The OWASP Foundation
http://www.owasp.org

A Web Developer's Illustrated Primer to Application Security

Justin Ferguson
UMB Bank
OWASP Kansas City

Template Copyright © The OWASP Foundation // Presentation Copyright © Justin C. Ferguson
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

PLATINUM SPONSORS

stackify

PAIGE TECHNOLOGIES Our relationships, Your success

Alfresco doGreatWork()

Balance Innovations

Ingenuity Consulting Partners, Inc. "Innovation for Your Success"

Sprint

GOLD SPONSORS

KEYHOLE SOFTWARE

UMB

ADVANTAGE TECH

CENTRIQ TRAINING

telerik deliver more than expected

AdventureTech

VERSIONONE

ADAPTIVE SOLUTIONS GROUP

kansas city PMI KC MID AMERICA CHAPTER

DST SYSTEMS

CARFAX

perceptivesoftware from Lexmark

SILVER SPONSORS

discount ASP.net

Microsoft

ComponentOne

Justin Ferguson

Information Security Engineer

- AppDev, InfoSec, Systems Admin
- Not a PowerPoint wizard.

Work: http://www.umb.com/

E-Mail: justin.ferguson@umb.com

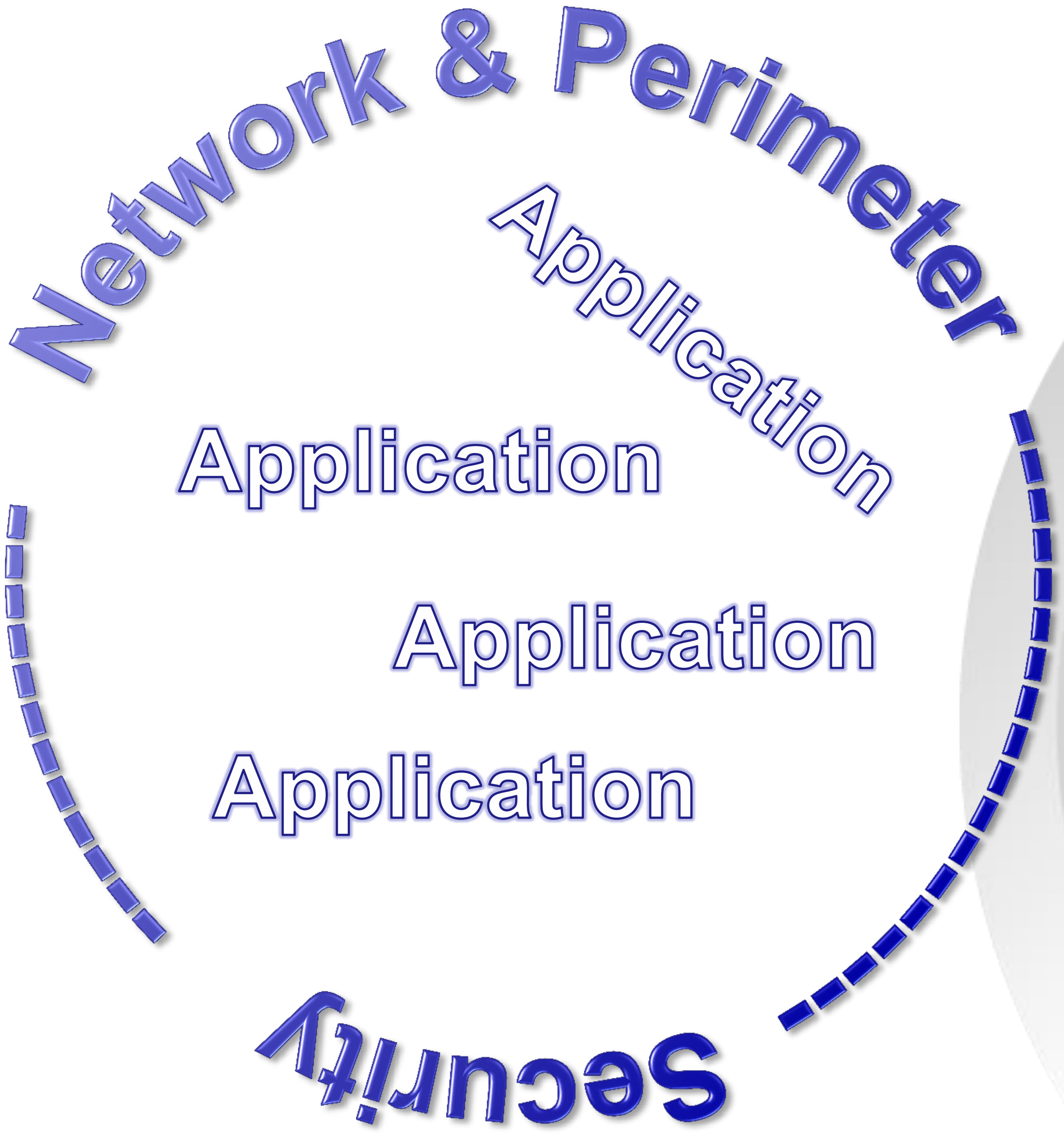
Personal: http://jferg.thedotin.net/

Twitter: @jferg

E-Mail: jferg@thedotin.net

What is AppSec?

What is AppSec?



AppSec vs. NetSec

NetSec – Put a hard, crunch shell around the soft, juicy middle.

