Security Awareness Training Simulation

Michael C. Tanner, Christopher Elsaesser, and Gregory M. Whittaker
Cognitive Science & Artificial Intelligence Center
The MITRE Corporation
7515 Colshire Drive
McLean, VA 22102-7508

Abstract –This paper describes how simulation can be used to train systems administrators and managers about computer security threats. The advantage of simulation-based training over commercial seminar and lab oriented classes is that a simulation can be customized to resemble the enclave the student is interested in securing. Customization makes the training more relevant to the student and, by factoring out irrelevant topics, allows the student time to explore the effects of relevant hacker attacks in more depth than is possible in typical classes. A proof-of-concept prototype is described, along with the remaining development necessary to accomplish all the training objectives.

1. Introduction

Commercial and Federal computer systems are targets of relentless assault by criminals and foreign intelligence operatives. Studies by the Department of Defense and Carnegie-Mellon University's CERT Coordination Center estimate that 80% of security incidents go unreported; no one knows how many go undetected. Intrusions and their lack of detection stem, in part, from lack of security awareness on the part of systems administrators and IT management.

The threat posed by ubiquitous computer attacks has spawned a security training industry. The SANS Institute[1] and Foundstone, Inc.[2] both provide security awareness training, for example. The objective of security awareness training is to vicariously introduce system administrators and IT management to the hacker threat. The typical curriculum aims to help students understand how hackers do what they do and why it works—a sort of "Hacker 101"[3]—through lectures and (in most cases) lab sessions. These courses generally are excellent, but they are expensive—on the order of $1000 per day—and time consuming, typically taking four to five days.

The high cost of security awareness training would appear to be unavoidable. There is quite a bit of introductory material to cover, and hands-on experience with hacker tools and techniques is essential. Due to the criminal and destructive nature of the activities being practiced, it is necessary for the training vendor to establish an isolated network of computers on which students can attempt attacks.

Despite the cost, practical considerations prevent these courses from providing all the prerequisite knowledge for security awareness. In a nutshell, the training is too broad and too shallow. Obviously, resemblance of the laboratory network to any student's home enclave is unlikely. Training usually includes all common modes of attack,[4] while individual students' interest is on those their enclave could suffer. Due to time limitations, each exploit can only be addressed superficially and little emphasis is possible on post-incident response and remediation.

Our hypothesis is that simulation-based computer security awareness training can be more focused and less expensive than lecture-lab courses in use today. Replacing the customized laboratory networks with a simulation can address the cost factor. Simulation based training can be *more* effective because it allows several things that are impossible with today's lecture-lab training:

---

[1] http://www.sans.org
[2] http://www.foundstone.com
[3] Advanced courses are few, and are mainly aimed at niches of expertise for the growing cadre of information security practitioners (e.g., how to use TCP Dump and Snort).
[4] Unix, NT, web, DBMS, script kiddies, social engineering, etc.

- Access to multiple viewpoints—what an attack looks like from the perspective of both the perpetrator and the defender.
- Customization to the student's enclave so training can be focused on threats the student is most likely to face, without wasting time on threats that are impossible in his situation
- Repeatability and What-if analysis, allowing the student to simulate and test possible remedial actions
- Student modeling that senses the student's grasp of the material, allowing novices to be eased into the material and experts to skip ahead to more complex issues.

This paper discusses a proof-of-concept training simulation created to test the hypothesis stated above. In the next section we outline what such a simulation must do to address security awareness objectives and the issues outlined above. The section following describes our prototype and how it accomplishes the design objectives. The final sections describe related work and areas for further development.

2. Design objectives

This section lists objectives to be accomplished if a simulation is to duplicate and surpass lecture-lab training.

2.1 Content: Understand the threat

The first step in information security awareness is to know the enemy. For example, novice hackers pose a pervasive but relatively easily mitigated threat. They employ off-the-shelf tools that exploit well-known vulnerabilities and exhibit recognizable patterns. Novice hackers usually lack the knowledge to adapt their scripts for differing targets and situations. Expert hackers are patient, methodical, and not limited to well-known vulnerabilities and readily available attack tools. They engage in elaborate preparation and reconnaissance activities that are not easy to detect.

2.2 Content: Awareness of known weakness and attack techniques

The objective of security awareness training is that the student understands the weaknesses of his networked computer systems and how hackers might exploit them.[5] Since most vulnerabilities and exposures are documented, the student could learn about them on his own, given enough time. As a practical matter, there are too many attack tools for "book learning" to be feasible.[6]

To be most effective, training should be tailored to vulnerabilities and exposures most likely to be found in the trainee's environment. The student must learn his degree of exposure and how to balance it against operational flexibility and system availability.

