

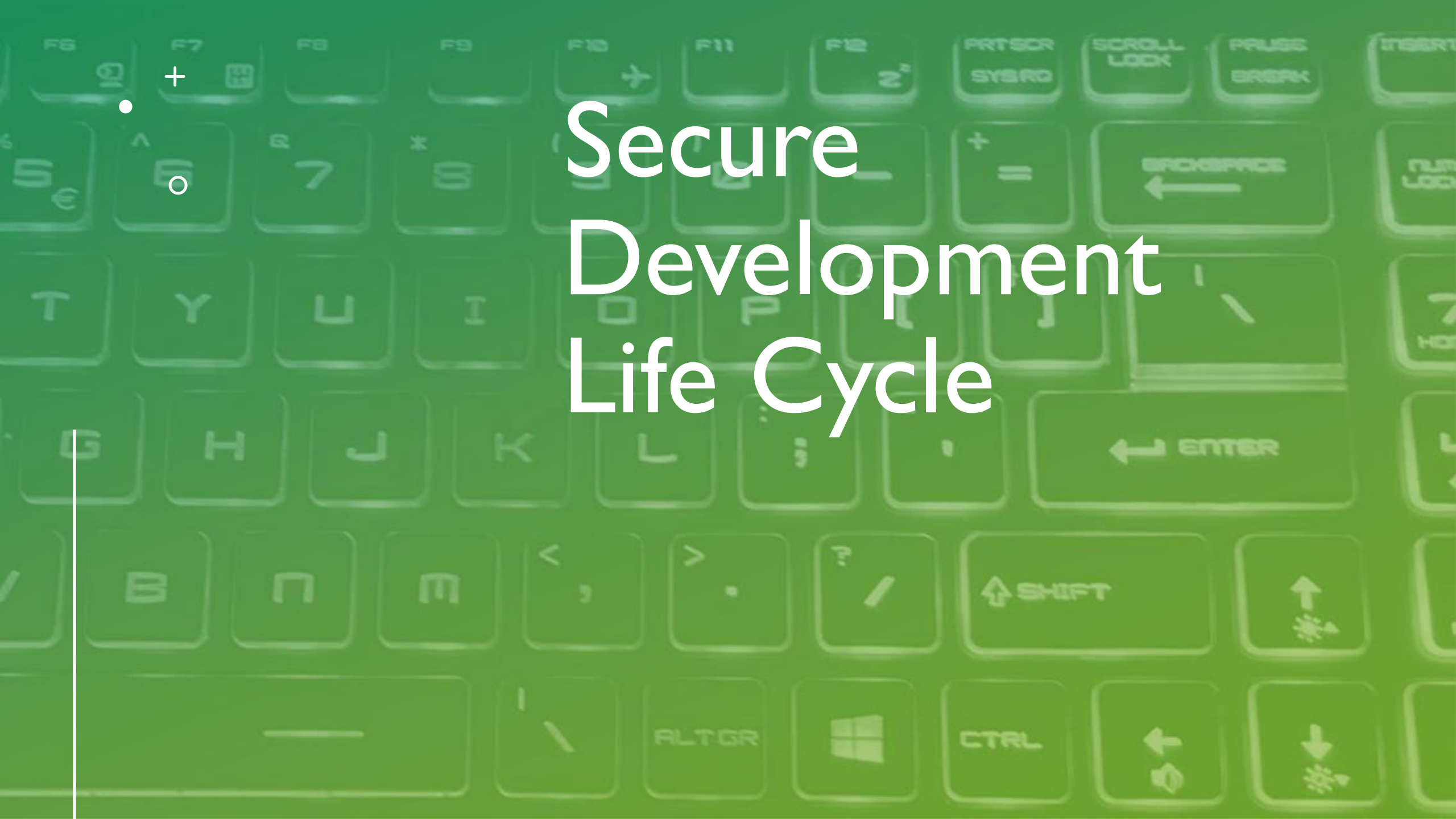
The background features a green gradient with a white line at the top. A series of light green cubes are arranged in a descending staircase pattern from the top left towards the bottom right. Small, stylized human figures are placed on various cubes, some standing, some sitting, and some working on laptops. In the top right corner, there are three small white symbols: a plus sign, an open circle, and a solid dot, arranged vertically.

