ENPM 809W
Introduction to Secure Software Engineering

Gananand Kini

Lecture 9

Session Management related security bugs - Attacks

SOLVING PROBLEMS FOR A SAFER WORLD™

Outline

- **Session based attacks**
  - Session Hijacking
- **Cross-site Request Forgery**
- **Session lifetimes and resource exhaustion**

Session based attacks

What is a session?

- Session: A persistent interaction between a subscriber and an endpoint, either a relying party or a credential service provider. A session begins with an authentication event and ends with a session termination event. A session is bound by use of a session secret that the subscriber's software (a browser, application or operating system) can present to the relying party or the Credential Service Provider in lieu of the subscriber's authentication credentials.
- Simply put, a session is used to add "state" to an otherwise stateless protocol.
- Imagine having to put your password in for every action and resource that gets loaded when you visit a webpage. That will probably take forever!
- So a session in a web application is used to "cache" important state like authentication tokens, authorization tokens, cookies etc.

Session management core concepts

- Any application or software system that wants to track state with its actors needs to implement some form of session management.
- Even for tracking users as part of the application needs session management.

Session Hijacking

- **Now the minute you add state to your application or software system, you end up having to protect that state from malicious actors.**
- **The state typically uses a session identifier to uniquely identify the session while also using it like an authentication or authorization token.**
  - For example, the user logs in and gains a session ID, now anyone that bears the session ID is also assumed to be trusted and carrying that user’s authorizations and roles!
  - We already saw this in the authentication and authorization bugs lecture.
  - Authentication and authorization need to be managed correctly and separately!
- **With a Man-in-the-middle adversary that can sniff and potentially redirect traffic, important state variables like a session identifier can be sniffed and used by the adversary.**

Cross-Site Scripting (XSS) again …

- We heard about XSS attacks in the Input validation bugs lecture.
- Cross-site scripting attacks alone are normally not enough to have serious security impacts.
- They are combined with session hijacking to then run malicious code using a legitimate user’s session(s)!
- Enter Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF/XSRF)

Cross-Site Request Forgery (CSRF/XSRF)

- **Cross-site request forgery (CSRF/XSRF)**
  - XSS exploits user’s trust in a server
  - CSRF/XSRF exploits server’s trust in a client
- **In CSRF/XSRF:**
  - Attacker tricks user into sending data to server
  - Server believes that user consciously & intentionally chose that action
- **XSS fools clients; XSRF fools <u>servers</u>**
- **A failure of authentication and authorization in the server’s session management implementation!**

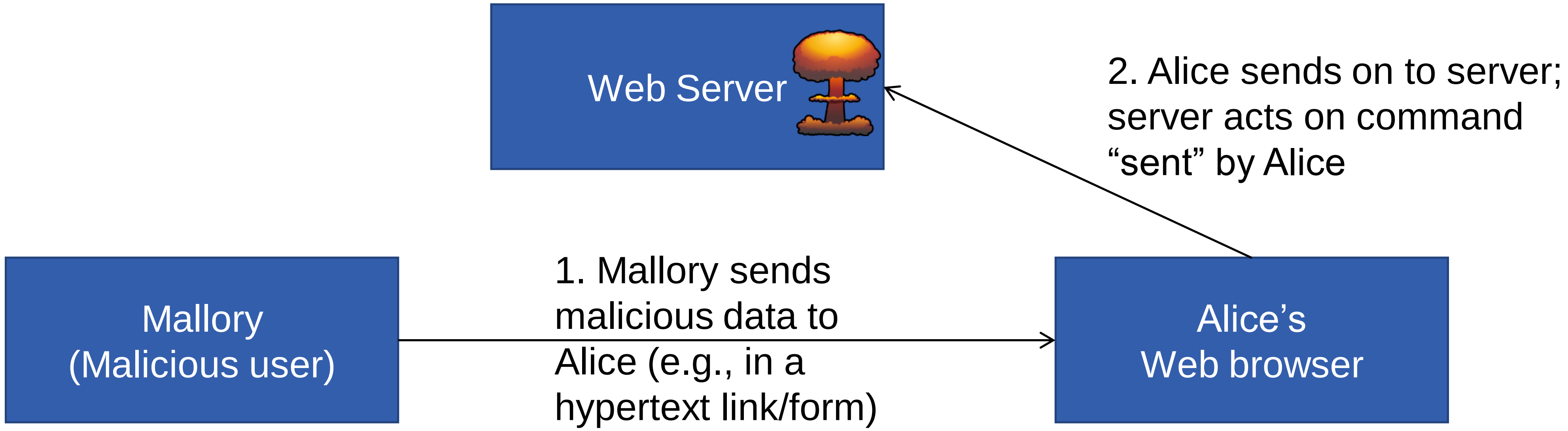
CWE-352: Cross-Site Request Forgery (CSRF)

