CyberCIEGE Scenario Design and Implementation

Michael F. Thompson and Cynthia E. Irvine[1]
Naval Postgraduate School
{*thompson, irvine*}*@nps.edu*

Abstract

In 2005, the initial version of CyberCIEGE, a network security simulation packaged as a video game, was released. Since then, we have developed a suite of game scenarios and have enhanced and extended the underlying game engine to cover a broad set of cybersecurity concepts. CyberCIEGE includes a Scenario Development Kit to customize existing game scenarios and create new ones. A Scenario Development Language lets instructors express security policies of interest and the circumstances in which these policies must be enforced. This language programs and augments the underlying CyberCIEGE simulation, enabling context-rich interaction with students, while relying on the simulation to assess network security and enterprise productivity.

Scenario creation requires both story telling and high-level programming techniques. Scenario designers use a forms-based integrated development environment to express a scenario in terms of its initial conditions, security policies, economic constraints, and student feedback.

1 Introduction

CyberCIEGE [10] is a construction and management resource simulation somewhat like the Tycoon series of video games [16]. Students play the role of a decision maker for some enterprise such as a small business or military command. The game includes a variety of scenarios that force students to make a series of choices that potentially affect the security of enterprise assets.

The game is well covered by published papers, including: its educational goals [7]; its application from the perspective of educators [18]; its support for expressing rich information security policies [8]; its utility as a training and awareness tool [5]; and its representation of public key infrastructure [9]. Less has been said about the techniques and mechanisms employed to create new game scenarios. The game is distributed with a *Scenario Development Kit* (SDK) and a corresponding 150-page users' guide [14]. This paper aims to describe scenario development from the perspective of an educator wishing to design or customize a scenario.

We provide an overview of the game in the next section, followed by a discussion of how to create CyberCIEGE scenarios. We then describe the various elements of scenario construction and methods for interacting with students through the scenario. Then we discuss managing the attack engine, followed by a summary of the current state of CyberCIEGE and potential future work.

2 Game overview

In a typical CyberCIEGE scenario, the *player* is the information assurance decision-maker for some enterprise. An *enterprise* may range from a large military facility, to a home office. The primary elements within the CyberCIEGE game engine are *assets*, *users* and *attackers*. Assets are information resources maintained within a system, which may range from a single host to a large enterprise network. Users are typically enterprise employees who have goals that require computerized access to assets. Students succeed by facilitating user access to assets. Some assets have substantial value to the enterprise based on their secrecy or integrity; some assets may have value based on their availability. Assets also have value to attackers, and this creates a motive that determines the effort and means the attacker will expend in attempts to compromise an asset. The motive determines the *threat*. Student choices modify the system's *vulnerabilities* and thus affect the opportunity (or lack thereof) for the attacker to compromise the assets. The enterprise (and by extension the student) is penalized the value of an asset should it be compromised or made unavailable.

[1]Approved for public release; distribution is unlimited. The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Table 1: Example mapping of learning objectives to scenario story line.

Learning objectives	Story line
Malicious software or devices can record passwords entered into computers.	A user must access sensitive corporate server from a hotels business center.
One-time password generators prevent reuse of captured passwords	
Email encryption protects email content when stored on email servers.	Internal developers have convinced management that they require root access to corporate servers, including the email server
Management mandates may limit the policies and mechanisms available for protecting information.	
Email attachments are a common cause of malicious software entering systems	A new employee must sort out the difference between malicious attachments and those that must be processed if the employee is to remain employed.
Prohibiting all email attachments may be impractical.	

Within any given scenario the users, assets, and attackers are, for the most part, fixed by the designer. Designers also specify the initial state of the scenario (e.g., an initial set of computers) and dynamic changes to the scenario (e.g., the introduction of new user goals.) Scenarios can be organized to gradually reveal various predefined users, assets, and attackers as game-play progresses.

Students see the enterprise as an animated three-dimensional representation of an office building or military headquarters. Each scenario has one main office and optional small offsite offices. Users inhabit these buildings and are represented as animated *characters* who wander about or productively sit at desks in front of computers. If computers are available, or purchased by the player, users will create and access assets using the computers and systems. This user behavior is driven by the user goals specified by the designer. If computers are networked, users may access assets over the network. Network devices such as routers enable users to access the Internet, and allow attackers on the Internet to potentially access enterprise resources. Suitably motivated attackers can enter buildings to physically compromise assets. They may compromise computer-based protection mechanisms, and may access network links to either monitor or insert traffic. Attackers may also bribe untrustworthy enterprise users to compromise assets. And users themselves may have motive to compromise assets.