#CODEHEDGEHOGS

CHANGING THE “S” IN SDLC TO “SECURE”

Penelope Rozhkova





Secure Development Life Cycle

Perform threat modeling throughout the process adjusting priorities and risk reduction strategies

Planning

ID Security &
Compliance
Objectives

Requirements

Establish Security
Standards

Perform Risk
Assessment

Design

Perform Code
Reviews

Perform Static
Analysis

Create Secure
Templates

Development

Analyze App's
Attack Surface

Document
App's Security
Architecture

Testing

Dynamic
Analysis

Review Attack
Surface &
Execute Pentests

Deployment

Final Security
Review

Create
Incident
Response Plan

Maintenance

Monitor CVEs
of App
Components

Document
Compliance
with Policies &
Standards

The Shift Left

OWASP Top 10 Web Application Security Risks

Injection

Broken Authentication

Sensitive Data Exposure

XML External Entities

Broken Access Control

Security Misconfiguration

Cross-Site Scripting (XSS)

Insecure Deserialization

Using Components with Known Vulnerabilities

Insufficient Logging & Monitoring

Explore

- Threat Agents / Attack Vectors
- Security Weakness
- Impacts
- Is the Application Vulnerable?
- How to Prevent It
- Example Attack Scenarios
- References

<https://owasp.org/www-project-top-ten/>

Explore

- Threat Agents / Attack Vectors
- Security Weakness
- Impacts

- Is the Application Vulnerable?
- How to Prevent It
- Example Attack Scenarios
- References

<https://owasp.org/www-project-top-ten/>

A2:2017-Broken Authentication

Languages: [en] de

← A1:2017-Injection

OWASP Top Ten 2017
PDF version

A3:2017-Sensitive Data Exposure →

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.		The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.	

Is the Application Vulnerable?	How to Prevent
<p>Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:</p> <ul style="list-style-type: none">* Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.* Permits brute force or other automated attacks.* Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".* Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.* Uses plain text, encrypted, or weakly hashed passwords (see A3:2017-Sensitive Data Exposure).* Has missing or ineffective multi-factor authentication.* Exposes Session IDs in the URL (e.g., URL rewriting).* Does not rotate Session IDs after successful login.* Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.	<ul style="list-style-type: none">* Where possible, implement multi-factor authentication to prevent automated credential stuffing, brute force, and stolen credential re-use attacks.* Do not ship or deploy with any default credentials, particularly for admin users.* Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.* Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.* Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.* Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.* Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Example Attack Scenarios	References
<p>Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.</p> <p>Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.</p> <p>Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.</p>	<p>OWASP</p> <ul style="list-style-type: none">* OWASP Proactive Controls: Implement Digital Identity* OWASP Application Security Verification Standard: V2 Authentication* OWASP Application Security Verification Standard: V3 Session Management* OWASP Testing Guide: Identity, Authentication* OWASP Cheat Sheet: Authentication* OWASP Cheat Sheet: Credential Stuffing* OWASP Cheat Sheet: Forgot Password* OWASP Cheat Sheet: Session Management* OWASP Automated Threats Handbook <p>External</p> <ul style="list-style-type: none">* NIST 800-63b: 5.1.1 Memorized Secrets* CWE-287: Improper Authentication* CWE-384: Session Fixation

ASSET-BASED PROTECTION — ENGINEERING FOR SUCCESS

**“Don’t focus on what is likely to happen—
but instead, focus on what can happen and
be prepared.”**

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf>

Gaining Access – Most Common Vulnerabilities Exploited During Pentesting

- Misconfiguration
- Kernel Flaws
- Buffer Overflows
- **Insufficient Input Validation**
- Symbolic Links
- File Descriptor Attacks
- Race Conditions
- Incorrect File & Directory Permissions

Preventing SQL Injection Vulnerabilities

- Avoid dynamically generated SQL statements
- Avoid (or take extra precautions when using) system stored procedures that use sp_execute, execute or exec
- Carefully consider permissions
- **VALIDATE EVERY USER INPUT**
- Use Allow Lists (preferred to Block Lists)
- ID and Properly Escape “special” characters



<https://www.youtube.com/watch?v=baY3Salhf10>



Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

Which Errors Are Included in the Top 25 Software Errors?

