The Weakest Precondition Protocol Analysis Environment

Alec Yasinsac Michael P. Runy
yasinsac@cs.fsu.edu runy@nortelnetworks.com
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530

Abstract

In this paper, we show how the formal methods are used in the Cryptographic Protocol Analysis Language Evaluation System (CPAL-ES) to analyze cryptographic protocols in a Windows NT environment. An editor and software library were developed to allow for the integration of CPAL-ES into a homogenous development and analysis environment. Additionally, CPAL-ES was updated and moved from an MS-DOS environment to a Windows NT environment. All of the tools mentioned were developed in order to facilitate future efforts by the analysis team in working with CPAL-ES. We demonstrate how the editor for CPAL was constructed and how it functions. Additionally, we show how the software library is maintained and how it functions. Finally, we show how CPAL-ES was converted from an MS-DOS environment to a Windows NT environment.

1. Introduction.

Formal methods are used in a wide variety of environments to verify that complex systems accomplish their intended functionality. One area where formal methods received much attention is in the area of security protocol verification [9], [8], [11].

Analysis of cryptographic protocols is a field that is growing in importance by the day. The widespread use of networking requires a method of securing communications over public channels. Cryptographic protocols were developed to accomplish this. Since these protocols are supposed to secure communications, we must make sure that the protocols themselves are sound and not vulnerable to attack. This necessitates analysis of these protocols.

The fundamental technology that allows protocols to protect our networks is encryption [17]. Reliable encryption is essential to many of these protocols and encryption can be done at different levels in the OSI Model, including Link-level, Network-level, and Transport-level encryption [10]. The level of encryption used depends on how much of the information transmitted you wish to keep secret and how much overhead you are willing to deal with to encrypt your information.

A common threat to communications is for one user to assume the identity of another user, called masquerading. Authentication techniques attempt to prevent masquerading. Encryption is integral to authentication and many of the network security protocols that exist today have the concept of authentication built into them.

Protocol verification is a very important part of the development of cryptographic protocols. We must know that the protocols are acting in the method that they are intended to act. Finding weaknesses such as the Denning and Saco flaw is a big part of this. Additionally, protocols must impart the necessary information to the principals without compromising the security of the communication. For example, they should be able to deliver the session keys to both principals involved in a communication without allowing those keys to be compromised.

2. Weakest Precondition Reasoning

Many methods for verifying security protocols have been developed to date including [4], [MCF87], [BAN88], [6], [14], [9], [12], [7], [8], [15], [11], [13], [1], [3] and many others. Yasinsac and Wulf first used weakest precondition reasoning in the evaluation of protocols [18], and they have since been shown to facilitate detection of flaws and inconsistencies in security protocols [16], [2]. Weakest precondition reasoning allows us to take a set of post conditions that we would like to result from a protocol run and find out what the weakest set of preconditions for those post conditions would be. The process takes the actions of a protocol into account as it is determining what the weakest preconditions for the given post condition are.

For example, P{S}Q is a logical statement, which can be evaluated to either true or false, where P is a set of preconditions, S is a protocol, and Q is the desired outcome expanded as a set of post conditions. So, we may fix a protocol and post condition Q, and then find a P, which if it is true when S begins executing, will guarantee that Q will be true after S is finished. The objective, then is to find a precondition, P, that when run through S will result in Q being true.

For security protocol analysis, weakest preconditions are used to find a desired precondition that will guarantee the correctness of a protocol. This is called the verification condition. This verification condition takes the form of a logical predicate that can be simplified to TRUE if a given protocol correctly executes.

Preconditions are formed from protocol assumptions, assertions, and actions. For example, in order to properly verify a protocol, we might have to assume that principal A holds the same session key as principal B. Actions can also represent a valid precondition. Actions such as retrieving a key from a key server or making sure that a nonce is fresh by verifying it with the other party to the communication may be necessary to a successful run of the protocol.

CPAL-ES evaluation consists of four steps:

1. Encode the protocol actions in CPAL
2. Derive and encode the protocol goals and assumptions
3. Generate the precondition
4. Simplify and analyze the result

The CPAL-ES system produces a verification condition for a given protocol. This verification condition is then simplified in an attempt to make the predicate easier to understand. By evaluating protocols using weakest precondition reasoning, we seek to find flaws that exist in these protocols. Additionally, even if no new flaws are found, a more thorough understanding of the workings of the protocol is gained. More detail about CPAL-ES can be found in [18] and [19].