2.3 Pedagogy: Support training objectives

The purpose of computer security awareness training is to *train* students. There is a rich body of research on the ingredients of good training (for example, Gagne, Briggs, & Wager, 1992; Gagne, 1985; Briggs, Gustafson, & Tillman, 1990; Dick & Carey, 1990). In addition, the Defense Training Standards Working Group, a joint service and industry initiative, has published a set of handbooks for training development (most importantly, for our purposes, MIL-HDBK-29612-2 and MIL-HDBK-29612-3). The following requirements are of particular relevance to the computer security domain:

- Connect concepts to practice
- Repeatability
- Progress from novice attacks through more sophisticated versions

[5] Students should also be aware which vulnerabilities can be eliminated and which are unavoidable exposures inherent in the design of the applications or services provided.

[6] Instructors often tell administrators to subscribe to e-mail alert services such as BUGTRAQ to stay abreast of threats. One quickly realizes that they could devote all their time reading about and remediating all the threats documented therein.

- Examine “what ifs” by reconfiguring systems and trying again.
- Practice skills in a realistic training environment
- Develop problem-solving and decision-making skills
- Learn to recognize operational indicators of normal, abnormal, and emergency conditions

Only the first objective can be accomplished by commercial awareness training. The others are not feasible in a classroom setting due to time constraints. Some are impossible without simulation. The most challenging requirement is to transition a student from novice to expert material at an appropriate pace. We have yet to accomplish this in our prototype. It is an objective of ongoing research.

2.4 Pedagogy: Support multiple views

A crucial pedagogical need in the computer security domain is that the student be able to take actions and see their manifestations and effects from any of the following three perspectives:

- Attacker – Getting the attacker’s point of view helps the student understand what, and sometimes why things are vulnerable. This is often the limit of commercial training labs—one tries an attack and sees what happens.
- Defender – In our experience, there is often not time in commercial classes to examine an attack from the “inside.” For example, what does a buffer overflow attack look like when it is happening? Are log entries created? Is there a console message? One would like also to vary defenses—for example by turning off certain services—to see what happens on the defender side.
- Forensic Analyst – Few attacks are intercepted as they occur. One of the most important aspects of security awareness training, rarely covered even in courses on incident handling, is how to distinguish attacks from everyday data collected by system audit and IDS logs.

2.5 Content: Model the student’s environment

Perhaps the most important way simulation can improve security awareness training is by making the simulated environment match the students’ operational environment. Laboratory exercises in commercial training classes must be general enough to encompass the majority of the environments the students might defend. There will always be platforms and services that, however ubiquitous, will not be of interest to every student. The ideal training environment for a particular student would include all, and only those systems and services a student has in his enclave. Such customized training would be more relevant, in-depth, focused, and effective than general-purpose classes.

Vulnerability information from network-mapping tools could be used to select the most dangerous attacks the students are likely to experience in their environments. We think this is carrying training past the optimal point of customization. Not all attacks will be correctly applied. It is useful to see failed attacks to be able to recognize that they were attempted. Being able to recognize a misdirected attack helps the defender realize that someone found his enclave worth penetrating—perhaps because of the type of business it supports—as well as the possibility of insider misuse. Therefore, filtering out all attacks likely to fail would limit the value of training.

2.6 Portable, self-contained laboratory

Few IT managers or system administrators can spare five days to attend a security awareness course. The promise of computer-based training is that it can be done on a local machine, allowing training to occur during on-the-job lulls in activity. Ideally, a training course would be available on a diskette that a student can use without instruction.

3. Approach

This section describes how our simulation-based security awareness training system addresses the objectives listed in above.

3.1 Background information delivery via "page turning" technology

In our design, background material about the motivational and tactical differences between novice and expert attackers will be presented by conventional interactive training technology. Packages used to implement what is sometimes termed "automated page turning" are widely available. In fact, Foundstone and SANS could easily use this method to present the majority of the introductory material they present in their courses. We base our introductory material on the *Know Your Enemy* series,[7] an excellent source of insight into the motivation, intellectual caliber, and technical expertise of hackers.

3.2 Meeting training requirements with simulation

Our hypothesis is that all the security awareness training currently delivered by classroom and laboratory experience can be duplicated, and in most cases improved using simulation-based training. Our current implementation—described in more detail in Section 4—demonstrates that it is possible to simulate attack actions and their effects. Here we describe how we tailor the simulation to a student's enclave, and how this provides a richer training experience than is possible in typical commercial classes.

One advantage of a simulation is control of ground truth. This capability is what allows simulation-based training to address a crucial pedagogical need in the computer security domain: it allows a student to take an action and examine its effects from any perspective. The simulation has access to user input, the (simulated) system response, and what part of the response the user is allowed to see.

The key to enhancing the training experience is to customize the simulator to resemble the student's enclave. This has two benefits that are not available in current commercial training. First, it shortens training by eliminating material that is irrelevant to the student. Second, shorter training allows more time to examine the attacks most likely on the student's network. We create a customization by converting the output of network and system administrators' tools such as Tkined, SATAN, and Merlin to the simulation's representation.

