In the Proc. of Computer Applications in Industry and Engineering (CAINE03), Las Vegas, Nevada USA, 11-13 November 2003

A UNIFIED ARCHITECTURE FOR THE IMPLEMENTATION OF SECURITY PROTOCOLS

Ibrahim S. Abdullah **Daniel A. Menascé**
Department of Computer Science, MS 4A5
George Mason University
Fairfax, VA 22030-4444, USA
iabdulla@gmu.edu **menasce@cs.gmu.edu**

ABSTRACT

Most security protocols share a similar set of algorithms and functions and exhibit common sequences and patterns in the way they operate**.** These observations led us to propose a unified architecture for the implementation of security protocols in the form of a security toolbox system. Our design, based on the concepts of Component Based Software Engineering (CBSE), provides fast and flexible implementation and deployment of security protocols.

Keywords: security protocol, automatic protocol implementation, CBSE, toolbox.

1. INTRODUCTION

The current approach to implementing security protocols is centered on designing a protocol as a single package comprised of two layers: control and a library of algorithms. This approach is based on the assumption that every protocol is complete and does not need to be integrated with other security protocols. This assumption may not be valid when several security protocols need to coexist leading to redundancy and conflicts. For example, running S/MIME over IPsec introduces redundancy. Both provide similar encryption: S/MIME does it at the document level and IPsec at the packet level. This situation is common in B2B applications when an application uses S/MIME as a document-level protection while using IPsec to protect the communication with a remote branch office.

Security protocols tend to have more commonalities than differences; they share significant functionality and utilize a common set of encryption, hash, and compression algorithms. They usually differ in the handshaking mechanism (which includes authentication), target data, header processing, key sizes, replay mechanisms, and in the order in which the various algorithms are applied. Based on this pattern we propose a new approach, called the *security toolbox,* for implementing security protocols. In this approach, functions are designed as independent components called *tools* and then grouped into a toolbox system. A specification mechanism called *template* is used to manage the functionality needed to provide the required security services.

Figure 1 illustrates the major components of such a toolbox. Every tool carries out a specific function such as: encryption, decryption, random number generation, integrity protection, anti-replay, and header processing. A template is a set of specifications that define the required security services. Given a template from a database of templates, the security toolbox system assembles agent programs out of the tools to carry out the overall protection. The novelty of this work is in the template mechanism, which contains the specification of a protocol. Based on this specification, protocol code is generated automatically.

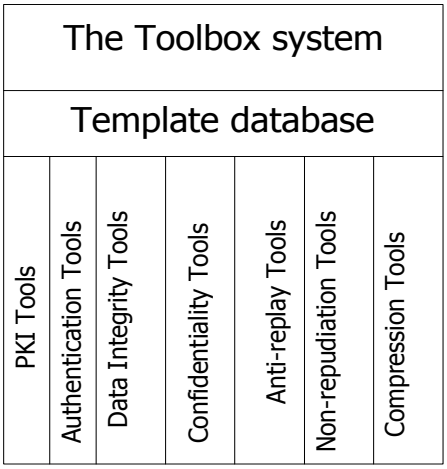

Figure 1 Major components of the toolbox system

Ours is, to our knowledge, the first work that takes a pragmatic approach of using CBSE for security services in conjunction with the existing Internet infrastructure. Projects such as the x-kernel [18, 19], Ensemble [3], Click [4], Cactus [17, 26, and 28] and their extensions [12, 13, 14, 21, 22, 25, 26, 27, and 28] propose partial or total substitution of the TCP/IP communication stack, which limits their applicability. On the other hand, our approach provides the required security features on top of predominant TCP/IP services.

In security protocols, The Conduits+ [11] The SecCom [17], The Antigone [23 and 24] and [15, 16] do not provide an independent separate mechanism to deliver protocol specification. Cryptographic libraries also implement security services as independent components. Examples of such libraries are: IBM's CCA, RSA's Cryptoki, Microsoft's CryptoAPI, Sun's JCA/JCE, X/Open's GCS-API, and Intel's CSSM-API However, there is no high level mechanism and easy way for producing useful valid compositions out of those libraries [29].