- **Description: The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.**
- **When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request.**
- **This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.**

Sources:
1. CWE-352: Cross-Site Request Forgery (CSRF). MITRE CWE. https://cwe.mitre.org/data/definitions/352.html. Retrieved July 20, 2021.

Cross-Site Request Forgery (CSRF/XSRF) Example

- **In CSRF/XSRF, like reflected XSS, attacker sends data to victim so victim will send it on to server**
  - Attacker's approach is in many ways like reflected XSS and similar to a confused deputy problem (here the user's authorization is being misused).
- **Attacker's purpose is for server to act on the command**
  - Target is server not client – difference from XSS


Web Server
2. Alice sends on to server; server acts on command "sent" by Alice
Mallory (Malicious user)
1. Mallory sends malicious data to Alice (e.g., in a hypertext link/form)
Alice's Web browser

Cross-origin resource sharing (CORS)

- **Relaxes single-origin policy of web browsers (be careful!)**
- **When JavaScript makes cross-origin request, “Origin:” sent**
- **Server examines origin & replies in HTTP header if allowed:**
  - Access-Control-Allow-Origin: permitted origin or “*” (or null)
  - Access-Control-Allow-Methods: permitted methods (e.g., GET)
  - Access-Control-Allow-Credentials: if true, share credentials (!)
- **Web browser checks if origin specifically allowed (default “no”)**
  - Thus, both server & web browser must agree access permitted
- **Sometimes GET & POST are *directly* requested (see W3C spec)**
  - In many cases web browser first makes “preflight” request to server using OPTIONS to determine permission before making *actual* request
- ***Beware* of Access-Control-Allow-Credentials**
  - If “true” then credentials directly shared – only use if you *totally* trust other site & it’s absolutely necessary

For more information, see “Exploiting CORS Misconfigurations for Bitcoins and Bounties” by Jim Kettle, Oct. 14, 2016, http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

CWE-942: Permissive Cross-domain Policy with Untrusted Domains

- **Description: The software uses a cross-domain policy file that includes domains that should not be trusted.**
- **A cross-domain policy file ("crossdomain.xml" in Flash and "clientaccesspolicy.xml" in Silverlight) defines a list of domains from which a server is allowed to make cross-domain requests. When making a cross-domain request, the Flash or Silverlight client will first look for the policy file on the target server. If it is found, and the domain hosting the application is explicitly allowed to make requests, the request is made.**
- **Therefore, if a cross-domain policy file includes domains that should not be trusted, such as when using wildcards, then the application could be attacked by these untrusted domains.**
- **An overly permissive policy file allows many of the same attacks seen in Cross-Site Scripting (CWE-79). Once the user has executed a malicious Flash or Silverlight application, they are vulnerable to a variety of attacks. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site.**

Sources:
1. CWE-942: Permissive Cross-domain Policy with Untrusted Domains. MITRE CWE. https://cwe.mitre.org/data/definitions/942.html. Retrieved July 20, 2021.

CORS

- Now you are potentially opening paths for your software system to execute code that is sourced from another software system somewhere on the internet.
- How is the session on the server impacted by allowing scripts from a remote server to execute in the same session context? (In terms of authentication, authorization).

