Web Security

Pierre Senellart

24 April 2014

Plan

Plan

Client-side security

- Anyone can create a Web site and make it referenced by search engines. . . including malevolent individuals
- No *a priori* reason to trust the creators of a visited Web site
- Browsers are supposed to provide a protected environment (sandbox) in which HTML/CSS layout is performed, JavaScript scripts are executed, etc.
- But this protection is not always perfect

Same-Origin Policy (1/2)

- General interaction principle across JavaScript execution contexts: two scripts (in different windows or frames of the browser) can interact only if the context URLs have the same protocol, port, and domain name
- In practice, numerous subtleties
- Possible to communicate across scripts collaborating with the postMessage API
- AJAX restricted in a similar fashion

Same-Origin Policy (2/2)

- No origin restriction for:
  - loading images, videos, applets
  - loading CSS scripts
  - HTML `<iframe>`'s
  - JavaScript loading using the `<script>` tag
- Cookies function in a more liberal manner (the notion of domain is extended, no port or protocol control) for reading/writing, but can be restricted to a path
- Flash, Silverlight, Java plugins implement a similar interaction policy... but sometimes dangerous differences in behavior

Client-side risks

- Erroneous behaviors in software
- Disclosure of confidential data (passwords, credit card numbers, email addresses, etc.)
- Loss of local data (vandalism, ransomware)
- Participation of a local computer to a botnet:
  - Storage of illegal data
  - Relay for other forms of attacks
  - Spam sending

Plan

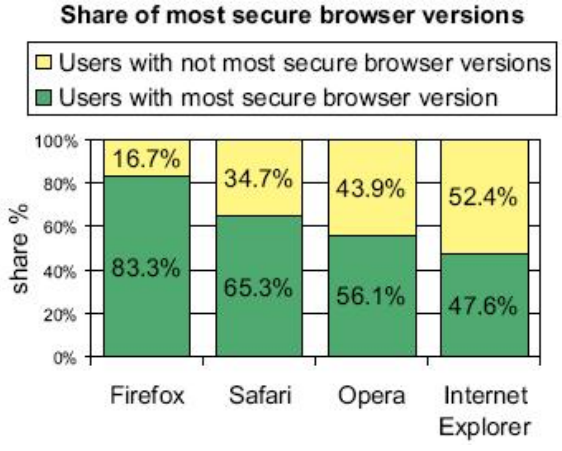
Browser security faults

Problem

A programming error in the Web browser allows unplanned behavior (from erroneous behavior or crashes to distant takeover of the computer).

Solution

Update one's browser very regularly (e.g., through automatic update mechanisms).



http://www.techzoom.net/publications/insecurity-iceberg/index.en

Security issue in another component

Problem

A programming error in an extension of a Web browser (ad blocker, Java, Flash or PDF plug-in, localization package, multimedia player, etc.) allows unplanned behavior.

Solution

Update all such software as regularly as possible (may be tricky). Remove unused extensions. Get informed about most important known issues. Prefer extensions whose update mechanisms are integrated within that of the operating system (or of the browser).

Plan

Malicious code execution

Problem

A user launches an executable, a Java applet without a sandbox, an ActiveX control, etc., linked from a Web page; arbitrary code can be run.

Solution

Take heed of browser warnings! And think it over!

Clickjacking

Problem

A user visiting a malicious Web site is led to click at a dangerous location (browser warning, zone within a frame belonging to a trustworthy Web site, etc.) by masking the area where the click is going to happen until the last minute.

Solution

Timing between display and click on an important action within browsers; be careful of weird behaviors of the mouse pointer within a site.

Phishing

Problem

Through a link (in an email, on the Web, etc.), a user is led to a Web page he thinks is that of a site he trusts (online banking, electronic commerce, etc.) and is asked for his connection identifiers. The site is only an imitation of the official site, and these identifiers are disclosed to malevolent individuals.

Solution

Always carefully check the URL of a site one reaches through a link. In case a site handles sensitive information, or allows dangerous operations, only use it with HTTPS, and check the SSL certificate (browsers display the validated identity of the site owner in the address bar). Use your common sense!

