ARIZONA Auditor General
Making a Positive Difference

Auditing Web Applications

NSAA Information Technology Workshop and Conference

September 24, 2019

Gina Alvarado, IT Auditor, Office of the Auditor General (AZ)

Sajay Rai, President and CEO, Securely Yours LLC

Auditing Web Applications Outline

Arizona audit plan

- Web application testing and web application development

Recommendations/common findings

Secure coding standards

Case study

ARIZONA Auditor General
Making a Positive Difference

Arizona Auditor General

- 207 Staff
- Financial, Federal and Performance Audits
- State Agency, School District, University, County, Boards, Commissions Audits
- In FY 2018 issued over 200 reports:
  - 104 performance audits/follow-ups of state agencies and school districts,
  - 7 financial investigations and alerts,
  - 47 financial and federal compliance audits,
  - 54 accountability reviews,
  - 5 special audits/reviews
- Information Technology Audit Team
  - 1 audit manager
  - 6 IT auditors

ARIZONA Auditor General
Making a Positive Difference

Arizona – Experience

Auditing and reporting on web application issues since 2006

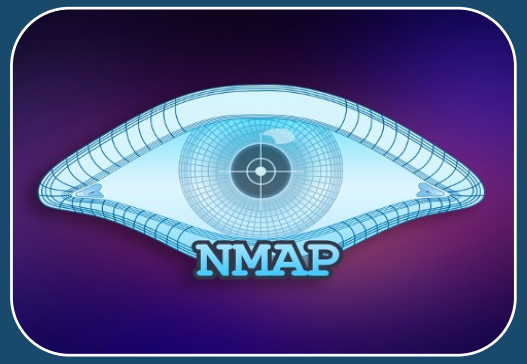
Varied practices by IT audit

- Have scanned, tested, and exploited web applications
- Worked with various IT consultants via an RFP process
- Also worked without IT consultants

Arizona – common security tools

Security Audit Machines

Burp Suite



NMAP

Nessus

Metasploit

Qualys SSL Scan

ARIZONA Auditor General
Making a Positive Difference

Arizona – training

Trainings

- Training from consultants during audit engagements
- SANS trainings
- Cybrary cybersecurity trainings and labs

Hands-on team exercises and hackathons

- Arizona Cyber Warfare Range
- Hacksplaining

Scan our own network monthly

ARIZONA Auditor General
Making a Positive Difference

Audit Planning

First decide the depth of the web app auditing

Scan and provide results

Test and analyze the likelihood of an exploit

Manual testing

Audit Planning – root cause

Consider the root cause

- What processes might contribute to vulnerabilities identified in scanning and testing?

We might consider other IT areas such as:

- Web application development
- Vulnerability management
- Patch management
- Configuration management

Audit Plan Objectives

Objective: Web application testing

- Perform web app scanning and security testing of the entity's mission critical applications using a risk-based approach

- Arizona - Audit Methodology
  - Step 1: Gather information
  - Step 2: Define testing parameters
  - Step 3: Perform reconnaissance
  - Step 4: Perform scanning and testing
  - Step 5: Communicate testing results and issues
  - Step 6: Assess and document best practices, cause, and effect

Gather Information

Ask entity to fill out application inventory spreadsheet

- Identifies the entity's mission critical applications
  - Includes web and non-web-based applications

- Provides key information about each application
  - Description
  - Type of data stored/processed
  - Number of users
  - How the application was developed
  - Size of the application
  - IP address

Define Testing Parameters

Select web applications using a risk-based approach

- Factors to consider:
  - ✓Sensitive data stored or processed in the application
  - ✓Accessibility (internet accessible vs. internally accessible)
  - ✓Number of dynamic pages and number of users
  - ✓Purpose of the application and how it ties into other audit objectives
  - ✓Input from the entity

Define Testing Parameters

Fill out Security Testing Notification Letter

Provides notification of our activities

- Necessary access

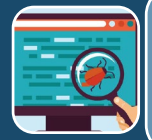
- Service degradation and/or interruption

- Confidentiality of scanning and testing results

Defines technical logistics

- Technical point of contact

- Scanning and testing period



- Examples of tests to be performed and tools to be used

- Scope of testing (IP ranges)

ARIZONA Auditor General
Making a Positive Difference

Perform Reconnaissance

Perform a discovery scan of external-facing IP range

- Discovers live hosts, open ports, and services running on these live hosts
- Compare the results of the discovery scan to the IP addresses in the application inventory
  - For all IP addresses not in the application inventory, follow-up with the entity
- Common tools
  - NMAP
  - Nessus

