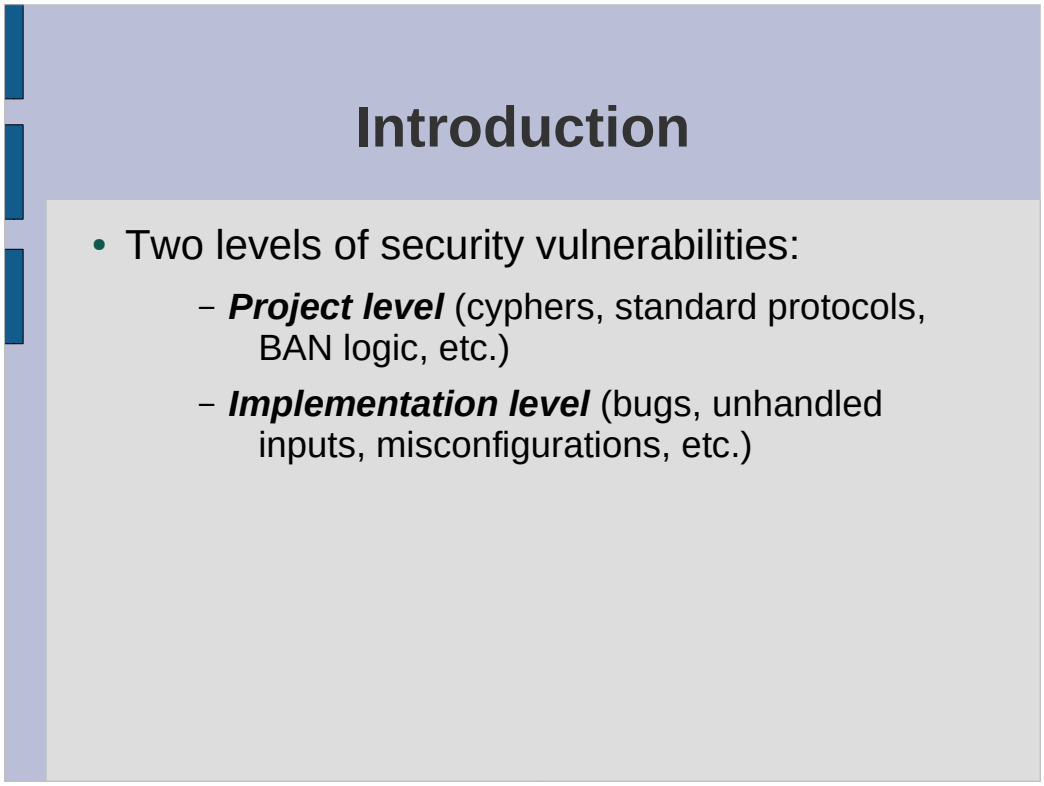
Web security

Pericle Perazzo
pericle.perazzo@iet.unipi.it



There are two levels of security vulnerabilities:

1) Project level, regarding how a system SHOULD work. This level includes cyphers, standard protocols, logic proof tools like BAN logic, etc.

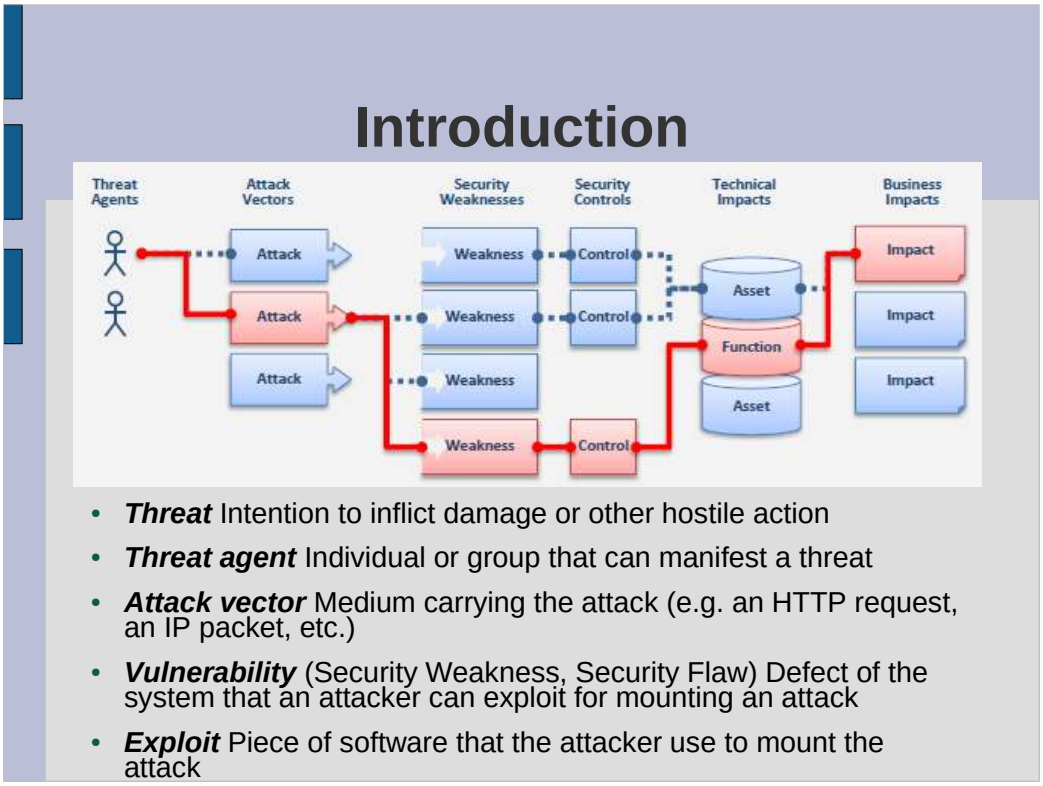
2) Implementation level, regarding how a system DO work, i.e. how it is effectively implemented and configured. This level includes bugs, unhandled inputs, errors in configuration, etc.

Introduction

- No “silver bullet” at implementation level (e.g. BAN logic)
  - Programming best-practices
  - Knowledge of the most common attacks
  - Penetration tests
  - Security certifications

At project level we can rely on general tools for proofing the security of our solution (e.g. BAN logic). At implementation level we can not. Implementation vulnerabilities are often very complex and buried under a mountain of code. We can never be sure that a system is free from security vulnerabilities. The best thing to do is to follow a set of programming best-practices, know the most common attacks, and pay human security experts, armed with good automatic tools, to catch vulnerabilities.

Some companies are specialized in performing security analysis and penetration tests. A penetration test is a simulation of various types of attack. After the penetration tests, they releases a security certification.



This is the OWASP's scheme of a typical attack. The threat agent chooses an “attack path”, including an attack vector, a security weakness to use, a security control to void, a technical and a business impact. The used terminology is the following:

- A threat is an intention of inflict damage or other hostile action.

- A threat agent is an individual or group that can manifest a threat. Modelling a threat agent (threat model) is very important in the security evaluation phase. Her capabilities, her intention and her past activities, must be taken into account.

- An attack vector is the medium through which the attack is performed. It could be an HTTP request, a simple IP packet, etc.

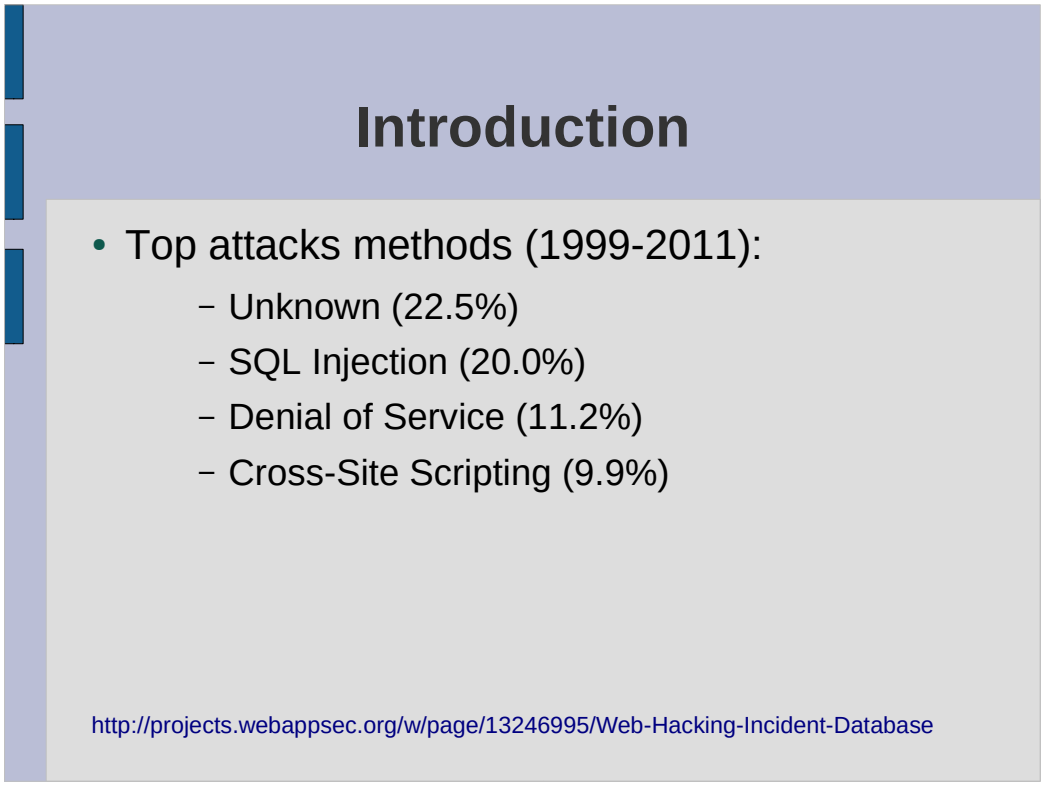
- A vulnerability (or Security Weakness, Security Flaw, Security Hole, etc.) is a defect of the system that an attacker can exploit for mounting her attack.