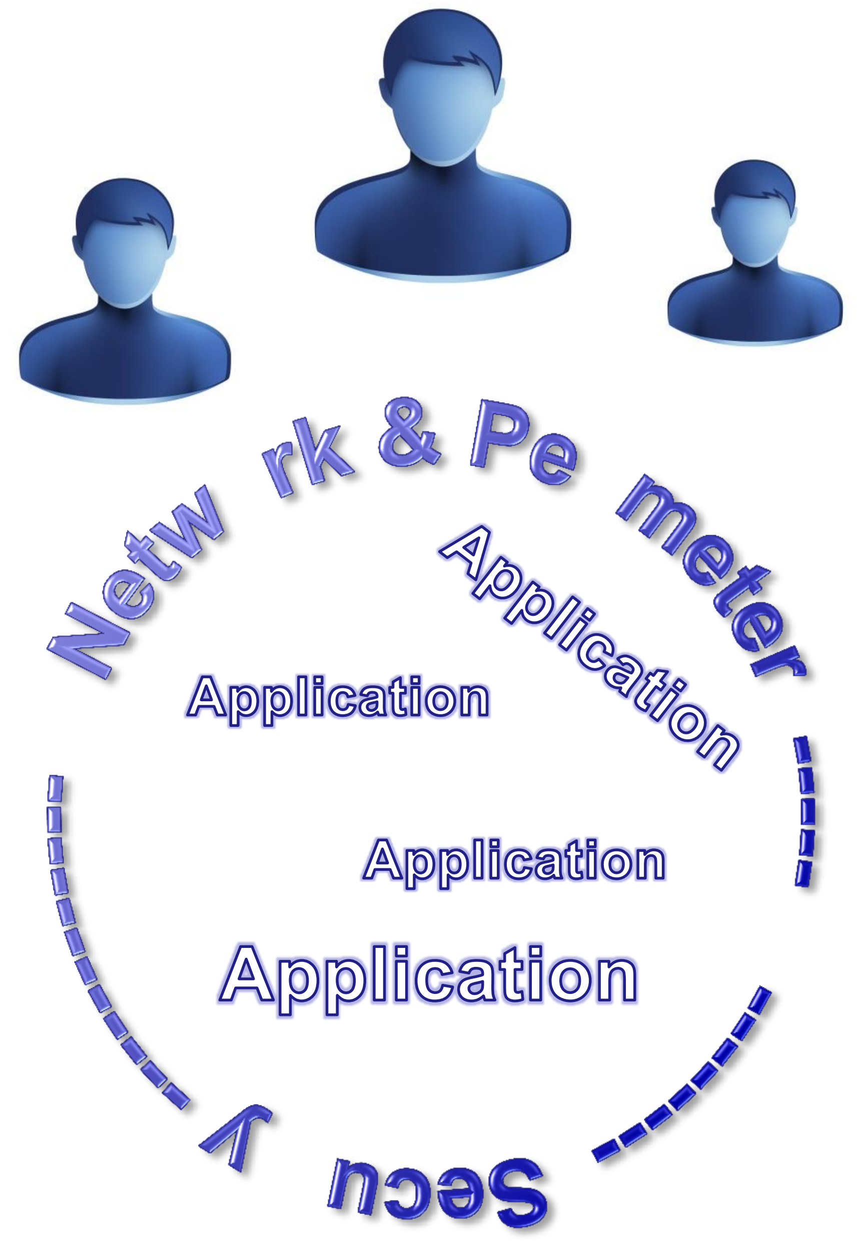
Easy, right?

(Not really that simple.)

What is AppSec?

Why not? This Guy.

What is AppSec?



AppSec is making the soft, juicy middle … not so soft and juicy.

- Authentication
- Access Control
- Secure Development

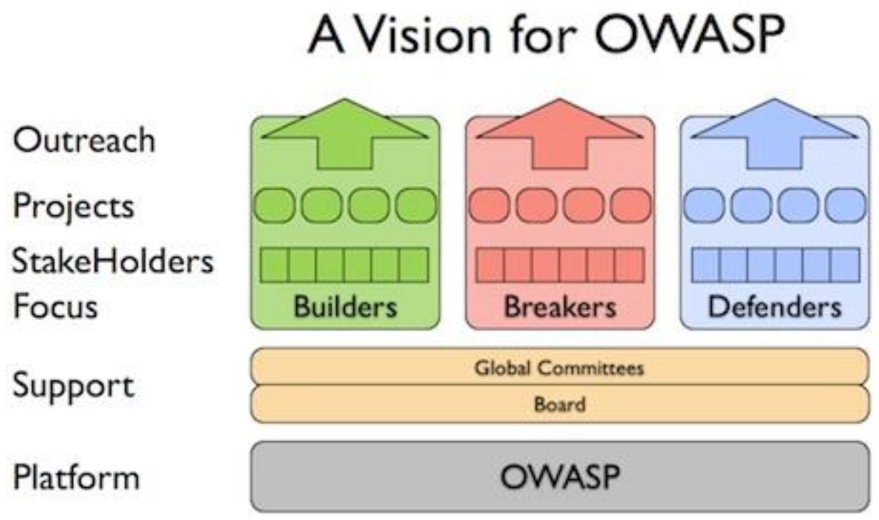
Builders, Breakers, and Defenders

AppSec is generally divided into 3 groups:

- Builders – Developers, Designers, Architects
- Breakers – Testers and Hackers
- Defenders – Securing the Builders from the Breakers

What is OWASP?