The rest of this paper will show how an integrated workbench is being developed which allows for easier use of CPAL-ES. First, the development of a software library that allows for version control and parallel protocol development will be discussed. Second, the conversion of CPAL-ES from MS-DOS-based to Windows NT-based code will be covered, including the development of a front-end program designed to make execution of the CPAL-ES easier. Finally, the development of a syntax-directed editor for CPAL will be discussed along with further goals for the workbench. These steps comprise the majority of the work involved in creating a homogeneous

environment, depicted in Figure 1.

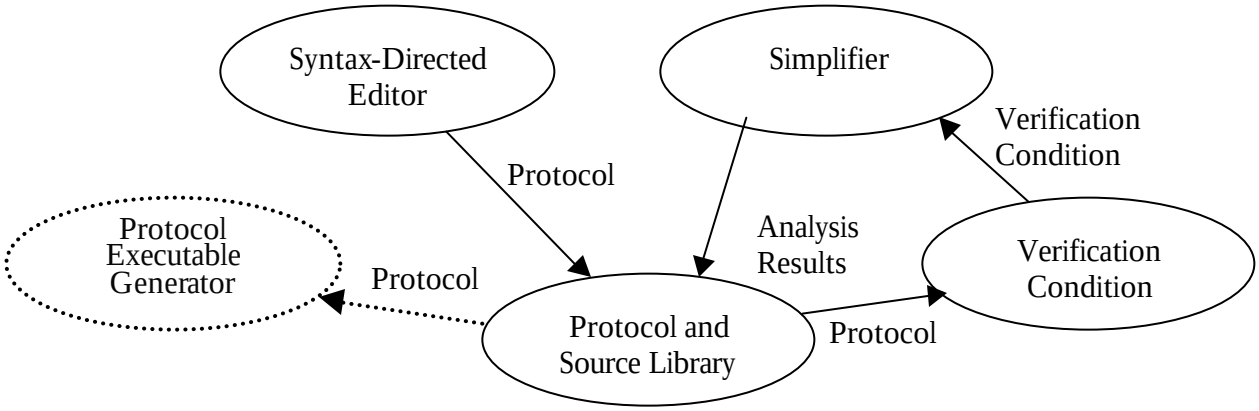


Figure 1

3. The Security Group Software Library

The first focus of developing the integrated workbench was to come up with a solution for version control of the code that was being developed. A software library in which various versions of the code being developed could be stored was the desired outcome. This library keeps track of the different versions of code that it contained. The tool would also be able to manage protocol versions so that we could improve protocols and then verify the improvements. Additionally, it would enable members of the security group to check out pieces of code in order to update or revise them. In doing so, it would ideally keep track of who made what changes and which version was the most current version of the code. The platform for the software is Windows NT.

The best solution that was obtained was public domain software named WinCVS. This software is an open source program that was developed by a group looking for a Windows front-end to the popular UNIX CVS program. It contains many of the features that we desired for our version control system. Other software such as QVCS and JCVS were evaluated but they were deemed ineffective for the job that we had in mind. WinCVS presented the best options and was actually designed to mix with other operating systems such as UNIX and the MacOS in addition to a Windows environment.