- An exploit (noun) is a piece of software that the attacker use to mount the attack.

For each kind of attack, OWASP rated the attack vector's exploitability, the weakness prevalence (less common, more common), the weakness detectability (whether the attacker can easily discover the vulnerability or not) and the technical impact's severity.

Introduction

Which are the most common attacks
and their goals?



According to webappsec.org, the most common attack methods of 2010 are:

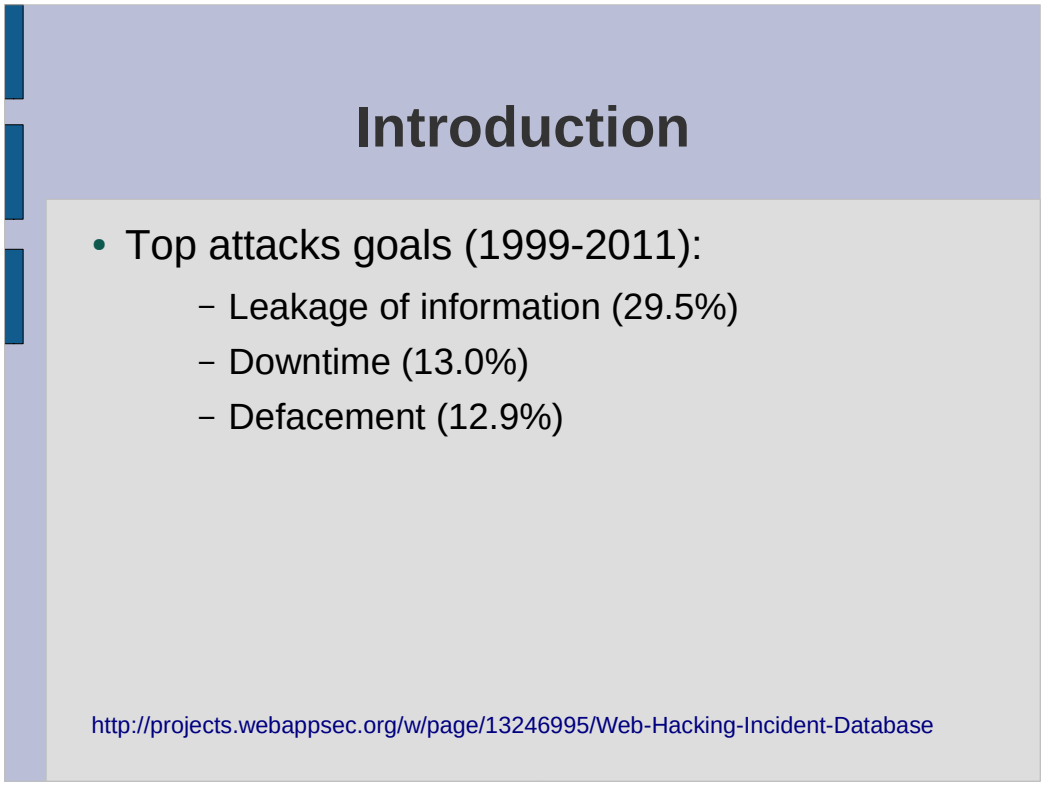
1) Unknown (22.5% of cases)

2) SQL Injection (20.0%)

3) Denial of Service (11.2%)

4) Cross-Site Scripting (9.9%)

The fact that in the 22.5% of cases the attack methodologies remain unknown tells us that it's very hard to discover them.



The most common attack goals are:

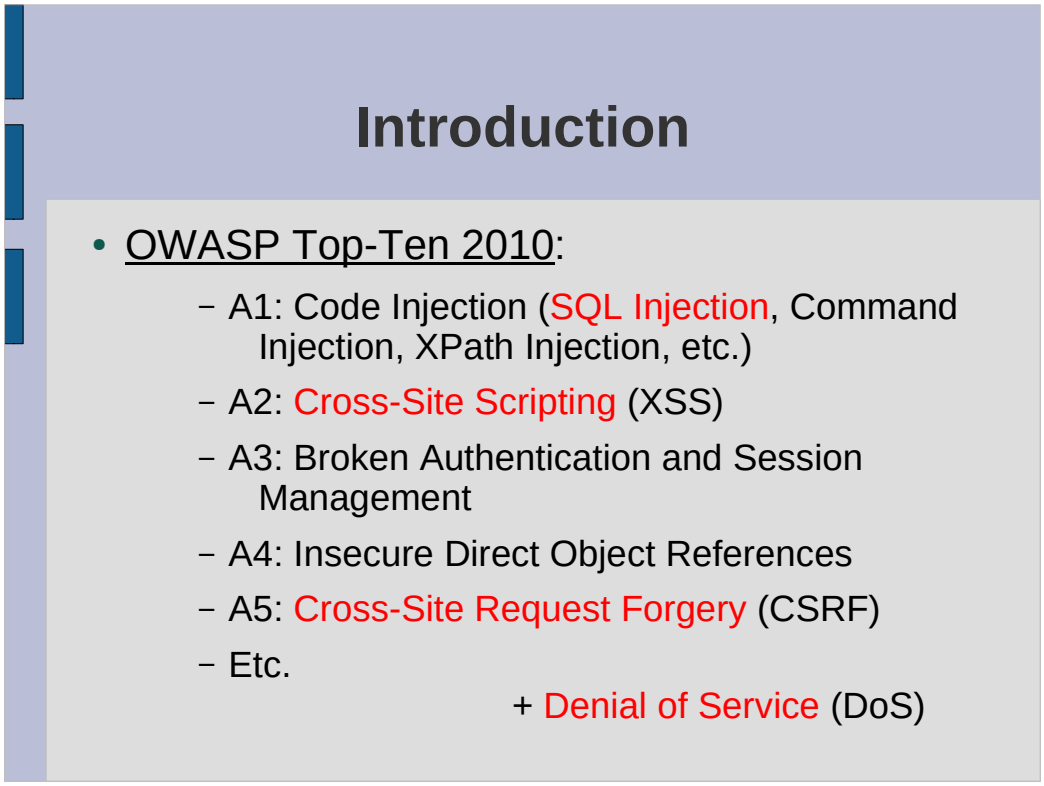
1) Leakage of information (steal some important information e.g. accounts, credit cards, etc.)

2) Downtime (the period of time in which a service remains unavailable)

3) Defacement (loss of credibility of a company or organization)

OWASP (Open Web Application Security Project) is an independent, nonprofit organization for Web security. It maintains a collection of web resources regarding web security and information security in general.

It periodically publish a Top-Ten list of web vulnerabilities, in order of dangerousness.



The 2010 Top 10 is the following:

A1) Code Injection in all the forms (SQL Injection, Command Injection, XPath Injection, etc.)

A2) Cross-Site Scripting (XSS)

A3) Broken Authentication and Session Management. This regards logic error on the management of web session, cookies, etc.

A4) Insecure Direct Object References

A5) Cross-Site Request Forgery (CSRF).

Etc.

OWASP did not included Denial of Service in this list because it is not a web attack: it actually affect the lower layers of the OSI stack (network, transport).

A broken authentication and session management is a bug in the management of the PHP session.

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place.

