IIG University of Freiburg

Web Security, Summer Term 2012

1) What is a web application ? What is YOUR web application?

Dr. E. Benoist

Sommer Semester

Table of Contents

What is a Web Application?

From the point of view of the client

- **Originally:**
  - A set of web pages connected using hyperlinks or forms
  - Pages are HTML files that contain CSS and images
- **Where do we have programms?**
  - Client Side : JavaScript is a language that can manipulate the HTML tree (Document Object Model - DOM).
  - Server Side programming is used to generate the HTML (or XHTML)

Static content

- **HTML Tags**
  - HTML tags build a Document Object Model (DOM)
  - A tree where each tag is a node and texts are leaves
- **Cascading Style Sheets**
  - CSS contain the layout
  - Which part has which color or font
  - Which part is visible or where is it placed

Dynamic Content

- **JavaScript**
  - Source code is transfered and evaluated on the client
  - Manipulates the DOM,
  - Center of AJAX applications (communication with the server) - Same Origin Policy
  - Can also be connected to third servers using JSON (a little bit more complicated)
- **Flash**
  - SWF program downloaded into the client (compiled code)
  - Programm executed inside the browser
  - Usually inside a sandbox (exceptions can be asked to the user)
- **Java Applets**
  - Java program compiled (`.class` or `.jar`)
  - Executed inside a sandbox
  - Exceptions can be authorized for signed applets.

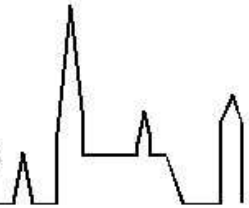
What is a web application?

From the point of view of the prgrammer

- **Simple architecture**
  - Typically a LAMP infrastructure (Linux, Apache, MySQL, PHP)
  - PHP accesses directly to MySQL and generates HTML
- **Other simple architectures**
  - Microsoft ASP
  - Servlet

Frameworks

- **Java Enterprise Edition - JEE**
  - Different independant layers
  - DataBase
  - Persistancy (Hibernate - Java Persistance API)
  - Business Logic in Java
  - Presentation Tier in Java Server Faces (JSF) or Struts, using a Model View Controler (MVC) pattern.
- **Framework for PHP: Zend**
  - Model View Controler
  - Layout Manager
  - DataBase Manager and Persistance Layer



- **Specific Type of Web Applications**
  - Stores the content in a DataBase
  - Displays it in a standardized form
  - Allows specific users to input new content (so called Back End uses)
  - Whereas most of the users are only "Front End users"
- **Structure of a CMS**
  - CMS's are based on a core system (define pages, basic content, navigation)
  - Features can be extended
  - Programmers have access to an API to integrate new functionalities
  - They can develop and sometime exchange new applications inside the CMS

CMS: Specificities

- **Open programs**
  - Most of them are Open Source
  - Or at least accessible for off-line testing
- **Extended with less secure "Extensions"**
  - Developped by standard developper
  - Given to the community without being tested seriousely
  - often available open source
- **Default configuration is known**
- Default admin password
- Default install page
- Default admin page
- ...

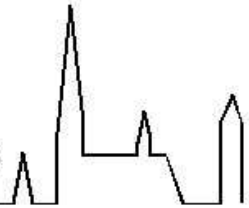
What is YOUR application?

- **Which language is used**
  - Java, PHP, C#
- **Which architecture is used**
  - Framework, layers,
- **Which CMS is used**
  - How is it configured
  - What are the installed extensions
- **Which structure has the web site**
  - Pages, navigation, secure area, unsecure area

What is the web site?

- **Who is the owner / hoster**
  - Official owner : Check the Who-Is Entry
  - Check the IP range to know the hoster
  - See in which Country it is hosted
- **Examples:**
  - `http://whatismyipaddress.com`
  - `http://www.whois.net/`
  - `https://www.nic.ch/reg/ds03/whois/view.html?lid=fr`

Draw a Site Map



- **Which map?**
  - Resources and their parameters
  - Coresponds to pages / subpages
  - Forms
- **Use a spider**
  - `wget` for instance
  - Visits the links from one page to the other

Site Map

- **Problems with spiders**
  - Fully authomated
  - Web Site created by complicated JavaScript
  - Input for forms are validated, they must be valid
- **Use WebScarab's Spider**
  - Run a spider over a web site
  - Visit all links starting on a page
  - Folows also forms : with default values
  - Values can be configured (e.g. username and password)
  - Shows a map of the site (from the point of view of the resources)
  - Shows for each link, the different values of parameters
- **Surf by yourself**
  - Visit the site systematically
  - Web Scarab can grab information and search for more

View the HTML source

- **Understand the structure of the page**
  - `div`'s and `tables`
  - Which CSS and JavaScript are loaded, for which purpose
- **See comments**
  - Can describe the structure
  - Can contain commented parts of the code
  - Description of the data and database
- **See meta-information**
  - Generator, Author

View the HTML Source (Cont.)

