SPEAR II
The Security Protocol Engineering and Analysis Resource

Elton Saul and Andrew Hutchison

Data Network Architectures Laboratory
Department of Computer Science
University of Cape Town
Rondebosch, 7701
South Africa

E-Mail: {esaul, hutch}@cs.uct.ac.za

ABSTRACT

Multi-dimensional security protocol engineering is effective in creating cryptographic protocols since it encompasses a variety of analysis techniques, thereby providing a higher security confidence than individual approaches. SPEAR, the Security Protocol Engineering and Analysis Resource, was a protocol engineering tool which focused on cryptographic protocols, with the specific aims of enabling secure and efficient protocol designs and support for the production process of implementing security protocols. The SPEAR II tool is a continuation of the highly successful SPEAR project and aims to build on the foundation laid by SPEAR. SPEAR II provides more advanced multi-dimensional support than SPEAR, enabling protocol specification via a graphical user interface, automated security analysis that applies a number of well-known analysis methods, performance reporting and evaluation, meta-execution and automated code generation.

1 INTRODUCTION

The use of open and unreliable computer networks is rapidly increasing as more companies and individuals connect to local and global networks. Since messages that are sent across open networks can be intercepted and manipulated by unknown entities, the security of these networked systems is crucial to protect the interests of all of its users.

One of the most important characteristics of a networked system is the distributed nature of the communicating entities. A protocol is a set of rules that is used to define an exchange of messages between two or more of these entities. In particular, cryptographic protocols make use of security techniques to achieve goals such as confidentiality, authentication, integrity and non-repudiation.

The fact that strong cryptographic algorithms exist does not guarantee the security of a communications system [26]. It is widely recognized that the engineering of security protocols is a very challenging task since protocols which appear secure can contain subtle flaws and vulnerabilities that attackers can exploit [2]. The odds definitely favour the attacker since defenders have to protect a system against every possible vulnerability.

SPEAR II is a security protocol engineering and analysis resource that builds on the work of the original SPEAR project that was conducted in 1997 [7]. The aim of SPEAR II is to provide a multi-dimensional framework which will enable secure and efficient security protocol designs and support for the 'production' process of implementing security protocols. The SPEAR II tool combines formal protocol specification, security and performance analysis, meta-execution and automatic code generation into one integrated and easy-to-use graphical interface.

This paper describes the SPEAR II tool that is currently being developed at the University of Cape Town. Section 2 gives a brief introduction to the field of security protocol engineering and analysis. In Section 3 we mention the primary goals of SPEAR II and then in Section 4 we elaborate on the modules of which SPEAR II comprises. The paper then briefly compares SPEAR II with the original SPEAR project in Section 5 and then concludes in Section 6.

2 SECURITY PROTOCOL ENGINEERING AND ANALYSIS

Designing a security protocol that fulfils its intended functions and degree of security is notoriously difficult [1]. Some examples of well-known protocols that have been found to be insecure include Microsoft's PPTP protocol [24], an early version of Netscape's SSL protocol [2] and the CCITT X.509 protocol [1]. As a result of this fact, recent years have witnessed significant efforts directed at developing methods to facilitate the design and analysis of cryptographic protocols [22].

Being able to clearly specify a security protocol forms the basis for further analysis and implementation. There are already systems which are used to design protocols in general, such as Message Sequence Charts (MSCs) [19] and the Specification and Description Language (SDL) [25]. SDL is widely used to describe communicating systems such as telecommunication protocols, while MSCs capture the exchange of protocol messages at a higher level than SDL, the central focus being on the exchange and proper sequencing of messages. The Common Authentication Protocol Specification Language (CAPSL) [9] is a high-level language whose goal is to permit a security protocol to be specified once in a form that can act as as an interface to any type of analysis tool or technique. Techniques to specify guidelines for security protocol design and modelling have also been presented [2, 21, 18, 3].

The development of cryptographic logics to analyze security protocols has provided one technique for ensuring the correctness of security protocols. One of the primary reasons for using security logics is to determine whether a given protocol achieves its design goals. Another use is to help eliminate protocol and message-field redundancy. Analysis using logics was first popularized in 1989 by the BAN logic [1]. BAN spawned a family of related logics, two well-known members being GNY [17] and SVO [30]. These and other logic systems have been used to reveal flaws in protocols that were previously accepted as correct [1, 17].