CWE-331: Insufficient Entropy

- **Description: The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.**

Sources:
1. CWE-331: Insufficient Entropy. MITRE CWE. https://cwe.mitre.org/data/definitions/331.html. Retrieved July 20, 2021.

Session Strength – Exploit Example

```
ACTIVE SESSIONS
-------------------------------------------
CGISESSID = 224b3666303552fd710867c8ac482a4
CGISESSID = 76a240fd7e228ee6e771a02d2c4921a
CGISESSID = ca3e617110de9fa3645a72c90f1677e
CGISESSID = ???????
```

```
ACTIVE SESSIONS
-------------------------------------------
CGISESSID = 1256
CGISESSID = 1257
CGISESSID = 1258
CGISESSID = ???????
```

Can you guess the next Session ID that will be issued in each case?

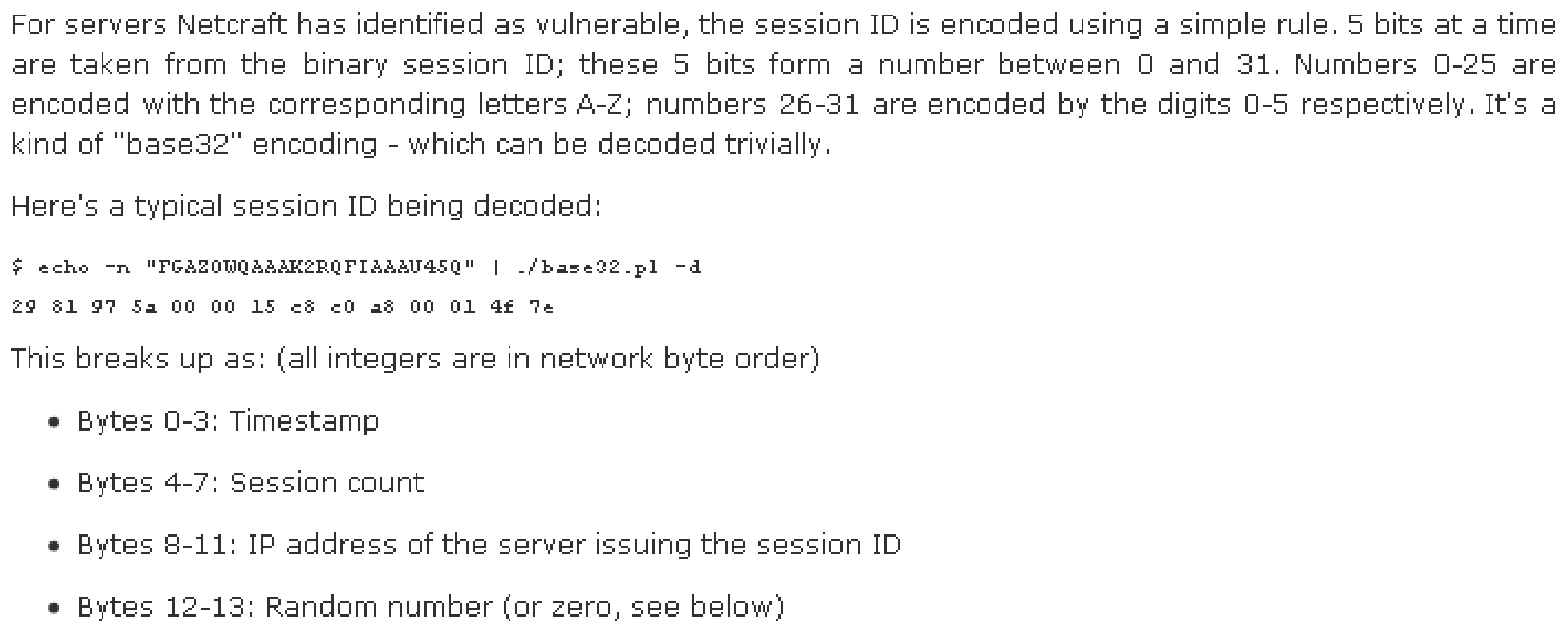
CWE-642: External Control of Critical State Data

- Description: The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors.
- State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker.
- When this state information is used to control security or determine resource usage, then it may create a vulnerability.