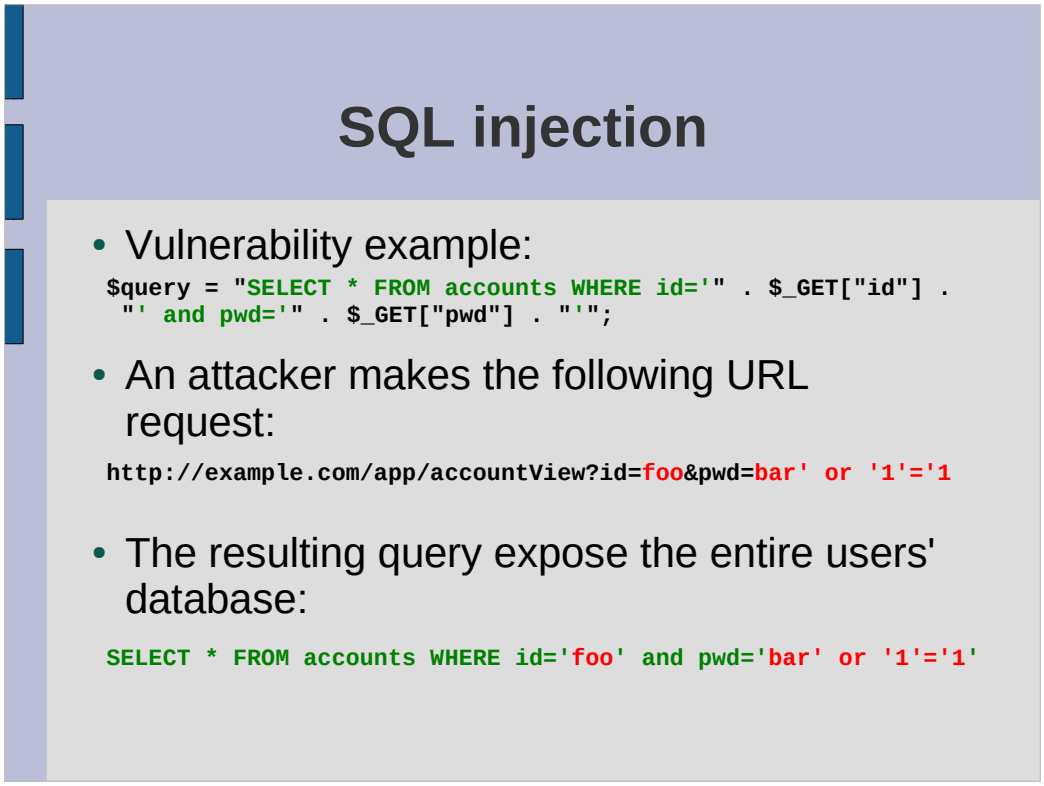
SQL injection

SQL injection

- Example: a web page shows information about a user given the user's id and pwd
- Id and pwd are GET parameters
- URL:

```
https://example.com/app/accountView?id=pippo&pwd=pluto
```

- The user's information is stored in an SQL database



This is an example of SQL injection vulnerability. The PHP code above performs a simple login procedure. It takes a username and a password as inputs by means of URL GET parameters. Then, it concatenates the data to build an SQL query, which retrieves user's information from the accounts' database. This information is then released to the client.

This code works normally if the inputs are those expected, but what if the input, accidentally or intentionally, contains some SQL special characters?

Consider an attacker that sends the above malicious request. The “pwd” parameter contains SQL special characters. When the PHP interpreter builds the query, the result is shown. The “`or '1'='1'`” condition in the WHERE clause bypasses the controls on the username and password and exposes the entire database. If some of this information is confidential, for example passwords or credit card numbers, the users and the website will be in big trouble.

SQL injection

- To add a new user with administrator privileges:

```
https://example.com/app/accountView?id=foo&pwd=bar'; INSERT INTO
 accounts VALUES("Mallory", "pa55word", "administrator"); --
```

- Resulting query:

```
SELECT * FROM accounts WHERE id='foo' and pwd='bar'; INSERT INTO
 accounts VALUES("Mallory", "pa55word", "administrator"); --'
```

- To delete the users' database:

```
https://example.com/app/accountView?id=foo&pwd=bar'; DROP
 TABLE accounts; --
```

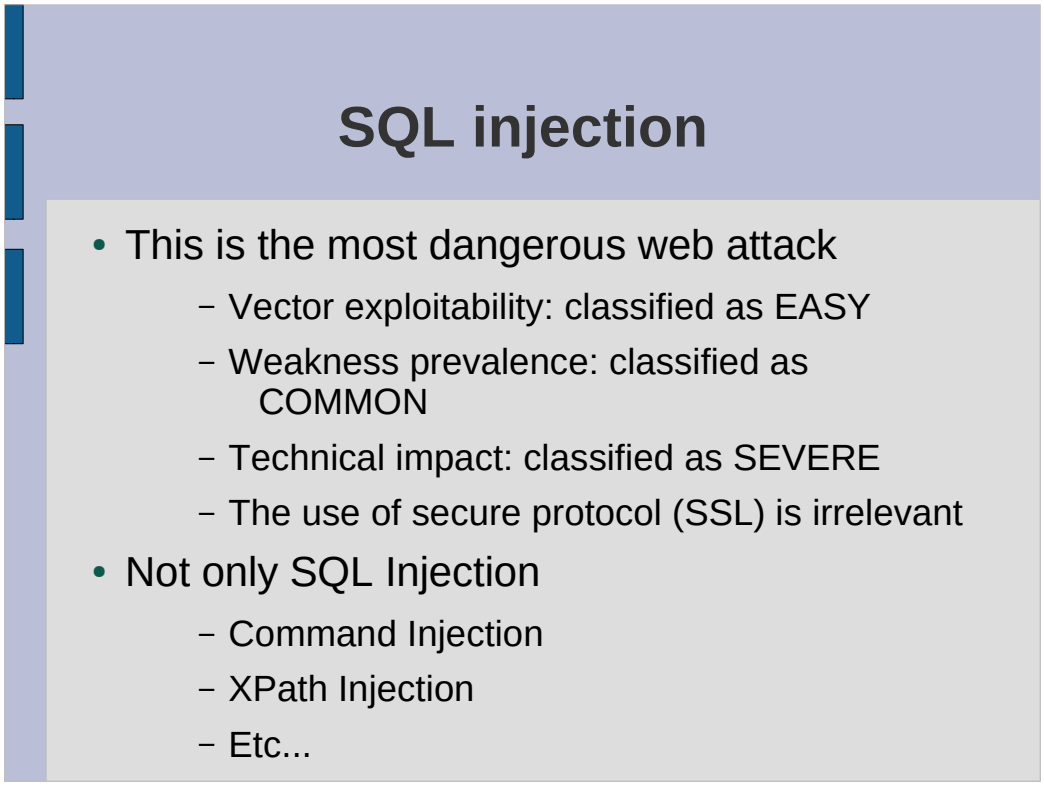
- Resulting query:

```
SELECT * FROM accounts WHERE id='foo' and pwd='bar'; DROP
 TABLE accounts; --'
```

Not only the confidentiality is threatened. With the above malicious requests, the attacker is able respectively to add a controlled administrator user (integrity threat), and to delete the entire users' database (availability threat).

Note the use of SQL spacial characters “`;`”, to separate two SQL commands, and “`--`” (SQL line comment), to eliminate the final single quote.

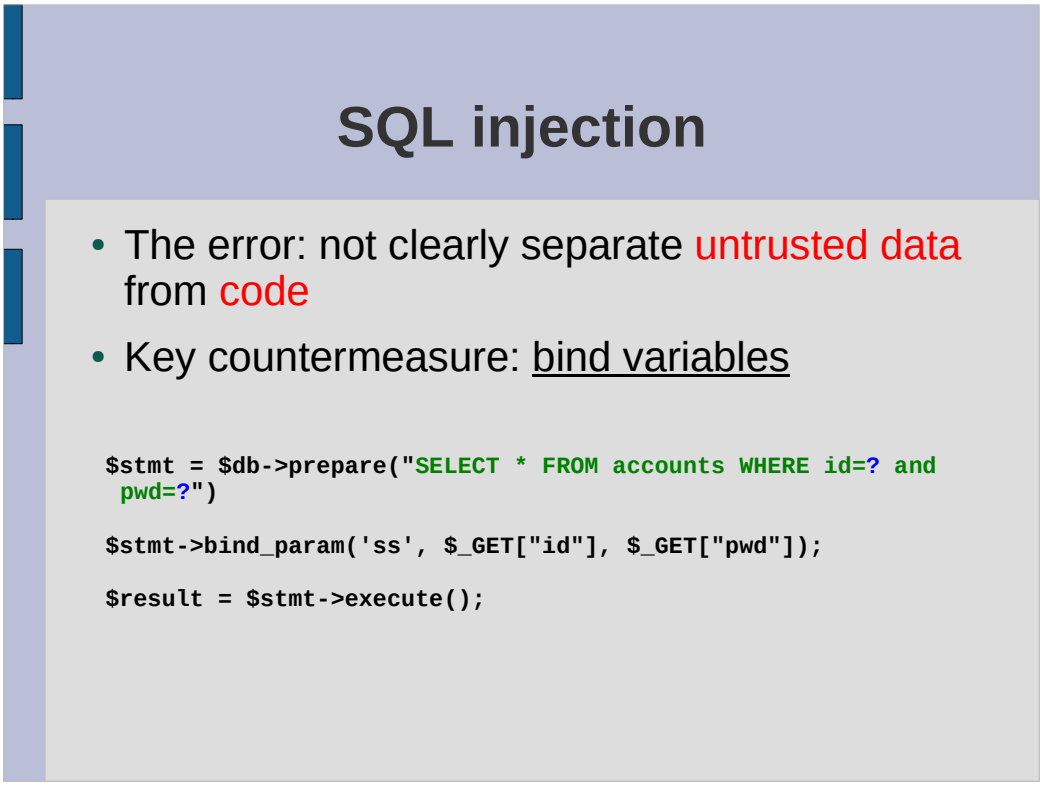
Note also that the attacker must know some implmentation details of the website, namely the SQL dialect and the tables' names and structures. This information is not normally available to the attacker, because the PHP code is not exposed outside the server. However, the attacker can leverage on other methods to scan PHP internal code, for example she can provoke PHP errors and study the debug messages.



