

GRASP: A Visualization Tool for Teaching Security Protocols

Dino Schweitzer, Leemon Baird, Michael Collins, Wayne Brown, Michael Sherman, *United States Air Force Academy*

Abstract – Security protocols are an important concept in teaching information security. Students need to understand both the sequence of passed information and computations, as well as the various attacks on them via eavesdroppers, forged messages, communication blocks, and message replays. A traditional approach to teaching protocols is to use a static diagram showing the transfer of messages between participants over time. This paper describes an interactive visualization tool that allows arbitrary protocols to be demonstrated visually in a user-controlled stepwise manner. The user can interactively change the protocol to see the effects of various attacks. The tool can be used in a classroom environment or as a laboratory exercise. The paper also describes sample scenarios for using the tool and the experience to date of use in an undergraduate information security class.

Index terms – Information assurance education, Classroom visualization, Security protocol visualization

I. INTRODUCTION

Information security education is becoming increasingly important in today's environment of malicious software, network attacks, and cyber-crime. Future professionals must understand the nature of the threats, the vulnerabilities in software and networks, and the mechanisms for protecting computer and information resources. Many undergraduate programs are developing curricula and teaching security-related courses to address this knowledge requirement.

At the United States Air Force Academy, security topics are covered in several classes. All students are introduced to security concepts in the introductory course in computer science. Computer science majors get additional exposure to security issues in their advanced coursework such as Networks and Software Engineering. Specific courses in Computer Security and Information Warfare, Cryptography, and Network Security allow computer science majors to complete a concentration in information assurance which is designated on their academic records. As a result of the Academy's emphasis on security education and research, it has been designated as a Center of Academic Excellence in Information Assurance Education [1].

An important topic in teaching information security is secure protocols. Secure communication between two or more individuals requires a carefully scripted sequence of messages and computations. This sequence is often illustrated for students as a vertical timeline of message passing, identifying known information and computed values (see Figure 1). Attacks on the protocol, such as an eavesdropper, or someone sending false messages, can be added to the illustration as extra participants with additional message traffic. As attacks and countermeasures are added to the protocol, the illustration can become complex.

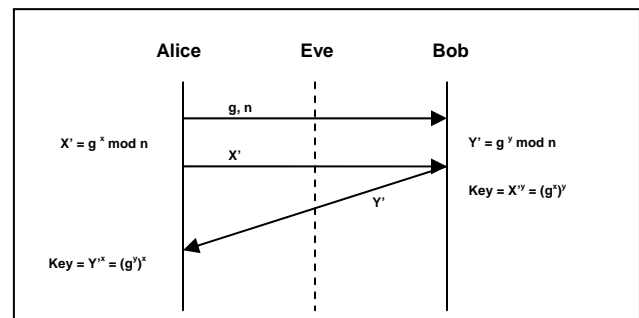


FIGURE 1. Traditional security protocol illustration.

To enhance student understanding of security protocols and to provide a means for demonstrating various protocol attacks and countermeasures, we developed an interactive protocol visualization tool known as GRASP (GRaphical Aid for Security Protocols). The purpose of this tool is to graphically illustrate security protocols and interactively allow changes to be made, such as attacks and countermeasures, to see the resulting effects. This paper describes the tool and our experience with using it to teach security protocols to undergraduate computer science majors.

II. PREVIOUS WORK

The majority of published work in information security education has been in curriculum development, classroom experience, and laboratory environments. Conferences such as the Colloquium for Information Systems Security Education (www.ncisse.org), and Information Security

Curriculum Development contain a great deal of information on how to integrate security topics into existing courses and programs, building academic programs, case studies, teaching techniques, sample exercises, and student competitions [2]. This information combined with an increasing number of available textbooks has provided access to a great deal of content for developing security courseware.

There is less documentation in the literature regarding interactive tools for teaching security topics. Some examples include virtual laboratories and networks that have been developed to provide “safe” environments for teaching security concepts [3]. An ambitious project to create a virtual world for teaching security concepts is CyberCIEGE [4]. User-controlled animation applets have been developed as part of interactive courseware for teaching information security concepts such as cryptography and buffer overflow [5][6].

