Ensuring the security of web applications operating on the basis of the SSL/TLS protocol

Pavel Razumov[1], *Larissa* Cherckesova[1], *Elena* Revyakina[1*], *Sergey* Morozov [2], *Dmitry* Medvedev[2], and *Andrei* Lobodenko[2]

[1] Don State Technical University, 344002 Rostov-on-Don, Russia
[2] Institute of Service and Entrepreneurship (branch) DSTU, 346500 Shakhty, Rostov region, Russia

Abstract. SSL/TLS (Secure Socket Layer/Transport Layer Security)-enabled web applications are designed to provide authentication based on a public key certificate, as well as generating a secure session key and traffic privacy based on a symmetric key. Today, a large number of e-commerce applications such as stock trading, banking, shopping and gaming rely on the robustness of the SSL/TLS protocol. Recently, a potential threat known as a Man-in-the-Middle or main-in-the-middle (MITM) attack has been used by attackers to attack SSL/TLS-enabled web applications, especially when users want to connect to an SSL/TLS-enabled web server. SSL/TLS. The current article discusses the Man-in-the-Middle attack threat for SSL/TLS-enabled web applications. The existing solution space for countering a MITM attack on SSL/TLS-enabled applications is also considered, and an effective solution is proposed that can resist a MITM attack on SSL/TLS-enabled applications. The proposed solution uses a soft token approach for user authentication in addition to SSL/TLS security features. The proposed solution is claimed to be safe, effective and user-friendly compared to similar approaches.

1 Introduction

The Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol [1] is typically integrated into an application to secure data sent over the HTTP protocol between a client and a server, also known as HTTP over TLS (HTTPS). In an SSL/TLS-enabled application, the client initially sends a hello message to the server, and then the server, after confirming the negotiated parameters, sends the server hello message along with the server's digital certificate to the client. The server's digital certificate provides information about the server's public key, the certificate's validity period, and information about the owner and issuer. Once the client authenticates to the server with the server's certificate, the client and server establish session keys, which are symmetric keys used to encrypt and decrypt information exchanged during an SSL/TLS session, and to verify message integrity. As a result, an SSL/TLS-enabled application is designed to provide server authentication based on digital certificates, key exchange based on a public key, and data confidentiality based on a session key using a

* Corresponding author: revyelena@yandex.ru

standard symmetric key encryption algorithm (eg, AES). In addition, message integrity is checked using message authentication codes, as shown in Figure 1.

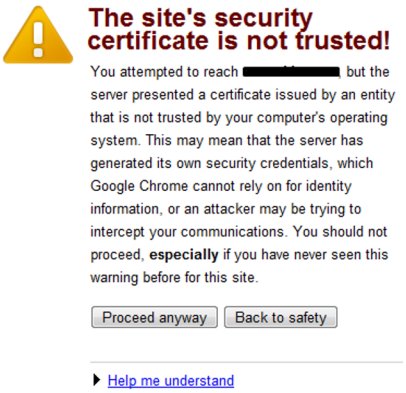

Fig. 1. Certificate warning to the client in an SSL/TLS-enabled application

The SSL/TLS protocol consists of four subprotocols - Handshake, Change Cipher Spec, Record, and Alert [2]. The handshake subprotocol (Handshake) allows the client to authenticate the communicating server using the server's public key certificate (note that client authentication is an additional security support offered by the protocol that can also be achieved using the client certificate), key exchange between client and server using the algorithm public key with subsequent generation of key material and data confidentiality for traffic security using a symmetric key encryption algorithm.

