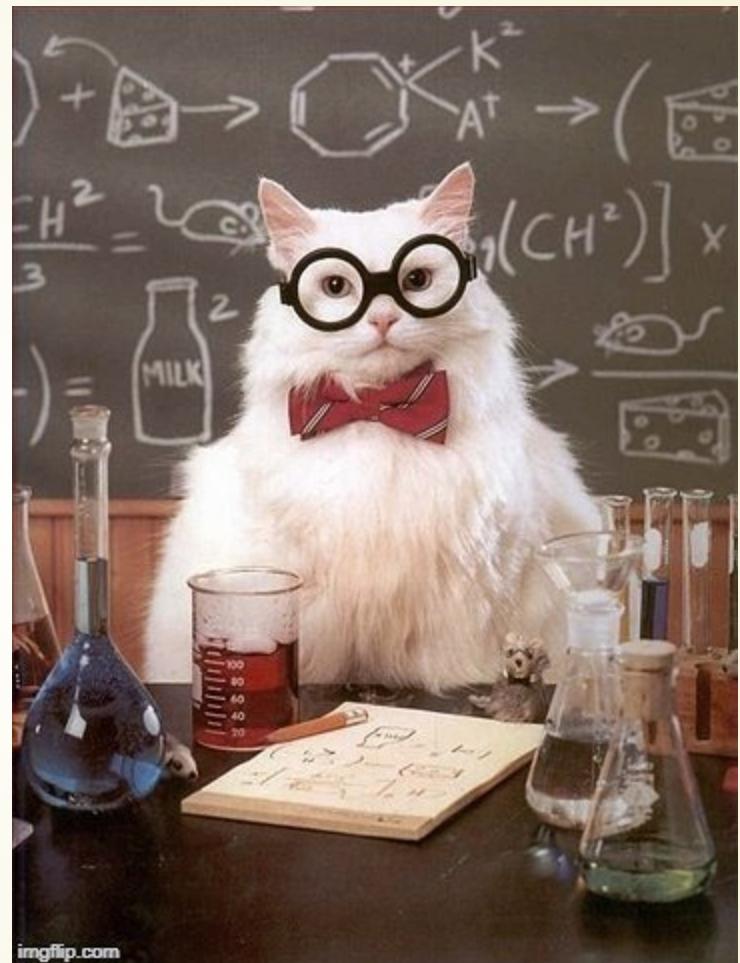
- High-level overview of the most common vulnerabilities
- Safeguards that **YOUR** application is using to defend against them
- Most common security decisions that developers need to be making while they are writing their code.

# Avoid

- Deep dives into every vulnerability class
- Assumptions that your development team is already familiar with the risks.

# Approaches

- In-person training is valuable.
- Online, interactive training is great for learning basics
- Typically fits well into new engineer onboarding process.