Specifically, in the area of communication protocols, various visualization tools have been developed to assist in protocol analysis and network monitoring [7][8][9][10]. For teaching security protocols, Burger and Rothermel describe a general purpose simulation architecture with a visualization component [11]. Their tool, HiSAP, simulates two users with an intruder and authentication server. The students and/or teacher assume the roles of the players in the simulation and exchange messages to see if replay attacks can be successful. Elmqvist reports on a protocol visualization tool named ProtoViz that allows arbitrary protocols be entered in a specification language, and animated in a step-by-step fashion [12]. The visualization consists of animated message packets moving back and forth between actors.

The limited amount of previous work in visualization for security education results in few guidelines for effective tool design strategies. However, there is an abundance of work in the related area of algorithm visualization that does provide insights into successful approaches. Several studies have been conducted regarding the effectiveness of algorithm animations and the characteristics of such visualizations [13]. The experiences gained from these studies were used in this project.

III. DESIGN GOALS

The purpose of this project was to design an effective visualization tool for teaching security protocols that could be used in either the classroom or as a student exercise. The specific design strategies were derived from the authors’ experience with other classroom visualization tools as well as the documented literature in algorithm visualization. The primary design considerations are described.

A. General purpose

The tool should be usable for arbitrary security protocols, and not “hardcoded” to a few static cases. An approach similar to ProtoViz was implemented with the definition of a simple protocol specification language (described below) that allows an arbitrary number of actors and message passing. The language is slightly more complex than ProtoViz allowing additional functions such as message blocking and forging. When designing the language, the choice of commands to include was based on anticipated usage of the tool in the classroom. In other words, which commands were necessary to demonstrate the specific protocols that were currently being taught in the security classes. Another design consideration in the specification language was to make it very “human readable.” Succinctness in syntax was traded off for ease of understanding by tool users, especially students in the classroom.

B. High interactivity

A key lesson from algorithm animation studies is that the user should be able to control and modify the environment and parameters of the visualization as it is running. This allows for real-time “what if” scenarios to demonstrate cause-effect relationships. For protocol visualization, this means the ability to interactively change the protocol, the number of actors, and protocol attacks such as message forging to see the resulting effects. In GRASP, this is accomplished by allowing the user to edit the protocol commands “on the fly”, re-execute the protocol commands, and visually see the differences.

C. User control

At every point in the visualization, the user should have control over what step of the protocol is being executed. The user should be able to back up if something is unclear, or reset the protocol if necessary. This level of control extends beyond simple animation playback controls. Combined with the ability to change parameters, or in this case the protocol specification, this allows the user to interactively discover the effect of environment changes that may not have been anticipated by the tool designers. A static animation does not provide this flexibility.

D. Visual meaningfulness

The visual display should be easy to understand without superfluous animations or icons. The protocol language should be displayed along with the graphical representation highlighting the protocol step currently being executed. As stated earlier, the protocol

specification syntax is sufficiently “English-like” to be easy to understand. The temporal sequence of events should be intuitive to understand. The visual representation of the sequence of message passing and computations in GRASP is similar to the traditional diagram shown in Figure 1. Another approach could have been the use of animation to show messages moving between actors. However, this motion is irrelevant to the protocol concepts being communicated, and can be a distraction. The choice of a vertical timeline versus a horizontal one was partly to be consistent with traditional representations, and partly because it is more intuitive when scrolling to be moving up and down versus right to left.

IV. TOOL DESCRIPTION

GRASP is written in Java using standard Swing components. The original version was developed by a computer science student as an independent study project during fall semester 2005. Based on the experience with the tool, a modified version with an enhanced user interface was created in spring 2006. The basic layout of the tool is shown in Figure 2. It consists of three primary sections: an editing window for the text version of the protocol specification, a summary section showing each “actor” and what knowledge they have at any point in time, and a graphical timeline showing the actions of the protocol as the user steps through the protocol commands. Each of these sections will be described.



FIGURE 2. Basic GRASP layout.