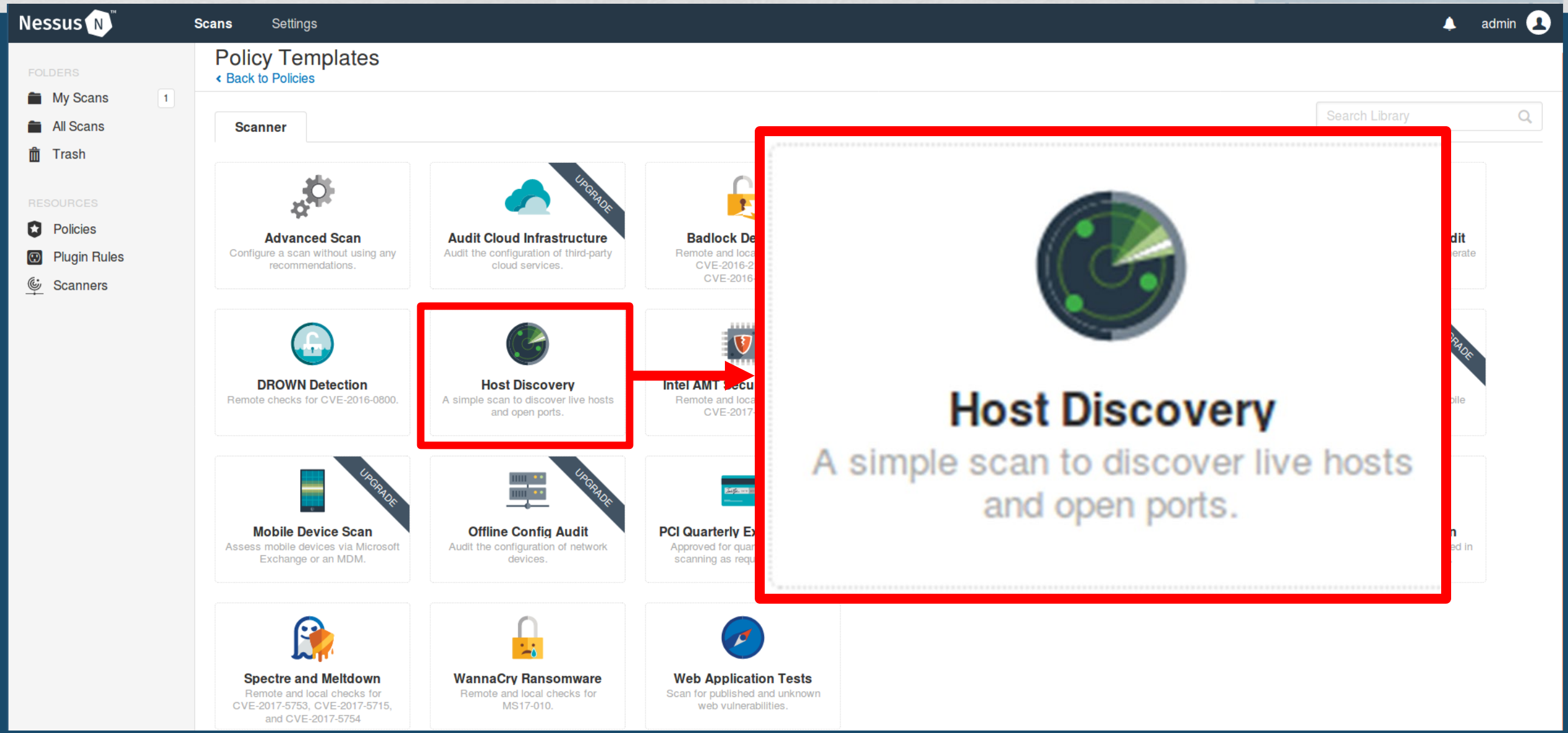
NMAP – Host Discovery

Host Discovery

- nmap -sn –n -v -oA outputfile -iL inputfile.txt
  - -sn instructs NMAP to not perform a port scan
  - -n instructs NMAP to not resolve any DNS names
  - -v instructs NMAP to increase verbosity during the scan
  - -oA instructs NMAP to export scan in normal, XML, and Grepable format
  - -iL instructs NMAP to scan all IP addresses listed in the specified file

NMAP Reference Guide – nmap.org

Nessus – Discovery Scan



Nessus – Discovery Scan Results

Plugin ID	Risk	Host	Protocol	Name	Synopsis	Description	Plugin Output
10180	None	111.11.11.01	tcp	Ping the remote host	It was possible to identify the status of the remote host (alive or dead).	Nessus was able to determine if the remote host is alive using one or more of the following ping types: - An ARP ping, provided the host is on the local subnet and Nessus is running over Ethernet. - An ICMP ping. - A TCP ping, in which the plugin sends to the remote host a packet with the flag SYN, and the host will reply with a RST or a SYN/ACK. - A UDP ping (e.g., DNS, RPC, and NTP).	The remote host is up The remote host replied to a TCP SYN packet sent to port 443 with a SYN,ACK packet
10180	None	111.11.11.02	tcp	Ping the remote host	It was possible to identify the status of the remote host (alive or dead).	Nessus was able to determine if the remote host is alive using one or more of the following ping types : - An ARP ping, provided the host is on the local subnet and Nessus is running over Ethernet. - An ICMP ping. - A TCP ping, in which the plugin sends to the remote host a packet with the flag SYN, and the host will reply with a RST or a SYN/ACK. - A UDP ping (e.g., DNS, RPC, and NTP).	The remote host is up The remote host replied to a TCP SYN packet sent to port 80 with a SYN,ACK packet

Nessus – Discovery Scan Results

Plugin ID	Host	Protocol	Name	Plugin Output
10180	111.11.11.01	tcp	Ping the remote host	The remote host is up The remote host replied to a TCP SYN packet sent to port 443 with a SYN, ACK packet