The underlying network security simulation determines whether user goals are met or assets are compromised based on the parameters of the scenario and successive player choices. The designer can specify a variety of different game triggers whose execution is predicated on a rich set of measurable game conditions. These triggers include actions such as scrolling message tickers; characters "speaking" via bubble text; popup messages; multiple choice questions; video clips and 3D objects bursting into flames. Measurable conditions include game state such as whether a particular asset has been compromised, achievement (or failure) of user goals, and answers to multiple choice questions.

Players may hire guards to physically protect buildings or offices within buildings. Furthermore, players may purchase physical protection mechanisms and determine which users are permitted to access different physical areas (i.e., *zones*) within the virtual buildings. Procedural security choices affect user behavior (e.g., leaving computers logged in). Students can purchase user training to improve user adherence to procedural policies.

3 Scenario construction

Constructing non-trivial playable scenarios requires a combination of story-telling and high level programming. The scenario designer begins with a clear idea of the topics to be covered and how students will be motivated to care about the scenario outcomes. Scenarios generally benefit from a story line that encompasses the desired learning objectives, e.g., a need to communicate securely at a distance. The designer should also identify the audience, (e.g., high school students or graduate students; awareness training or education), and select the security policies to be covered. The designer then develops a story that includes circumstances that illustrate policy enforcement implications and consequences. Table 1 illustrates a few learning objective mapped to a story-line.

3.1 Understand the underlying engine

Typically, scenarios are built upon the underlying network security simulation, and the designer relies on the simulation to determine if assets are vulnerable to compromise and if users are achieving their goals. However, CyberCIEGE also includes a rich and flexible set of functions for constructing scripted scenarios that make little or no use of the underlying simulation. Instead, these scenarios branch based on game conditions and player choices. Much of the discussion below assumes the designer will make use of the underlying simulation.

Before attempting to build a scenario, we suggest a series of steps. First, play several existing scenarios of varying complexity to become familiar with the game elements. As part of this step CyberCIEGE will generate a game log each time a scenario is played. Second, construct a relatively simple scenario such as the tutorial scenario detailed in the SDT User's Guide [14].

Finally, go back and replay existing scenarios while identifying the genesis of each game event through analysis of that scenario's scenario definition forms in the SDT, and review of the game logs.

The steps listed above should help the potential designer to understand many of the game features and the general manner in which scenarios are constructed. From there, the designer begins to formulate the particulars of a new scenario, informed by an appreciation of the capabilities and limitations of the Scenario Definition Language (SDL). Imagine the scenario play as a story having many potential branches, much like the *Choose Your Own Adventure* books, e.g *The Cave of Time* [15]. Formulate a narrative that explains the initial state of the scenario, i.e., what the student sees and can explore without commencing the simulation. And play the story forward in your mind following different paths determined by student choices. Organize the story into a set of phases, each having one or more objectives that must be accomplished to complete the phase. In other words, you should be able to approximate scenario play in your head (or a state diagram) before it is codified in the SDL.

3.2 Express the scenario in the SDL

Mechanically, scenarios are constructed using an integrated development environment called the *Scenario Definition Tool* (SDT). [14] This enables forms-based selection and definition of game elements without requiring the designer to learn the SDL syntax. A scenario is defined by a collection of forms which can be treated as the source code of the scenario.

Broadly, the scenario designer uses these forms to define the initial game state in terms of users, assets, policies, initial networking components, purchasable components and the physical layout of the simulated environment. The designer must also define the conditions by which a student will achieve scenario-specific objectives and proceed through scenario phases. And, while the game engine automatically generates some feedback informing the student of asset compromises and goal failures, this should be augmented through the use of game triggers, e.g., causing a computer to burst into flames if the player makes a particularly egregious choice.

4 Elements of scenario design

This section covers the major elements of typical CyberCIEGE scenario construction. It describes CyberCIEGE game elements and the relationships between them. Completed SDT forms define the scenario. (See the SDT users guide [14] for details.)

