

A Visual Approach to Teaching Formal Access Models in Security

Dino Schweitzer, Mike Collins, and Leemon Baird, *United States Air Force Academy*

Abstract – Formal models are important in information security education. The ability to abstract security concepts and apply formal reasoning techniques provides a basis for students to understand fundamental results and have a broader perspective on security issues. Our experience at the undergraduate level is that students often struggle with the abstract models, how to apply them, and the associated implications. To provide students a more concrete approach to working with and understanding security protection models, we have developed interactive visualization tools that allow students to create, manipulate, and experiment with the models. As a result, students demonstrate a greater understanding of the core concepts than in previous course offerings. This paper will describe the visual tools, how they are used in the classroom, and our experience with their effectiveness.

Index terms – Security Education, Visualization, Security Protection Models.

I. BACKGROUND

Information assurance education is an important part of a computer science curriculum at both the undergraduate and graduate levels of study. ACM's Computing Curricula 2005 lists Security: Issues and Principles as a required topic for undergraduate Computer Science and related disciplines [1]. Schools meeting the requirements for the Center of Academic Excellence (CAE) in Information Assurance Education must satisfy a long list of security topics somewhere in the curriculum [2]. There are different approaches to how to teach security within the curriculum. Some advocate integrating security with existing CS courses [3] while others advocate a minimum number of security-specific courses in addition to traditional CS courses [4]. There are also different approaches in the methodology of how security is presented. Many programs emphasize a hands-on approach with the use of tools and projects [5].

An issue that information assurance programs must address is the role of formal models when teaching

security. It is possible to teach security practices and issues without referring to formal theoretical models. However, such an approach may be more appropriate for a service course in security literacy rather than an advanced academic course.

At the Air Force Academy we teach information security topics throughout the CS curriculum. We also have three security-specific courses in Cryptography, Computer Security, and Network Security that provide students with the underlying principles of security concepts. In our opinion, the use of formal models is important to ensure students are able to reason about security in an abstract and general way. Unfortunately, as in other topics within computer science, undergraduate students do not always have the academic maturity to quickly grasp the abstract nature of the formal methodology. This paper will address an approach that we have taken in an attempt to make the abstract nature of formal security models more concrete for students.

II. FORMAL MODELS IN UNDERGRADUATE EDUCATION

Computer science faculty across the country recognize, in general, that formal models are an important aspect of understanding underlying principles and as a way to get students to think critically about issues such as the limitations of computing machines and ensuring software correctness [6]. It is also acknowledged that undergraduate students struggle with the abstract nature of formal models and their relevance to real-world applications. General education theory acknowledges that undergraduates are much more concrete than abstract and, as such, different approaches have been proposed for teaching formal methods in computer science [7].

In 1981, Carl Landwehr published a landmark paper surveying the state of formal modeling within computer security research [8]. Dr. Landwehr mentions briefly that "Harrison, Ruzzo, and Ullman investigated an access matrix model with six rules similar to the examples posed by Graham and Denning and found undecidable the question of whether, given an initial access matrix configuration, an arbitrary access right can later appear at an arbitrary location in the access matrix." This serves as

Authors are with the Academy Center for Information Security, Department of Computer Science, US Air Force Academy, CO.

a useful starting point for a discussion about formal modeling at the undergraduate level, especially for computer science or technical majors.

One of the first questions a beginning student of security is likely to ask is: "Can a given user get access to my stuff?" which might be a less formal inquiry but sets the stage for further lesson development.

Secondly, the original paper by Harrison, Ruzzo, and Ullman, provides a straightforward proof of undecidability by means of reduction to the Halting problem [9]. For an undergraduate technical major, reduction proofs and Turing Machines are likely covered in a course on Complexity Theory or Compilers. However, many students taking the security course, such as Systems Engineering majors, may never receive such exposure. These elementary examples serve as a very good introduction to this critical and foundational skill of an Information Security professional.

