A Secure Session Management Based on Threat Modeling

Reem Jafar Ismail*
Departement of Computer Science, University of Technology, Baghdad, Iraq.

Abstract

A session is a period of time linked to a user, which is initiated when he/she arrives at a web application and it ends when his/her browser is closed or after a certain time of inactivity. Attackers can hijack a user's session by exploiting session management vulnerabilities by means of session fixation and cross-site request forgery attacks.

Very often, session IDs are not only identification tokens, but also authenticators. This means that upon login, users are authenticated based on their credentials (e.g., usernames/passwords or digital certificates) and issued session IDs that will effectively serve as temporary static passwords for accessing their sessions. This makes session IDs a very appealing target for attackers. In many cases, an attacker who manages to obtain a valid ID of user's session can use it to directly enter that session – often without arising user's suspicion. A secure session management must be implemented in the development phase of web applications because it is the responsibility of the web application, and not the underlying web server.

Threat modeling is a systematic process that is used to identify threats and vulnerabilities in software and has become popular technique to help system designers think about the security threats that their system might face.

In this paper we design the threat modeling for session's ID threat by using SeaMonster security modeling software, and then propose a secure session management that avoids the vulnerabilities. The proposed secure session management is designed to give trust authentication between the client and the server to avoid session hijacing attack by using both: server session's ID and MAC address of the client.Visual Studio. Net 2008 is used in implementing the proposed system.

Keywords: Session management, Hijacking session, Threat modeling, Attack tree, SeaMonster Security Modeling Software.

نظام أمن لادارة الجلسة بالاعتماد على نموذج التهديدات

ريم جعفر اسماعيل
قسم علوم الحاسوب، الجامعة التكنولوجية، بغداد، العراق

الخلاصة:

عندما يتصفح المستخدم شبكة الانترنت فان هنالك جلسة تستمر لفترة من الوقت تبدأ مع دخول المستخدم الى الموقع وتنتهي هذه الجلسة عندما يغلق المستخدم الموقع. ان هذه الجلسة ستكون معرضة الى الهجوم عن طريق السرقة او التزوير.

حيث ان لكل جلسة رقم معين سيستخدم لتحديدها وكذلك يستخدم للتخويل ايضا، وهذا يعني عند التحقق من المستخدم عن طريق اسم المستخدم او كلمة السر فان الجلسة ستكون محددة لذلك المستخدم المخول. ان

*Email: reemaljanabi@yahoo.com

رقم الجلسة سيكون عرضة للهجوم عن طريق الحصول على هذا الرقم واستخدامه والدخول الى الجلسة بدون ان يشك المستخدم بذلك.

لذا يجب ان يكون هنالك اسلوب امن في ادارة الجلسة يتم تنفيذه في مرحلة تصميم تطبيق الانترنت وليس بالاعتماد على مزود الخدمة فقط. ان نموذج التهديد سوف يحدد جميع التهديدات التي تتعرض لها البرامج مما يساعد المصممين عند التصميم على بناء نظام امن خالي من التهديدات.

في هذا البحث تم تصميم نموذج التهديدات الخاص بالتهديدات المؤثرة على رقم الجلسة باستخدام برنامج SeaMonster لتصميم نماذج امنه، ثم اقتراح نظام امن لادارة الجلسة. ان نظام ادارة الجلسة الامن تم تصميمه لاعطاء ثقة وتخويل بين المستخدم ومزود الخدمة لتجنب الهجوم عن طريق استخدام رقم الجلسة وذلك عن طريق الاعتماد على عنوان جهاز مستخدم الانترنت حيث تم استخدام برنامج 2008 Visual Studio.Net لتنفيذ النظام المقترح.

1. Intoduction:

Security is an important property of any software. Many applications are outsourced, where the application development lacks strong integration of software security. The growing need to address software security measures across development life cycle is important. So Software Development Lifecycle requires transformation at various phases to support application security [1].

The session management mechanism is a fundamental security component in the majority of web applications. It is what enables the application to uniquely identify a given user across a number of different requests and to handle the data that it accumulates about the state of that user's interaction with the application. Where an application implements login functionality, session management is of particular importance, because it is what enables the application to persist its assurance of any given user's identity beyond the request in which he supplies his credentials.