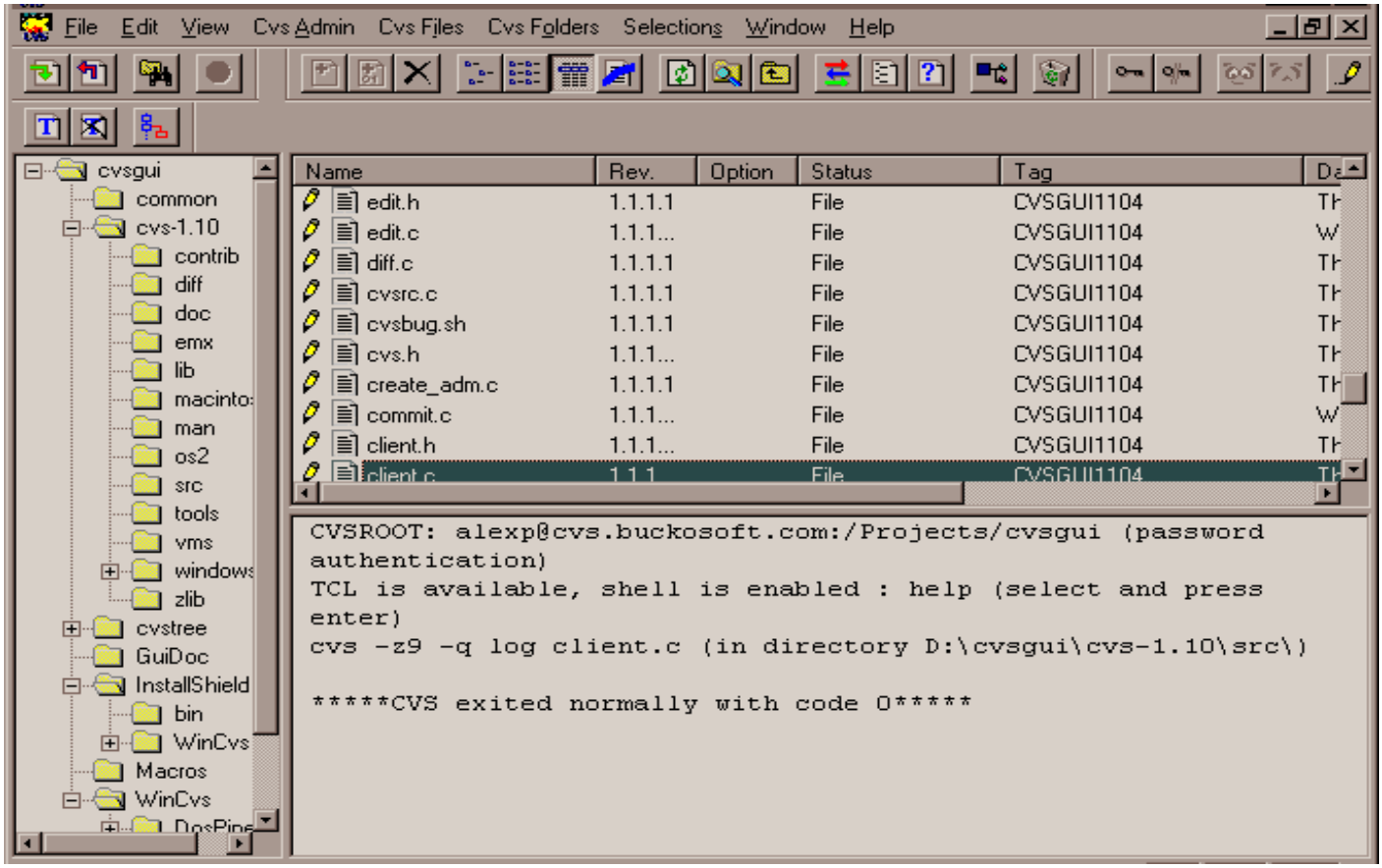
WinCVS is a menu-driven configuration management system. In using WinCVS, the first step is creating a "module" that contains all of the code that the user wants protected and monitored. The directory hierarchy that the user wants to be part of the library is imported into the CVS server. Once there, it can be checked out, viewed, or archived. A password file that is maintained on the server determines who can access the files that are being protected by WinCVS.

The figure below shows a sample main window and what it looks like when populated with a module. The buttons at the top of the screen allow the user to check out or check in files, lock them so no one else can use them, add labels to them, and compare them with previous versions of the file in addition to several other operations. The main window of the program shows the directory structure of the server on the left side along with the files that are contained within the currently selected project on the right-hand side. Below the file list, is a window that contains debug information on particular commands, as well as a command line where CVS commands can be entered without using the menu options. Additionally, information is displayed there when revision histories and file information is asked for.

The WinCVS software allows us to control access to source code for the various components of the security workbench. Anyone can create a module for code that they are working on and allow whomever they want to

have access to those files. Additionally, older revisions of code are stored so that they can be accessed later on if necessary. A wide variety of reports can also be created that show how revisions have been made and who made them. Command-line utilities are also provided that allow for automation of some configuration management tasks.

As an added bonus, WinCVS allow for what it calls branching. This is where several different versions of code can be developed along different "branches" simultaneously. Whenever the code is in working order, these branches can be merged back together into a single piece of code. The "branches" can also be maintained independently for an indefinite period of time.



4. CPAL-ES From DOS to Windows

The second focus of the development environment was to port CPAL-ES to Windows NT from the MS-DOS environment. This consisted of several steps, including the initial conversion to Windows-based code, conversion from that code into C++ code and the development of a Windows-based front-end for the CPAL Evaluation System. We briefly discuss the Windows front-end.

The Windows-based front-end for CPAL-ES was created using the Visual C++ development environment. A Microsoft Foundation Class (MFC) based program was developed that utilized the existing CPAL-ES program and allowed the user to input the protocol that they desired to be evaluated in a dialog box. The MFC libraries allow for use of standard Windows functions and portability of the code across various Windows platforms. This dialog box then launched the CPAL-ES application with the given protocol as input.

