Security in Web Applications

Contents

1 Introduction

The purpose of this document is to introduce the basic concepts of security in web applications. In the software engineering world, web applications share some specifics: they use many elements, some of them beyond the control of the developer. Each of them can be a security breach or an opportunity to prevent some attacks. We want to stress out that security isn't limited to systems and cryptography. It concerns the developer when he writes his code; the architect when he designs his complete solution. The analyst has to choose the right trade-off between usability and security. Even the project manager has to think about security while keeping the balance between cost, time and quality. Moreover, it is also important to be able to identify an eventual attack and to diagnose it afterwards. Both of these capabilities require money and competency but above all it requires preparation.

2 Psycho-sociological Aspects

2.1 Security concerns

Internet consists in interconnected networks. These networks were developed in a closed and trusted environment by the U.S. Department of Defense. Security was not a concern. At this time, only a few research and government sites were linked together to share data. Later, in the middle of the eighties, the first real network security incident appeared: a German spy, named Markus Hess, managed to get into the US military network and stole information.

The Network then continued to grow to finally become the Internet as we know it today, but security problems continued to appear in the same time. Both aggressive and defensive technologies become more sophisticated. Nowadays, internet is present everywhere in our digital life: we use it for our personal informations and leisures (YouTube, Facebook), for financial management (online banking, Tax-On-Web) and of course in our professional lives (e-mail, calendars). As we are putting those, always expanding, parts of our life on the web, security concerns becomes more important than ever.

2.2 Sentiment of security

Security efforts are made to bring trust to users, sometimes giving them a false sentiment of security. Web services need to gain some level of trust, still they must not lure the users. Indeed, don't we believe that HTTPS websites could be seen as fully secured web sites? In fact, they just prevent some of the attacks. Every web site should indeed earn the confidence of its users because personal information will be shared on the network and it should be unacceptable to see this data stolen because of a compromised web site. That's why security aspects require more specific measures, more guarantees and a stronger protection against all kinds of attacks.

2.3 Users as a security flaw

Security is also a user's responsibility. They can divulge their credentials, use unsecured web sites or compromised software without ever knowing the threat they pose.

For example, many critical web service now requires to use a password. But with a number of services getting more and more important, remembering multiple passwords can become a hard task. A consequence is that most users use the same password for several web sites

or just write them down, for example on post-its. Moreover, these passwords are generally short and pass-phrases are uncommon. Some online password storage services exist to help managing the passwords, but these services are not always used or even known.

2.4 Security vs usability

A compromise must be found between high security and usability. This implies to make a choice between the usage of few or many security measures, simple or more complex passwords, plain text or encoded data, etc. It is easier to just use a service without worrying about anything, still security checks and authorization are necessary to ensure that the data is quite safe.

3 Authentication and identity

The *identity* of somebody is who he claims to be. The *authentication* is the act of asserting the identity of something or someone. Authentication is needed when you want to transmit confidential information, you want to be sure that your correspondent isn't impersonated by somebody else. The *authorization* is the act of determining whether the user has the permission to perform some action. Beware that authentication by itself does not imply confidentiality.

3.1 Technical introduction

HTTP

Listing 1: HTTP Request Example

```
GET /wiki/Hypertext_Transfer_Protocol HTTP/1.1
Host: en.wikipedia.org
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:19.0)...
Accept: text/html,application/xhtml+xml,...
Cookie: mediaWiki.user.id=m5MFt3bI4Lr1GcMvj4rPk4mwtwZqTWhx; ...
Connection: keep-alive
...
```

A *request* consists of a method (`GET`, `POST`, ...), a resource path and options such as the host, the user-agent, cookies, etc. The *response* follows the same schema and is followed by the content.

HTTP is *stateless*: no information is saved between requests. To overcome this limitation, cookies were introduced. These are key-value data stored on the client's computer. They are sent in each request and can be defined in the response by the server.

To track clients, websites use a session ID cookie for each client. It is a "unique number" generated when the user accesses the website for the first time. There is still no guarantee that this number is unique, even if conflicts are improbable.

Cryptography introduction

Symmetric cryptography The goal of the encryption is to obtain a message in a way that is readable only by the authorized parties. This encryption can be symmetric, that is, a single key is used to encode and decode the message.

$$\text{text} \xrightarrow{key} \text{ciphered text} \xrightarrow{key} \text{text}$$

Widespread symmetric cryptography algorithms are AES, DES, Twofish.