Introduced During Design

Introduced During Implementation

Quality Weaknesses with Indirect Security Impacts

Software Written in C

Software Written in C++

Software Written in Java

Software Written in PHP

Weaknesses in Mobile Applications

CWE Composites

CWE Named Chains

CWE Cross-Section

CWE Simplified Mapping

CWE Entries with Maintenance Notes

CWE Deprecated Entries

CWE Comprehensive View

Weaknesses without Software Fault Patterns

Weakness Base Elements

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77

<http://cwe.mitre.org/data/index.html>



#codehedgehogs

THREAT MODELING



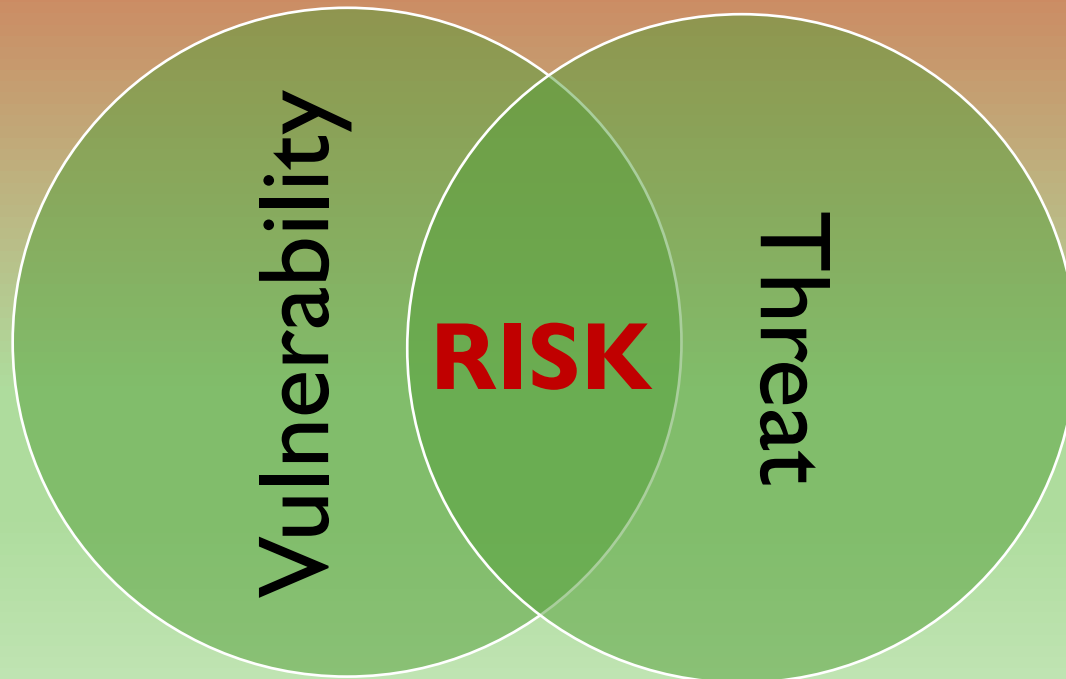
Attacker Centric



Software Centric



Asset Centric



- Identify Known Vulnerabilities
- Identify Threats to the Software System Design
- Understand the Risks
- Understand the Security Controls to Mitigate Risks
- Prioritize Risks
- Create Risk Reduction Strategy

Considerations

Technologies leveraged by the application
Client-Side Databases
REST
Native Programming Languages
Integrations

Non-technical Risks:

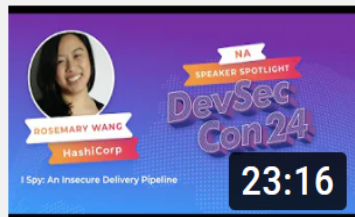
Loss Theft Delays

The earlier in the development lifecycle a risk can be mitigated the more secure and cost-effective securing the app will be.

How do I build my secure coding skills?