Define assets. Identify the information assets that the player must protect and make available for access by game characters. Assets have a variety of attributes that affect whether they will be compromised and the impact to the enterprise if they are compromised [8]. These attributes allow the scenario designer to express policies for each asset based on secrecy, integrity and availability. Provide a description of each asset to inform the student of its purpose and value to the enterprise, and the potential motivation of attackers to compromise the asset.

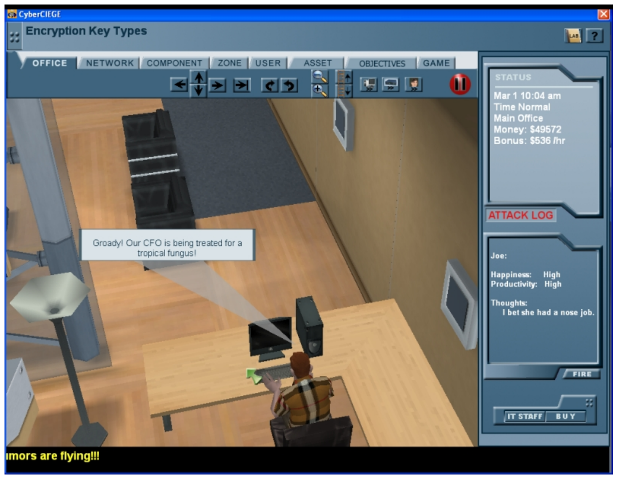
The scenario designer may initially allocate assets to existing computers, or have game characters create assets once the simulation commences and a suitable computer is available. At the start of a scenario, assets need not exist (or be visible to students). They can be introduced as the scenario progresses by assigning some character a goal that involves accessing the asset.

Define game characters. Select game characters who will access the assets. Assign names and provide descriptions for each character. Most importantly, identify the specific assets the character must access, the modes of access and the types of software that must be available while accessing the assets. This provides the productivity side of the trade-off between security and getting work done. SDT Goal forms define access modes; software types; and the potential costs to the enterprise if characters cannot access assets. SDT User forms allocate these goals to specific game characters as a percentage of the character's potential overall productivity.

If multiple characters must access the same asset, then each character must have access to a computer that can reach the asset. This provides scenario designers with a means to constrain the topology of successful network designs. A goal that requires a particular software type can include a "filtered" attribute that indicates the software must reside on the same computer as the asset. Network filters may block remote access to an asset by blocking application software types, and thus, filtered goals can fail based on network filter settings.

Characters can come and go over the course of a scenario. If all of a character's goals have a productivity contribution of zero, then the character is not visible to the student. Game triggers dynamically adjust a character's productivity dependency on each goal, and thus can cause the character to appear or disappear.

Decide whether each character will have a predefined workspace, (with or without a computer), or if the char-



Encryption Key Types

Mar 1 10:04 am
Time Normal
Main Office
Money: $49572
Bonus: $536 /hr

Groady! Our CFO is being treated for a tropical fungus!

Joe:

Happiness: High
Productivity: High

Thoughts:
I bet she had a nose job.

ımors are flying!!!

Figure 1: CyberCIEGE screen shot of a typical office that shows a user in deep in thought.

acter will just stand or wander until the student assigns a workspace. A potential source of student frustration is the need to make many inconsequential purchases and seating choices. More advanced scenarios should avoid construction steps that don't affect scenario outcomes.

Some characters may be malicious insiders motivated to attack enterprise assets. These motives can be derived from the asset attributes that drive external attackers, or may be character-specific. Insiders may target assets for which they have valid access or those that the enterprise policy constrains them from accessing. Students can purchase background checks for groups of users based on which assets the user must access. The designer can employ a trigger to replace untrustworthy characters as a result of these checks. The designer must leverage the game economy to prevent the student from replacing all potentially malicious insiders with more trustworthy characters.

Another variety of game character offers students advice and guidance. The *Speak triggers* generate bubble text and optionally pause the game and move the camera to the speaking character. The designer should experiment to create a balance between useful game pauses and those that disrupt the continuity of game-play.

Define the physical environment. The visible area containing game elements is a single X-Y coordinate grid. This grid contains each building, (including a main site and multiple smaller offset offices.) The SDT Scenario form defines the positions and types of buildings. Each building contains workspaces with desks, computers, and server racks. Workspaces are enumerated in the Workspace form and assigned an index value, which is used to define the initial locations of game characters.