Asymmetric cryptography A private-public key pair is used to realize asymmetric encryption. One key is used to encrypt the message and the other to decrypt the digest depending upon the goal. If the sender wants his message to be decodable only by a receiver, he encrypts it with the receiver public key and the receiver decrypts it with his private key.

$$\text{text} \xrightarrow{key_{public}} \text{ciphered text} \xrightarrow{key_{private}} \text{text}$$

If the sender wants the receiver to be sure that he is the one that sent the message, things are done the other way round.

$$\text{text} \xrightarrow{key_{private}} \text{ciphered text} \xrightarrow{key_{public}} \text{text}$$

Widespread asymmetric cryptography algorithms are RSA and ElGamal.

Symmetric versus asymmetric

Symmetric faster, needs a pre-shared key and all the people holding the key can decrypt the message.

Asymmetric slower but does not need a pre-shared key, messages can be decrypted by the person holding the right key.

The usual usage is to use asymmetric algorithm to exchange keys and then symmetric algorithm is used for the rest of the connection as it is faster.

By definition, the public key is shared.

Hash and salt A hash of a message is a fixed size string built using an irreversible (or at least difficult to reverse) function from that message. Sometimes, it's desirable that two identical messages don't have the same hash. The trick is to use a "salt" which is a random value appended to the message before hashing. The hash is then useless without the salt so they are stored together.

Signatures The signature is then a mean to assert the sender identity of a message. You can sign a message by encrypting a hash of it with your private key. Hashes are used to guarantee the *integrity* of the document without having to encrypt and decrypt the whole document.

3.2 Types of authentication

3.2.1 Password

This is the most common form of authentication. Passwords must have a sufficient complexity and must be easy to change periodically. Complexity has to be checked on both client and server side. People should use different passwords for different services.

3.2.2 One-time passwords

One time passwords are used to prevent replay attacks. The main difficulty of this technique is to give users their passwords (logistical problem). This technique is used mainly by services who can afford big infrastructures (banks, states, ...).

3.2.3 Signature challenge

A challenge uses a nonce, this stands for number used once. It's a random number used once to avoid replay attack. The nonce is encrypted with the recipient's public key and he is then asked for the decrypted (with his private key). This form of authentication is often used to sign online transactions.

3.2.4 Certificates

A trusted authority issues certificates to confirm the identity of something. They are used in SSL or TLS. Only one participant of the conversation can be certified (client or server) or both. A Public Key Infrastructure (PKI) can be used to distribute public keys in a secure way. The role of the PKI is to validate the identity of the owner of the key. A typical example is for SSL/TLS in web browsers. A CA is a Certification Authority, an entity that issues digital certificates.

- Each CA does generate its public-private key pair. They self-sign their certificates with their public key
- Browsers are distributed with some pre-installed certificates. You can also add them manually
- When websites want to sign a message, they sign it with their private key and send the signed object and their certificate
- The browser checks the certificates against the CA certificate and the message against the checked certificate

How to obtain a certificate? When a website owner wants a certificate, he generates a public-private key pair and then sends the public key to a CA. The CA checks the integrity of the request and performs some checks on the identity of the owner. Finally, it signs the certificate.

3.2.5 Token

Something whose ownership gives a form of identification. For example: Keys, Bank cards, Badges, Digipass, ...

3.2.6 2-factor authentication

Identifies the entity using multiple authentication methods. This way, if one method is compromised, the attacker will still not be able to log into the system. The authentication methods are the following:

- What it *is* (biometrics)
- What it *owns* (bank card,...)
- What it *knows* (password, pin code,...)

3.3 Single Sign-On

> The SSO is a property of access control of multiple related but independent software systems. With this property a user logs in once and gains access to all systems without being prompted to log in again for each of them. – Wikipedia

Pro

- Easier for the user
- Implementation already exists
- If they have only one password, users tend to treat it with more care
- Enter the password less often

Cons

- All your eggs in the same basket (impact greater if compromised)
- You are dependent upon your provider (confidence, availability,...)

3.4 Kerberos

Kerberos is a SSO implementation developed at MIT to solve the problem of allowing some users to use restricted resources. MIT provides a free implementation of the protocol but it's also found in many commercial products.

How it works

1. The client authenticate itself with the distribution center and get an authentication ticket.
2. When he wants to use a service, the client asks the distribution center for a service ticket giving his authentication ticket as a proof of authentication.
3. The client gives the service ticket, which contains the service request and the authorisation, to the service provider who performs the service.

3.5 Provider extensions

Common SSO used over the web:

- Facebook account
- Google
- OpenID (myOpenID, ClaimID, VeriSign, etc.)