Despite it is simple, this is the most dangerous web attack. OWASP Top Ten 2010 categorized its vector exploitability as EASY, the weakness prevalence as COMMON and its technical impact as SEVERE.

Note that the use of a secure transport protocol (SSL, TLS, HTTPS) is IRRELEVANT for the attack, because it works on the higher application level. In fact, from the SSL point of view, the malicious code injected is just normal data.

The SQL injection is the most common case of code injection, but it's not unique. In general, code injection vulnerability is possible whenever an interpreted command (e.g. an SQL query) is constructed using untrusted data and then run. Other possible code injections are Command injection (injection of shell commands), XPath injection (XPath expressions are statements that identify tags or list of tags in an XML document or database), etc.

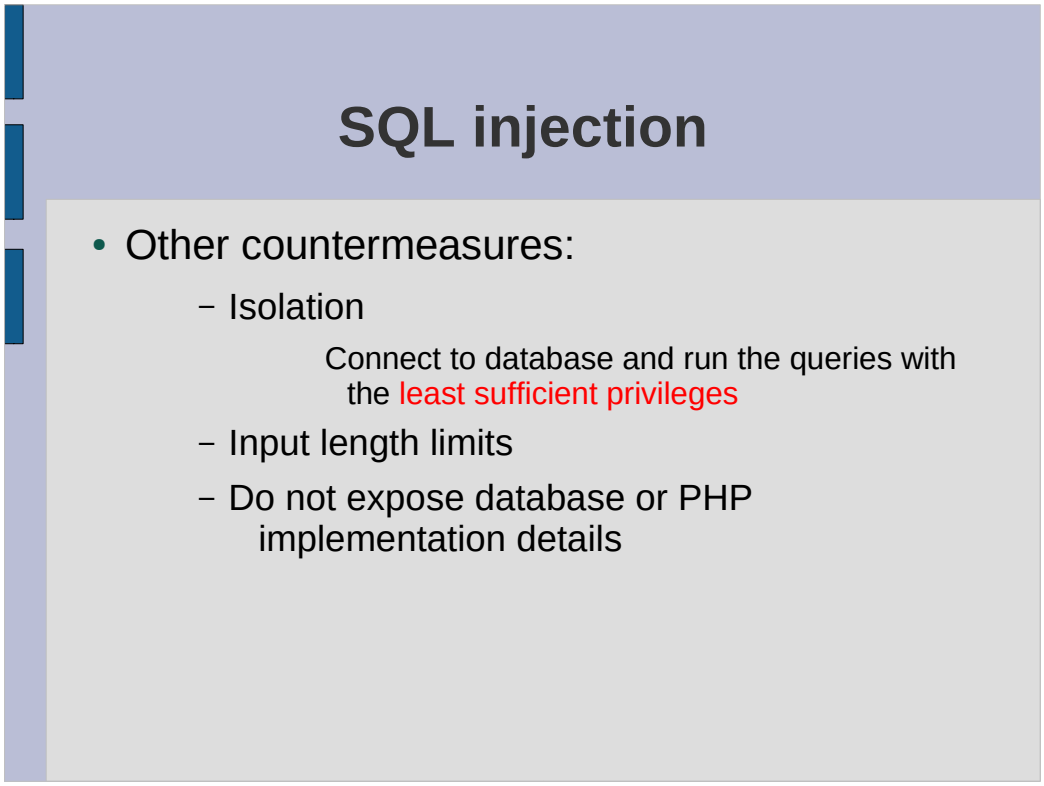


In general, the key error is not to clearly separate untrusted data from code. The solution recommended by OWASP is to use “bind variables”. The technique is referred also as “parametrized query”. An example is reported above.

With the first statement we “prepare” the query, that is, the query is interpreted and represented by an object ($stmt). The query is defined except for two parameters (the “?” characters). At the second stage, these missing parameters are bound to the actual values. Then the query is executed.

Note that with this method, there is a clear separation between code and data. The code is inserted with the “prepare” statement, and it's trusted because it does not contain any untrusted data. The data is inserted with the “bind_param” statement, and we are sure it will be treated as data.

The “ss” parameter in the bind_param method represents the types of the bind variables: two strings.



Other best practices are:

1) The isolation, i.e. connect the database and execute the queries with the least sufficient privileges. Or, at least, never connect a database with administrator privileges.

2) Limit the length of the inputs (GET or POST parameters). This is because many SQL Injection attacks require inserting long and complex input parameters, containing a lot of code.

3) Do not expose database implementation details or PHP code details. Especially on error messages, these details are sent to the browser. An attacker could use this information to seek for vulnerabilities.

SQL injection

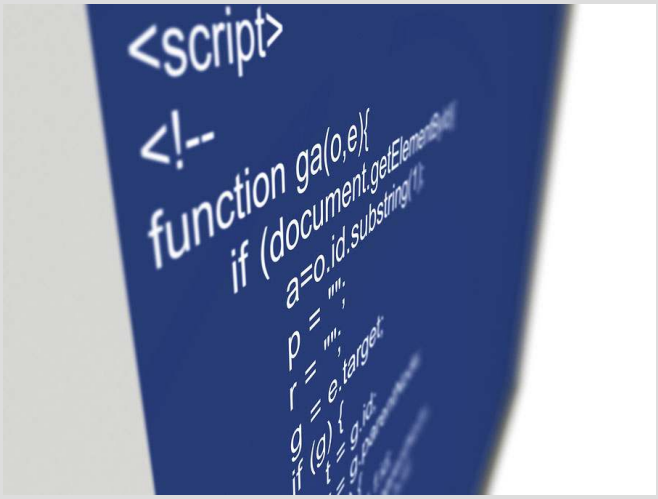
- A famous example: the CardSystem incident
  - June 2005. CardSystems Solutions reported that hackers stole 263,000 credit card numbers, exposed 40 million of them
  - Estimated damage of $16,000,000
  - April 2006. The hacking system was discovered: Code Injection

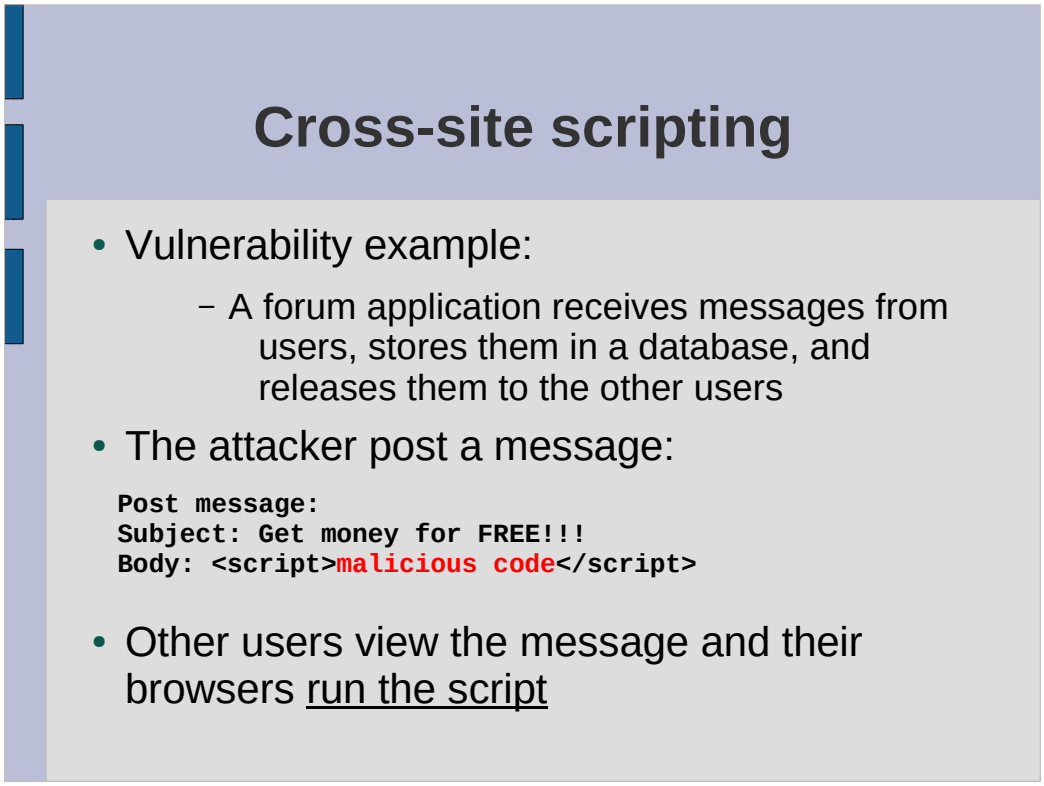
A famous example of Code Injection is the CardSystem incident. CardSystem Solution was a company managing the credit card information and home banking transactions for Visa and Mastercard customers. In June 2005, they reported that hackers have stolen 263,000 credit card numbers and exposed (in cipher text) 40 millions other. The money loss for the company is estimated in $16,000,000, for refunds, lawsuits and fines.