Because of the key role played by session management mechanisms, they are a prime target for malicious attacks against the application. If attackers can break an application's session management, they can effectively bypass its authentication controls and masquerade as other application users without knowing their credentials. If an attacker compromises an administrative user in this way, the attacker can own the entire application [2].

Several models have been used for secure session management; some of the related works are listed below:

- In [3] the authors propose a secure cookie mechanism that implements an intermediary reverse Proxy patterns to ensure users' sessions management and to provide the following security services: source authentication, integrity control and no-replay attacks.
- In [4] the authors propose "One-Time Cookies" (OTC), an HTTP session authentication protocol that is efficient, easy to deploy and resistant to session hijacking. OTC's security relies on the use of disposable credentials based on a modified hash chain construction.
- In [5] the authors present SessionShield, a lightweight client-side protection mechanism against session hijacking that allows users to protect themselves even if a vulnerable website's operator neglects to mitigate existing Cross-site Scripting (XSS) problems. SessionShield is based on the observation that session identifier values are not used by legitimate client- side scripts and, thus, need not to be available to the scripting languages running in the browser. This system requires no training period and imposes negligible overhead to the browser, therefore, making it ideal for desktop and mobile systems.

2. Secure Session Management:

In web applications, sessions are what allow users to use the application while only authenticating themselves once at the beginning of the session. Once a user is authenticated, the application issues a Session ID to ensure that all actions during the session are being performed by the user who originally supplied their authentication information. Attackers often manipulate the Session ID to steal a valid session from an authenticated user. Defense against such attacks include changing a user's Session ID by asking the user to re-authenticate when the session has timed out or when the user attempts to use a functionality that is designated as sensitive. Examples of these attacks and the design principles are listed in Table 1-, [6].

Table 1- Session management solutions

Session Management Issue	How to Avoid It
Attacker guessing the user's Session ID	Session IDs should be created with the same standards as passwords. This means that the Session ID should be of considerable length and complexity. There should not be any noticeable pattern in the Session IDs that could be used to predict the next ID to be issued.
Attacker stealing the user's Session ID	Session IDs, like all sensitive data, should be transmitted by secure means (such as HTTPS) and stored in a secure location (not publically readable).
Attacker setting a user's Session ID (session fixation)	The application should check that all Session IDs being used were originally distributed by the application server.

3. Session Hijacking:

A typical session hijacking is a well-known man-in-the-middle attack in the world of network security and its one of the favorite attack for the attackers because of the nature of the attack. A user who is already logged in (authenticated) to a web server and has a valid session existing between the user and the server, the attacker takes control over such a session, basically hijacks the session from the user and continues the connection to the server pretending to be the user. This has become increasingly common because the attackers are in a great advantage of not having to waste hours and hours to crack a password, or to try and conduct a dictionary attack against the server, since the user has already been authenticated and in active session it makes it so much easier to just listen to the traffic on the network without the knowledge of the user [7].

Table 2- presents the session hijacking vulnerabilities in terms of attack timing, impact duration, and attack target area [8].

Table 2 – Session hijacking attacking features

Feature	How the attacks is done
Timing of Attack	Attacker attacks the user's browser *after* he logs in to the target server.
Impact Duration	Attacker usually gains *one-time* access to the user's session and has to repeat the attack in order to gain access to another one.
Attack Target Area	Communication link, target web server.

Figure 1- shows an example of session hijacking attack, as we can see, first the attacker uses a sniffer to capture a valid token session called "Session ID", then he uses the valid token session to gain unauthorized access to the Web Server.