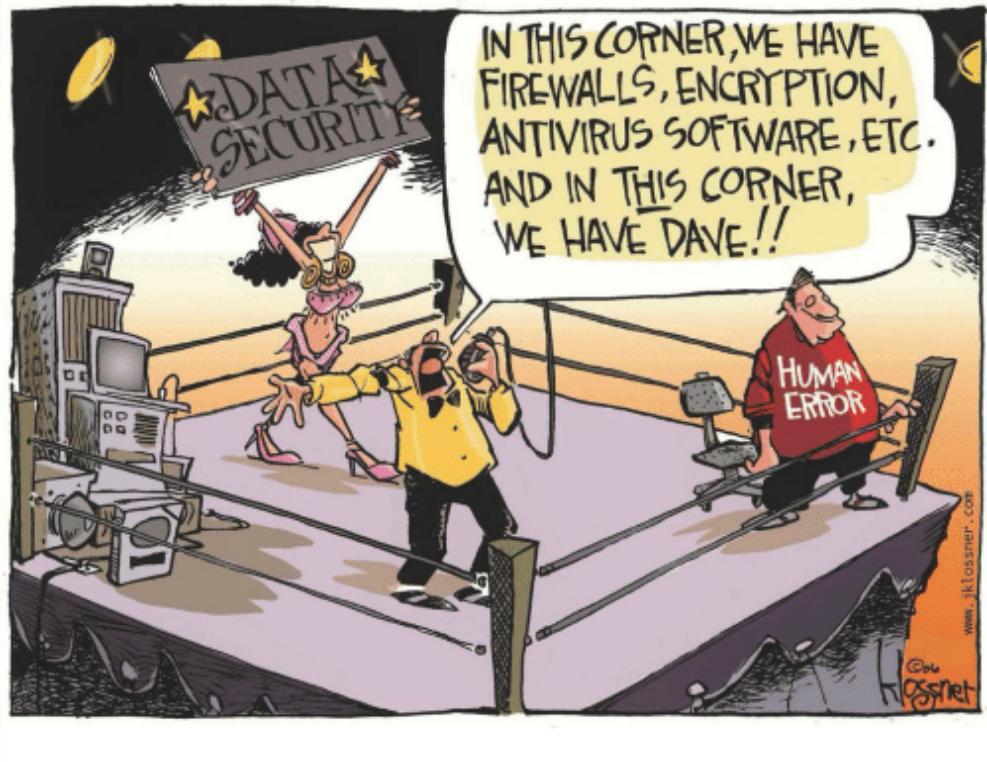
# Training Resources

- Pager Duty: Security Training for Engineers
- Hacksplaining.com
- OWASP Documentation && Cheat Sheets
- Paid training platforms

# Planning

# Threat Modeling

- Be realistic
- Include key stakeholders
- Ideally should be done early - findings can influence design
- Walk away understanding the controls required



# 4 3 Questions

1. What are we building?
2. What can go wrong?
3. What are we going to do about it?
4. ~~Did we do a good enough job?\*~~

# Example: Bank Web App

Wants to build a web app where users can log in, manage their accounts, view statements, etc.

Decision is made that they are going to start out by building authentication first...



# 4 3 Questions Applied

## 1. What are we building?

- The authentication piece & login experience of a banking web app

## 2. What can go wrong

- Broken Authentication / Session Management
- Data Breach / Credential Dumping

## 3. What are we going to do about that?

- Store passwords using a strong 1-way hashing algorithm
- Generic authentication error messages
- Login failure attempt account lockout

# Threat Modeling Resources

- OWASP Threat Modeling Cheat Sheet
- OWASP Application Threat Modeling
- Dan Tentler Talk @LayerOne 2017
- Checkmarx Threat Modeling Cheat Sheet

# Development

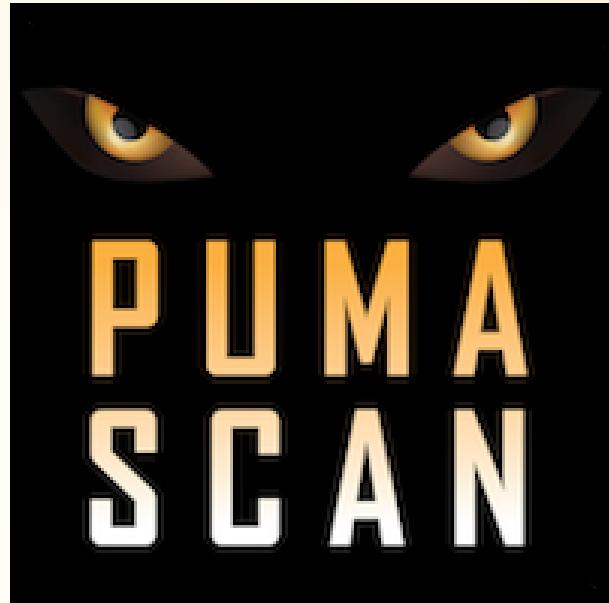
# Static Analyzers

- Incredibly good at catching low-hanging fruit
  - Insecure function calls
  - Code smells
- IDE plugin && CI integration most of the time
- Save development time by catching bugs before a branch is even merged



