Poster: Prevent Session Hijacking

Willem Burgers Roel Verdult Marko van Eekelen

Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands
willemburgers@student.ru.nl, {rverdult,marko}@cs.ru.nl

I. Introduction

Web applications are hard to secure. Many web applications suffer from security vulnerabilities that can be exploited by an attacker. A widely used method to secure web applications involves the creation of an application session for which the user has to authenticate using a registered login name and corresponding password. Before such a session is established, a secure encrypted communication channel is negotiated at a network level to ensure confidentiality. However, the creation of a session and the use of encrypted communication is not sufficient to make an application secure against all attacks.

The focus of our paper [1] is on one of the serious attacks: *session stealing* or *session hijacking*. This is aimed at the session mechanism itself. An adversary takes over a valid user session with a recovered authentication token that is distributed to an genuine user. From this point on we call such a valid authentication token a session identifier (*session ID*). Most modern websites use encrypted communication between the client and the server to prevent an adversary from eavesdropping this session ID. However, it does not prevent stealing the session ID by means of malicious scripts or rogue browser plug-ins.

A user whose session is stolen may not notice anything strange while the attack is performed, since the execution of the script may run in the background without changing anything on the screen of the user. This means that the user can be offered little advice in order to prevent such attacks. The main advice is to avoid surfing to pages hosted on the same domain that could be infected by malicious scripts during an active session. This means always closing an open session before surfing to a website that does not require the same session credentials. Such advice does not help much if the application for which the session is opened, is itself vulnerable to cross site scripting. This is the case for many web applications where data can be entered by users and is to be read by other users. Vulnerabilities can occur if the output, generated from the entered data, is not properly encoded. Output encoding prevents executable scripts by replacing meaningful characters with harmless annotated symbols. For example, when an adversary is able to post a malicious script, it could compromise the complete website and steal all active sessions. In that case an attack can happen directly after a genuine user visits the website only once.

Vulnerabilities like these may greatly reduce the trust of the user in the system. The user feels very insecure since there seems to be no way for the user to prevent such an attack.

A. Our contribution

The contributions of this paper are summarized accordingly:

- A new method of binding the application session to the cryptographic network credentials is presented that effectively prevents hijacking of web sessions.
- A fully functional prototype implementation of the method (for cookies) has been built and released under the royalty free BSD license

II. Related Work

A. Session hijacking prevention

There are several other proposals to prevent session hijacking. Johns (2006) [4] proposes a solution where the cookies in which the session ID is kept are sent from a different subdomain. This way the JavaScript code cannot get the cookie, because it does not fall under the same-origin policy, so the cookie is safe. This does not prevent every type of attack though. With browser hijacking or XSS propagation, session cookies can still be obtained by an attacker. Johns uses URL randomization and one-time URLs to prevent these attacks from being executed. He also writes that these methods are not meant as a complete replacement for input and output validation in the application, but it is an extra layer of protection. This sure is a good way of preventing session hijacking, though it is a lot of hassle to implement. Most of the application needs to be rewritten.

Another method is to run a piece of software on the client computer which intercepts the 'Set-Cookie' header before it is sent to the browser. This way the cookies will never be in the browser at all. This method is proposed by Nikiforakis et al. (2011) [6]. Without much overhead this system will prevent JavaScript code from accessing the cookie information. This still relies on the client side. A secure implementation without memory leaks makes this a good solution. As mentioned in Sect. **??**, this paper is based on the work of Oppliger et al. [7]. They propose to bind the application session to the SSL/TLS session to prevent MITM attacks. To bind the two sessions, they use either a software token (like a client cerficate or a private key) or a hardware token (like a smartcard or dedicated device). This is a safe solution, but it requires the distribution of a pre shared key to the client/user. The same binding idea can be used for session hijacking, but we propose a different binding method.

The only other paper that uses the binding of SSL/TLS Session-Aware User Authentication as a basis is a proposal by Chen et al. [2]. They make use of a two factor authentication method by means of a separate device (3g phone). With this device they bind

the SSL/TLS session to the application session. It requires both client and server side changes.

In Fig. II.1, an overview is given of the modifications that are required to secure sessions with the various proposed methods.

Protection method	Side	Software patches		System changes		
		browser	application	software token	hardware token	server
SessionSafe [4]	Server	no	yes	no	no	no
SessionShield [6]	Client	yes	no	no	no	no
Session-Aware [7]	Server	no	yes	yes[1]	yes[1]	no
TLS-SA + GAA [2]	Both	no	yes[2]	no	yes	yes[2]
SBP	Server	**no**	**no**	**no**	**no**	**no**

[1]The implementation can work with either a software token or a hardware token

[2]Either the server application needs to be modified or install additional software

Fig. II.1: Comparison of patch requirements to prevent session stealing

B. Related attack setups

There are also papers that describe attacks on cookies and sessions. As mentioned in Sect. **??** there exist attacks like Browser Exploit Against SSL/TLS (BEAST) [3] and CRIME to steal cookies. Both attacks are implemented in JavaScript for speed, but can be run on any user level. BEAST and CRIME use known plaintext attacks to guess the unencrypted cookie that is sent over an encrypted SSL connection. The cookie is guessed character by character. This brute force method allows to guess an entire cookie. Where BEAST works only on certain versions of SSL/TLS, CRIME works for any version. CRIME makes use of the compression in SSL/TLS to guess the cookie. The flaws of compression in combination with encryption are already described in a paper by John Kelsey in 2002 [5]. There is a proof of concept for the CRIME attack. With some modifications it can be used to actually capture cookies even though they have the http-only property. This is just another method to get the cookie from the client. Our proposed method also defends against both attacks as depicted in the attacker model. Even though they can steal the cookie. It is still hard to copy the SSL session.

