An Effect of Cyber-Attacks on Session Management

K Bhupathi Reddy
2nd year MCA,
KMM Institute of post graduate studies, Tirupati.

P Ashok Kumar
2nd year MCA,
KMM Institute of Post graduate studies, Tirupati.

Dr G. V. Ramesh Babu
Asst.,Prof., Dept., of MCA
S.V.University, Tirupathi

***Abstract* --An attacker specifically targeting the session management process is growing on a daily basis. The hacker can steal password by means of several practices, like guessing or brute force attack etc. a lot of social websites like Facebook and Gmail along with banks and other financial institutions websites, are using two-factor authentication for security. Under these fake identities, attackers can steal sensitive data, alter private settings, and compromise website structure and content. This article describes Web application design flaws that could be exploited for session management attacks and discusses these flaws' current prevalence.**

Keywords: Session Management, Tokens, Session attacks, session vulnerabilities.

I. INTRODUCTION

Today browsers have become complex software applications. It allows users to do more than just viewing web pages, adding up to their usability level through many other functions. One of these functions is to manage user passwords for different sites. The need for this option come from the fact that Web developers implement in their applications secure sessions that require users to authenticate with their username and password. In computer science, in particular networking, a session is a semi-permanent interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user[1]. Session management tracks user's activity across sessions of interaction with a website.

II. SESSION MANAGEMENT:

Session management most wide spread use is login, but it's also used when the user isn't required to log in, as in the case of many e-commerce websites or web based social networks. The typical way to implement it is to associate each user with unique identifier- the session ID or session token. The Token implementation typically employs one of these mechanisms:

- Tokens are stored in cookies.
- Tokens are sent in hidden fields of a specific form on the website.
- Tokens, once created by server, are added to each link the user clicks on.

Some application use HTTP authentication. The Browser could use the HTTP header, rather than the application's web page code, to send user credentials. The main vulnerabilities concern token generation and session management mechanisms.

III. SESSION MANAGEMENT TOOLS:

There are three most representative tools for session management vulnerabilities.

- Rapid7's Nexspose is a platform for assessing web application vulnerabilities. The vulnerabilities which are focused by this are SQL injection and cross-site scripting.
- EEye analyzes a website's structure, content, and resources to find vulnerabilities. It focus on local applets or objects, and hidden fields.
- Nessus is a scanner for security policy assessment with some features for web application security, but it doesn't focus on session management.
-

IV. TOKEN GENERATION:

This kind of vulnerability lets attackers generate and use a valid token. Tokens can be created by composing some pieces of user information, such as a username or e-email address. If these schemas are reversible, an attackers could decode the token and create a valid one.

Attackers can predict tokens with higher probability when the token-creating algorithm uses one of three strategies. Hidden sequences generates tokens by coding a normal sequence of numbers. In time dependences, tokens are function of generation time. The third strategy is the weak generation algorithm. They employ pseudorandom number generators (PRNG's).

V.SESSION MANAGEMENT MECHANISMS

Even if a token is properly generated and unpredictable, attackers could intercept it. They can do this by exploiting unencrypted transmissions or weak mechanisms for preserving the cryptographic keys that a website uses to generate tokens.

Another way to intercept tokens is by detecting them from log files, such as browser logs, Web server logs, and server proxy logs. If then token is passed as a URL, parameter, an attacker can read it on the log.

Yet another way is to find tokens in a browser or proxy cache, which can record the entire webpage and the response header.

Other ways include exploiting faulty mechanisms user to assign tokens, assigning multiple tokens to the same user, and using static tokens for each user.

Additionally, poor session termination policies create many opportunities for attack. To reduce the temporal window for attacks, the session should be as short as possible. Some applications provide no mechanism for a session's expiration, which enables attackers to try many values before the session expires. When a user logs out, logs out, the server removes that token from the user's browser, but if the user (or attacker) sends a previously used token, the server receives no request at logout and doesn't invalidate the session. If an attacker obtains this token, the attacker could use the session, just as the user who never logged out could.

Attackers could intercept by following ways

- Exploiting unencrypted transmissions or weak mechanisms for preserving the cryptographic keys that a website uses to generate tokens.
- By detecting them from log files, such as browser logs, Web server logs, and server proxy logs. If then token is passed as a URL, parameter, an attacker can read it on the log.
- Another way is to find tokens in a browser or proxy cache, which can record the entire webpage and the response header.

And also poor session termination policies create many opportunities for attack. When a user logs out, the server removes that token from the user's browser, but if the user (or attacker) sends a previously used token the server keeps accepting it. In the worst case, the server receives no request at logout and does not in validate the session. If an attacker obtains this token, the attacker could use the session, just as the user who never logged out could.

If the token is captured in a cookie, cookie parameter settings might contain other vulnerabilities. If a cookie does not have secure flag set, the cookie will be send in unencrypted transmissions. If the HTTP only flag isn't set, attacker can catch it through cross side scripting (XSS) attacks. Attackers could also explode a cookies scope.