Open Web Application Security Project



OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted.

Local Chapter: https://www.owasp.org/index.php/Kansas_City
Twitter: @OWASPKC

Meetings about every two months.

What is OWASP?

Tools for Builders, Breakers, and Defenders:

- Software Assurance Maturity Model (SAMM), Secure Coding Practices, Application Security Verification Standards (ASVS)
  https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model
  https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide
  https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

- WebGoat Project
  https://www.owasp.org/index.php/Webgoat

- Zed Attack Proxy (ZAP)
  https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

- Enterprise Security API (ESAPI), CSRFGuard, AntiSamy
  https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
  https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project
  https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project

What is OWASP?

And one other thing …

The OWASP Top 10

The OWASP Top 10

A ranked list of the Top Ten threats to web application security.

A1. Injection
A2. Broken Authentication & Session Management
A3. Cross-Site Scripting (XSS)
A4. Insecure Direct Object References
A5. Security Misconfiguration
A6. Sensitive Data Exposure
A7. Missing Function Level Access Control
A8. Cross-Site Request Forgery
A9. Using Known-Vulnerable Components
A10. Unvalidated Redirects and Forwards

- OWASP Top 10 for 2013 (Release Candidate)
  https://www.owasp.org/index.php/OWASP_Top_Ten

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

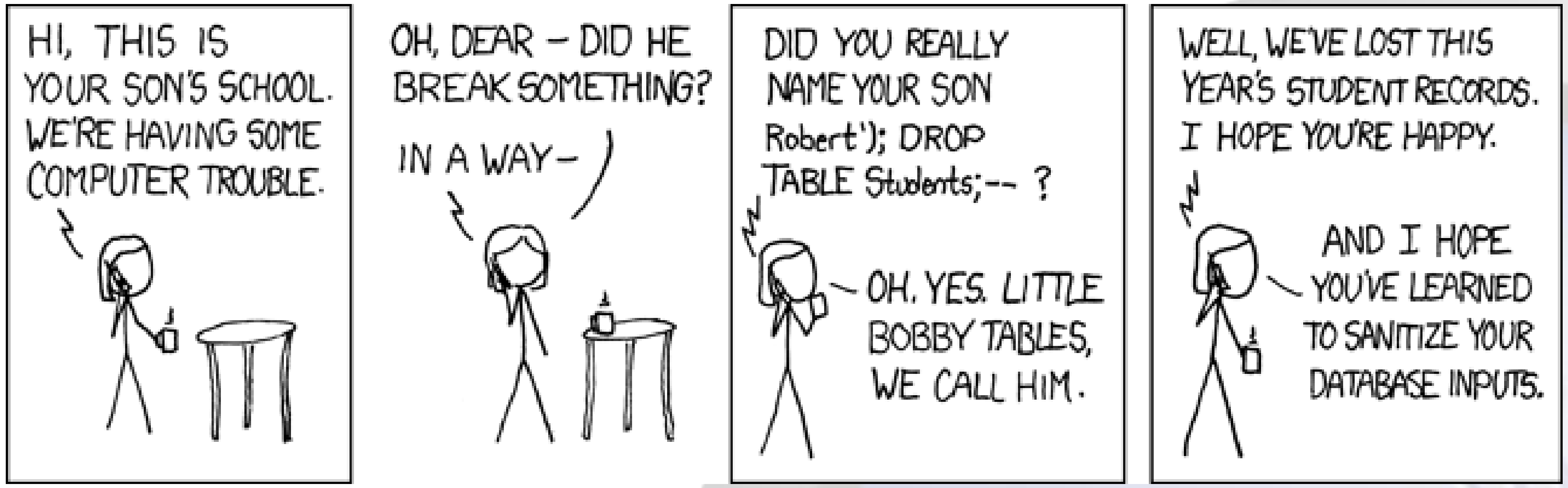
A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
xkcd.com

Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

SQL Injection

```
userName = <user input>;
password = <user input>;

// …

String SQLQuery = “SELECT * FROM USERS WHERE
USERNAME = ‘“ + userName + “’ AND PASSWORD =
‘“ + password + “’;”;

results = SQLConn.execute( SQLQuery );

if ( results.count == 0 ) {
   return errorPage;
}
```

Trusted vs. Untrusted Data

Trusted Data – Data that comes from a known source, and is known to be “clean”. i.e. data from a backend system, or data from a database (if it was scrubbed before insertion).

Untrusted Data – Data that comes from an outside source, i.e. users (or hackers).

Trust Boundary – A place within an application where data moves from one trust status to another.

WebAppSec Rule #1: If data has been to the browser and back – DON’T TRUST IT!

SQL Injection

Injection happens when untrusted data crosses a trust boundary and is treated as trusted data.

```
userName = <user input>;
password = <user input>;

…

String SQLQuery = “SELECT * FROM USERS WHERE USERNAME = ‘“
+ userName + “‘ AND PASSWORD = ‘“ + password + “‘;”;

results = SQLConn.execute( SQLQuery );

if ( results.count == 0 ) {
    return errorPage;
}
```

SQL Injection

```
userName = “justin”;
password = “secure”;
```

```
SELECT * FROM USERS WHERE USERNAME =
‘justin’ AND PASSWORD = ‘secure’;
```

One record returned. Great!

