Eradication of Web Application Security Incidents

Objectives:

At the end of this episode, I will be able to:

Understand what eradication of Web Application security incidents is.

Explain what the eradication practices for web application security incidents that IH&R team members should be familiar with are.

Identify what the different types of Encoding Schemes are.

External Resources:

Eradication of Web Application Security Incidents

▪ SQL Injection Attacks - Following are different ways to eradicate SQL injection attacks:

```
o Limit the length of user input
o Monitor DB traffic using an IDS, WAF
o Disable commands like xp_cmdshell
o Run database service account with minimal rights
o Move extended stored procedures to an isolated server
o Use typesafe variables or functions
```

▪ Command Injection Attacks - Following are different ways to eradicate command injection attacks:

```
o Perform input validation
o Escape dangerous characters
o Use language-specific libraries
o Perform input and output encoding
o Use a safe API
o Structure requests
o Use parameterized SQL queries
```

▪ File Injection Attacks - Following are different ways to eradicate file injection attacks:

```
o Strongly validate user input
o Consider implementing a chroot jail
o PHP: Disable allow_url_fopen and allow_url_include in php.ini
o PHP: Disable register_globals and use E_STRICT to find uninitialized variables
o PHP: Ensure that all file and streams functions (stream_*) are carefully vetted
```

▪ LDAP Injection Attacks - Following are different ways to eradicate LDAP injection attacks:

```
o Perform type, pattern, and domain value validation on all input data
o Make LDAP filter as specific as possible
o Validate and restrict the amount of data returned to the user
o Implement tight access control on the data in the LDAP directory
o Perform dynamic testing and source code analysis
```

Following are some of the ways through which incident handlers can eradicate broken authentication and session management attacks:

```
▪ Use SSL for all authenticated parts of the application
▪ Verify whether all the users' identities and credentials are stored in a
hashed form
▪ Use multi-factor authentication
▪ Ensure registration, credential recovery, and API pathways are hardened
against account enumeration attacks by using the same messages for all outcomes
▪ Impose restrictions on the minimum length and complexity of the password
▪ Provide defined number of login attempts to the user and log the number of
failed login attempts
▪ System should not specify whether it was the username or password that was
entered incorrectly at the time of failed login attempt
▪ Log the date/time to user of each successful as well as failed login attempt
▪ While changing a password, system should require both new and old password
from the user
▪ Ensure Session IDs are not in the URL and are stored securely as well as
invalidated after logout, idle, and timeouts
```

Following are the different ways through which incident handlers can eradicate sensitive data exposure attacks:

```
▪ Identify sensitive data through classification as per organizational
policies or business needs
▪ Ensure that all the sensitive data is encrypted and is not stored
unnecessarily
▪ Do not use weak cryptographic algorithms
▪ Generate encryption keys offline and store them securely
▪ Use proper key management
▪ Ensure that caching is disabled for responses that contain sensitive data
```

Following are the different ways through which incident handlers can eradicate XML External Entity attacks:

```
▪ Parse the document with a securely configured parser
▪ Configure the XML processor to use local static document type
definition (DTD) and disable any declared DTD included in XML document
▪ Disable DOCTYPE tag or use input validation to block input containing it
▪ Avoid serialization of sensitive data
▪ Ensure all XML processors and libraries that are used by the web application
or on the underlying operating system are patched or upgraded
▪ Incorporate positive whitelisting of server-side input validation,
filtering, or sanitization to avoid hostile data within XML documents,
headers, or nodes
▪ Check whether XML or XSL file upload functionality authenticates incoming
XML using XSD validation
```

Following are different ways through which incident handlers can eradicate the broken access control attacks:

```
▪ Perform access control checks before redirecting the authorized user to
requested resource
▪ Provide session timeout mechanism
▪ Limit file permissions to authorized users
▪ Avoid client-side caching mechanism
▪ Remove session tokens on server side upon user logout
```

Following are different ways through which incident handlers can eradicate security misconfiguration attacks:

```
▪ Configure all security mechanisms and disable all unused services
▪ Setup roles, permissions, and accounts and disable all default accounts or
change their default passwords
▪ Scan for latest security vulnerabilities and apply the latest security patches
▪ Non-SSL requests to web pages should be redirected to the SSL page
▪ Set the 'secure' flag on all sensitive cookies
▪ Configure SSL provider to support only strong algorithms
▪ Ensure the certificate is valid, not expired, and matches all domains used
by the site
▪ Backend and other connections should also use SSL or other encryption
technologies
```