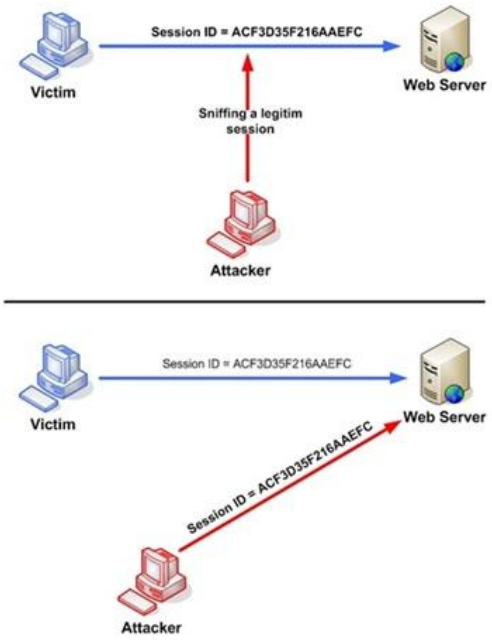

Figure 1 – Session hijacking attack

4. Threat Modeling from the Attacker's Perspective [6]:

A **threat** is a potential occurrence, malicious or otherwise, that might damage or compromise system resources. Threat modeling is a systematic process that is used to identify threats and vulnerabilities in software and has become popular technique to help system designers think about the security threats that their system might face. Therefore, threat modeling can be seen as risk assessment for software development. It enables the designer to develop mitigation strategies for potential vulnerabilities and helps them focus their limited resources and attention on the parts of the system most "at risk". These are many threat models that are used, namely:

4.1 Use case diagrams - describe an application in action. The emphasis is on what a system does rather than how. Use cases can be represented either in text or graphics, and there is no restriction on what the use case diagrams should include or look like.

4.2 Misuse/Abuse Cases - Like use cases, misuse cases require understanding the services that are present in the system. A use case generally describes behavior that the system owner wants the system to implement. Misuse cases apply the concept of a negative scenario—that is, a situation that the system's owner does *not* want to occur.

4.3 Attack Trees – Attack trees provide a formal, methodical technique for describing the security of systems, based on varying attacks. Attacks against a system are represented in a tree structure, with the goal as the root node and the different routes of achieving that goal as the leaf nodes.

SeaMonster is a security modelling tool continuously being developed by an open source community lead by SINTEF. The unique feature of *SeaMonster* is that it supports notations and modelling techniques that security experts and analysers are already familiar with, like: misuse cases and attack trees threat models [10].

5. The Proposed Secure Session Manaemengnt:

The proposed secure session management solve the problem of session hijacking for session ID as follows:

5.1 Design and implementation of session ID threat model

Figure 2- shows the attack tree for session ID threat by using *SeaMonster* Security Modeling Software. The root of the tree represents the name of the threat which is "Session ID threat" and the attacks are represented as nodes in the tree. This threat model is important and useful in our proposed secure session management to identify threats and vulnerabilities.