SQL Injection

```
userName = “justin”;
password = “’ OR 1=1; --”;
```

```
SELECT * FROM USERS WHERE USERNAME =
‘justin’ AND PASSWORD = ‘’ OR 1=1; -
-’;
```

All records returned. Uh-oh!

Blind Injection

Blind SQL Injection occurs when the results of the query are not visible to the end user … but just because you can’t see it doesn’t mean it’s not happening!

Blind SQL Injection - Demo

WordPress 2.1.3 contained an unauthenticated blind SQL injection in admin.php.

All your WordPress are belong to us!

Other Injection

Injection can occur anytime that untrusted data is used in a trusted context.

- LDAP lookups
- Building commandlines

```
System.exec(“processor “ + userName + “ /var/log/datafile”);
```

What if userName = “; rm –rf /*; cat”?

- JavaScript injection (beware of eval()!)

Code Injection

Injection doesn’t have to just be in data access – it can happen in code too!

eval() and its brethren are dangerous in any language.

Avoiding Injection

SQL Injection

- Parameterized Queries!
- Structured Data Access Layers
- Stored Procedures … *but*

```
SQLConn.execute(“myStoredProc(‘“+userInput+“’);”);
```

doesn’t fix anything!

- Input filtering and sanitization

Instead of:

```
sql = String.concat( “select * from my_table where username = `“,
         username, “’ and password = ‘”, password, “’” );
SQLConn.execute( sql );
```

Use:

```
sql = “select * from my_table where username = ? and password = ?”
SQLconn.execute(sql, username, password)
```

Avoiding Injection

All Injection

- Parameterization!

```
System.execute( “process”, “-u”, userName, “/var/run/output” );
```

- Input filtering and sanitization – sometimes the only option … but be careful! (More on that topic later.)

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Broken Auth & Session Mgmt

Authentication and session management is frequently implemented incorrectly.

As a result, an attacker can:

- Compromise credentials
- Steal sessions
- Assume victims’ identities within the application

Broken Session Mgmt

Example 1:

```
userName = getCookie( “userName” );

if ( userName == null ) {
      userName = doLogin();
      setCookie( “userName”, userName );
} else {
      display(“Logged in as “ + username + “!” );
      getPrivileges( userName );
}
```

Broken Session Mgmt

Example 2:

```
class Session {
      String new() {
           this.id = Time.current();  // Current time in ms
           return this.id;
      }
}

---------

if ( user.session == null ) {
      user.session = new Session();
      setCookie( “SessionID”, “user.session” );
}
```

Avoiding Broken Session Mgmt

- Simple: Use your framework's session management and authentication capabilities!
- But, if you must implement your own:
  - Randomly generate session IDs. (More on randomness later.)
  - Pass *only* the session ID to the browser.
  - Change the session ID regularly.
  - Store the source IP in the session, and check it with every request.
  - Always give sessions a time limit.

Broken Authentication

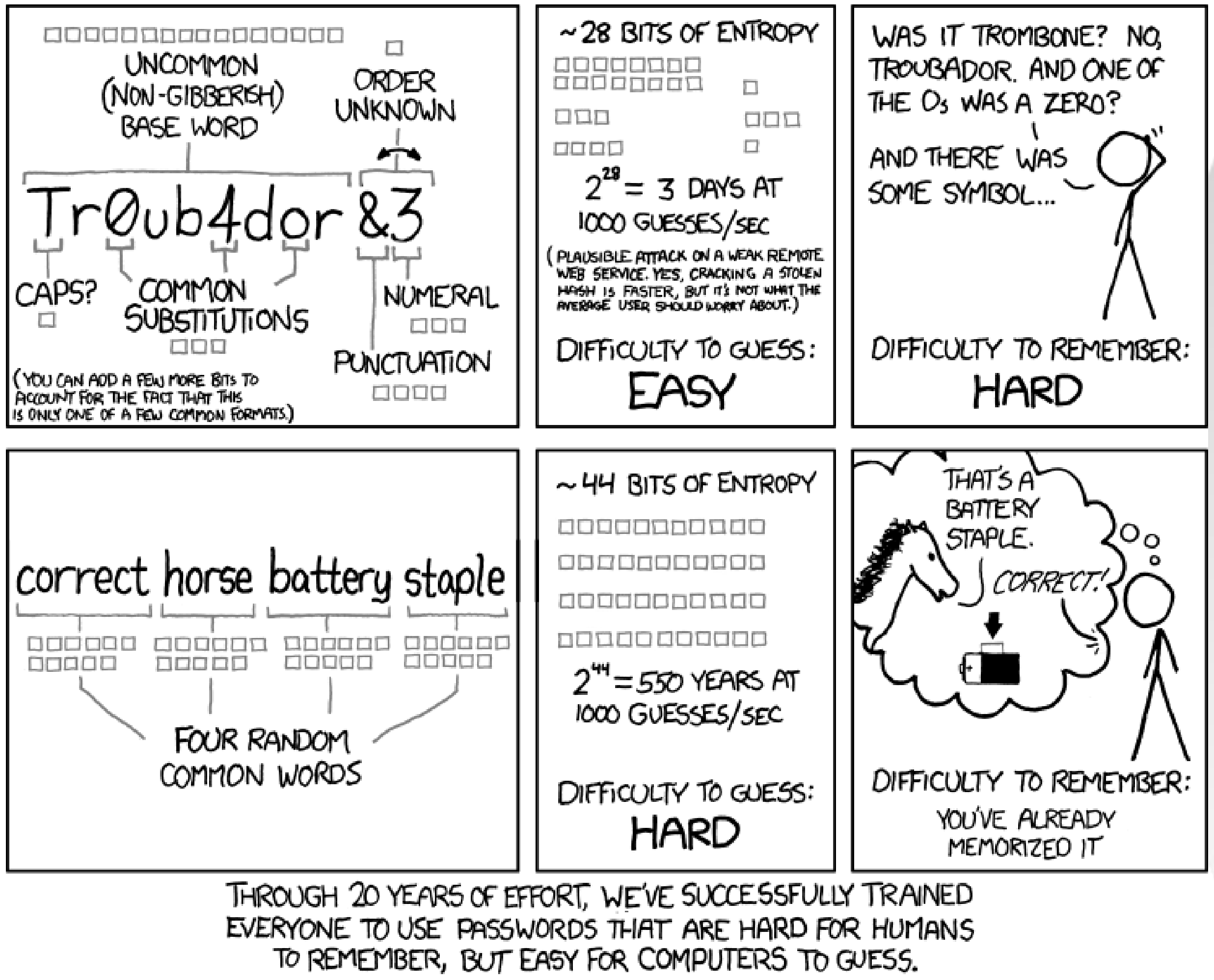
Can encompass several different problems:

- Improper Password Storage
- Weak Passwords (Length + Complexity)
- Credentials Transmitted in Clear-Text
- Too Much Information in Error Messages
- Lack of Controls – No Lockout after Failures

Avoiding Broken Auth

Proper Password Storage

- Salted one-way hash with a large keyspace.


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

xkcd.com

Enforce Strong Passwords

- Longer passwords are harder to guess.
- Password quality gauge
- Tell users what's wrong.

Avoiding Broken Auth

Never transmit usernames/passwords in plain-text.

- Even login pages should be SSL.

Don't give errors details in login-failure messages.

- Just say "login failed" – no more.

Implement proper authentication controls

- Lockouts.
- Notify user of password changes.

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

What is XSS?

Cross-Site Scripting occurs when a web application takes untrusted data and displays it to the user as if it was trusted data.

I can make my content show up in your application.

What’s the risk?

So … I can make my content show up in somebody else’s website. What’s the big deal?

- Social Engineering
- Phishing
- Cookie Stealing

Types of XSS

Three types of XSS:

- Reflected
- Persisted (or Stored)
- DOM-based

Reflected XSS

- Most Common
- Least Dangerous
- Basically Stateless
- Still Dangerous

What if your grandpa got an e-mail saying:

Your bank account records need updating. Click this link to update them!

```
https://www.umb.com/mywebapp?address=%22%3e%3c%73%63%72%69%70
%74%3e%61%6c%65%72%74%28%22%54%68%61%6e%6b%20%79%6f%75%21%22%
29%3c%2f%73%63%72%69%70%74%3e
```

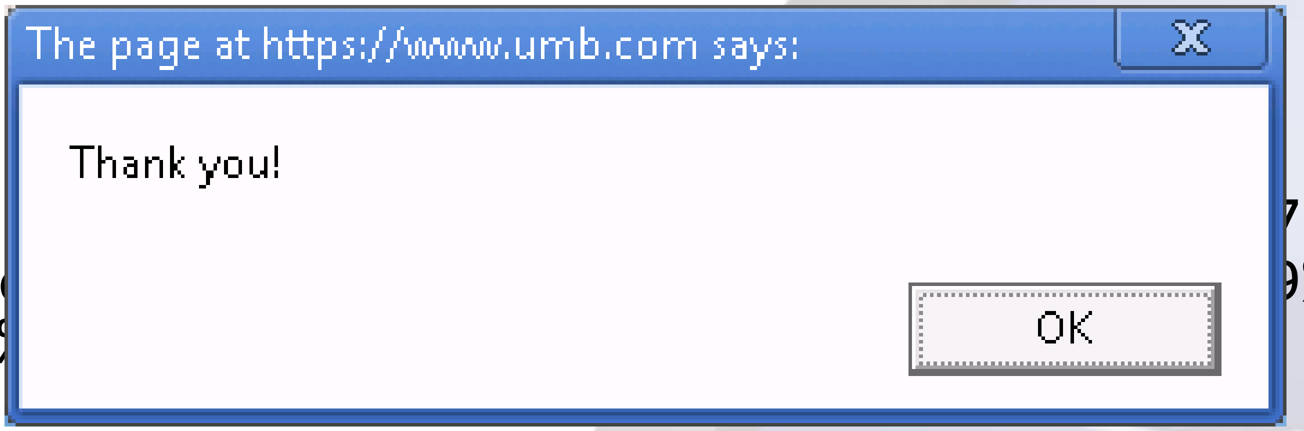
Reflected XSS

The passed in value for “address” gets inserted into:

```
<input type=“hidden” name=“addr” value=“[address]”>
```

So that U

```
https://www                                            73%63%72%69%70
%74%3e%61%6                                            9%6f%75%21%22%
29%3c%2f%73
```


The page at https://www.umb.com says:

Thank you!

