Computers & Security (2005) **24**, 76—82

Computers & Security

www.elsevier.com/locate/cose

Improvements on the WTLS protocol to avoid denial of service attacks

Ruishan Zhang*, **Kefei Chen**

Department of Computer Science and Engineering, Shanghai Jiaotong University, 1954 Hua Shan Road, Shanghai 200030, PR China

Received 7 November 2003; revised 12 September 2004; accepted 7 October 2004
Available online 28 January 2005

KEYWORDS
WTLS;
WAP;
DoS attacks;
Client cookies;
Client puzzles

Abstract The current WTLS protocol is closely modeled after the well-studied SSL protocol. However, since some differences exist between these two protocols, even if the SSL protocol is secure, the WTLS protocol may not.

We propose three kinds of possible Denial of Service (DoS) attacks on the existing WTLS protocol, which can be categorized into two types: memory exhaustion attacks and CPU exhaustion attacks. The first and the second kinds of attacks belong to memory exhaustion attacks, and the third kind of attack is a CPU exhaustion attack.

Not only wireless network clients but also Internet clients can launch these three kinds of attacks, which are very simple and effective. Since Internet clients are more powerful in network bandwidth and CPU resources, damages made by these attackers are more serious, which can even make the WTLS server stop providing services for legitimate clients.

Client cookies, client puzzles and an application timer is used to improve the current protocol, and our improvements are secure against such attacks.

Introduction

The primary goal of the WTLS protocol is to provide privacy, data integrity and authentication between two communicating applications. The current WTLS protocol (WAP Forum, 2001) is closely modeled after the well-studied Secure Socket Layer (SSL) protocol (Freier et al., 1996). However, some differences exist between these two protocols. The SSL protocol operates on the top of the reliable TCP protocol, whereas the WTLS protocol runs on the top of unreliable connectionless Wireless Datagram Protocol (WDP). So even if the SSL protocol is secure, the WTLS protocol may not.

In this paper, we propose three kinds of DoS attacks on the WTLS protocol. All of them are easy

* Corresponding author.
E-mail addresses: zhang-rs@cs.sjtu.edu.cn (R. Zhang), chen-kf@cs.sjtu.edu.cn (K. Chen).

0167-4048/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2004.10.002

to perform. However, the damages are serious. For example, though based on the unreliable connectionless WDP protocol, the WTLS protocol only specifies the timeout timer and the retransmission timer for the client and doesn't specify any timer for the server, which makes the protocol susceptible to DoS attacks, and exhausts the server's memory. To launch such an attack, an attacker just needs to use a forged IP address and sends some messages. Even in such a simple way, the attacker can exhaust the server's memory and make it fail to provide services for legitimate clients.

DoS attacks on protocols and corresponding resistant methods in wired networks are not new topics. The SYN flooding attack against the TCP connection protocol on the Internet was described, e.g., in CERT Coordination Center (1996). The possible remedies were analyzed in detailed in Inc. Berkeley Software Design (1996) and Cox (1996). Some general techniques, such as the two-phase authentication, client puzzles and client cookies, to resist DoS attacks on protocols were presented and studied (Karn and Simpson, 1999; Aura et al., 2000). For example, Cookies have been preciously used in the Photuris protocol (Karn and Simpson, 1999) and in the IKE protocol. Client puzzles were used to resist DoS attack in Aura et al. (2000). Furthermore, some DoS-resistant protocols were designed (Karn and Simpson, 1999; Aiello et al., 2002). Meadows (1999) formalized the idea of gradually strengthening authentication. So far, the basic principles are well studied and become clearer (Karn and Simpson, 1999; Aura et al., 2000; Aiello et al., 2002). However, no one proposed the method of DoS attacks on the WTLS protocol until now.

So in this paper, we firstly present three possible kinds of DoS attacks on the current WTLS protocol. And then, client cookies and client puzzles mentioned above are applied to improve the current protocol.

The remainder of this paper is organized as follows. In the next section, we review the WTLS protocol. Then, three kinds of DoS attacks on the WTLS protocol are proposed which is followed by three improvements on the current WTLS protocol. Further, we analyze the improved protocol. Finally, a conclusion is given in last section.

