Wireless Transport Layer Security

Introduction

My project is on the study of WTLS (Wireless Transportation Layer Security), which is the standard of WAP (Wireless Application Protocol)’s security protocol. This project is going to find out the working mechanism of WTLS, especially the handshaking protocol. The result of this project will be integrated with Na Tang’s WAP Gateway project to strengthen the security capability between WAP client and WAP gateway/proxy. She works on WSP layer of WAP protocol stack to support WML transmission between WAP client, WAP gateway and HTTP/WAP server.

WTLS Survey

WTLS is designed by www.wapforum.org to provide security for WAP protocol. WTLS’s design is based on TLS architecture and wireless computing environment requirements.

**TLS and WTLS**

WTLS originates from TLS (Transportation Layer Security), which is the IETF standard for Internet transaction security. SSL is the implementation of TLS on Internet and is widely used in online business (E-business). WTLS follows the structure of TLS, but it must consider the specific features of wireless computing environment (PDA, Pocket PC, Handheld, etc…).

1. In wireless computing environment, **propagation delay** is much more significant than **transmission rate**. So WTLS must reduce the communication cost of security processing.
2. Wireless communication links have weaker reliability than wired links. So WTLS must have mechanisms to allow certain extent of unreliability while still guaranteeing the security.
3. Mobile nodes have limited computation capability compared with desktop computers. So the transmission rate on mobile node is much lower than desktop computers, either. SSL requires high computation capability to encrypt/decrypt data. So WTLS must redefine the cypher suites to fit for this requirement. Also, mobile nodes have lower power and less memory.

From the aspect of encryption/decryption, we can consider WTLS as a simplified SSL (so-called light-weight security protocol). But WTLS supports secure connection resume, which allows mobile node to reuse the previous secure connection. This is a compromise between reliability and security.

The protocol stack of WAP is defined at [www.wapforum.org](http://www.wapforum.org). WTLS works between WTP (Wireless Transportation Protocol) and WDP (Wireless Datagram Protocol). WTP guarantees the reliability between peer nodes. WDP is connectionless datagram protocol (like UDP in TCP/IP). So we see that WTLS works on an unreliable protocol, while SSL is based on TCP. This is a big issue for WTLS.

**WTLS Architecture**

<1> WAP Gateway

**WAP gateway** is the transition point between WAP client and Internet online servers. When a connection is established between WAP client and Internet online server, WTLS is used to
establish the secure connection between WAP client and WAP gateway, in which data is encrypted. The data is decrypted at gateway. Then gateway encrypts the data using SSL and relays the data to requested online servers. Here WAP gateway is the point of security problems because data is exposed at this point. At present, WAP forum uses end-to-end security protocol to solve this problem. That is, there is a higher level of secure connection between WAP client and online server. WAP gateway is not involved in this secure connection. The idea of tunneling [7] is used to implement this mechanism.

<2> WTLS Protocol Stack

The protocol stack of WTLS has two layers. The lower layer is Record Protocol, which is the part to encrypt/decrypt data using agreed security policy (agreement on algorithms, keys and certificates). The upper layer has 4 different sub-protocols:

1. **Handshaking Protocol** – to establish/resume the secure connection between WAP client and WAP gateway.
2. **Alert Protocol** – to send urgent data or signals. The termination of secure connection uses alert message, which is a drawback of WTLS, I think.
3. **Change Cypher Protocol** – to exchange keys on the fly to guarantee the security dynamically.
4. **Application Protocol** – to send data from application to Record Protocol and deliver the received data from Record Protocol to applications.

**Project Interests**

I am curious about WTLS secure connection (session) establish process and the effects of connectionless WDP on WTLS. The latter one means that since WDP is not reliable, how WTLS guarantee reliability in both connection establishment and data transfer.

**Handshaking Protocol**

Handshaking protocol is to establish a secure session between WAP client and WAP gateway. If the session is suspended, it’s used to resume the session. Session is suspended in case of WAP client power-off, wireless link break-up, etc…

Why does WTLS support session resume? In wireless environment, propagation delay is much more significant. The communication cost of handshaking is expensive because handshaking involves several rounds of signal information exchange. However, once the secure session is established, both client and gateway will keep the agreed security parameters. So when client is resumed, instead of starting a new session, we can use the previously established security parameters.