Each building is divided into one or more rectangular zones defined in terms of X and Y coordinates. Typically, zones correspond to offices. The designer must ensure that these fall along wall boundaries. The designer must also ensure that characters have room to walk between offices, e.g., that desks do not block doors.

Artwork exists for four different main sites and three different offsite offices. Figure 1 shows one of the main site office buildings.

Physical security may rely on armed guards. The scenario designer identifies whether guards will be present in the scenario, and if the student is able to hire guards. Guards may also man physical checkpoints.

Character Animation and Interaction. Game characters have a limited number of animated actions. In general, if a user has an assigned workstation, the corresponding animated character sits at it and works. If the user is failing to achieve goals, the character pounds the keyboard. If the user lacks an assigned computer, it may wander the office, or if some unassigned computer can be used, the character will sit at that computer. Some user and scenario settings allow characters to wander periodically while achieving goals. Users create assets as needed, but do not necessarily interact with associated computers when creating assets, viz., user interactions with assets are not synchronous with their animations.

Characters interact with zone checkpoints. When checkpoints are defined for a zone, a user cannot enter the zone without passing the checkpoint, which may require interacting with a device such as a card reader or iris scanner. Character entry into zones is synchronous with the animations. However, a character can achieve asset goals without being physically inside the zone the character only needs to be able to enter the zone.

Initial network and catalogue. SDT Component forms define the initial set of computers and network equipment, and their initial configuration, including network connections and software. The SDT Network form identifies each different physical network. Consider whether the initial state of the network will be secure with respect to the policies of interest, or whether the student will be required to assess and mitigate vulnerabilities prior to starting the simulation.

The SDT Catalogue form determines what equipment the student can purchase, and its cost. The initial cash-on-hand in the Scenario form can be set to limit equipment options. Additionally, triggers can introduce new items into the catalog as the game progresses.

Each computer and most network components include operating systems. The game includes a variety of OSes having differing levels of assurance and policy enforcement functions. The designer allocates operating systems to initial computers and those available in the catalogue. Students cannot switch a computer's OS.

The SDT Applications form defines the applications software that can be installed on components, either initially or purchased by the student during the game. The designer defines different applications in terms of the software's integrity (e.g., software security during its development and distribution), and its need for patches.

Software can include cryptographic functions that protect assets read and written over networks by either peer-to-peer or client-server applications. The strength of these protections is based on the integrity of the software.

Interact with the student. When a scenario begins, its initial briefing is displayed first. This should provide a brief introduction to the scenario. Since many students won't read more than a sentence or two, don't rely on the briefing, (or any other single display of text) to convey information necessary to play the scenario. Descriptions of game characters, assets and zones are also available before the simulation starts.

In our experience, the most frequented source of textual information is the Objectives screen, which describes the unmet objectives for the current game phase and provides brief summaries of upcoming game phases. Goal-oriented students look here to understand what they must do to get to the next step. This is also a good place to reference other sources of key information such as, "Refer to the Assets screen for the Mocha recipe's value."

Objectives and phases are the primary tool for organizing a scenario's flow and giving the student a sense of progress. For each objective to be met, the game condition state that must exist should be identified. This may include conditions such as the passing of some amount of simulated time with all user goals met and no asset compromises. The *SetObjectiveStatus* triggers control whether the game engine views any given objective as being completed. Similarly, the *SetPhase* trigger causes the game to enter a new phase. The firing of these triggers depends entirely on designer-defined conditions. For phases with multiple objectives, the student may meet one objective, but then backslide while attempting to meet another. A scenario may need triggers to reverse the completion of an objective.

The SDT offers over eighty game conditions for use when determining if a trigger should fire. Within a given scenario, only a subset of the conditions might be used. Many of the conditions are intended to allow the designer to provide the player with feedback prior to an attack engine action. For example, the condition that measures the number of open application ports in a router can be used to warn the player. Use of that (or any other) condition is not necessary to cause the game engine to attack assets. Similarly, some conditions measure a user's ability to achieve a goal. Such conditions are not needed to penalize the player for unproductive users - rather it can be used to provide the player with context-specific help.

Most conditional state assessment is straight-forward. For example, a simple condition measures if the component containing some asset has a specific policy. It is easy to assess whether a user is achieving a specific goal, and whether the user has achieved the goal for a given duration. The question of whether a given asset has been attacked (i.e., the *AssetAttacked* condition) is subtler. (See the Trigger form description in the SDT users guide [14].) The *AssetAttacked* condition value depends on whether a successful attack occurred subsequent to the previous evaluation of the trigger that references the *AssetAttack* condition. This state management is local to each trigger, and is intended to allow the scenario designer to avoid reporting the same attack multiple times.