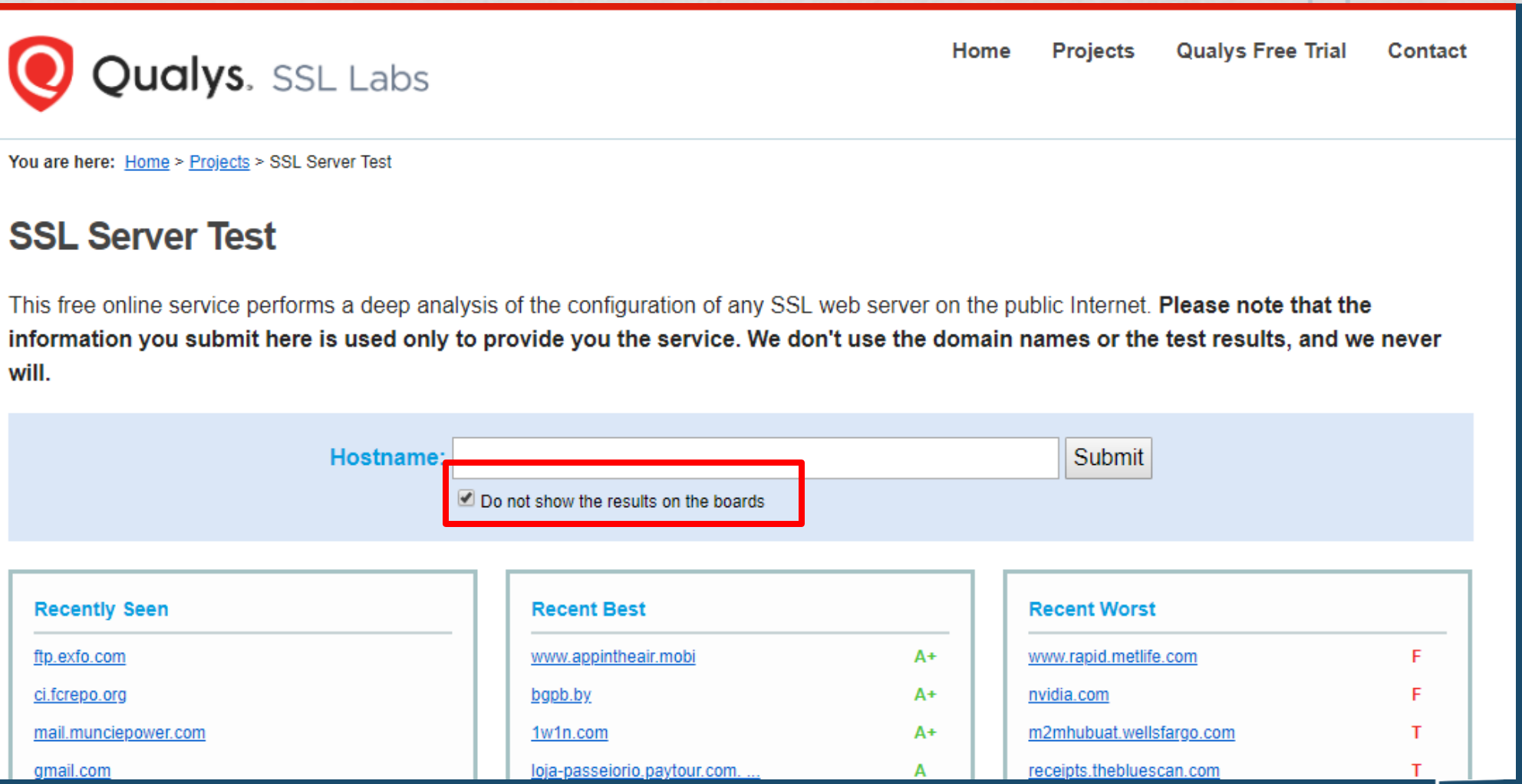
Perform Reconnaissance

Identify and research potential vulnerabilities

- Query service versions/configurations of the entity's infrastructure
- Cross reference services and versions to vulnerabilities to identify exploitable conditions

- Common security tools we use
  - Qualys SSL Server Test
  - Exploit-DB/ SearchSploit
  - Common Vulnerabilities and Exposures (CVE) listing
  - Security Headers scan

Qualys SSL Server Test



www.ssllabs.com/ssltest/

Qualys SSL Server Test

Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: Home > Projects > SSL Server Test > google.com

SSL Report: google.com

Assessed on: Tue, 27 Aug 2019 20:09:50 UTC | **HIDDEN** | Clear cache

Scan Another >>

	Server	Test time	Grade
1	216.58.194.174 sfo07s13-in-f174.1e100.net Ready	Tue, 27 Aug 2019 20:06:27 UTC **Duration:** 100.414 sec	A
2	2607:f8b0:4005:804:0:0:0:200e sfo07s13-in-x0e.1e100.net Ready	Tue, 27 Aug 2019 20:08:07 UTC **Duration:** 102.958 sec	A

Qualys SSL Test – detailed results

Protocol Details

DROWN	No, server keys and hostname not seen elsewhere with SSLv2 **(1) For a better understanding of this test, please read this longer explanation** (2) Key usage data kindly provided by the Censys network search engine; original DROWN website here (3) Censys data is only indicative of possible key and certificate reuse; possibly out-of-date and not complete
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	No
Insecure Client-Initiated Renegotiation	No
BEAST attack	Not mitigated server-side (more info) TLS 1.0: 0xc009
POODLE (SSLv3)	No, SSL 3 not supported (more info)
POODLE (TLS)	No (more info)
Zombie POODLE	No (more info) TLS 1.2 : 0xc009
GOLDENDOODLE	No (more info) TLS 1.2 : 0xc009
OpenSSL 0-Length	No (more info) TLS 1.2 : 0xc009
Sleeping POODLE	No (more info) TLS 1.2 : 0xc009
Downgrade attack prevention	Yes, TLS_FALLBACK_SCSV supported (more info)
SSL/TLS compression	No