Talks



I Spy: An Insecure Delivery Pipeline - Rosemary Wang, Developer Advocate at HashiCorp

DevSecCon -

<https://www.devseccon.com/devseccon24-2021/>



Master the Art of Secure Software Coding and App Deve CSSLP

John Ng, Mox Bank; Pishu Mahtani, (ISC)² Authorised Instructor; To

The modern software developer faces an enormous amount of chal continuously creating innovative apps to ensuring high quality and n

1 week ago | 66 mins



VeraTalks: Tackling Developer Security Training

Rey Bango, Veracode Director of Developer Relations

Most AppSec programs forget that there is only one team that can fix development team. While an AppSec strategy based on scanni...

1 week ago | 26 mins



Secure Coding Best Practices

Matthew Butler, Principal Engineer

Computer systems are under siege 24 hours a day, day in and day o infrastructure designed to protect those systems, won't. Th...

1 month ago | 58 mins



BrightTALK

We Hack Purple is a Canadian company dedicated to helping anyone and everyone create secure software. We have an [online academy](#) with on-demand virtual security training, an [online community](#) for security professionals to connect with their peers and learn, a podcast for newcomers to in our industry, and a [newsletter](#) chock full of free content and funny memes.

Blogs

<https://wehackpurple.com/about/>



Founder: Tanya Janca

We Hack Purple Blog



Pushing Left, Like a Boss—Part 5.7—URL Parameters

Never put information in the parameters in the URL of your application...



Pushing Left, Like a Boss—Part 5.6—Redirects and Forwards

Recently removed from the OWASP Top Ten, unvalidated redirects and forwards are...



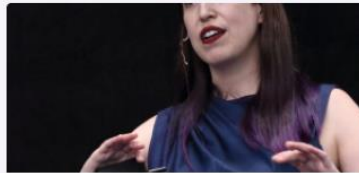
Pushing Left, Like a Boss—Part 5.5 File Uploads

Allowing files to be uploaded to your applications (and therefore your network)...



Pushing Left, Like a Boss—Part 5.4 Session Management

It is my firm opinion that only the session management features in...



Pushing Left, Like a Boss—Part 5.3—Browser and Client-Side Hardening

Browser and client-side hardening focuses on enabling and using the security features...



Pushing Left, Like a Boss—Part 5.2- Use Safe Dependencies

According to many sources between 70–90% of application code is contained within...



Pushing Left, Like a Boss—Part 5.1—Input Validation, Output Encoding and Parameterized Queries

The next several posts will break up the secure coding guideline from...



Pushing Left, Like a Boss: Part 4—Secure Coding

In the previous article in this series we discussed secure design concepts...



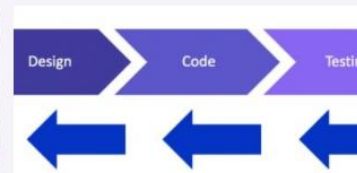
Pushing Left, Like a Boss!—Part 3: Secure Design

In the previous article in this series we discussed security requirements. When...



Pushing Left, Like a Boss!—Part 2: Security Requirements

In the previous article in this series we discussed why ensuring the...



Pushing Left, Like a Boss: Part 1

In all of the talks and articles I have ever written and...



One Year Anniversary of We Hack Purple

One year ago, I decided to start my own company. It's called We...

Identify Solution

Determine the correct fix from a number of different proposed solutions for the vulnerability listed below. These solutions will be full code repositories, where completely different approaches may have been taken to address the problem.

[VIEW SOLUTIONS](#)

Vulnerability Category

Injection Flaws - SQL Injection

This is the vulnerability you are trying to identify the correct solution for.


[Continue](#)


Fun & Games


To prevent the Unprotected Transport of Credentials, developers should:


- ⑤ Use strong protocols in TLS for all login pages
- ⑤ Ensure strong ciphers are in place
- ⑤ Protect the session token with valid certificates and monitor certificate validity to ensure uninterrupted protection
- ⑤ Do not allow sensitive data to be submitted without a secure channel in place


Active Missions


Level 1: A **cyber-criminal** from  Botswana is attacking the **Longitude Financial** application


Level 2: A **Hacker** from  Niger is attacking the **Energy Blue** application


Level 3: A **cyber-criminal** from  Canada is attacking the **Solar Electric** application

