Presentation Outline

• Introduction on Computer security protocols
  • Example Protocol: Needham-Schroeder symmetric key protocol
  • Drawbacks of manual security protocol implementation
  • The need for Automatic generation of security protocols
  • Grammar Notation
  • Protocol high-level specification
  • System architecture
  • Lexical analysis
  • Syntactical analysis
  • Semantic analysis
  • Code generator
  • Experimental results
  • Conclusion & Future work.

Automatic Generation of Computer Security Protocols Implementation

2005/7/6
Why Use Security Protocols

• Rapid advancements in hardware and software technologies.

• Exponential growth of computer networks and internets.

• The rising popularity of on-line services, i.e. e-commerce.

• On-line services have the drawback of information exchange over insecure network channels.
Automatic Generation of Computer Security Protocols Implementation

Why Use Security Protocols – Cont’d

<generated false data>

45693290039

45129876336
Significance of Security Protocols

- Provides protection against malicious attacks.

- An ideal protocol must provide its users with three core services:
  - Confidentiality
  - Integrity
  - Authentication
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal</td>
<td>Communicating Party</td>
</tr>
<tr>
<td>Intruder</td>
<td>Party eavesdropping on a private conversation</td>
</tr>
<tr>
<td>Message</td>
<td>Data exchanged between principals</td>
</tr>
<tr>
<td>Nonce</td>
<td>Number used once that is randomly generated</td>
</tr>
<tr>
<td>Plain/clear-text</td>
<td>Data sent without being encrypted</td>
</tr>
<tr>
<td>Encrypt/Cipher</td>
<td>Data unintelligible to intruders</td>
</tr>
<tr>
<td>Decrypt/Decipher</td>
<td>Convert cipher data to clear text</td>
</tr>
<tr>
<td>CA</td>
<td>Trusted network authority managing security credentials.</td>
</tr>
</tbody>
</table>
• A computer security protocol is a distributed software.

• Executed prior to the process of information exchange.

• A mean of assuring parties of each other’s validity and authenticity.
Presentation Outline

• Introduction on Computer security protocols
• **Example Protocol: Needham-Schroeder symmetric key protocol**
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results
• Conclusion & Future work.
Needham-Schroeder Security Protocol

\[ S \rightarrow A : E(K_{as} : Na, B, K_{ab}, E(K_{bs} : K_{ab}, A)) \]

Automatic Generation of Computer Security Protocols Implementation

2005/7/6
Needham-Schroeder Security Protocol – Cont’d

Automatic Generation of Computer Security Protocols Implementation 2005/7/6
• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol

**Drawbacks of manual security protocol implementation**
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results

• Conclusion & Future work.
Drawbacks of Manual Security Protocols Implementation

- Often prove to be suffering from severe security pitfalls.
- Developed in an ad-hoc manner with minimum compliance to any formal guiding principals.
- Suffers from human-prone errors.
- Lack of optimality, due to the use of unnecessary functions.
- Inefficiency and high cost.
The need for Automatic generation of security protocols

Presentation Outline

- Introduction on Computer security protocols
- Example Protocol: Needham-Schroeder symmetric key protocol
- Drawbacks of manual security protocol implementation

**The need for Automatic generation of security protocols**

- Grammar Notation
- Protocol high-level specification
- System architecture
- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Code generator
- Experimental results
- Conclusion & Future work.
The Need for Automatic Generation of Security Protocols

- Protocol design and implementation is not trivial.

- Careless protocol design and implementation has drastic consequences.

- The need for highly dependable flawless protocols.

- Manually developed protocols often suffer from severe security pitfalls despite thorough testing.
• Most security failures occur at the implementation level.

• Automatic implementation generation allows for optimal utilization of resources (time, cost, efforts).

• To eliminate human-prone errors.
• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols

**Grammar Notation**

• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results

• Conclusion & Future work.
• Used as a foundation for the formation of security protocols specifications.

• A grammar is a collection of production rules.

  <LHS> → <RHS>

• <LHS> : an abstraction
• <RHS> : definition of abstraction (tokens – abstractions)

• Protocol specification is a sequence of applications of grammar rules.
Grammar Notation - Cont’d

• Allows for describing a protocol in terms of:
  – Principals
    • Number of declared principals
    • List of keys initially known by each
  – Messages
    • Number of messages
    • Order in which they are being exchanged
    • Contents of each messages
Protocol \rightarrow \text{Principals Messages}
Principals \rightarrow \text{Principal Principals} \mid \text{Principal}
Message \rightarrow \text{Message} \mid \text{Message Messages}
Principal \rightarrow \text{id} \mid \text{id} \text{‘knows’} \text{‘(‘ Keys ‘)’}
Keys \rightarrow \text{Key ‘,’} \text{Keys} \mid \text{Key}
Key \rightarrow \text{‘key’} \text{‘(‘ id ‘,’ id ‘)’} \mid \text{‘Key’} \text{‘(‘ id ‘)’} \mid \text{‘IKey’} \text{‘(‘ id ‘)’}
Message \rightarrow \text{‘(‘ id ‘,’ id’)’} \text{CData}
CData \rightarrow \text{Data ‘,’} \text{CData} \mid \text{Data}
Data \rightarrow \text{id} \mid \text{Key} \mid \text{Nonce} \mid \text{text} \mid \text{‘E’} \text{‘{‘ Key ‘}’} \text{‘{‘ CData ‘}’}
Nonce \rightarrow \text{‘N’} \text{‘(‘ id ‘,’ num ‘)’}
• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation

• **Protocol high-level specification**
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results
• Conclusion & Future work.
Protocol High-level Specification

Automatic Generation of Computer Security Protocols Implementation
- Introduction on Computer security protocols
- Example Protocol: Needham-Schroeder symmetric key protocol
- Drawbacks of manual security protocol implementation
- The need for Automatic generation of security protocols
- Grammar Notation
- Protocol high-level specification

**System architecture**

- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Code generator
- Experimental results
- Conclusion & Future work.
Automatic Generation of Computer Security Protocols Implementation

System Architecture

Lexical Analyzer → Syntactical Analyzer → Semantic Analyzer → Code Generator

Compiler

Automatic Generation of Computer Security Protocols Implementation

2005/7/6
Presentation Outline

- Introduction on Computer security protocols
- Example Protocol: Needham-Schroeder symmetric key protocol
- Drawbacks of manual security protocol implementation
- The need for Automatic generation of security protocols
- Grammar Notation
- Protocol high-level specification
- System architecture
- **Lexical analysis**
  - Syntactical analysis
  - Semantic analysis
  - Code generator
  - Experimental results
- Conclusion & Future work.
Lexical Analysis

• Performed by the scanner module.

• Analyzes the input file on word-by-word basis.

• **Goal:** ensuring the validity of letter compositions forming tokens, and reject invalid ones.
<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Identifiers. E.g. A,B,…</td>
</tr>
<tr>
<td>text</td>
<td>String of text. E.g. “Hello W”</td>
</tr>
<tr>
<td>num</td>
<td>Set of digits of integer values</td>
</tr>
<tr>
<td>‘(‘, ‘)’, ‘{‘, ‘}’, ‘,’</td>
<td>Symbolic tokens</td>
</tr>
</tbody>
</table>

Automatic Generation of Computer Security Protocols Implementation
2005/7/6
• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis

**Syntactical analysis**
• Semantic analysis
• Code generator
• Experimental results

• Conclusion & Future work.
• Performed by the parser.

• Performs sentence-based analysis.

• **Goal:** to ensure the validity of the compositions of tokens.

• Constructs a hierarchical parse structure which is referred to as an abstract syntax tree.
Presentation Outline

• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis

• **Semantic analysis**

• Code generator
• Experimental results

• Conclusion & Future work.
Semantic Analysis

- Third and last analytical module.
- Analyzes the parse tree for context-sensitive information.
- Detects errors that are impossible to detect in previous stages.
- The semantic analyzer checks the input file compliance to 10 semantic constraints.
1. The protocol must involve at least two communicating principals.

2. At least a single message must be exchanged between principals.

3. Only declared principals may take part in the action of sending or receiving a message.

4. A principal sending a message may encrypt the message using only a key that is known to him.
5. If a principal is sending a message encrypted with a key that is not known to him, the key must be received by the principal via a previous message. “Middle man situation”

6. A principal receiving a message may decrypt the message using only a key that is known to him.

7. A secret key may not be sent in clear text.
8. A principal cannot send a nonce which he does not know.

9. A principal cannot appear as both communicating ends of a message.

10. A principal identifier sent as a data must be one of the declared communicating principals.
Semantic Analysis-Cont’d

• **Results:**

  – A list of all semantic errors, if any.
  
  – List of known keys to each principal *after* a complete execution of the protocol.
  
  – List of all the keys received but not intended to a principal knowledge base.
  
  – List of known set of nonces to each principal *after* protocol execution.
Presentation Outline

• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results
• Conclusion & Future work.
• “Program that takes a higher-level specification of a piece of software and produces its implementation”

• Last phase of the tool.

• Traverses the parse tree once and produces the code accordingly.

• Number of resultant files is not fixed.
• The generator faces a series of decisions with varying complexity:
  – Number of principals.
  – Role assumed by each principal.
    • Client → Active connection
    • Server → Passive connection
    • Server\Client → Passive + Active connections
  – Data variables declarations.
  – Connect\Disconnect – *for who!*?
// Create a listening socket.
sI = CreateIncomingSocket(PORT3);
if(sI == INVALID_SOCKET)
{
    WSACleanup();
    printf("** Error creating a socket for incoming connections!");
    return(-1);
}

// Accept connections from peers forever.
while((sC = accept(sI, NULL, NULL)) != INVALID_SOCKET)
{

// Create a Sending Socket
ss = ConnectTo(sPeer, PORT3);
if(ss == INVALID_SOCKET)
{
    WSACleanup();
    printf("** Error connecting to remote station!");
    return(-1);
}