The issue of security protocol efficiency has been given rather low priority over recent years. One possible explanation is that since cryptographic protocols tend to involve few messages, optimization is not seen as an urgent requirement. However, determining whether a given protocol is optimal can serve as a valuable reference for designers. Foundational work has recently been conducted on how to obtain the lower bound on the number of messages and rounds required for network authentication protocols [16]. Using these techniques it is possible to provide a more flexible authentication system that varies the number of messages or communication rounds required depending on the network link speed or quality.

Replay attacks are a very powerful mechanism that can be employed to compromise security protocols. The most general definition of a replay attack is just any reuse of a past or current message that may have been manipulated [15]. A taxonomy of replay attacks has been developed that categorizes replay attacks in terms of message origin and destination [29]. Security logics and protocol analysis methods exist that are able to represent almost all possible replay attacks. However, to date only the NRL protocol analyzer appears to be generally capable of detecting all types of replays. Techniques and protocol design principals have also been formulated with the aim of developing protocols that are resistant to replay attacks [5, 18, 8].

Having cryptographically strong primitives and a theoretically correct protocol does not guarantee the development of a secure cryptosystem once the design has been completed. The mapping from the design phase to the implementation phase is often the most error-prone gap to breach [27]. For example, some systems don't ensure that plaintext is destroyed after its encrypted. Others use temporary files to protect against data loss during a system crash or virtual memory to increase the available memory; resulting in possible plaintext lying around on secondary storage. Encoding messages to be sent to another entity must also be carried out correctly in order for decoding to proceed without any errors – and this is often a difficult task.

Tools such as the NRL Protocol Analyzer [23], the Interrogator [14] and the Higher Order Logic (HOL) based cryptographic tool [10] have been developed to aid in analyzing security protocols. The Interrogator is a Prolog based program that searches for security vulnerabilities in network protocols used for automatic cryptographic key distribution, while the NRL Protocol Analyzer can be used to assist in either the verification of security properties or in the detection of security flaws in authentication and key distribution protocols. The HOL based cryptographic tool allows for the automated proof of authentication properties of security protocols. The tool uses an extended version of the GNY logic and was able to aid in finding and correcting errors in the Kerberos protocol.

3 SCOPE AND GOALS

The aim of SPEAR II is to allow for the specification of cryptographic protocols in such a way so as to distil the critical issues and present the user with a higher-level design overview. This approach gives users the freedom to explore the fundamental concepts of security in a controlled and expressive environment without having to actually carry out an implementation or make use of multiple tools to analyze the different facets of a protocol. The consolidation of different analysis and engineering techniques into a single application will also help to provide a single tool solution for the task of cryptographic protocol construction and design, filling an important gap in the area of security CASE tools.

The SPEAR II tool will comprise of five distinct but interconnected areas:

1. Security protocol *design* using a graphical user interface.
2. Automated *security analysis* that applies a number of well-known analysis methods.
3. *Performance evaluation and reporting* that helps to determine the efficiency of the protocol.
4. Automatic *code generation*, with the output languages being Java and SDL.
5. *Meta-execution* of security protocol runs.

Besides these existing five areas, we also foresee adding additional features that will allow a protocol conforming to a specific security model to be produced, for example a fail-stop or fail-safe protocol [18].

4 THE MULTI-DIMENSIONAL FRAMEWORK

The scope for a multi-dimensional framework that covers all the aspects of security protocol design, analysis and implementation is enormous. As a result of this fact, only the critical areas of this proposed framework have been identified. The possible functionality that each dimension should provide is described in the following sections.

4.1 SECURITY PROTOCOL DESIGN

Any CASE tool that is concerned with designing protocols should provide an interface that facilitates the rapid specification of a protocol, but at the same time is flexible enough to accommodate new types of protocols and security methods. To define a security protocol, the following steps are necessary:

1. The entities (principals) that are involved in the protocol must be declared.
2. The message passing specification must be clearly represented and must also accurately describe the working of the protocol. A graphical notation, such as MSCs, could be used to describe the message flow at a high level, while the message structure could be represented using a hierarchical tree structure.

Secondary to these requirements, the following steps can be carried out:

- Items which are to be used in the protocol can be extracted from the message passing specification. The underlying structure of these items must then be clearly specified using a representation such as ASN.1 [28].
- The external functions that are to be applied to the message components may be defined. The linkage between these external functions and generated source code must also be clearly stated.
- Communications settings such as transport protocols and instance timeouts can also be specified.
- Information specific to modal logic analysis may be declared. This can include initial beliefs and possessions of principals and preconditions of formulae in the message passing specification.

The goal of the design module is to provide an easy-to-use and powerful interface for the specification of security protocols. Components of the GUI must be written to ensure that compulsory steps in the specification process are clear and simple to complete. The importance and relevance of information that is related to the analysis and code generation phases must be clearly indicated and provided when necessary.