Level 4: A **state-sponsored adversary** from  Mexico is attacking the **Longitude Financial** application

Level 5: A **state-sponsored adversary** from  China is attacking the **Energy Blue** application

Level 6: A **Hacker** from  Germany is attacking the **World Bank** application

Level 7: A **Hacktivist** from  Italy is attacking the **Hot Deals** application


Level 8: A **Hacker** from  France is attacking the **World Bank** application

[View](#)[View](#)[View](#)[View](#)[View](#)[View](#)[View](#)[View](#)

RANGES

Financial Shadow Bank

Shadow Bank is the premier bank for people who love cryptocurrencies and have those pesky minimum password requirements. Transfer money, request a loan, or buy and sell stocks and currencies.




0 challenges solved

NOT AVAILABLE

HR AccountAll

AccountAll is the HR portal that turns humans into resources. Employees, managers and HR administrators can log in to manage payroll, timesheets, performance reviews and more.




0 challenges solved

NOT AVAILABLE

Retail Shred

Shred is your one-stop shop for skateboards, spray paint, stencils and all the other trappings of hoodlumism. Show off your best work in the graffiti gallery, or buy a gift card for the petty criminal in your life.




0 challenges solved

NOT AVAILABLE

Mobile/IoT Runstoppable

An Android fitness tracker that will turn you into a Runstoppable human being. NOTE: many challenges must be solved through flag submission found on the "Challenges" page under "My Stats".




0 challenges solved

NOT AVAILABLE

Financial The Gold Standard

Shadow Bank account got hacked? Switch to the Gold Standard for heightened security and a ROOM FULL OF GOLD.




0 challenges solved

NOT AVAILABLE

Cryptocurrency DigiExchange

Go crazy with crypto-currencies!




0 challenges solved

NOT AVAILABLE

Cloud Forensic

Welcome to your first day at Forensic! Please create a Portal account to start your journey.




0 challenges solved

NOT AVAILABLE

Cloud SecureSafeCrate


So Secure. So Safe. So CRATE. Powder Uniquely Inside Your Crate.



0 challenges solved

NOT AVAILABLE

BHIS ANTISYPHON CYBER RANGE


[Problems](#)
[Scoreboard](#)
[Rules](#)
[Stats](#)

[Logout](#)
[Dashboard](#)

With National University's scholarship opportunities, you may be eligible to save 25% on cybersecurity tuition. Start sooner and finish faster with 4-week classes and year-round enrollment at National University. Choose from 75+ programs and 100% online classes. Get started today at NU.edu

Binary-Exploitation

Cryptography

Forensics

Reconnaissance

Reverse Engineering

Web Exploitation

Other

★ Bonus

🛠 Help

⚙ UI Settings

🔪 CyberChef

Flag Format (solved by 1055 teams) 50

All flags should be obvious and a `string_separated_with_und3rscores`. Most of the flags will be surrounded by `MetaCTF{}` as well, but in the cases where that would make the problem too trivial, we did not include the `MetaCTF{}` part.

If the flag will be in a different format, we will specify that in the problem description. Additionally, all flags are case-insensitive. If you solve this challenge, make sure to tell your teammates about the flag format!

Submit!

Please rate this problem: ☆☆☆☆☆

Code a Porcupine!



1. Application Security both on the offensive and defensive aspect
2. Deep understanding of web vulnerabilities / Fixing vulnerabilities - Hands on is a must
3. Experience with SQLi and XSS
4. XSRF Experience
5. Code review
6. API
7. XXE and remediation
8. Appsec and CI/CD tools.

**IF YOU CODE SECURELY,
YOU ARE WELL-POSITIONED
TO TRANSITION INTO
CYBERSECURITY.**

#codehedgehogs

Bio

- Independent Contractor
- MUTC CyberAcademy Grad
- WiCyS Mentor
- CyberPatriot Coach
- BHIS Nerd Herder
- GenCyber Instructor



piranhamama



piranhamama#8888



penelopezrozhkovacsia



@RozhkovaCSIA



Animal (Owl, mouse, hedgehog and porcupine) photos courtesy of <https://unsplash.com/>

Slide deck available at <https://github.com/piranhamama/>