C. Session management

This section describes how the SBP handles a request. The first thing that the SBP server does when it gets a request, is redirect the user to the HTTPS port if the user did not connect on that port already. This will start the SSL/TLS handshake to establish the necessary identifiers. Fig. II.2 shows the SSL negotiation in the first block (lines 0 to 10). All further traffic will go through this SSL connection.

When the SSL connection is made, the application session can be established. The first request sent to the server does not contain a cookie, because the server has not set any cookies yet. The SBP can simply replay the request to the application server. Any request on a page without a session cookie results in a redirect to the login page. When the user logs in, the application server will send a 'Set-Cookie' header. This header is intercepted by SBP and the value of the cookie is encrypted with the key k_c, which is a hash of a secret system key K_p and the SSL master key k concatenated, performed by $k_c \leftarrow hash(K_p||k)$. In our prototype, we use SHA256 as the hashing algorithm. Every encryption with AES-256-cbc (denoted by $\{-\}$) requires a fresh random Initialization Vector (IV) such that an attacker cannot generate multiple session ID values encrypted with the same key and IV. We generate a new random IV for every new 'Set-Cookie' header. The IV is not required to be secret. The cookie is encrypted as follows $\{cookie\}_{k_c} \leftarrow encrypt(cookie, k_c, IV)$. In order to later retrieve the IV, we concatenate it with the encrypted cookie. The encrypted version of the header $\{cookie\}_{k_c}$ and the IV is sent to the client. This process is shown in the second block (lines 11 to 19) in Fig. II.2. From this point on with each request by the user, the client sends the encrypted cookie along with every request. When a request is received with encrypted session data, SBP decrypts the value of the cookie and send the plaintext cookie to the back end server. This can be seen in the final block (lines 20 to 25) of Fig. II.2.

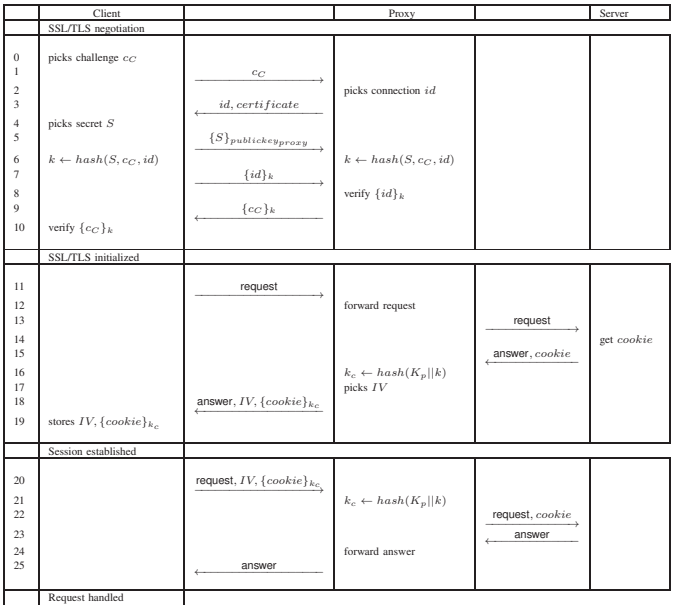

Fig. II.2: Session Binding Proxy protocol

References

[1] Willem Burgers, Roel Verdult, and Marko van Eekelen. Prevent session hijacking by binding the session to the cryptographic network credentials. In *18th Nordic Conference on Secure IT Systems (NordSec 2013)*, volume 8208 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2013.

[2] Chunhua Chen, Chris J. Mitchell, and Shaohua Tang. SSL/TLS session-aware user authentication using a gaa bootstrapped key. In *5th IFIP WG 11.2 international conference on Information security theory and practice: security and privacy of mobile devices in wireless communication (WISTP 2011)*, volume 6633 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 2011.

[3] Thai Duong and Juliano Rizzo. Here come the XOR Ninjas. White paper, Netifera, May 2011.

[4] Martin Johns. SessionSafe: Implementing XSS immune session handling. In *11th European Conference on Research in Computer Security (ESORICS 2006)*, volume 4189 of *Lecture Notes in Computer Science*, pages 444–460. Springer-Verlag, 2006.

[5] John Kelsey. Compression and information leakage of plaintext. In Joan Daemen and Vincent Rijmen, editors, *9th Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 95–102. Springer-Verlag, 2002.

[6] Nick Nikiforakis, Wannes Meert, Yves Younan, Martin Johns, and Wouter Joosen. SessionShield: Lightweight protection against session hijacking. In *3rd International Symposium Engineering Secure Software and Systems (ESSoS 2011)*, volume 6542 of *Lecture Notes in Computer Science*, pages 87–100. Springer-Verlag, 2011.

[7] Rolf Oppliger, Ralf Hauser, and David Basin. SSL/TLS session-aware user authentication – or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12):2238–2246, August 2006.