The Institute of Internal Auditors
Detroit Chapter
Presents
Use Cases – Webinar Series IIA Detroit Chapter
Webinar #3 – Web Applications, Vulnerabilities and Security
If You Have Questions…

Submit Questions and/or Comments via chat
Earning CPE Credit

In order to receive CPE credit for this webcast, participants must:

- Attend the webcast on individual device (one person per computer/device)
- Answer polling questions asked throughout the webcast
- CPE certificates will be sent to the e-mail address on your Zoom account within two weeks of this webinar.
- If you are unsure if that correct email address is on your zoom account, please send us the correct email in a chat message.
Please tell us your member status

A) Member Detroit Chapter

B) Member – Central Region District 2 (Fort Wayne, Toledo, Michiana, W. Mich., Lansing)

C) Member – Other District

D) Non-member
Agenda

- Web Application Vulnerabilities
- OWASP
- Web Application Firewall
- Demo/Case Study
Web Application Risks
Why Web Application Vulnerabilities Occur

Security Professionals Don’t Know The Applications

“As a Network Security Professional, I don’t know how my companies web applications are supposed to work so I deploy a protective solution...but don’t know if it’s protecting what it’s supposed to.”

Web Application Developers Don’t Know Security

“As an Application Developer, I can build great features and functions while meeting deadlines, but I don’t know how to develop my web application with security as a feature.”

The Web Application Security Gap
Web Application Vulnerabilities

“If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.”

-Weinberg's Second Law
Web Application Vulnerabilities

Web application vulnerabilities occur in multiple areas.

**Platform**
- Known Vulnerabilities in Operating Systems, Databases, and supporting infrastructure

**Administration**
- Extension Checking
- Common File Checks
- Data Extension Checking
- Backup Checking
- Directory Enumeration
- Path Truncation
- Hidden Web Paths
- Forceful Browsing

**Application**
- Application Mapping
- Cookie Manipulation
- Custom Application Scripting
- Parameter Manipulation
- Reverse Directory Transversal
- Brute Force
- Application Mapping
- Cookie Poisoning/Theft
- Buffer Overflow
- SQL Injection
- Cross-site scripting
Web Application Vulnerabilities

Application Programming:

- Common coding techniques do not necessarily include security
- Input is assumed to be valid, but not tested
- Unexamined input from a browser can inject scripts into page for replay against later visitors
- Unhandled error messages reveal application and database structures
- Unchecked database calls can be ‘piggybacked’ with a hacker’s own database call, giving direct access to business data through a web browser
Secure Coding Standards

- Secure coding refers to the practice of building secure software with a high level of security and quality. Software with a high level of security requires:
  - Understanding common software weaknesses that lead to security vulnerabilities
  - Following secure coding standards and practices
  - Performing in-depth code reviews
Polling Question #2

Who is responsible for securing web applications?

A. The Developer
B. The Network Manager
C. The Security Manager
D. All of the above
Open Web Application Security Project (OWASP)
OWASP

- OWASP provides the following:
  - Application security tools and standards
  - Complete books on application security testing, secure code development, and secure code review
  - Presentations and videos
  - “cheat sheets” on many common topics
  - Standard security controls and libraries
  - Local chapters
  - Cutting edge research
  - Conferences and education
OWASP – Top 10 Vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using components with Known Vulnerabilities
10. Insufficient Logging and Monitoring
Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

### Is the Application Vulnerable?

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters.

Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source (SAST) and dynamic application test (DAST) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.
SQL Injection

- Many web pages communicate directly to a backend database for processing.

- For example, a username and password is asked for on the Web page and the web page will pass it to the database to validate the information.

- Some applications will not validate the field adequately before passing it to the database, and the database will process whatever it will receive.

- Hackers will pass SQL commands directly to the database, and in some cases tables like “passwords” are returned because the SQL commands are not being filtered adequately.

- SQL may return errors in the web page that even lists the correct tables to query so that the hacker may make more accurate attempts to get data.
SQL Injection

- SQL Injection is the ability to inject malicious SQL commands into the backend code.