This paper is organized as follows. Section 2 presents the traditional approach used in the implementation of security protocols. Section 3 identifies common patterns in security protocols. Section 4 presents the proposed architecture. Section 5 presents some design issues of the components, the tools, the templates, and the performance of the system. The next section provides some application scenarios. Section 7 presents some concluding remarks.

2. THE TRADITIONAL APPROACH

In this work, we focus on communication security protocols based on TCP/IP such as SSL, IPsec, S/MIME, and SET. Access control security protocols at the application level, such are Kerberos, and those that are based on non-TCP/IP protocols are outside the scope of this research.

The current approach of designing autonomous and complete protocols has several disadvantages. First, it may result in conflicts when various protocols need to coexist. For example, if compression is done at an upper layer (e.g., S/MIME), repeating it at a lower layer may increase the size of the message. Second, there is not enough flexibility in most security protocols (e.g., S/MIME, SSL, IPsec) and, consequently, their users have to adopt them without being able to make changes. For example, a user may want to change the header by adding or removing some fields for QoS purposes. Third, these protocols offer coarse grained security services and the developer of an application does not have the ability to fine tune operations within the use of these protocols. For example, S/MIME services are applied to an entire document, preventing a user from applying it on selective parts of the document. Fourth, the use of coarse grain services may lead to unnecessary performance degradation. For example, SSL has to calculate a block of four shared keys and a couple of IVs even if the user does not want to use some of the modes such as, the read mode, the write mode, the encryption service, or the integrity service [1].

It is difficult to make changes to a monolithic implementation of a protocol. Users often do not have access to source code and even in the cases of open source; it is not trivial to change large pieces of in code. In contrast, a *toolbox approach a*ssumes a dynamic environment and provides enough flexibility so that applications can select the tools that fit their exact needs [5]. Users can also group the tools into a template as specifications of the required protocol and pass it over to their correspondent's parties to generate the code. Some security protocols (e.g., Openssl) do provide a configuration mechanism, in the form of configuration files, to provide flexibility [2]. However, these configuration files are provided to facilitate the selection of available options within these protocols. These configuration files are not portable and limited to this implementation of SSL.

3. SECURITY PROTOCOLS PATTERN

Most security protocols have a similar behavior: they perform some preparation (handshake or key management), receive data from one layer, apply a sequence of cryptographic operations, deliver protected data to another layer, and close the session. Figure 2 shows a diagram that compares the patterns of three security protocols: SSL, IPsec, and S/MIME.

In addition to that, there is a limited set of cryptographic algorithms, on which all security protocols depend on, that provide confidentiality, integrity, non-repudiation, and authentication. For example, RSA, DH, MD5, IDEA, DSS, SHA, DES, RC5, random number generators, and compression algorithms are often used in all major security protocols.

The differences between various security protocols come from the following sources:

1. The way they establish the secure communication (handshake), which includes the authentication of the parties involved.
2. The layer in which the algorithm operates.
3. The scope of the encryption algorithms; whether encryption is applied to the entire message or part of the message, on a packet or on part of the packet. For example, S/MIME applies encryption to the entire message, while SSL applies it to message fragments, and IPsec applies it to packets.
4. The length of the key used in the security algorithms.
5. The order in which cryptographic algorithms are applied. Applying encryption before message authentication code (MAC) exposes its information. However, applying the MAC before the encryption has the advantage of protecting the MAC information but generates more processing overhead to decrypt the message before verifying its validity.

6. Anti-replay prevention mechanisms. While most mechanisms depend on a timestamp, they differ in the way they combine it with random numbers, their format, and location in the header.