These organisations provide single sign-on to other websites for free using a dedicated API which is much simpler than devising your own sign-on mechanism.

3.6 Compromised authentication

When you implement authentication you have to take into account that the user may lose its authentication mean or it can get stolen.

The process for renewing the authentication is the following:

1. Authenticate the user with a still safe mean
2. Deactivate the compromised authentication mean
3. Give the user a new primary authentication mean

Password recovery

Another usability versus security trade-off. How to do it properly?

1. Use of security question or other strong authentication
2. Send a token over a side channel
3. Allow the user to change password
4. Confirm change

Beware that an attacker may use social engineering to get answers to security questions.

4 Attack

4.1 Attack trees

We will start our study of different attacks by describing the attack tree model. The attack tree describes how an asset can be attacked. In other words, this kind of model is used to determine and understand threats that may arise.

For example, administration access can be obtained thanks to the knowledge of the password. One way to get this password is to steal it. To steal it we can use social engineering or sniffing techniques. This modeling technique enables us to build the attack tree (see Figure 1) showing threats for the administration access.

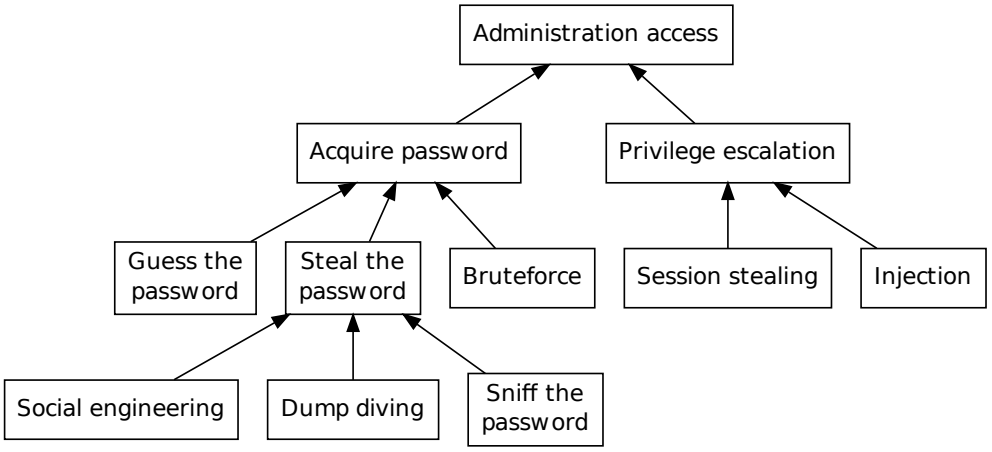


Figure 1: Attack tree example

4.2 Threats

Motivations are numerous and diversified.

- Break things. These attacks can come from internet hackers or viruses but also from internal threats in your enterprise such as unsatisfied employees.
- Financial gain. That's the case with organized crime or competitors wanting to acquire financial or technical information.
- Accidental. Examples include: loss of electricity, software failure, hardware failure, but also natural disasters or human errors.

4.3 Risk

To understand the risk encountered by systems vulnerable to such attacks, let's first define what is a risk.

> The probable frequency and probable magnitude of future loss.
> – *The Open Group*

$$risk = likelihood * impact$$

In other terms, the risk is the relation between vulnerability factors such as

- Ease of discovery
- Ease of exploit
- Awareness
- Intrusion detection

and impact factors such as

- Loss of confidentiality, integrity, availability or accountability
- Financial damage
- Reputation damage
- Privacy violation

4.4 Context of web applications

Basically, the web is based on a client-server model. The protocol used to retrieve web pages is HTTP. Anyone can access a public server and send requests.

In reality, the client is not directly connected to the server. A lot of intermediaries are present, each of them plays a different role and an attack can happen at any level. For example, the client can be on a WiFi network that is connected to the internet. The web server is on another network where other computers are present: other clients, intrusion detection systems, databases, etc. In between, we can find firewalls, demilitarized zones and others. Any of those components can be compromised in one way or an other.

4.5 Open Web Application Security Project (OWASP)

A worldwide organization known as Open Web Application Security Project (OWASP) focuses on improving the security of software through the publication of standards, software libraries and books.

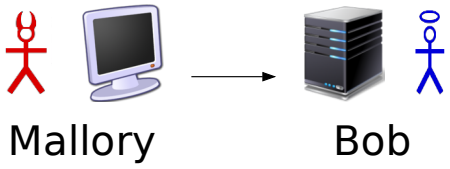
4.6 Top 10 of most critical web applications risks

OWASP provides a top ten of most critical web applications risks.