In April 2006, the hacking method was discovered: a Code Injection.

Don't try this at home!

Cross-site scripting





Cross-Site Scripting (XSS) is the second most dangerous web attack according to OWASP. An example is shown above.

A simple forum application receives messages from users, stores them in a database, and then sends the messages to the users that want to read them.

Suppose an attacker sends a message containing a malicious script. Without proper controls, the forum application would store it in the database and send it to other users. The users trust the server, so their browsers execute every script they receive from it. The malicious script is run on a lot of computers.

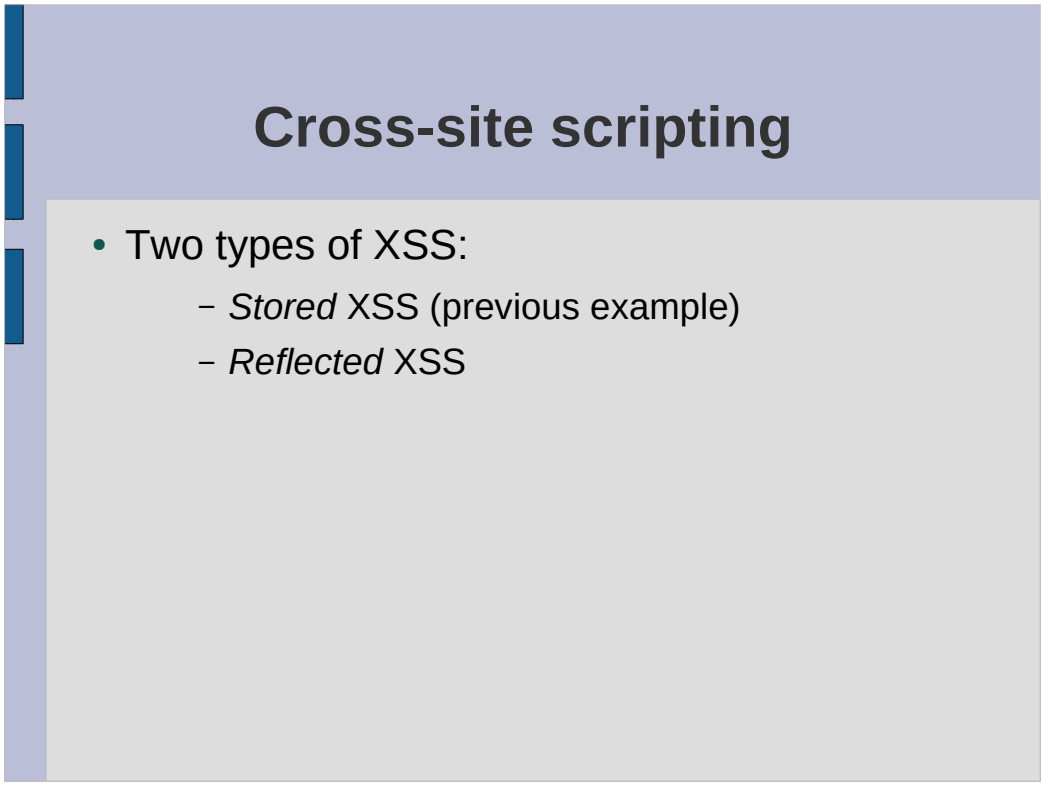
Cross-site scripting

- The vulnerability is on the server, but clients are the victims
- The website is a victim too
- The clients trust the server (digital certificate), so they execute the script
- The browser can't recognize “good” scripts from malicious ones

Note that the vulnerability is on the server, but clients are the final victims. However, the website is a victim too, because it loses credibility.

The key problem is that the clients trust the server. The server could even own a digital certificate and install secure communications with clients. Consider that nowadays almost every site contains scripts, and the browsers are configured not to block the scripts coming from a trusted server.

This attack is very hard to block at the client-side, because the browser can not recognize good script from malicious ones.



There are two main types of XSS:

1) Stored XSS, whenever the script is permanently stored in the server's database and run by many clients, as depicted in the previous example.

2) Reflected XSS, a bit more complicated, whenever the script is not stored, but sent to a single client.

Cross-site scripting

- Example of reflected XSS
- A web page displays an edit field with a default value inside:

CC: 1234567

- The default value “1234567” is a GET parameter

```
https://www.creditcards.com?CC=1234567
```

Cross-site scripting

- Reflected XSS:
  - On the server (www.creditcards.com):

```
echo "<input name='creditcard' type='TEXT' value='" .
 _GET["CC"] . "'>";
```

  - The attacker tricks the user (by email or by another controlled site) to follow the link:

```
www.creditcards.com?CC='><script>malicious code</script>'
```

  - The resulting HTML will be:

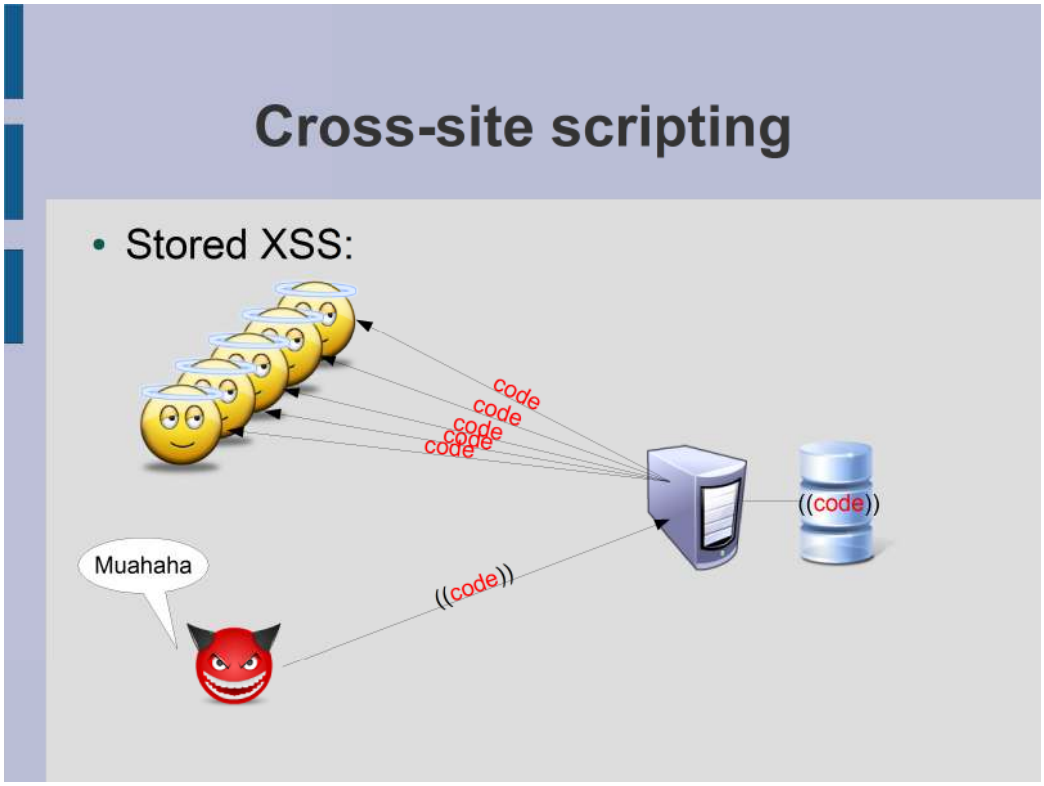
```
<input name='creditcard' type='TEXT' value=''><script>malicious
 code</script>''>
```

The above example shows how a reflected XSS works.