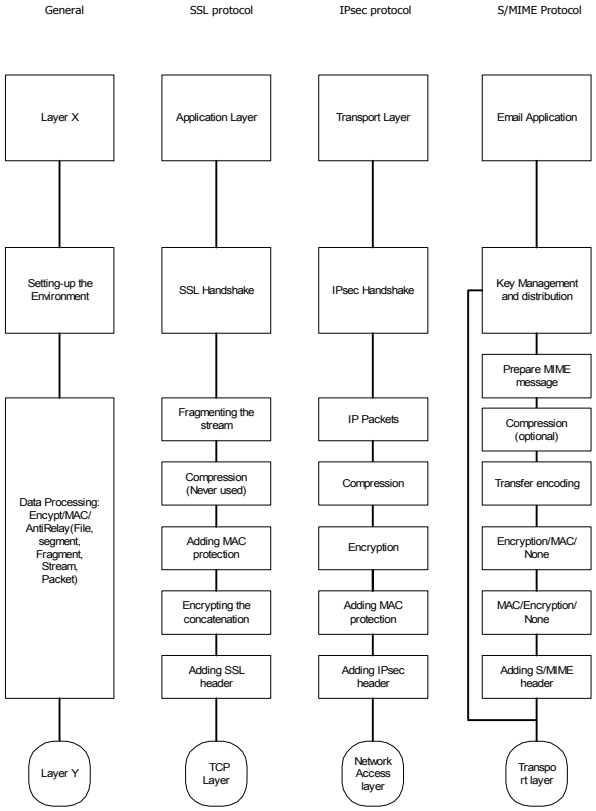

Figure 2 Comparison of security protocols

Recognizing these differences opens the door for many design options. The adoption of one of these options does not necessarily mean that other choices are incorrect or less secure. Different designs affect a system's interoperability with other security systems. Since some of the design choices may be invalid, the protocol designer has to validate the design against its requirements. Designers try to determine the best choice for the situations in which the protocol will operate. However, when the situations change, prior design choices may need to be reevaluated.

4. THE TOOLBOX ARCHITECTURE

The concept introduced here is to decompose all available security protocols into their basic components and enter them into a security toolbox. Then, from this tool set, one can design templates to generate any existing protocol or facilitate the implementation of any new protocol as needed in an easy, fast, and flexible way.

The concept of security toolbox in our work comes from the idea of modular or component-based design approach, which is common in many engineering fields and industries. For instance, most electronic devices, such as computers, are assembled from parts (e.g., motherboards, chips, memory, keyboard, cables) that interact within a single system. Standard interfaces allow components from many different manufacturers to be easily interconnected [7].

The toolbox architecture consists of two parts: one that must be secured as part of the trusted domain of the operating system (CBT) and another that may be part of the user domain. Figure 3 shows the general architecture of the proposed toolbox system.

The secure part consists of the following components:

1. Databases: containing information about different aspects of the operations of the toolbox, such as: private and secret keys, templates, registry for the tools and template names, alert messages, authorization information, policies, and the toolbox configuration information
2. Interpretation engine: it is the central control mechanism of the toolbox. It interprets protocol templates and carries out the actions described in them
3. Security tools: the set of tools that implement the security algorithms available for use.
4. Cache: a temporary memory used to store temporary keys and association information needed during secure sessions.
5. Inter-communication manager: the mechanism responsible to manage control messages between toolboxes running at different hosts (e.g., during handshake).

The second part of the toolbox consists of:

1. Template developer and analyzer: a mechanism that facilitates template creation, verification, and maintenance.
2. Certificate repository: contains copies of the certificates that the toolbox consults for authentication. These certificates may be placed in public storage, because they are protected by its creator's digital signature. This repository could part of a directory service application.
3. Directory services are standard applications used to provide user's authentication and authorization services, such as: LDAP, iPlant, and Active Directories.

