2Secure

OWASP – Top 10

Patrik Karlsson patrik.karlsson@2secure.se
Martin Holst Swende martin.swende@2secure.se
2011-11-11

Patrik Karlsson

- Senior Security Expert at 2Secure
- 12+ years of IT-Security experience
- Speaker at OWASP, Defcon, SEC-T, T2 …
- Active Nmap developer
- Founder and maintainer of www.cqure.net
- @nevdull77 on twitter

Martin Holst Swende

- Senior Security Consultant at 2Secure
- Active member of Owasp Sweden (speaker, arranger)
- Speaker at Defcon 19, 2011 Las Vegas
- M Sc Computer Science and Engineering (D-linjen) LiTH 2004
- Security since 2008
- Programming since ~2000

2Secure – Business areas

Executive Security

Executive protection
Risk assessment
Security analyses
Training
Incident management

Corporate Security

Chief Security Officer
Security analysis
Security audit
Crisis management
Incident management
Training
Investigations

Screening Services

Background checks
- Recruiting
- Company Acquisitions
- Subcontractors & partners
- Legal Matters

Investigations

Information Security

Infosec Management
- Policies and procedures
- Risk assessment and audit
- Security requirements

IT Security
- Security penetration testing
- Vulnerability analysis
- IT Forensics

OWASP

- 501c3 not-for-profit worldwide charitable organization
- Focused on improving the security of application software
- Everyone is free to participate in OWASP
- All materials are available under a free and open source license
- Swedish chapters – Stockholm/Gothenburg
- Mattias Bergling – 2Secure co-leaders of Stockholm chapter

OWASP

- Tools and documents are organized into the following categories:
  - **PROTECT** - can be used to guard against security-related design and implementation flaws.
  - **DETECT** - can be used to find security-related design and implementation flaws.
  - **LIFE CYCLE** - can be used to add security-related activities into the SDLC.
- Documentation – DETECT
  - OWASP Application Security Verification Standard Project
  - OWASP Code Review Guide
  - OWASP Testing Guide
  - **OWASP Top Ten Project**

OWASP – TOP 10

- OWASP Top 10 Web Application Security Risks for 2010 are:
  - A1: Injection
  - A2: Cross-Site Scripting (XSS)
  - A3: Broken Authentication and Session Management
  - A4: Insecure Direct Object References
  - A5: Cross-Site Request Forgery (CSRF)
  - A6: Security Misconfiguration
  - A7: Insecure Cryptographic Storage
  - A8: Failure to Restrict URL Access
  - A9: Insufficient Transport Layer Protection
  - A10: Unvalidated Redirects and Forwards

A1: Injection

- A collection of several different injection vulnerabilities
  - SQL injection
  - LDAP Injection
  - XPATH Injection
  - OS Command Injection
- Involves manipulating user supplied data (url parameters, forms, cookies, http headers)
- If the server backend doesn't perform sufficient validation of user supplied data, chances are that arbitrary data may be injected
- SQL injection will be presented in more depth

A2: Cross-Site Scripting (XSS)

- A specific injection vulnerability which enables an attacker to break the browser security model by injecting content onto a vulnerable page.
- Cross site scripting will be presented in more depth later

A3: Broken Auth. and Session Management

- Scenario #1: Airline reservations application supports URL rewriting, putting session IDs in the URL:
  - **http://example.com/sale/;jsessionid=2P….C2JD?dest=Hawaii**
  - An authenticated user lets his friends know about the sale, by sending him an e-mail
  - His friend now has access to his session and credit card.

- Scenario #2: Application's timeouts aren't set properly.
  - A user uses a public computer to access site.
  - Instead of selecting "logout" the user simply closes the browser tab and walks away.
  - Attacker uses the same browser an hour later, and that browser is still authenticated.

A4: Insecure Direct Object References

- Scenario #1: An application allows users to edit their account profiles by going to the URL:
  - http://app/myprofile.aspx?id=1234
  - The URL includes the identity of the authenticated user, so that the proper profile can be loaded.
  - An attacker simply modifies the identity by increasing the ID and can access other users profiles.
- Scenario #2: An application allows users to upload and share documents.
  - The documents are stored on the servers file system
  - The following URL is used to access a document: http://app/download.aspx?id=1234&document=cv.doc
  - An attacker could replace the document with eg. ../../download.aspx and download the application source code.