Sources:
1. CWE-642: External Control of Critical State Data. MITRE CWE. https://cwe.mitre.org/data/definitions/642.html. Retrieved July 20, 2021.

Real World – Session ID Weakness

Just because it looks random…



For servers Netcraft has identified as vulnerable, the session ID is encoded using a simple rule. 5 bits at a time are taken from the binary session ID; these 5 bits form a number between 0 and 31. Numbers 0-25 are encoded with the corresponding letters A-Z; numbers 26-31 are encoded by the digits 0-5 respectively. It's a kind of "base32" encoding - which can be decoded trivially.

Here's a typical session ID being decoded:

```
$ echo -n "FGAZ0WQAAAK2RQFIAAAU45Q" | ./base32.pl -d
29 81 97 5a 00 00 15 c8 c0 a8 00 01 4f 7e
```

This breaks up as: (all integers are in network byte order)

- Bytes 0-3: Timestamp
- Bytes 4-7: Session count
- Bytes 8-11: IP address of the server issuing the session ID
- Bytes 12-13: Random number (or zero, see below)

Timestamp goes up predictably, session count just increments, IP is static, and the 2 random bytes at the end are fixed at server start time.

Tovey, M. (2003, January 1). *Security advisory 2001-01.1*. Retrieved February 17, 2011, from http://news.netcraft.com/archives/2003/01/01/security_advisory_2001011_predictable_session_ids.html

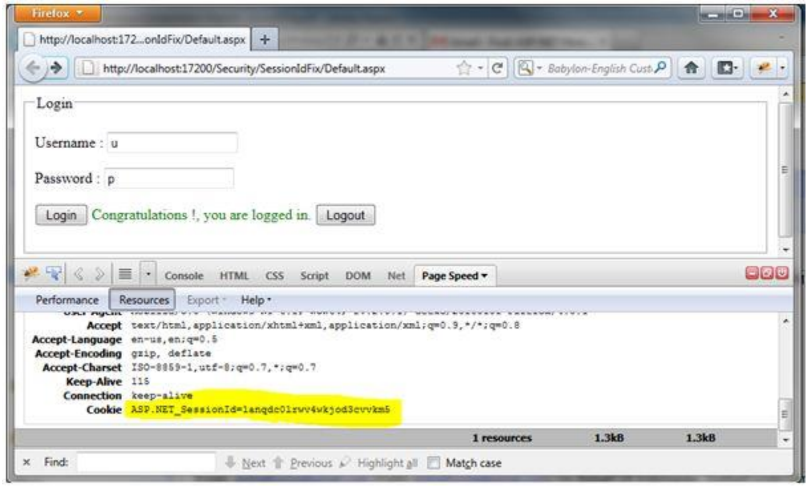
CWE-384: Session Fixation

- **Description: Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.**
- **Such a scenario is commonly observed when:**

1. A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user.
2. An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.
3. The application or container uses predictable session identifiers. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

Session Fixation Attack

- Authenticating a user or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.
- Another kind of confused deputy problem as well.