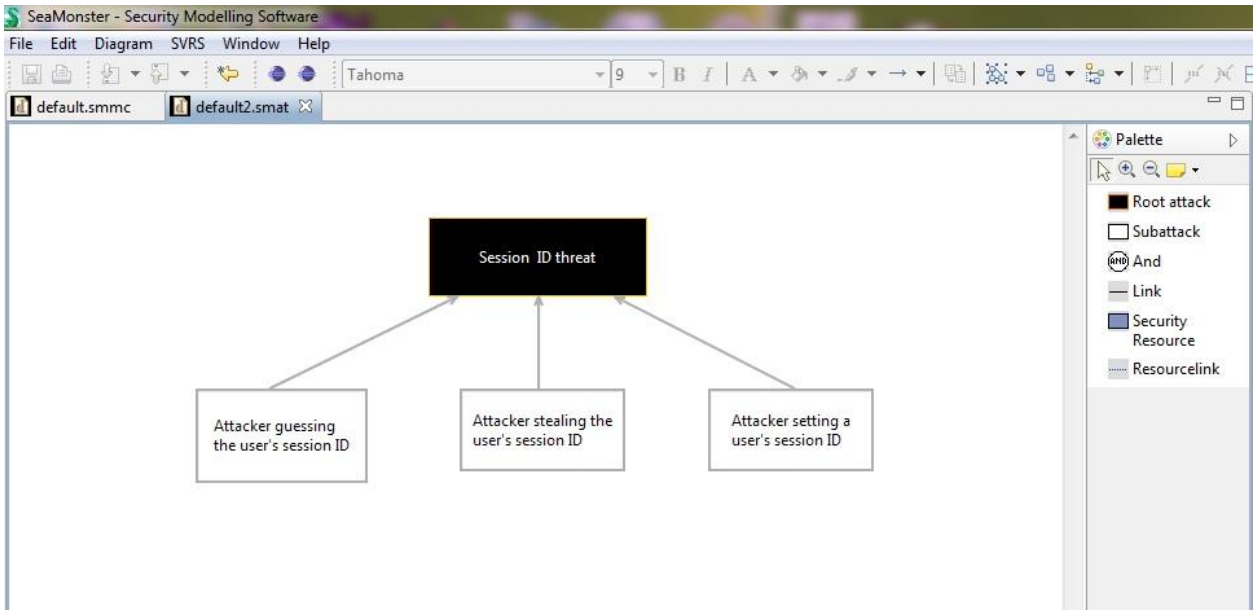

Figure 2 – Attack tree for session ID threat

5.2 Propsed Secure Session Management

As we see in the threat model of figure 2-, the problem is in attacked ***Session ID*** so the proposed secure session management algorithm will not depend only on the user's session ID as an authentcation because it may be attcked by gessing or stealing or setting it as explained in attack tree for session ID threat in figure 2-. So another authentcation must be added to verify the user and make the session ID not changed during the session.

In the proposed algorithm the Media Access Control address (**MAC address)** of the computer that the user has, is used. Since each computer has a uniqe MAC address that is attached to it during manufacturing it, we can use this MAC address in our proposed authentication.

MAC address is a unique code assigned to every piece of hardware that connect to the Internet. When a manufacturer creates a network capable piece of hardware they will assign the MAC address which will usually begin with a

code that is tied to the manufactuurer. The code will be unique to every device, even two devices of the same type. A device's MAC address is composed of six pairs of hexadecimal numbers. The numbers are separated by colons as in the following example:
6E:51:F5:C1:11:00

5.3 Proposed Secure Session Management Algorithm

Step 1: Get the session ID from the server
Step2: The server get the MAC address of the client's computer
Step3: Register the (session ID, client's computer MAC address) on the server as a new client
Step4: Open a session between server and client for a period of time
Step5: If a period of time is finished then
Stop the current session
Step6: Re-get the MAC address of the client's computer
Step7: Check the current MAC address of the client's computer in step6 with the registered one in step3
If the current MAC address <> registered MAC address then
The server discover an attacker
Finish the current session and go to step8
Else
Go to step4
Step8: End

6. Implementing the Proposed Secure Session Manaemengnt:

The proposed secure session management is implemented by using both: Visual Basic.Net 2008 and ASP.Net programming languages, which is both available in Visal Studio.Net 2008.

Code 1: It is implemented in Visual Bacic.Net 2008 to get the host name for the computer, to implement this code creat a "Console Application" in Visual Basic.Net 2008 and add a module to run it. The output of this code will produce the computer name.

Get the MAC address of the client's computer:

```
Imports System.IO

Imports System.Net.Sockets

Module Module1

    Sub Main()
Dim myHost As String = System.Net.Dns.GetHostName

 Dim myPCname As System.Net.IPHostEntry =
        System.Net.Dns.GetHostByName(myHost)

 Console.WriteLine("The name of the host is = " & myPCname.HostName)

    End Sub

End Module
```

Code 2: It is implemented in ASP.Net to manage the session time out, to implement this code create "ASP.Net Web Site" in ASP.Net under Visual Basic. The peroid of time of a session is set to (60) seconds.