For additional fidelity, students experiment with attacks they are most likely to experience based on their enclave's configuration, services, and so on. We base the attacks on the SANS Top 10 list. Whether an attack works is determined by vulnerability information from network-mapping tools. Enclave vulnerabilities are not what determine which attacks are in the training because it is useful to see attacks that fail and recognize that they occurred. Importantly, a student's enclave may be targeted in certain ways because of the business it supports, say banking or on-line shopping. Because this is something only a person can know, attack selection by the user is also contemplated.

A simulated network permits automation of the learning process through advanced computer-based training technology. We anticipate incorporating such technology in the future to lead students through training exercises, progressing from novice to expert levels.

4. Implementation

This section describes a working simulation system that demonstrates that most of the requirements listed above can be realized. The objective of our continuing research, described in the final section of this paper, is to fully address any unfulfilled requirements and eventually result in a complete training system.

Our simulation's original purpose was to test feasibility of hypothesized attacks for an automated intrusion forensics aid (Elsaesser & Tanner, 2001). In the course of that research, we took several training courses, including the SANS and Foundstone courses mentioned in Section 1. During one of these courses it occurred to us that we could create a security awareness training system that duplicated the courses by adding a graphical interface to our simulation.

4.1 Simulation structure and definition

Our simulation consists of object-oriented models of networks, computers, and their associated file systems. Saying the simulation is object-oriented means that generic behaviors are specified as methods on

[7] http://project.honeynet.org/papers

classes at various levels of the object hierarchy. The methods may be customized for particular subclasses, either by writing new methods for that subclass or by adding "before" or "after" methods that do extra computation when the main method is applied to an instance of the subclass. For example, all computers have some means of logging in, but the login method for a Unix host has different effects than the login for an NT host. Figure 1 shows the class hierarchy of our simulation.

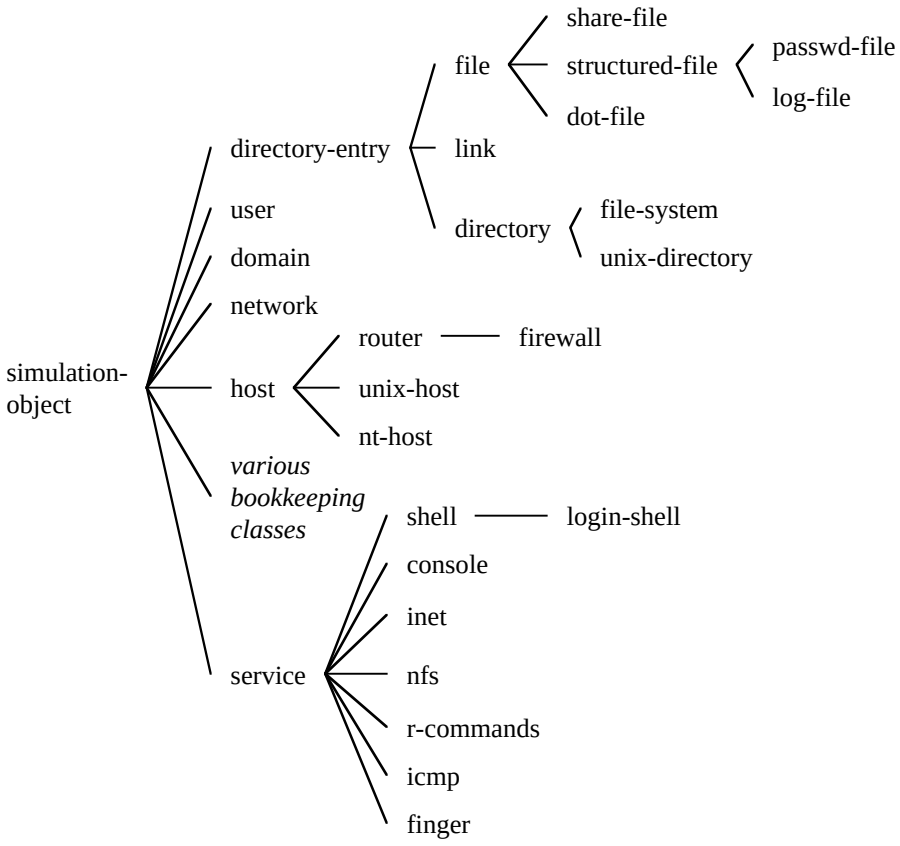

Figure 1: Simulation class hierarchy

The user class represents persons. Each instance of user (hacker, system administrator, etc.) includes information about the current identity (UID) of that person on each host, the services in use, and the person's history during the session.

Our simulation is actually a shell for modeling networks of computers. To create an instance of a training network, configuration information is collected by mapping tools and recorded in text files in syntax called the Planning Domain Description Language (PDDL) (Ghallab, et al., 1998). PDDL allows three basic levels of description that are relevant here: domain, situation, and problem. A snippet of our network domain description is shown in Figure 2.

```
(define (domain network)
  (:extends computer)
  (:types network domain – simulation-object
          router – host
          firewall - router)
  (:predicates
   (connected-to ?node - object ?net – simulation-object)
   (part-of ?net - network ?domain - domain)
   …)
```

Figure 2: Example domain description

The :extends clause lists domains from which the new domain inherits everything. Therefore, only new classes—specified by the :types clause—and relationships—specified by the :predicates clause—need to be described.