Plan

IP packet capture

Problem

On a local wired network, on an unencrypted WiFi network (or with an easy-to-crack WEP encryption), an attacker can monitor IP packet exchanges that contain all communication between a browser and a Web server, including all HTTP parameters, etc. This problem also exists, but less dramatically, on the Internet at large.

Solution

Do not use HTTP to transmit sensitive information through the Internet, even more so when on a shared wired network or unencrypted WiFi network. HTTPS, another protocol allowing encryption of every message on the Web, must be used.

Session spoofing

Problem

Using one of the techniques presented in this lecture (e.g., XSS or IP packet capture) to retrieve the session identifier of a user (usually stored within a cookie) to usurp her identity.

Solution

Solve other issues! On the server side, allow disconnecting (in PHP, with session_destroy) as soon as the session is not useful anymore.

Plan

Plan

- Web : hostile environment
- Unless one controls access to the Web server, anyone can access a Web site. . . including malevolent individuals
- Some things are the responsibility of the system administrator (e.g., having up-to-date Web server software), others are that of the Web master

Server-side risks

- Disclosure of confidential data (passwords, credit card numbers, email addresses, etc.) of your users
- Jeopardize the security of your users
- Loss of local data (vandalism, ransomware)
- Participation of a local computer to a botnet:
  - Storage of illegal data
  - Relay for other forms of attacks
  - Spam sending

Validation of HTTP parameters

One can impose a number of restrictions on data that can be put in a Web form, on the client side:

- `maxlength` on a `<input type="text">` for a maximal length on text fields
- `<select>`'s, radio buttons, checkboxes forcing to choose one (or several) proposed values
- A `<input type="hidden">` hidden field or a `readonly` field sets a parameter's value
- JavaScript validation code runs arbitrary verifications

Validation of HTTP parameters

One can impose a number of restrictions on data that can be put in a Web form, on the client side:

- `maxlength` on a `<input type="text">` for a maximal length on text fields
- `<select>`'s, radio buttons, checkboxes forcing to choose one (or several) proposed values
- A `<input type="hidden">` hidden field or a `readonly` field sets a parameter's value
- JavaScript validation code runs arbitrary verifications

But all of this is only true if the Web client plays nice. Very easy to overcome (turning JavaScript off, modifying the page).

Never ever trust data sent by a Web client! Always (re)do the verifications on the server side.

Plan

HTML injection

Problem

A user can put, within an HTTP parameter that will be displayed in the page, HTML code (and therefore CSS styling or JavaScript code). He thus modifies the code of the produced HTML page. If this parameter is stored to be displayed again (e.g., blog comments), the code can influence the content of the site for other users.

Example

Assume an HTTP parameter `login` contains:
`<div style='color: red'>` within the PHP code:

```
echo "Bonjour ".$_REQUEST["login"]." !";
```

Solution

Use functions for escaping HTML special characters (in PHP, htmlspecialchars).

XSS (Cross-Site Scripting)

Problem

Special case of the previous attack: inserting JavaScript code inside an HTML page, which will be displayed to other users; the JavaScript code “steals” information typed by a user to transfer them to another server.

Solution

As before, use htmlspecialchars, in particular when some text filled in by a user will be displayed to another.

XSRF (Cross-Site Request Forgery)

Problem

A third-party site loads (e.g., in an `<iframe>`) a page of the attacked site. The browser accesses this page, logged in as the original user, which may allow the attacker to accomplish an action instead of the user (e.g., ordering something) or infer information about the user.

Solution

- Use the HTTP POST method for every behavior that has side effects.
- Avoid inclusion of a page within another. (with the `X-Frame-Options` HTTP header).
- Make sure access to this page is controlled by non-guessable unique tokens.

SQL injection

Problem

A user can modify a SQL query by an appropriate choice of special characters in SQL in a variable used to construct the query.

Example

Assume `$passwd` contains "`' OR 1=1 --`" in the PHP code:

```
 $result=mysql_query("SELECT * FROM T WHERE
passwd='$passwd'");
```

Solution

Use escaping functions for SQL characters (e.g., `mysql_real_escape_string` in PHP). Better, never construct a query like this and use prepared queries.

Command line injection

Problem

A user can modify calls to external programs on the server side (e.g., in PHP, called with `system`, `exec`, `passthru`, etc.).

