Web Application Vulnerabilities

Dusan Palider

Warning

Do not try any of the techniques discussed in this presentation on a system you do not own.

It is illegal and you will get caught.

Agenda

- Introduction
- OWASP & OWASP Top 10
- More on Injection & XSS
- Q&A

Why do I care?

Why do I care?

- do you use the internet?
- visit forums?
- use multiple tabs in your browser?
- use online banking?

A quick example

- you are logged into your bank
- in a separate tab you visit a forum (or a webpage)
- in the forum/page someone placed a JS code that executes and transfers money out of your account

How?

- a very unsafe forum site or an evil page
  - allows you to upload the script
- slightly unsafe bank site
  - allows the script to execute.. but there may not be much the bank can do here

- code specific to a bank
  - which bank?
    - HSBC
    - Key Bank
    - Chase
    - Bank Of America

So what do I do?

OWASP

- www.owasp.org
- The Open Web Application Security Project
- "a … not-for-profit worldwide charitable organization focused on improving the security of application software"

OWASP Top 10

- top 10 vulnerabilities
- updated every year
- for 2010:
  - 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

- What:
  - attacker gets the application to carry out a command
- How:
  - we allow unsafe input to get into an interpreter & execute it as a command
- What to do:
  - canonicalize and validate user input
  - encode application output
  - use parameterized queries
  - don’t call OS directly
  - use ESAPI library
  - use APIs that wrap OS

A2 – XSS (Cross-Site Scripting)

- What:
  - attacker executes a script against a user
- How:
  - we allow unsafe input containing a script to be carried out against an unsuspecting user visiting a website
- What to do:
  - canonicalize and validate user input
  - encode application output
  - use Microsoft’s AntiXSS library
  - use ESAPI library

A3 – Broken Authentication and Session Management

- What:
  - attacker manages to impersonate another user
- How:
  - we do not manage the session properly or use an unsafe authentication
- What to do:
  - clear out the session @ start and @ end
  - do not store session ID in URL
  - store and transport user credentials safely (SSL)
  - ask user to re-authenticate before carrying out a sensitive operation
  - expire session after a timeout

A4 – Insecure Direct Object Reference

- What:
  - the application exposes a direct object reference to the attacker, which allows the attacker to attack the application
- How:
  - use identifiers (such as primary keys) in dropdowns/URLs/tables..
- What to do:
  - create a mapping, so that way you don’t expose objects
  - verify input against a white list
  - validate user permissions to the action that was requested

A5 – CSRF (Cross-Site Request Forgery)

- What:
  - attacker forces the browser to send a request to a target website
- How:
  - script is executed on the malicious site, hoping to attack the target one
- What to do:
  - re-authenticate before allowing a sensitive operation
  - use a CSRF cookie to identify that the request is coming from your page & verify it before processing
  - use ESAPI
  - use SessionID in Page.ViewStateUserKey and verify it

A6 – Security Misconfiguration

- What:
  - default settings on an IIS/webserver/…
  - leaking too much error information
- How:
  - The infrastructure was not configured properly.
  - We did not create user-friendly error messages.
- What to do:
  - use custom errors in web.config
  - do not allow debugging in web.config
  - make sure all default accounts are disabled/protected

A7 – Insecure Cryptographic Storage

- What:
  - encrypted data gets hacked (or data was never encrypted)
- How:
  - we don’t use (or incorrectly use) encryption
- What to do:
  - do NOT write your own algorithm
  - use hash of SHA-256 or better to hash passwords
  - use AES, RSA to encrypt persisted data
  - use ESAPI

A8 – Failure To Restrict URL Access

- What:
  - a user “guesses” a link in our application
- How:
  - we do not check permissions for users landing on a page, but rely on the page being “invisible” in menus
- What to do:
  - block access to files never used in IIS
  - use a permission matrix
  - always validate role on PageLoad
  - do NOT hide, but DISABLE links/buttons to screens that the user should not see

A9 – Insufficient Transport Layer Protection

- What:
  - credentials or other data gets hacked while in transport
- How:
  - did not use SSL or encryption (or weak encryption) to protect the data
- What to do:
  - use SSL when sending sensitive data/passwords
  - use a secure SQL server connection (Encrypt=Yes)
  - use correct encryption
  - use ESAPI

A10 – Unvalidated Redirects and Forwards

- What:
  - site is tricked into redirecting a user to an unsafe site
- How:
  - we did not validate our forward, and an attacker tricked the site into forwarding somewhere else
- What to do:
  - do not forward
  - if you have to forward, don’t combine the link w/ user input
  - check that the domain of the site matches

More on Injection & XSS

Injection

- allows an attacker supplied text be passed into an interpreter, where the interpreter runs it as a command, instead of treating it as a parameter

Injection types

- SQL injection
- OS injection
- SOAP injection
- Xpath injection
- LDAP injection
- SMTP injection
- XML injection
- JS injection
- …

SQL Injection example

"select * from users where userID = '" + userFromSite + "'"

- if userFromSite is aaa' OR '1' = '1
- if userFromSite is aaa'; DROP TABLE users; --

OS Injection

“move file1.txt “ + fileNameFromUser

- if fileNameFromUser is file2.txt & delete c:\*.* \quiet

SQL Injection Code

Cross Site Scripting XSS

- similar to injection – JS is executed against a user (web browser acting as an interpreter)

XSS Example

- <SCRIPT>alert(‘Hello’);</SCRIPT>
- <SCRIPT>alert(document.cookie);</SCRIPT>

XSS Code

Where to go from here?

Safe Practices

- never trust user input
- canonicalize user input (convert to a known encoding)
- validate user input on the server (white list)
    - if fails, reject, do not filter
- encode data being shown to the user
- always use parameterized queries
- use prepared statements as much as you can
    - do not generate them dynamically
- use existing APIs instead of direct calls to OS/other interpreters
- restrict access as much as possible
- never trust the user

Resources

- Owasp & ESAPI – www.owasp.org

Questions?