Scanning and Testing – getting started

Before starting scanning or testing

- Update and configure all scanners and security tools

- Remind the web application owners of the testing windows
  - The entity should take necessary precautions to avoid unexpected outages
  - The entity should not make any changes to the application during the testing window

- Email the entity at the start and end of testing each day

Perform Web Application Scanning

Web application scanning – uses automated scanners to crawl a website to identify vulnerabilities.

- Common web application scanners:
  - Burp Suite
  - Nessus
  - Nexpose
- Evidence and documentation
  - Burp Suite issues report
  - Screenshots
  - Screen capture videos
- Keep an open eye for any strange behaviors or issues

Perform Web Application Testing

Web application testing – simulating attacks on a web application to exploit identified vulnerabilities.

- Once a vulnerability is found, continue to test for that vulnerability in other areas of the application
- Common web application testing tools:
  - Burp Suite
  - Metasploit
  - Nikto
  - Responder
- Guidance – OWASP Testing Guide

Communicate Testing Results

Communicate the testing results as soon as possible

- Share vulnerabilities discovered on specific applications
  - Schedule a meeting to discuss testing results verbally
  - Share the detailed testing reports
- Recommend remediation solutions, when possible

Missing HTTP Security Headers (Medium)

A Security Headers scan was performed on the 5 web applications selected for testing. These security headers provide additional security to web applications. The following HTTP Headers were missing from the applications tested:

- Strict-Transport-Security (App 1, App 2, App 3, App 4, App 5)
- Content-Security-Policy (App 1, App 2, App 3, App 4, App 5)
- X-Frame-Options (App 2, App 4, App 5)
- X-XSS-Protection (App 1, App 2, App 3, App 4, App 5)
- X-Content-Type-Options (App 1, App 2, App 3, App 5)
- Referrer-Policy (App 1, App 2, App 3, App 4, App 5)
- Feature-Policy (App 1, App 2, App 3, App 4, App 5)

Remediation: Add these headers to the appropriate web applications.

Arizona – challenges

Getting the notification letter signed

Lack of non-production environment for testing

Sharing the testing results verbally

Reporting testing issues in a public report

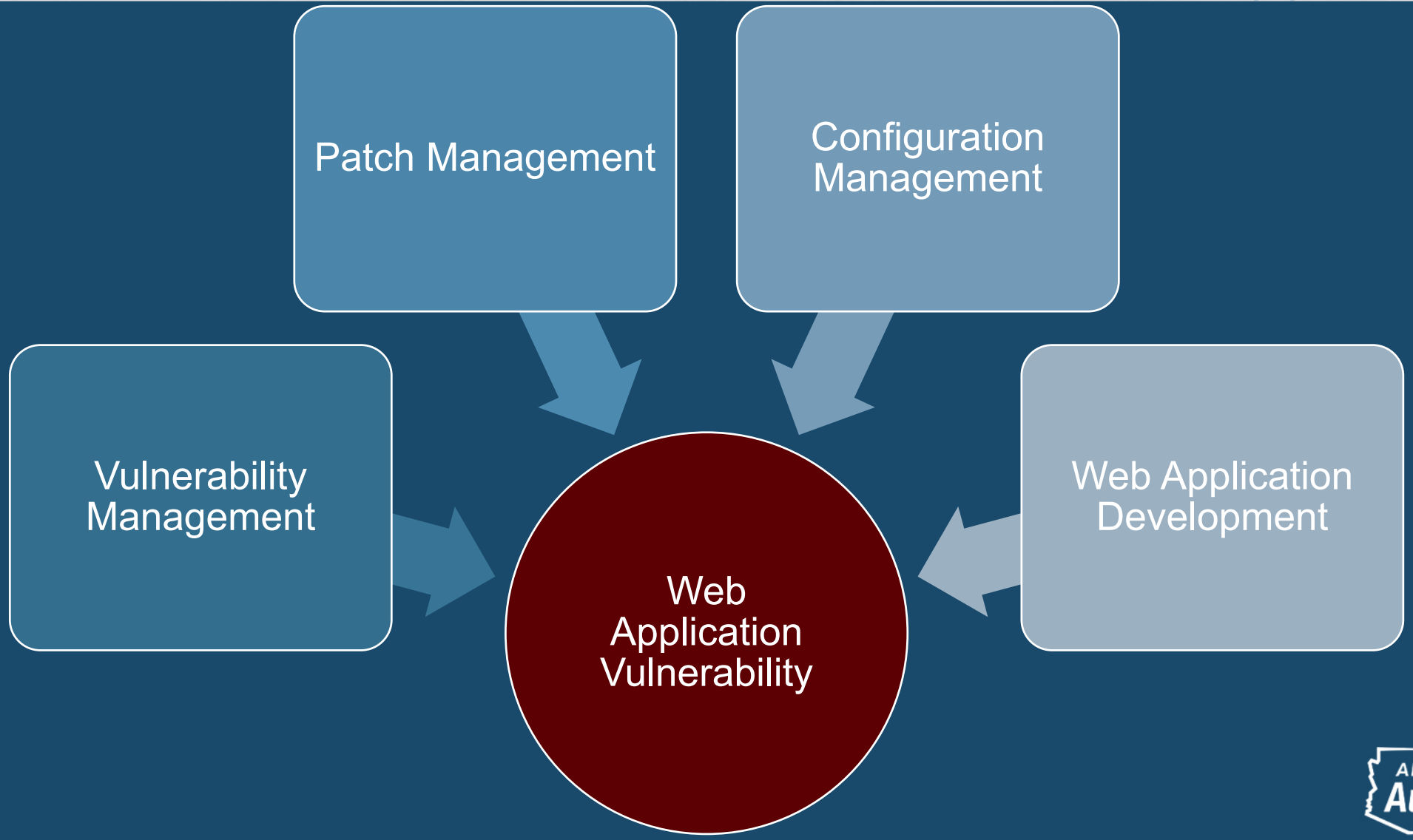
Universities' security controls slowed simulated attacks, but vulnerabilities allowed unauthorized access to some IT systems and sensitive data

