1st Chapter Meeting Columbus, OH

Chris Hayes
Chapter Leader
Nationwide Insurance
hayesc8@nationwide.com
614-249-5532

OWASP
3/24/2008

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

Agenda

6:00-6:05	Welcome
6:05-6:30	OWASP and Columbus Chapter Overview
6:30-7:30	Web Session Security (OWASP Top 10 - Broken Authentication and Session Management)
7:30-7:45	General Discussion / Meet & Greet

Welcome!

- Welcome to the 1st Columbus, Ohio OWASP Chapter Meeting
- Thank-you Nationwide Insurance for being the meeting sponsor tonight. They provided both the space we are utilizing and the food.
- Restrooms are located in the back of this room.
- Please remember to sign the CPE List (incl. an email address that a PDF can be sent to for proof of attendance)
- We will be conducting a web-based survey in the coming weeks. Your feedback will be used for planning future meetings!

Special Event

- Cincinnati OWASP Chapter Special Event / sponsored by Fortify
- Premiere of the short movie (22 minutes): The New Face of Cybercrime “@Work Premiere”
- Where: Citigroup NA, Blue Ash
- Time: 5:30-7:30 PM
- RSVP by 4/20/2008 To: marco.m.morana@gmail.com or http://www.owasp.org/index.php/Cincinnati
- Trailer: http://www.youtube.com/watch?v=-5zxOLZ5jXM

What is OWASP?

- Open Web Application Security Project (OWASP)
- A worldwide free and open community focused on improving the security of application software
- The OWASP mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks.
- 114 Chapters Worldwide (as of 3/20/2008)

What is OWASP? (cont.)

- An abundance of free resources (publications, software, local chapters, wealth of information)
- OWASP is becoming referenced in various standards like the PCI DSS (Payment Card Industry Data Security Standard); referenced in numerous US Government guidelines.
- Checking for OWASP Top 10 vulnerabilities is becoming a common feature in various security products.
- Bottom line: the information is relevant, this is a global effort, and the information is free!

OWASP in Columbus, OH

- Chapter was restarted in December of 2007 as part of an effort to identify some training resources that could be recommended for some of my peers and application team members I work with.

- It felt appropriate for there to be a chapter here in Columbus given the number of Fortune 500 companies that call Columbus home (healthcare, financial services, transportation, and manufacturing verticals). There is a large educational presence in central OH. A large number of IT consulting firms focusing on both SMB and/or large enterprises. Large state government presence.

- Current Chapter Leaders

  - Chapter Leader: Chris Hayes
  - Vice Chapter Leader: Greg Green
  - Chapter Secretary: ?? YOU ??

Participation

- Your involvement is critical to the chapter.
- Attend these meetings; encourage others to attend.
- Volunteer to present! You do not need to be the subject matter expert!
- These meetings are for collaborative education, information sharing, and getting to know others!

Popular OWASP Projects

- There is an abundance of resources available to you for free or under a commercial license.
- This portion of the presentation covers some of the most popular OWASP resources.

OWASP Top 10

** http://www.owasp.org/index.php/Top_10_2007 **
** Available in book form

- The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are.
- Project members include a variety of security experts from around the world who have shared their expertise to produce this list.
- Numerous US Government, commercial bodies, commercial companies, and education institutions have adopted or recommend use of the OWASP Top 10 (http://www.owasp.org/index.php/OWASP_Top_Ten_Project)
- Every Top 10 vulnerability page includes numerous samples, protection steps, and reference links.
- Adopting the OWASP Top Ten is perhaps the most effective first step towards changing the software development culture within your organization into one that produces secure code.