The big picture implications of these results are important and have a critical relevance to normal computer system operations. As future Air Force officers, many of our graduates will come face-to-face with vendors promising them multi-level, cross-domain, robust secure solutions involving state-of-the-art technology based on revolutionary paradigm shifts for network centric warfare: a single system that will do everything necessary to protect the citizens of this country. Giving our students a thorough understanding of the possible and impossible while maximizing their ability to make informed decisions based on critical thinking is essential.

Thirdly, we want the students to engage and experience the topic through learning-focused techniques. Specifically, HRU and Take-Grant are good models for interactive tools that allow students to "play" with formal models. This learning-focused approach is reinforced with lecture and readings from the security literature and texts [10, 11].

At the United States Air Force Academy, we teach an undergraduate senior-level course in Computer Security and Information Warfare. Students in the class are a combination of computer science, computer engineering, and system engineering majors. One of the modules in this course teaches formal access control models. The primary objective of this module is to have students understand and synthesize the results of Harrison, Ruzzo, and Ullman as well as interpret advance concepts involving the Take-Grant model. A key aspect of this goal is the assumption that our students represent a wide variety of technical majors, not just computer science or mathematics which typical teach complexity theory and theorem proving.

Secondary objectives include knowledge and comprehension of a variety of types of formal models in security and knowledge about how such rigor is incorporated into system life cycles to include design phase through testing and evaluation.

III. FORMAL SECURITY PROTECTION MODELS

HRU and Take-Grant are both Access Matrix Models [9,10]. Both of these are taught to the students so they will understand what access control models are, and some of their inherent limitations.

In both models, the current set of access rights at any given time can be represented by a matrix, with one row for each subject and one column for each subject and object (all subjects are also objects). Each cell in the table contains the list of access rights that the particular subject has for the particular object. For example, a user might have read access or write access to a particular file. In Take-Grant, the matrix is normally represented as a directed graph with nodes representing subjects and objects, and arcs for the rights between nodes.

In both models, there are then a set of operations that can be performed. In the Take-Grant model, the operations are predefined. In HRU, the primitive operations are predefined, and the user defines "macro commands" composed of conditions and primitive operations. In both models, the simplest question that might be asked is "is there any sequence of operations that will allow subject S to gain the access right R for object O". For example, it might be asked whether a particular user will ever gain write access to a particular file.

In the Take-Grant model, a subject can *grant* some or all of its rights to another subject or object. It can also *take* rights from another, if it has the right to do so. It can also create another subject, to which it can grant rights, and from which it can take them. Given these few simple rules, the behavior of the model can be fairly complex. Often, the shortest path to transfer a right from one subject to another involves a very long sequence of operations, and the creation of numerous subjects to temporarily hold rights. Students generally find it difficult to predict what is possible until they have spent a fair amount of time experimenting with the model and exploring its capabilities. There are, however, deterministic, polynomial-time algorithms for predicting whether it is possible for a given right to be transferred to a given subject. So the behavior is predictable in theory, even if it is difficult for students.

In contrast, the HRU model is not always predictable, even in theory. The question of whether a given right can

reach a given subject, for a given set of macro commands, is undecidable. This is an extremely important point, but can be very difficult for the students to grasp. The proof consists of demonstrating how a user can define a particular set of macro commands which cause the access control matrix to behave like a Turing machine. In that case, the question of whether a particular subject gets a particular right is equivalent to asking whether a particular Turing machine will halt, which is undecidable. The students have only a little knowledge of Turing machines when they enter this course, and are not always comfortable with mathematical proofs in general.

IV. CLASSROOM VISUALIZATION TOOLS

Visualization tools can be an effective means for teaching and reinforcing complex concepts [12]. Within computer science, several examples can be found online in a variety of topics such as algorithms, data structures, graph theory, computer graphics, and computer theory. Within computer security, tools have been developed for teaching cryptography, security protocols, and attack graphs [13, 14]. Such tools can be used in a variety of ways to support student learning such as in-class demonstration, student labs, out-of-class exercises, and as general reference material. Ideally, students find the tools enjoyable to use and easy to understand, so that the tool can become a positive active learning experience [15]. The connection between the tool and the abstract concept being demonstrated should be easy to understand and reinforce the central ideas. By allowing students to directly interact with the visualization tool, set up “what if” scenarios, and see the results, they get a hands-on concrete experience. For many students this makes the abstract concept easier to understand.