Open a session between server and client for a period of time:

```
<%@ Page Language="VB" %>

<script runat="server">

    Sub Page_Load()
        If Not Page.IsPostBack Then
            ResetStartTime()
        End If
    End Sub

    Protected Sub btnAgain_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        ResetStartTime()
    End Sub

    Sub ResetStartTime()
        Session("StartTime") = DateTime.Now
    End Sub
```

```
    Protected Sub valAnswer_ServerValidate(ByVal source As Object, ByVal args As System.Web.UI
.WebControls.ServerValidateEventArgs)
        Dim startTime As DateTime = CType(Session("StartTime"), DateTime)
        If startTime.AddSeconds(60) > DateTime.Now Then
            args.IsValid = True
        Else
            args.IsValid = False
        End If
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Timed Test</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <p>
    You have 60 seconds to work with session
   </p>

    <asp:Label
        id="lblQuestion"
        Text="Get some data form user”
        AssociatedControlID="txtAnswer"
        Runat="server" />
    <br />
    <asp:TextBox
        id="txtAnswer"
        Runat="server" />
    <asp:CustomValidator
        id="valAnswer"
        Text="(You time is finished!)"
        OnServerValidate="valAnswer_ServerValidate"
        Runat="server"  />

    <br /><br />

    <asp:Button
        id="btnSubmit"
        Text="Submit"
        Runat="server" />

    <asp:Button
        id="btnAgain"
        Text="Try Again!"
        CausesValidation="false"
        OnClick="btnAgain_Click"
        Runat="server" />

    </div>
    </form>
</body>
</html>
```

7.Conclusion and Recomendation:

First of all, we need to make it clear that preventing session attacks is mainly the responsibility of the web application, and not the underlying web server.

This paper solves the problem of session hijacking for session ID by desiging a secure session management based on session's ID thread modelling.

The proposed secure session management is designed to give trust authentication between the client and the server to avoid session hijacing attack by using both: server session's ID and MAC address of the client. Once the session is setup between the client and the server, the server will get the MAC address of the client and setup a session for a peroid of time, when this time is finished the current session is stopped to get the MAC address again from the client to check its authentication.

The proposed secure session management algorithm has the following features:

1. Instead of having one long time of session between the cilent and the server, our proposed secure session management algorithm will have many sessions each session will have a specific period of time when this time is finished

the session is stopped to check the client's physical address.

2. The authentication will depend on a specific MAC address of the computer and not on session ID only.

3. The IP Address is not used because it can be changed during session; an attacker may change the original IP Address for the client and use it in his own machine, so MAC address give more authentication in our proposed algorithm.

Finally, it is recommended that all software systems have a threat model developed and documented. Threat models should be created and should be revisited as the system evolves and development progresses.

References:

1. Dharmesh M Mehta. **2010**. Effective Software Security Management. OWASP: Open Web Application Security Project.
2. **Stuttard** D. and Pinto M. . **2011**. The Web Application Hacker's Handbook. Second Edition. John Wiley & Sons Publication. Canada. p: 205.
3. Pujolle. **2009**. Secure Session Management with Cookies. In 7th International Conference on Information, Communications and Signal Processing, 8-10 Decmber. pp:689.
4. Dacosta I. **2012**. One-Time Cookies: Preventing Session Hijacking Attacks with Disposable Credentials. ACM Transactions on Internet Technology (TOIT), 12(1), pp:1.
5. Nikiforakis N. **2011**. SessionShield: Lightweight Protection Against Session Hijacking. In 3rd International Symposium on Engineering Secure Software and Systems (ESSoS '11). pp: 1.
6. Software Assurance Pocket Guide Series. **2012**. Architecture and Design Considerations for Secure Software. Version 2.0, Vol. V.
7. Feng X. and Louis J. **2011**. Detection of Session Hijacking. Msc. Thesis. Department of Computer Security and Technology, University of Bedfordshire. Bedfordshire . UK. p.11.
8. Kolšek M. **2002**. Session Fixation Vulnerability in Web-based Applications. ACROS Security. http://www.acrossecurity.com/papers/session_fixation.pdf
9. OWASP: Open Web Application Security Project. **2011**. Session Hijacking Attack.https://www.owasp.org/index.php/Session_hijacking_attack
10. "SeaMonster 5 Inslallation and User Guide";http://ftp.jaist.ac.jp/pub/sourceforge/s/project/se/seamonster/SeaMonster%205/SeaMonster%20installation%20and%20user%20guide.pdf