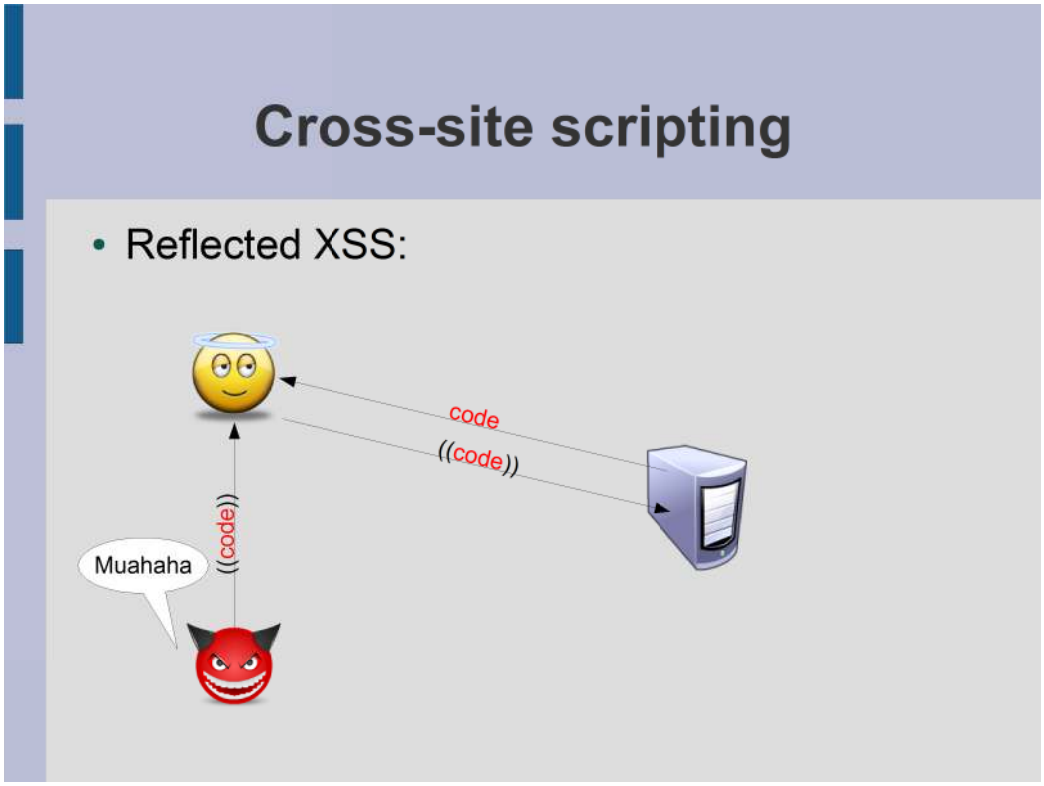
The server at www.creditcards.com builds a dynamic HTML page with an input field. The content of such input field can be specified by the client with a GET parameter. Note that the construction of the dynamic HTML code contains an HTML-Injection flaw.

Suppose an attacker induces an user to follow a particular URL. This can be done through a simple email or even by means of a site controlled by the attacker. The URL contains an HTML Injection attack and some malicious code. The site builds the dynamic HTML page, that contains now the malicious script, and sends it to the user that runs the script.

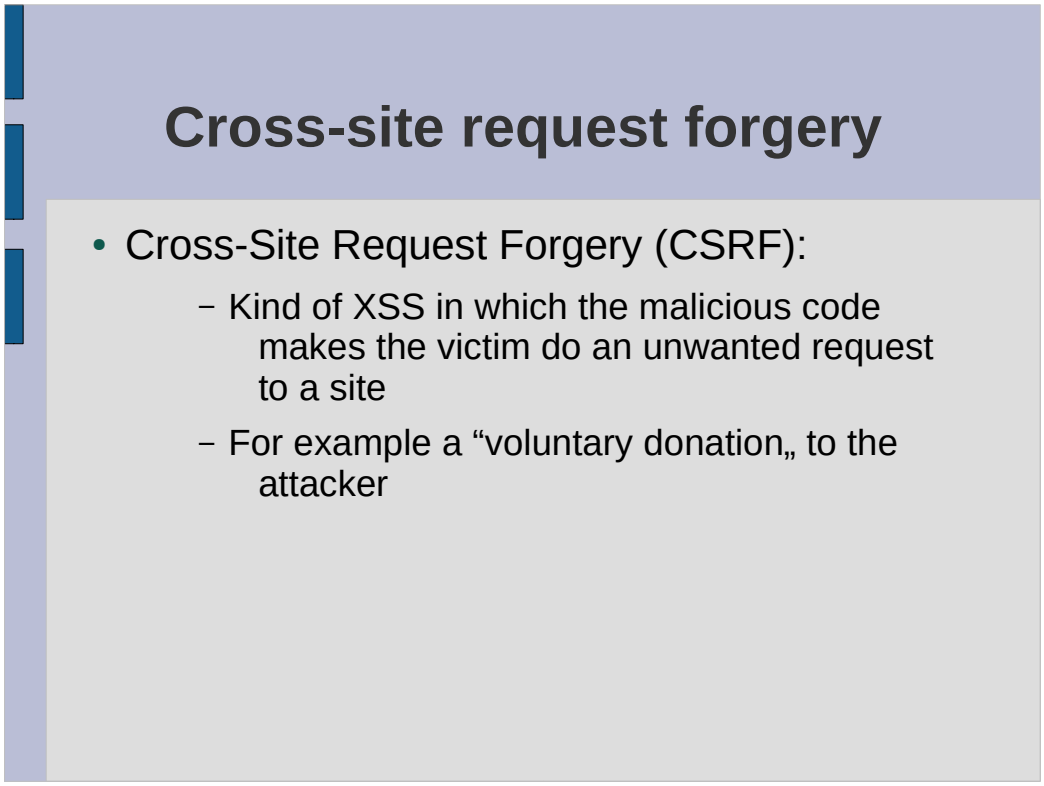
Note that the final “ ' ' >” characters of the resulting HTML is ignored by the browser, for HTML syntax tolerance.



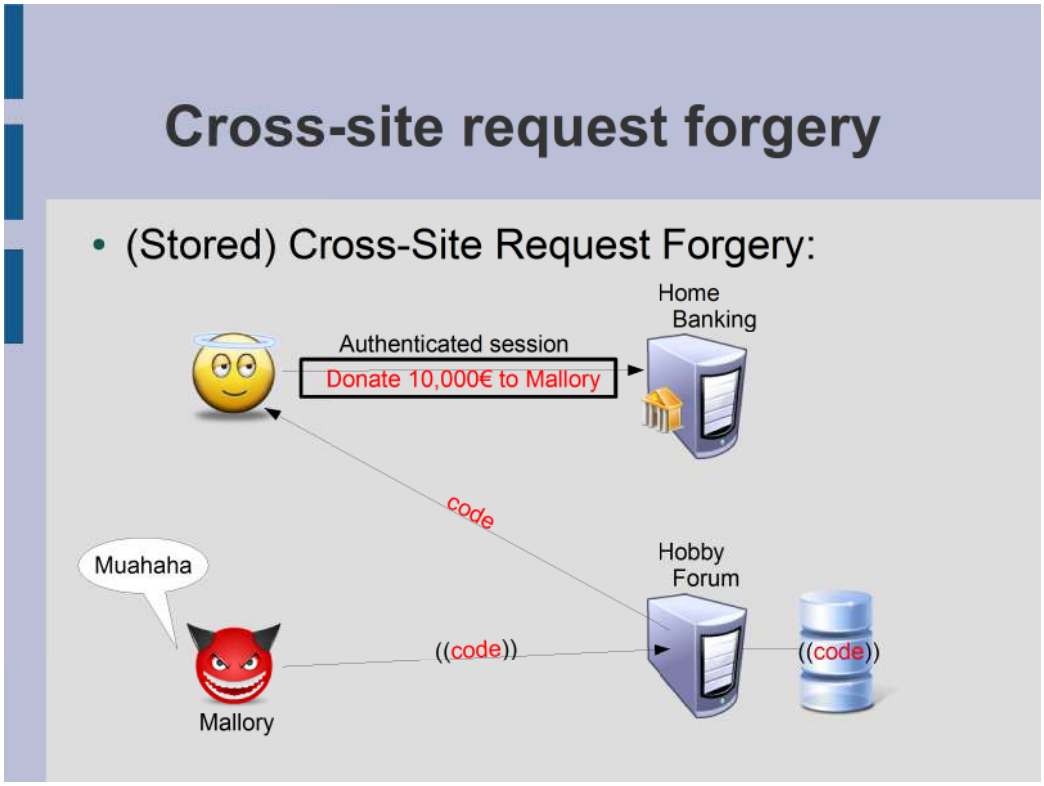
This is the strategy scheme of a Stored XSS attack.



This is the strategy scheme of a Reflected XSS attack.