//Close the socket connected to S
closesocket(ss);
```c
#include "./crypto_common.h"
#include "./network_common.h"

int main(int argc, char ** argv)
{
    WSADATA WsaDat;
    SOCKET sI;
    SOCKET sC;
    char *buf;
    unsigned char *e_data;
    unsigned char *e_data_len;
    int ret;
    int data_len;
    unsigned long NS1;
    unsigned char KBS[MAX_NAME_LEN];
    EVP_CIPHER_CTX cBS;
    char Apeer[MAX_NAME_LEN];
    unsigned char KAB[MAX_NAME_LEN];
    EVP_CIPHER_CTX cAB;

    // Get the local host name
    strcpy(localname, argv[0]);

    // Initialize the Windows socket library.
    if (WSAStartup(MAKEWORD(1, 1), &WsaDat) != 0)
    {
        printf("** Error initializing the WinSock library!");
        return(-1);
    }
}
– How to obtain values of data items.
  • Principal ID: command line
  • String of text: from protocol specification
  • Nonce: Generate nonce function
  • Key: Generate key

– Encrypt/Decrypt
  • Load keys.
  • Prepare encryption/decryption ciphers.
  • Encrypt/Decrypt

– Send/Receive
  • Determine data type.
  • Determine whether or not data is encrypted.

– Close Sockets and exit.
Presentation Outline

• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator

• Experimental results

• Conclusion & Future work.
1. Example 1: Semantically Incorrect Protocol Specification

A knows (Key(A,S))  
B knows (Key(B,S))  
S knows (Key(A,S), Key(B,S))  
(A,S) A,Q,N(U,1), "HELLO W"  
(A,A) Key(A,B), E{Key(A,B)}{N(A,1), B, E{Key(B,S)}{Key(A,B), A}}  
(A,R) E{Key(B,S)}{Key(A,B), A}  
(B,A) E{Key(A,B)}{N(B,1)}  
(A,B) E{Key(A,B)}{N(B,1)}  
(A,S) E{Key(A,B)}{"HI THERE"}
<table>
<thead>
<tr>
<th>Error</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>R is an invalid message receiver in message 3</td>
<td>R was never declared as a participating principal</td>
</tr>
<tr>
<td>Secret key (A,B) was sent in clear text in message 2</td>
<td>Violation of semantic constraint number 7</td>
</tr>
<tr>
<td>Principal identifier Q sent in message 1 is invalid</td>
<td>Q was never declared as a participating principal</td>
</tr>
<tr>
<td>A is ending Nonce (U,1) that is not known to him in message 2</td>
<td>A neither generated nor received the nonce via a previous message</td>
</tr>
<tr>
<td>Sender A is identical to receiver A in message 2</td>
<td>Violation of semantic constraint number 9</td>
</tr>
<tr>
<td>Decryption key KEY(A, B) used by B is invalid in message 5</td>
<td>B never received this session key since the recipient in message 3 was mistyped as R</td>
</tr>
<tr>
<td>Decryption key KEY(A,B) used by S is invalid in message 6</td>
<td>CA server does not store generated session keys, hence S does not know he specified key.</td>
</tr>
</tbody>
</table>
2- Example 2: Wide Mouted Frog Protocol

A knows (Key(A,S))
B knows (Key(B,S))
S knows (Key(A,S),Key(B,S))
(A,S) A,E{Key(A,S)}{N(A,1),B,Key(A,B)}
(S,B) E{Key(B,S)}{N(S,1),A,Key(A,B)}
Presentation Outline

• Introduction on Computer security protocols
• Example Protocol: Needham-Schroeder symmetric key protocol
• Drawbacks of manual security protocol implementation
• The need for Automatic generation of security protocols
• Grammar Notation
• Protocol high-level specification
• System architecture
• Lexical analysis
• Syntactical analysis
• Semantic analysis
• Code generator
• Experimental results

• Conclusion & Future work.
Conclusion

- In the first part of this work we have presented a C++ implementation of Needham-Schroeder symmetric key protocol.

- In the second part we have presented a C implementation of our method for the automatic generation of computer security protocols implementation.

- The resultant protocol implementation makes use of our two custom libraries; “crypt_common.h” and network_common.h”

- The automatic generation of computer security protocols is a crucial and active field of study.
• Generators development is not trivial, it requires careful planning and multi-tier application testing.

• Security protocol generators save time, money and efforts.

• Security protocol generators eliminate the factor or human-error at implementation level.

• The necessity of generalizing the use of generators in other disciplines due to their positive impact on the process of software development.
Future Work

• The implementation of a verifier module to our compiler to analyze security protocols to be implemented and ensure its correctness in terms of possible security pitfalls and which suggests necessary amendments when appropriate.

• The modification of our grammar to efficiently accommodate for a cornerstone of modern day security protocols, which is the “middle man” situation.

• Increase the expressiveness of our grammar to allow for the inclusion of possible nonce arithmetic operations.

• Include the implementation of RSA public key cryptography to allow for the generation of asymmetric key computer security protocols.
Thank you.