How to do that? Every session has a unique ID specified by gateway. So session ID represents the agreement between client and gateway. Since session ID is meaningful only to client and server, we can just use this ID to resume the connection. So client just sends this ID to gateway. If this session is still kept in gateway, gateway will resume the session.

The goal of handshaking protocol is to generate *MASTER SECRET* between client and gateway. Master Secret is used at both client and gateway to generate real key for data encryption/decryption.
**Handshaking Process**

When client wants to establish a new session, it will start a full handshaking process. The illustration of the process can be found in WTLS specification. Here I just give a brief description.

1. Client sends out a **ClientHello** message, in which Client puts a random number and all choices of security policy that it supports, such as compression algorithms, encryption algorithms, public keys.

2. Gateway gets the ClientHello. It will consider this request a new session request if session ID in ClientHello is set to 0. Then gateway will compare security polices that client supports with it's own security policies, then decide a choice. Then gateway will compose **ServerHello** message, in which the session ID, a random number, security policy choice made by gateway, and gateway's public key are included. Certificate is also included if class 2 or class 3 of security level is specified.

3. Client gets the ServerHello. It will get the session ID. Then client sends its pre-master-secret(a secret used to calculate master secret) to gateway in **ClientFinished** message. The pre-master-secret is encrypted using server's public key. It also includes an encrypted string "client_finished". Certificate is also included if class 2 or class 3 of security level is specified.

4. Gateway gets ClientFinished message. It will get the pre-master-secret. Then Gateway will calculate master secret. Gateway sends back a **ServerFinished** message in which encrypted string "server_finished" is included.

5. Both sides start to exchange application data.

Client can also resume a previous session if it still has the state of that session. The illustration of the *resume handshaking* process can be found in WTLS specificaion. Here I just give a brief description.

1. Client sends out a **ClientHello** message in which it puts the previous session ID and a random number.

2. Gateway gets the ClientHello message. It will find the non-zero session ID and search in its session state table. If it still has the information about his session, gateway will send back a message in which an encrypted string "server_finished" and a random number are included.

3. Client gets this message and sends back a message in which an encrypted string "client_finished" is included.

4. Both sides start to exchange application data.

Since WDP is unreliable, how can handshaking deal with handshaking message lose, duplicate and out-of-order? The solution by WTLS is quite simple. It may not be a good method, either. WTLS just sets a rule that all the hello messages to one direction will be packed in one Record Protocol packet. So the effect is that either all are lost, or all are got by the other side.

**Key Generation Process**

First, *public keys* are exchanged to send *pre-master-secret*. On either side:
master-secret = PRF(pre-master-secret, client.random, server.random)
key = PRF(master-secret, sequence number, client.random, server.random)

/* PRF – Pseudo Random Function */

"key" is the real key used to encrypt/decrypt data.

Note that when resume is done, both sides don’t change master secret. But since they exchanged
different random number, a new key will be generated.

Program Features
I wrote a telnet-like WTLS client and a WTLS gateway to mimic the behavior of handshaking.
The following functions involved with handshaking are supported at client:

- new - start a new session
- suspend - suspend a session
- resume - resume a session
- end - close a session

For both client and gateway, each of them has a set of security profiles which contains the
security parameters. When running, each side will select a security profile as default. The
handshaking is done by comparing the profiles. Once a new session is created, a session record
which includes the session ID, master secret, status, and agreed security parameters are created.

Each session has the status of:

- new - under handshaking
- open - opened for use
- suspend - suspended
- close - closed

My program has the following features:

1. WTLS client supports multi-session on one client. A session can be used by multiple
applications. Those application will share the feature of one session. This also saves time of new
handshaking. HTTP has the similar idea to utilize one TCP connection for multiple HTTP
requests/replies. In my implementation, an application is just a pair of WAP request/reply.

2. Client-Gateway communication uses request/reply model, which reflects the behavior of WAP
protocol(WAP request/reply).

3. Full handshaking and Resume handshaking are supported.

4. Hash tables are used to store the extremely sensitive private data, such security profiles and
session records.

5. Message encapsulation/extract on different protocol layers are implemented. This means a set
of message compose/decompose functions and definitions.

6. UDP is used as WDP in IP network.
I didn't implement encryption/decryption. So actually all the data is plaintext format. I didn't do authentication, either. So no certificate exchange is done. This is a class 1 security implementation.