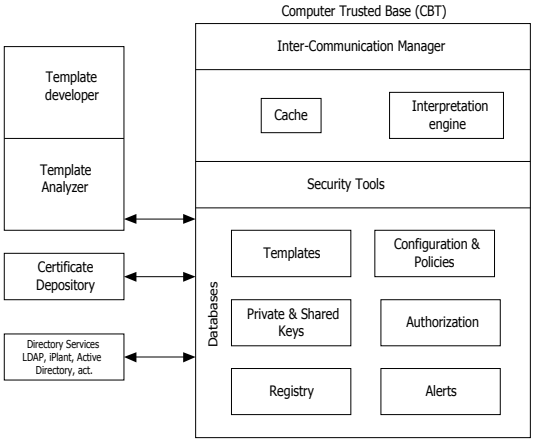

Figure 3 the security toolbox architecture

5. THE TOOLBOX DESIGN

The toolbox system is not just a collection of libraries of code, rather it adds several features:

1- A mechanism called "template" to specify the sequence of operations, target data, and security policies.
2- A mechanism to analyze the template and verify the authorization of usage of the template and tools.
3- A mechanism to generate code, called an "Agent", by linking several tools to work together in a certain sequence specified by the template. During this process the agent is also assigned to work on certain scope of data.
4- A framework to distribute tools and templates among users in a secure manner. The purpose of this framework is to add missing tools and update the toolbox with new versions of tools or templates.

Therefore, the toolbox is not just the code for a set of algorithms in a library but it represents a new way of implementing and deploying security protocols. Current approaches to the implementation of security protocols view an implementation as a single package that has to be developed and installed in every machine. In contrast, the toolbox approach assumes building an infrastructure that accommodates new implementations in an incremental and dynamic process of adding new tools or templates. This approach is possible because security protocols have many aspects in common.

Our work does not call for free composition of the tools, it calls for flexibility. Tool composition should be done by experts and standardization committees. The toolbox approach makes it easy for qualified people to design and experiment with new security protocols in short-time as opposed to implementing every new standard from scratch.

5.1 Tool Design

Every tool is an executable program that is designed based on CBSE principles. A tool can be designed out of a single component or it could comprise several components. Therefore, a component may be shared by more than one tool. For example, an RSA encryption algorithm is a single component tool; however a handshake component may consist of several subcomponents, one of which is the RSA encryption in addition to others such as certificate verification, data signing, and encryption tools.

The set of all tools does not need to be stored in every toolbox. Communicating parties can download the code of a tool from one box to another if the code is needed but missing from one of the boxes. Moreover, tools that are needed for a short time period or by a certain process can be deleted from the toolbox after their use. Moreover, upgrading a tool is easier than upgrading a version of a protocol in the traditional sense. This same concept also applies to templates, which could be downloaded from one user to another or from a trusted center to the user for temporary usage or for an upgrade.

To protect these tools from malicious alteration they have to be signed by their authors and checked by the toolbox administrator before adding them to the system. Trusted tool providers may place them in public repositories on the Internet so users may pull them whenever they need. Newer versions may also be pushed from those trusted providers to subscribed users to keep their systems up-to-date.

5.2 Template Design

Templates provide an abstraction mechanism that groups several tools and coordinates them to perform a certain security task. In traditional thinking, a template can represent a protocol suite, or at least represent a task that is executed by some protocol under a specific set of options or modes. For instance, the operation that IPsec performs in the Authentication Header (AH) mode can be emulated by a single template that manages several security tools in the toolbox to achieve the goals of the AH mode.

We chose XML as the language to express the templates. Five types of XML elements are required to produce a template: name, first-state, object, instance, and state elements. Name is an optional single element used to identify the protocol; the XSLT processor can ignore or

replace it with any other user-supplied identifier. All other elements are mandatory. The first-state, used to identify the starting state, is a single occurrence element because any protocol has a single point of entry for execution. All other elements may have multiple occurrences.

We also successfully developed an XSLT stylesheet that transforms protocol specifications given as an XML document to Java code. Template design and transformation to code is presented in our work in [30]. The XML specifications, the Java XSLT stylesheet, and the XML Schema that was developed for validation of the approach are at: http://mason.gmu.edu/~iabdulla/.

6. APPLICATION SCENARIOS

The following examples show how the security toolbox approach is flexible enough to accommodate new security requirements and applications scenarios.