Measuring time. The passage of time can be measured via the *Time* condition, either from the beginning of the scenario or from the beginning of the current phase. Also, each Trigger form includes fixed and random delay values that allow the designer to incorporate time into the decision to fire a trigger. The semantics of these delays is quite different from the *Time* condition. The condition simply determines if a given amount of time has passed without regard to other game conditions. In contrast, per-trigger delays allow the designer to prevent a trigger from firing until all the trigger's conditions have been met continually for the period specified by the delay.

Since the designer controls the frequency of *AssetAttack* triggers, the designer can determine that the asset has indeed been protected from one or more attacks. Similarly, the designer can determine that the users have had uninterrupted access to the assets for a specified period. Thus, delays can be used to determine if game state conditions have persisted over some period during which some defined set of events have occurred.

If the trigger's condition evaluation changes to a false state during the specified delay period, the trigger will not fire. To cause a trigger to fire some fixed time after some game state has been reached (or some fixed time after the game state has persisted per a delay value), split the trigger into two triggers: the first fires when the conditions are met and should be invisible to the player (e.g., a *LogTrigger*); the second trigger would then include the first triggers firing as an intrinsic condition and the second trigger's specified delay controls the period between the first trigger's firing and the firing of the second.

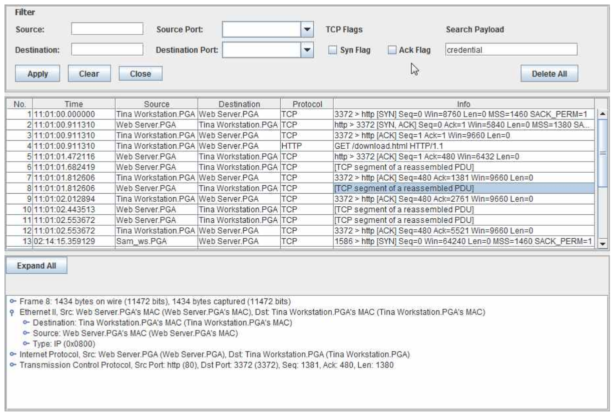
Attack and vulnerability visualization. Automated, generalizable visualization of vulnerabilities and attacks is challenging. We explored this topic [17], and concluded it was unacheiveable without creating distractions and disrupting game flow. The game engine does not create graphical representations of attacks or system vulnerabilities. That is allocated to animated tutorial videos displayed by triggers that fire based on scenario designer-defined game conditions. Judicious use of this type of "cut scene" is suggested because they can take the student out of the context of the game. It is sometimes better to let the student choose when to view such material. Our experience indicates that is more important to give the student a clear means by which to discover cause and effect. Students can reference a game engine-managed attack log any time after a successful attack. It identifies the source of the attack and the reason for its success.

When assets are compromised, the in-game economy reflects the losses in the player's current cash-on-hand. Compromise of valuable assets can be catastrophic, and, depending on the scenario design, fatal, requiring the player to restart from some previous game state. As the game evolved, informal player feedback made clear that sudden loss without warning leads to frustration, particularly in early scenarios played by less experienced students. In many cases, game conditions can be used by the scenario designer to detect impending doom and warn the player. For example, a game condition might report on whether a particular asset can be reached from a given network using a specific application protocol. We continued to add this type of game condition to the SDL, but eventually realized we were often trying to predict the outcome of the simulation, which can become complex as students deploy multiple components and adjust configurations in attempts to ward off attacks. To simplify the issuance of warnings, we introduced a means of generating benign attacks that are not detected by the in-game economy engine, but are reported as conditions.

Network traffic analysis. A relatively new CyberCIEGE feature allows the scenario designer to provide the student with insight into game events through network traffic analysis. [2] PCAP files of actual network traffic can be imported into a scenario and appear as traffic captured at one of the simulated network connections.

The PacketXform form maps in-game component names to generic component names found within network traffic packet samples. These forms (and the network packet samples) are named by PacketXform triggers. The transformed packets are written into a packet log for the selected component. These packet logs can be viewed using the *CyberChark* tool, illustrated in Figure 2, which incorporates many Wireshark [4] functions.