The way the Handshake protocol works is that a client that wants to connect to an SSL/TLS-enabled web application must provide the server with a list of ciphers and algorithms (known as cipher suites) that the client can support. Upon receiving the client's request, the server selects the appropriate cipher and required algorithms from the client's cipher suites. If the server does not agree on a common cipher suite from the list provided by the client, then both client and server must negotiate further on an acceptable cipher suite. Once the server selects a suitable cipher suite from the client's list, the server sends its public key certificate along with the selected cipher suite to the client, whereupon the client verifies the server's certificate, and if the certificate is valid, the server is authenticated. Note that client authentication can also be verified by the server, but this is not a required step. If the server wants to verify the client's authentication, it requests the client's certificate. After confirming server authentication, the client and server establish a master secret using a key exchange algorithm based on RSA, Diffie-Hellman, or Fortezza. With this master key, both client and server generate key material (e.g., encryption key, MAC key) using a pseudo-random function (PRF) and other ephemeral parameters when using the SSL protocol, and for the TLS protocol, it uses HMAC and PRF to generate the key. material. The handshake subprotocol ends with client and server message authentication codes for all previously exchanged messages to protect the handshake from message forgery. The Record subprotocol protects application data using keys (MAC keys and encryption keys) computed in the Handshake subprotocol.

Essentially, the Record subprotocol is responsible for securing application data in the current session. The Alert subprotocolis called when any error or warning occurs while executing other subprotocols.

Informally, not all SSL/TLS-enabled applications require client authentication. This optional client authentication in an SSL/TLS-enabled application opens up a potential security loophole for attackers [3] who can try to convince the client with a fake server

certificate, and if they succeed, they can steal all the sensitive data from the client. This type of attack typically puts an SSL/TLS enabled connection into some vulnerable state by spoofing security indicators and tricking the user into believing the connection has been made to a legitimate server, but in fact the client is connected to the attacker's server.

For example, suppose an attacker creates a fake server certificate instead of a legitimate server certificate, and the client accepts the fake certificate without verifying that it is correct. After that, when the client wants to connect to the same legitimate server (for which the client has already accepted a fake certificate), the client will be redirected to the attacker's server. This type of attack is known as MITM attacks against SSL/TLS-enabled applications. Recently, SSL Stripping [4] attacks also pose a serious threat to breaking the SSL/TLS security for web applications. In an SSL stripping attack, the user believes that an SSL/TLS connection has been established with the target server, while the attacker has the ability to view the user's traffic in the clear. The attacker suppresses the SSL/TLS peer negotiation messages and provides the user with a "cut" version of the requested website, tricking the user into communicating with the server through an insecure channel.Current work discusses existing solutions to counter MITM attacks on SSL/TLS-enabled applications. Because most solutions consider client authentication to be an important factor, the solution space for passwords, GUI, and token-based approaches for user authentication is discussed. Moreover, it appears to be an effective solution using soft token-based client authentication that can protect an SSL/TLS-enabled application from MITM attacks. An analysis of the proposed solution is also carried out for its functional security over the security of the SSL/TLS protocol and ensuring its effectiveness in comparison with other existing approaches.

The rest of the article is organized as follows. Section 1 discusses the SSL/TLS protocol, discusses MITM attacks against SSL/TLS-enabled applications, and describes existing solutions for MITM threats in SSL/TLS-enabled transactions. Section 2 presents a proposed solution for protecting SSL/TLS-enabled applications from MITM attacks. Section 3 analyzes the proposed solution and compares the proposed solution with related approaches. The work in section 5 is completed, after which its results are summed up.

2 MITM attacks on SSL/TLS applications

A MITM attack is a form of active attack in which an attacker intercepts and selectively modifies the intercepted data in order to impersonate the legitimate party involved in the communication between the client and the server. Based on the intended service and business perspective, some SSL/TLS-enabled web applications do not use client authentication as a requirement, instead, web applications enable SSL/TLS in server authentication mode.

The reason for this is that the server provides services as long as the client pays the server the required amount (for example, online purchases made on various services), so client authentication is not required for such scenarios. Therefore, it is the client's responsibility to verify that the interaction is with the correct one. In the SSL/TLS protocol, when obtaining the certificate of the communicating server, the client may receive a warning if any of the following events occur:

1. The root certification authority (CA) is not recognized (not trusted) by the client;
2. The certificate is invalid or has expired;
3. The common name (CN) of the certificate does not match the domain name server.

