Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Hacking Web Sites OWASP Top 10

Emmanuel Benoist

Fall Term 2021/2022

▶ Computer Science Division

Web Security: Overview of other security risks

- OWASP Top 10
- Top 10 Web Security Risks
- A4 XML External Entities
- A6 - Security Misconfiguration
- A8 Insecure Deserialization
- A9 - Using components with known vulnerabilities
- A10 Insufficient Logging and Monitoring
- Cross Site Request Forgery CSRF
  - Widespead vulnerability
  - Vulnerability?
  - Attacks using CSRF
  - Protection
  - CSRF prevention without a synchronizer token
- Conclusion

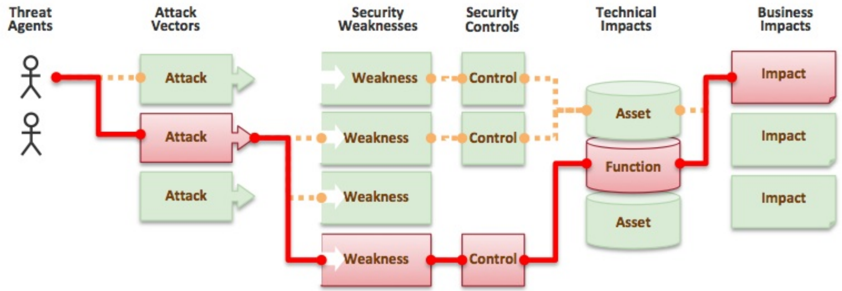
OWASP Top 10

OWASP Top 10

- **10 most critical security risks for web applications**
- **Goal**
  - Raise awareness of people about application security
- **Based on real examples**
  - 8 datasets from 7 firms specialized in application security
  - 500'000 vulnerabilities, thousands of applications
  - Sorted on the prevalence of data in combination with risks (exploitability, detectability and impact estimation)

What are application security risks?

- **Attackers can use many different paths to do harm**



Top 10 Web Security Risks

OWASP Top 10

- **Presents the 10 most critical web application security risks**
  - Produced by the Open Web Application Security Project (OWASP)
  - Available on line `www.owasp.org`
  - Updated in 2017
- **Not Exhaustive**
  - hundreds of other issues occure in Web Security
  - But it is foccused on the most critical ones

OWASP Top 10

Version 2017

- **A1:2017 - Injection** Seen
- **A2:2017 - Broken Authentication** Seen
- **A3:2017 - Sensitive Data Exposure** Seen
- **A4:2017 - XML External Entities (XXE)**
- **A5:2017 - Broken Access Control**
  Seen
- **A6:2017 - Security Misconfiguration**
- **A7:2017 - Cross-Site Scripting (XSS)** Seen
- **A8:2017 - Insecure Deserialization**
- **A9:2017 - Using components with known vulnerabilities**
- **A10:2017 - Insufficient Logging and Monitoring**

A4 XML External Entities

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.

- **XML processors**
  - Older and poorly configured
  - evaluate external entity references within XML documents.
- **External entities used to:**
  - disclose internal files using the file URI handler
  - disclose internal file shares,
  - internal port scanning,
  - remote code execution,
  - and denial of service attacks.

Attack Vector

- **Vulnerable XML Processor**
  - Attacker can upload XML
  - include hostile content in an XML document
- **Exploits**
  - vulnerable code,
  - dependencies,
  - integrations.

