Application Layer Security
General overview
Ma. Angel Marquez Andrade
 › Benefits of web Applications:

- No need to distribute separate client software
- Changes to the interface take effect immediately
- Client-side scripting pushes processing to the client
- The technologies have been standardized
Current web applications handle sensitive data and functionality:
- Access payroll information
- Sharing personal documents
- Enterprise reports and resource planning software
- Financial institutions
- E-commerce
<table>
<thead>
<tr>
<th>CIA</th>
<th>Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>Loss of privacy. Unauthorized access to information. Identity Theft</td>
</tr>
<tr>
<td>Integrity</td>
<td>Information is no longer reliable or accurate. Fraud</td>
</tr>
<tr>
<td>Availability</td>
<td>Business disruption, Loss of customer confidence, Loss of revenue</td>
</tr>
</tbody>
</table>

- **Risk**: the chance a risk event will occur and the loss or harm resulting from the occurrence.

- **Return On Investment (ROI)**: identify security measures yielding a positive return
- **Cost To Break (CTB)**: lowest expected cost for anyone to discover and exploit a vulnerability
• OWASP Top 10 focused on identifying the most common vulnerabilities, but were also designed around risk measures
Open Web Application Security Project (OWASP) Foundation is a non-profit organization.

- Enables organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted.
- Produces open-source documentation, tools, and standards.
- Facilitates conferences, local chapters, articles, and message forums.

OWASP Top 10 Application Security Risks

1. Injection
2. Cross Site Scripting (XSS)
3. Broken 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 Redirects and Forwards
Foreword:

“We can no longer afford to tolerate relatively simple security problems like those presented in the OWASP Top 10”

“...digital infrastructures get increasingly complex and interconnected”

“Insecure software is already undermining our ... critical infrastructure”
Web sites of the past

› Repositories of static documents.

› Before there was no sensitive information, the server was already open to public view.

› Main problem:
  - Vulnerabilities in server software.
  - Site defacing, stealing server’s storage and bandwidth.
Web applications

Key Problems:

- Third party packages abstract developers from underlying technologies (less security awareness)
- Ready made code vulnerabilities affect many unrelated applications.
- Time constraints to develop the application
- Security through obscurity
- Increasing functionality demands
Present core security problem:

Users can supply arbitrary input

- users can interfere with request parameters, cookies, and HTTP headers.
- users can send requests in any sequence.
- users are not restricted to using the web browser only.
Welcome to Atlas
Surname: M*
First Name: 
E-mail Address: 
Telephone Extension: 
Title: 
Department: 
Scope: Regular Search
Sort By: Surname
Search Results
You searched for: Surname: M* Scope: Regular Search Sort By: Surname

* Your search has exceeded the maximum of 250 allowed results.
If you do not find what you are looking for, please provide a more specific search by changing the search criteria.
Displaying only the first 250 results found.

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Telephone</th>
<th>Voicemail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colaillo, Marianna</td>
<td>Student Services &amp; International Relations, Schulich School of Business</td>
<td>(416)736-5081</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(416)736-2100 x 20654</td>
<td></td>
</tr>
<tr>
<td>Kensett, Mathew</td>
<td>Art Gallery, Glendon College</td>
<td>(416)487-6722 (Voicemail)</td>
<td></td>
</tr>
<tr>
<td>Ma, Burton</td>
<td>Dept of Computer Science &amp; Engineering, Faculty of Science &amp; Engineering</td>
<td>(416)736-2100 x 33252 (Voicemail)</td>
<td></td>
</tr>
</tbody>
</table>
Use this page to handcraft a HTTP Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.

**Parsed**

- **POST**
- URL: `http://www.site.cxx/`
- Protocol: `HTTP/1.1`

**Request Headers**

- `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8`
- `Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3`
- `Accept-Encoding: gzip, deflate, sdch`
- `Accept-Language: en-US,en;q=0.8`
- `Cache-Control: max-age=0`
- `Connection: keep-alive`
- `Cookie: ' OR '1' = '1`
- `Host: www.site.cxx`
- `If-Modified-Since: Mon, 04 Jul 2011 04:34:27 GMT`
- `User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/534.30 (KHTML, like Gecko)`

**Request Body**

- `color=' DROP TABLE Users; --`
(SQL) A1-Injection

› SQL is a standard programming language for relational databases

› Consists of a data definition language and a data manipulation language

› SELECT FirstName, LastName FROM Persons

WHERE Province = ' ON '

› SELECT specifies columns of the queried tables

› FROM indicates the table(s) from which data is to be retrieved

› WHERE eliminates all rows for which the comparison is not true.
database.executeQuery(
    "SELECT FirstName, LastName FROM Salesperson 
    WHERE State = '" + selectedState + '""
)

SELECT FirstName, LastName FROM Salesperson 
WHERE State = ''; DROP TABLE Users; --

SELECT * FROM Users WHERE username='foo' AND 
password='bar' OR '1' = '1'
An SQL query is concatenated with user-controllable data and submitted to a backend database.

```java
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") +"'";
```

Preventing injection requires keeping untrusted data separate from commands and queries.

All data could be stolen, modified, or deleted.
Attacking new users and stealing data beyond the database

Before Injection

After Injection
Blind SQL injection

SELECT OrderID FROM Sales WHERE CustomerID = '' OR MID((SELECT table_name FROM INFORMATION_SCHEMA.tables LIMIT 1),1,1) = 'A'
Prevention

› Avoid returning detailed error messages, stack traces:

› Validate input:
  - Casting (numeric or date)
  - Blacklists vs. Whitelists (regular expressions/ only simple patterns)
  - Escaping input:
    SELECT OrderID FROM Sales WHERE CustomerID = ''' OR ''1'' = ''1''
  - Parameterized queries:
    SELECT OrderID FROM Sales WHERE CustomerID = ?
Anonymous user functionality

- Search for products
- Search for salespeople

Authenticated user functionality

- Place an order
- Review past orders

Administrative user functionality

- Add new products
- Change product prices
A4-Insecure Direct Object References

Banking system

Bank Accounts

Bank Accounts

User

Anonymous User

E-store system

Carts

Carts

User

Anonymous User

Ref:123

Ref:012

#123

#012

#123

#012

Ref:123

Ref:012
A4-Insecure Direct Object References

› An authorized user changes a parameter value (which directly refers to an object) to another object the user isn’t authorized for.