OWASP Books (VIRAL)

** http://stores.lulu.com/owasp **

- Currently 10 OWASP books are available

- Electronic versions can be downloaded for free and as well as freely distributed

OWASP Guide 2.0 (2005)
OWASP World (Nov 2007)
OWASP SpoC 2007
OWASP Top 10 - Ruby on Rails version
OWASP Evaluation And Certification Criteria
OWASP Code Review - 2008 (RC2)
OWASP WebGoat and WebScarab
OWASP Top10 - Testing - Legal 07
OWASP Top 10 - 2007 Edition
OWASP CLASP v1.2

- These books are provided AT COST and OWASP is not making any profit with these sales.

OWASP WebGoat

** http://www.owasp.org/index.php/OWASP_WebGoat_Project **

- WebGoat is a deliberately insecure J2EE web application maintained by OWASP designed to teach web application security lessons (GNU-GPL).

- The primary goal of the WebGoat project is simple: create a de-facto interactive teaching environment for web application security.

OWASP WebGoat (cont.)

- In each lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat application.

- There are currently over 30 lessons, including those dealing with the following issues:

Cross Site Scripting
Thread Safety
Parameter Manipulation
Blind SQL Injection
String SQL Injection
Fail Open Authentication
… and many more!

Access Control
Hidden Form Field Manipulation
Weak Session Cookies
Numeric SQL Injection
Web Services
Dangers of HTML Comments

- Why the name "WebGoat"? Developers should not feel bad about not knowing security. Even the best programmers make security errors. What they need is a scapegoat, right? Just blame it on the 'Goat!

OWASP WebScarab

** http://www.owasp.org/index.php/OWASP_WebScarab_Project **

- WebScarab is a framework for analyzing applications that communicate using the HTTP and HTTPS protocols.
- Written in Java, and is thus portable to many platforms.
- WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser.
- WebScarab is able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have passed through WebScarab.
- There are plug-ins for Web-Scarab for use with SOAP and XSS/CRLF testing.
- The lesson topic for this evening will give you a glimpse of WebScarab.

OWASP – Web Session

Gregory S. Green
OWASP Columbus Chapter
Nationwide Insurance
greeng1@nationwide.com
614-249-6375

OWASP
03/24/2008

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

OWASP – Web Session

Gregory S. Green
OWASP Columbus Chapter
Nationwide Insurance
greeng1@nationwide.com
614-249-6375

OWASP
03/24/2008

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

Web Session Management*

- Persistent Information
- Java Servlets
- Session Tracking
- Hidden Fields
- Cookies
- Servlet HTTP Request
- Servlet HTTP Session
- Summary
- Demos
- Q&A / Discussion

**Courtesy of David Matuszek, Computer and Information Science, University of Pennsylvania*

Persistent Information

- A server site typically needs to maintain two kinds of persistent (remembered) information:
  - Information about the session
    - A session starts when the user logs in or otherwise identifies himself/herself, and continues until the user logs out or completes the transaction (for example, makes a purchase)
  - Information about the user
    - User information must generally be maintained much longer than session information (for example, remembering a purchase)
    - This information must be stored on the server, for example on a file or in a database

Java Servlets

- Servlets, like Applets, can be trusted or untrusted
  - A servlet can use a unique ID to store and retrieve information about a given session
  - User information usually requires a login ID and a password
  - Since servlets don't quit between requests, *any* servlet can maintain information in its internal data structures, as long as the server keeps running
  - A *trusted* servlet can read and write files on the server, hence can maintain information about sessions and users even when the server is stopped and restarted
  - An untrusted servlet will lose *all* information when the servlet or server stops for any reason
    - This is sometimes good enough for session information
    - This is almost never good enough for user information

Session Tracking

- HTTP is stateless: When it gets a page request, it has *no memory* of any previous requests from the same client
  - This makes it difficult to hold a “conversation”
    - Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests
  - The server must be able to keep track of multiple conversations with multiple users
- Session tracking is keeping track of what has gone before in this particular conversation
  - Since HTTP is stateless, it does not do this for you
  - You have to do it yourself, in your servlets

Session Tracking (continued)