The WTLS protocol

The WTLS protocol is similar to the SSL protocol. When the WTLS client wants to establish a secure connection with the WTLS server, the client and the server need to exchange a sequence of messages, which is called a handshake. There are three types of handshakes: a full handshake, an optimized handshake and an abbreviated handshake. For simplicity, we just introduce the full handshake here.

There are five steps in a full handshake, as are depicted in Fig. 1. In the first step, the client sends a ClientHello message. In the second step, the server sends ServerHello, Certificate*, ServerKeyExchange*, CertificateRequest* and ServerHelloDone messages to the client. In the third step, the client sends Certificate*, ClientKeyExchange*, CertificateVerify*, [ChangeCipherSpec] and Finished messages. In the fourth step, the server sends [ChangeCipherSpec] and Finished messages. In this step, the handshake is complete and the secure connection is established. In the fifth step, the client and the server begin to exchange application layer data under the negotiated secure connection. In Fig. 1, "*" indicates optional messages that are not always sent, and "[]" means that the ChangeCipherSpec message is not a part of the handshake protocol.

DoS attacks

Before we describe DoS attacks, we should first point out that not only a wireless WAP client but also an Internet client have the ability to be an

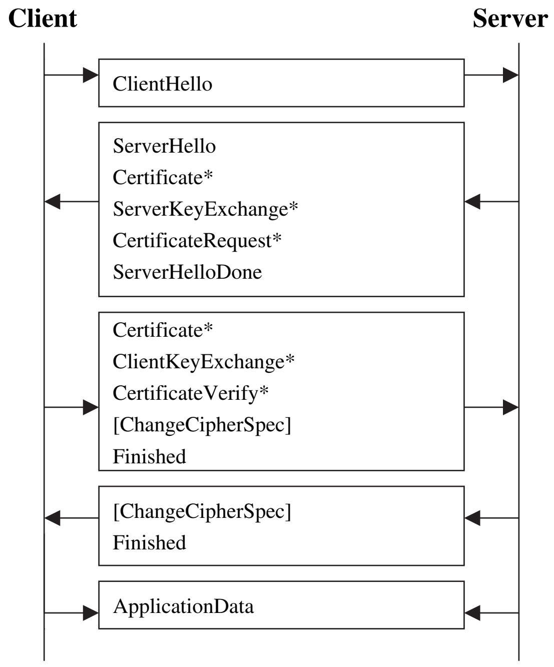

Figure 1 A full handshake.

attacker. To clarify this, let’s look into the whole picture of the WAP protocol and its interaction with the wireless network (for simplicity, we omit its interaction with the Web server). The WDP protocol operates over various bearer services. In order to explain DoS attacks on the WTLS protocol, we give a specific example with Global System for Mobile Communication/Circuit Switched Data (GSM/CSD) as the bearer layer. Fig. 2 illustrates the protocol profile for the WDP layer when operating over a circuit-switched data connection. The Inter Working Function (IWF) provides non-transparent CSD services and is not present in transparent circuit data calls. The Remote Access Server (RAS) or the Internet Service Provider (ISP) provides connectivity to the Internet network so that the mobile and WAP proxy server can address each other. The WAP proxy/server can terminate the WAE or serve as a proxy to other applications on the Internet. Note that the WAP server is a part of the wired Internet, instead of the wireless network. Thus not only a WAP client can connect to the WAP server by the ISP, but also an Internet client, such as a desktop computer. Therefore, to carry out DoS attacks, an Internet client is enough, and no wireless client is needed. The case for the WAP protocol over General Packet Radio Service (GPRS) is similar. For simplicity, we won’t discuss that case here. An Internet client is more powerful and convenient than a wireless client, in both network bandwidth and computation power. As a result, DoS attack damages made by Internet clients are more serious, and must be avoided. So we will just discuss Internet attackers in the following part.

Now we begin to describe three kinds of DoS attacks, which can be classified into two categories: memory exhaustion attacks and CPU exhaustion attacks.