Security Weakness

- **Older XML-Processors allow specification of an external entity**
  - External Entity = URI that is dereferenced and evaluated in XML processing
- **Source Code Analysis Tools**
  - Static Application Security Testing (SAST)
  - Analyse source to find flaws
  - Search for dependencies and configuration
- **Vulnerability Scanning Tools**
  - Dynamic Application Security Testing
  - Test the web site from the outside
  - require additional manual steps to detect this issue
- **Detectability is difficult**
  - Manual testers need to be trained how to test for XXE
  - Not commonly tested as of 2017

Example 1

- **The attacker uploads a XML file on the server**
  - The parsing may occur anywhere in the code, very deeply.
  - The easiest way is to upload a file and see.
- **Upload file:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo[
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///etc/passwd">]>
  <foo>&xxe;</foo>
```

- **Parser accesses the file** `/etc/passwd` **and includes it inside the document.**

Example 2

- **Suppose we change the ENTITY definition**

  ```
  <!ENTITY xxe SYSTEM "https://192.168.1.1/private">]>
  ```
- **The attacker can test if a resource exists on a local server**
  - Can be used to scan the internal network
  - Can open access to some resources
  - Allow to send requests to servers

Example 3

- **Can be used for denial-of-service**
- **Access a endless file**

```
<!ENTITY xxe SYSTEM "file:///dev/random">]>
```

Your application is vulnerable

- **If the application accepts XML directly or XML uploads**
- **If XML processor has DTDs enabled**
  - DTD = Document Type Definition
  - Can be in an application or a SOAP based web service
  - Disabling DTD is different for each system
- **If your application uses SAML (for SSO or federated security)**
  - SAML uses XML for identity assertions, it may be vulnerable.
- **The application uses SOAP prior to version 1.2**
  - susceptible to XXE attacks if XML entities are being passed to the SOAP framework.

How to prevent?

- **Prefere JSON to XML (less complex)**
- **Update SOAP to SOAP 1.2 or higher**
- **Disable external entity and DTD**
- **Validate input**
  - prefere “white listing” against “black listing”
- **Validate uploaded XML and XSL files with XSD validation**
- **Code review necessary**
  - SAST tools may help
  - Do not replace manual code review

A6 - Security Misconfiguration

A6 - Security Misconfiguration

- **Process for keeping software up-to-date**
  - OS
  - Web /App Server
  - DBMS
- **Is everything unnecessary disabled?**
  - ports, services, pages, accounts, priviledges
- **Have been default account passwords changed or disabled?**
  - Before the first connection to the net
- **Is your error handling set to prevent informative messages?**
  - Stack traces
  - SQL errors
- **Are the security settings in your development frameworks understood and configured properly**
  - Struts, JSF, Spring, ASP.NET
  - Libraries
- **Repeatable process is required**

Security Misconfiguration (Cont.)

- **Application relies on a framework (JSF, Struts, Spring)**
    - A flaw is found in the framework
    - An update is released
    - You don't install the update (sometimes you can't)
    - Attackers will use the known vulnerability
- **The application has a default admin page with default pwd**
    - You forget to remove the tool and to change the pwd
    - Attack logs in using default value

Security Misconfiguration (Cont.)

- **Directory listing is not disabled**
  - Attackers can browse directories and find any file.
  - They download Java .class files and uncompile them, then know your code.
- **Access to “configuration” files not properly restricted**
  - Config files inside the *DocumentRoot*.

How to determine if you are vulnerable

- **If you did nothing: you are vulnerable**
  Almost no application is secure "out of the box"
- **Secure configuration of the server should be documented**
  Regularly updated
  You should check if the actual configuration is still conform regularly
- **Scanner can check for known vulnerabilities**
  - *Nessus* or *Nikito* for instance
  - You should run them on a regular basis

Protection

- **Write Hardening guideline for your application**
  - Configuring all security mechanisms
  - Turning off all unused services
  - Setting up roles, permissions, and accounts, including disabling all default accounts or changing their passwords
  - Logging and alerts
- **Use an automatic configuration tool**
  - Must maintain the configuration on all your servers
- **The maintenance process should include:**
  - Monitoring the latest security vulnerabilities published
  - Applying the latest security patches
  - Updating the security configuration guideline
  - Regular vulnerability scanning from both internal and external perspectives
  - Regular internal reviews of the server's security configuration as compared to your configuration guide
  - Regular status reports to upper management documenting overall security posture

A8 Insecure Deserialization

A8 Insecure Deserialization

- **Applications and API are vulnerable if they deserialize hostile or tampered objects supplied by an attacker**
  - Exploiting is somewhat difficult.
- **Typical Data tampering**
  - Access control related attacks
  - Existing data structure but 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

Example Attack Scenario 1

- **A React application calls a set of Spring Boot microservices.**
  - Being functional programmers, they tried to ensure that their code is immutable.
- **The solution they came up with is serializing user state and passing it back and forth with each request.**
- **An attacker notices the "R00" Java object signature**
  - He uses the Java Serial Killer tool to gain remote code execution on the application server.

Example Attack Scenario 2

- **A PHP forum uses PHP object serialization to save a ”super” cookie,**
  - It contains the user’s user ID, role, password hash, and other state:

    ```
    a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
    i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
    ```
- **An attacker changes the serialized object to give themselves admin privileges:**