The network class models a wire connecting many hosts, which are defined in the "computer" domain. The computer domain specifies the existence of services, which run on hosts. Hosts contain files and directories. The network domain specifies some subclasses of the host class. Defining a domain simply creates these subclasses and relations. Code to make them meaningful is, of course, the main part of our system.

Creating a unique simulation instance consists of defining a "situation" that describes the simulation's objects and relationships. Figure 3 is an example. A simulation is created by parsing such a description, creating objects from the :objects clause, which lists object names and their class. The :init clause describes relationships among objects using :predicates specified by the :domain. These relations, along with the class types of the objects, are used by methods to determine legal actions and how they are executed. For example, the topological information in Figure 3 specifies that the Unix-host named "eve" is part of the isp.com network. The effect of this declaration is that eve can be reached via that network, just as in a real network.

```
(define (situation main2)
  (:domain network)
  (:objects xyz.mil.net stu.edu.net abc.com.net - domain
   routermain routerxyz router1 routerabc - router
   traffic able baker charlie cheeto bugle skor - nt-host
   mite butterfinger eve spider ant beetle - unix-host
   abc_vpn gnat tsetse - firewall
   internet isp.com xyz.mil stu.edu abc.com - network)
  (:init
   (part-of xyz.mil xyz.mil.net)
   (part-of stu.edu stu.edu.net)
   (part-of abc.com abc.com.net)
   (connected-to isp.com eve)
   …
   (connected-to beetle stu.edu)
   (connected-to cheeto stu.edu)
   (connected-to bugle stu.edu)
   (connected-to stu.edu Router1)
   (connected-to mite stu.edu)
   (connected-to butterfinger stu.edu)
   (connected-to stu.edu tsetse)
   (connected-to internet tsetse)))
```

Figure 3: Specification of a simulated network

A problem gives specific information supporting the objective of the exercise. A problem is ephemeral; loading a new problem can overwrite any of the information. The :init list of a problem specifies enabling conditions of the attack that the student will be taught to accomplish. In the example in Figure 4, facts such as the network services provided by a host, the user accounts on a host, and the contents of critical files are specified.

```
(define (problem attack1)
  (:domain network)
  (:situation main2)
  (:init
   (offers-service butterfinger r-commands)
   (offers-service butterfinger nfs)
   (offers-service butterfinger finger)
   (offers-service eve finger)
   (has-account butterfinger guest user /export/foo)
   (found /export/foo butterfinger /etc sharetab)
   (found + butterfinger /etc hosts.equiv)
   …))
```

Figure 4: Description of a problem based on the situation "main2" in Figure 3

Importantly, predicates in a problem description are the enabling conditions of an attack. Identifying the conditions allows the training system to explain to the student why an attack worked and what to change to disable it.

Since these configuration files are human-readable text, an instructor can create and edit them with a text editor. In our implementation it is also possible to draw or edit the training network in a graphical interface to generate or modify the situation description. This is useful for augmenting configuration information initialized from the output of off the shelf network and system administration tools, the main device for creating situations. In this way, we are able to create a training simulation customized to the student's enclave.

4.2 Simulating user actions

Simulated computers (hosts) operate by executing command-line actions of users and generating associated effects, including recording actions in log files and triggering intrusion detection system alarms.

For someone to do something on the network, he must begin a session on a host on the network. Our simulation models this by console logins. Suppose a user (call him "hacker") wants to access host "Butterfinger" from host "eve." Hacker must first log in to eve at a console, then (say) rlogin to Butterfinger. In our simulation the initial login would be executed by evaluating the following function call:

```
(command hacker login root eve.isp.com)
```

"Command" is the function that fields all user-simulation interactions.[8] It causes methods that implement service directives to be applied to the arguments and implements logging. In the example above, command would create a console service on the host eve, then dispatch the arguments "root" and "eve.isp.com" to the login function. Login checks eve's /etc/passwd file[9] to determine that there is an account named root and whether the user knows the password for that account. Password knowledge is simulated by a list of the passwords that each user knows. In this example, if hacker knows the password of the root account on eve, then login will create a login-shell service and establish hacker's identity on eve as root.

In this problem, because the host Butterfinger is misconfigured to trust all other hosts, hacker can rlogin to it from eve by executing the following:

```
(command hacker rlogin butterfinger.stu.edu root)
```

The command interpreter looks up hacker's current service and finds a login-shell, which accepts rlogin commands, then dispatche the arguments "butterfinger.stu.edu" and "root" to rlogin method. Rlogin

[8] Command is an Application Programmer Interface function. A trainee would cause it to be invoked by typing a login command to a shell, just as if s/he were on a Unix machine.