# Puma Scan

- C# code analysis tool built on top of the Roslyn .NET compiler platform
  - very extensible
- IDE Plugin available from Visual Studio Marketplace
- Analyzes source code and displays warnings as you type
- Pro and Enterprise versions available to integrate into CI pipeline



# Puma Scan in Action

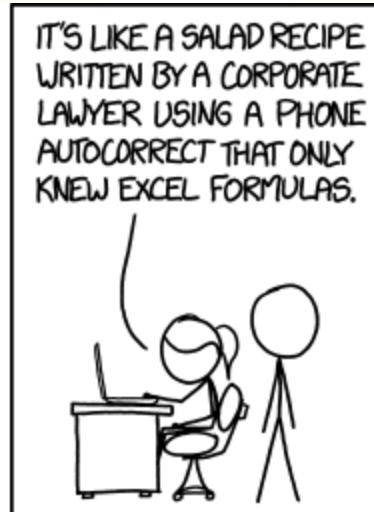
The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** PumaPrey-Scan - Microsoft Visual Studio
- Toolbar:** File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, Help
- Project Selector:** Release, Any CPU, IIS Express (Firefox)
- Solution Explorer:** Shows the project structure for 'PumaPrey-Scan' (4 projects) under 'Fox'. The 'Controllers' folder contains 'AccountController.cs', 'HuntController.cs' (highlighted in red), and 'ValuesController.cs'. Other files include 'Connected Services', 'Properties', 'References', 'App\_Start', 'Areas', 'Content', 'fonts', and 'Images'.
- Code Editor:** Displays the 'HuntController.cs' file with the following code:

```
52
53     // PUT api/hunt/5
54     public void Put(int id, [FromBody]string value)
55     {
56     }
57
58     // DELETE api/hunt/5
59     public void Delete(string id)
60     {
61         using (var context = new RabbitDBContext())
62         {
63             string q = string.Format("DELETE FROM Hunt WHERE Id = {0}", i
64             context.Database.ExecuteSqlCommand(q);
65         }
66     }
67 }
68
69 }
```
- Tooltips:** A tooltip for the 'ExecuteSqlCommand' call on line 64 indicates a "SQL Injection - EF method executes dynamic SQL without parameters". It also provides a link to "Show potential fixes" and a note about using Alt+Enter or Ctrl+.
- Bottom Navigation:** Error List, Output, Package Manager Console, Immediate Window
- Status Bar:** Ready, Ln 1, Col 1, Ch 1, INS, 0, 6, puma-prey, master

# Code Reviews

- 2 approvers
  - 1 subject matter expert
- Ensure code is complete, does what is designed for.
- Look for logic bugs
- Insecure function calls (though these should be caught by static analyzers)



# Testing

# Penetration Testing

- Specifically internal, white box, penetration testing
- Fits well into Agile - test a release before it goes out to production
- Scope should be small
  - Testing focus' on code changes being released in a given sprint



# Penetration Testing Goals

- Using an attacker's mindset
- Attempt to exploit vulnerabilities in the **new** features of the application that are up for release
- Findings should be actionable for the developers, ideally accompanied with a fix for the issue
  - No blame towards the developers
  - Feeds back into developer training

# Deployment

# Final Security Review

- Does the code satisfy both functional and security requirements?
- Has any new code been introduced that could affect the security of the code to be released?
- Do not want to reduce release velocity - only stop if a serious problem is discovered.



# Monitoring

# Dynamic Scanning (DAST)

- Constantly scans your application looking for vulnerabilities
- Often does a good job of regression testing new code
- Expedites the creation of hotfixes || rollback if a vulnerability made it out this far.



# Dynamic Scanning Resources

- Netsparker
- Veracode
- AppSpider



# Questions?