Dynamic student assessment. *Question* triggers can test a student's understanding of material and alter the direction of a scenario. Questions can be either true-false or multiple choice. There are two forms of *Question* triggers. The *Question* class of trigger assumes the scenario designer will create responses using other triggers. The *QuestionMult* trigger makes reference to a question form that defines the question and the responses (i.e., pop-up messages to display in response to each player answer.) Both forms of *Question* trigger are assigned what should be a unique Register condition. A *register* is a named state variable with an initial value of zero. When a *Question* trigger fires, its associated register is assigned the value of the keyboard response provided by the player.



Search Payload: credential

No.	Time	Source	Destination	Protocol	Info
1	11:01:00.000000	Tina Workstation.PGA	Web Server.PGA	TCP	3372 > http [SYN] Seq=0 Win=8760 Len=0 MSS=1460 SACK_PERM=1
2	11:01:00.911310	Web Server.PGA	Tina Workstation.PGA	TCP	http > 3372 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380 SA...
3	11:01:00.911310	Tina Workstation.PGA	Web Server.PGA	TCP	3372 > http [ACK] Seq=1 Ack=1 Win=9660 Len=0
4	11:01:00.911310	Tina Workstation.PGA	Web Server.PGA	HTTP	GET /download.html HTTP/1.1
5	11:01:01.472116	Web Server.PGA	Tina Workstation.PGA	TCP	http > 3372 [ACK] Seq=1 Ack=480 Win=6432 Len=0
6	11:01:01.682419	Web Server.PGA	Tina Workstation.PGA	TCP	[TCP segment of a reassembled PDU]
7	11:01:01.812606	Tina Workstation.PGA	Web Server.PGA	TCP	3372 > http [ACK] Seq=480 Ack=1381 Win=9660 Len=0
8	11:01:01.812606	Web Server.PGA	Tina Workstation.PGA	TCP	[TCP segment of a reassembled PDU]
9	11:01:02.012894	Tina Workstation.PGA	Web Server.PGA	TCP	3372 > http [ACK] Seq=480 Ack=2761 Win=9660 Len=0
10	11:01:02.443513	Web Server.PGA	Tina Workstation.PGA	TCP	[TCP segment of a reassembled PDU]
11	11:01:02.553672	Web Server.PGA	Tina Workstation.PGA	TCP	[TCP segment of a reassembled PDU]
12	11:01:02.553672	Tina Workstation.PGA	Web Server.PGA	TCP	3372 > http [ACK] Seq=480 Ack=5521 Win=9660 Len=0
13	02:14:15.359129	Sam_ws.PGA	Web Server.PGA	TCP	1586 > http [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1

```
Frame 8: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits)
Ethernet II, Src: Web Server.PGA's MAC (Web Server.PGA's MAC), Dst: Tina Workstation.PGA's MAC (Tina Workstation.PGA's MAC)
    Destination: Tina Workstation.PGA's MAC (Tina Workstation.PGA's MAC)
    Source: Web Server.PGA's MAC (Web Server.PGA's MAC)
    Type: IP (0x0800)
Internet Protocol, Src: Web Server.PGA (Web Server.PGA), Dst: Tina Workstation.PGA (Tina Workstation.PGA)
Transmission Control Protocol, Src Port: http (80), Dst Port: 3372 (3372), Seq: 1381, Ack: 480, Len: 1380
```

Figure 2: Students use CyberChark to analyze network traffic, e.g., by setting filters to view selected traffic.

The *QuestionMult* triggers should be used to give player feedback as pop-up messages. The Question forms simplify question and response management for such questions. If the player feedback will not include pop-up messages, then Question triggers, wich require additional logic, should be used.

When using *Question* triggers, designers trigger responses to player-provided answers based on the value of registers. For example, a *speaks* trigger might cause a user to say: `correct!` when the *Question* trigger's register has a selected value, e.g., *b*. The response to an incorrect player answer can be uniquely defined for each potential incorrect answer. Or a single *wrong, ... response* trigger can be designed to fire when the associated register value is *NOT 0* and *NOT b*.

When responding to individual incorrect answers, it may be useful to re-ask the question until the player selects the correct reply (see the SDT Users Guide [14].)