As described earlier, the HRU and Take-Grant formal protection models were chosen to be represented with interactive visualization tools. The tools will be individually described followed by a description of how the tools are used in the classroom.

A. HRU Tool

The HRU Tool, HRUVis, consists of three parts:

- Access control matrix
- Command definition
- Commands to be executed

A screen shot of the tool’s layout is shown in Figure 1. The right hand panel is the access control matrix which represents the current state of the system. Rows of the matrix are used for subjects and columns for subjects and objects. The user can edit this matrix to a) add or remove subjects and objects, and b) add or remove rights between

subjects and objects. Right mouse clicking within the panel brings up a menu of options followed by a dialog to complete the requested action. Editing the access control matrix allows the user to set up the initial conditions of the problem to investigate.

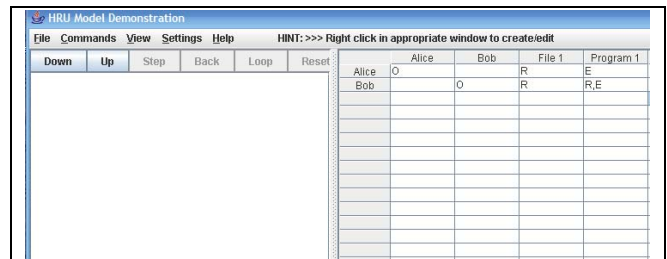


Figure 1. HRUVis screen layout.

Command definition is accomplished by right clicking in the left hand panel, or command panel, and entering the appropriate information in the follow-up dialog as shown in Figure 2. In the HRU model, commands are defined by specifying a set of conditions (a specified right at a specified location in the matrix) and a set of primitive commands such as creating/removing subjects and objects and adding/removing rights. The command definition optionally allows for parameters that are substituted for subjects and objects when the command is instantiated. Adding conditions or primitive commands result in follow-up dialog boxes showing possible options. The goal is to create a “fill in the blank” experience for the student in defining commands.



Figure 2. Command definition dialog box.

Commands to be executed appear in the left hand panel and are created/edited/removed with a right click of the mouse. The user is shown a list of defined commands to select from and asked to enter the actual parameters. Once entered, the user can choose to look at the commands as entered, or in a fully expanded mode that

shows the command definition for each command. Figure 3 shows a sample set of commands in expanded mode. One variation on the original HRU model is that the tool allows for “wild card” parameters. That is, if an “*” is used as an actual parameter, it is equivalent to having multiple instances of the command with all possible subjects and/or objects as the actual parameter. This allows for a short hand way of specifying many commands succinctly and was very useful when solving general Turing machine problems.

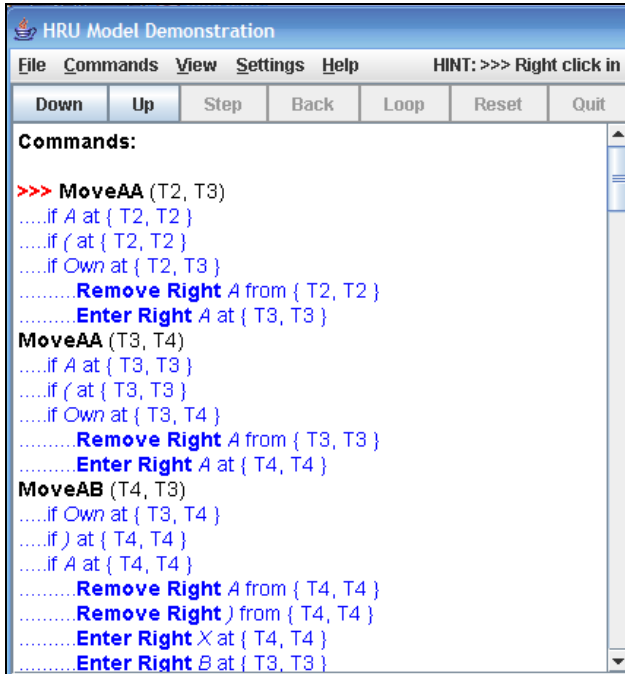


Figure 3. Sample commands in expanded view.