Following are the different ways through which incident handlers can eradicate XSS attacks:

```
▪ Ensure that the application performs proper validation of all the parameters
of a user request such as headers, cookies, query strings, form fields, and
hidden fields against a rigorous specification.

▪ Use testing tools extensively during the design phase to eliminate XSS
vulnerabilities in the application before it goes into use

▪ The attackers can use techniques like Remote File Inclusion and try to
inject a malicious file into web server to execute malicious scripts for data
manipulation or data theft. Incident handlers must deploy web application
firewalls (WAFs) into the web server to block the malicious files.
```

Following are different ways through which incident handlers can eradicate insecure deserialization attacks:

```
▪ Validate untrusted input which is to be serialized to ensure serialized
data contains only trusted classes
▪ Deserialization of trusted data must cross a trust boundary
▪ Avoid serialization for security-sensitive classes
▪ Filter untrusted serial data
▪ Duplicate Security Manager checks enforced in a class during serialization
and deserialization
▪ Understand the security permissions given to serialization and deserialization
```

Following are different ways through which incident handlers can eradicate web services attacks:

```
▪ The SOAP messages contain sensitive information, which attackers can use to
perform web service attacks and gain admin-level access. Configure WSDL Access
Control Permissions to grant or deny access to any type of WSDL-based SOAP
messages.

▪ Use document-centric authentication credentials that use SAML. This provides
a single SOAP message with single sign in method and eradicate various web
service attacks.

▪ Restrict web service attacks and implement scalable deployment by employing
multiple security credentials such as X.509 Cert, SAML assertions
and WS-Security.

▪ Deploy web services-capable firewalls capable of SOAP and ISAPI level
filtering.

▪ Configure firewalls/IDS systems for a web services anomaly and signature
detection.

▪ Configure firewalls/IDS systems to filter improper SOAP and XML syntax.

▪ Implement centralized in-line requests and responses schema validation.

▪ Block external references and use pre-fetched content when de-referencing
URLs.

▪ Maintain and update a secure repository of XML schemas.
```

Best practices for implementation of secure CAPTCHA include:

```
▪ Disable CAPTCHA reuse and present randomly distorted CAPTCHA image of
text to the user
▪ Warp individual letters so that OCR engines cannot recognize them
▪ Include random letters in the security code to avoid dictionary attacks
▪ Encrypt all communications between the website and the CAPTCHA system
▪ Use multiple fonts inside a CAPTCHA to increase the complexity of OCR
engines to solve the CAPTCHA
```

What are the different types of Encoding Schemes? -

```
▪ URL Encoding - Web browsers/web servers permit URLs to contain only the
printable characters of ASCII code that can be understood by them for
addressing. URL encoding is the process of converting URL into valid ASCII
format so that data can be safely transported over HTTP.
```

Several characters in this range have special meaning when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in hexadecimal such as:

```
o %3d =
o %0a New line
o %20 space


▪ HTML Encoding - Used to represent unusual characters so that they can be
safely combined within an HTML document. HTML encoding replaces unusual
characters with strings that HTML can recognize, while the various characters
define structure of the document.
```

It defines several HTML entities to represent particularly usual characters such as:

o & &
o < <
o > >

▪ Unicode Encoding - two types: 16-bit Unicode Encoding and UTF-8.

  o 16-bit Unicode Encoding - replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed in hexadecimal.

    • %u2215 /

  o UTF-8 - variable-length encoding standard which uses each byte expressed in hexadecimal and preceded by the % prefix.

    • %c2%a9 ©

▪ Base64 Encoding - represents any binary data using only printable ASCII characters.

▪ Hex Encoding - a transfer encoding in which each byte is converted to the 2-digit base-16 encoding of that byte (preserving leading zeroes), which is then usually encoded in ASCII. It is inefficient, but it is a simple, commonly-used way to represent binary data in plain text. For, example:

"Hex encoding test" = 48657820656e636f64696e6720746573740a