1. Injection
2. Cross-site scripting (XSS)
3. Authentication and Session Management
4. Insecure Direct Object References
5. Cross-site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Forwards and Redirects

4.6.1 Injection

Injection is the mechanism consisting in sending untrusted data to an interpreter. In other words, hackers enter code directly into input fields of a web form that is executed on the server. This is a very common attack, easy to exploit, and can produce a huge impact such as data loss, data corruption and lack of accountability.



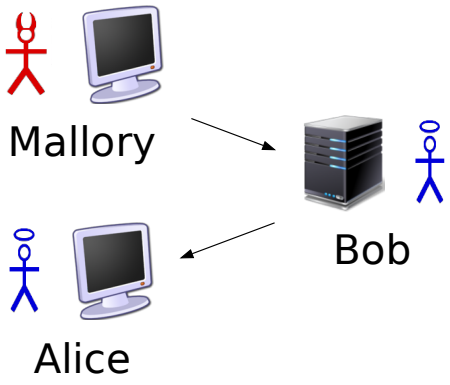
To avoid such security flaws, it is recommended to use a parameterized interface (see Listing 2). One can also use routines that "escape" (i.e. tell the interpreter that control character shouldn't be interpreted) user input or to use white-list validation to specify valid input patterns. Escape black lists are discouraged, because it is easier to forget some rule and it is often possible to use workarounds. Usually, this vulnerability is present when the code to execute is built using simple concatenation.

Listing 2: Parameterized interface in Java

```
String query = "select * from users where user_name = ?";
PreparedStatement st = con.prepareStatement(query);
st.setString(1, name);
ResultSet rs = st.executeQuery();
```

4.6.2 Cross-site scripting (XSS)

Cross-site scripting is an attack consisting in code injection directly into the page that will be sent to the user. This code can be a script that will finally be interpreted by the browser. This kind of attack requires some knowledge from the attacker but impacts are generally moderated. However, this is the most widespread vulnerability on the web.



Possible attacks include session hijacking, the change of a page content or user redirection to another web site. Once again, using escaping routines before sending values to the users or using white list validation prevents this attack.

Listing 3: XSS test

```
Hello <script>alert("Hello world!");</script> world!
```

4.6.3 Authentication and Session Management

This attack uses flaws in authentication and session management to steal someone else's identity. It's usually done using session hijacking or session fixation. This type of attack is very common and impacts are often important. Indeed, with the identity of someone else, the attacker can do anything the real user can do.

To avoid this kind of attack, a few "good practices" should be used. First do not forget to log out on public computer because your session could be easily used by someone else. The use of session timeout and visible log out buttons are also "good practices". Then, it is important to prevent the use of cross-site scripting attack because a session ID could be easily stolen thanks to this mechanism.

For example, session hijacking consists in stealing the session ID of another user through the use of an attack like XSS. Session fixation consists in forcing a user to use a particular session ID; it is sometimes possible to inject the session ID into the URL.

4.6.4 Insecure Direct Object References

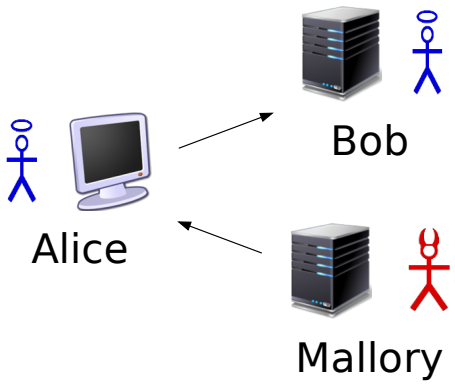
Insecure direct object references consists in changing a parameter in the URL so that another entity or document shared by the web site could be accessed. For example by replacing an identifier into the request referencing an accessible object by another identifier referencing an initially non accessible object. This kind of attack is very simple to exploit and can compromise referenced data.

```
http://host.com/user/152
```

To bring more security, source code in web applications should always check whether the user has the rights to access the object before manipulating it.

4.6.5 Cross-site Request Forgery (CSRF)

Cross-site request forgery is a mechanism that consists in generating requests when a user, the victim, visits the attacker's web page. The visited web page contains malicious code that can submit requests to the attacked web site using the visiting user's identity. This is a widespread attack but fortunately easy to detect.