[9] In this example eve is a Unix machine, so the login method will be the one for Unix-host. Other host classes have their own login methods that do things in the appropriate system-specific manner.

attempts to connect to the r-commands service of butterfinger.stu.edu by placing the connection request on network isp.com. A router that bridges isp.com and the Internet (see Figure 6) sees a packet on isp.com bound for stu.edu and has a rule that applies, so it picks up the packet[10] from isp.com and places it on the Internet[11]. Similarly, when the packet is on the Internet, several routers see it but only one has a rule that covers it, which causes the packet to move from the Internet to stu.edu. When the packet is on stu.edu, Butterfinger notices it and picks it up to process. Since Butterfinger provides r-commands services, it creates an instance of r-commands and returns a "ready to process" signal back through the network to eve. This causes eve to send rlogin back to Butterfinger. The r-commands service on Butterfinger then checks to see that Butterfinger trusts eve and verifies that the user is in the same account on eve that he is logging into on Butterfinger[12]. In this case that all holds, so Butterfinger creates a shell service and makes hacker's current identity be root on Butterfinger.

As is apparent from the description above, our simulator does not emulate Unix or TCP/IP. We implemented sufficient fidelity for the purpose of training basic system security. The simulator models enough detail to show a beginner just how dangerous trust relationships on his computers can be, and how to find and eliminate them. It also lets the student try basic network access commands to test the effect of changing the trust relationships. How much detail is necessary and sufficient for other training problems remains an open question. In future work we will explore that question by expanding our modeling of the SANS Top 10[13] known security flaws.

4.3 Graphical student interface

The graphical interface to our simulation was designed for interactive training. For example, a student user can click on the icon of a host computer, which displays an interaction window (Figure 5). The student can "log in" to the simulated machine, acquire a shell, and type commands as if to a real computer. The graphical interface shows the student the system response after each command.

[10] This is a simulated packet. It does not look like an IP packet, but it has similar information and plays a similar role in carrying information from place to place on the network.
[11] "Internet" is a network object that models the Internet backbone as a wire on which packets move between other networks.
[12] Nowadays it is common to prohibit root from doing rlogin regardless of trust. However, given the misconfiguration, it is easy enough for the hacker to create an account on his machine with the same name as one on the target machine and rlogin from there. This example just simplifies the process.
[13] http://www.sans.org/topten.htm

