Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

6) Security

Emmanuel Benoist

Spring Term 2021

Computer Science Division

Security

Last week

Last week: Services

- **GET to read information**
  - Parameter is in the query string
  - Can be read in `msg.req.query.XXXXXX`
- **POST to send information to the server**
  - Can contain much larger data
  - We used JSON
  - needs CURL to be tested
  - Can be read at `msg.payload`
- **JSON is used to communicate with the other applications**
  - See the App part.

Introduction

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
- **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)
- **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 - Injection**
- **A2 - Broken Authentication and Session Management**
- **A3 - Data Sensitive Exposure**
- **A4 - XML External Entities (XXE)**
- **A5 - Broken Access Control**
- **A6 - Security Misconfiguration**
- **A7 - Cross-Site Scripting (XSS)**
- **A8 - Insecure Deserialization**
- **A9 - Using Components with Known Vulnerabilities**
- **A10 - Insufficient Logging & Monitoring**

A1 - Injections

Injections

- **User Supplied Data sent to an interpreter**
  - SQL injection
  - Interpreter injection (Shell, XML, . . .)
- **Attacker tricks the interpreter into executing unintended commands**
  - Can control the Database
  - Can execute command on the server

How does it work?

- **Attacker tricks the interpreter into executing unintended command**
- **Attacker supplies unexpected content to a site**
  - Data is especially designed to fool the site
- **Attacker may take control of the interpreter, for instance SQL:**
  - Read data (unintended, of course)
  - update, delete or create any arbitrary data
- **For the Operating System interpreter**
  - Attacker may have the opportunity to execute any command

Vulnerability

- **Environments affected**
  - Any framework using an interpreter or invoke process
  - SQL
  - Command line
  - . . .
- **System is vulnerable when user input is passed without tests**

Node-RED JavaScript

```
if(typeof(msg.req.query.patient)!== "undefined"){
    var pID = msg.req.query.patient;
    node.log("UserID␣="+pID);
    msg.topic="select␣*␣from␣patient,␣vital_sign␣where␣↘
    →patient.patientID␣=␣vital_sign.patientID";
    msg.topic+="␣and␣patient.patientID="+pID;
}
else{
    msg.topic="select␣*␣from␣patient";
}
```


SQL-Injection

- **Normal input**

  http://*localhost:1880/prog1?patient=1*
- **Generated query**

  select ∗ from patient, vital_sign where patient.patientID = ↘
  →vital_sign.patientID and patient.patientID=1;
- **Possible input**

  http://*localhost:1880/prog1?patient=0+union+all+select*↘
  →*+username,+name+from+staff+where+1*
- **Generates the following Query**

  select ∗ from patient, vital_sign where patient.patientID = ↘
  →vital_sign.patientID and patient.patientID=0 union all ↘
  →select username, name from staff where 1

A2 - Broken Authentication

- **Account credentials and sessions tokens are often not properly protected**
  - A third can access to one's account
  - Attacker compromise password, keys or authentication token
- **Risks**
  - Undermine authorization and accountability controls
  - cause privacy violation
  - Identity Theft
- **Method of attack: use weaknesses in authentication mechanism**
  - Logout
  - Password Management
  - Timeout
  - Remember me
  - ...

Brute Force Attack

- **Automated process of trial and error**
  - Guess a person username and password, credit-card number, cryptographic key, ...
  - System sends a value and waits for the response, then tries another value, and so on.
- **Many systems allow the use of weak passwords**
  - An attacker will cycle through a dictionary (word by word)
  - Generates thousands (potentially millions) of incorrect guesses
  - When the guessed password is OK, attacker can access the account!
- **Same technic can be used to guess encryption keys**
  - When the size of the key is small,
  - An attacker will test all possible keys

Session Spotting

- **Attacker has the possibility to listen to the traffic of the victim**
  - Listens to the traffic at the IP level (sniffer - Wireshark)
- **Client connects to the HTTPS server** `https://www.mysite.com`
  - Visits a page containing a login form (url is HTTPS)
  - Receives a cookie containing his session ID
  - Sends his credentials encrypted (HTTPS)
- **Client access any resource from** `http://www.mysite.com` **(not secure, HTTP)**
  - If the session cookie is not taged **secure** then it is sent without protection
- **Attacker receives following information**
  - Session ID
  - Sees that the user has sent his credentials (using an encrypted connection to the server)
- **Attacker can use the cookie to be recognized as the legitimate user!**

A3 - Sensitive Data Exposure

Sensitive Data Exposure

- **OWASP TOP 10 A3:2017**
- **Where sensitive data can be accessed due to lack of encryption**
  - Local Storage
  - Database
  - Transit (LAN)
- **Backups contain sensitive data**
  - Backup policy is part of security policy
  - Data stored must be readable
  - ... but not to much!

Exploitability

- **Attackers typically don't break crypto directly**
  - Break something else
  - Steal keys
  - man-in-the-middle
- **Steal clear text data**
  - on the server
  - in transit
  - from user's browser

Hard to exploit

- **Simple: no encryption at all**
  - the most common flaw
- **When crypto is employed**
  - weak key generation
  - weak key management
  - weak algorithm usage
- **Difficult to detect server side flaws**
  - limited access
  - hard to exploit

Impact is Severe

- **Compomises sensitive data**
  - Health records
  - credentials
  - personal data
  - credit cards
  - ...
- **Impact on your business**
  - Value of data for competitors
  - Reputation
  - Compliance