Using the Dialog classes that are part of the MFC library, the program creates a dialog box that contains an edit box which asks the user for the location of the protocol which they wish to evaluate. Upon entering the name of the file containing the protocol specification, the program then calls the CPAL-ES program with the filename as an argument. The CPAL-ES system uses the Windows console to display its temporary output and the files that are created for analysis are opened by the program so that the user can continue their evaluation of the protocol.

The controlling method, OnOk(), is called when the user presses the OK button in the dialog box. It first saves the information in the edit box that the user has input to a CString variable. This is the string class developed for use with MFC applications. This string is then converted to a form that can be passed to the CPAL-ES program. Following this, the ShellExecute() command is used to execute the CPAL-ES program with the parameter specified by the user. When its execution has completed, the console window closes and the three files that have been created by CPAL-ES are opened for evaluation.

This front-end program allows execution of CPAL-ES in a Windows environment. It makes no change to the functionality of the previously existing code, and allows one to use a command line interface if so desired. However, it does make it easier to use within the Windows context.

5. The CPAL Editor

The final step in creating the integrated work environment was the development of a syntax-directed editor for CPAL. This editor highlights syntax much like most Windows-based programming tools do. Additionally, it identifies syntax errors to the person using it and compiles the protocols if asked to do so. The editor was developed in order to facilitate the translation of protocols into CPAL and to allow people who will translate these protocols to do their job more quickly and efficiently with fewer mistakes.

The CPAL editor was designed as an SDI (Single Document Interface) application with MFC support. The framework for the editor was actually created by Microsoft Visual C++. Additional support was added after the basic framework was already in place to highlight the syntax and allow for compilation of the protocol being edited from within the editor itself.

The document/view architecture of MFC works by associating a particular view with each document. In this case, the CCPALEditorView class implements the functionality of the syntax highlighting and the compilation. CCPALEditorView is a child of the CRichEditView class. This class is one which is usable by all MFC applications in order to implement not only common editing functions such as cut and paste but also character formatting and other more complicated operations. Both the CRichEditView and the CEditView class deal with operations having to do with text editing. The use of the CRichEditView class was necessary in order to provide the coloration of the text.

The view is the actual editing window. The first thing that is done upon creation of the view is to execute the OnCreate() function of the CCPALEditorView class. This function, listed below, first creates a list of keywords that the editor will use for syntax highlighting. It does this through use of the GetTokenTypes() function. GetTokenTypes() retrieves the names of the keywords from a structure that is generated whenever the YACC grammar specifying the language is compiled. In this way, the grammar of the language can be changed without having to change the editor.

The keywords of the language are added to a static list by the AddKeywords() function. This list is then used as a basis of comparison while editing in order to effect the syntax coloring. Nonces and keys are also set up in the OnCreate() function. Basically, any nonce or key will be highlighted accordingly as it is typed in.

The remainder of the work is done throughout the CCPALEditorView class. The FormatTextRange() function is

the most important one as it does all of the syntax highlighting as the user types in the protocol. Shown below, the code first creates a buffer and then checks this buffer against the known types of keywords, comments, etc.. Whenever it finds one of these, it colors it accordingly using the designated color that we determined earlier in the code. It processes comments, nonces, keys, numbers, keywords, and constants all from within the same function.

This FormatTextRange() function is called whenever the text in the view is changed. The range on which the formatting is done is determined by the operation that is changing the text. When simply typing in text, the range is just the current cursor position. However, operations for cut and paste, etc. were added to insure that all of the text will remain formatted with the syntax coloring.

The other main addition to the simple editor view is the addition of a compile button in the toolbar. The button is a simple C (standing for Compile) that was added to the end of the toolbar. When the button is pressed, a message is passed which calls the OnCompile() function of the CCPALEditorView class. This function streams the text out to a temporary file which can is then passed to the parser created by LEX and YACC. Some modification of the standard error function created by LEX and YACC was necessary as well in order to have message boxes pop up which detail any syntax errors encountered in the code. Following successful compilation, the three output files created by CPAL-ES are opened for analysis. The code used to call the parser and implement the CPAL-ES are mostly unchanged from the original implementation of the program.

The files are all closed after their use which allows the ShellExecute() command to open the three documents that are created by the CPAL-ES. Additionally, the temporary file that was created in order to use as input to the parser is destroyed as it merely contains a text representation of what is already contained in the editor.