- **Access to internal ID's**
  - View the page ID's
  - View user ID's
- **View other internal structures**
  - View the files requested

Encoding

- **URL encoded**
  - only with characters, + and encodes all other chars using %XX notation
- **Base 16 (Hexadecimal)**
  - numbers between 0 and 15
  - Encoded using digits (0 to 9) and letters A B C D E and F
  - Example: a Mac address 00:16:00:89:0a:cf
- **Base 64**
  - Can be used to encode any string (is more verbose)
  - Should contain a number of chars that is a multiple of 4
  - Otherwise the string is completed with placeholders (=)
  - Example *YmllOmJpZQ==*

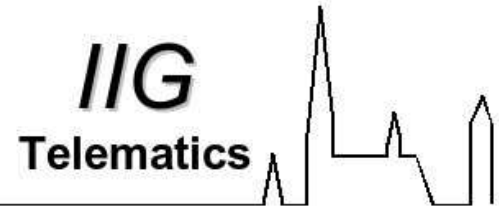
Use Google to access structure information

- **Return every resource within the target site which Google has a reference to.**
  - `site:www.my-target.ch`
- **All the pages with the expression `login` on the site**
  - `site:www.my-target.ch login`
- **Return all of the pages on other web sites and applications that contain a link to the target**
  - `link:www.my-target.ch`
- **Return pages that are similar to the target**
  - `related:www.my-target.ch`

Discover Hidden Content

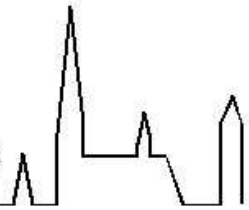
- **Suppose we discovered the following resources**
  - `www.my-target.ch/login.php`
  - `www.my-target.ch/home/myaccount.php`
  - `www.my-target.ch/home/logout.php`
- **We should search for additional directories**
  - `www.my-target.ch/access/`
  - `www.my-target.ch/account/`
  - `www.my-target.ch/admin/`
  - `www.my-target.ch/administration/`
  - `www.my-target.ch/management/`
  - `www.my-target.ch/templates/`
  - `www.my-target.ch/smarty/`
  - `www.my-target.ch/inc/`
- **Then look for possible files in each of the directories**
- `access.php`, `login.php`, `password.inc`, `pwd.inc`, `User.class.php`, ...

Discover Hidden Content (Cont.)



- **See how errors are handled**
  - Page Not Found (404 or 200 with error message)
  - Directory listing or not
- **Use the map generated as a basis for automated discovery of hidden content**
- **Make automated requests for common filenames and directories within each directory or path known to exist within the application.**
  - Use `Burp Intruder` or a custom script with wordlists of common files
  - Configure the software to handle the invalid responses
- **Manualy review the responses received to identify valid resources**
  - Perform the exercise recursively as new content is discovered

Identify Entry Points for User Input



- **Review HTTP Requests to find entry points**
- **Key Locations to pay attention to:**
  - Every URL string up to the query string marker
  - Every parameter submitted within the URL query string
  - Every parameter submitted within the body of a `POST` request.
  - Every cookie
  - Every other HTTP header that in rare cases may be processed by the application (e.g. `User-Agent`, `Referer`, `Accept`,`Accept-Language`, and `Host`
- **Some applications do not employ the standard query string format**
  - `/dir/file;foo=bar&foo2=bar2`
  - `/dir/file?foo=bar$foo2=bar2`
  - `dir/foo.bar/file`
  - `dir/foo=bar/file`
  - `dir/file?param=foo:bar`

- **Banner Graining**
  - In Http Response Header
  - example:

    Server: Apache/2.2.11 (Unix) DAV/2 mod_ssl/2.2.11 OpenSSL/0.9.8l

  - Easy to restrict

    # Set to one of: Full | OS | Minor | Minimal | Major | Prod
    # where Full conveys the most information, and Prod the least.
    #
    ServerTokens Full

- **HTTP Fingerprinting**
  - Some web servers can deliberately falsify the banner
  - It is usually possible to determine which server is used
  - `Httprint` is a handy tool that performs a number of tests in an attempt to fingerprint a web server's software.