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">]>
```


A5 - Broken Access Control

Broken Access Control

- **Access control should**
  - force users to act inside the intended permissions
- **Failures lead to**
  - unauthorized information disclosure
  - modification or destruction of data
  - performing a business function outside the limits of the user
- **Missing function level access control:**
  - access pages without authorization.
- **Insecure direct object references:**
  - access data or files directly.

Introduction

- **Possibility to access pages without necessary privileges**
- **Vertical escalation**
  - Anonymous users access private functionalities
  - Regular users access administrator functionalities
- **Horizontal escalation**
  - Users access data of other users
- **Possibility to access resources without required privileges**
  - Access Files intended for other users
  - Access directories and files outside the scope of the web server.

Insecure direct object references

- **Occurs when developer uses HTTP parameter to refer to internal object**
  - For instance `http://mysite.com/program.php?lang=fr`
  - And in the program:

    ```
    require_once($_REQUEST['lang']."lang.php");
    ```
- **Can also access to identifiers**
  - For instance `http://mysite.com/program.php?page=124`
  - It may be possible to change the page ID. The rights to see the page have to be tested.
- **Which objects are subject to attacks?**
  - Files: for upload and/or for reading, or accessing
  - Identifiers : for showing them, or changing them

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 flow 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

A7 - Cross Site Scripting - XSS

Cross Site Scripting - XSS

- **If the web site allows uncontrolled content to be supplied by users**
  - User can write content in a Guest-book or Forum
  - User can introduce malicious code in the content
- **Example of malicious code**
  - Modification of the Document Object Model - DOM (change some links, add some buttons)
  - Send personal information to thirds (javascript can send cookies to other sites)

Cross Site Scripting - XSS

- **If the web site allows uncontrolled content to be supplied by users**
  - User can write content in a Guest-book or Forum
  - User can introduce malicious code in the content
- **Example of malicious code**
  - Modification of the Document Object Model - DOM (change some links, add some buttons)
  - Send personal information to thirds (javascript can send cookies to other sites)

modus Operandi

- **Attacker Executes Script on the Victim's machine**
  - Is usually Javascript
  - Can be any script language supported by the victim's browser
- **Different types of Cross Site Scripting**
  - *Reflected*
  - *Stored*

Reflected XSS

- **The easiest exploit**
- **A page will reflect user supplied data directly back to the user**

```
echo $_REQUEST['userinput'];
```

- **So when the user types:**

```
<script type="text/javascript">
alert("Hello World");
</script>
```

- **He receives an alert in his browser**
- **Danger**
  - If the URL (containing GET parameters) is delivered by a third to the victim
  - The Victim will access a modified page
  - SSL certificate and security warning are OK!!!

Stored XSS

- **Hostile Data is taken and stored**
  - In a file
  - In a Database
  - or in any other backend system
- **Then Data is sent back to any visitor of the web site**
- **Risk when large number of users can see unfiltered content**
  - Very dangerous for Content Management Systems (CMS)
  - Blogs
  - forums

Insecure Direct Object Reference

- **Occurs when developer uses HTTP parameter to refer to internal object**
  - For instance `http://mysite.com/program.php?lang=fr`
  - And in the program:

```
require_once($_REQUEST['lang']."lang.php");
```

- **Can also access to other accounts**
  - For instance `http://mysite.com/program.php?page=124`
  - It may be possible to change the page ID. The rights to see the page have to be tested.

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.**

Recommendations

Recommendations

- **Some easy measures for preventing attacks**
  - Test your input
  - Verify the rights on the page (resp. the data)
  - Do not expose internal values
  - Restrict priviledges to the minimum

Test your input

- **You must test the input on the server**
  - Test all incoming information
  - Do not trust tests on client (JavaScript)
- **Test the type of the input**
  - String, integer, double, etc.
  - Test the length
- **Test against Meta Characters**
  - For SQL injection ' ",
  - For XML injection < >
  - etc.
- **Better: "accept only known good"**
  - A number,
  - a name = string without special chars (maybe ')
  - a phone number, ...

Verify the rights of the user

- **For viewing a page**
  - Each page must verify that the user has the rights
  - Do not use "hidden pages"
- **For accessing data**
  - Not all users can view all data
  - Do not trust what is received from the client
  - When accessing data, verify the rights
- **Allow a real logout**
  - If a client logs out: delete the session server side
  - Prevents an "history" attack
  - Secure if the session's credentials are stollen

Do not expose internal values

- **Do not expose file name**
  - Risk = file system traversal
    - Suppose `http://www.site.com/index?action=login`
    - loads file `login.php`
    - What would do:`http://www.site.com/index?action=/etc/passwd%00` ?
  - **Do not expose internal IDs**
    - They can be modified
    - or used for SQL injection

Restrict Priviledges to the minimum

- **Start as less as possible services on one host**
  - Separate using virtual hosts if necessary
  - Host only HTTP server on a web server
- **Give access only to the files required**
  - Never grant write access to the web user (`nobody` or `www`) inside the document root.
- **Restrict access to the DataBase**
  - If the web just reads one table
  - Create a specific DataBase user that can just read one single table
  - Never use root

Conclusion

Conclusion

- **Web Security**
  - Security by Design
  - Must be integrated ASAP in the Development life cycle
- **Risk with modern applications**
  - Same team develops the app or the client side application
  - and the server side.
  - But communication can be eavesdropped / manipulated.
  - Huge risks in Modern applications
- **Solutions**
  - Always validate server-side,
  - test your inputs
  - test your serialized objects
  - encrypt communications and data
  - Do not always rely on TLS (HTTPS).

References

- **OWASP**
  `https://owasp.org`
- **OWASP Top Ten**
  `https://owasp.org/www-project-top-ten/`
- **OWASP Switzerland**
  `https://owasp.org/www-chapter-switzerland/`