A. Protocol Specification Window

This window is a fully editable text description of the protocol being visualized. Protocols can be created within the window and saved to a text file, or created from outside the program and loaded into GRASP. Protocol commands were chosen based on anticipated usage for classroom instruction. The protocol command syntax and a description of each command follows with the keywords shown in bold type:

actor(s) *actorName actorName...*

This specifies the list of participants for the protocol. GRASP will keep track of the list of variables that each participant knows at any point in time and there is a timeline associated with

each participant. This command is usually the first one in the protocol specification.

actorName knows *variable variable...*

Specifies what an actor knows. These are typically initial conditions for the protocol. A variable is any sequence of characters that contains no commas or spaces.

actorName computes *variable (variable,...) = description*

Actors can compute new values using the values they already know. The variables within the parentheses are items that the actor must know in order to perform the computation. GRASP performs a validation check and reports an error condition if the actor did not know all of the required variables. The optional description describes how the computation is performed.

actorName encrypts *variable {key}*

Encrypting variables uses another variable as the key. Both the variable and the key must be known by the actor before this command is executed. Once executed, the *variable{key}* string becomes like another variable that is known, can be sent, used in a computation, etc.

actorName decrypts *variable {key}*

Decrypting is the opposite of encrypting. The actor must know the *variable{key}* string as well as the key. If successfully executed, the decrypted *variable* is added to actor’s known list.

actorName sends *actorName variable...*

An actor can send one or more variables that they know to another actor.

actorName intercepts traffic from *actorName...*

This is the specification of an “eavesdropper” in the protocol. The eavesdropper can listen to items sent from a specified set of actors. Optionally, the second *actorName* parameter can be set to **All** indicating the eavesdropper hears all communication. The ability to eavesdrop can be started and/or stopped at any point in the protocol.

actorName blocks traffic from *actorName...*

This command allows a malicious actor to disrupt communication between actors. Similar to eavesdropping, it can start and stop at any point.

actorName forges *actorName sends actorName variable...*

This command represents a communication from an actor pretending to come from a different

actor. This is used to demonstrate how spoofing can affect a protocol.

variable description

Variable descriptions are used to assist in the visualization. The descriptive text is displayed in the visualization window when the mouse is held over the variable. This allows the user to see a more complete description of what the variable represents. The *description* can be any string.

- comment

Comment lines in the protocol are specified with a beginning dash.

Color coding for actors, italics for comments, and bold-face for commands are used in the protocol window to assist the user in understanding the protocol text. The window is fully editable and affects the other windows as commands are changed. Syntactically incorrect lines are treated as comments, and it is immediately apparent to the user when a line of text is not recognized as a valid command. Executable lines are numbered to allow correlation with the timeline window.

B. Actor Summary Window

The actor summary window displays a list of all variables “known” by each actor in the protocol. Variables can become known in one of five ways: through execution of the *knows* command, by having it sent to them from another actor, by computing it, decrypting it, or by intercepting it. The summary window is updated after each protocol command is executed. During execution some simple checks are performed based on the knowledge that each actor has. An actor must know a variable before they can send it, and they must know all variables used in a computation, decryption, or encryption. Failure of any of these checks results in an error condition being displayed.

C. Protocol Timeline Window

This window is the main focus of the visualization and is modeled after the traditional timeline used for describing protocols. As each step of the protocol is executed, arrows are used to show variables being sent from one actor to another. Computations are also shown in the timeline at the appropriate point. When variables are sent between actors, the send action is shown with a wide arrow pointing from the sender to the receiver. If values are intercepted by other actors, this is indicated with smaller arrows.

D. Executing Protocols

A set of command buttons above the editor window allows the user to execute the protocol once created/loaded. The *Next Cmd* button executes a single step in the protocol and updates the actors’ window and timeline. The ability to *Back up* the execution one step allows the user to review what just happened. At each step the appropriate line in the editor window is highlighted to show what step is being executed, and the graphics timeline is updated. *Reset* and *Finish* allow the user to quickly go the beginning/end of the protocol with the other windows updated accordingly.

V. SAMPLE SCENARIOS

Following are two examples of using GRASP in the classroom to describe security protocols.

