Detecting and Analyzing Web Application Security Incidents

Objectives:

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

Understand what the indicators of Web Application security incidents are.

Explain how IH&R team members can use both automated and manual detection methods to discover web application security incidents.

Identify what are the different manual detection methods that the IH&R team members can use, based on the type of web application security incident they are responding to.

External Resources:

Detecting and Analyzing Web Application Security Incidents

What are the Indicators of Web Application Security Incidents? -

```
▪ Unavailability of Website, Web service, or applications
▪ Alerts and notification from tools like WAF, SIEM, and IDS
▪ Leakage of sensitive data
▪ Redirection of URLs to incorrect sites
▪ Web page defacements
▪ Unusually slow network performance
▪ Frequent rebooting of the server
▪ Anomalies in log files (web server, application and database)
▪ Suspicious activities in user accounts like new processes, users, and jobs
▪ Error messages such as 400-500 errors
▪ Abnormal behavior in web applications such as pop-ups and spam messages
▪ Changes in web application files
▪ Search engines flag the website as insecure
▪ Complaints from customers
▪ Unusual outbound and inbound network traffic
▪ Inappropriate login attempts from various geographical locations
▪ Variation in the sizes of the HTTP request and response
▪ Getting too many requests for accessing the same file
```

Detecting Web Incidents: Automated Detection -

Tools such as WAF, IDS, and SIEM solutions can be used to detect web application incidents automatically. Alerts from Web Application Firewalls (WAF), IDS, and SIEM solutions help incident handlers in detecting and analyzing web incidents.

Detecting Web Incidents: Manual Detection - SQL Injection

The responders must examine log files of the IDS, web servers, and database to detect SQL injection attacks against a web application. For example, if an attacker tries to access the backend data of the web database by sending malicious SQL queries, then the log entry for the attacker’s request in the web server would look like the following:

```
▪ 01:22:19 192.22.31.40 HEAD GET /login.asp?username=blah’ or 1=1 #

▪ 01:22:19 192.22.31.40 HEAD GET /login.asp?username=blah’ or 1=1 --

▪ 01:22:19 192.22.31.40 HEAD GET /login.asp?username=blah’ or exec master..xp_cmdshell 'net user test testpass -
-
```

In this attack, the attacker has tried to bypass authentication of a web application using a SQL injection attack by using different scripts and codes for the web application username and password.

Incident responders can use various log analysis tools, such as Loggly, Logentries, GoAccess, and Splunk for analyzing the log files of IDS, web server, and database.

Incident responders can use regex search to find HTML tags, other SQL signature words, and their equivalents in web access logs to check for SQL injection attacks.

Detecting Web Incidents: Manual Detection using Regex - SQL Injection

The responders must develop and deploy rules in the IDS to detect regular expressions used in the SQL injection attack on a web server. Incident responders can use regex searches to detect SQL meta-characters.

```
▪ Regular expression for detection of SQL meta-characters

        /(\')|(\%27)|(\-\-)|(#)|(\%23)/ix
```

Incident responders must check for regular expressions, such as single-quote (') character in the web requests or its equivalent hex value to detect SQL injection attack. They must look for the double-dash (--) character, as it is not an HTML character and the web request does not perform any encoding for it.

For some SQL servers, the responders must detect hash (#) character and its equivalent hex.

The following text is a log entry derived from Snort:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS(msg: "SQL Injection – Paranoid"; flow:to_server, established; uricontent:".pl";pcre:"/(')|(%27)|(--)|(#)|(%23)/ix"; classtype:Web-application-attack; sid:9099; rev:5;)

The analysis of the log entry is as follows:

The 'alert' attribute represents that the log is an alert generated when the IDS solution detected the attack signature in an HTTP request.

The 'tcp' stands for use of TCP protocol, '$EXTERNAL_NET' indicates the external network's IP address and 'any' is for any source port.

The operator '->' allows for segregation of the destination from source.

The '$HTTP_SERVERS' is a variable attribute indicates the number of web servers an organization contains, and '$HTTP_PORTS' represents the common ports used for the HTTP traffic, such as 80 and 8080.

The 'msg:' denotes message, while the 'flow:to_server' attribute indicates direction of the traffic.