- Cookies are small files that the servlet can store on the client computer, and retrieve later
- URL rewriting: You can append a unique ID after the URL to identify the user
- Hidden <form> fields can be used to store a unique ID
- Java's Session Tracking API can be used to do most of the work for you

Hidden (HTML form) Fields

- <input type="hidden" name="sessionID" value="...">
- Advantage:
  - Requires the least knowledge: All you need to know is how to read and write parameters
- Disadvantages:
  - Not kept across sessions, so useless for maintaining persistent information about a user
  - Since the session ID must be incorporated into every HTML page, every HTML page must be dynamically generated
- There's not much more to say about using hidden form fields, since you should already know enough to do it

Cookies

- A cookie is a small bit of text sent to the client that can be read again later
  - Limitations (for the protection of the client):
    - Not more than 4KB per cookie (more than enough in general)
    - Not more than 20 cookies per site
    - Not more than 300 cookies total
- Cookies are *not* a security threat
- Cookies *can be* a privacy threat
  - Cookies can be used to customize advertisements
  - Outlook Express allows cookies to be embedded in email
  - A servlet can read your cookies
    - Incompetent companies might keep your credit card info in a cookie
    - Netscape lets you refuse cookies to sites other than that to which you connected

Cookies (continued)

- import javax.servlet.http.*;
- Constructor: Cookie(String name, String value)
- Assuming ***request*** is an HttpServletRequest and ***response*** is an HttpServletResponse,
  - ***response***.addCookie(cookie);
  - Cookie[ ] cookies = ***request***.getCookies();
    - String name = cookies[i].getName();
    - String value = cookies[i].getValue();
- There are, of course, many more methods in the HttpServletRequest, HttpServletResponse, and Cookie classes in the javax.servlet.http package

Cookies (continued)

- public void setComment(String purpose)
  - public String getComment()
- public void setMaxAge(int expiry)
  - public int getMaxAge()
  - Max age in seconds after which cookie will expire
  - If expiry is negative, delete when browser exits
  - If expiry is zero, delete cookie immediately
- setSecure(boolean flag)
  - public boolean getSecure()
  - Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL

Servlet HTTP Request

- public HttpSession getSession()
  - Gets the session object for this request (or creates one if necessary)
- public Enumeration getHeaderNames()
  - Gets an Enumeration of all the field names in the HTTP header
- public String getHeader(String name)
  - Given the header name, return its value
- public int getIntHeader(String name)
  - Given the header name, return its value as an int
  - Returns -1 if no such header
  - Could throw a NumberFormatException
- public Enumeration getHeaders(String name)
  - Given the header name, return an Enumeration of all its values

Servlet HTTP Session

- The session tracking API is in javax.servlet.http.HttpSession and is built on top of cookies
- To use the session tracking API:
  - Create a session:
    - HttpSession session = ***request***.getSession();
      - Returns the session associated with this request
      - If there was no associated session, one is created
  - Store information in the session and retrieve it as needed:
    - session.setAttribute(***name***, ***value***);
    - Object ***obj*** = getAttribute(***name***);
- Session information is automatically maintained across requests

Summary

- A session is a continuous interaction with the user
  - HTTP is stateless, so the programmer must do something to remember session information
  - There are multiple ways to remember session information
  - The session ends when the user quits the browser (or a session may be set to time out)
- Some information must be kept longer than just within a session
  - For example, if the user orders a product, that information must be kept in a database
  - Long-term storage of information requires that the servlet have some additional privileges

Demos

- Threat community
  - Authorized user elevating privileges
  - Why? HTTP vs. HTTPS
- Demo #1
  - OWASP WebGoat cookie usage
  - OWASP WebScarab HTTP tampering tool
- Demo #2
  - Cookie servlet in IBM Rational (Eclipse) IDE
  - OWASP WebScarab
- Demo #3
  - Jsession servlet in IBM Rational (Eclipse) IDE
  - OWASP WebScarab

Q&A / Discussion

Wrap-Up

- A PDF of the presentation will be posted on the main OWASP website. I will send out a link.