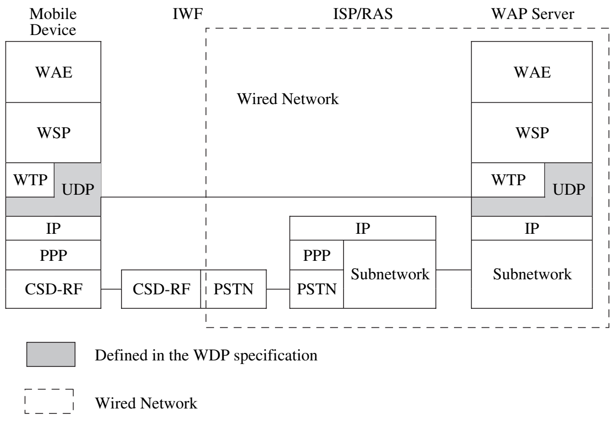

Figure 2 The whole picture of the WAP protocol and its interaction with the wireless network.

Attack 1

Attack 1 is a kind of memory exhaustion attack. It occurs when the attacker, as a client, exits the handshake abnormally. After the second step in a full handshake, the server allocates some buffer in its memory to save the connection state. If the client exits before the third step, because the current WTLS protocol doesn’t specify the timeout timer for the server, the server can’t notice the drop-out, still waiting for the client’s response and keeping allocated buffer. Thus, the allocated buffer is wasted. If the attacker sends a lot of such hostile ClientHello messages, the server’s memory resources will be exhausted. This kind of attack is very simple to carry out. Just a forged IP address is needed. The attacker only needs to forge a ClientHello message with a bogus IP address and send it to the server, and then succeeds.

Attack 2

Similar to attack 1, attack 2 is also a kind of memory exhaustion attack. It occurs when the client exits the negotiated secure connection without notifying the server after an accomplished handshake. The server won’t realize this and will still keep allocated buffer. Thus, the memory resources of the server will also be exhausted in the end. To launch such an attack, the attacker needs to make a successful handshake to establish a secure connection first, and so the IP address of the attacker should be genuine.

Attack 3

Different from attack 1 and 2, attack 3 is a kind of CPU exhaustion attack. It occurs at the third step in a full handshake. At this step, the server needs to compute a pre_master_secret, which requires the server to perform an RSA decryption for an RSA key exchange suite, a Diffie—Hellman computation for a DH key exchange suite, or an ECDH computation for an EC Diffle—Hellman key exchange suite. Such computations need to perform a large number of costly modular exponentiations, which are computation-intensive. So in step 3, if the attacker sends numerous Certificate*, ClientKeyExchange*, CertificateVerify*, [ChangeCipherSpec] and Finished messages in correct format, the server’s CPU resources will soon be exhausted. To launch such an attack, a spoofed IP address is enough. Furthermore, to be accepted by the server, the messages in step 1 and step 3 forged by the attacker should have the same IP address and be in correct format. Clearly, after performing computation-intensive operations, the server will deal with the Finished message, and then it will realize the attacker is not a legitimate client, but an attacker. However, since the attacker uses forged IP address, it is hard to trace back to the real attacker.

Improvements

From the above description of three kinds of DoS attacks, we can see that attacks 1 and 3 occur during a handshake, whereas, attack 2 occurs after the handshake is finished. Accordingly, our improvements include two parts: part 1 improves the resistance to DoS attacks during a handshake, and part 2 improves the resistance to DoS attacks when the handshake is complete. The improved protocol is supposed to have the following characteristics:

- memory-DoS: it must resist memory exhaustion attacks during a handshake and after a successful handshake;
- computation-DoS: it must resist CPU exhaustion attacks on the server during a handshake;
- efficiency: the improvements shouldn't operate at the cost of the efficiency of the current protocol.

Part 1

The goal of part 1 is to improve the resistance to memory-DoS attacks (attack 1) and computation-DoS attacks (attack 3) during a handshake. The idea is a two-phase authentication: a weak authentication and a strong authentication. Passing a weak authentication means the IP address of the client is genuine, while passing a strong authentication means the identity of the client is genuine. In other words, the idea is to start the protocol with a weak authentication (of IP addresses), and possibly later perform a strong authentication (of certificates). During the strong authentication phase, the server needs to perform some computation-expensive operations. Only if the weak authentication is passed, the strong authentication is to be performed. Thus the two-phase authentication is immune against IP spoofing attacks.