A. Diffie-Hellman

As a simple example of the protocol specification language, Figure 3 shows the GRASP commands that would implement a version of the Diffie-Hellman key exchange protocol that was illustrated in the traditional way in Figure 1 (note the exception that *g* and *n* are publicly known so are not shown as sent from Alice to Bob as in Figure 1).

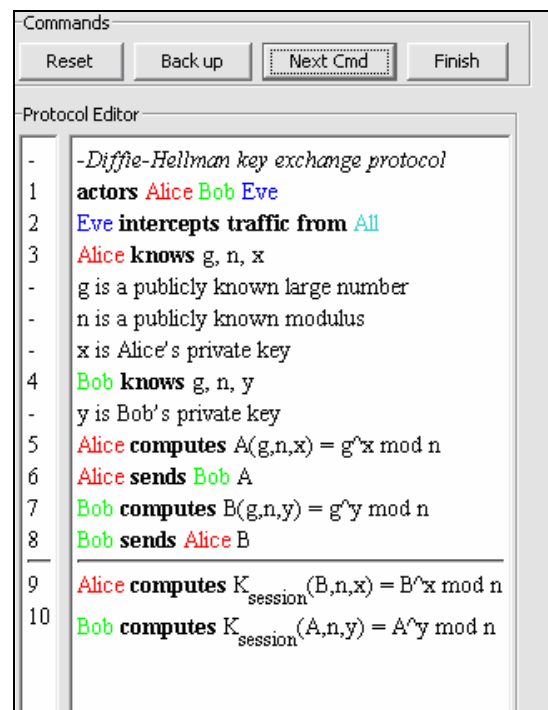


FIGURE 3. GRASP commands for Diffie-Hellman.

Only the “executable” lines are numbered. Descriptive text and comment lines assist the students in understanding the meaning of the variables beyond simple variable names. For a classroom demonstration, the instructor would have this file pre-defined and ready to load into GRASP. Figure 4 shows the GRASP visualization window for this protocol. To demonstrate the protocol the instructor would step through the commands discussing why each step is occurring and how the protocol works. Eve plays the role of “eavesdropper”, and while she is not an active participant in the protocol, the instructor can use her to illustrate that even though she sees all of the traffic, she is unable to compute the session key.

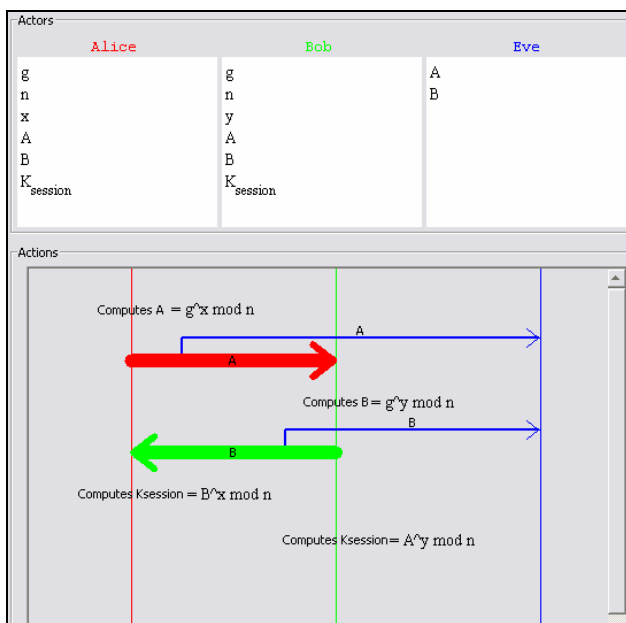


FIGURE 4. Diffie-Hellman example in GRASP.

B. Key Exchange using a Trusted Party

As a more complicated example of using GRASP to teach security protocols, the following scenario represents a demonstration of the classical cryptographic key exchange using a trusted party and its associated problems based on the description in Bishop[14].