However, auditors identified and exploited vulnerabilities to gain unauthorized access to some of the universities' IT systems and sensitive data contained in them, such as educational records, medical documents, and information about IT systems that could allow attackers to conduct further attacks. Specifically:

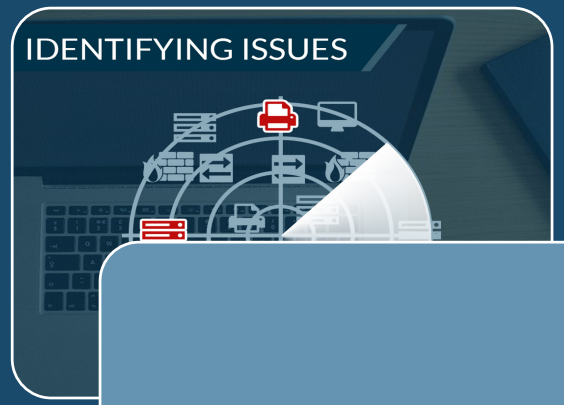
- **Sensitive data in web applications accessed—**Auditors identified vulnerabilities in web applications that could have allowed attackers to gain unauthorized access to sensitive data at ASU, NAU, and UA. At ASU, auditors exploited a vulnerability and obtained unauthorized access to sensitive data on hundreds of thousands of individuals, including names, addresses, phone numbers, grades, grade point averages, and other information. Auditors exploited the vulnerability after ASU had removed some controls to provide auditors more access to the web application. Removing controls is a common practice during penetration testing (see page 13 for information on penetration testing) to help penetration testers more quickly identify and exploit vulnerabilities during simulated attacks. Although ASU's controls would have slowed down an attacker, they would not necessarily have prevented an attacker from identifying the vulnerability and obtaining sensitive data. At NAU, auditors exploited a vulnerability and obtained unauthorized access to thousands of legal documents and unauthorized access to legally protected and sensitive data such as records related to medical issues.[14] At UA, auditors identified various vulnerabilities that could have provided access to sensitive information about some web applications and potentially compromised them.

  Auditors promptly notified the universities of the vulnerabilities. ASU and NAU immediately fixed their respective vulnerability and reported that they had reviewed their activity logs to confirm that there were no other instances of unauthorized access on the web applications during the time that their respective vulnerability existed (see pages 19 through 20 for more information about activity logs). In addition, UA staff reported that they immediately began to address their vulnerabilities.

Root Cause of Vulnerabilities



Vulnerability Management



Scanning

Penetration Testing

Remediation or Risk Acceptance

- Review entity's vulnerability management policies and procedures
- Conduct interviews about the entity's processes
- Perform vulnerability management remediation testwork

ARIZONA Auditor General
Making a Positive Difference

Missing vulnerability management components may have contributed to auditors' ability to identify and exploit vulnerabilities.

UA should ensure it conducts sufficient scans and penetration tests at appropriate intervals—UA scans only a portion of the IT systems on its network, does not scan web applications, and does not conduct penetration testing. Specifically:

- **UA's network scanning is limited, and it does not scan web applications**—Although UA has scanning policies and procedures, these policies and procedures do not require all of UA's IT systems on its network to be scanned, thus increasing the potential that vulnerabilities may not be detected. Additionally, a UA official reported that UA does not scan its web applications even though UA's policies and procedures require annual web application scanning. Further, IT standards and best practices recommend that organizations analyze scan results and share these results across the organization to help eliminate similar vulnerabilities in other IT systems, but a UA official reported that some of its scan results are not being analyzed and therefore cannot be shared across the university.

- **UA does not conduct penetration testing**—UA does not perform penetration testing for the IT systems on its network or its web applications. In addition, UA has not developed penetration testing policies and procedures that define which systems should be tested and the required time frames for doing so, as recommended by IT standards and best practices.