Once the matrix has been initialized and commands defined and entered, the user can “execute” the commands to see the resulting effects on the matrix. When the user chooses to execute the commands, a set of buttons for controlling execution become active. These buttons allow the user to single step, backup, reset, or loop execution. The matrix is updated as the commands are executed.

Various utility options are available in the program to load/save data files, add and rename rights, clear the commands and matrix, and help.

B. Take-Grant Tool

The Take-Grant tool is very similar in user functionality and screen layout to the HRU tool. The basic screen layout is shown in Figure 4 (some nodes and arcs have already been entered). Similar to HRU, the right hand panel is the canvas containing the current state of the

system. For Take-Grant, that is a directed graph with subjects and objects as nodes, and privileges as arcs between nodes. The user can add/delete/edit nodes and arcs in the graph panel by right clicking and following the appropriate dialog. To assist readability, all objects in the graph panel can be moved with the mouse to produce the desired layout.

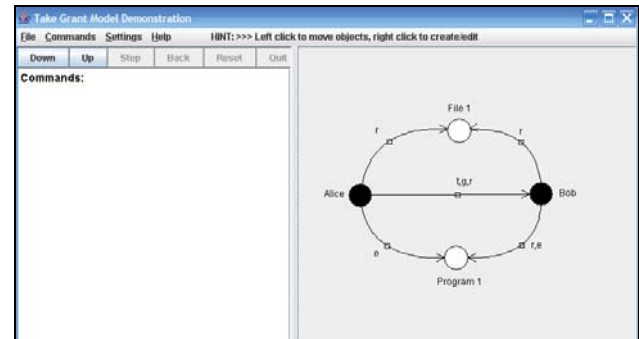


Figure 4. Take-Grant tool screen layout.

Also similar to HRU, the left panel is the command window in which commands can be created/deleted/edited. In the Take-Grant model, commands are of a simpler form and do not require command definition. The user selects which command they desire, and provide the appropriate subjects, objects, and rights as required by the command. A sample set of commands is shown in Figure 5.

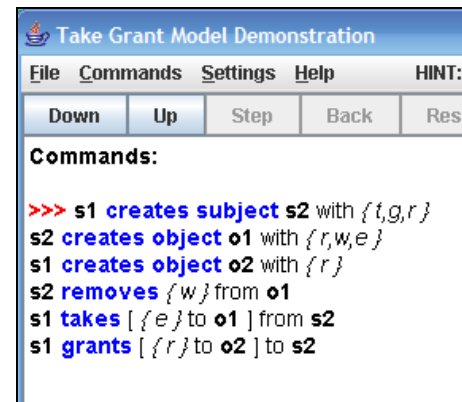


Figure 5. Sample commands in Take-Grant tool.

Command execution is similar between both HRU and Take-Grant with the ability to step, back up, and reset. As commands are executed, the graph panel is updated to reflect the change in states. There is currently no Loop functionality as in HRU, as there was no application for it in the examples that were being used in the classroom.

V. USING THE TOOLS IN THE CLASSROOM

The Computer Security and Information Warfare course specifically addresses the HRU and Take-Grant models with associated objectives in six lessons. The overall structure to these lessons in the past was straight lecture with students performing exercises at the chalk board.

The structure of presenting the formal models for Spring of 2007 changed based on the visualization tools. The general models and primary attributes were described and demonstrated using the tools. Simple examples and exercises were then used by the students both in and out of class to spur questions about the syntax and semantics of the tool's implementation. Once the model was introduced, students were assigned problems in writing model commands to demonstrate their understanding of the models along with follow-up questions delving into deeper related topics. Finally, students were assessed based on short answer questions in a controlled environment.