In order to prevent CSRF a website should give a unique random token to each visitor. This token is sent back with the request. When receiving a request the website check the validity of the token. If the token received wasn't the one issued, the request is discarded. As the attacker's website won't have the token, he won't be able to generate a valid request.

The most common implementation is through the use of a hidden field in a form such as in Listing 4. The hidden field is set by the website and sent back unmodified.

Listing 4: A protected form against CSRF

```
<form method="post" action="addComment">
   <input type="hidden" name="token" value="sd5646sdfse8wd" />
   ...
</form>
```

4.6.6 Security Misconfiguration

Security flaws can also appear through security misconfiguration. Default configuration used for application, server or database are not necessarily a good choice. It could be possible to gain information such as versions, stack traces or debug messages that can lead to bigger threats which could compromise the whole system (gain unauthorized access, use known bugs to hack an application, etc.).

Keeping an up-to-date system is the first recommendation to prevent such attacks. In combination with a good separation between components, the systems will be only partially affected.

4.6.7 Insecure Cryptographic Storage

Sensitive data should be stored securely. When an attacker gains access to the data, a cryptographic barrier should be used to prevent the attackers from reading it. Unencrypted data will be compromised. This kind of attack is fortunately quite difficult to perform for attackers.

For example, a lost unencrypted backup of the database would contain all the information for the attacker. Also, information such as passwords must be hashed and a salt should be applied to prevent an attacker to use a precomputed list of message/hash pairs to find a match. Such lists are named rainbow tables.

4.6.8 Failure to Restrict URL Access

A missing URL access restriction is always a vulnerability. Attackers can change their URL to switch to a privileged page with functions requiring permissions. This is a quite uncommon flaw but it is easy to exploit. The main impact is of course the threat to give administrative functions to the attackers.

To prevent this attack, authorisation mechanisms should be used even if pages are not necessarily referenced from any other page.

4.6.9 Insufficient Transport Layer Protection

It is possible to capture traffic between a client and a server. Insufficient transport layer protection is the lack of encryption mechanisms when sensitive data is being sent. Examples of common attacks are sniffing a WiFi network, scanning for HTTP requests or even analysing conversations.

HTTPS can secure connections using SSL/TLS. Secure cookies will not be sent through an unencrypted connection and certificates will be verified by the browser to ensure their validity.

4.6.10 Unvalidated Forwards and Redirects

The last attack implies unvalidated forwards and redirects. This means that an URL may appear as trustworthy but actually contains unvalidated redirect to an unsecure page or it can contain a forward attempting to bypass authorization functionalities. This is an easy attack to detect and nowadays a rarely present kind of attack.

A barrier against these attacks is to avoid redirects and forwards as much as possible, and in the case when this is not possible, to validate and authorize the user after each redirection or forward.

4.7 Other common attacks

Besides these attacks listed by the OWASP organization, we can also mention other common attacks. Here are a few of them.

4.7.1 Denial of service

The attack by denial of service aims to bring down (or at least make unavailable) some service usually by causing server overload.

SYN flood When a client initiates a connection with a server, it sends a SYN request. The server then sends a confirmation (ACK) to the client that the connection is accepted and the client terminates this three-way handshake by responding to the server with an acknowledgment (SYN-ACK). A SYN flood attack consists in sending a huge quantity of SYN requests without responding to the server with an acknowledgment. This way, the server uses more and more resources but never frees them, causing either a denial of service or, for the worst case, a crash.

Application level flood Request can be sent to the server to initiate CPU-intensive tasks or to save a lot of data causing the disks to become full.

R-U-Dead-Yet The attack consists in sending data very slowly to the server. This is similar to a SYN flood, but the connection stays alive.

DDoS (Distributed Denial Of Service) Follows the same principle as a traditional Denial Of Service attack but it is launched by several attackers in the same time.

Unintentional DoS A web site ends up denied not because of a deliberated attack but because of a sudden peak of popularity.

Using firewalls as well as Intrusion Prevention System can reduce the risks to be a victim of those kinds of attack.

4.7.2 Man in the middle

Another well known attack is the man in the middle attack. The attacker plays the role of the server to the client and the role of the client to the server. This way, all communications transit through the attacker who can modify messages as they are sent. The victims believe they use a private connection but the attacker, the man in the middle, controls the whole exchange.

4.7.3 Social engineering

A less technical but nevertheless damaging attack is the use of social engineering. Social engineering simply consists in manipulating people so they divulge confidential information. Different scenarios are used by attackers in social engineering to achieve their goal. Pretexting is one of them and consists in using invented scenario to engage a targeted victim in a manner that increases the chance the victim will divulge information or perform actions that would be unlikely in ordinary circumstances. Another technique is *quid pro quo*, which consists in helping some user while getting from him confidential information such as passwords.