One example is to sign a document and protect its source IP address at the same time. Such a need is found in some business applications such as dissemination of information to the public: stock prices, financial reports, public legal announcements, etc. This kind of protection is also needed in some B2B transactions. This approach provides two levels of protection: one at the document and the other at the address of the source. This will increase the confidence of the user because the probability of exploiting two layers is less than that of exploiting a single layer.

No existing security protocol can address this need. A user has to use IPsec at the IP layer and another protocol at the application level to sign the document or the transaction. However in our approach the toolbox can generate two security agent programs: one to provide the source IP protection and the other to sign the document, based on the policy that is described in the template.

Similar to that is the example of remote user's applications. Either at home or at a branch office, it is obvious that source IP addresses have to be protected against spoofing especially when workstations are off-line. In addition to that, applications also need a document-level or transaction-level encryption or integrity protection.

Another example is to modify the SSL handshake; SSL generates all its shared keys from a 48 byte pre-master secret. The SSL strength is based on the strength of the public key encryption applied on this pre-master secret. This is a good approach in the case of lengthy communication sessions. However, some applications need to exchange very short and highly confidential messages, such as military signals. In such cases, applying public key cryptography to the message directly provides stronger security for the transactions without much loss in performance.

7. CONCLUSIONS

In our approach, we group all the security tools in a box and place them next to the TCP/IP stack in such a way that they can interact with every one of its layers. In addition, we propose to place templates on top of the toolbox, thereby enabling application developers to set up many security protocols, test them, and deploy them rapidly.

This CBSE approach is superior in many aspects to the traditional approach involving monolithic implementation of one complete security suite. The idea of having a set of tools helps in improving security systems in: performance, efficiency, component reusability, ease of testing, flexibility, and scalability. Given that we are using a component-based approach to engineer security protocols [5, 6], we believe that the development, testing, and deployment time can be significantly reduced. The template mechanism that must be provided as part of this approach makes it easy to design a template or set of templates that produce a complete and rigorous protocol.

Acknowledgment

The research of Ibrahim Abdullah was sponsored by King Abdulaziz University and by the Ministry of Higher Education, Saudi Arabia

REFERENCES

[1] Rescorla, E. SSL and TLS Designing and Building Secure Systems, Addison Wesley, 2002.

[2] Viega, J. Messier, M. and Chandra P. Network Security with Open SSL, O'Reilly 2002.

[3] Hayden, M. The Ensemble System. PhD thesis, Cornell University, Jan. 1998.

[4] R. Morris, E. Kohler, J. Jannotti, and M Kasshoek "The Click router," Proc. *17th ACM Symposium on Operating Sys. Principles*, Dec 1999, pp. 217-231.

[5] Tran, V. and Liu, B. "Application of CBSE to projects with Evolving Requirements- A Lesson-Learned," *Proc. Sixth Asia Pacific Conference in Software Engineering*, Dec. 1999, pp. 28-37.

[6] Waddington, D. and Viswanathan, R. "Interaction points: exploiting operating system mechanisms for inter-component communications," *ACM SIGOPS Operating Systems Review*, Vol. 36, Issue 2, April 2002, pp. 19-35

[7] Sportt, D. "Componentization the Enterprise Application Packages." *Communication of the ACM*, Vol. 43, No. 4, April 2000, pp. 63-69.

[8] Oppliger, R. "Security at the Internet Layer." *IEEE Computer*, Vol. 31, No. 9, Sep. 1998, pp. 43-47.

[9] Barbieri, R. Bruschi, D. and Rosti, E. "Voice over IPsec: analysis and solution." *Proc. IEEE 18th Annual Computer Security Application Conference*, Dec 2002, pp. 261-269

[10] Decasper, D. Ditta, Z. Parulkar, G. Plattner, B. "Router Plgins: A Software Architecture for next-generation routers," IEEE/ACM Transactions on networking, Feb 2002, pp. 2-15.

[11] Nikander, P. and Karila, A. "A Java Bean Component Archit. for Cryptographic Protocols," *Proc. 7th Usenix Security Symposium*, 1997.