The first step is to show the initial protocol as shown in Figure 5. Alice sends Cathy (the trusted party) a request for a secret session key to communicate with Bob. Cathy responds with the key which Alice sends to Bob. The protocol language demonstrates both the information sent back and forth as well as the necessary encryptions and decryptions. Eve (the eavesdropper) is included in the example similar to the previous scenario to demonstrate that even though she knows the variables sent back and

forth, she has insufficient knowledge to decrypt messages. As each step is executed in GRASP, the instructor can query the student as to what is happening and why it is necessary. What-if situations can be made up on the fly and the protocol can be modified and backed up to demonstrate the results. For consistency, keys are indicated with a subscripted K and messages with a subscripted M.

```
Actors Alice Bob Cathy Eve
Eve intercepts traffic from All
Alice knows KAlice Mkeyrequest MBob
Cathy knows KAlice KBob Ksession
Bob knows KBob
Alice encrypts Mkeyrequest{KAlice}
Alice sends Cathy Mkeyrequest{KAlice}
Cathy decrypts Mkeyrequest{KAlice}
Cathy encrypts Ksession{KAlice}
Cathy encrypts Ksession{KBob}
Cathy sends Alice Ksession{KAlice} Ksession{KBob}
Alice decrypts Ksession{KAlice}
Alice sends Bob Ksession{KBob}
Bob decrypts Ksession{KBob}
Alice encrypts MBob{Ksession}
Alice sends Bob MBob{Ksession}
Bob decrypts MBob{Ksession}
```

FIGURE 5. Initial key exchange scenario.

The next step in the scenario is to demonstrate how Eve can perform a replay attack on the protocol as shown in Figure 6. At the end of the previous scenario, Eve simply sends Bob the original encrypted session key she intercepted from Cathy, and resends the encrypted message from Alice. Since no decryption is necessary to repeat the messages, Bob can believe he is receiving a real message even though Eve does not know any of the keys. Once again, what-if situations can be set up and demonstrated to further describe the attack.

```
Eve sends Bob Ksession{KBob}
Eve sends Bob MBob{Ksession}
```

FIGURE 6. Adding a replay attack.

Finally, the Needham-Schroeder protocol is shown as a defense against the replay attack as shown in Figure 7. While this defense adds complexity to the entire process, the instructor can step through critical parts and explain the defense technique in detail.

```
Alice knows AliceName BobName Rand1 KAlice
Bob knows KBob Rand2
Cathy knows KAlice KBob Ksession
Alice sends Cathy AliceName BobName Rand1
Cathy computes A(AliceName, Ksession)
```

Cathy **encrypts** $A\{K_{Bob}\}$
Cathy **computes** $B(\text{AliceName}, \text{BobName},$
 $\text{Rand1}, K_{\text{Session}}, A\{K_{Bob}\})$
Cathy **encrypts** $B\{K_{Alice}\}$
Cathy **sends** Alice $B\{K_{Alice}\}$
Alice **decrypts** $B\{K_{Alice}\}$
Alice **computes** $K_{\text{Session}}(B)$
Alice **computes** $A\{K_{Bob}\}(B)$
Alice **sends** Bob $A\{K_{Bob}\}$
Bob **decrypts** $A\{K_{Bob}\}$
Bob **computes** $K_{\text{Session}}(A)$
Bob **encrypts** $\text{Rand2}\{K_{\text{Session}}\}$
Bob **sends** Alice $\text{Rand2}\{K_{\text{Session}}\}$
Alice **decrypts** $\text{Rand2}\{K_{\text{Session}}\}$
Alice **computes** $\text{Rand3}(\text{Rand2}) = \text{Rand2}-1$
Alice **encrypts** $\text{Rand3}\{K_{\text{Session}}\}$
Alice **sends** Bob $\text{Rand3}\{K_{\text{Session}}\}$

FIGURE 7. A replay attack defense.

VI. EXPERIENCE IN THE CLASSROOM

GRASP is being used to in both the Cryptography class and Computer Security class at the Air Force Academy to demonstrate key exchange protocols, common attacks, and defenses. GRASP is projected onto a large screen and used in the classroom to support lectures. The advantage of using GRASP over the traditional “chalkboard” approach is the ability to make changes on the fly, rerun the protocol, and demonstrate “what-if” cases for student discussion. Another advantage is the ability to have complicated protocols pre-defined in a file to be loaded for demonstration.