Developing and implementing revised policies and procedures for its vulnerability management process that include requirements and/or guidance for:

- Regularly scanning all of the IT systems on its network and its web applications, with specified scanning frequencies based on risk factors such as the amount and nature of sensitive data contained in certain IT systems and web applications, and the extent that scanning is used to assess whether individual units are identifying and addressing vulnerabilities, such as configuration and patch-related vulnerabilities;

- Analyzing scan results, including specifying time frames for conducting the reviews, and sharing these results across the university to help eliminate similar vulnerabilities in other IT systems;

- Conducting penetration testing at specified frequencies based on risk;

- Using a risk-based approach for conducting penetration testing for the IT systems on its network and its web applications, including specifying risk factors that should be considered for conducting this testing, the frequency at which risks will be assessed, and procedures for conducting penetration testing based on identified risks; and

- Helping to ensure all higher-risk web applications are tested within a specified time frame, such as determining whether to allocate additional resources for penetration testing or reducing the scope or frequency of penetration tests for some or all high-risk web applications.

Web App Development Auditing

- Review entity's policies and procedures
- Interview web app developers and make observations
- Tie-in web application testing results
- Summarize the entity's efforts

ARIZONA Auditor General
Making a Positive Difference

Web App Development Criteria

When developing web applications, organizations should:

1. Gather security requirements
2. Use up-to-date secure coding standards
3. Perform threat modeling during development
4. Review source code
5. Perform security testing before releasing a web application to the live environment
6. Provide role-based training to web application developers

Web App Development Best Practices

Open Web Application Security Project (OWASP)

- OWASP Testing Guide
- OWASP Code Review Guide
- OWASP Top 10 – 2017
- OWASP Top 10 Proactive Controls

National Institute of Standards and Technology (NIST)

- NIST Special Publication 800-53 revision 4 (AT-3)

ARIZONA Auditor General
Making a Positive Difference

Lack of some web app security components may have contributed to auditors' ability to identify and exploit vulnerabilities.

As previously discussed (see page 12), auditors were able to exploit vulnerabilities in some web applications to gain unauthorized access to sensitive data at ASU and NAU. In addition, auditors identified some common security vulnerabilities in all the web applications tested at all three universities that could have been used for further attacks. Although all three universities have developed policies and procedures for web application development, these policies and procedures lack some of the security-related components recommended by IT standards and best practices, which may have contributed to these vulnerabilities. Specifically:

Recommendations

Developing and implementing additional web application development policies and procedures that include the following IT standards and best practices:

- Gathering web application security requirements when developing web applications;
- Using secure coding standards when developing web applications;
- Requiring web application developers to be trained on developing secure software;
- Conducting threat modeling during web application development or security testing before releasing web applications to the live environment;
- Reviewing web application source code for web applications it develops internally before these web applications are released; and
- Performing security testing before web applications are released.

Securely Yours LLC

Auditing Web Applications

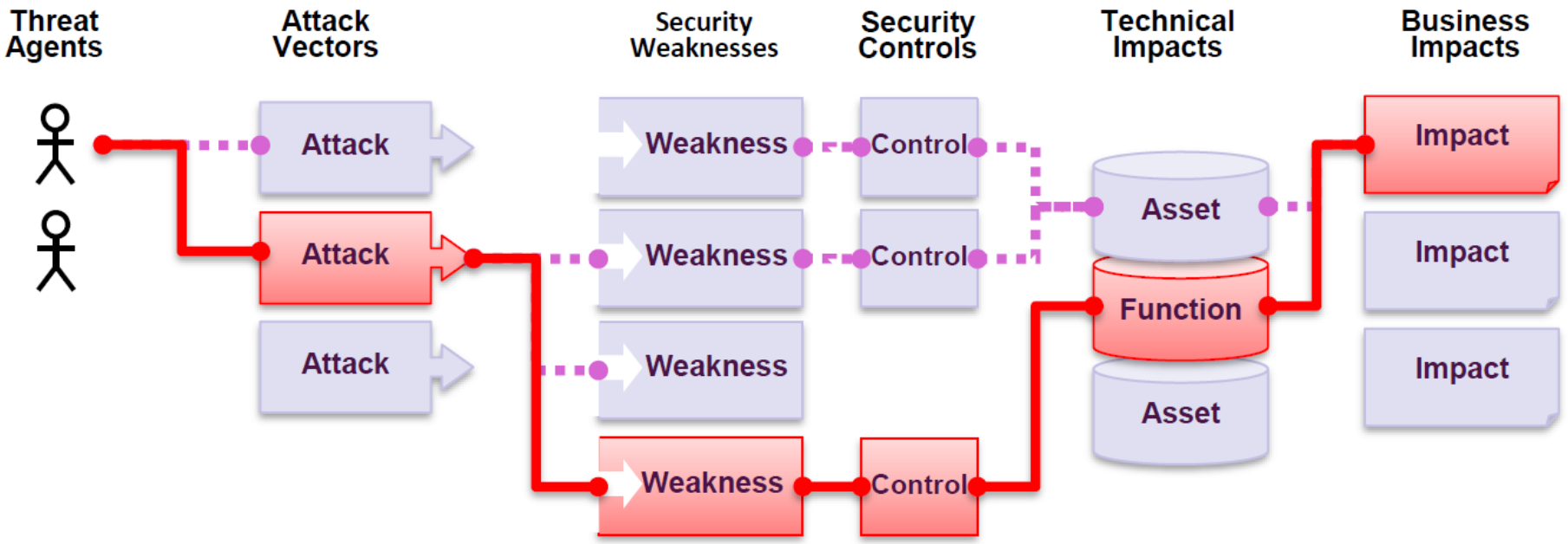
NSAA Information Technology Workshop and Conference
Grand Rapids
September 24, 2019

Sajay Rai, CPA, CISSP, CISM
sajayrai@securelyyoursllc.com
248-723-5224

Securely Yours LLC

Web Application Risks & Vulnerabilities

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."