In such cases, the client should check for a possible cause and terminate the connection unless convinced by a warning message box. It is noticed that in most cases the client does not pay attention to such an important warning, accepts the exception and saves the server certificate in the specified location. Once the client stores the fake certificate in its memory, the damage is already done, as that client will continue to interact with the fake server, despite the connection being secured by SSL/TLS. This type of situation opens the door for a MITM

attacker in an SSL/TLS enabled application. Another potential scenario is attacks [5] on SSL-enabled web applications. SSL stripping attacks assume that the attacker is between the victim and the server.

Many websites provide a secure connection between client and server using HTTPS to protect user credentials, sensitive information such as login form, password change, money transaction, etc.

Although the server's response comes over HTTPS, the attacker modifies <a href = "https://…"> to <a href = "http://…"> before the victim gets the target web page. The victim believes that the credentials are being sent over a secure connection (HTTPS) to the server, but the attacker obtains the information by terminating the connection over HTTP instead of HTTPS. It is also important for an attacker to keep a list of all the HTTP replacements he has made so that he can send an HTTPS request back to the communicating server. Finally, the MITM attacker leaves the client in a vulnerable state and steals useful parameters from the client. Despite this, modern browsers support many indicators (such as the padlock indicator) to provide information about the connection, but the client usually ignores such indicators and falls into a trap. In some cases, an attacker can manipulate these indicators without leaving any information to the user.

In [5], the authors proposed a tamper-proof TrustBar in the browser to visualize certificate information. In [6], the concept of synchronized random boundaries was proposed to distinguish between authentic parts of the browser GUI and rendered content received by the server. Another approach that requires fewer changes to the browser is the use of a dynamic security shell [7], where the browser has a personalized area that the user must customize as desired. It is believed that an attacker cannot forge a personalized area without knowing the exact specification of the area. Next, some user authentication approaches will be discussed beyond the security strengths of the SSL/TLS protocol to mitigate MITM attacks against SSL/TLS-enabled web applications.

2.1 Method using a secret code (password)

Password-based user authentication is widely used in many real world applications. [8] proposed a protocol for binding an SSL/TLS session to a client using a user's password. Protocol [9] binds HTTP authentication over SSL/TLS in server authentication mode. Because HTTP authentication does not encrypt its channel and the client does not authenticate the server, binding client authentication over SSL/TLS in server authentication mode can make communication secure. An interesting feature of this protocol is that it binds user authentication without changing the SSL/TLS, HTTP, and HTTPS protocols.

The protocol works as follows. The client and server first establish an SSL/TLS connection in server authentication mode. The server then authenticates the user with HTTP authentication at random time. However, the server can control when user's password is received. This is more convenient feature than fixed authentication time in SSL/TLS Handshake. To achieve these goals, protocol consists of the following steps:

1. Install SSL/TLS Handshake in server authentication mode.
   A. Client-server: clientgreeting.
   B. Server-Client: Greeting the server and providing the server's public key certificate (cert).
   C. Client-Server: $\{transmission\ of\ \ the\ pre-master\ secret\}_{cert}$.
   D. Client-Server: $\{MAC_{K_m} for\ (A-B)\}_{K_e}$.
   E. Server-Client: $\{MAC_{K_m}\ for\ (A-\mathrm{C})\}_{K_e}$.
2. HTTP user authentication over SSL/TLS channel.
   F. Client-Server: $\{Pagerequest\}_{K_e}$.

G. Server-Client: $\{User\ authorization\ request\}_{K_e}$.

3. Exchange and validate user credentials for binding authentication.
   H. Client-Server:
   $\{user\_id, h(CN, cert, h(user\_id, password, K_m))\}_{K_e}$.
   I. Server-Client:
   $\{CN, h(user_{id}, password, h(CN, cert, K_m)), requested\ page\}_{K_e}$

Symbols CN, MAC, K_m, $K_e$and h(.) indicate the common name in the server certificate, the message authentication codes, the MAC key, the encryption key, and the hash function, respectively. Messages A - E are the SSL/TLS handshake protocol.Using the Handshake protocol, the client and server establish a master secret key, with which they can obtain a MAC key (K_m) and an encryption key (K_e). Messages F and G include user authentication steps using HTTP and the encryption key K_e. Messages H and I contain the user's password.