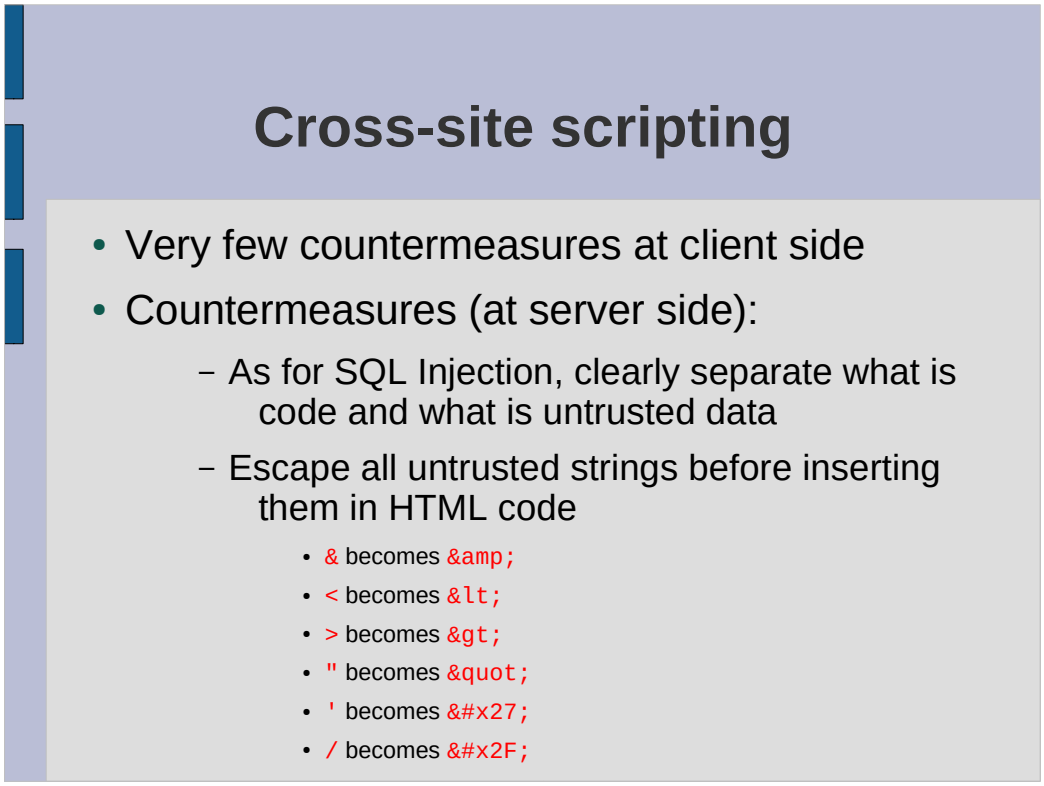


A Cross-Site Request Forgery (CSRF) is a special case of XSS, in which the malicious script performs an unwanted request to a site, for example a “voluntary donation” to the attacker.



The image above shows an example of (Stored) CSRF.

The victim is contemporaneously holding a secure connection to a Home-Banking site and visiting another site, for example the forum of the bank's costumers. Suppose the attacker stores a malicious script on the forum (by Stored XSS) and the victim downloads and runs it. The script uses the open session to make a donation of a lot of money to a bank account controlled by the attacker.



There is no effective countermeasure at the client side, for two main reasons: 1) browsers can't distinguish between “good” scripts and “bad” scripts; 2) we can't force all the people on the world to take security countermeasures.

From the server point of view, we must protect our costumers from an XSS attack. The key countermeasure is, as for SQL Injection, to clearly separate what is code from what is untrusted data. Developers must escape all untrusted strings before inserting them in the HTML code. “Escaping” means to substitute all the special HTML characters (&, <, >, ", ', /) in escaping sequences (respectively `&`, `<`, `>`, `&quot;`, `&#x27;`, `&#x2F;`).

After that the string is safe, and can be concatenated to form HTML code, without problems.

Cross-site scripting

- Escaping methods:

```
$safe = $encoder->encodeForHTML(_GET["CC"]);
```

- The “safe” string can then be inserted in HTML code (by concatenation)

OWASP provided ESAPI (Enterprise Security API), a PHP library for building secure Web Applications. Among other functionalities, there is some methods to properly escape untrusted data for inserting in HTML code. The above example shows how.

ESAPI is available in other languages: Java, .NET, Python, ASP and Coldfusion.

ESAPI library

- ***ESAPI*** (OWASP Enterprise Security API): library for web security utilities
  - PHP
  - Server Java
  - .NET
  - ColdFusion
  - etc.

Denial of service

Denial of service

- Denial of Service (DoS): attack aimed at making unavailable a resource
  - ***Bug-based DoS***
    - Ping of Death
  - ***Asymmetry-based DoS***
    - SYN flood
  - ***Flooding-based DoS***
    - Smurf attack (Amplification-based)
    - Distributed DoS (DDoS)

A Denial of Service (DoS) is an attack aimed at making unavailable a resource (site, application, server, etc.).

There are three main strategies to do this:

1) Bug-based DoS exploits some bug in the input handling of the victim, for example a buffer overflow, to cause a system crash. The bug may reside in the application as well as in the operative system, in the network driver or even in the network card firmware. An (old) example of bug-based DoS is the Ping of Death.

2) Asymmetry-based DoS does not exploit a bug, but an inefficiency of the victim in handling particular inputs. Generally speaking, there is a vulnerability every time an attacker can, using a small amount of her resources, cause a high resources consumption of the victim (hence the name “asymmetry”). An example is the SYN flood attack.

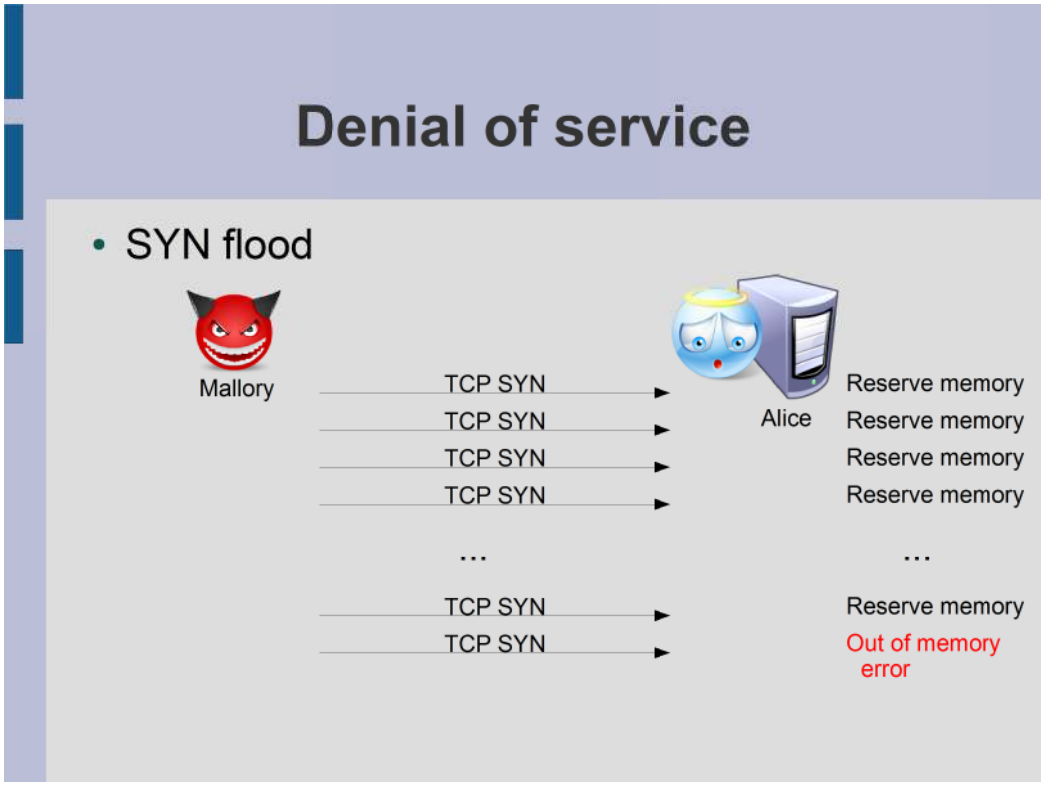
3) Flooding-based DoS does not exploit bugs or inefficiencies. It simply overwhelm the victim with a huge amount of legitimate requests. This can be done through amplification technique (Smurf attack) or through Distributed DoS (DDoS).