http://localhost:17200/Security/SessionIdFix/Default.aspx

Login

Username : u

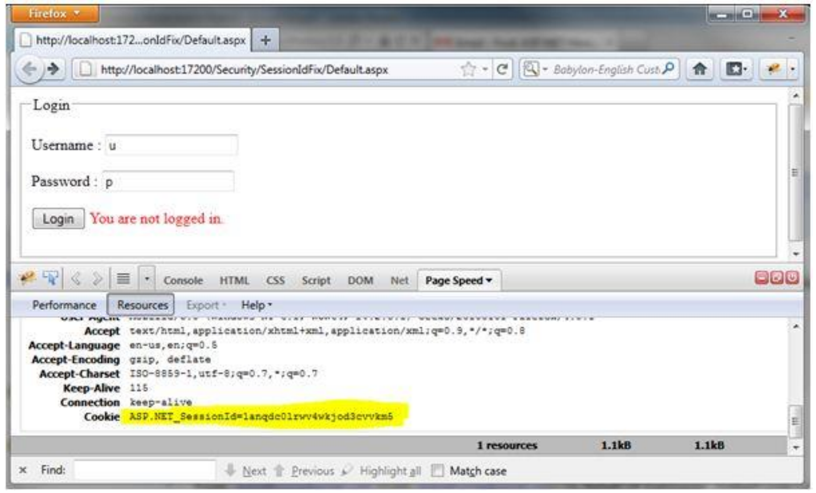
Password : p

Login Congratulations !, you are logged in. Logout

```
         Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip, deflate
 Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
     Keep-Alive 115
     Connection keep-alive
         Cookie ASP.NET_SessionId=1anqdc0lrwv4wkjod3cvvkm5
```

1 resources 1.3kB 1.3kB

After Login



http://localhost:17200/Security/SessionIdFix/Default.aspx

Login

Username : u

Password : p

Login You are not logged in.

```
         Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip, deflate
 Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
     Keep-Alive 115
     Connection keep-alive
         Cookie ASP.NET_SessionId=1anqdc0lrwv4wkjod3cvvkm5
```

1 resources 1.1kB 1.1kB

After Logout

Sources:
1. ITFunda. Session Fixation Vulnerability in ASP.NET. https://www.codeproject.com/articles/210993/session-fixation-vulnerability-in-asp-net.

Session Fixation Example

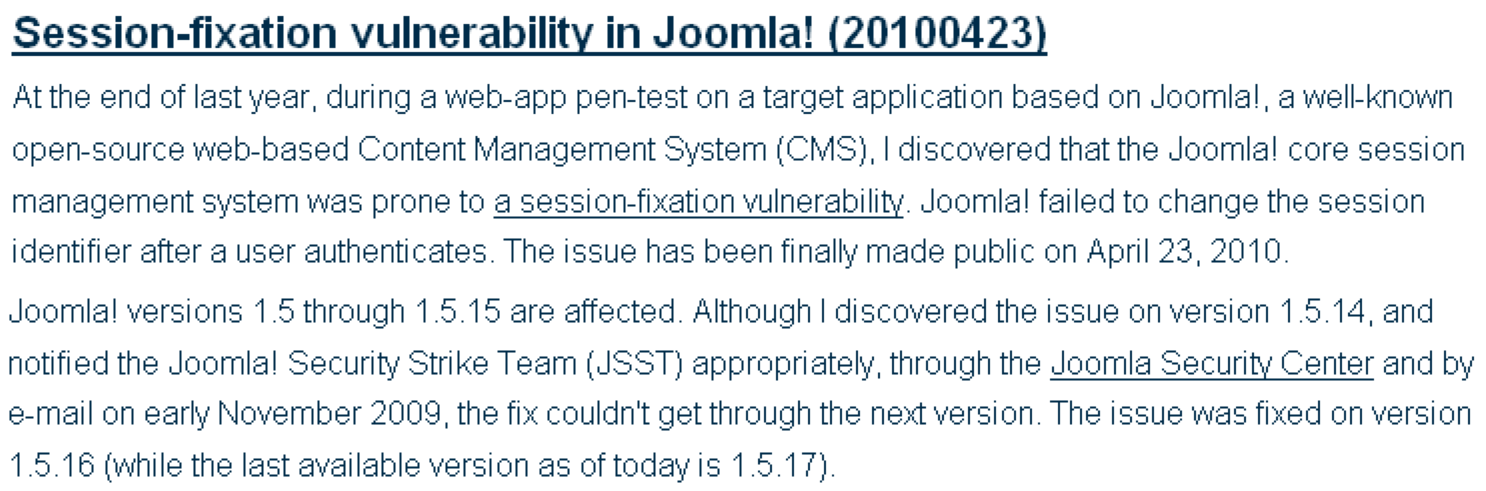
```
public int authenticate (HttpSession session)
{
   string username = GetInput("Enter Username");
   string password = GetInput("Enter Password");

   // Check maximum logins attempts
   if (session.getValue("loginAttempts") > MAX_LOGIN_ATTEMPTS)
   {
      lockAccount(username);
      return(FAILURE);
   }

   if (ValidUser(username, password) == SUCCESS)
   {
      session.putValue("login", TRUE);
      return(SUCCESS);
   }
   else return(FAILURE);
}
```

In order to exploit the code above, an attacker could first create a session (by visiting the login page of the application) from a public terminal, record the session identifier assigned by the application, and then leave the login page open. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session.