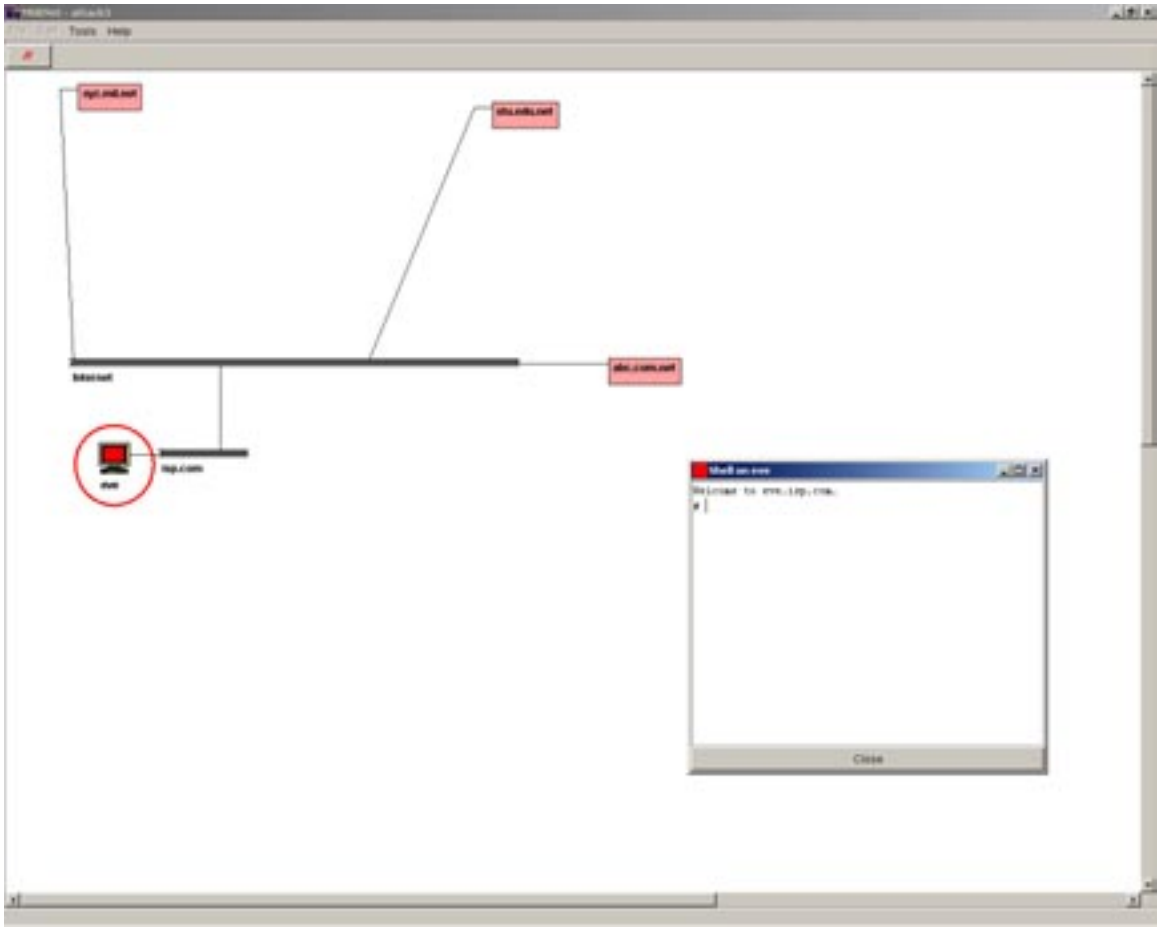

Figure 5: Initial display of network information

To meet the needs of training, the interface goes beyond simulated interaction windows. In addition to the information returned in the shell window, it presents a graphical representation of the network to help the student to visualize the effects of activities. This is a significant advance over the lab exercises in commercial courses. By changing in response to the new information, the graphical interface allows the student to understand the effects of actions and picture what a hacker would learn if the actions were taken on the actual enclave.

The most effective demonstration of the value of this is hacker reconnaissance. Figure 6 shows the results after running nmap and traceroute to figure out "the lay of the land." The stu.edu network has expanded to show the machines discovered, along with some topology of the network. In this example, reconnaissance actions determined the operating system of two of the machines. This is indicated on the graphical interface by changing Bugle's icon to indicate an NT workstation and Butterfinger's icon to indicate a Unix host. The generic icons representing the other machines indicate that the student has done nothing that tells him what they are. This "cognitive map" of his current information is particularly important for novices.

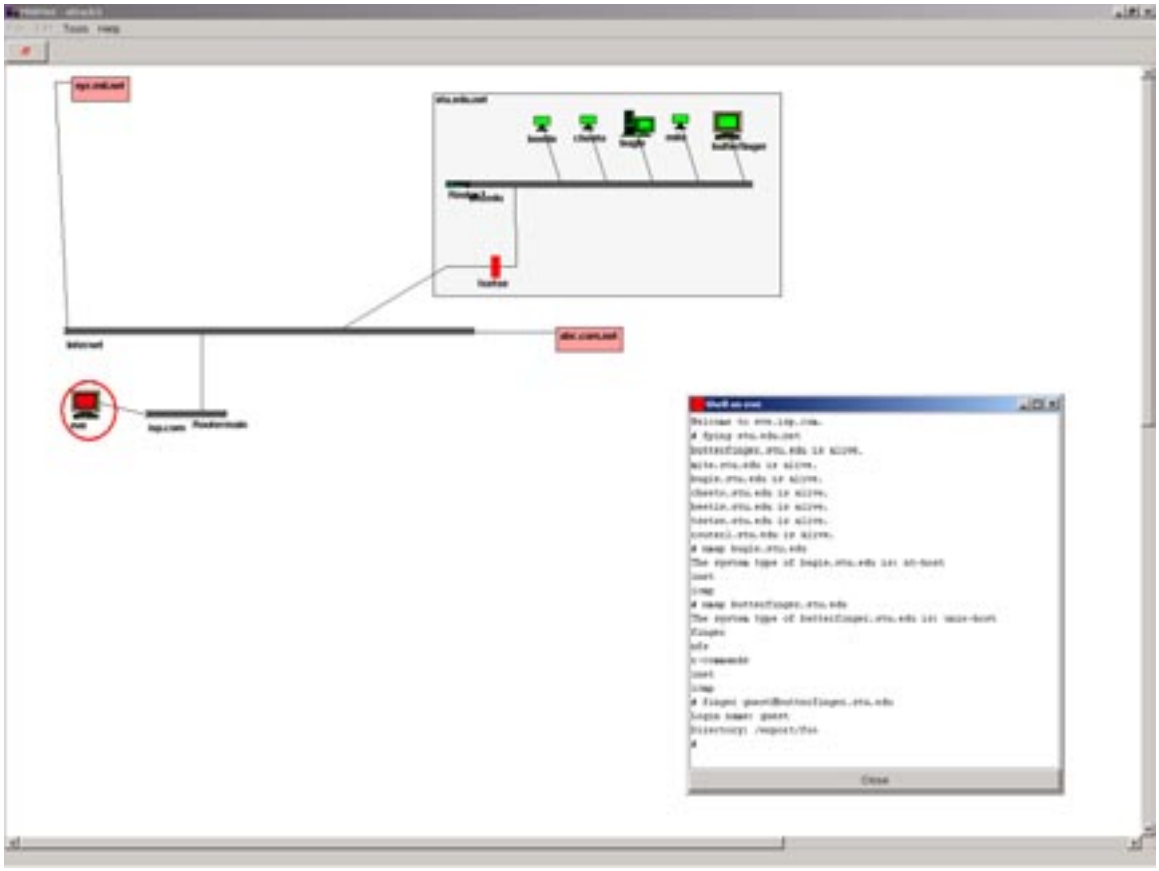

Figure 6: View of network information revised by reconnaissance activity

5. How it works

This section goes step by step through a typical exercise using our training simulation prototype. We omit description of the tools used to review static background information on different types of hackers and so on. We try to make clear what has been implemented and what is "planned."