A5: Cross-Site Request Forgery (CSRF)

- Scenario #1: The application allows a user to submit a state changing request that does not include anything secret:
  - http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
  - The attacker constructs a request that will transfer money from the victim's account to their account
  - The attacker embeds this request in an image request or iframe stored on various sites under the attacker's control.
    - ****
  - When the victim visits any of these sites while already authenticated to example.com, any forged requests will include the user's session info, inadvertently authorizing the request.

A6: Security misconfiguration

- Scenario #1: Your app relies on a framework like Struts or Spring
  - XSS flaws are found in these framework components you rely on.
  - An update is released but you don't update your libraries.
  - Until you do, attackers can easily find and exploit these flaws
- Scenario #2: The app server admin console is not removed
  - The default usernames and passwords aren't changed.
  - An attacker discovers the console and take over the server
- Scenario #3: Directory listing is not disabled on your server.
  - An attacker finds it possible to list directories to find any file.
  - The attacker finds and downloads compiled Java classes
  - By reverse engineering the classes she then finds a serious access control flaw in your application.

A7: Insecure cryptographic storage

- Scenario #1: An application encrypts credit cards in a database
  - However, the database is set to automatically decrypt queries against the credit card columns
  - An attacker finds a SQL injection vulnerability and retrieves all credit card information in plain-text
  - The system should have been configured to allow only back end applications to decrypt them, not the front end web application.
- Scenario #2:

"Det kändes som att det var dags nu. 210 000 användare, med lösenord i klartext: http://www.multiupload.com/Q5EQT6NDKF **#gratisbio #flashback"**

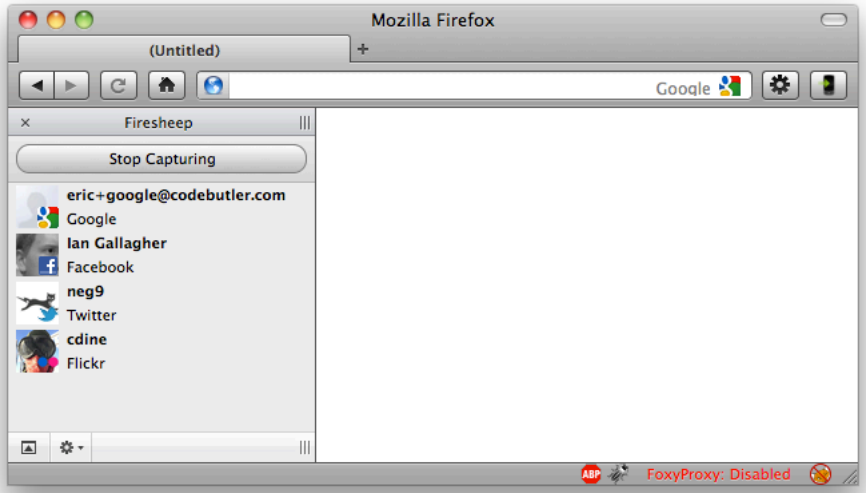
A8: Failure to restrict URL access

- Scenario #1: The attacker attempts to guess hidden application urls in order to get access to restricted pages
  - The attacker could attempt to guess the names of common urls or directories such as /admin.jsp /admin.aspx
  - If the security of the application relies on the urls being unknown the attacker may be able to get unauthorized access

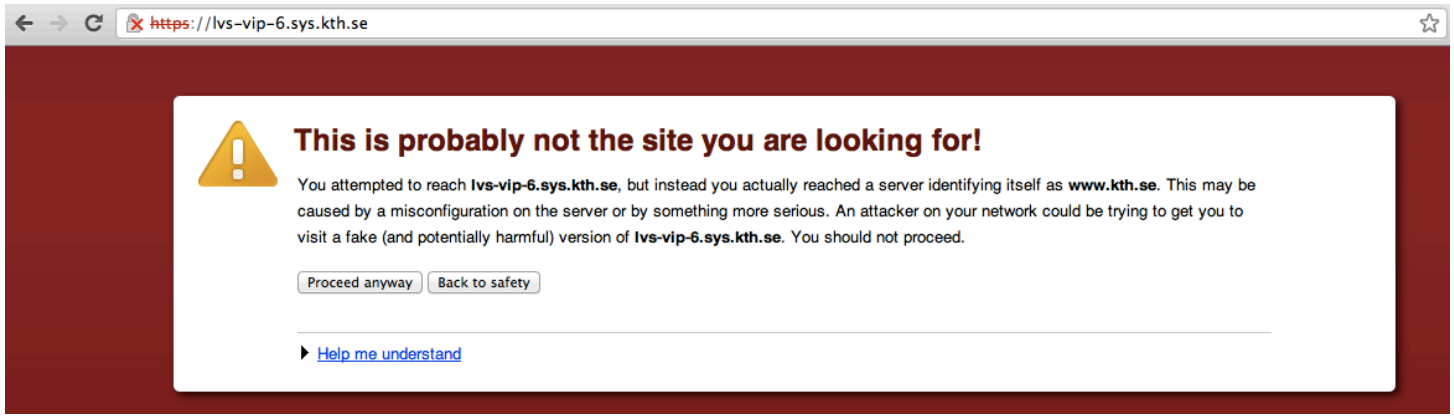
- Scenario #2: The attacker studies the client source code for inactive components or links
  - The attacker finds that the navigation components have several inactive links to pages that require higher privileges
  - By directly referencing these urls the attacker can gain access to these pages and their functionality