- For example:
  - `SELECT * FROM users WHERE username = 'USRTEXT' AND password = 'PASSTEXT'`
  - Passing ' OR 1=1-- in the USRTEXT field generates:
    - `SELECT * FROM users WHERE username = ' OR 1=1 -- AND password = 'PASSTEXT'`
  - The OR 1=1 returns true and the rest is commented out
SQL Injection – common attack strings

- ‘ or 27(hex) – delineates SQL string values.
- “ or 22 (hex) – also delineates SQL string values.
- ; or 3B (hex) - terminates statements.
- # or 23(hex) - also terminates a statement. (Access DB)
- /* or 2F2A (hex) - comment delimiter.
- -- or 2D2D (hex) – also comment delimiter.
- ( or 28 (hex) or ) or 29 (hex) – logical sub clauses.
- { or 7B (hex) or } or 7D (hex) – terminates a question. exec – used to call MS-SQL stored procedures.
- union – a SQL command very common to SQL injection.
SQL Injection – common fixes

- Validate the form field to only accept specific input for the fields.
  - For example, for login name use ^[0-9a-zA-Z]*$, which is Regular expressions for an alpha-numerical field.
  - For JSPs/Servlets, validate in the Servlet using the with the “java.util.regex” framework in a similar manner.

- Don’t use SQL
  - Use Prepared Statements, or Hibernate, to call the database. http://www.owasp.org/index.php/Preventing_SQL_Injection_in_Java
Polling Question #3

What is the most straightforward technique to apply that helps prevent SQL Injection?

A. Process Authorization
B. Process Authentication
C. Input Validation
D. Access Control
Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.

**Is the Application Vulnerable?**

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:

- 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.
Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

Is the Application Vulnerable?

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU’s General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
- Is sensitive data stored in clear text, including backups?
- Are any old or weak cryptographic algorithms used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
- Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

See ASVS Crypto (V7), Data Prot (V9) and SSL/TLS (V10)
Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

**Is the Application Vulnerable?**

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:

- The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.

- Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the [OWASP Cheat Sheet 'XXE Prevention'](https://www.owasp.org/images/pdf/cheatsheets/OWASP_Cheat_Sheet_-_XXE_Prevention.pdf).

- If your application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.

- If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.

- Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack.
Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as accessing other users’ accounts, viewing sensitive files, modifying other users’ data, changing access rights, etc.

**Is the Application Vulnerable?**

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another user’s record, permitting viewing or editing someone else’s account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation.
- CORS misconfiguration allows unauthorized API access.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user. Accessing API with missing access controls for POST, PUT, and DELETE.
OWASP – 6. Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.

Is the Application Vulnerable?
The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
- The server does not send security headers or directives or they are not set to secure values.
- The software is out of date or vulnerable (see A9:2017-Using Components with Known Vulnerabilities).

Without a concerted, repeatable application security configuration process, systems are at a higher risk.
OWASP – 7. Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Is the Application Vulnerable?

There are three forms of XSS, usually targeting users' browsers:

**Reflected XSS:** The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim’s browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

**Stored XSS:** The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.

**DOM XSS:** JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs. Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.
Polling Question #4

Why is Stored XSS considered a “high or critical risk”? 

A. Provides information on network topography  
B. Exposes sensitive user data  
C. Obfuscates critical system data, such as passwords and timestamps
Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

**Is the Application Vulnerable?**

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker. This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

Serialization may be used in applications for:

- Remote- and inter-process communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens
OWASP – 9. Using Components with known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

Is the Application Vulnerable?
You are likely vulnerable:

• If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
• If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
• If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
• If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
• If software developers do not test the compatibility of updated, upgraded, or patched libraries.
• If you do not secure the components’ configurations (see A6:2017-Security Misconfiguration).
Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

**Is the Application Vulnerable?**

Insufficient logging, detection, monitoring and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

You are vulnerable to information leakage if you make logging and alerting events visible to a user or an attacker (see A3:2017-Sensitive Information Exposure).
OWASP Learning Tool – Juice Shop

https://sy-juice-app.herokuapp.com/#/

Help with understanding the Juice Shop App:
https://github.com/bsqrl/juice-shop-walkthrough
Blockbuster Video - Demo
Demo 1

Vulnerability Scan

1. The website (URL: http://52.39.121.63/) never went through proper vulnerability scanning. This is one of the external facing web application

2. The IT department would like you to perform a vulnerability scan on the URL and determine if there are any security risks. Use any vulnerability scanning tool you wish: For example; “OWASP ZAP (Zed Attack Proxy)” which can be found in Kali Linux or downloaded at “https://github.com/zaproxy/zaproxy/wiki/Downloads”

Please note that it is illegal to scan any website without prior authorization. We have given you the authorization to only scan this URL. Please do not use this tool to scan other websites prior to approval from the owners of the websites)

3. You may have to go to Analyze and the policy manager and select a policy of HIGH for SQL injection

4. Be prepared to discuss the experience with others
Demo 2

Attack Vectors – SQL Injection

1. Use the same website (URL: http://52.39.121.63/ ) and the output from the ZAP PROXY and determine your attack strategy.