We use client cookies to defend against memory-DoS attacks. During a handshake, the server can use received client cookies to restore the connection state. Thus it doesn't need to save connection state until the handshake is complete. Meanwhile, we use client puzzles to counter computation-DoS attacks. When the client with a certain IP address opens a lot of WTLS connection requests, the server request the client to solve the client puzzle before it passes the weak authentication. To solve the client puzzle, the client needs to consume a large amount of computational resources, so that the client can only pass a few weak authentications before entering the strong authentication phase. Thus, the server only needs to perform a few strong authentications, and is immune against computation-DoS attacks.

The full handshake for part 1 is shown in Fig. 3 (changes in comparison to Fig. 1 are underlined in the figure). Before we illustrate the improved protocol, let's first look into how to generate the ClientCookie and the ClientCookieRes.

Principles to generate the ClientCookie are given in Karn and Simpson (1999), including:

> the cookie MUST depend on the specific parties; it MUST NOT be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity; the cookie generation and verification methods MUST be fast to thwart attacks intended to sabotage CPU resources.

A recommended method for generating the cookies (Karn and Simpson, 1999) is to use a cryptographic hash of both IP addresses, both UDP

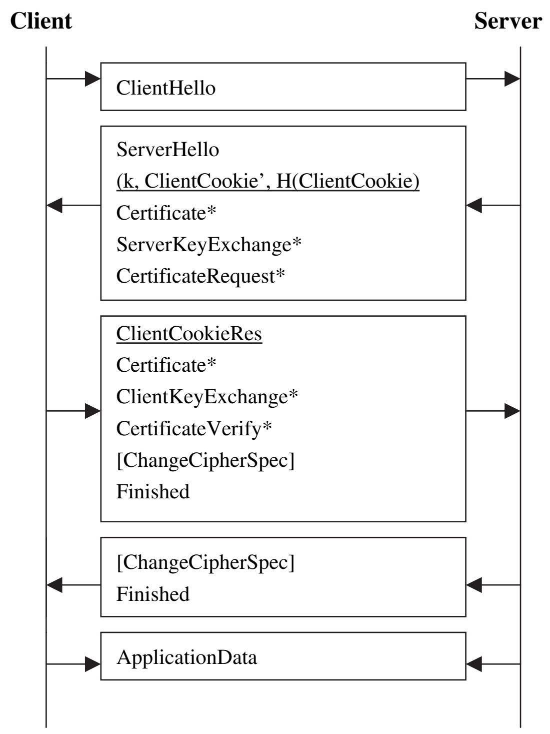

Figure 3 The full handshake for part 1.

ports, some locally generated secret value (which must be same for all clients, and must be periodically changed), and some other context dependent information. We adopt a similar method to generate ClientCookie in this paper, as follows.

$$\begin{aligned}\text{ClientCookie} = H(&\text{client.random} + \text{server.random} + \text{client.UDP port} + \text{client.IP address} + \text{server.IP address}\\ &+ \text{server.port} + \text{session id} + \text{Cipher suite} + \text{client key id} + \text{compression method}\\ &+ \text{sequence number mode} + \text{key refresh} + \text{certificate request} + \text{secret})\end{aligned} \qquad (1)$$

In formula (1), "H" is a cryptographic hashing function (such as SHA-1). "Client.random" is the client's random number sent in the ClientHello message, while "server.random" is the server's random number sent in the ServerHello message. 'Session id", "cipher suite", "client key id", "compression method", "sequence number mode" and "key refresh" are items sent in the ServerHello message. If the server sends the CertificateRequest message in the second step, the value of "certificate request" is "1", else "0". "Secret" is a random number kept by the server secretly, which is updated periodically (default 30 min). "Secret" is used to ensure that only the server can generate correct cookies.

In Fig. 3, the ClientCookieRes message includes the following items: "Session id", "cipher suite", "client key id", "compression method", "sequence number mode", "key refresh", "client. random", "server.random", "certificate request" and "ClientCookie".