The only way to fight against social engineering is to educate users and use a framework of trust.

Another kind of social engineering attack is phishing, which is described in the next section.

4.7.4 Phishing

Phishing is the attempt to acquire user information, such as passwords or credit card details, through masquerading as a legitimate website. Tricks used can be cloned websites, link

manipulation or even phone phishing but most of the time phishing is carried out by e-mail or instant messaging. Most of the time, phishing e-mails contain links that redirect the user to malicious web sites. Victims then enter their confidential information that are finally recovered by the attacker.

4.7.5 Brute force

Brute force is an attack whose purpose is to try all possible combinations of passwords to discover the right one. A dictionary attack is based on the same principle, but only tries a list of common passwords. This kind of attack is very sensitive to the length and the complexity of the passwords. The longer the password, the more the attacker will have difficulties to discover it.

4.7.6 Path traversal

The goal of this attack is to access files that are present on a server. It is possible by browsing the application, looking for relative links to file on the server. Then, by manipulating those file references, it may be possible to access confidential or critical files stored into the server.

Listing 5: Getting a file from the server

```
http://host.com/forum/download.jsp?path=/etc/shadow
```

4.7.7 Failure to restrict automation

Often, web sites are not protected against automation. A program can launch commands to execute functionalities as if it was a human being doing the manipulation. The web server could be overloaded or spammed in this way.

To prevent this threat, a challenge-response test, called captcha, can be put in place. A captcha generally asks a user to complete a simple test which is difficult for a computer to answer. But captcha cracking stays possible. Indeed, a malicious web site could present the same captcha existing on another web site. Users surfing on the evil web site would crack unknowingly the other web site's protection by giving the correct answer to the captcha.

Against overloading, a maximum number of requests per IP for a fixed time period could be set up.

4.8 Common points

We can distinguish common points among these multiple attacks explaining why such attacks should arise.

- Improper input validation is a redundant problem. The data can be used in many ways, at each step the data must be checked whether it is valid and must also be escaped when it is meant to be interpreted.
- Missing authorization checks are always a problem.
- Misconfigurations giving more information for the attacker.
- Other assumptions such as ignoring maximum server load, encryption, etc.

- The education factor plays an important role in web security as we saw it with social engineering techniques.

As we can see, many forms of attacks exist and, as technology evolves, new attacks appear. So, instead of trying to fix vulnerabilities, we should focus on establishing strong security controls.

5 Prevention

5.1 Different moments

Preventing security flaws is something that has to be done during the whole life cycle of the application.

5.1.1 Architecture

Integrate a security review into your architecture design process. Start early on, and as your design changes, review those changes with the steps given in this chapter. Make your security review evolve. To complete the review process, you might also need to add specific questions that are unique to your application. Know the threats you are reviewing against. List the threats that affect the various components and layers that make up your application. Knowing these threats is essential to improve the results of your review process. There is no miracle receipt when it comes to designing a secure application. Although there are some guidelines that can help you. You have to ask yourself the good questions and make decisions on a case by case basis.

- Who are the end-users?
- In what environment will the application run?
- Who has access to the data?
- Are there other systems that access data? What are the rules?

5.1.2 Development

There are lots of best practices that can make your application more secure.

- Passwords: enforcing a minimum complexity policy, storing them securely, having the user rotate them, preventing brute-force by implementing temporary lock-out,...
- Session management: setting an inactivity time out, setting the secure-flag for sensitive cookies, using the HTTP-Only flag for cookies in order to restrict their use to transmitting HTTP request (it reduces the risk of session cookie theft by XSS), implementing an easily accessible logout,...
- Input validation: always on server side, use a positive approach: instead of forbidding some inputs, define the set of allowed input (usually through the use of *regexp*)
- Use ad-hoc protection for known attacks. Ad-hoc protection means a protection specialized for a particular attack in this context.

However, a secure development cycle should set aside some time for:

- Code review (security-oriented and general)
- Penetration testing (trying to attack the website, with the owner authorization, to find security holes)
- Security training of the developers
- Sensitizing business owners to security trade-off
- Choosing a secure and well maintained framework
- Devising a secure configuration for deployment

5.1.3 Maintenance

When vulnerabilities have been discovered they have to be patched as soon as possible. Beware of regressions and new security breaches; this is where automated testing comes in.