The attributes 'established' and 'uricontent:".pl"' indicate that the alert fires on only established TCP connection and Perl script-based URI content (applications) respectively.

```
▪ Modified Regular expression for detection of SQL meta-characters

    /((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/ix
```

The responders must use above regular expression to check the '=' sign from the user request or its hex value (%3D). The expression '[^\n] * ' indicates that it can some non-newline characters. After that, it checks for single-quote ('), double-dash (--), and semi-colon (;).

```
▪ Regular expression for typical SQL injection attack

    /\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix
```

The responders must use the above expression to detect zero or more alphanumeric and underscore characters that are involved in an attack. The single-quote (') character or its equivalent hex value are detected using the expression '((%27)|('))'. The remaining expression detects the word 'or' ('or', 'Or', 'oR', or 'OR') and their respective hex values.

```
▪ Regular expression for detecting SQL injection with the UNION keyword
```

Some attackers use UNION keywords in the SQL injection queries to enhance to attack to further exploitation. The responders must use the following expression for detecting the SQL queries that contain UNION keywords.

```
    /((\%27)|(\'))union/ix
```

This checks for the single-quote (') or its equivalent hex value, and then for the union keyword in the HTTP requests. The responders must develop similar expressions for keywords insert, update, select, delete, and drop to detect SQL injection attempts.

```
▪ Regular expression for detecting SQL injection attacks on a MS SQL Server
```

At any stage of the attack, if the attackers found that the web application is vulnerable to injection attacks and the database connected to the web server is MS SQL. They even can use most complex queries that contains stored procedures (sp) and extended procedures (xp).

They will try to use the extended procedures such as 'xp_cmdshel', 'xp_regread', and 'xp_regwrite' for executing the shell commands from the SQL server, and alter the registries.

```
    /exec(\s|\+)+(s|x)p\w+/ix
```

The responders must use the above expression to check 'exec' keyword, whitespaces (or their hex equivalent value), the letter combination sp or xp for stored procedures or extended procedures, and finally an alphanumeric or underscore character.

Detecting Web Incidents: Manual Detection - XSS Attacks

In a Cross Site Scripting (XSS) attack, the attackers exploit a vulnerability in the web application by injecting malicious script, such as JavaScript, HTML, or CSS markup in the web application code.

Common XSS attacks use HTML tags, such as <script></script>, <IMG> ,<INPUT>, and <BODY>.

The responders must use regex searches to find HTML tags, other XSS signature words, and their equivalents in web access logs to check for XSS attacks.

To avoid detection by security devices such as firewalls, IDS, IPS, and antivirus, the attackers encode the injection data by using the following obfuscation techniques:

```
▪ Hex Encoding - Attackers use hex values of the characters to bypass the
security mechanisms.

    o Normal XSS script: <script>alert("XSS")</script>

    o Hex encoded XSS script: <%3cscript%3ealert(”XSS”)%3c/script%3e>


▪ In-line Comment - Attackers use Inline comments in middle of attack strings
to bypass security mechanisms. Code with Inline Comment:
```

http://www.itpro.tv/accounts.php?id=/!union/+/!select/+1,2,concat(/ !table_name/)+FrOm/!information_schema/.tables/!WhErE/+/!TaBlE_sChEMa/+like+database()—

```
▪ Char Encoding/Double Encoding - Some of the Web Application
Firewalls (WAFs) decode the hex encoded input and filter it, preventing an
attack.
```

To bypass them, the attackers might double encode the input. In such cases, some WAFs may not decode the input a second time, inferring that the attackers successfully bypassed the WAF.

Code with Char Encoding:

http://www.itpro.tv/accounts.php?id=1%252f%252a/union%252f%252a/select %25f%252a/1,2,3%252f%252a/from%252f%252a/users--

```
▪ Toggle Case - Some applications block the lowercase SQL keyword. Attackers
can toggle the code to bypass them. Some firewalls contain the Regex Filter:

            /union\sselect/g
```

So, they may filter the suspicious code written in lower case letters.

Code with Toggle Case:

http://www.itpro.tv/accounts.php?id=1+UnIoN//**SeLecT**//1,2,3--

```
▪ White Space Manipulation - When attackers replace the keywords, some WAFs
may replace the keywords with white space. In such a case, the attackers
use "%0b" to eliminate the space and bypass the firewalls.
```

Code with White Space Manipulation:

http://www.itpro.tv/accounts.php?id=1+uni%0bon+se%0blect+1,2,3--

Detecting Web Incidents: Manual Detection using Regex - XSS Attacks

Incident responders can detect XSS attacks by examining the captured network traffic using network sniffers such as Wireshark. They must look for suspicious requests that contain XSS specific meta-characters, such as the single-quote (') or the double-dash (--) with any text inside and their hex equivalents.