  ```
  a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
  ```

Prevention

- **Do not accept serialized objects from untrusted sources**
- **If not possible:**
  - Implement integrity checks such as digital signatures on any serialized objects
  - Enforce strict type constraints during deserialization before object creation
  - Isolating and running code that deserializes in low privilege environments when possible.
  - Logging deserialization exceptions and failures,
  - Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
  - Monitoring deserialization, alerting if a user deserializes constantly.

A9 - Using components with known vulnerabilities

Using components with known vulnerabilities

- **Exploitability between easy to hard**
  - Some exploits are already-written
  - Some require effort to be developed.
- **Issue is very widespread**
  - Some teams do not even understand which components they use
  - so can not keep them up to date.
- **Scanners can find the list of components**
  - `retire.js` for instance
  - Does not help to exploit (just to find).
- **Impact**
  - Starts with hard to exploit breaches to massive dangers

Is the application vulnerable? I

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.**

Is the application vulnerable? II

- **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.
  - It 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.**

Example Attack Scenarios

- **Internet of Things**
  - Some devices do not allow patching
- **Shodan IoT search engine finds devices with problems**
  - Can be used to find items with Heartbleed vulnerability.

Protection

There should be a patch management process in place to:

- **Remove unused dependencies, unnecessary features, components, files, and documentation.**
- **Continuously inventory the versions of both client-side and server-side components**
  - Monitor their dependencies
  - Continuously monitor sources like CVE or NVD for your components
- **Only obtain components from official sources over secure links.**
  - Prefere signed packets
- **Monitor for libraries and components that are unmaintained or do not create security patches for older versions.**
  - If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

A10 Insufficient Logging and Monitoring

A10 Insufficient Logging and Monitoring

- **Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context**
  - Should be used to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- **Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.**
- **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 in a timely fashion.**

Cross Site Request Forgery CSRF

Cross Site Request Forgery

- **Not a new attack, but simple and devastating**
- **CSRF attack forces a logged-on victim's browser to send a request to a vulnerable web application**
- **Target: Perform the chosen action on behalf of the victim**

Sending a request without the consent of the victim?

- **Insert an image in a HTML file**

  ```
  <img src="http://www.benoist.ch/image/test.gif">
  ```

  Browser: Downloads an image

  ```
  GET /image/test.gif HTTP/1.1
  ...
  ```

- **An image can be generated by a PHP program (or any program)**

  ```
  <img src="http://www.benoist.ch/image/test.php">
  ```

  Browser: Downloads an image

Sending a request? (Cont.)

- **An image can be generated according to some parameters**

  ```
  <img src="/barcode.php?number=12345678901">
  ```

  Browser: Downloads an image

- **An image tag can contain something else**

  ```
  <img src="http://www.benoist.ch/index.php?action↘
  →=logout">
  ```

  Browser ?????

Widespread vulnerability

Widespread vulnerability

Touches any web application that

- **has no authorization checks for vulnerable actions**
- **will process an action if a default login can be entered in the request**