5.1 Configure the simulator

The first step a user would do is to run mapping tools to create a configuration file containing a "situation" description. This customizes the simulation to the student's home environment. So far we have implemented an interface that translates output from Tkined to a PDDL representation.

We have yet to integrate all the mapping tools necessary to automatically create a complete configuration of an arbitrary enclave. The majority of the configuration process still requires using the graphical interface or a text editor. Our suspicion is that some external configuration may always be necessary, if only because not every student environment will have all the vulnerabilities and exposures the student should know of.

After creating a situation representing the student's home enclave, behind the scenes processing customizes a list of problems to fit the target enclave. Importantly, only problems that are relevant to the mapped enclave are instantiated,[14] although some will include vulnerabilities and exposures that might not be present, in order to train the student to be aware of what could happen in his particular enclave.

[14] The current set of attacks are based on the SANS Top Ten list. This is adequate for beginners, but as it does not include expert-level attacks it would not be adequate for advanced training.

5.2 Start the system

After opening the GUI, the student would select the laboratory workbook function. Our intention is to integrate an electronic page turning application, i.e., a computer-based training system (CBT). The student would progress through a graduated sequence of activities progressing from novice to sophisticated attacks. Loading an exercise would initialize the simulation with the corresponding problem description. This will note appropriate trust relationships, software patch levels, file contents, and so forth to enable the attacks in the exercise scenario.

Once the problem is loaded, the CBT would direct the student to run a series of attacks. At the novice level, these consist of attack scripts that the student selects from a menu. As the student advances, he will be required to carry out the attack directly. At all times the GUI will show the student what attackers see, and how they make decisions, while in the middle of an attack. This is important because defensive actions can change the attackers' view of the target systems and this is precisely what we want students to learn.

After an attack, the CBT suggests exploring the target system, and intermediate systems, looking for signs of the attack. Novice exercises will specify where to look and what to look for. Expert exercises would require the student to investigate on his own. Next, the CBT will suggest how to reconfigure the systems to protect against the attack. In novice exercises these reconfigurations will be menu driven, and command driven at the expert level. Novice exercises will have fewer reconfiguration options. For example, initially it would only have out-of-the-box options available (i.e., files and commands that come with the standard installation of the operating system). Later, after-market products, such as IDS, would be available and need configuration. After the student reconfigures the simulated network, the student would re-run the attack and examine the network for the consequences. The student would learn, by seeing the effects, how configuration changes can block attacks or make it easier to detect them.

We are considering two directions for extension of these basic ideas. We may add an automated attacker to provide an intelligent adversary to the student. For effective training, the automated attacker would have to have capabilities that are under the control of the training system, i.e., not too smart for the student's current knowledge. The basis for automated attack has been developed by other research (Elsaesser & Tanner, 2001; Ho, Frinke, & Tobin1999; Zerkel & Levitt 1996; Roberts, 1995). Another extension is to concentrate on the automated training. Rather than an off-line (i.e., not connected to the simulation) CBT, we could develop an intelligent tutoring system. This would enable the system to offer advice tailored to the students' actions, rather than the generic advice of the CBT. It would be able to do this because it would have access to the simulation and to students' interactions with the GUI. The simulation access enables it to determine what actually happened, i.e., ground truth. The GUI access enables it to see what students do. By relating these, an intelligent tutoring subsystem could recognize when student action is or is not likely to discover the truth.

6. Related Work

Several products have been developed for the purposes of observing, deceiving and in some cases collecting evidence for use in legal prosecution of intruders. Honey-nets[15], ManTraps[16], CyberCops[17], and Deception Tool Kits[18], vary from using real production quality machines and network resources to using simulated target systems in the Deception Tool Kit. The purpose of these tools is to decoy hacker activity away from real systems and to provide detailed observation of hacker activity and threat signatures for forensic analysis. In concept these systems could be used as the basis of security awareness training, but they do not solve many of the shortcomings of contemporary commercial training.