A9: Insufficient transport layer protection

- Scenario #1: A site simply doesn't use SSL for all pages that require authentication.
  - The attacker simply monitors network traffic and observes an authenticated victim's session cookie.
  - The Attacker then replays this cookie and takes over the user's session



- Scenario #2:



A10: Unvalidated redirects and forwards

- Scenario #1: The application has a page called “redirect.jsp” which takes a single parameter named “url”.
  - http://www.example.com/redirect.jsp?url=/login.jsp
  - The attacker crafts a malicious URL that redirects users to a malicious site that performs phishing and installs malware.
    - http://www.example.com/redirect.jsp?url=http://www.malware.com

- Scenario #2:The application uses forward to route requests between different parts of the site.
  - To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful.
  - In this case, the attacker crafts a URL that will pass the application’s access control check and pass her to admin.jsp
  - https://www.example.com/valid.jsp?url=admin.jsp

SQL injection

A1: Injection – Overview

- SQL injection is one of many injection based vulnerabilities
- Currently the most common one, often resulting in considerable technical consequences
- Occurs due to improper input validation and use of insecure methods of building dynamic SQL statements
- Consider the following SQL statement being part of application authentication

```
String sql = "SELECT * FROM app_logins WHERE username='"
+ user + "' AND password='" + pass + "'"
```

- What happens when the user ***o'malley*** logs on to the system?
- What happens when the user **' or 1=1 --** logs on to the system?

A1: Injection – Overview

- How is information typically extracted through SQL injection?
- Error messages
  - By forcing error conditions; OR 1=@@version
  - “Conversion failed when converting the nvarchar value 'Microsoft SQL Server 2005”
- Blind SQL injection
  - By using statements that would evaluate to true or false the attacker could study the behavior of the application
- Out-of-band channeling
  - Timing – by using time based delays and monitoring response
  - DNS – using the DB to do DNS queries with the extracted data
- Files
  - Writing results to files that can be downloaded from the server

A1: Injection – Consequences

- What could the technical consequences of a SQL injection vulnerability be?
  - Unauthorized access to information, both read and write

2011-10-26 10:40 Computer Sweden

Hackarhärvan fortsätter - 211 000 nya konton ute

Av Linus Larsson | CS ComputerSweden

NYHETER Hackarvågen fortsätter. Ytterligare 200 000 konton lösenord har läckt ut på nätet. De uppges komma från sajten Gratisbio.se.

- A common security problem in web applications is the single proxy account from the application to the database
  - The application always has at least as much access as the most powerful application user

A1: Injection – Consequences

- What could the technical consequences of a SQL injection vulnerability be?
  - Depends on the make, patch level, configuration and privileges of the application account
  - Could allow access to operating system commands and thereby total compromise of the system
    - Microsoft SQL Server – xp_cmdshell
    - Oracle – Java native code or ExtProc
  - Could allow access to the file system eg:
    - Microsoft SQL Server – xp_dirtree
    - MySQL – LOAD_FILE
    - Oracle – UTL_FILE
  - Could allow network access and connections to other DB's
    - Microsoft SQL Server – linked servers, OPENROWSET …
    - Oracle – database links

A1: Injection – recommendations

- How do you avoid SQL injection in your web applications?
- OWASP: SQL Injection Prevention Cheat Sheet
  - Option #1: Use of Prepared Statements (Parameterized Queries)
    - Separates SQL code from user supplied parameters through placeholders
  - Option #2: Use of Stored Procedures
    - This is not entirely true as calling them improperly may still open up the application to SQL injection
  - Option #3: Escaping all User Supplied Input
    - Difficult to achieve adequate security by simply escaping
  - Additional Defenses:
    - Also Enforce: Least Privilege
    - Also Perform: White List Input Validation