6. Conclusion

The widespread development and use of cryptographic protocols for security has brought with it a necessity to analyze these protocols, not only to find flaws but also to be able to understand the protocols and how they work more effectively. The use of logics and formal methods in this field were developed expressly to do just that. These logics are combined with weakest precondition reasoning in the CPAL-ES.

The development of a full-scale workbench for use with CPAL-ES is just beginning. These are only the first few steps in a process that will eventually lead to an integrated tool that enables users to easily enter protocol specifications in CPAL, run them through CPAL-ES, and analyze their results. This workbench will facilitate the development of more secure network protocols by exposing flaws that may be discovered or just by allowing a more in-depth understanding of what makes a good or bad protocol.

The tools that have been created so far: The CPAL Editor, and Front-end combined with the software engineering environment supplied by the QVCS program will allow continued development of this workbench. They are the first components in what will eventually be a much more robust tool. An additional tool is presently in development that will generate executable modules from CPAL protocol specifications. These executing protocols will be installed and analyzed in the Security and Assurance in Information Technology Laboratory at Florida State University.

7. Bibliography

[1] S. Brackin, "Automatically Detecting Most Vulnerabilities in Cryptographic Protocols", in The DARPA Information Survivability Conf and Exposition, Jan 00, V.1, pp 222-36

[2] Justin Childs, 'Evaluating the TLS Family of Protocols with Weakest Precondition Reasoning', Master's Thesis, Florida State University, Computer Science Dept, June 00

[3] G. Denker and J. Millen, "CAPSL Integrated Protocol Environment", in *Proceedings of the DARPA Information Survivability Conference and Expo 2000*, Vol. 1, pp. 207-21

[4] Dolev, D., and Yao, A.C. "On the Security of Public Key Protocols". IEEE Trans. Inf. Theory IT-29, 2(Mar. 1983), pp. 198-208. Also Stan-CS-81-854, May 1981, Stanford U.

[5] Gong, Li, Needham, Roger, and Yahalom, Raphael. "Reasoning about Belief in Cryptographic Protocols," in *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, 1990.

[6] R. A. Kemmerer, "Using Formal Methods to Analyze Encryption Protocols," IEEE Journal on Selected Areas in Communications, vol. 7, mo. 4, pp. 448-457, May 1989

[7] Gavin Lowe, "Breaking and Fixing the Needham-Schroeder Public Key Protocol Using FDR", In *Proceedings of TACAS*, Vol. 1055 of *LNCS*, pp 147-166, Springer-Verlag, 1996.

[8] Gavin Lowe, "Casper: A Compiler for the Analysis of Security Protocols", Journal of Computer Security, Volume 6, pp 53-84, 1998.

[9] Catherine Meadows, "Formal Verification of Cryptographic Protocols: A Survey," Advances in Cryptology - Asiacrypt '94, LNSC 917, Springer-Verlag, 1995, pp. 133-150

[10] Nessett, D. M. "Factors Affecting Distributed System Security," Lawrence Livermore National Laboratory, 1986.

[11] Lawrence C. Paulson, "Inductive analysis of the Internet protocol TLS", *ACM Transactions on Computer and System Security* 2 3 (1999), 332-351

[12] A. W. Roscoe, "The Theory and Practice of Concurrency", Prentice Hall, 1997

[13] Dawn Xiaodong Song, "Athena: A New Efficient Automatic Checker for Security Protocol Analysis", 12th IEEE CSFW, Jun 28-30, 99, Mordano, Italy

[14] P. Syverson, and P.C. van Oorshot, "On Unifying Some Cryptographic Protocol Logics", in Proc of 1994 IEEE Computer Society Symposium on Security and Privacy, May 16-18 1994

[15] F. Thayer, J.C. Herzog, and J.D. Guttman, "Strand Spaces: Why is a Security Protocol Correct?", In Proceedings of 1998 IEEE Symposium on Security and Privacy, 1998

[16] Brett Tjaden, "A Method for Examining Cryptographic Protocols" University of Virginia Dissertation, Jan 1997

[17] Walker, Stephen T. "Network Security: The Parts of The Sum," Trusted Information Systems, Inc., 1989.

[18] Alec Yasinsac and Wm. A. Wulf, "Using Weakest Preconditions to Evaluate Cryptographic Protocols", Cambridge Workshop on Security Protocols, March 1996

[19] Alec Yasinsac and Wm. A. Wulf, "A Framework for A Cryptographic Protocol Evaluation Workbench", to appear in the International Journal of Reliability, Quality and Safety Engineering (IJRQSE), Apr 2001