Roberts [Roberts, 1995] created a plan-based simulation of malicious intruders to generate realistic audit logs that illustrate malicious behavior. This is similar to the purpose of our simulation. The "plan-based" notion comes from using means-ends analysis to assemble sequences of actions to accomplish a (malicious) goal. This simulation is geared to training systems administrators to be able to recognize patterns of

[15] http://www2.linuxsecurity.com/feature_stories/feature_story-84.html
[16] http://www.recourse.com/download/white/ManTrap.pdf
[17] http://www.pgp.com/products/cybercop-sting/default.asp
[18] http://www.all.net/dtk/index.html

malicious behavior by showing them logs of actions that represent the behavior. Our simulation is geared to giving the students hands-on experience with the tools that hackers use and the vulnerabilities and exposures that enable the attacks to work. The attacks are not automated as in Roberts' simulation, but the associated log files are created. The research we report in [Elsaesser, 2001] adds automated attack generation to our simulation.

7. Conclusion

This research began with a computer network simulation designed to support forensics analysis. After several "Hacker 101" courses to learn about the domain, we hypothesized that a simulation like ours could replace the laboratory exercises in such courses. More thought convinced us that a simulation could do more than reduce training costs. Simulation, due to the control of ground truth, provides an opportunity for a student to see the ramifications of actions from several viewpoints. Adding automated instruction technology might even allow a simulation to customize course work to the level of knowledge of the student.

Our proof-of-concept system indicates that at least the first part of our hypothesis is true. One can simulate networked computer systems in appropriate detail to allow students to practice standard hacker techniques. A key result is that it is not necessary to emulate an entire networked computer system; it is sufficient to simulate the effects of the attacks. It is also feasible to initialize the simulator with a set of commercial tools so that the training experience closely resembles the student's home enclave.

We have yet to implement all the capabilities necessary for a complete training experience. Here is the list of what our simulator can do to date:

- Establish an identity on a host (logon, rlogin when UID is in .rhosts)
- Surveillance operations (nmap, etc.)
- Remote mount (precondition: the directory is in sharetab)
- Add accounts (precondition: write access to /etc/passwd, /etc/shadow)
- Modify files
- Use attack scripts
- Intrusion Detection
- Review log files
- Review current mounts

There is much to be done both to make our simulator all that we hypothesize it could be, as well as producing a tool that a novice could take home and use.

The "easy" part of the future work will be in adapting the simulation to real environments. Some of this work simply (!) consists of linking the simulation to commercial network-mapping tools to initialize situations with a student's own data. The challenge is to integrate the output of several network- and host-mapping tools as no single tool captures all the information needed to completely initialize the simulation. A ticklish issue we will have to overcome will be getting to such data without alarming management that the student—or our simulation—is not itself a hacker attack. Much of the data, for example, requires administrative access to privileged files. However, our initial target student population is system administrators who are likely to already have the necessary access and have a legitimate need to gather the kind of information needed to initialize the simulation.

Furthermore, it is important to collect a representative perimeter description. This entails mapping out in some degree of detail all trusted external networks and details of elements of the DMZ environment between external firewalls and internal subnets. We have not elaborated much of this process in our proof of principle demonstrations, but this will need to be addressed in our future work.

Translating vulnerability data to situations and problems will be more challenging than getting configuration data because knowing what to collect involves considerable hacker knowledge. This knowledge is what the providers of commercial training have that lets them sell their service.

The greatest potential of our simulation approach is that it allows an ideal environment for incorporating automated training technology. Here we do not simply mean on-line course materials. What needs to be incorporated to completely verify our hypothesis is automation that understands a student's level of

expertise, tracks his interactions, measures distance to pedagogical goals, and adapts the training to maximize knowledge gained. Eventually, we think one could use the simulation for evaluating performance, perhaps as part of a certification process.

References

Briggs, L. J., D. L. Gustafson, M. H. Tillman, *Instructional Design: Principles and Applications*, Educational Technology Publications, 1990.

Dick, W., L. Carey, *The Systematic Design of Instruction* (3rd Ed.), Scott-Foresman, 1990.

Elsaesser, C., M. C. Tanner, *Automated diagnosis for computer forensics*, The MITRE Corporation, in preparation, 2001.

Gagne, R. M., *The Conditions of Learning and the Theory of Instruction*, (4th ed.), Holt, Rinehart, and Winston, 1985.

Gagne, R., L. Briggs, W. Wager, *Principles of Instructional Design* (4th Ed.), HBJ College Publishers, 1992.

Ghallab, M., A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, *PDDL—the planning domain definition language*, Technical report, Yale University, 1998.

Ho, Y., D. Frincke, D. Tobin, *Planning, Petri Nets, and Intrusion Detection*, Department of Computer Science, University of Idaho, Moscow, ID, 1999.

MIL-HDBK-29612-2, *Instructional systems development/systems approach to training and education*, Part 2 of 4, 1999.

MIL-HDBK-29612-3, *Development of interactive multimedia instruction (IMI)*, Part 3 of 4, 1999.

Roberts, C. C., *Plan-based Simulation of Malicious Intruders on a Computer System*, Naval Postgraduate School, Monterey, CA, 1995.

Zerkle, D., K. Levitt, "NetKuang—A Multi-Host Configuration Vulnerability Checker", *Proc. of the 6th USENIX Security Symposium*, San Jose, California, July 22–25, 1996, pp. 195-204.