The whole improved protocol is as follows. In step 2, the server sends a (k, ClientCookie′, H (ClientCookie)) message to the client, where "ClientCookie′ " is "ClientCookie" with its k lowest bits set to 0, and "H" is a cryptographic hashing function, and is preimage resistant. In step 3, the client responds to the server's (k, ClientCookie′, H (ClientCookie)) with the ClientCookieRes message, in which the client needs to solve the ClientCookie. Since "H" is a preimage resistant hash function, the best way for a client to solve the ClientCookie, is to try values in the domain bounded by the ClientCookie′ and the ClientCookie until a match is found. Since the ClientCookie′ is different from the ClientCookie only in their k lowest bits, the number of solving the ClientCookie should take, on average, 2^{k-1} calculations of "H". For example, assume $k = 4$, then the client needs to perform 8 calculations of "H", on average, to solve the ClientCookie. Thus, the server can limit the client's computation exhaustion attacks by regulating k. When k is bigger, the client needs more computations to get the ClientCookie.

In step 3, when the server receives the ClientCookieRes message, it can recalculate the ClientCookie with items in it. Assuming that the result is "ClientCookie′", the server compares " ClientCookie′" with "ClientCookie" in the ClientCookieRes message. If not equal, the weak authentication fails, and the server will deny the connection request, else the weak authentication is passed, and the server performs the strong authentication. After the strong authentication is ok, the server checks if the concurrent number cn of the finished secure connection of a single IP address is below the limit (default 4). If yes, the server permits the client to establish a secure connection and allocates some buffer to save the connection state, else it denies the connection request.

How to regulate k? The metric we used is the number of wrong modular exponentiation computations being performed for a single IP address within a short time (default 10 min). Wrong exponentiation computations mean wrong RSA decryptions for an RSA key exchange suite, wrong Diffie—Hellman key exchange computations for a DH key exchange suite, or wrong ECDH key exchange computations for an EC key exchange suite. Formula (2) is to compute k.

$$k = 2^n - 1 \qquad (2)$$

In formula (2), "n" means the number of wrong modular exponentiation computations within a short time (default 10 min). When n increases, k increases too, vice versa. When $n = 0$, k is 0. In this case, the ClientCookie′ is equal to the ClientCookie, which implies the client doesn't need to solve the client puzzle when everything is ok.

Part 2

The goal of part 2 is to improve the resistance to memory-DoS attacks (attack 2) after the handshake is complete. If the client exits the connection without notifying the server, the connection state will be kept in its memory, then the memory resources of the server will be exhausted in the end. To avoid such attacks, an application data timer is needed. When the secure connection is complete, the server starts an application data timer. If no data exchange during a long time (default 30 min), the timer will time out. Thus the connection will be closed and the allocated memory resources will be freed. Since in part 1 we limit the concurrent number cn of the finished secure connection of a single IP address to a small number (default 4), the memory resources wasted by the attacker with a single IP address are greatly reduced to a negligible level.

Discussion

Now, we begin to discuss whether the improved protocol satisfies our security requirements, including its resistance to memory-DoS attacks, computation-DoS attacks and its efficiency.

Since when the server receives the ClientCookieRes message, it can recalculate ClientCookie with items in it, and the connect state can be restored. Thus before the server and the client establish a complete secure connection, the server needn't allocate any buffer to store connection state, and the improved protocol is immune against memory-DoS attacks during a handshake. In addition, after a secure connection is established, if no data exchange during 30 min, the server will close the secure connection and free the allocated memory resources. Thus the improved protocol is also immune against memory-DoS attacks after a successful accomplished handshake.

The two-phase authentication is secure against IP spoofing attack. So successful computation-DoS attacks can only be carried out with valid IP addresses. However, client puzzles can limit the number of a single IP address passing the weak authentication within a short time (default 10 min), and reduce the CPU resources wasted by the attacker with a single IP address. So the improved protocol is immune against computation-DoS attacks.

Compared with the current protocol, the improved one only needs a few extra hash operations and two extra messages, since a single hash operation won't cost many CPU resources and two extra messages won't cost a lot of network bandwidth, the improved protocol's efficiency is comparable to the current protocol's.