Decodes to:

```
https://www.umb.com/mywebapp?address=“><script>alert(“Thank
you!”)</script>
```

Results in:

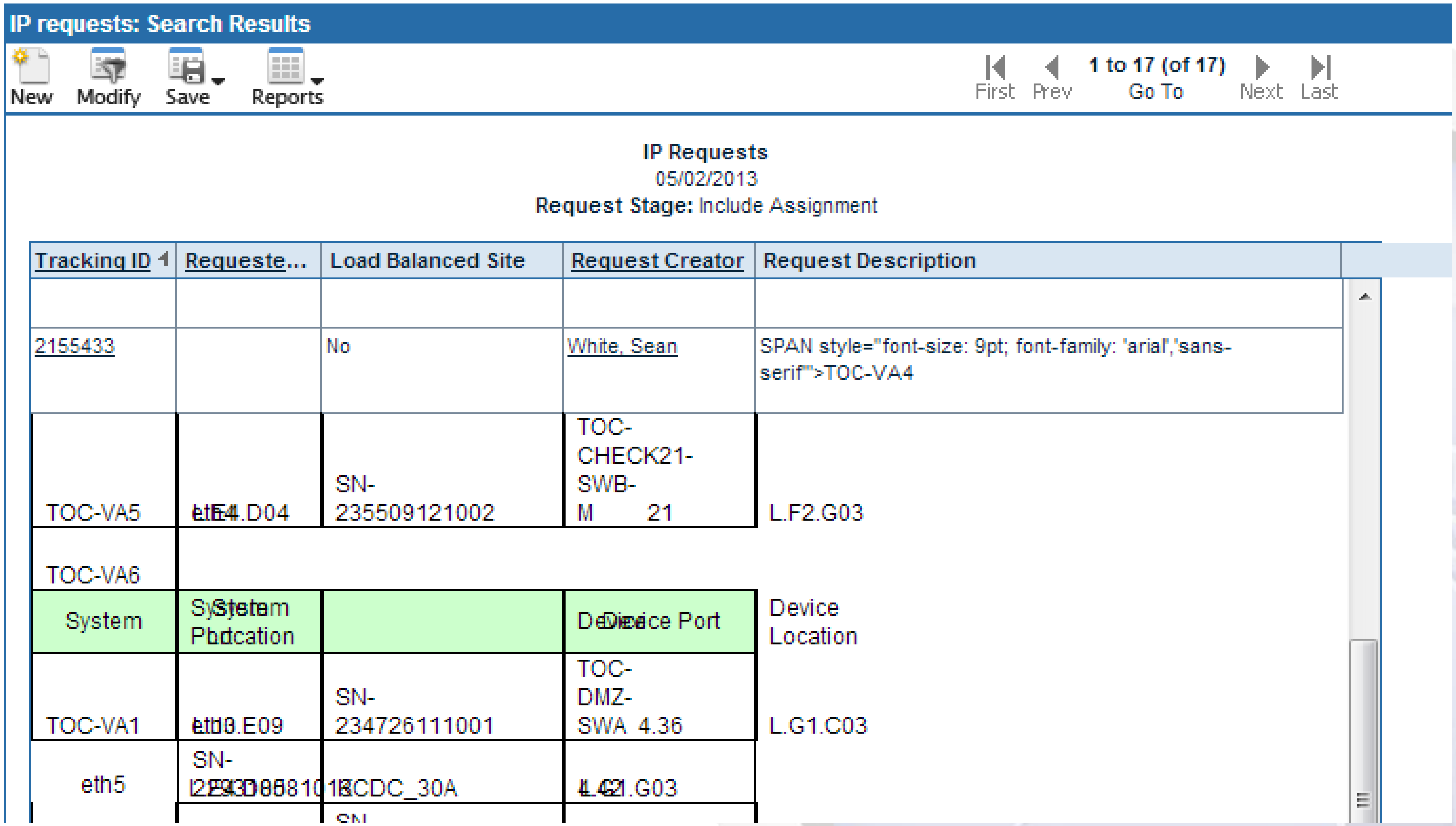
```
<input type=“hidden” name=“addr” value=“”>
<script>alert(“Thank you!”)</script>”>
```

Persisted XSS

Just like Reflected XSS, but stored permanently.

- Less Common
- Much More Dangerous
- Leaves Permanent Content on the Target Site
- Can be an issue any time user-generated content is displayed.

Persisted XSS



IP requests: Search Results

IP Requests
05/02/2013
Request Stage: Include Assignment

Tracking ID	Requeste...	Load Balanced Site	Request Creator	Request Description
2155433		No	White, Sean	SPAN style="font-size: 9pt; font-family: 'arial','sans-serif'">TOC-VA4

TOC-VA5	ethE4.D04	SN-235509121002	TOC-CHECK21-SWB-M 21	L.F2.G03
TOC-VA6				
System	SyStetam Pbdcation		DeDiceice Port	Device Location
TOC-VA1	eth3.E09	SN-234726111001	TOC-DMZ-SWA 4.36	L.G1.C03
eth5	SN-[illegible]	[illegible]CDC_30A	[illegible]1.G03	
		SN-		

DOM-Based XSS

Similar to the previous two, but more complex:

- Occurs when JavaScript in the page expects a simple input, but is passed a DOM object instead.
- Usually Reflected, but more JavaScript-y. (Basically just a different vector for entry.)

Avoiding XSS

- Input filtering and sanitization – once again, this is hard to do right.
  - HTML Entity-encoding not always sufficient.
  - OWASP AntiSamy & ESAPI
  - MS Anti-CSS Library
- Proper escaping of user-provided content.
- Whitelisting HTML tags.