Denial of service

- Ping of Death (PoD):
  - IP packet with `fragment_offset=65535` and `payload_size>1`
  - The victim reserves a memory buffer of >65536 bytes
  - Buffer overflow, or at least high resource consumption
  - Windows, Unix, Linux, Mac, printers and routers were vulnerable
  - Corrected in '97-'98

The ping of Death is a famous case of Denial of Service attack. The attacker forges a IP packet (not necessarily a ping packet) that appear to be an IP fragment, with a fragment offset of 65535 bytes (the maximum) and a payload length greater than 1 byte. The victim's operative system would reserve a buffer of >65536 bytes, causing either a buffer overflow or a high resource consumption

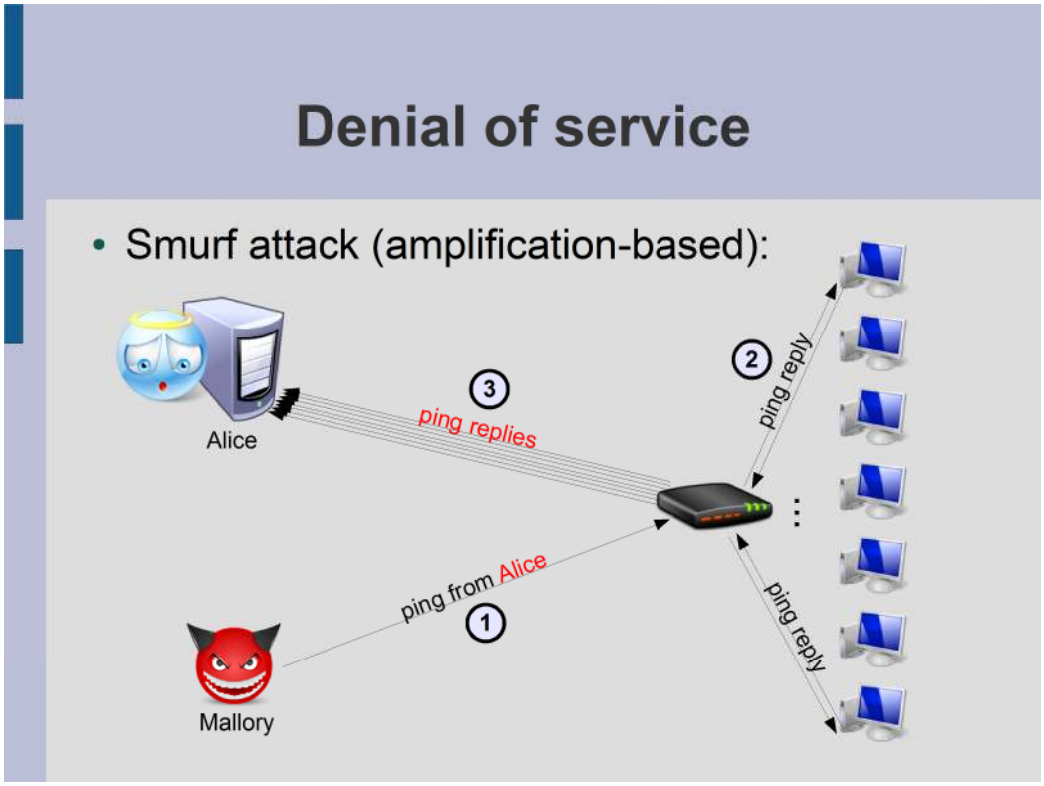
All the major operative systems were vulnerable to this attack, including Windows, Unix, Linux, Max and the dedicated software on network printers and even routers. The vulnerabilities have been corrected in '97-'98.



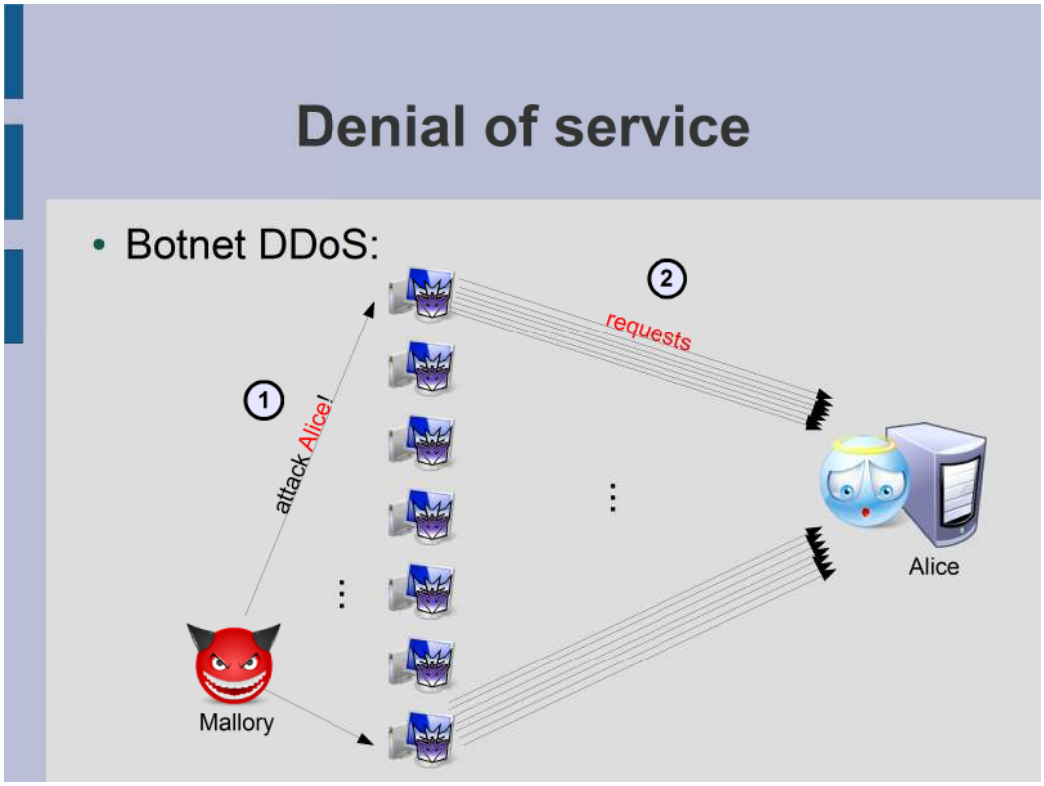
A TCP connection starts with a three-way handshake (SYN, SYN-ACK, ACK). At the reception of the first SYN, the connection is said to be “half-opened” and the server reserves some memory for containing the related information (socket descriptor, etc.).

Suppose that an attacker begins a lot of three-way handshakes without completing them. For each SYN received, the victim is forced to occupy more and more memory. The attack continues until the victim's memory is depleted.

Note that the attacker spends a small resource (the bandwidth to send a SYN packet) but the victim wastes a lot of resources (the memory).



The “Smurf” attack is a classic example of flood-based “amplification” DoS. The attacker produces a ping message with the victim's address as the source IP (spoofing), and sends it in broadcast to a network. The network's gateway is misconfigured and effectively broadcasts the message (amplification). The network reply the ping messages, exhausting the victim's resources (in this case, the IP receiving queue).



In Distributed DoS (DDoS) the attacker induces a lot of honest systems to make a lot of requests to a single site, exhausting its resources.

An example is the botnet attack, in which the attacker installs malware (bot) on honest systems (servers or clients). The bots remain inactive until a specific command is received. After that, the bots start making a lot of requests to the victim. The infected systems are called “zombie agents”, because they act against their will.

A computer can take part of a DDoS attack even with the owner's consent. For example, in 2010, many supporters of WikiLeaks voluntarily ran bots to perform a DDoS attack against the major credit cards companies after Julian Assange was arrested.

Denial of service

- Countermeasures for Bug-based DoS
  - Firewalls
  - Security updates
- Countermeasures for Asymmetry-based DoS
  - Upgrade the server's performance
  - Downgrade the client's performance

Firewalls and security updates are normally used to counter bug-based DoS. We can never be sure that our software is bug-free. So a firewall is mandatory for every system connecting to Internet.

To counter asymmetry-based DoS, we must find methods to break the asymmetry in resource consumption. There are two methods:

1) Upgrade the performance of the server. For example, in the SYN flood attack, the server could put a limit on the number of half-open connections.

2) Downgrade the performance of the clients. An example are puzzle-based defences: a new connection is accepted only if the client is able to perform some computation (puzzle). The puzzle computing time is negligible for legitimate users, but it's huge for attackers.

Denial of service

- Countermeasures for Amplification-based DoS
  - Avoid amplification effects (routers should not broadcast)
- Countermeasures for DDoS
  - The requests are legitimate but with a malicious intent
  - Load balancing between multiple proxy servers
  - Intrusion Detection Systems (IDS, e.g. Snort) analyse the traffic profile

For amplification-based DoS, the best defence is to configure the system to avoid the amplification. In the Smarf attack, no gateway router should be allowed to broadcast packets.

The Distributed DoS is harder to fight. A simple security system like a firewall cannot distinguish legitimate requests from malicious ones. In effect, in most DDoS attacks, the requests forming the attack ARE legitimate (at least in their form), but they have a malicious intent.

Intrusion Detection Systems (IDS) can analyse statistically the incoming traffic and detect DDoS attacks by means of smart algorithms. They can be configured with complex and flexible rules and can set up on-the-fly “deny rules” to firewalls. IDSs are effective also against a large gamma of attacks, including Bug-based and Asymmetry-based DoS.

References

- OWASP online resource collection: https://www.owasp.org/index.php/Main_Page
- M. Bishop *Introduction to Computer Security*, Addison-Wesley Professional
- *Hacking Exposed* series (McGraw-Hill)
  - S. McClure, J. Scambray, G. Kurtz *Hacking Exposed: Network Security Secrets and Solutions*
  - J. Scambray, V. Liu, C. Sima *Hacking Exposed: Web Applications Security Secrets and Solutions*