The server computes $h(CN, cert, h(user\_id, password, K_m))$ locally and compares it with the one received in H, while the client computes $h(user\_id, password, h(CN, cert, K_m))$ and compares it with the one received in H I. If they successfully verify the received data, then they consider that the connection is authenticated mutually. In step (3), the protocol [10] can bind the client authentication using the user's password, thus avoiding a MITM attack on the SSL/TLS session in server authentication mode.

2.2 GUI based approach

Authentication mechanisms such as PassFaces and PASSpicture use the natural human ability to recognize images.The PASSpicture approach replaces the plain text password with a sequence of image clicks.

There are other options that work in a similar way (like clicking on an image or drawing a hidden picture). Instead of using only a text password, a picture password can use bi-directional authentication, in which the server authenticates itself to the user by displaying the user's picture password that was selected by the user during registration. Once confirmed, the user is authenticated to the server using some sort of challenge-response mechanism. Although the GUI-based approach provides some resistance to MITM attacks, this mechanism cannot be robust after an attacker intercepts the user's picture password.

2.3 One Time Password Approach

There are many One Time Password (OTP) mechanisms such as manual OTP, automatic OTP, synchronous OTP, and asynchronous OTP. These mechanisms are briefly explained as follows.

1. A guide to using a one-time password (for example, a scratch card): a sheet of paper or a card containing a one-time password. The OTP must be securely printed and mailed to the customer.
2. Using OTP, the user can log in and after that OTP is irrelevant. In other words, there is nothing an attacker can do with knowing the one-time password once it has been used, because the user is going to use a different one-time password next time.
3. Automatic OTP (eg SMS): Instead of erasing the OTP from the card, the OTP is sent to the registered user's mobile phone/PDA via Short Message Service (SMS). Many banking applications use this concept to protect their applications from phishing.
4. Synchronous OTP (e.g. hardware/software token): Timed OTP is similar to SecurID, a hardware/software token, where each user is provided with a personalized token that

generates an OTP every 30 or 60 seconds. The clock of the token must be synchronized with the clock of the server.

5. Asynchronous OTP: In this case, the authentication server generates a random request and sends it to the user. The user enters a challenge into their token, which in turn generates a result based on the challenge and some seed/secret value stored in the token's memory (which is known to the authentication server). The user then sends the result back to the authentication server. Here, the challenge is valid for a short time determined by the server, and only the authentication server needs to keep track of the validity period. The advantage of asynchronous OTP is that the clocks of the user token and the authentication server do not need to be synchronized.

Also note that manual user authentication based on OTP will make it convenient for the user to access the server frequently. Automatic OTP is secure compared to manual OTP, but reliance on a third party (such as a mobile phone service provider) would be a vulnerable factor in this mechanism. Token-based OTP for user authentication is a secure mechanism, but synchronization is again a weak point.

2.4 Token Based Approach

Token-based user authentication uses a two-factor authentication mechanism. The user enters their PIN and the token generates a 6-digit or 8-digit random code that the user enters into the authentication server used for the intended application.

The authentication server calculates the code on its own and checks whether the user's code matches his code or not. Both the user and the server must be within the same clock deviation in order to accept the code generated by the user token. RSA SecurID is a standard reference for token-based two-factor authentication that has gained wide acceptance in industry and the public sector. You can also integrate token-based user authentication into an SSL-VPN-enabled application for strong authentication. Since the generated code token is random, MITM cannot use it for the next run, re-passing it to the server, which will be discarded by the server. Some SSL/TLS session-aware user authentication implementations have been proposed in Oppliger et al., 2006, Oppliger et al., 2008, which provide the following two methods.