Specifically, in order to introduce the correspondence between HRU Access Control Matrices and Turing Machines, the instructor programmed a Turing machine that counts in binary using the HRU tool. The implementation of the binary counter is available at our web site.

The instructor then created a contrived but useful story about a strange access control policy for a fictitious company that explained the security policy for this protection system. This scenario served as a gentle introduction to the HRU tool and the instructor paid strict attention to using only terminology associated with access control models and security. No hint of a Turing Machine was mentioned. The use of the commands was demonstrated and analysis was discussed about which command would be valid next. The instructor insured that every student could at least initially configure a protection system and could interpret commands that were previously defined.

Then the instructor simply asked the class to find a specific sequence of commands within HRU that would put a specific permission into a specific cell of the given protection system. The first examples were easy and required two to three commands. The final two examples were quite difficult and not obvious. In terms of the binary counter, they were asked if you could count to a very large number or whether there is any bit to the right of the least significant bit. Using HRU terminology there is little indication as to the answer to these questions and students found the puzzle-like nature of the environment intriguing.

Finally, the instructor ran a given set of commands over and over (using the * feature described earlier) and had a student record on the white board the diagonal entries at specific times during the run. Eventually, it was obvious to all students that a binary counter was occurring. When one student asked "Why on earth would anyone do that?," it was a great lead-in to the concept that you can implement Turing Machines in an HRU access control model and to prove the undecidability results that Harrison, Ruzzo, and Ullman proved.

One challenge for students implementing a specific Turing machine, was to understand the mapping between the two different worlds. At this point, the instructor made use of a graphical Turing Machine visualization available from www.cs.princeton.edu/introcs/75turing/TuringMachine.java.html. Using both the TM tool and the HRU tool side by side the class walked through the creation of a second HRU Turing Machine implementation. Class time was then provided for students to implement a third machine in HRU on their own.

Once students got the idea of the mapping, they found programming the demo programs into HRU fairly easy and starting asking questions such as "Now explain to me how this proves the point about undecidability?" This was answered by going back over and explaining in more detail, through a more traditional lecture, the Halting problem.

The approach to teaching Take-Grant was similar. We introduced the tool and syntax. We emphasized the goal of the model is to determine whether it can decide (in linear time) if a given subject can get rights to a specific object. After familiarization with the tool, students were asked to write commands for simple problems that demonstrated a specific sequence in the model. Similarly, they were given a homework problem using the tool and asked a question on an in-class exam. The model is simpler than HRU and students quickly grasped its operation and meaning. The fact that the layout and user interface was similar to HRU helped in quickly mastering the tool use. As a result of the simplicity and familiarity, we spent less class time on the model and tool.

VI. OUR EXPERIENCE

Anecdotal evidence strongly supports the claim that these tools help students learn the desired concepts. In the past, student feedback on the course (which used solely traditional lecture) indicated they never fully understood the relevance and implications of this formal model block of the course. While student feedback for this year has not been formally received, students in class indicated that it

seemed to be such a simple concept that we likely spent too much time on it in class.

We assess student understanding of the HRU model through homework questions, the original exercise asked for students to handwrite and draw an initial configuration that represented a parentheses checker. In 2006 the average score for this problem was 33% of the available points. In 2007, with the use of the tool, the average score was 93% of the points for that problem. While it is possible there are other explanations for such a dramatic increase in the scores, the discussions in class, student questions, and pointed comments suggest the tool was extremely beneficial in reinforcing the concept.

In regard to the Take-Grant tool, we do not feel that its usefulness has been fully tested due to the elementary level at which we currently address the topic. It is clear from student comments that Take-Grant is much more readily accessible and simple when compared to the reduction proof of the HRU model. We expect that the tool will be much more useful as we increase the depth of our study in this topic in terms of theft and conspiracy.

VII. FUTURE PLANS

The two tools presented in this paper are part of a larger suite of visualization tools being developed at the Air Force Academy for information assurance education. Several of these tools including the HRU and Take-Grant models are available for download and use at our web site. Other tools, such as PKI visualization, are currently under development. Additional security concepts are being planned for implementation in the future.