V. SESSION SUSCEPTIBILITIES

- Some applications identify protected area that use HTTPs but use the same token out side the protected area. Attacker can obtain the token by intercepting HTTP transmissions.
- Some application allow HTTP connections even in protected areas, where HTTP's should be used. So, attackers can induce users to make an HTTP request and then steal the token. Such attacks commonly used phishing mails, banners, or social engineering.
- Some application use an HTTP connection to access static content as well as images, scripts and cascading style sheets attacker can captures tokens by intercepting these request.

VI. ATTACKS ON SESSION

Attackers can perform attacks such as

- Session sniffing
  - http packet sniffing
  - log sniffing
  - cache sniffing
  - XSS cookie sniffing
- Predictable session ID
- Session validity
- CSRF
- Session Fixation

A. Session sniffing:

These attacks consists of passively intercepting a sessions dataset that's being transmitted

i. HTTP packet sniffing:

These attacks intercepts http packets. Attacker must look at a sniffer in a machine in the network of the victim or the organization responsible for the web application. There are four enabling vulnerabilities.

The area of the web site doesn't use http's is identifiable.

The secure flag isn't set. The application allows http request for pages under https. The application uses http before authentication.

ii. Log sniffing:

These attacks obtains the token by analyzing log files in the different systems involved in client server communication there are two enabling vulnerabilities. The token is transmitted as a URL parameter, in which case it might be recorded in the log files.

The token is transmitted as hidden field and the server accept get request in the place of post requests. Such a request inversion could be realized by a client side script.

iii. Cache sniffing:

If the attacker accesses the browser or proxy cache, the attacker could obtain the token in any format contain as cookie, URL parameter, or hidden field. Cache can manage by two types directive aren't in the HTTP response header and the directive cache control: private enables the cache only on the machine on which the user is working. This is the main risk for shared machines.

iv. XSS cookie sniffing:

A cross-site scripting attack is a kind of attack on web applications in which attackers try to inject malicious scripts to perform malicious actions on trusted websites. In

cross-site scripting, malicious code executes on the browser side and affects users. Cross-site Scripting is also known as an XSS attack. The first question that comes in mind is why we call it "XSS" instead of "CSS." The answer is simple and known to all who work in web development. In web design, we have cascading style sheet s (CSS). So cross-site scripting is called XSS so it does not get confused with CSS.

If the input is not properly encoded and sanitized, this injected malicious script will be sent to users. And a browser has no way to know that it should not trust a script. When the browser executes the script, a malicious action is performed on the client side. Most of the times, XSS is used to steal cookies and steal session tokens of a valid user to perform session hijacking.

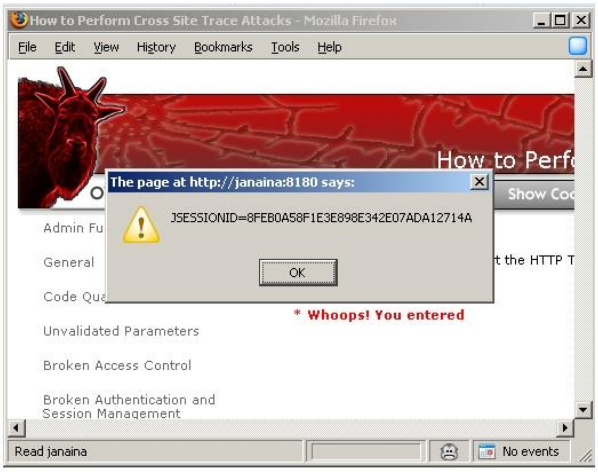

Fig. Cross Site Trace Attacks

Example:

Suppose there is a website with a messaging feature. In this website, users can send messages to their contacts. A basic form will look something like this:

```
<form action="sendmessage.php" method="post"'>
<textarea name="message"> </textarea>
<input type="submit" value="send" />
</form>
```

When this form is submitted, the message will be stored in the database. Another person will see the message when he opens the message from the inbox. Suppose an attacker has sent some cookie-stealing script in the message. This script will be stored on the website as a message. When the other person tries to read the message, the cookie-stealing script will be executed and his session id is now on the attacker's side. With a valid session id, the attacker can hijack the other person's account.

B. Predictable session id:

The most common flaw in session ID usage has always been predictability. As discussed earlier, the two causes are a lack of randomness, or length, or both.

- Sequential allocation of Session ID's - Each visitor to the site is allocated a session ID in sequential order. Thus, by observing your own session ID information, the simple practice of replacing it with another value a few iterations up or down will allow the attacker to impersonate another user.
- Session ID values are too short - The full range of valid session ID's could be covered during an automated attack before there is time for the session to expire.
- Common hashing techniques - While many commercial web services have built in functions for calculating hashed information, these mechanisms are well known and available for reproduction. A hashing function will indeed create a session ID value that appears to be unique and great care should be taken to ensure that predicable information is not used in the generation of the hash. For example, there have been cases where the "unique" hash was based upon the local system time, and the IP address of the connecting host. Using the same hashing function, the attacker would be able to pre-calculate a large number of time dependent hashes for a popular internet portal or proxy service (i.e. AOL), and use them to brute force any existing session from that service.
- Session Obfuscation - The use of a custom method of obscuring data and using it for session management. It is never a sound idea to include client or other confidential information within a session ID. For example, some organizations have even tried encoding the user's name and password within the session ID using a shifted Unicode and hexadecimal representation of the information.