1. Implementation based on hardware tokens: This approach uses impersonal tokens (that is, tokens independent of the user) with secret keys specific to the token. These impersonal tokens are used to authenticate the user against the ACE server. The token has a small display where the user enters their PIN and generates a code. The code is then used to authenticate the user. While this approach provides protection against MITM, using such a token-based implementation adds complexity and also comes at an additional cost.
2. Implementation based on software tokens: In this approach, the concept of a hardware token is converted into a software token by emulating its functionality in software. The software token solution is cheaper than the hardware token solution, but it is subject to additional security risks such as keylogger attacks and visual spoofing.

3 Suggested protocol

Current work presents a soft token-based solution to mitigate the MITM threat in SSL/TLS-enabled web applications. The following parties are involved in the protocol under consideration:

— a user who wants to access applications with SSL/TLS support;
— a server hosting an application with SSL/TLS support.

Before accessing the required application, the user must go through a secure registration process on the server, after which he transfers his identification data (UID), password (PWD) and code template (PC) to the server. The template code refers to an additional security measure added to the proposed solution. The template code is a form or sequence of matrix elements A (i, j) of the template matrix [11]. Each cell of the template matrix is an image representing a character. During the registration process, the user is presented with a randomly generated pattern matrix and needs to select some sequence of data by entering the characters present in the pattern matrix, which then becomes the user's pattern code. On fig. Figure 2 shows the user's PC captured in the shaded area (Note that the user's PC sequence is labeled 1 to 7 in the shaded area), that is, the PC in Figure 2 is B{# (S7&. The protocol works as follows. When the user wants to access the server, the client and server establish an SSL/TLS session key K using the SSL/TLS handshake protocol.The server then generates a soft token containing the PC, UID, and AC authentication code, where AC = MAC(PC; h(all previous SSL/TLS messages)) and MAC(.) is the message authentication code.

The software token provides the user with visual character-by-character feedback from the PC after the user enters AC. The browser will also have a small display that will show the compressed hash value of all previous SSL/TLS messages. This area can be customized by the user to avoid any visual spoofing attack. The user generates an AC using the hash value displayed in the browser and their PC code template. Next, the user sends the UID and AC to the server. Upon receiving the AC, the server also calculates the AC using the hash value and the PC corresponding to the UID. If the computed AC matches the received AC, then the client is authenticated by the server. The user can stop entering the AS if he does not see the template code corresponding to the selected template during registration. The process of injecting into an AC and displaying a PC is shown in Figure 3. Each character of an AC is associated with an SSL/TLS session, so an attacker cannot use the same message in another SSL/TLS session. The protocol in question is shownbelow.

Client - Server:greeting.
Server - Client:greeting the server and providing the server's public key certificate (cert).
Client - Server:$\{transmissionofthepre-mastersecret\}_{cert}$.
Client - Server:$MAC_{K_m}$for $(1-3)$.
Server - Client:MAC_{K_m} for $(1-4)$.
Client - Server:$\{datarequest\}_{cert}$.
Server - Client:
$\{user\ authorization\ token\ request\}_{cert}$.
Client - Server:$\{user_{id}, AC\}_{K_e}$.
Server - Client:$\{requesteddata\}_{cert}$.

K	t	^	(4	U	?	@
%	`	#3	&	S5	r	k
g	{2	|	~	5	76	(
B1	*	Q	/	t	u	&7
z	x	J	$	N	m	H
<	,	>	“	a	d	F
b	L	!	P	v	[	]

Fig. 2. Template matrix and template code.

K	t	^	(4	U	?	@
%	`	#3	&	S5	r	k
g	{2	|	~	5	76	(
B1	*	Q	/	t	u	&7
z	x	J	$	N	m	H
<	,	>	“	a	d	F
b	L	!	P	v	[	]
UID: tiger						
AC: ********						
PC: B{#(S7&						

Fig. 3. Template Matrix and Authentication Code.

Messages 8 and 9 appear as many times as the number of times the server sends a call to the client regarding the requested data.