Attack triggers. When an *Attack* trigger fires, all assets are attacked via the defined attack type by all attackers (including insiders where appropriate), in the modes prescribed by the asset's motive values. Each asset will not be compromised more than one time as an effect of an *Attack* trigger firing. The designer can also cause the attacks on other assets (due to this same trigger firing) to cease subsequent to a compromise and delay the evaluation of an *Attack* trigger following any asset compromise. The trigger frequency determines the delay period (e.g., a frequency of 0.05 ensures the trigger cannot possibly fire within 0.05 days of the most recent asset compromise.)

The order in which assets are attacked is based on the motive that an outside attacker has to compromise the asset in any mode, with higher motive attacks occurring first. Insider motives do not affect the order of attacks.

5 Understanding attacks

The CyberCIEGE attack engine simulates a broad range of attacks, including: Trojan horses, viruses, subversion of protection mechanisms, unpatched software flaws, insider attacks, network-based attacks and physical attacks. A frequent question has been how we planned to update the game to reflect new attacks and counter-measures. Our answer is that the fundamental nature of attacks has changed little since the seminal vulnerability identifications decades ago, e.g. [19, 1, 12], although the size and complexity of the attacks and their target systems have increased dramatically. The most challenging issue is the potent mix of vulnerable and malicious software.

Malicious software. The CyberCIEGE simulation largely represents ongoing problems associated with malicious software. As in the real world, malicious software often exploits vulnerabilities in poorly designed and implemented systems. However, the CyberCIEGE simulation includes certain counter-measures not available in the real world, e.g., high assurance trusted systems. Within the game, these protections appear in selected components, which the scenario designer can include or omit from a given scenario.

We assume that most application software can contain Trojan horses, and that most operating system services contain flaws or trap doors. In contemporary terms, we assume that access points and tools have been planted as part of advanced persistent threat (APT) operations. This perspective influenced the network simulation treatment of assets, where high value can strongly motivate attackers. Obviously, not all environments include assets of sufficient value to attract APTs, so a variable motive is a central construct of the simulation. Yet, for high-value assets, a trustworthy security mechanism can make the difference between protection and compromise.

Software flaws. A major element of the simulation is the modeling of software flaws that result either in direct compromise of assets or the implantation of malicious software. The precise nature of the flaw is unimportant: from the students perspective, the effects are the same, and they are likely to occur, particularly for a motivated attacker. In CyberCIEGE, the player can observe programs that might be susceptible to data-driven attacks because they are not patched. A network scanning function provides a view similar to tools based on *nmap* [13]. The scenario designer indicates the frequency at which patches are required for a given application, and that is compared to the student-selected patch policies. One assignable attribute value reflects an insatiable need for patches, hence such applications must then be kept from attacker-facing software, e.g., by using network filters.

Software flaws resulting from a lack of application software integrity are invisible to students unless explicitly described by the designer, e.g. as a scenario [3]. By allocating distinct patching and integrity attributes to software applications, the designer can simulate environments in which students must institute software patching policies and limit exposure of some software to potentially hostile data in order to avoid known flaws; while simultaneously living with the threat of zero-day exploits.

6 Educational efficacy

Earlier work summarized studies conducted toward measuring the efficacy of CyberCIEGE [18, 6, 11]. There we described game event logging and a student assessment tool that is available to educators. Unfortunately, we have lacked the resources to pursue the assessment tool enhancements described in that paper.

In addition to the previously detailed challenges associated with the assessment of an educational tool such as CyberCIEGE, the game's encouragement of experimentation and failure also complicate assessment. For example, how does an automated tool, (or even a trained instructor), distinguish between a student who is greatly confused and one who is experimenting with the consequences of deliberately wrong choices? Along similar lines, what is the effect on student interaction with the game if the student learns that the logs will be scrutinized? Clearly, care is needed to design appropriate experiments and avoid conflated results.

7 Availability and future work

CyberCIEGE is freely available for use by the U.S. Government, and educational institutions have a no-cost educational license. Educators in the government, universities, community colleges and high schools request CyberCIEGE at a rate of several times a week, generally due to word-of-mouth and web searches. Directions for requesting CyberCIEGE may be found at

`http://cisr.nps.edu/cyberciege`

CyberCIEGE runs with WINE on Linux systems [20]. Its native platform is Windows, either running natively or on VMs hosted by VMWare or Parallels.

We continue to seek sponsorship to adapt the CyberCIEGE simulation to incorporate wireless devices, (e.g., access points and user devices), to design scenery and scenarios that appeal to females and younger students, to conduct further assessments of its utility, and to create a browser-based port of the game for use on mobile devices (e.g., tablets).