File Extensions

- **File extensions used inside URL's often disclose the language used:**
  - `asp` Microsoft Active Server Pages
  - `aspx` Microsoft ASP.NET
  - `jsp` Java Server Pages
  - `cfm` Cold Fusion
  - `php` the PHP language
  - `d2w` WebSphere
  - `pl` the Perl language
  - `pw` the Python language
  - `dll` usually compiled native code (C or C++)
  - `nsf` or `ntf` Lotus Domino

Session Tockens

- **Session Tockens often allow to see which language is used**
  - `JSESSIONID` The Java Platform
  - `ASPSESSIONID` Microsoft IIS server
  - `CFID/CFTOKEN` Cold Fusion
  - `ASP.NET_SessionId` Microsoft ASP.NET
  - `PHPSESSIONID` PHP

Third Party Code Components

- **Review the Map of the Application**
  - Find interesting-looking file extensions
  - Can also be a subsequence in a URL
- **Review the names of all session tokens issued by the application**
  - Help to identify the technologies being used
- **Use list of common technologies or Google to establish which technology is used**
  - Search for cookies or particular extensions for example
- **Perform searches on Google for the names of any unusual cookies, scripts, HTTP headers, . . .**

Thick Application Components

- **Ajax Projects are mainly based on JavaScript**
  - Download and study the structure of the code,
  - Where are which events called
  - Where does the communication takes place (Ajax requests), and how are the responses treated
- **Web site may contain thick client components**
  - Java Applets
  - ActiveX Controls
  - Shockwave Flash objects

Uncompile a thick client component

- **Java Applet**
  - Java Applet are compiled and integrated as bytecode
  - They can be `.class` or `.jar` files
  - Download each file, `unzip` the `jar` files, check if they correspond to an opensource library
  - Uncompile the rest files: `jad` can uncompile byte code, `Jode` is a tool for uncompiling or obfuscating your code
- **Source code can be obfuscated**
  - Source can be hardly read (and not recompiled), but signature of public methods remain stable
  - You can use JavaScript to access the Java public methods and see what it does.

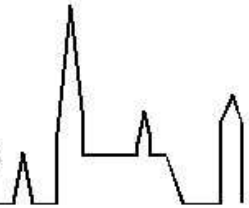
Uncompile Flash objects

- **Flash animations are deployed in `.swf` format**
  - Compiled bytecode
  - Downloaded by the browser, then executed in the flash plug-in
- **The byte code can also be uncompiled**
  - You can access the (almost) original ActionScript source
  - Need to be recompiled
- **Alternative: simply desassemble the byte code**
  - `flasm` unassemble byte code
  - You can edit directly the file and reassemble it

Mapping the Attack Surface

- **Last step: map the attack surface of the application**
- **Key features to look at and the corresponding vulnerability**
  - *Client-side validation* Checks may not be replicated on the server
  - *Database interaction* SQL Injection
  - *File uploading and downloading* Path traversal vulnerabilities
  - *Display of user-supplied data* Cross-site scripting
  - *Dynamic redirects* Redirection and header injection attacks
  - *Login* Username enumeration, weak passwords, ability to use brute force
  - *Multistage login* Logic flaws
  - *Session state* Predicatable tokens, insecure handling of tokens

Mapping the Attack Surface (Cont.)



- *Access controls* Horizontal and vertical privilege escalation
- *Use of clear text communications* Session hijacking, capture of credentials and other sensitive data
- *Off-site links* Leakage of query sting parameter in the `Referer` HTTP-Header
- *Interface to external systems* Shortcuts in handling of sessions and/or access controls
- *Error messages* Information leakage
- *Email interaction* Email and/or command injection
- *Native code components or interaction* Buffer overflows
- *Use of third-party application components* Known vulnerabilities
- *Identifiable web server software* Common configuration weakness, known software bugs.

Conclusion

- **In this course, we only access to publicly available information**
  - No evil interaction is done with the server
  - Just interact with the server and see how it has been built
- **Information is already large**
  - One need to understand the logic of the developper
  - One need to understand how the system is build,
  - Idea: Find the weak points fast
- **Use google extensively**
  - To search for known vulnerabilities on a generic system
  - To search for information about one specific system
  - To gather information about one site
- **"Know your enemy"**
  - Gather information
  - Do not forget social engineering! (Facebook, 123people.com, etc.)

References

- **Web Security Testing Cookbook**, Paco Hope and Ben Walther, O'Reilly, 2008.
- **The Web Application Hacker's Handbook**, Dafydd Stuttard and Marcus Pinto, Whiley, 2008

- **Tools**
  - `Web Scarab` from the OWASP Project `http://www.owasp.org/`
  - `Burp Intruder`
  - `Httprint` determine the server name using its fingerprinting
  - `jad` or `jode` to uncompile java applets