[12] Wong, G. Hiltunen, M. and Schlichting, R. "A configurable and Extensible Transport Protocol," *Proc. IEEE INFOCOM Twentieth Annual Joint Conference of the IEEE Computer and Comm. Societies*. Vol. 1, 2001, pp. 319-328.

[13] O'Malley, S. Peterson, L. "A Dynamic Network Architecture," *Proc. ACM Transactions on Computer System*, Vol. 10, No. 2, May 1992, pp. 110-143.

[14] Rensesse, R.Briman, K. Friedman, R. Hayden, M. and Karr, D. "A Framework for Protocol Composition in Horus," *Proc. Annual ACM Symposium on Principles of Distributed computing*, 1995, pp. 80-89.

[15] Jung, M. and Biersack, E. "A Component-Based Architecture for Software Communication Systems," *Proc. IEEE ECBS*, Edinburgh, Scotland, April 2000.

[16] Hiltunen, M. Jaiprakash, S. and Schlichting, R. "Fine-Grain Configurability for Secure Comm.," Technical Report 00-05, Department of Computer Science, Univ. of Arizona, Tucson, AZ, Jun 2000

[17] http://www.cs.arizona.edu/cactus/

[18] Hutchinson, N. and Peterson, L. "Design of the x-kernel." *ACM Symposium proceedings on Comm. Architectures and Protocols*, 1988, pp 65-75.

[19] Hutchinson, N. and Peterson, L. "The x-kernel: an architecture for implementing network protocols" IEEE Transactions on Software Engineering, Vol. 17, Issue 1, 1991, pp 64-76.

[20] Bhatti, N. and Schlichting, R. "A System for Constructing Configurable High-Level Protocols" *Proceeding of conference on application, technologies, architectures, and protocols for computer communications*, 1995, pp. 138-150.

[21] Bhatti, N. Hiltunen, M. Schlichting, R. and Chiu, W. "COYOTE A System for Constructing Fine-Grain Configurable Communication Services," ACM transactions on Computer Systems, Vol. 16, Issue 4, Nov 1998, pp. 321-366.

[22] Tschudin, C. "Flexible Protocol Stacks," *Proceeding of the ACM conference on Communications architecture & protocols*, 1991, pp. 197-205.

[23] McDaiel, P. Prakash, A. and Honeyman, P. "Antigone: A Flexible Framework for Secure Group Communication," *Proceeding of 8th USENIX UNIX Security Symposium*, August 1999, pp. 99-114.

[24] McDaiel, P. and Prakash, A. "Ismene: Provisioning and Policy Reconciliation in Secure Group Communication, Electrical Engineering and Computer Science, University of Michigan, Technical Report CSE-TR-438-00, December 2000.

[25] Hiltunen, M. and Schlichting, R. "A Configurable Membership services," IEEE transactions on Computers, Vol. 47, Issue 5, May 1998, pp. 537-586.

[26] Hiltunen, M. Schlichting, R. and Wong, G. "Implementing Integrated Fine-Grain Customizable QoS using Cactus" The 29th Annual International Symposium on Fault-Tolerant Computing (Fast Abstract), Madison, WI, June 1999, 59--60.

[27] Renesse, R. Birman, K. and Maffeis, S. "Horus: A Flexible Group Communication System," Comm. of the ACM, Vol. 39, Issue 4, April 1996, pp. 76-83.

[28] Hiltunen, M. and Schlichting, R. "The Cactus Approach to Building Configurable Middleware Services," *Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000)*, Nuremberg, Germany, October 2000.

[29] Braga, A. Dahad, R. and Rubira, C. "Composing Cryptographic Services: A Comparison of Six Cryptographic APIs," Technical report IC-99-05, State University of Campinas, Institute of Computing, 1999.

[30] Abdullah, I. and Menascé, D. "Protocol Specification and Automatic Implementation Using XML and CBSE," Proc. of the International Conference on Communications, Internet and Information Technology, Scottsdale, AZ, 2003.