4.2 SECURITY ANALYSIS

The security of a protocol refers to the secrecy of the actual message exchange, the applicability of cryptographic methods, the possible attacks to which the protocol may be susceptible and the type and quality of data sent across insecure channels. Thus, the security module of the framework should provide a number of facilities:

1. The progression of beliefs and growth of possession sets during the run of a protocol should be analyzed by using a security logic system such

as GNY with possible extensions to cater for advanced crytographic tokens such as digital signatures or digital certificates.

2. Security analysis should be able to determine the degree of redundancy in the protocol, since message components that don't add any extra security or information can be eliminated. Also, keys that are received but not used could also be flagged as redundant.

3. The type of authentication provided by the protocol should be determined during security analysis. Thus the protocol designer could be certain of whether the authentication provided by the protocol is one-way or n-way, depending on the number of principals involved.

4. The ability to compare and indicate the information sent across an open channel as ciphertext or plaintext should be available. Some protocols send a portion of a message as cleartext. This can be especially dangerous if the same item is sent both in an encrypted and then a decrypted form during the protocol, since certain attacks can be applied to obtain the encryption key.

5. The subject of fail-safe protocols [18] has been an area of active research. The ability to determine whether a given protocol is fail-safe will lead to greater confidence in the security of a protocol. Thus, fail-safe analysis should be a vital component of the security analysis module.

6. The security of a protocol that is implemented using single sessions can be totally different to one that allows concurrent sessions to be run. For example, subtle parallel session replay attacks can be used by an attacker to comprise a concurrently executing protocol. The security analyzer should attempt to examine and enumerate the possible replay attacks which can be carried out against the protocol being tested.

Given the complexity of security protocols and the environment in which they operate, it is impossible to completely guarantee that a given protocol is totally secure. However, the security module can provide an increased degree of confidence in a given protocol, the knowledge that certain classes of attacks are not possible and the certainty that specific flaws are not exhibited. This information alone is a valuable service to provide in the security arena.

4.3 PERFORMANCE ANALYSIS

The security of cryptographic protocols is vital, however the performance of these protocols under operational conditions will eventually decide their usefulness. If a protocol is totally secure but requires hours of processing time and megabytes of network traffic to implement, then it is unlikely to be of great commercial benefit. In order to determine the ability of a protocol to perform adequately, the performance of the protocol under differing conditions and stresses must be determined.

Message-passing information regarding the protocol can be automatically generated to create a general estimate of its efficiency. This information could include statistics such as the number and size of messages and the number of connections that the protocol requires. If a designer is refining his protocol, then this information could be especially useful for comparison purposes. Another useful piece of information would be determining whether a given protocol is optimal or not in terms of the number of messages and rounds. Techniques have been developed which allow us to compare the number of rounds and bounds in a protocol to the possible optimum number in a given cryptoprotocol class [16].

A tool that implements performance analysis should also carry out some sort of examination to determine how well the protocol would function under concurrent operating conditions. With automatic generation of multiple clients the operation of the protocol in concurrent conditions can be gauged. This analysis can determine whether more servers are necessary and how many would be required to deal with predicted traffic.

During performance analysis it would also be useful to compare the impact of different algorithms and the way in which they are used in the protocol. For example, if an entire message was encrypted using a public key cryptosystem such as RSA, then it would take significantly longer than encrypting the message with a symmetric key cryptosystem such as DES.

Specialized tools exist that are specifically designed to perform thorough performance analysis of distributed protocols. Some of these accept input in a standard modelling representation such as Petri nets [6], SDL or Estelle. Programs of this nature include Geode [31] and SPECS II [12] for analyzing SDL models, DaNAMiCS [13] and DNANet [4] for analyzing Petri net models, and EDT [11] for examining Estelle specifications. Thus, the performance analyzer should also be able to produce protocol models that can be used as input to tools of this nature so that a more thorough analysis can be carried out if the designer so desires.

The performance of any protocol will be one of the important factors in determining its viability in the commercial sector. Thus, the performance module is an essential part of a multi-dimensional framework for supporting the development of cryptographic protocols.

4.4 META-EXECUTION

Meta-execution of a protocol makes a test-bed available to the designer where the protocol that he has designed can be tested without possibly having to compromise a “live” system. The environment wherein the protocol is tested should be capable of simulating protocol runs that take place under extreme conditions. For example, it should be possible to simulate the failure of servers and different message replay strategies. This type of execution platform will also allow protocols to be tested for future classes of attacks.