Virtual patching Virtual patching is a security policy layer that prevents the exploitation of a known vulnerability until a patch is released. It does so by filtering the traffic that goes to and from the application server and blocking requests that are exploits.

5.2 Using frameworks

You do not develop the first web application. Frameworks already exist to ease the development of your application while providing several security mechanisms. A framework will come generally with authentication support, escaping of values and standard input validation.

However, when a bug is found in a framework, all the web sites that use the framework are affected. Changing a framework may prove difficult as the written code may be coupled to the framework or follow the conventions used by that framework.

Examples of frameworks include Spring (Java) or OWASP which provides an Enterprise Security API that defines a security API for various languages.

5.3 Security in application servers

Application servers offer various services that can help you to configure secured web applications. Often the following aspects are configurable: authentication management, directory listing, user session management, error handling, input encoding, etc.

5.4 Web Application firewall

Application firewall are filters which inspect the traffic that comes from and goes to your application and is able to block inappropriate content. They provide an additional layer of protection, for example, against SQL injection by scanning parameters for SQL content. Also, an application firewall can detect connection patterns (see IDS). They can be used to patch security holes of applications which aren't modifiable for some reason (section 5.1.3).

5.5 HTTPS

TLS (Transport Layer Security) is the successor of **SSL** (Secured Socket Layer). They are equal in terms of security. HTTPS provides *encryption* and *authentication* for HTTP.
Protects against eavesdropping (somebody listening the conversation) and spoofing (somebody impersonating your correspondent). HTTPS is a part of a solution, it does not ensure full security. TLS is an open-community standard and allows both secure and unsecure communications.

5.6 Browser restrictions

Nowadays, browsers come with preconfigured security measures.

- Cross-domain `XMLHttpRequests` (XHR) are blocked. Browsers disallow contacting other domains through the use of JavaScript. A website can ask the browser to lift selectively this restriction via the header `Access-Control`.
- Some browsers allow to block (selectively) the execution of JavaScript, Flash, or other plug-ins to protect the user against malicious code hosted by those sites.
- A *private navigation* mode is often present which prevents the browser from storing data on disk.
- Cookies are stored on the user's computer; access to them is limited by the URL and the time stamp marks the validity expiration.
- Multiple opened pages are sandboxed: isolated from other pages to prevent accessing data in between websites visited by the client.
- Validation of SSL certificates is done.
- Some provide a blacklist of known rogue web sites.

5.7 E-mail

E-mail is one of the most used applications. It's plagued by spam and spoofing but solution exists: Sender Policy Framework and Domain Keys Identified Mail.

5.7.1 Sender Policy Framework

SPF validates e-mails by verifying sender's IP address against DNS records. Administrators can define allowed hosts for some domain.

5.7.2 Domain Keys Identified Mail

DKIM associates domain names and e-mail messages using a digital signature added to the header of messages, a public key is added to the DNS record of the domain for the recipient to read the message's header.

5.8 Variety of environments

The variety of environments available to the users leads to difficulties when it comes to testing possible setups for vulnerabilities.

We have different servers and their configurations, networks (proxy, caches, etc.), browser clients, mobile clients, machine clients (machine to machine communication) and so on. Moreover some browsers (e.g. IE6) require workarounds to display pages correctly which extend the amount of code and thus increase vulnerability likeliness.

6 Detection

6.1 Penetration testing

To find security holes in a system, penetration testing, or *pentesting*, is a method used to attack this system with the consent of the owner, using both external and internal attackers. The goals of penetration testing are:

- Identifying vulnerabilities
- Exploiting vulnerabilities
- Testing the ability to detect and respond to an attack
- Using social engineering

Several tools have been designed to perform these tasks. Here are some of them:

- *Nmap* (abbreviation for Network Mapper) is a free open source tool dedicated for network discovery and security auditing. Nmap suite includes a port scanning tool, a packet generation and response analysis tool as well as a scan results comparator tool. It is nowadays one of the most popular security tool, distributed with many open source operating systems and supported by a large community of users and developers.
- *Metasploit* is able to identify security vulnerabilities and automates assessments.
- *Nessus* is a proprietary vulnerability scanner. A vulnerability scanner is a tool designed to detect computers, systems, networks and applications weaknesses.
- *WebScarab* is a web security application testing tool, developed by OWASP, which serves as an intercepting proxy. It is able to intercept web browser requests and web server responses. A WebScarab user will then be able to capture and modify requests exchanged between the client and the server.