```
<img src=
"http://www.benoist.ch/doSomething?user=admin&pwd=↘
→admin">
```

- **Authorizes requests based only on credentials that are automatically submitted**
  - cookies if currently logged into the application
  - or “Remember me” functionality if not logged into the application
  - or a Kerberos token if part of an Intranet participating in integrated logon with Active Directory.

State of the art

- **Most of web applications rely solely on automatically submitted credentials**
  - cookies
  - basic authentication credentials
  - source IP addresses
  - SSL certificates
  - or windows domain credentials
- **Vulnerability also known as**
  - Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking, and Automation Attack
  - Acronym XSRF is also used together with CSRF

Vulnerability?

Vulnerability

- **A typical CSRF attack directs the user to invoke some function**
  - for instance application's logout page
- **The following tag can be inserted in any page viewed by the victim**

  ```
  <img src="http://www.benoist.ch/logout.php">
  ```

  it generates the same request as clicking on a link containing this address!
- **Example: Online banking transfer**

  ```
  <img src="http://www.mybank.de/transfer.do?
  fromAccount=document.form.frmAcct&
  toAccount=4567890&amount=3434.43">
  ```

  Could transfer the money from the account of the user, to a given account.

Attacks using CSRF

How such a link could reach a victim

- **Web-site Owner embedded JavaScript malware**
- **Web page defaced with embedded JavaScript malware**
- **JavaScript Malware injected into a public area of a website. (persistent XSS)**
- **Clicked on, a specially-crafted link causing the website to echo JavaScript malware. (non-persistent XSS)**

CSRF allows to access the intranet

- **The attacker sends requests from inside the Intranet**
  - Doesn't have to go throw the firewall, the victim is already
- **CSRF combined with javascript allows to send many requests sequentially**
  - javascript adds an image in the DOM (possibly invisible).
  - when the request is sent, another image is added
  - and so on

Why do CSRF attacks work?

- **User authorization credential is automatically included in any request by the browse**
  - Typical: Session Cookie
- **The Attacker doesn't need to supply that credential**
  - It belongs to the victim's browser
- **Success of CSRF belongs on the probability that the victim is logged in the attacked system**
  - Idea: attack the site the victim visits
  - Mean : XSS

Protection

CSRF Token

- **Application must ensure that they are not only relying on credentials or tokens that are automatically submitted by browsers**
  - Session Cookies
  - Certificates
  - Remember me
  - . . .
- **Application should use a custom token that the browser will not “Remember”**
  - So it can not be included in the Requests sent automatically

Strategies

- **Ensure that there are no XSS vulnerabilities in your application**
  - Otherwise, any protection is useless, since javascript could access the hidden data.
- **Insert custom random tokens into every form and URL**
  - It will not be automatically submitted by the browser
  - Example:

    ```
    <form action="/transfer.do" method="POST">
      <input type="hidden" name="383838" value="↘
      →1234323433">
      ...
    </form>
    ```

  - Then you have to verify that token
  - Token can be unique for a session or even for each page
  - The more focused the token is, the higher the security is, but the application is then much more complicated to write

Strategies (Cont.)

- **For sensitive data or value transactions, re-authenticate or use transaction signing**
  - to ensure that the request is genuine.
  - Set up external mechanism to verify requests (phone, e-mail)
  - Notify the user of the request using an e-mail
- **Do not use GET requests for sensitive data or to perform value transactions**
  - Use only POST methods when processing sensitive data from the user.
  - However the URL may contain the random token as this creates a unique URL, which makes CSRF almost impossible to perform
- **POST alone is an insufficient protection**
  - You must also combine it with random tokens

Protect your token

- **Disclosure of Token in URL**
  - If you include token in GET requests (i.e. URL)