› Automated crawler can find all directly accessible files in the system.
Prevention

› Authorization in database vs application

› Using per user or session indirect object references

› Take product records and store them in an array specific to that user. Credit card selection box:

```html
<select name="choosophone">
  <option value="1">myPhone3</option>
  <option value="2">myPhone4</option>
</select>
```

http://retailsite.cxx/catalog/productIndex=123
Join Product and UserProduct tables on the ProductName column and filter by UserID.

<table>
<thead>
<tr>
<th>User</th>
<th>ProductName</th>
<th>Price</th>
<th>ReleaseDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous</td>
<td>myPhone3</td>
<td>99.00</td>
<td>6/19/2009</td>
</tr>
<tr>
<td>Anonymous</td>
<td>myPhone4</td>
<td>199.00</td>
<td>7/24/2010</td>
</tr>
<tr>
<td>SteveJ</td>
<td>myPhone5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anonymous</td>
<td>myPhone3</td>
<td>99.00</td>
<td>6/19/2009</td>
</tr>
<tr>
<td>Anonymous</td>
<td>myPhone4</td>
<td>199.00</td>
<td>7/24/2010</td>
</tr>
</tbody>
</table>
The details of this user are not available to you
Do not assume that users will be unaware of special or hidden URLs or APIs.

Block access to all file types that your application should never serve (source files)

- .../auth/AddPassword probably there is:
  - .../auth/ResetPassword
  - .../auth/GetPassword
  - .../auth/UpdatePassword
A10-Unvalidated Redirects and Forwards

Don’t involve user parameters in destination.
Or ensure that the supplied value is valid.
Access unauthorized pages (where the user should be sent if a transaction is successful).
A3-Broken Authentication and Session Management

- Weak session identifiers
- The application returns the session token as part of the page URL
- Weak account management functions (account creation, change password, recover password). Example (Ebay account lockout DoS)

> Sessions do not timeout

http://tickets.com/itinerary;jsessionid=2PoOC2JDPXM0OQSNDLPSKHCJUN2JV?conf=ABB21

* This account has been locked indefinitely due to an excessive number of bad login attempts. Please contact the COIB at eFiling@coib.nyc.gov to have
the account unlocked

Login ID

Password

Login
A6-Security Misconfiguration

- Development or default settings remain once deployed.
- Missing patches.
- Stack traces and other overly informative error messages.

Hacker fears 'UFO cover-up'

In 2002, Gary McKinnon was arrested by the UK's national high-tech crime unit, after being accused of hacking into Nasa and the US military computer networks.

He says he spent two years looking for photographic evidence of alien spacecraft and advanced power technology.

America now wants to put him on trial, and if tried there he could face 60 years behind bars.

Banned from using the internet, Gary spoke to Click presenter Spencer Kelly to tell his side of the story, ahead of his extradition hearing on Wednesday, 10 May. You can read what he had to say here.
A7- Insecure Cryptographic Storage

› Personal information is not properly encrypted or hashed, or missing salt (example).

› Continued use of proven weak algorithms (MD5, SHA-1, RC3, RC4, etc...)

› Encryption keys are not stored securely (Hard coding keys, and storing keys in unprotected stores) or renewed properly.
<table>
<thead>
<tr>
<th>Password</th>
<th>Username</th>
</tr>
</thead>
<tbody>
<tr>
<td>5b9a6f41b5f05b16195eaf24f1fa43efdc3d317dd</td>
<td>Michael</td>
</tr>
<tr>
<td>2a72a1f522016fe4fd660fd19aa415ac5c3d33568</td>
<td>123456</td>
</tr>
<tr>
<td>4145abd8e29df6738096b117c771c538c3d319bb</td>
<td>Superman</td>
</tr>
<tr>
<td>5b9a6f41b5f05b16195eaf24f1fa43efdc3d317dd</td>
<td>Password</td>
</tr>
<tr>
<td>c6e173c0f381158c32f787e1d5c67530c3d32339</td>
<td>Qwerty</td>
</tr>
<tr>
<td>e69177b3636633b524162be07573abeec3d31fc0</td>
<td>Letmein</td>
</tr>
</tbody>
</table>

**LinkedIn Password Leak: Salt Their Hide**

Posted by **Soulskill** on Friday June 08 2012, @12:59PM from the i-see-what-you-did-there dept.

**CowboyRobot** writes

"Following [yesterday's post](#) about Poul-Henning Kamp no longer supporting md5crypt, the author has a new column at the ACM where he details all the ways that LinkedIn failed, specifically related to how they failed to 'salt' their passwords, making them that much easier to crack. On a system with many users, the chances that some of them have chosen the same password are pretty good. Humans are notoriously lousy at selecting good passwords. For the evil attacker, that means all users who have the same hashed password in the database have chosen the same password, so it is probably not a very good one, and the attacker can target that with a brute force attempt."
The site doesn’t use SSL for all pages that require authentication (stolen cookie, eavesdropping, man-in-the-middle)

Improperly configured SSL certificate generates warnings (users are confused)
A2-Cross Site Scripting (XSS)

A5-Cross Site Request Forgery (CSRF)

&

A2-Cross Site Scripting (XSS)
Thank you