The ability to model attacks is related to the security analysis module, but in this case the attacks are modelled interactively. A flexible environment is needed to simulate attacks as it is difficult to pre-empt certain types of attacks without viewing the functioning of the protocol and how it reacts to various influences. When defining protocols statically, it is difficult to visualize the interaction of principals since no notion of state exists. Thus, facilities should be provided that allow a protocol designer to view the actual progression of the protocol and the manner in which the principals involved interact with each other.

The meta-execution module can also serve as a valuable training and educational resource since it will be able to give users insight into how a protocol functions, as well as help demonstrate attacks. For this reason the module should be configurable and allow users to set up previously simulated attacks and scenarios for demonstrations or continued research and analysis.

Meta-execution allows subtle aspects of protocols to become apparent while at the same time allowing various scenarios to be simulated. It also allows for experimentation in order to gain insight into possible side-effects and security flaws that reside within security protocols. Thus, a meta-execution module should form an integral part of the multi-dimensional framework that the SPEAR II tool will realize.

4.5 SOURCE CODE GENERATION

The code generation module should generate fully functioning source code which can be used as the actual client and server software stubs. This removes the networking programming and most, if not all, of the protocol coding. The source code that is generated should be as portable as possible since protocols are often implemented across heterogeneous networks. It is also advisable to generate source code as output since this gives protocol designers greater flexibility and negates the possibilities of back-doors that could exist in binary executables.

A tool implementing the framework should provide an extensive array of security protocol functions and algorithms, such as DES, RSA, MD5, key generation and nonce generation. The libraries that are written for the different output languages should follow a clear and accepted interface that already has the acceptance of the security community, such as GSS-API interface [20].

Since a large portion of errors usually result from implementing a security protocol, this code generation module will fill an important gap in protocol engineering by allowing for mundane protocol coding to be automated.

5 COMPARISON TO THE ORIGINAL SPEAR PROJECT

The original SPEAR application was developed as an Honours project by Bekmann and de Goede to facilitate in the design and analysis of cryptographic protocols. SPEAR proposed a unifying multi-dimensional framework that catered for the following:

- Protocol design using an intuitive GUI to describe the protocol in an MSC-type syntax.
- Automated security analysis using the BAN modal logic.
- Java code generation for all principals involved in a protocol.
- Controlled meta-execution of Java source code.
- Performance analysis using a custom rounds analyzer.

The emphasis of the SPEAR tool was more on design and as such its usefulness to test and implement existing protocols was limited. Both designers of SPEAR agreed on the fact that it only implemented a subset of their proposed framework and that further work was required in order to implement the entire framework. As a result, the SPEAR II project was initiated. Elaborating on all the diffences between the SPEAR projects is beyond the scope of this paper, however we list some of the primary ones below:

- Specifying complex protocols is tedious using the SPEAR GUI since the interface does not easily lend itself to constructing complex messages.

SPEAR II will allow the structure of a message component to be specified using the industry-standard ASN.1 notation and a hierarchical tree structure describing the messages.

- SPEAR II will produce security protocol code in Java and SDL. The program source code that is output will be able to have embedded hooks that will allow for performance evaluation and reporting, while the SDL modelling code will make it possible to analyze the protocol using a third-party tool.

- The source code produced by SPEAR does not marshall data for transmission over the network in a standardized manner. Since SPEAR II will be using ASN.1 to represent the structure of message components, standardised encoding rules such as BER or DER could be used [28].

- SPEAR offers limited cryptographic analysis capabilities since it employs the BAN logic which is only suited to analysing a restricted class of cryptographic protocols. A logic such as GNY is substantially more powerful and expressive than BAN. Thus, SPEAR II will make use of a customized version of GNY that contains extensions to cater for digital signatures and certificates, further expanding the class of protocols that can be analysed.

The SPEAR tool was essentially a "proof of concept" to demonstrate the viability of a multi-dimensional engineering framework. SPEAR II will build on this foundation and refine the SPEAR concept by building a more complete tool in line with the principles laid down by SPEAR, thus dramatically expanding the functionality that SPEAR offered.

6 CONCLUSION

Cryptographic protocols are crucial to the increasing use of computer networks for commercial and private communication. For this reason it is imperative that during their design such protocols are systematically scrutinized and refined using the most appropriate techniques. SPEAR II is a design environment which supports protocol specification in such a way that security and performance analysis, meta-execution and automatic code generation of a cryptographic protocol can be conducted. Thus, the intent of SPEAR II is to fulfil a useful and important function as an integrator and enabler in the discipline of security protocol design.