4 Analysis

4.1 Security analysis

The SSL (version 3.0)/TLS (version 1.0 or later) protocol provides server authentication, session key generation, and privacy protection services. In addition to these security services, the proposed protocol provides an additional security measure to validate a user's authentication code using a template matrix.

The pattern matrix was selected by the user during the registration process and is securely stored on the server. When a user wants to connect to a server, he must select the appropriate template code from the displayed template matrix on the legitimate server. Without knowing the pattern matrix chosen by the user, the attacker cannot provide the correct pattern matrix to the user/client, which precludes MITM attacks on SSL/TLS-enabled applications.

4.1.1 Server authentication

In the SSL/TLS server authentication mode, the client obtains the server's certificate via a "Hello server" message. Upon seeing the server's certificate, the client can check its validity (by checking the issuer, subject, expiration date, chain of trust, purpose of the certificate), whether it is communicating with the correct server or not. However, validation of the server certificate by an inexperienced user is a vulnerable factor. Moreover, if the client (i.e. browser) accepts (or somehow stores) a fake server (attacking) certificate, then the user's password will be passed to the attacker, since the user will enter his password after establishing an SSL / TLS connection with the attacker's server, assuming it is communicating with the correct server. Below, we will show how the proposed protocol can counter this threat by authenticating the client against information stored on the server about the client [9].

4.1.2 Client authentication

Many SSL/TLS-enabled applications do not use public key certificate-based client authentication. The reason is that such applications do not require client authentication. Other possible reasons are:

— the use of a public key certificate for client authentication increases computational costs;

— the client must purchase a certificate from a trusted authority;
— portability issues for the client certificate, that is, if the client wants to access the server from different locations through multiple computers or browsers, then the client must have its own certificate on the device.

Further, if the server enforces client authentication as a requirement, instead of using certificate-based client authentication, a soft token approach can be used. The proposed solution is based on a soft token to authenticate the client based on client information stored on the server, thus avoiding a MITM attack on SSL/TLS-enabled applications, as described below.

Let's assume that an attacker is trying to stage a MITM attack by impersonating a user who wants to access a server with SSL/TLS enabled. We assume that an attacker establishes an SSL/TLS session with a user and another SSL/TLS session with a server hosting an SSL/TLS-enabled application. In a scenario between a user and an attacker, the attacker provides the user with a false token and asks for the UID and AC. The attacker then forwards the UID to the server to spoof the user. In this situation, the server will terminate the session because the server can determine that the user-supplied AC does not belong to the same SSL/TLS session because the server sees a different hash of the SSL/TLS messages. In addition, the user does not reveal their credentials at any stage. Therefore, an attacker cannot obtain the PC user template code to impersonate the user [10].

The scenario is shown in Figure 4 (K1 and K2 are SSL/TLS session keys), where the attacker will not be able to retrieve the data because two different AC values for two different sessions are performed by the attacker when interacting with the user and the server.

Even if the attacker sends a fake server certificate to the client, and if the client accepts it, the attacker cannot get anything from the client because the attacker has no knowledge of the user's AC stored on the server. Therefore, MITM attacks cannot be relevant in the proposed solution[11].

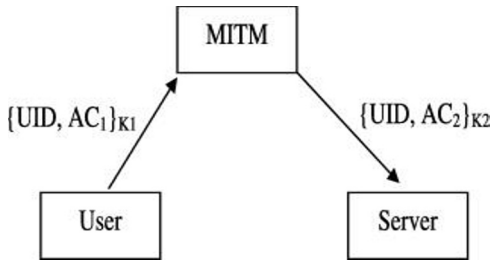

Fig. 4. Protection against MITM attacks on applications with SSL/TLS support.

4.1.3 Attack repeat

The session key of the proposed protocol is generated based on the pre-secret key, client and server random numbers, and other session-specific parameters. If an attacker intercepts messages from the current or previous protocol run and tries to replay any messages in a new protocol run in order to gain access to the server, then the attacker will not be successful because the server will reject the request as original since the messages will not transmit up-to-date data. The use of client and server random numbers in a session helps to counter a replay attack. Therefore, theproposedprotocol is immune to replayattacks.