Real World – Session Fixation



Session-fixation vulnerability in Joomla! (20100423)

At the end of last year, during a web-app pen-test on a target application based on Joomla!, a well-known open-source web-based Content Management System (CMS), I discovered that the Joomla! core session management system was prone to a session-fixation vulnerability. Joomla! failed to change the session identifier after a user authenticates. The issue has been finally made public on April 23, 2010.

Joomla! versions 1.5 through 1.5.15 are affected. Although I discovered the issue on version 1.5.14, and notified the Joomla! Security Strike Team (JSST) appropriately, through the Joomla Security Center and by e-mail on early November 2009, the fix couldn't get through the next version. The issue was fixed on version 1.5.16 (while the last available version as of today is 1.5.17).

Joomla is a free, open-source content management system (CMS) and application framework that powers 2.7% of the entire web. With a vibrant volunteer community, user-friendly features and plenty of power, Joomla is perfect for your site! Learn more about 1.6 ...

Taddong. (2010, April 23). *Session-fixation vulnerability in Joomla!*. Retrieved February 17, 2011, from http://blog.taddong.com/2010/05/session-fixation-vulnerability-in.html

Session lifetimes and resource exhaustion

Session lifetimes and resource exhaustion

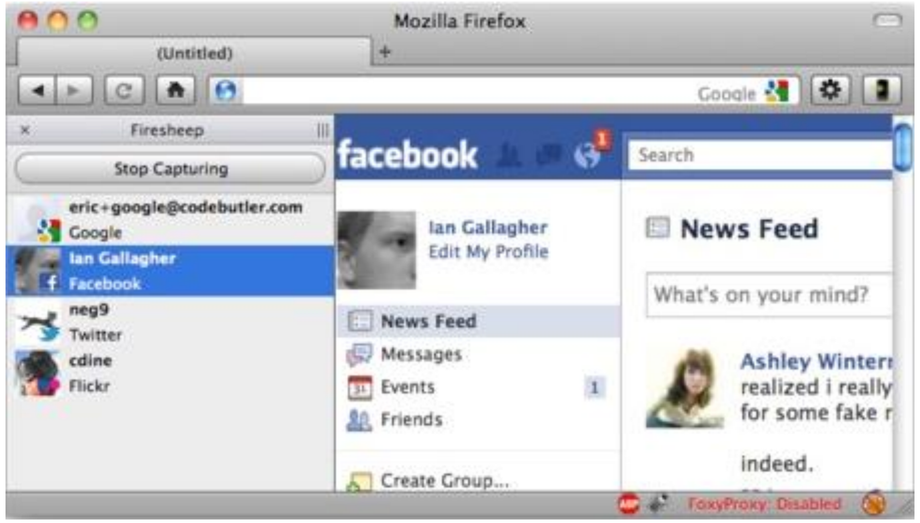
- **Sessions and session identifiers are typically not managed by the platform and left up to the system implementors or integrators.**
- **This can result in poor session management with unchecked long to infinite lasting sessions.**
- **An attacker can use the session identifier to ‘resume’ the user’s session.**
- **There is no tie between session and the client IP address, not mention either can be spoofed.**
- **Resources tied up in a session might stay claimed for as long as the session lasts.**
  - A crafty attacker can easily create multiple spurious sessions causing a resource exhaustion attack (server session quota is full etc.)