Avoiding XSS

- Rule 0: Never insert untrusted data except in allowed locations. (See rules 1-5.)
- Rule 1: HTML Escape before inserting untrusted data into HTML element content.
- Rule 2: Attribute Escape before inserting untrusted data into HTML attributes.
- Rule 3: JavaScript Escape before inserting untrusted data into JavaScript Data Values
  - Rule 3.1: HTML Escape JSON values in an HTML context and read the data with JSON.parse.
- Rule 4: CSS Escape and strictly validate before inserting untrusted data into HTML Style Properties
- Rule 5: URL Escape before inserting untrusted data into HTML URL Parameter Values.

- OWASP XSS Prevention Cheat Sheet

https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

What are IDORs?

IDORs

```
<h2>Choose a statement: </h2>
<ul>
<li><a href="getstatement.aspx?id=4095269">Jan 2013</a></li>
<li><a href="getstatement.aspx?id=5029938">Feb 2013</a></li>
<li><a href="getstatement.aspx?id=5982040">Mar 2013</a></li>
<li><a href="getstatement.aspx?id=6502921">Apr 2013</a></li>
</ul>
```

getstatement.aspx does nothing beyond return the statement corresponding to the id which was passed to it.

So…what if you pass id=5982041 instead?

Path Traversals

- Not as common these days, but still an issue.
- Common problem in forum apps, etc

```
<form method="post" action="/uploadAttachment">
Enter destination file: <input type="text" name="dest">
Choose file to upload: <input type="file" name="upload">
</form>

[uploadAttachment]
$fh = open( "rw", $_POST['dest'] );
write $fh, $_POST['upload'];
close $fh;
```

But what if somebody specifies
"../../../../../../../../../etc/passwd" as "dest"?

Avoiding IDORs

- Always check permissions before returning an object to a user!
- Don’t use direct file references – retrieve non-public objects via code.
- Enforce object-level permissions at the server level.
- Properly escape or remove slashes from filenames that are specified by the user.
- And as a last resort: Randomized object identifiers.

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Security Misconfiguration

Misconfigurations can often provide an attacker with a significant level of information about your infrastructure, components, and code.

Sometimes they can even result in vulnerabilities.

Security Misconfiguration

Security misconfigurations can manifest themselves in many ways:

- Failure to change default passwords and remove sample applications.
- Failure to customize error pages to remove detailed information about systems.
  - Version Numbers
  - Stack Traces
- Failure to disable unused features.
  - Directory Listing

Avoiding Security Misconfiguration

- Well-documented deployment and hardening processes.
  - OS, Web/App Server, DBMS, Applications, and Libraries
  - Disable all un-necessary features and functionality.
  - Limit access to administrative facets of the application.
- Robust patch and configuration management.
  - Keep all of your infrastructure up-to-date.
- Solid, multi-tier application and network architecture.
- Regular auditing, scanning, and penetration testing.
  - Don’t wait for the bad guys to find your vulnerabilities.

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Sensitive Data Exposure

Web applications containing sensitive information must take extra precautions, or risk a breach.

Breaches can come with mandatory reporting requirements, depending on your jurisdiction and the type of data.

What is Sensitive Data?

- Credit Card Numbers (aka PAN)
- Social Security/Driver’s License Numbers
- Bank Account Numbers
- Many other forms of ID number that are issued to an individual.
- Healthcare Information (HIPAA)

And in some jurisdictions:

- Name, Birthdate, Address

The Coffee Shop Rule

If you wouldn’t be comfortable yelling a piece of information about yourself to someone across a busy coffee shop, it’s quite possibly sensitive data.

Treat it as such.

Avoiding Sensitive Data Exposure

- If you don’t have to store it, don’t store it.
- Make sure to use encrypted connections.
- If you do have to store it, encrypt it!
  - Row- or cell-level encryption is best.
  - Use a strong key, stored securely.
- Backups, too!

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Missing Function-Level Access Control

Occurs when web applications limit access to areas of the application simply by not showing links to those areas or resources.

An attacker, through guessing or through prior knowledge of the application, can access these resources by entering the URL directly.

Missing Function-Level Access Control

```
[index.php]
<h1>Main Menu:</h1>
<li><a href="showPage?userList">User List</a></li>
<?php if ( userType=="admin" ) { ?>
    <li><a href="showPage?adminTools">Admin Tools</a></li>
<?php } ?>
<li><a href="showPage?logout">Log Out</a></li>

…
[showPage.php]
$action = $_SERVER['QUERY_STRING'];
if ($action == 'userList') {showUserList();}
else if ($action == 'adminTools') {showAdminTools();}
else if ($action == 'logout') {logout();}
else { showMainMenu(); }
```

What happens if a malicious user types http://mywebapp.com/showPage?adminTools into the browser after logging in? How do we fix it?

Missing Function-Level Access Control

```
[index.php]
<h1>Main Menu:</h1>
<li><a href=“showPage?userList”>User List</a></li>
<?php if ( $userType==“admin” ) { ?>
    <li><a href=“showPage?adminTools”>Admin Tools</a></li>
<?php } ?>
<li><a href=“showPage?logout”>Log Out</a></li>

…
[showPage.php]
$action = $_SERVER[“QUERY_STRING”];
if ($action == “userList”) {showUserList();}
else if ($action == “adminTools” &&
         $userType == “admin”) {showAdminTools();}
else if ($action == “logout”) {logout();}
else { showMainMenu(); }
```

Simple – add function-level access checks!