Initial student and instructor reactions to GRASP are very positive. In addition to the advantages listed above, students enjoy using a visualization tool for the “gee whiz” factor and tend to pay closer attention to what is happening. Additional feedback and experience on the tool will be gathered throughout the spring semester.

Another use for the tool which will be explored in the future is to use it for out-of-class student exercises. For example, the student can be provided the GRASP tool and a protocol and asked to describe possible attacks and/or defenses. The protocol specification language is simple enough to allow students to quickly make changes without needing to understand a complex syntax.

VII. FUTURE WORK

As experience is gained with the use of GRASP in the classroom, additional features and or modifications will be considered. Several key issues need to be addressed to

determine the tool’s effectiveness. One is whether the tool actually increases student’s understanding of security protocols. While difficult to measure, studies are planned with another university to perform classroom experiments. Another issue is what level of protocol complexity the tool can realistically represent without becoming too difficult to understand. Another area to investigate is whether the tool can be used for other purposes, besides education, such as visual protocol analysis.

GRASP is one tool in a proposed suite of visualization tools for security education being developed at the Air Force Academy known as VISE (Visualization for Information Security Education). The purpose of VISE is to provide a set of education visualizations for teaching undergraduate information security classes. The tools cover common security topics such as cryptography, buffer overflow, network attacks, and public key infrastructure. Source code for GRASP is publicly available at <http://usafa.af.mil/acis>. Comments and user experiences are welcomed.

VIII. REFERENCES

- [1] The National Centers of Academic Excellence in Information Assurance Education Program. <http://www.nsa.gov/ia/academia/caeiae.cfm> 2005.
- [2] *Proceedings of the 1st annual conference on Information security curriculum development*, Kennesaw, Georgia, October 08 - 08, 2004.
- [3] Walden, J. 2005. A real-time information warfare exercise on a virtual network. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM Press, New York, NY, 86-90.
- [4] Irvine, C. E., Thompson, M. F., and Allen, K. 2005. CyberCIEGE: Gaming for Information Assurance. *IEEE Security and Privacy* 3, 3 (May. 2005), 61-64.
- [5] Gerhart, S. L. Increasing Security Expertise in Aviation-Oriented Computing Education: A Modular Approach. <http://nsfsecurity.pr.erau.edu>. 2005.
- [6] Wang, A. J. 2005. Web-based interactive courseware for information security. In *Proceedings of the 6th Conference on information Technology Education* (Newark, NJ, USA, October 20 - 22, 2005). SIGITE '05. ACM Press, New York, NY, 199-204.
- [7] Amer, P. D. and New, D. 1993. Protocol visualization in Estelle. *Comput. Netw. ISDN Syst.* 25, 7 (Feb. 1993), 741-760.

- [8] Hall, J., Moore, A., Pratt, I., and Leslie, I. 2003. Multi-protocol visualization: a tool demonstration. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools For Reproducible Network Research* (Karlsruhe, Germany, August 25 - 27, 2003). MoMeTools '03. ACM Press, New York, NY, 13-22.
- [9] Turner, K. and Robin, A. An Interactive Visual Protocol Simulator. *Computer Standards and Interfaces*, 23 (Oct. 2001), pp 279-310.
- [10] Estrin, D., Handley, M., Heidemann, J., McCanne, S., Xu, Y., Uy, H. Network Visualization with Nam, the VINT Network Animator. *IEEE Computer*. (Nov. 2000), vol. 33 (11), pp 63-68.
- [11] Burger, C. and Rothermel, K. 2001. A framework to support teaching in distributed systems. *J. Educ. Resour. Comput.* 1, 1es (Mar. 2001), 3.
- [12] Elmqvist, N. 2004. ProtoViz: A Simple Security Protocol Visualization. Report posted at <http://www.cs.chalmers.se/~elm/courses/security/report.pdf>
- [13] Hundhausen, C.D., Douglas, S.A., and Stasko, J.T., A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing* (2002), vol. 13, pp 259-290.
- [14] Bishop, M., *Computer Security, Art and Science*, Addison-Wesley, Boston, 2003.