8 Acknowledgments

The authors are grateful to the many educators who have used CyberCIEGE and whose feedback made this paper possible.

References

[1] ANDERSON, J. P. Computer security technology planning study. Tech. Rep. ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. (Also available as Vol. I,DITCAD-758206. Vol. II, DITCAD-772806).

[2] CHANG, X. S., AND CHUA, K. Y. A CyberCIEGE Traffic Analysis Extension for Teaching Network Security. Master's thesis, Naval Postgraduate School, Monterey, California, December 2011.

[3] CHAY, C. A cyberciege scenario illustrating software integrity and management of air-gapped newtworks in a military environment. Master's thesis, Naval Postgraduate School, Monterey, CA, December 2005.

[4] COMBS, G. Wireshark. http://www.wireshark.org/, Last accessed 30 April 2014 2006.

[5] CONE, B. D., IRVINE, C. E., THOMPSON, M. F., AND NGUYEN, T. D. A video game for cyber security training and awareness. *Computers and Security 26*, 1 (2007), 63–72.

[6] FUNG, C. C., KHERA, V., DEPICKERE, A., TANTATSANAWONG, P., AND BOONBRAHM, P. Raising inforation security awareness in digital ecosystem with games - a pilot study in thailand. In *Proceedings of the Second IEEE International Conference on Digital Ecosystems and Technologies (DEST 2008)* (2008), pp. 375–380.

[7] IRVINE, C. E., AND THOMPSON, M. F. Teaching objectives of a simulation game for computer security. In *Proceedings of Informing Science and Information Technology Joint Conference* (Pori, Finland, June 2003).

[8] IRVINE, C. E., AND THOMPSON, M. F. Expressing an information security policy within a security simulation game. In *Proc. 6th Workshop on Education in Computer Security: Avoiding Fear Uncertainty and Doubt Through Effective Security Education* (Monterey, California, July 2004), C. Irvine, Ed., pp. 43–49.

[9] IRVINE, C. E., AND THOMPSON, M. F. Simulation of PKI-enabled Communication for Identity Management Using CyberCIEGE. In *Proceedings of the 2010 Military Communications Conference (MILCOM 2010)* (San Jose, CA, October 2010), pp. 1758–1763.

[10] IRVINE, C. E., THOMPSON, M. F., AND ALLEN, K. Cyberciege: Gaminig for information assurance. *IEEE Security and Privacy 3*, 3 (May 2005), 61–64.

[11] JONES, J., YUAN, X., CARR, E., AND YU, H. A comparative study of cyberciege game and department of defense information assurance awareness video. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)* (March 2010), pp. 176 –180.

[12] KARGER, P. A., AND SCHELL, R. R. Multics security evaluation: Vulnerability analysis. Tech. Rep. ESD-TR-74-193, Vol. II, Information Systems Technology Application Office Deputy for Command and Management Systems Electronic Systems Division (AFSC), Hanscom AFB, Bedford, MA 01730, 1974.

[13] LYON, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.*, 3rd "zero day" edition ed. Insecure.com LLC, 2008.

[14] NAVAL POSTGRADUATE SCHOOL. CyberCIEGE Scenario Development Tool User's Guide. http://cisr.nps.edu/cyberciege/downloads/sdt.pdf, Last accessed 17 April 2014.

[15] PACKARD, E. *The Cave of Time*. Bantam Books, 1979.

[16] ROLLINGS, A., AND ADAMS, E. *Fundamentals of Game Design*. Prentice Hall, 2006.

[17] SLEDZ, D. A., AND COOMES, D. E. A dynamic three-dimensional network visualization program for integration into CyberCIEGE and other network visualization scenarios. Master's thesis, Naval Postgraduate School, Monterey, California, June 2007.

[18] THOMPSON, M., AND IRVINE, C. Active learning with the cyberciege video game. In *Proceedings of the 4th Conference on Cyber Security Experimentation and Test* (Berkeley, CA, USA, 2011), CSET'11, USENIX Association, pp. 10–10.

[19] WARE, W. H. Security controls for computer systems: Report of defense science board task force on computer security. Tech. Rep. R-609-1, Rand Corporation, Santa Monica, CA, 1970.

[20] WINE. http://www.winehq.com, June 2000.