CWE-613: Insufficient Session Expiration

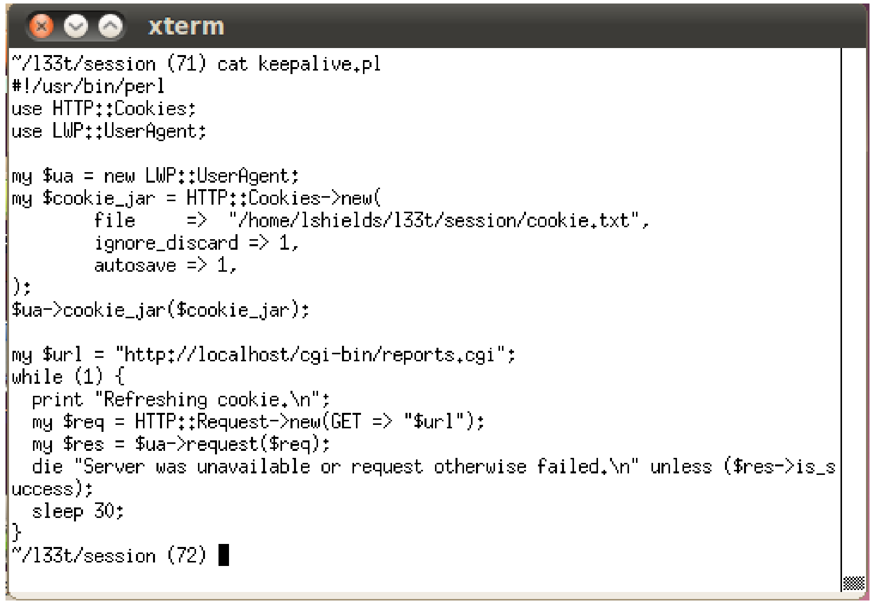
- **Description: According to Web Application Security Consortium, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization.“**

Sources:
1. CWE-613: Insufficient Session Expiration. MITRE CWE. https://cwe.mitre.org/data/definitions/613.html. Retrieved July 20, 2021.

Session Lifespan – Exploit Example



Determine or capture another user's session ID – e.g., Firesheep



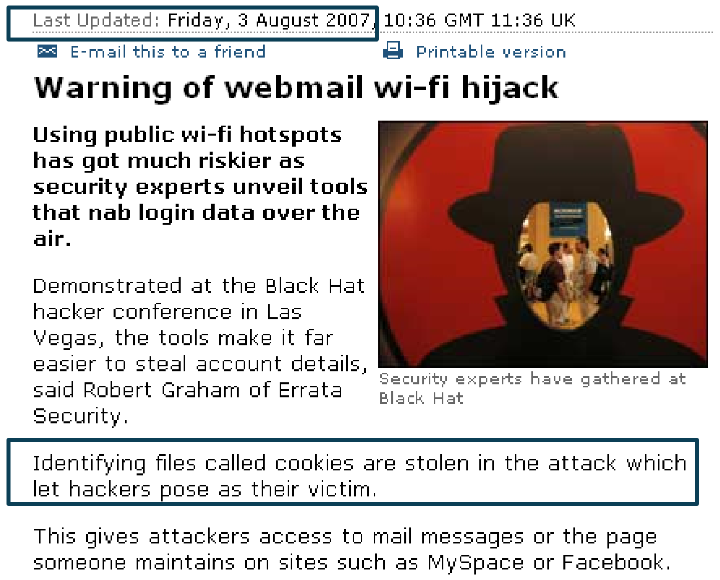
```
~/133t/session (71) cat keepalive.pl
#!/usr/bin/perl
use HTTP::Cookies;
use LWP::UserAgent;

my $ua = new LWP::UserAgent;
my $cookie_jar = HTTP::Cookies->new(
        file     =>  "/home/lshields/133t/session/cookie.txt",
        ignore_discard => 1,
        autosave => 1,
);
$ua->cookie_jar($cookie_jar);

my $url = "http://localhost/cgi-bin/reports.cgi";
while (1) {
  print "Refreshing cookie.\n";
  my $req = HTTP::Request->new(GET => "$url");
  my $res = $ua->request($req);
  die "Server was unavailable or request otherwise failed.\n" unless ($res->is_s
uccess);
  sleep 30;
}
~/133t/session (72) █
```

Keep the session alive and valid until the server reboots…

Real World – Session Lifetimes – Remember to hit logout!