Detecting Web Incidents: Manual Detection - Directory Traversal Attacks

Attackers perform a path traversal attack, (directory traversal attack) to access files and directories stored in the web root folder of systems and servers linked to the web application.

Attackers try to manipulate the path of root directory files with "dot-dot-slash (../)" sequence and access the application's source code, configuration files, and other crucial system files.

The attackers even use encoding mechanisms to bypass security measures implemented to restrict the "../" character sequence by replacing the sequence with a double encoding sequence "%252E%252E%252F"

The responders should check the log files and/or captured traffic and look for special characters in the data packets such as
file= ../../../../etc/ or ......\etc\passwd

By doing so, they can see the path traversal request received.

Detecting Web Incidents: Manual Detection using Regex - Directory Traversal Attacks

Incident responders must check for the directory traversal attacks performed by detecting the meta-characters using the regular expression. The regular expression mentioned below checks for logs that may contain directory path traversal specific meta-characters, such as '../'

You can search the logs for ../ using the following regex:

```
        ((\.|%2E)(\.|%2E)(\/|%2F|\\|%5C))
```

The following regular expression can also help you in finding the directory traversal attacks:

```
        ((\.\.\\)|(\.\.\/))
```

Detecting Web Incidents: Manual Detection - Dictionary Attacks

Incident responders can detect dictionary attacks by monitoring status codes of the application across the logs of the web servers. When the status code changes, the attack is successful.

Detecting Web Incidents: Manual Detection - Stored Cross Site Script Attacks

In stored cross site script attacks, the attackers inject malicious code into the input sections, which web applications store permanently. Whenever a user tries to navigate to the web site or applications, the browser executes the injected code and performs the attack.

Incident responders must perform detailed log analysis to find the malicious code injected in the input fields.

NOTE: It is difficult to find the malicious injected code using regex as the application encodes it before processing and validation.

For the attack to happen, the attacker should have an account in the web application

1. Attacker logs into website http://www.itpro.tv
2. The itprotv index page appears
3. Attacker clicks on the blog page (/blog.aspx)
4. The blog page appears, the attacker posts something (stored cross site script) in an input field
5. Victim logs into the itprotv website
6. Victim clicks on the blog page
7. The victim clicks on the script and is redirected to another website, baditprotv.com. The URL carries the cookie value of the victim's active session. The victim then is redirected to the blog.aspx page.
8. The attacker now has the cookie value, so they edit the cookie of their active session and reload the webpage
9. The attacker has now gained control over the session of the victim

Detecting Web Incidents: Manual Detection - DoS/DDoS Attacks

Indicators of an active DDoS attack include slower network communications due to unavailability of the bandwidth, decrease in performance, and unavailable services. Security tools such as IDS, IPS, and firewalls can also alert administrators about possible DDoS attacks.

The incident responders can use the "netstat -an" command from a command prompt in a Windows system to check if the same IP is constantly making attempts to connect or access the application or connected ports and if the connections show TIME_WAIT.

Responders can also use log viewers such as Apache Logs Viewer, IIS Log Analyzer, and anomaly detection tools such as Loggly to analyze logs of various servers and identify anomalies. They can use tools such as tcpdump, tshark, Snort, Argus, ntop, AGURI, and MRTG to analyze network traffic. The responders can detect DoS/DDoS attack on the web servers using the Apache Logs Viewer.

With Apache Logs Viewer, the responders can easily filter and analyze Apache/IIS/nginx log files. Apache Logs Viewer displays the logs count by IP address and errors messages (4** -5** status codes).

Detecting Web Incidents: Manual Detection - Potentially Malicious Elements within HTML

If the client browser interprets scripts embedded within HTML content which is enabled by default then the attacker can insert multiple script tags such as <SCRIPT>, <OBJECT>, <APPLET>, or <EMBED> into the website and the web browser's JavaScript engine will execute it.

Incident responders can detect such attacks by examining the HTML tags like <FK>, <LI>, <BR>, <DIV>, and background-image. The <TITLE> tag should also be examined as it is the most vulnerable tag if it is generated dynamically.

Attackers can also perform a framing attack by using the <IFRAME> tag of HTML to exploit the frame support characteristics of a web browser. This type of attack involves a web page integrated into another web page using iFrame elements of HTML.

An attacker exploits iFrame functionality used in the target website, embeds his/her malicious web page, and uses clickjacking to steal users' sensitive information. Incident responders can detect framing attacks by examining the <IFRAME> HTML tag.

In HTML, field values are stored as hidden fields, which are not rendered to the screen by the browser but are collected and submitted as parameters during form submissions.

Attackers examine the HTML code of the page and change the hidden field values in order to change post requests to the server. Incident responders should examine hidden field values to check if there are changes.

```
<input type="hidden" id="item_1" name="Price" value="1000.00">
```

Detecting Web Incidents: Manual Detection - Malicious Elements in Common Web File Types

Attackers execute arbitrary code by uploading malicious files that are automatically executed by the recipient.

Web pages with extensions such as .php, and .aspx when uploaded are often a utomatically executed.

Incident responders need to inspect the code within the malicious files that were uploaded by the attackers.

Detecting Web Incidents: Manual Detection - RFI Attacks

Remote File Injection (RFI) is a technique that targets underlying web application vulnerabilities and launch attacks from a remote server.

A successful execution of this attack includes revealing sensitive information, compromised server, and content modification.

Improper input handling and application misconfiguration are the two main vulnerabilities that are exploited by the attackers to perform remote file inclusion attacks.

In order to detect these types of attacks, incident responders need to use regular expression to search for a signature such as "(ht|f)tps?://" within malicious parameter payloads.

Using this approach URL can be tested for malicious payloads and remote file injection attack can be detected.

Some of the methods used by the attackers to perform remote file injection attack and their respective detection techniques are:

```
▪ Using URLs Containing IP Address

▪ Using PHP Functions - the attacker can trick the web application into
including data from external site by using internal PHP keyword functions
such as “include()”

▪ Using URLs with Appended Question Mark(s) - an attacker adds question marks
at the end of the RFI payload. By appending question marks the remaining PHP
code is treated as a parameter to the RFI included code. As a result, the RFI
code ignores the legitimate code and executes attacker injected code.

▪ Using Off-site URLs - an attacker specifies hostname/domain in the parameter
payload and mismatch it with host header data to perform an attack. If the
domain name and the host header are not the same, then it would indicate an
RFI attack.
```

Detecting Web Incidents: Manual Detection - LFI Attacks

Local File Injection (LFI) is an attack where an attacker exploits vulnerable
inclusion procedures implemented in the web application.

In this attack, an attacker can trick a web application to execute a malicious
code present locally on the web server.

Incident responders can detect LFI attacks by analyzing Apache web server logs.
A web server log file stores all the actions and events that take place during
the runtime of an application or service in a file called access.log

The default location of this file in Linux systems is: /var/log/apache2/access.log

In the access.log file, the incident responder can search for the requests that
tried to read “/etc/passwd” which indicate an LFI attack, using the below command:

```
cat access.log.1 | grep “../../../../etc/passwd”
```

Detecting Web Incidents: Manual Detection - Watering Hole Attacks

Attackers perform watering hole attacks by infecting frequently visited web
applications with malware by injecting or modifying HTML or JavaScript code.

Incident handlers can detect this attack by using “sequential pattern mining” in
order to detect the web applications that are frequently accessed. The main
objective of using sequential pattern mining is to determine the sequential
patterns in time-ordered data.

The quality of a sequential pattern is typically measured by its support and
confidence.

Phishing sites are detected by determining the low support and high confidence
sequential pattern.

Incident handlers need to perform data preprocessing before analyzing and
detecting the watering hole attack.

Data preprocessing is performed on outbound network traffic data that is present
in the firewall and proxy server logs that are used to store the internal user’s
web access activities.

Incident handlers need to follow the steps discussed below to perform data
preprocessing:

```
▪ Host Name Normalization - done in order to create a stable list of
identifiers for appropriate host names, instead of tracing different variants
of a single entity. For example, blog.itpro.tv is normalized to itpro.tv.
However, there are some cases where certain host names are set un-normalized
like the subdomains of a popular dynamic DNS site.

▪ Filtering Invalid Host Names - Host names often consist of invalid characters
that should be removed so as to account for data collection errors.

▪ Identifying Unpopular Domains - analysis aims to identify the links with
domains that are accessed by only a few number of users.

▪ User-Specific Sessionization - refers to a data processing process that is
used to analyze the user activities within a certain time period. Incident
handlers need to sessionize proxy logs to create time-ordered domain sequences
to perform analysis on sequential patterns afterwards. Incident handlers need
to consider a timeout threshold of 60 seconds as generally watering hole
attacks are performed within a short span of time.
```

The output of data preprocessing is the sequential patterns of the URLs accessed by the users.