Some minor adjustments to the HRU and Take-Grant tools are planned based on feedback from the students. These include making it more intuitive in places and adding additional help and examples for standalone student use. One major planned addition to the HRU tool is to tie it directly to a Turing machine visualization tool to reinforce the translation from Turing machine to HRU implementation. The Princeton tool mentioned earlier allows the user to define the machine and simulate it graphically. Our plan is to modify this tool so that it follows our notation, and have it automatically produce the necessary input data file for the HRU implementation of the defined machine. This will allow the student to define an arbitrary Turing machine (or load one of the predefined ones), watch it operate, and then load it into the HRU tool to see how it is defined and operates in that model.

Additional plans include introducing more advanced concepts in Take-Grant such as conspiracy and perhaps tools for other classic models. In particular, we will

research means to add functionality to the existing HRU tool to enable the implementation of other security models within it in a user-friendly manner. The hope is that a discussion of the power of different models can be started and that comparison of differences in models may be easier for students to comprehend.

VIII. REFERENCES

- [1] Computing Curricula 2005 found at <http://www.acm.org/education/curricula.html>.
- [2] Schweitzer, D., Humphries, J., and Baird, L. 2006. Meeting the criteria for a Center of Academic Excellence (CAE) in information assurance education. *J. Comput. Small Coll.* 22, 1, pp. 151-160.
- [3] Petrova, K., Philpott, A., Kaskenpalo, P., and Buchan, J. 2004. Embedding information security curricula in existing programmes. In *Proceedings of the 1st Annual Conference on information Security Curriculum Development*, pp. 20-29.
- [4] Vaughn, R. B., Dampier, D. A., and Warkentin, M. B. 2004. Building an information security education program. In *Proceedings of the 1st Annual Conference on information Security Curriculum*, pp. 41-45.
- [5] Mattord, H. J. and Whitman, M. E. 2004. Planning, building and operating the information security and assurance laboratory. In *Proceedings of the 1st Annual Conference on information Security Curriculum Development*. InfoSecCD '04, pp. 8-14.
- [6] Palmer, T. V. and Pleasant, J. C. 1995. Attitudes toward the teaching of formal methods of software development in the undergraduate computer science curriculum: a survey. *SIGCSE Bull.* 27, 3, pp. 53-59.
- [7] Almstrum, V. L., Dean, C. N., Goelman, D., Hilburn, T. B., and Smith, J. 2001. Support for teaching formal methods. In *Working Group Reports From ITiCSE on innovation and Technology in Computer Science Education ITiCSE-WGR '00*, pp. 71-88.
- [8] Landwehr, C. E. 1981. Formal Models for Computer Security. *ACM Comput. Surv.* 13, 3, pp. 247-278.
- [9] Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. 1976. Protection in operating systems. *Commun. ACM* 19, 8, pp. 461-471.
- [10] Bishop, M. 2003. *Computer security: art and science*. Addison-Wesley, Boston.

- [11] Pfleeger, C.P. and Pfleeger, S.L. 2003. Security in Computing. Prentice-Hall, N.J.
- [12] Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velázquez-Iturbide, J. Á. 2002. Exploring the role of visualization and engagement in computer science education. In *Working Group Reports From ITiCSE on innovation and Technology in Computer Science Education*. ITiCSE-WGR '02, pp. 131-152.
- [13] Schweitzer, D., Baird, L., Collins, M., Brown, W., and Sherman, M. 2006. GRASP: a visualization tool for teaching security protocols. In *Proceedings from the Tenth Colloquium for Information Systems Security Education*. Adelphi, MD.
- [14] Schweitzer, D. and Baird, L. 2006. The design and use of interactive visualization applets for teaching ciphers. In *Proceedings of the 7th IEEE Workshop on Information Assurance*. U.S. Military Academy, West Point, NY.
- [15] Schweitzer, D. and Brown W. 2007. Interactive visualization for the active learning classroom. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '07, March 2007.