Last Updated: Friday, 3 August 2007, 10:36 GMT 11:36 UK

E-mail this to a friend | Printable version

Warning of webmail wi-fi hijack

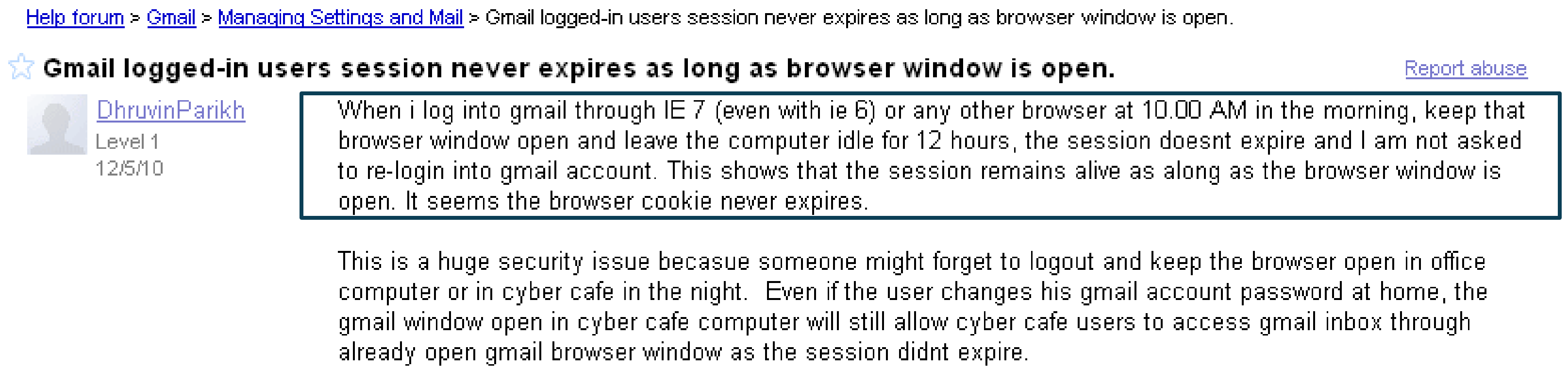
Using public wi-fi hotspots has got much riskier as security experts unveil tools that nab login data over the air.

Demonstrated at the Black Hat hacker conference in Las Vegas, the tools make it far easier to steal account details, said Robert Graham of Errata Security.

Security experts have gathered at Black Hat

Identifying files called cookies are stolen in the attack which let hackers pose as their victim.

This gives attackers access to mail messages or the page someone maintains on sites such as MySpace or Facebook.


Help forum > Gmail > Managing Settings and Mail > Gmail logged-in users session never expires as long as browser window is open.

Gmail logged-in users session never expires as long as browser window is open. Report abuse

DhruvinParikh
Level 1
12/5/10

When i log into gmail through IE 7 (even with ie 6) or any other browser at 10.00 AM in the morning, keep that browser window open and leave the computer idle for 12 hours, the session doesnt expire and I am not asked to re-login into gmail account. This shows that the session remains alive as along as the browser window is open. It seems the browser cookie never expires.

This is a huge security issue becasue someone might forget to logout and keep the browser open in office computer or in cyber cafe in the night. Even if the user changes his gmail account password at home, the gmail window open in cyber cafe computer will still allow cyber cafe users to access gmail inbox through already open gmail browser window as the session didnt expire.

Parikh, D. (2010, December 5). *Gmail logged-in users session never expires as long as browser window is open*. Retrieved February 17, 2011, from http://www.google.com.vc/support/forum/p/gmail/thread?tid=39841202c60f6b08&hl=en
Warning of webmail wi-fi hijack. (2007, August 3). Retrieved February 17, 2011, from http://news.bbc.co.uk/2/hi/technology/6929258.stm

Next time …

- Session Management related bugs - Defenses