  - It mitigates the risk of CSRF attacks
  - But the unique per session token is exposed
- **Exposition of URL's**
  - Browser history
  - HTTP log files
  - network appliance loging the first line of HTTP requests
  - URL-Referer is transfered to third parties
- **Third party knowing the token**
  - CSRF is trivial to be launched
  - Can target the attack effectively (referer tells the user is visiting the site)
  - Can run entirely in JavaScript
  - Just need a JavaScript call in the page.

Protect your token (Cont.)

- **Prevention**
  - Referer is omitted if origin of the request is HTTPS
  - Solution: make web site HTTPS only
  - Or use only POST methods for sensitive actions
  - Do not include CSRF token in GET requests

Double Submit Cookies

- **Send a random value in both cookie and request parameter**
  - The value is stored as a cookie
  - It is included as hidden parameter in all forms
  - No need to store the value on the server
- **Server verifies: cookie = received value in form**
- **The attacker**
  - Can modify the form parameter
  - Can neither read nor modify the cookie
- **Attacker can not submit both elements at the same time**
- **Solution adopted by Java Library Direct Web Remoting (DWR)**

Encrypted token pattern

- **Server generates a token**
  - Placed in all forms as a hidden field
  - Not stored on the server
- **Token contains**
  - User's ID
  - timestamp
  - nonce
  - Encrypt the information with a symetric key (only known by the server)
- **Token is included in all the requests**
  - AJAX requests will send the information in the URL
  - Non AJAX requests will include information as a hidden field
- **On receipt of the token**
  - Server checks the values stored in the token
  - User's ID
  - timestamp (agains replay attacks).

CSRF prevention without a synchronizer token

Checking the referer header

- **Trivial to spoof referer header**
  - But only on your own browser
  - impossible to do so in a CSRF attack
  - Solution commonly used with unauthenticated requests (before login)
- **Weak protection against CSRF**
  - Open redirect vulnerabilities can be used to exploit GET-based requests
  - Some organisations (or browsers) may remove Referer from requests (privacy protection).
- **Common implementation mistakes**
  - If attacks originates from an https server: Referer is omitted
  - Lack of referer : marks an attack (at least of state changing functions)
- **Referer could be lightly manipulated**
  - If victim site is `site.com`
  - Attacker could send requests from `site.com.attacker.com`
  - Could fool easy testings

Challenge-Response

- **Captcha**
  - Generate an image
  - Verify that a human sees the image and interprets it
  - Can not be automatic in CSRF
- **Re-Authentication (password)**
  - Necessary for very sensitive actions
  - Change password for instance
  - Very user-unfriendly
- **One-time token**
  - Token is changed with each request

Mitigating risks from the user's point of view

- **Logoff immediately after using a Web application**
- **Do not allow browser to save username/passwords**
- **Do not use the same browser to access sensitive applications and to surf the internet**
  - Tabbed browsing
- **Plugins like "No-Script" makes POST based CSRF difficult to exploit**
  - JavaScript is used to automatically submit the form when the exploit is loaded
  - Without JavaScript, the attacker must trick the user to do so manually

Conclusion

Conclusion

- **Web Security belongs to security**
  - Encryption,
  - Testing of inputs
  - Teaching of users
- **It is somehow different**
  - Restricted entrypoint port 80 (may be more easy to protect)
  - Open infrastructure (anybody can visit and attack)
  - International Architecture
  - No control on the client

References

- **OWASP Top 10, The ten most critical Web Application Security Vulnerabilities, 2013 Update, and 2017**
  http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- **Talk given at OWASP Switzerland (December 15, 2015), Antonio Sanso (Adobe Research Switzerland)**
  https://www.owasp.org/index.php/File:20151215-Top_X_OAuth_2_Hacks-asanso.pdf
- **RSnake, "What is CSRF?"**
  http://ha.ckers.org/blog/20061030/what-is-csrf/
- **OWASP CSRF Prevention Cheat Sheet**
  https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet