jamk.fi

Web Application Security

Home Exam

Mikael Romanov
K1521

February 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma
Kyberturvallisuus

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Content

1 Question 1: Web Browsers and Content

(11 points)

A. (1 point) What is an "origin" and what it attempts to achieve?

It is a header that indicates where the request originated from. It only includes the server name.

HTTP IP/DOMAIN NAME 8080
<scheme> "://" <hostname> [":" <port>]

The origin header is used to describe security context that caused user agent to initiate a HTTP request

B. (2 points) What is the Same-origin -policy, and how the browsers enforce it in

Same-origin policy is for isolating documents that can be malicious and cause harm. It is a critical security measure, that defines how to restrict scripts and documents from one origin to interact with the resources of another origin. e.g. same-origin policy allows inter-origin HTTP request with GET & POST, but denies PUT & DELETE requests. The origins can send custom HTTP headers when they send requests to themselves, but not when sending to other origins.

1. cookies?
   Domain only. (e.g. login.nakki.fi can only set cookies for all of .nakki.fi but not for other sites e.g. google.com or Top Level Domain)
2. DOM nodes?
   Same-origin is calculated with:
   - Scheme: https
   - Host: nakki
   - Port: 80

   Two resources are considered as same origin if all of the above values are same. e.g. http://www.nakki.fi:80 and http://www.juuheliks.nakki.fi:80 are not same origin.

   Dom restriction can be bypassed with (window.document.domain variable manipulation, proxy or cross document messaging)
3. JavaScript?
   Same rules go for Javascript as DOM rules. (Scheme, host and port must be same) All though the SOP rules prevent AJAX request to different origin, there are possible ways to bypass them by e.g. proxy server script or JSONP.

C. (5 points) What are XSS and CSRF attacks? What is the difference between the two?

XSS

Cross-site scripting is an injection type of attack where malicious scripts are injected into trusted web applications. XSS attacks happen when attacker uses a web application to send malicious scripts. The end user has no way of knowing if the script can be trusted. The scripts can steal cookies, session tokens and other sensitive information. The scripts can also rewrite the whole page by e.g. defacing it. Most widely abused way of using injections is by writing javascript to the payloads. There are plenty of attack vectors:

- <script> <body> <img> <iframe> <input>
  <link> <table> <object>

Types of attacks:

- Stored XSS
  - Stored XSS is the most damaging type of the XSS injections, because the payloads are stored permanently on the web applications database. The usual way to exploit stored XSS is comment field. The payload will be loaded once victim navigates to the infected site, which means that the script is executed without victim knowledge.
- Reflected XSS
  - In reflected XSS attacker send a payload to the web application via request and it reflects back as a HTTP response with the same payload that the request had. Reflected XSS is achieved by gaining victim to click the bait with social engineering or email phishing to make request to the server which contains the XSS payload. The reflected script executes in the browser without victim knowledge. Usually reflected XSS clickbaits are sent via social media e.g. Facebook
- DOM-based XSS
  - Web application reads data from client-side and outputs it in the browser. When the data is handled incorrectly, attacker can craft malicious scripts which are stored in DOM. The attacks are executed only when the data is read back from DOM. Because DOM-based XSS is a client-side attack, it allows also attacking the client without of the web applications knowledge. e.g. logs will never see the attack happening.

CSRF

Cross-Site Request Forgery (CSRF) is an attack where the attacker tricks a victim to execute unwanted actions on a web application. The attacks target state-changing requests so he has no way to see the response for the forged request. There are two main parts for a successful attack. First part usually is to trick the victim to click link or loading page. The way to achieve first part is to use form of social engineering. It is done by tricking users curiosity to invoke a need to the malicious link. Second part of the attack is for sending a specially crafted request to the victim browser. The request needs to look legimate to the target web application e.g. bank. Attacker will send values he needs to complete his task, including cookies that the victim has

used with the web application. All requests with cookies or HTTP credentials are considered legimate from the aspect of the web application even if the commands are done by the hacker. When the cookies are included in the requests, the session will be seamless experience and will not need reauthentication. The idea of CSRF is to perform state changing request that e.g. transfer funds or changes email address or compromise whole web application if the victim is an admin.

Difference between XSS & CSRF

XSS doesn't need authenticated session and it can used when the web application doesn't validate or escape input. The attacker can input request via parameter mentioned above and write malicious scripts inside them with json. The attacker can see the feedback of the attack instantly. e.g. onerror=alert('bwahaha'). XSS is javascript based execution.

CSRF needs an authenticated session and needs web application that trusts the user and browser. The attacks are done via requests and there is no way to what the response is. Also CSRF needs social engineering to work. CSRF is about auth_tokens whereas the web application trusts that the user is the user

D. (3 points) Describe three ways (in total) to mitigate XSS and CSRF attacks.

1. **Escaping**
   To prevent XSS vulnerabilities the application need to use escaping on user inputs. This is achieved by escaping the data the application receives which ensures that the data is secure before rendering it to end users. It is a way to censor data by disallowing characters such as <> -- ' " being from rendering. These type of characters can cause plenty of harm. The usual way of escaping is to escape all URL, Javascript and HTML entities.

2. **Validating Input**
   The second way is to ensure that the application renders correct data and prevents malicious data from doing harm in the applications recourses. The method that migrated from SQL is to whitelist valid characters, because blacklisting will turn in an endless battle of whack-o-mole. When the good characters are whitelisted all the invalid character request will be refused. Input validation is helpful when using forms and fields where the user could add special characters.

3. **Sanitizing**
   Sanitizing data should be used with escaping and validating input. The data sanitizing will scrub data and replace unaccepted characters to an accepted format. Sanitizing ensures that the data received will cause no harm to users nor database.

2 Question 2: Injections

(8 points)

A. (2 points) Briefly describe what an SQL injection is, and what types there are.

SQL injection is an attack where the attacker alters SQL statements to malicious SQL statements that are used in the web application that uses SQL-based database to achieve a goal. There are many attack vectors that can used e.g. Authentication By-pass (admin privilege without username or password), Compromised Data Integrity (altering contents of the database) and Remote Command Execution (executing commands through compromised host)

- Error based SQL injections
    - send errors to user because debugging is on the server
- Union based SQL injections
    - database will display results of the search query
- Blind SQL injections
    - with tools like sqlmap that tries attacks against different types of SQL databases

B. (4 points) How do stored procedures and prepared statements prevent/mitigate injections.

Prepared statement

The prepared statement with parameterized queries force developer to define all of the SQL code and pass each parameter query later. This distinguishes databases code and data which means it doesn't matter what user inputs are committed. The prepared statement is for ensuring that the attacker can't change the query. Parameterized queries means that when the e.g. username is '1'='1 it would be looked at as what it actually is a 'string' and the username is '1'='1. The problem with prepared statement is that it can impact the performance when an attack is occurred. That is why the data should be validated and use escape for all user inputs.

Stored procedures

Stored procedures will have same effect as parameterized queries. Stored procedures require the developer to build a database where all the parameters are automatically parameterized. In stored procedures the SQL code is defined and stored in the database and then called from the application. As well as the prepared statement stored procedures should use proper input validation and escaping.

C. (2 points) Describe two other injection types.

1. XPATH Injection
   XPATH injection is an attack, where attacker exploits XML Path queries by sending malformed information to a web application that uses XPATH queries from user supplied data. This is achieved, if the web application embeds unprotected data in XPATH query, which means the query can be altered and it's not parsed in the originally intended manner. This means that the application can return data that no one is supposed to see or bypassing authentication and extracting information about how the XML data is structured. XPATH injection is more dangerous than SQL injections, because XPATH lacks access control. XPATH allows querying through complete database.

2. CRLF Injection

   Carriage Return & Line Feed (CRLF) injection is HTTP response splitting which can lead to XSS (e.g. web cache poisoning or hijacking client session). CRLF injection is a software coding vulnerability. The attacker can gain control over HTTP response contents, if he can inject his own CRLF sequence (e.g. %0d%a%0d%a'maliciousseqence') into a HTTP stream. When the attacker injects a CRLF sequence where it isn't expected it can cause data input that is unsanitized or not neutralized correctly. The CRLF sequence can cause unexpected and harmful actions to the web application and compromise the applications integrity.

3 Question 3: Transport Layer Security

(12 points)

A. (1 point) What are the (cryptographic) goals of TLS?

TLS goals are to provide data integrity, privacy, digital certificates, identification, confidentiality and authentication.

B. (8 points) Describe (briefly) how the TLS handshake works.

In TLS handshake the server and client exchange random number and special number which is called Pre-master-secret. Pre-master key and the random numbers are combined with additional data permitting. Both client and server generate shared secret 'Master Secret', which is used to on the server ja client to generate a MAC secret. MAC secret is the session key and it is used for encryption.

TLS Handshake

First client sends 'Client hello' message with client random value and supported ciphers (DH, DHE, ECDH, ECDHE, RSA). Server then responds with 'Server hello' with server random value. Server sends certificate for authentication and may request a certificate from the client. Server send a 'Server hello done' message then. If the certificate is requested, the client then sends it. After the certificates are exchanged, the client creates a random Pre-Master Secret and encrypts it with the server certificates public key, then sends it to the server. After the server receives the Pre-Master Secret, the client and server each generate the Master Secret and session keys which are based on the Pre-Master Secret.

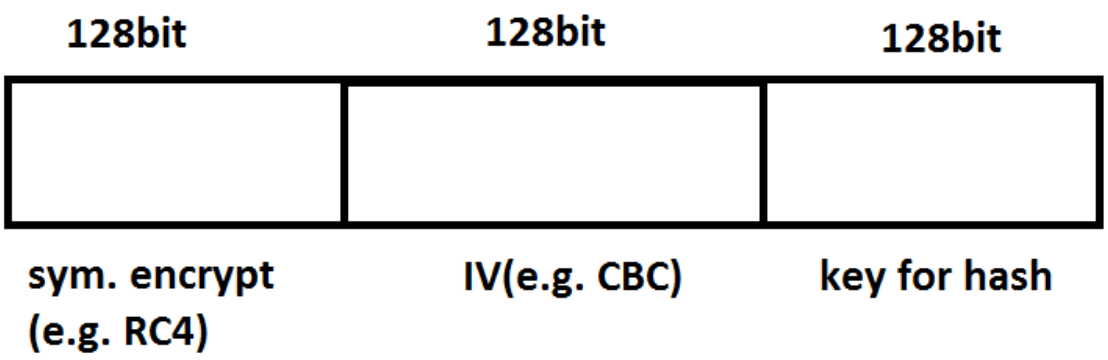
master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)[0..47];

After the Master Secrets are generated, the client sends "Change cipher spec" notification to the server for indication that the client will start using the new session key for hashing and encryption. Client send a "Client finished" message. The server receives the "Change cipher spec" and switches security state to symmetric encryption using the session key. Server then sends "Server finished" message to the client. A secured channel has been established and the client and server can send encrypted messages.

C. (3 points) What is the Master Secret, and where it is used and how?

The Master Secret is generated from a pseudo random function which is:
master_secret = PRF(pre_master_secret, secret, ClientHello.random + ServerHello.random)[0..47];

Master secret is 48bytes long key:



The master secret is used to generate symmetric keys and message authentication code (MAC) keys. Master Secret is used in TLS after Pre-Master Secret and client.random and server.random numbers are gained.

4 Question 4: Unsafe serialization

(11 points)

A. (2 points) What does serialization mean, and how can it be unsafe?

Serialization is a process where an object is converted into a stream of bytes in order to transmit the object into a database, file or in to memory or to store the object. The object that is serialized into a stream carries the data, but also information about the objects type (version, assembly name and culture). The format in which object is serialized into can be binary or structured text (e.g. XML)
When serializing user-controlled data, could lead to data being compromised. This can happen when the serialized objects do not validate or check untrusted input before deserializing it. All applications that accept serialized Java objects can provide a way to gain complete remote control of the server.

B. (4 points) Find a case where unsafe serialization was used to attack a web application

Apache struts bug
https://arstechnica.com/information-technology/2017/09/exploit-goes-public-for-severe-bug-affecting-high-impact-sites/
Apache Struts is software kit for crating Java based web applications. It can be used to build online shops and forums on the fly. The vulnerability affected web applications that used Apache Struts REST communication plugin. REST process doesn't specify how represent or encode the data you send or receive. The vulnerability was

caused by the way that the framework deserialized user input. Booby-trapped XML could be fed to the Struts server so the attacker can embed commands that was supposed to be plain data. Struts bug is a RCE where e.g. a innocent looking product search could trigger the server to leak data.

C. (2 points) Where would you encounter a serialization vulnerability? Present both an obvious and subtle case.

The most common case where serialization vulnerability is encountered is JAVA applications that use serialization.
Subtle case is when a coding library is used for serialiazation e.g. http://NET

D. (3 points) Describe how this vulnerability can be mitigated

The vulnerability can be mitigated through both white & blacklisting, but it isn't a long term solution. By blacklisting vulnerable classes it often breaks the whole application. Whitelisting by overriding the ObjectStream with a SecureObjectStream which is used to validate classes that are expected by the application. Only to deserialize trusted data. Removing all gadgets, because they can contain vulnerabilities (e.g. CommonsCollections). Lastly use of defensive deserialization and encryption. But the best way is to turn off deserialization.