A1: Injection – recommendations

- The following code sample illustrates the use of parameterized SQL through place holders and bind variables

```
String query =
        "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
       OleDbCommand command = new OleDbCommand(query, connection);
       command.Parameters.Add(new OleDbParameter("customerName", CustomerName Name.Text));
       OleDbDataReader reader = command.ExecuteReader();
       // …
} catch (OleDbException se) {
       // error handling
}
```

XSS

A2: XSS – Overview

- Browsers abide by the Same-origin-principle (S.O.P):
  “If your page is served from yourdomain.com, that page may not read any data served from mydomain.com”
  - Loading images from other domains is ok.
  - Loading scripts from other domains is also ok.
  - Posting forms to other domains is also ok.
- What does this mean?
  - If you visit evil.com, while that page **can** load gmail in an iframe, it cannot harvest your contacts or spam all your friends
  - It cannot read you bank statements or click through your account and issue payments
- XSS kills the S.O.P

A2: XSS – Overview

- XSS is when a victim domain (victim.com) is manipulated into executing scripts (typically with malicious intent)
- Three types of XSS vulnerabilities:
  - Reflected
  - Stored
  - DOM-based
- Reflected XSS, User-generated data is reflected back into the viewed page

A2: XSS – Overview

- Reflected XSS continued
  - Typically a 1:1 distribution vector
  - A victim must somehow be made to visit a specially crafted URL
- Stored XSS
  - The payload is stored on the site.
  - Canonical example : comments, forums
  - 1:N distribution - Anyone who visits the infected page becomes a victim
- DOM-based XSS
  - The payload is never sent to the server, thus very difficult to detect
  - Occurs if the client-side javascript reads parameters from e.g. document.location

A2: XSS – Overview

- DOM-based XSS, continued
  - Twitter example, convert http://twitter.com/#!mhswende into http://twitter.com/mhswende

```
(function(g)
{
        var a=location.href.split("#!")[1];
        if(a)
        {
                g.location=g.HBR=a;
        }
})(window);
```

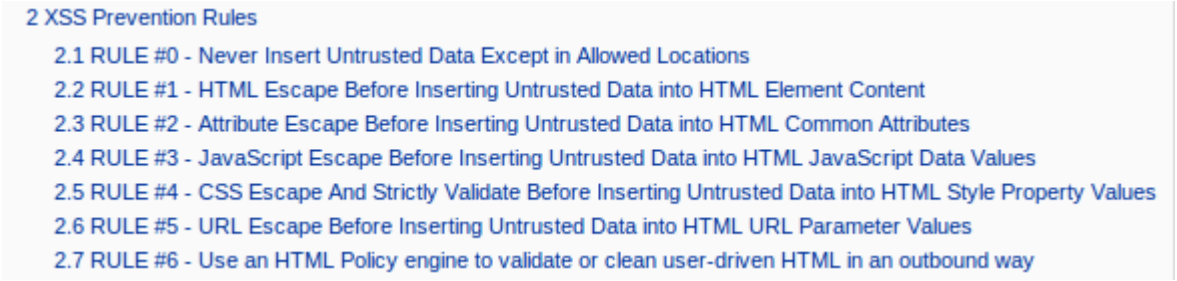
  - Unfortunatly, also converts http://twitter.com/#!javascript:alert(1) into executable javascript

A2: XSS – Consequences

- By executing javascript, an attacker can
  - Execute actions in the application, as the authenticated victim (e.g. lose money in a poker application)
  - Read data in the application, and send to attacker (e.g. user data in an internal administration interface)
  - Modify the page (e.g. prompt for password or credit card information)
- Javascript worm based on stored XSS took down Myspace 2005

A2: XSS – Recommendations

- How can you avoid XSS issues?
  - Always do proper output encoding, **for the context into which data is being placed**.
    - Use the guidelines: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet



  - Input validation. Whenever it is possible to validate input: validate using white-list approach.
    - Only allow numeric characters for numbers, only allow [a-zA-Z] for usernames.
      - However, many types of data has no simple definition of valid.

A2: XSS – Recommendations

- Mitigating XSS:
  - Set the HttpOnly attribute on cookies. This disallows javascript from reading the value of cookies.
    - However, XSS can rewrite the page to obtain the full credentials anyway.
  - Use Content-Security-Policy
    - A new HTTP header which specifies where page resources are allowed to be loaded from
  - For end-users : use NoScript and up-to-date browsers. Many browsers now have native XSS-protection (while still not perfect, they become better all the time)

Contact

Patrik Karlsson
patrik.karlsson@2secure.se
0768-31 70 82

Martin Holst Swende
martin.swende@2secure.se
0768-31 81 38