Missing Function-Level Access Control

Other common examples:

- Not checking that the user is authenticated at the beginning of every request.
- “Hiding” functionality by not linking to it.

MFLAC Demo

Another bug in an old version of WordPress.

Avoiding Forced Access

- Check the user’s access when the user attempts any action within the application.
- Never rely solely on presentation-layer access control.
- Code access control into the core of the application, or use a framework that embeds access control.

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Cross-Site Request Forgery

CSRF attacks are the result of a malicious website generating a valid request to a target site, relying on the fact that the user accessing the malicious site also has an active session on the target site.

Cross-Site Request Forgery

Example: Jane User is doing a little online banking.

Jane gets an e-mail from her dad telling her to “check out this cute cat picture”, and a link.

Jane clicks on the link…

…and gets a cute cat! And maybe a link to “click here for more cats!”.

That link gives her more cats … and also transfers $1000 from her account to an account in Bermuda!

Cross-Site Request Forgery

So how did this happen?

```
<form method="post" action="https://mybank.com/doTransfer">
Enter amount to transfer: <input name="amount">
Enter ID of account to transfer from: <input name="from"
value="Checking">
Enter account number to transfer to: <input name="to" >
</form>
```

Generates a form POST request that looks like:

```
amount=1500.00&from=Checking&to=319029109
```

Easy, and predictable. And easy for the attacker to replicate, as long as Jane is logged into her banking site.

Cross-Site Request Forgery

- Combine this with an XSS attack.
- Now we can use XMLHttpRequest!

Avoiding CSRF

Fairly simple to avoid:

- Place a unique (per-session, at least) token in any form that will post data back to the application.
- OWASP CSRF Guard and ESAPI

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Known-Vulnerable Components

Libraries, frameworks, servers, and operating systems have vulnerabilities discovered by security researchers.

Continuing to use those vulnerable components puts your application and infrastructure at risk.

Avoiding Vulnerable Components

- Track your components usage, versions, and sources.
- Stay on top of updates.
- Be prepared to mitigate vulnerabilities with infrastructure.

The OWASP Top 10

A1. Injection

A2. Broken Authentication & Session Management

A3. Cross-Site Scripting (XSS)

A4. Insecure Direct Object References

A5. Security Misconfiguration

A6. Sensitive Data Exposure

A7. Missing Function-Level Access Control

A8. Cross-Site Request Forgery (CSRF)

A9. Using Known-Vulnerable Components

A10. Unvalidated Redirects and Forwards

Unvalidated Redirect/Forwards

Applications which fail to validate untrusted data which is then used as all or part of a redirect destinations can allow their users to be unknowingly sent to malicious sites.

Unvalidated Redirects/Forwards

Example:

http://www.yourbank.com/mainMenu

redirects to:

http://www.yourbank.com/doLogin?next=http://www.yourbank.com/mainMenu

User logs in, then is redirected to URL specified as 'next'.

http://www.yourbank.com/mainMenu

Simple, right?

Unvalidated Redirects/Forwards

Well, what if a phisher sends out an e-mail that looks like:

To: yourmom@yahoo.com
From: security@yourbank.com

Your account is at risk!

Our contact records are out of date. Please log in to our website at:

`https://www.yourbank.com/doLogin?next=https://www.scam.com/yourbank.com`

She's probably going to click, right?

Avoiding Unvalidated Redirects/Forwards

Easier than some other issues:

- Avoid using redirects and forwards.
- If you do have to use them:
    - Don’t base them directly on data coming from the browser.
    - Name your destinations.
    - Use relative paths (but be sure to sanitize the path).
    - If you must use full URLs, have a lookup table of allowed URLs to redirect to.

AppSec and the SDLC

AppSec and the SDLC

How can developers make sure the systems you're developing are secure?

Make sure systems are developed securely!

Integrate security into your SDLC.

AppSec and the SDLC

"Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase."

http://www.cs.umd.edu/~basili/publications/proceedings/P95.pdf

The goal: Move security as early into the SDLC as possible.

AppSec and the SDLC

Start with figuring out your current state.

- Penetration Testing
- Code Review

Where did the problems originate?

- Coding practices?
- Infrastructure issues?
- Architecture issues?
- Process Issues?

AppSec and the SDLC

- Fix the issues discovered.
- Develop repeatable processes and procedures to prevent the issues from happening again.
  - Code standards
    (GOOD CODE == SECURE CODE!)
  - Deployment standards
  - Architecture standards

AppSec and the SDLC

- Continue working backwards in the SDLC.
  - First security testing along with QA testing, then…
  - Automated security testing alongside unit testing.
  - Library and configuration management.
  - Static code analysis tools in pre-commit hooks.
  - Manual spot checks of checked in code for security issues.

AppSec and the SDLC

And last, but not least…training. The more developers in your organization that understand AppSec, the more secure your organization will be.

Eventually, Application Security will become simply another piece of the development process.

Summary & Conclusion

Summary

The Internet can be a nasty place for a web application…

…but if security is baked into that application, it can be a lot less nasty for everyone involved.

???

Questions?

Additional Reading

OWASP Top Ten 2013 – RC1

- *http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013%20-%20RC1.pdf*

OWASP Cheat Sheets

- *https://www.owasp.org/index.php/Cheat_Sheets*

And check http://jferg.thedotin.net/AppSec/ later next week for some other resources.