The Web Application Security Gap

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."

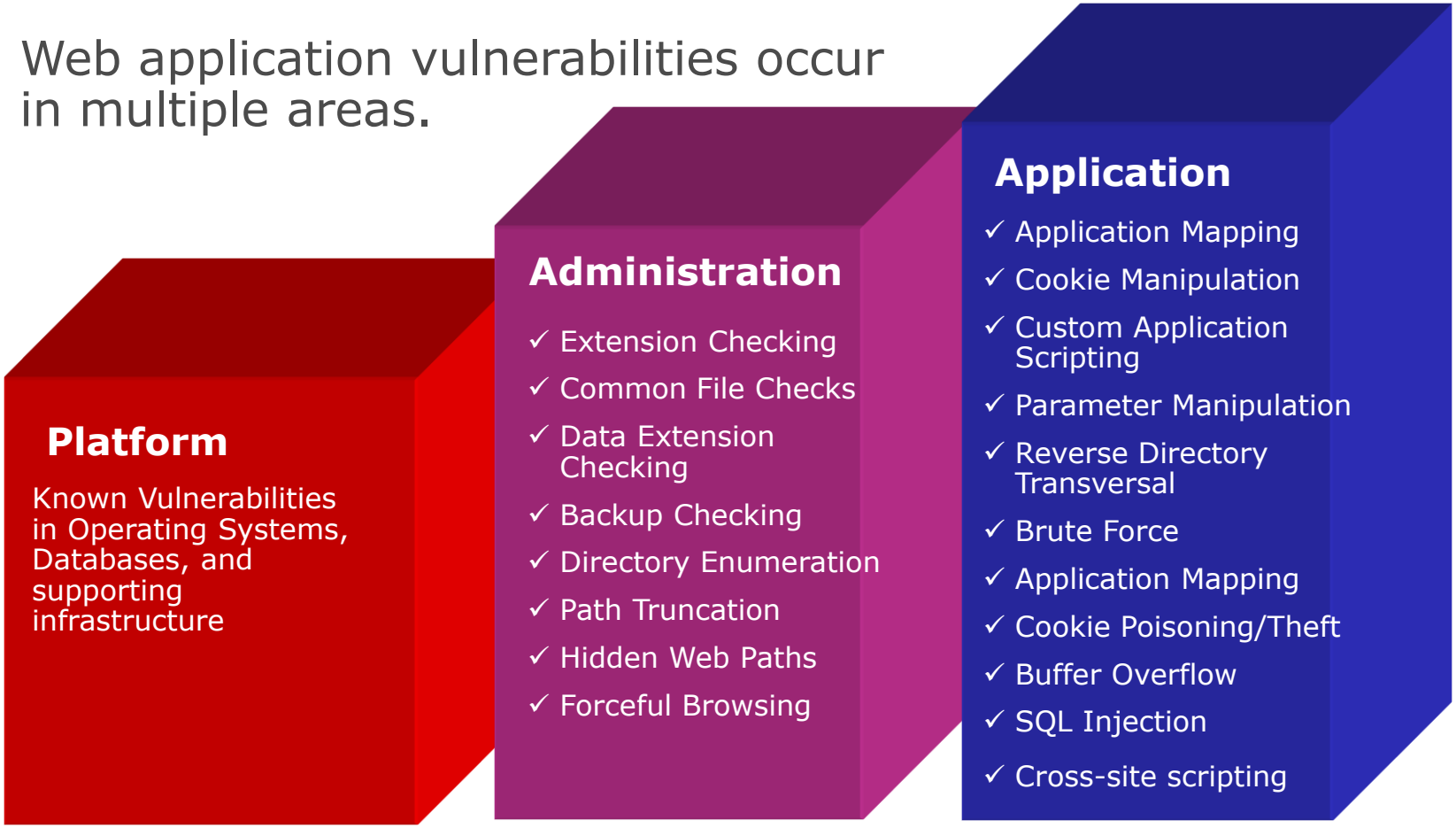
Web Application Vulnerabilities

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

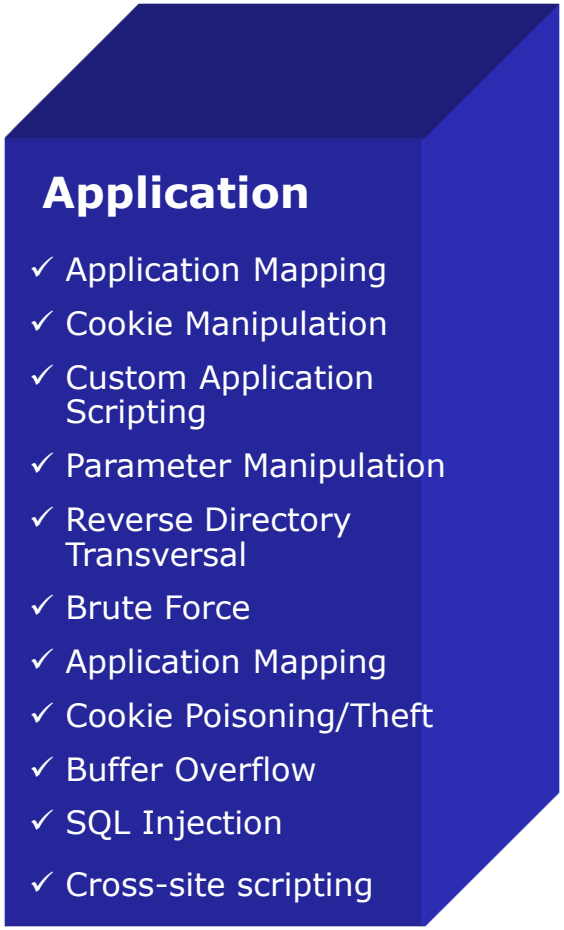
-Weinberg's Second Law

Web Application Vulnerabilities

Web application vulnerabilities occur in multiple areas.



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

Securely Yours LLC

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

OWASP – 1. Injection

A1:2017-Injection

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.

OWASP – 2. Broken Authentication

A2:2017-Broken Authentication

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.

OWASP – 3. Sensitive Data Exposure

A3:2017-Sensitive Data Exposure

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)

OWASP – 4. XML External Entities (XXE)

A4:2017-XML External Entities (XXE)

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'.
- 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.

OWASP – 5. Broken Access Control

A5:2017-Broken Access Control

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 access other users' accounts, view sensitive files, modify other users' data, change 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 users 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

A6:2017-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)