6.2 Honeypot

Internet provides a way to detect and counteract attacks upon systems thanks to dedicated assets. Those traps are called *Honeypots*. A Honeypot is a mocked website or web application running on a server with known vulnerabilities and that is exposed for attackers who try to penetrate some systems.

Honeypots can be distinguished in two types: production honeypots and research honeypots. The first group refers to honeypots used by organizations. They use production

honeypots to improve their overall state of security by capturing limited information. The second category concerns research honeypots. They are used to give a company informations about the motivations and tactics of the attackers by capturing much more information than a production honeypot, showing how to better defend itself against these attackers, but do not add direct value to an organization.

6.3 Network Intrusion Detection System

Network Intrusion Detection Systems (NIDS) are systems that can scan network activity and try to detect attacks.

For example, *Snort* is a NIDS and can detect attacks and take actions accordingly. Among those attacks, Snort can perform signature-based detection, which means it can find patterns in packets on the network, detect SQL injection or XSS attacks. Snort can also perform statistical anomaly-based detection. It looks for peaks in traffic, connection coming from a single host and other anomalies or detects brute force attacks. In the end, Snort can realize stateful protocol detection: it's able to follow a protocol conversation to detect anomalies. It enables it to catch SYN port scans.

6.4 Post-mortem analysis and performances

When an attack is detected on a system, several questions can be asked to try to solve the problem:

- When did the attack happen?
- Is the attack still in progress?
- Was the data compromised or modified?
- Which systems failed to work?
- How to fix the problem?
- How to prevent future attacks?

But now, how to discover whether an attack happened? A first clue could be to read traffic logs and graphs as well as application logs. Looking for peaks in the server load or traffic could show an anomaly. In combination, using analysis tools such as ntop, Snare or OSSEC can also help to detect attacks:

- *Ntop* is a traffic monitoring platform designed to probe network usage.
- *Snare* (abbreviation for *System iNtrusion Analysis and Reporting Environment*) is a tool created to audit log data and to centralize log analysis coming from different applications or operating systems.
- *OSSEC* is an open source NIDS that can perform various tasks such as log analysis supporting multiple formats, file integrity checking, policy monitoring, rootkit detection and intrusion detection.

However, looking for a particular attack can mask another one which would serve as a diversion for the real attack. Denial of service can be used to generate a lot of traffic and can mask other attacks as generated logs become huge. In the field of threat detection and analysis, testing performances can also be used to detect possible attacks. Indeed, slow response time and high load might be symptoms of an attack.

7 Conclusion

In this document we have reviewed various threats to web applications and some hints to fix, avoid or mitigate them but this list is by no means exhaustive. We hope that it will help you to look at your application with security in mind. Not in an extreme way, throwing money out the window for the sake of building a new digital Fort Knox. But rather with balance in mind and knowing the trade off you make.

References

- http://awstats.sourceforge.net (Monitoring Tool)
- http://blog.ringcentral.com
- http://developers.facebook.com
- http://en.gandi.net/ssl (SSL certificate provider)
- http://encyclopedia.thefreedictionary.com
- http://forums.asp.net
- http://googleonlinesecurity.blogspot.be
- http://msdn.microsoft.com
- http://openclipart.org (images)
- http://openid.net
- http://resource.onlinetech.com (IT blog)
- http://security.stackexchange.com
- http://shiro.apache.org (Security Framework)
- http://ssl.comodo.com (SSL certificate provider)
- http://stackoverflow.com
- http://web.mit.edu/Kerberos
- http://www.allaboutcookies.org
- http://www.backtrack-linux.org (Linux Distribution)
- http://www.citrix.com
- http://www.godaddy.com (SSL certificate provider)
- http://www.intersectalliance.com/projects/Snare (Monitoring Tool)
- http://www.metasploit.com (Pentensting tool)
- http://www.microsoft.com/net
- http://www.modsecurity.org
- http://www.ntop.org (Monitoring Tool)
- http://www.oracle.com
- http://www.ossec.net (Monitoring Tool)
- http://www.owasp.org

- http://www.pcinpact.com
- http://www.snort.org (IDS)
- http://www.sourcefire.com
- http://www.springsource.org (Framework)
- http://www.symantec.com (SSL certificate provider)
- http://www.tenable.com (Nessus - Vulnerability scanner)
- http://www.webdesignerdepot.com
- http://www.weblogexpert.com (Monitoring Tool)
- http://www.webopedia.com
- http://www.wikipedia.org
- http://xkcd.com (comics)