The source code of my project can be downloaded [here](#).

The components of program:

```plaintext
wtls.h     -  Definition of security profile and session record
wtlsprot.h -  Definition of WTLS protocol message formats
wtlshash.c -  Hash functions operated on two hash tables
func.c     -  Common printing, copying, conversion functions
handshk.c  -  Handshaking layer message processing functions
rp.c       -  Record layer message processing functions
client.c   -  WTLS client
server.c   -  WTLS gateway
```

Compilation of program:

```plaintext
cc wtlshash.c func.c handshk.c func.c client.c -o client
c c wtlshash.c func.c handshk.c func.c server.c -o server
```

Execution of program:

```plaintext
server <ServerPort>
client <ServerIP> <ServerPort>
```

**Conclusion**

**About WTLS**

Through this project, I know a lot about WTLS. Also I found some drawbacks of WTLS that may compromise the performance of WTLS.

1. Too simple termination of secure connection. In WTLS, the termination of a secure session is surprisingly simple. Only an alert message is sent by the side that wants to close the session. No reply is needed. Is that sufficient? I don't think so. Consider TCP protocol, the termination of a TCP connection requires almost the same complicated state transition as TCP connection establish. TCP is connection oriented, and WTLS, in its nature, is also a connection, a secure connection. I think the session termination part should be strengthened in later versions of WTLS.

2. The back holes left by alert messages. This is also illustrated in [3]. Some alert messages use plaintext message. Alert message uses application's session to be transferred. This is dangerous. Detailed discussion is in [3].

3. The position of WTLS in WAP protocol stack is confusing. Not like TLS, which is between application layer and TCP layer, WTLS is between WTP and WDP. So the transfer of WTLS messages is not reliable. Messages can be lost, duplicated, and reordered. However, the connection nature of a secure session requires the mechanism to guarantee reliability. Actually, I was so surprised to see in the specification of WTLS that Record Protocol even has sliding window protocol to deal with out-of-order messages. This is the work of transportation layer, isn't it? I think WTLS may be a complete protocol for security, but it's far from a well-designed network protocol.
4. To combine all one-way handshaking messages into one Record Protocol packet has negative effect on the performance. If this packet is lost, more cost is needed to complete handshaking.

**About Project Program**

I need to improve my program in the following two aspects:

1. At present, the WTLS client is a sequential version. That means, although I support the mechanism of multiple sessions, it's done by sequential request/reply. To make it parallel, Na Tang and I come up with a solution which uses UDP peeking and thread synchronization.

   **UDP peeking** is an option when calling recvfrom(). The process using this option to call recvfrom() will read the content of message first, but not receive it. If the process finds out the message is for him(according to session ID), it issues normal recvfrom().

   The calling of recvfrom() is synchronized using semaphore.

2. I chose bad hash table functions to store session records and security profiles. First, I use hcreate(), hsearch() and hdestroy(). But I found that using that, only one global hash table can be used in whole program. Then I used reentrant version hcreate_r(), hsearch_r() and hdestroy_r(). That supports multiple hash tables in one program, but it's so unstable that I worked two weeks get it through. I learned from Professor Herman that there are many other hash functions available for use, such as **MD5**.

   The most important thing I learned from doing this project is that keeping layering principle is so important in network protocol design and implementation. In network protocol design, I observed that WTLS is really not a well-layered protocol, as I discussed above. In implementation, because of the problem I encountered in using hash table, I had to write the code of message encapsulation in one function. I had to be extremely careful not to mix or confuse what current protocol layer should do.

   This project will be integrated with Na Tang ‘s WAP Gateway/Proxy project to strengthen the security feature of WAP gateway. Actually Na Tang and I had a lot of very much useful discussions on project ideas, programming techniques, etc. Thank her very much for her help. She is a good group partner to work with.

   I enjoyed a lot from Professor Herman’s teaching. Besides the network technology, I learned much from him on how to think efficiently and precisely.

**Reference**

Here is the reference list. [1] has very good introduction to basic concepts of security. [2] is a good introduction to WTLS. [3] is a famous article discussing the weakness of WTLS. [6] compares WAP with other wireless application protocols.


[7]. WTLS TLS Profile and Tunneling, [WAP-219-TLS], http://www.wapforum.org