C. Session validity:

For secure applications all session information should be time limited and allow for client-side cancellation or server-side revocation.

- Client Cancellation - Many web applications fail to allow for client-side cancellation such as "log-out". If the intention is to allow users to interact with the application from anywhere, including Internet Cafes, organizations need to be aware that other users can use the same machine and trawl through the "history" and cached page information. If the session has not been cancelled, it is a trivial exercise for the next user of the computer to "resume" the last connection.
- Session Timeout - Again, when dealing with the possibility of shared client computers, it is extremely important that there is a limited lifetime (or period of inactivity) after which the session will automatically expire. The expiry time should be kept to a minimum period, and is dependent upon the nature of the application. Ideally the application should be capable of monitoring the period of inactivity for each session ID and be able to delete or revoke the session ID when a threshold has been reached.
- Server Revocation - In some circumstances it may be necessary to cancel a session at the server- side. Likely events include when the user leaves the insecure part of the application and enters the secure

part with a new session ID. Alternatively, should some kind of attack be recorded by the server, it would be advisable to revoke the session associated with the attackers system.

D. CSRF (Cross-Site Request Forgery):

Attackers can potentially hijack sessions without even knowing session tokens:

Browser always includes cookies in requests to a particular server.

E. Session obsession:

The attackers fixes the token before the victim's authentication. The attack has three steps:

i. Session setup.
ii. Session Fixation.
iii. Session entrance.

i. Session setup:

The attacker creates a session on the server (a trap session) and receives or creates the token. In some cases, the attackers must keep the session alive ("session maintenance") by sending requests at regular intervals.

ii. Session Fixation:

The attacker introduce the token into the victim's browser.

iii. Session entrance:

The attacker wait for the user to the enter the session,

at which time the attacker can also enter.

Fig. Session Fixation

VII. ANAYSIS OF CURRENT VULNERABITIES

The assessment provides recommendations for improvements and revised procedures. Our security specialists have the knowledge of current technologies and the solid experience necessary to design:

- Perimeter protection
- Access control
- Video surveillance
- Intrusion detection
- Counter-terrorism and threat mitigation systems

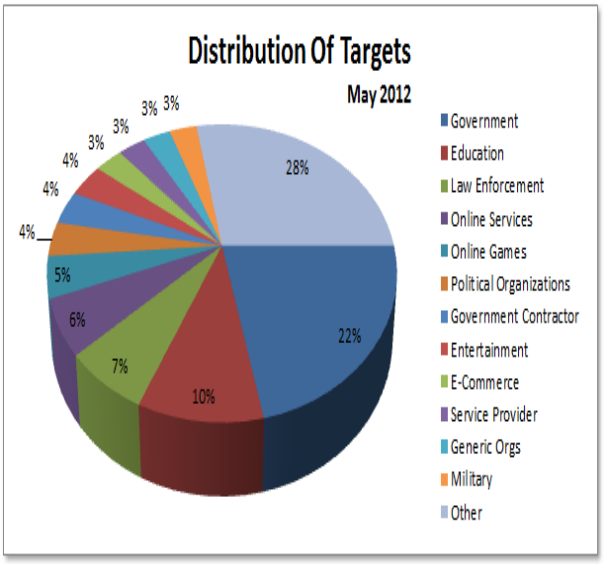

Fig. 1. Survey on current Vulnerabilities

VIII. SOLUTIONS

A. Use Message Authentication Codes (MACs) to validate the sensitive data.

1. MAC function takes arbitrary-length text, secret key, produces a MAC that provides a unique signature for the text.
2. Without knowing the key, cannot generate a valid MAC.
3. Server includes MAC with data sent to the browser.
4. Browser must return both MAC and data.

B. Keep The State On The Server

1. Session could include state from several different pages; must keep all of this information separate.
2. User can display a form, go onto other pages, come back to form, and submit: still need state information.
3. Can't keep forever: results into much session state on server.

X CONCLUSION

The stateless nature of HTTP requires organizations to use their own custom method of managing state through the use of session specific information. While there are a number of ways of implementing a session management solution, there are benefits and restrictions to each implementation. It is vital that developers understand both the mechanisms available to them, as well as the limitations. For applications requiring an application user to authenticate to access resources, it is imperative that the session management process is implemented securely.

XI REFERENCES

[1] http://www.google.com
[2] http://web.stanford.edu/~ouster/cgi-bin/cs142-fall10/lecture.php?topic=secSession.
[3] http://www.technicalinfo.net/papers/WebBasedSessionManagement.html
[4] http://resources.infosecinstitute.com/how-to-prevent-cross-site-scripting-attacks/
[5] http://www.specialresponse.com/security-srvey-vulnerability-assessment.php
[6] http://paulsparrows.files.wordpress.com/2012/06/distribution-attack-techniques-may-2012.png
[7] http://www.owasp.org/images/b/b6/Code_Injection.JPG