- An email will be sent with a link to a web survey. Please take a minute or two to respond to the survey. Your feedback will be used for planning future meetings.

- Let Greg or I know if you are interested in presenting at or sponsoring future meetings.

THANK YOU FOR ATTENDING !!

```java
import java.io.*;
import java.io.IOException;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CookieDemo extends HttpServlet implements Servlet {
  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#HttpServlet()
   */
  public CookieDemo() {
    super();
  }

  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest arg0,
   HttpServletResponse arg1)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    Cookie[] cookieArray = request.getCookies();
    int i;
    boolean authFound = false;
    String authLevel = "1";
    PrintWriter out = response.getWriter();

    out.println("<HTML><BODY>");
    out.println("<TITLE>OWASP Demo Application Session Cookie</TITLE>");
    out.println("<H1>OWASP Demo</H1>");
    out.println("<H2>Application Session Cookie without Jsession Check</H2>");

    if (cookieArray != null) {
      for (i=0; i<cookieArray.length; i++) {
        Cookie c = cookieArray[i];
        if (c.getName().equals("authLevel")) {
          authFound = true;
          authLevel = c.getValue();
          if (authLevel.equals("1")) {
            out.println("Regular user (level 1) access");
          }
          else {
            if (authLevel.equals("2")) {
              out.println("Super user (level 2) access");
            }
            else {
              if (authLevel.equals("3")) {
                out.println("Administrator (level 3) access");
              }
              else {
                out.println("Invalid access level");
              }
```

```
            }
          }
        }
      }
    }
    if (!authFound) {
      out.println("Auth level has been set to level " + authLevel);
    }
    Cookie authCookie = new Cookie ("authLevel", authLevel);
    response.addCookie(authCookie);
    out.println("<BR><BR></BODY></HTML>");
  }

  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest arg0,
   HttpServletResponse arg1)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException {
    // TODO Auto-generated method stub
  }

}
```

```java
import java.io.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class JsessionDemo extends HttpServlet implements Servlet {
  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#HttpServlet()
   */
  public JsessionDemo() {
    super();
  }

  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest arg0,
   HttpServletResponse arg1)
   */
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    // TODO Auto-generated method stub
      PrintWriter out = response.getWriter();
      // response.setContentType("text/html");
      HttpSession session = request.getSession(true);
    int i;

    out.println("<HTML><BODY>");
    out.println("<TITLE>OWASP Demo Jsession</TITLE>");
    out.println("<H1>OWASP Demo</H1>");
    out.println("<H2>Jsession Check before Application Access</H2>");
      out.println("Jsessionid = " + session.getId() + "<BR><BR>");
      out.println("Creation Time = " + new Date(session.getCreationTime()) +
      "<BR><BR>");
      out.println("Time of Last Access = " + new
      Date(session.getLastAccessedTime()) + "<BR><BR>");

      Integer accessCount = (Integer)session.getAttribute("accessCount");
      if (accessCount == null) {
          accessCount = new Integer(0);
        session.setAttribute("accessCount", accessCount);
        session.setAttribute("authLevel", "1");
      out.println("Any auth level ignored - forced to be set to 1<BR>");
      } else {
          accessCount = new Integer(accessCount.intValue() + 1);
        session.setAttribute("accessCount", accessCount);
        String authLevel = (String)session.getAttribute("authLevel");
        out.println("Number of Previous Accesses = " + accessCount +
        "<BR><BR>");
        out.println("Auth level = " + authLevel + "<BR>");
      }
      out.println("</BODY></HTML>");
```

```
  }

  /* (non-Java-doc)
   * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest arg0,
   HttpServletResponse arg1)
   */
  /** Handle GET and POST requests identically. */
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
      doGet(request, response);
    }

}
```