4.1.4 Password guessing attack

In a conventional password-based user authentication protocol, an attacker tries to guess the user's password by intercepting messages between the user and the server. Once the attacker correctly guesses the user's password, he will impersonate the user in order to gain access to the server. In the proposed protocol, the user's password is not transmitted to the server. Instead, the user generates an AC authentication code using a template matrix selected by the user during registration with the server. This avoids password guessing attacks in the proposed protocol. In other words, the proposed protocol prevents a password guessing attack.

4.2 Comparison of approaches

We give a comparison of all the approaches we discussed in the article in Table 1. It can be seen from the table that the proposed soft token approach provides the optimal solution in terms of security and efficiency.

Table 1. Comparison of approaches.

Approach	Resistance to MITM	Requirement for computing ability	Ease of use by the end user
Based on password	Yes	Low	Yes
Based on visual interface	No	High	Yes
Based on one-time password	Yes	High	No
Based on hardware token	Yes	High	No
Based on suggested software token	Yes	Low	Yes

The suggested solution also applies. The user must enter the AC, after which he will be able to see the PC selected during registration. The approach does not require the registered user to carry any device or token while they move from one location to another. The only thing it needs to remember while connecting to the server is the PC's own password [12,13].

5 Conclusion

In the current work, the issue of the threat of a MITM attack for SSL/TLS-enabled web applications has been raised. Existing approaches to user authentication were considered and user authentication based on software tokens was proposed. By enabling the proposed solution in an SSL/TLS-enabled web application, MITM attacks can be avoided. The proposed solution requires a trusted rendering (managed by a legitimate server) in the browser to display the compressed hash value of the SSL/TLS handshake messages. The proposed solution is computationally efficient and provides additional security in addition to the reliability of the SSL/TLS protocol.

References

1. P. Burkhold, (2002) Man-in-the-Middle SSL Attacks. SANS Information Security Institute
2. R. Dhamija, J. D. Tygar, Phishing Fight: Dynamic Security Skins, Symposium on Useful Privacy and Security (ACM Press, 2005)

3. T. Dirks, E. Rescorla, Transport layer security protocol. Network Working Group, RFC 5246 (2008)
4. T.R. Kumar, S.V. Raghavan, PassPattern System (PPS): A template-based user authentication scheme. IFIP-TC6 International Networking Conference on AdHoc and Sensor Networks, Wireless Networks, Next Generation Internet, ACM Press 162-169 (2008)
5. T.Yu. Lee, Y. Wu, Study. International Conference on Applied Cryptography and Network Security (ACNS), LNCS **2846,** 241–253 (2003)
6. R. Oppliger, S. Gaek, Media Communications and Security pp. 32–41 (2005)
7. A.D. Rubin, USENIX Association for Computing Systems **9,** 15–27 (1996)
8. T. Saito, K. Sekiguchi, R. Hatsugai, LNCS **5186,** 252–262 (2008)
9. D. Shin, R. Lopez, Study. Conference on Computer Security Applications (ACSAC 2011), ACM, pp. 287–296 (2011)
10. Z.E. Ye, S. Smith, Study. USENIX Security Symposium pp. 263–279 (2002)
11. X. Zhao, D. Wang, X. Zhao, W. Yang, X. Ma, International Conference on Information and Communications Security (ICICS'12), LNCS **7618,** 365–372 (2012)
12. L.V. Cherckesova et al., Electronics **11(23),** 3954 (2022)
13. Y. Kovtun et al., *Methodology for Neural Networks Training at Analyzing the Context of Event at Emotions Recognizing, in* Robotics, Machinery and Engineering Technology for Precision Agriculture: Proceedings of XIV International Scientific Conference “INTERAGROMASH 2021”, Springer Singapore, 2022