REFERENCES

[1] M. Abadi, M. Burrows, and R. Needham. A Logic of Authentication. In *Proceedings of the Royal Society*, Series A, 426, 1871, pages 233 – 271, December 1989.

[2] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6 – 15, January 1996.

[3] R. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11):32 – 40, November 1994.

[4] A. Attieh, M.C. Brady, P.S. Kritzinger, and W.J. Knottenbelt. Functional and Temporal Analysis of Concurrent Systems. In *Protocol Performance Workshop, held in conjunction with the* 16th International Conference on the Theory and Application of Petri nets, pages 79 – 96, Turin, Italy, June 1995.

[5] T. Aura. Strategies against replay attacks. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 59–68, Rockport, MA, June 1997. IEEE Computer Society Press.

[6] F. Bause and P.S. Kritzinger. *Introduction to Stochastic Petri Net Theory*. Advanced Studies in Computer Science. Vieweg Verlag, 1995. ISBN 3-528-05535-9.

[7] J.P. Beckmann, P. De Goede, and A.C.M. Hutchison. SPEAR: Security Protocol Engineering and Analysis Resources. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, September 1997.

[8] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of a Family of Attack Resistent Protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679 – 693, June 1993.

[9] S. Brackin, C. Meadows, and J. Millen. CAPSL Interface for the NRL Protocol Analyzer. In *IEEE Symposium on Application-Specific Systems and Software Engineering Technology '99*, Dallas, March 1999.

[10] S.H. Brackin. A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In *Proceedings of IEEE Computer Security Foundations Workshop IX*, June 1996. County Kerry, Ireland.

[11] S. Budkowski. Estelle Development Toolset (EDT). *Computer Networks and ISDN Systems*, 25:63 – 82, 1992.

[12] M. Bütow, P.S. Kritzinger, M. Mestern, and C. Schapiro. Performance Modelling with the Specification Language SDL. In *FORTE-PSTV96: XVth International Symposium on Protocol Specification, Testing and Verification*, pages 213 – 225, Kaiserslautern, Germany, 1996.

[13] B. Changuion, I. Davies, P.S. Kritzinger, and M.A. Nelte. DaNAMiCS - Modelling Concurrent Systems. In *First Annual South African Telecommunications, Networks and Applications Conference*, pages 606 – 610, Cape Town, September 1998.

[14] S.C. Clark, S.B. Freedman, and J.K. Millen. The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[15] L. Gong. Variations on the Themes of Message Freshness and Replay. In *Proceedings of the IEEE Computer Security Foundations Workshop VI*, pages 131 – 136, Franconia, New Hampshire, June 1993.

[16] L. Gong. Efficient Network Authentication Protocols: Lower Bounds and Optimal Implementations. *Distributed Computing*, 9(3):131 – 145, 1995.

[17] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 234 – 248, Oakland, California, 1990.

[18] L. Gong and P.F. Syverson. Fail-Stop Protocols: An Approach to Designing Secure Protocols. In *The Fifth International Working Conference on Dependable Computing for Critical Applications*, pages 44 – 55. Springer-Verlag, September 1995.

[19] ITU-TS, Geneva. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, 1996.

[20] J. Linn. *RFC 2078: Generic Security Service Application Program Interface, Version 2*. OpenVision Technologies, 1997.

[21] J.D. McLean. The Specification and Modeling of Computer Security. *Computer*, 23(1), January 1990.

[22] C.A. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology - Asiacrypt '94*, pages 133 – 150. Springer-Verlag, 1995.

[23] C.A. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113 – 131, February 1996.

[24] Mudge and B. Schneier. Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP). Counterpane Systems and L0pht Heavy Industries, 1998.

[25] O. Faergemand and A. Olsen. Introduction to SDL-92. *Computer Networks and ISDN Systems 26*, 8(1):18 – 36, 1994.

[26] B. Schneier. Why Cryptography is Harder than it Looks. 1997.

[27] B. Schneier. Security Pitfalls in Cryptography. Counterpane Systems, 1998.

[28] D. Steedman. *ASN.1 - The Tutorial and Reference*. Technology Appraisals, 1990.

[29] P.F. Syverson. A Taxonomy of Replay Attacks. In *Proceedings of the Computer Security Foundations Workshop VII*, Franconia, New Hampshire, 1994. IEEE Computer Society Press.

[30] P.F. Syverson and P.C. van Oorschot. On Unifying Some Cryptographic Protocol Logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, California, May 1994. IEEE Computer Society Press.

[31] Verilog. *GEODE Reference Manual*, 1995.