A7:2017-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.

OWASP – 8. Insecure Deserialization

A8:2017-Insecure Deserialization

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

A9:2017-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).

OWASP – 10. Insufficient Logging & Monitoring

A10:2017-Insufficient Logging & Monitoring

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

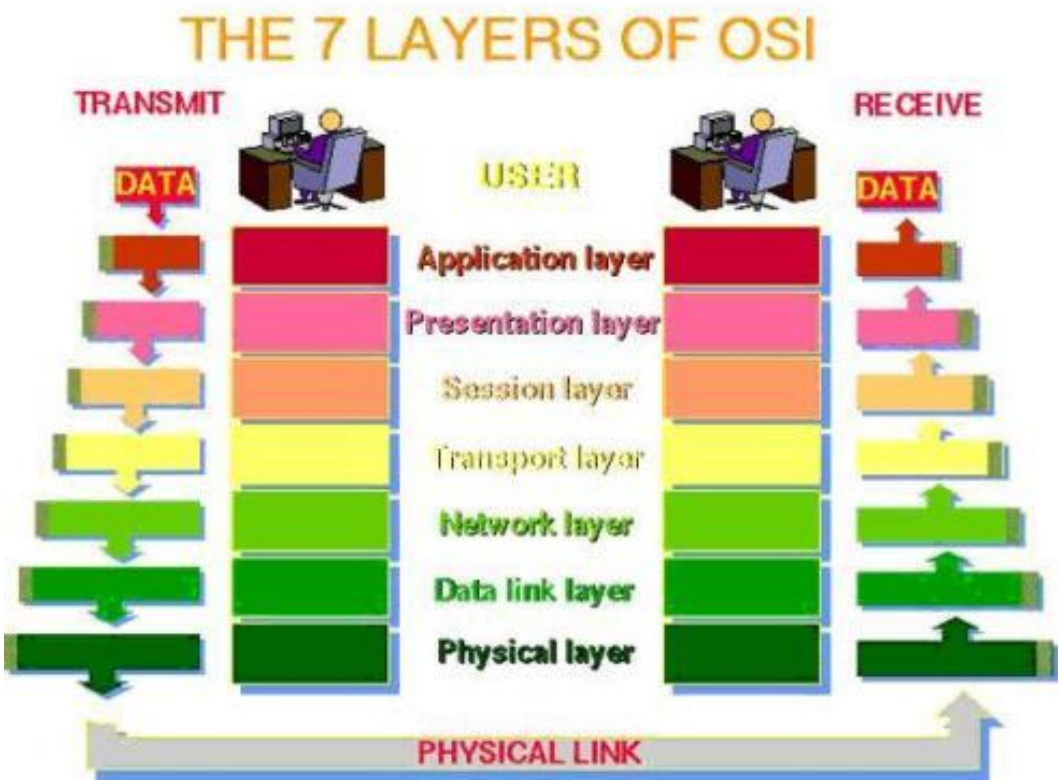
Securely Yours LLC

Securing Web Applications - WAF

How does the user transmit / receive data?

- WAF works at Layers 5-7

- Normal F/W works at Layer 3 or 4



How does the user transmit / receive data?

OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

Web Application Firewalls

Web Application Firewalls

- How does it work?

https://www.youtube.com/watch?v=p8CQcF_9280

Securely Yours LLC

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

Lab/Demo 1

Vulnerability Scan

1. The website (URL: http://52.38.65.32/) 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. Be prepared to discuss the experience with others

Lab/Demo 2

Source Code Scan

We have installed a free source code scan software called RIPS:

http://ec2-52-38-65-32.us-west-2.compute.amazonaws.com/rips-0.55/rips-0.55/

The source code is at:

/inetpub\wwwroot\index.php

Good version:

The source code is at:

/inetpub\wwwroot\index2.php