Example

Assume `$rep` contains "`&& cat /etc/password`" in the PHP code:

```
passthru("ls $rep");
```

Solution

Avoid calling external programs from the Web server. Carefully check the content of command line. In PHP, use escapeshellcmd or escapeshellarg to protect the special characters of the shell.

Plan

Directory traversal

Problem

A user has the possibility, when accessing files on a server (e.g., in PHP, with `fopen`, `readfile`), using '/', '..', to gain access he is not supposed to have.

Example

Assume `$file` contains "`../../../../../etc/passwd`" in the PHP code:

```
readfile($fichier);
```

Solution

Check that arguments of functions that access files do not point to files that are not supposed to be accessible (e.g., check there is no '/' inside).

Plan

Race condition

Problem

An attacker can induce unplanned behavior on the server side by exploiting a programming error that assumes one block of instructions to be executed as one, with no concurrence from other instructions.

Example

A PHP script retrieves the largest integer stored in a MySQL table, increments it, saves a file with name this integer. Concurrence if two scripts run simultaneously, and both look up the table to know the largest integer, before adding a line.

Solution

Think about race conditions cases. Use transactions of the DBMS (e.g., InnoDB in MySQL 5), use locks on files.

Denial Of Service (DOS)

Problem

Attack a Web site (or whatever Internet service) by requesting from the server more than it can handle (very large number of connections, costly computation, etc.).

Solution

Essentially the responsibility of the administrator, but the Web master can prevent some attacks by 1) avoiding files that are too heavy to download 2) avoid useless computation on the server side.

Brute-force password cracking

Problem

Weak passwords (common or proper nouns without or with little variation, very short) can be cracked by brute force, by exploring one by one a list of all possible words; even less costly if a (crypted) password file can be retrieved locally.

Solution

Impose strong passwords to users (and oneself). Never make available a password file, even crypted. Monitor accesses to a Web site that try out various passwords, and block them.

Social engineering

Problem

Likely the most used form of attack: exploit a fault not in software, but in human reasoning! Force a honest user to give confidential information, etc.

Solution

Keep a critical mind in every thing, use common sense, and never get abused by technical ignored!

Other common sense rules

To prevent attacks or to limit their consequences:

- Use known cryptography algorithms, not home-made cyphers
- Never store passwords in a database, only store non-reversible cryptographic hashes
- Never store credit card numbers or other sensitive information in a database
- Make sure the Web server software has limited access rights on the computer it runs on
- Do not blindly trust third-party code
- Think it over and put oneself in an attacker's shoes

Plan

Client-Side Security

Server-Side Security

References

References

- *The Tangled Web: A Guide to Securing Modern Web Applications*, Michael Zalewski, to understand how Web technologies interact security-wise
- *The Art of Deception*, Kevin Mitnick, to get familiar with the dangers of social engineering
- *Hacking, the Next Generation*, Nitesh Dhanjani, Billy Rios, Brett Hardi (O'Reilly) for concrete examples of modern attacks
- Firebug for Firefox, or other browser features and extensions, to study and analyse sites from the client side
- Hacking challenge sites like `http://www.hackthissite.org/` to put oneself in an attacker's shoes

Law of the Republic of Indonesia #11 of 2008

Article 30 (3)

Setiap Orang dengan sengaja dan tanpa hak atau melawan hukum mengakses Komputer dan/atau Sistem Elektronik dengan cara apapun dengan melanggar, menerobos, melampaui, atau menjebol sistem pengamanan.
Any Person who knowingly and without authority or unlawfully accesses Computers and/or Electronic Systems in any manner whatsoever by breaching, hacking into, trespassing into, or breaking through security systems.

Article 46 (3)

Setiap Orang yang memenuhi unsur sebagaimana dimaksud dalam Pasal 30 (3) ayat dipidana dengan pidana penjara paling lama 8 (delapan) tahun dan/atau denda paling banyak Rp800.000.000,00 (delapan ratus juta rupiah).
Any Person who satisfies the elements as intended by Article 30 section (3) shall be sentenced to imprisonment not exceeding 8 (eight) years and/or a fine not exceeding Rp800,000,000 (eight hundred million rupiah).