However, there is still a minor flaw in the improved protocol. For every recently visiting client (these clients should pass the weak authentication, thus the IP spoofing attack is avoided), the server needs to keep concurrent number cn, wrong modular exponentiation computation number *n* and the corresponding client IP address in its memory, which opens opportunity for a coordinated attacker to consume much of the server's memory by using a very large number of valid IP addresses. In other words, our improved protocol is susceptive to distributed denial of service (DDoS) attack. We still have no solution to this problem now. However, DDoS attacks need a lot of valid IP addresses, which are very hard to obtain for an attacker in normal cases. In addition, if the attacker controls a lot of valid IP addresses, the attacker can always easily exhaust the server's CPU and memory resources by simply performing a lot of normal secure connections. So, though the improved protocol is still vulnerable to DDoS attacks, we believe that it's reasonable.

Conclusion

The recent explosive growth of the WAP application has brought with it a need to securely protect sensitive communications sent over this network. Though, the WTLS protocol is used for protection of WAP traffic, it is still not secure enough.

The WTLS protocol is closely modeled after the famous SSL protocol. But there are some differences between them. The WTLS protocol runs on the top of the unreliable WDP protocol, whereas the SSL protocol operates on the top of the reliable TCP protocol. The WDP protocol in the WAP protocol suite is just like the IP protocol in the TCP/IP protocol suite. Clearly, it can be concluded that the WTLS protocol in the WAP protocol suite is more like the IPSEC protocol than the SSL protocol in the wired Internet protocol suite. Thus designing the WTLS protocol in modeling after the SSL protocol may be not a good idea.

In this paper, we present three kinds of attacks on the current WTLS protocol and propose some improvements to avoid these attacks. The improved protocol is immune against memory-DoS, computation-DoS attacks, and its efficiency is comparable to the current protocol.

Acknowledgments

This paper was partially supported by NSFC under the grant 60273049 and 90104005 in China. We wish to extend thanks to the anonymous referees for their valuable comments that helped to improve the presentation of this paper.

References

Aiello William, Bellovin Steven M, Blaze Matt, Ioannidis John, Reingold Omer, Canetti Ran, et al. Efficient, doS-resistant, secure key exchange for Internet protocols. In: Proceedings of the 9th ACM conference on computer and communications security; November 2002. p. 48–58. Washington, DC, USA.

Aura T, Nikander P, Leiwo J. DOS-resistant authentication with client puzzles, Proceedings of the 8th international workshop on security protocols; April 2000.

CERT Coordination Center. TCP SYN flooding and IP spoofing attacks. CERT Advisory CA-1996-21. Available from: <http://www.cert.org/advisories/CA-1996-21.html>; September 1996.

Cox Alan. Linux TCP changes for protection against the SYN attack. Available from: <http://www.wcug.wwu.edu/lists/netdev/199609/msg00091.html>; September 1996.

Freier A, Karlton P, Kocher P. The SSL protocol version 3.0. Available from: <ftp://ftp.netscape.com/pub/review/ssl-spec.tar.Z>; March 1996.

Inc. Berkeley Software Design. BSDI releases defense for Internet denial-of-service attacks. Available from: <http://www.bsdi.com/press/19961002.html>; October 1996.

Karn Phil, Simpson William A. Photuris: session-key management protocol. Experimental RFC 2522, IETF; March 1999.

Meadows Catherine. A formal framework and evaluation method for network denial of service. In: Proceedings of the 12th IEEE computer security foundations workshop; June 1999. p. 4–13. Mordano, Italy.

WAP Forum. Wireless transport layer security version 06-Apr-2001. WAP-261-WTLS-20010406-a; April 2001.

Ruishan Zhang was born in Harbin, P.R. China, in 1977. He received his B.S. degree in Computer Science & Engineering from Shanghai Tiedao University, Shanghai, in 1998, and his M.S. degree in Computer Science & Engineering from North China Institute of Computing Technology, Beijing, in 2001. He is currently pursuing his Ph.D. degree in the in the Department of Computer Science & Engineering, Shanghai Jiaotong University. His current research interests include Network Security and Ad Hoc Network.

Kefei Chen was born in Beijing, China. He received the B.S. and the M.S. degrees in applied mathematics from Xidian University, Xi'an, in 1982 and 1985, respectively, and the Ph.D. degree from Justus-Liebig University, Gießen, Germany, in 1994. Dr. Chen is the author of more than 60 research papers, and he received a National Natural Science Award of China in 1990, and Science and Technology Progress Awards by the Chinese State Commission of Education, in 1989 and 1993, respectively.