2. What attack vectors can be used to attack the website

3. Perform a SQL Injection attack

4. Be prepared to discuss the experience with others
Demo 3

SQLMAP pen test

Run the SQLMAP pen test on the same URL.
Demo 4

Attack Vectors – XSS

1. Use the same website (URL: http://52.39.121.63/ ) and the output from the ZAP PROXY and determine your attack strategy.
2. What attack vectors can be used to attack the website
3. Perform a XSS attack
4. Be prepared to discuss the experience with others
Polling Question #5

One means of denying SQL Injection attacks is to use:

A. Prepared Statements
B. Output filtering
C. Virtual proxy servers
Securing Web Applications - WAF
WAF – A Quick Fix?

- Instead of rewriting code, some potentially quicker methods is to put an application to intercept the HTTP traffic ahead of the HTTP server known as a Web Application Firewall (WAF).

- The WAF takes configurations like a normal firewall on what traffic to pass and reject. The difference is that it is responding specifically to an HTTP server like Apache or IIS.

- For Apache, the most popular approach is to use its Open Source plugin called mod_security. [http://www.modsecurity.org/](http://www.modsecurity.org/)

- For IIS, WebKnight from AQTronix, [http://aqtronix.com/?PageID=99](http://aqtronix.com/?PageID=99) is the most popular Open Source solution.

- Not everything can be covered by a WAF, especially session hijacking flaws, but XSS and SQL Injection can be mitigated.
OSI 7 Layers facilitate transmit / receive data

Question to the Class:
- Which layer does a regular Firewall operate?
- Which layer does a WAF operate?
Common Protocols for each layer

<table>
<thead>
<tr>
<th>Layer</th>
<th>Name</th>
<th>Example protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application Layer</td>
<td>HTTP, FTP, DNS, SNMP, Telnet</td>
</tr>
<tr>
<td>6</td>
<td>Presentation Layer</td>
<td>SSL, TLS</td>
</tr>
<tr>
<td>5</td>
<td>Session Layer</td>
<td>NetBIOS, PPTP</td>
</tr>
<tr>
<td>4</td>
<td>Transport Layer</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>3</td>
<td>Network Layer</td>
<td>IP, ARP, ICMP, IPSec</td>
</tr>
<tr>
<td>2</td>
<td>Data Link Layer</td>
<td>PPP, ATM, Ethernet</td>
</tr>
<tr>
<td>1</td>
<td>Physical Layer</td>
<td>Ethernet, USB, Bluetooth, IEEE802.11</td>
</tr>
</tbody>
</table>
Web Application Firewalls

- WAFs are filters that sit in front of the Web Application.
- Depending on their configuration, they will deny, or log, validated information from the Internet into the Application.
- They are a good source in auditing the information that is hitting the Web site and the scans that are constantly taking place.
Web Application Firewalls – Pros and Cons

Pro’s:

- Installing a WAF is quicker, in most cases, than changing code and re-deploying a Web Application.
- WAF’s may find issues, by using its rule sets, that the code may not be prepared to find. This is because WAFs have thousands of rules generated by industry experts.

Con’s:

- WAFs are limited by the rules that are installed in them. Therefore, if the rule is not there, it cannot protect against it.
- Validation is a better protection, because form level validation will use white-listing on what input is allowed, versus black-listing on the input that is denied.
Web Application Firewalls

- How does it work?

https://www.youtube.com/watch?v=p8CQcF_9280
Polling Question #6

Are WAFs a substitute for Secure Coding?

A. Yes, they can block attacks that developers may not be aware of.
B. No, they only block what their rules tell them to
Web Application Firewalls

- Demo of WAF (WebKnight)
- Activities:
  - Where is the logfile kept?
  - Is password 654321 blocked?
  - Is & scanned for SQL injection?
Auditing Web Applications – Guidelines and Demos
Audit Guidelines

- Source Code Scan (as part of contract if the application is outsourced)
- Vulnerability scan of the URL
- Credentialed internal scan
- Penetration Testing
Polling Question #7

Credentialed Internal Scans require access to:
A. User credentials
B. Virtual Private Networks
C. Authorized hardware
Demo 5

Attack Vectors – SQL Injection

1. Use the WAF protected website ec2-34-220-146-208.us-west-2.compute.amazonaws.com

2. Try the same attack vector on this website to see the impact of WAF on SQL Injection attack

3. Did it work? If so, why? If not, why not?

4. Be prepared to discuss the experience with others
Demo 5

Go to WebKnight WAF and show them where SQL Injection change could occur for the Attack Vector. ec2-34-220-146-208.us-west-2.compute.amazonaws.com

1. Show how the SQL attack vector can be stopped if the attack vector succeeded.

2. Discuss why the attack vector did not succeed (for those students where the attack vector was blocked)

3. Be prepared to discuss the experience with others