Adapting CyberCIEGE for Ubiquitous Platforms

CyberCIEGE is a network security simulation packaged as an educational video game used by educators world-wide to help teach cybersecurity concepts and information protection strategies. More than twenty different scenarios address a variety of security topics ranging from basic training, (e.g., the risks of email attachments and links), to network security architectures built around virtual private networks and public key infrastructure. CyberCIEGE is used by US government agencies and military services, (who have rights to unlimited government use), as well as educators in universities, community colleges and high schools, (through a no-cost educational license).

Goal: Port a successful tool to expand its use

The Naval Postgraduate School (NPS) originally oversaw implementation of CyberCIEGE for use on Windows platforms over ten years ago. During the intervening years, NPS has extended and enhanced the simulation engine and has developed many game scenarios using our publicly-available Scenario Development Kit. However, CyberCIEGE remains bound to the Windows platform. Below, we describe a plan to port CyberCIEGE to the Unity3D game engine, which will make the game available for use on a wider range of computing platforms, to eventually include tablets and smart phones.

In addition to broadening the set of platforms running CyberCIEGE, the port will simplify extending the game to include new artwork, virtual characters, and simulated environments. And the port will enable eventual delivery of CyberCIEGE to web browsers from centralized servers, eliminating the need to install software to benefit from this powerful educational tool.

A demonstrated strategy

Much of the initial design work and proof-of-concept prototyping for this porting effort has already been accomplished through a joint project with NAVSEA. That project identified and demonstrated a porting strategy that allows us to re-use over 10,000 lines of complex code within the network security simulation, greatly reducing develop costs and risk. The CyberCIEGE game engine has been separated into two parts: the network security simulation (NSS), and the UI component. The NSS now executes as a separate process and it implements the full logic of the network security simulation including topology traversal, the game economy, and the sophisticated conditional trigger subsystem that coordinates interaction with the user and manages game flow. The NSS uses standard communication sockets to interact with the UI component, which is implemented with the Unity3D game engine. The UI component is responsible for all visual presentation and the direct interaction with users.

An initial UI component has been implemented using Unity3D, and this provides enough interaction with the simulation engine to play the major elements of several existing scenarios. The UI component includes several prototyped user interface dialogs. The remaining work primarily involves development of user interface controls and integration of existing artwork into the Unity3D environment. A summary of the remaining development plan follows:

- Complete port to Unity3D on Windows.
    - Refine previously developed Unity3D user controls.
    - Re-implement remaining Java-based dialogs within Unity3D, including VPN and PKI dialogs and multiple choice questions.
    - Refine and test simulation engine interaction with Unity3D components
    - Import existing game art, including virtual environments and animated characters, and integrate within the new 3D space.
- Adapt the simulation engine for execution in Linux and OSX environments. Since these functions have been stripped of UI dependencies, the porting will be a straight-forward replacement of basic C-library interfaces. Since the Unity3D implementation of the UI component will run on OSX and Linux systems, this modest step of adapting the simulation engine will yield a complete game that runs on three platforms.
- Adapt the simulation engine for Android execution environments.
- Extend the Scenario Development Kit to include tools to aid in the introduction of new artwork, environments and animated characters into the CyberCIEGE scenarios.

Separating CyberCIEGE into its UI and NSS components also makes it possible to consider alternate UI implementations. Part of the NAVSEA project included a feasibility assessment of porting CyberCIEGE to work within the U.S. Navy's "Virtual Worlds" education platform. That effort is described in Appendix A.

Maintenance for Multiple Target Platforms

Supporting any product on multiple platforms can potentially add substantial ongoing maintenance costs associated with re-implementing extensions and enhancements for multiple execution environments. The proposed strategy greatly reduces maintenance costs by leveraging the Unity3D multi-platform support so that changes to the UI component need only be made once. And, since the simulation engine code is now independent of presentation layer dependencies, (e.g., DirectX vs OpenGL), changes to the simulation engine can be made to a single shared code base and recompiled for each target environment.

The work done jointly with NAVSEA preserved the structure of the existing CyberCIEGE game shares a common code base with the newly independent simulation engine. Thus, ongoing enhancements and extensions to the simulation portion of the game are automatically incorporated into porting work. Once the first step of the above development plan is completed, the original CyberCIEGE game engine can be retired.

Estimates of Effort

The Unity3D porting work summarized above would require about nine staff months to complete. An alternate port to the Navy's Virtual World platform would require about five staff years. Either effort would benefit from developers with experience creating user interface layouts for the respective platforms.

Detailed Porting Considerations

The CyberCIEGE game engine is about 15K of C++, and several thousand lines of Java that implement several of the in-game dialogs. Additionally, there are several thousand lines of Java implementing support tools, e.g., the SDT, Campaign Player, etc.

The original game engine is (mostly) a single process that loops through a dynamically determined set of functions. Broadly, the game engine can be seen as having two major functions, a network security simulation and a user interface, (UI), that includes a 3D virtual environment with several animated characters. This user interface is implemented on a proprietary set of libraries that include 3D functions that rely on Microsoft's Directx9 graphical functions.

The network security simulation includes:

- Network topology management and traversal, including simulation of workstations, servers, routers (having network filters), VPNs, email services, and PKI components.
- An economy engine that tracks user (character) goal achievement, penalties, bonuses resource costs;
- An attack engine that assesses information flow, security policy implementations, attacker motives, and vulnerabilities.
- A game condition and trigger subsystem that alters game state and provides player feedback based on game conditions and events.

High Level Porting Strategies

The UI and network security simulation are mostly implemented within distinct sets of functions, however, the functions are tightly integrated within the game's main processing loop, with significant mutual dependencies.

Re-implementing the network security simulation within Unity would not be a simple language translation. The game engine includes significant use of global data structures that would require redesign within C#. The effects on performance of such a port are hard to estimate.

Porting the UI to a new platform might be easiest if the network security simulation were also ported to the same platform, thereby enabling direct procedural interfaces in whatever direction is expedient. On the other hand, restructuring the interfaces between the UI and the network security simulation might benefit the ongoing maintenance and extension of each.

The existing game engine implementation includes Java dialog interfaces for some of the more complex components. Players interact with these dialogs to configure network filters, VPNs, PKI elements and email & SSL settings. These interfaces use the Java Native Interface (JNI) to provide procedural interfaces to a set of Java classes that consume and modify XML files shared between the Java elements and the game engine. These XML file interfaces might serve as a model for network-based interfaces between the a UI-less game engine and a Unity3D UI.

Results of Preliminary Porting Effort

All UI functions were removed from the original CyberCIEGE game engine through the use of ifdef's, allowing us to maintain a single code base. The resulting NSS configuration of the game engine includes a new IPC module for interacting with the UI component. Most of the input to the NSS from the UI is modeled after the replay log playback functions previously built into the game engine to allow replay of game sessions based on collected logs. The existing XML-based log formats are repurposed, allowing the game engine to receive and process directives from the UI, e.g., (configuration changes to virtual components) in the same manner as it processes them during log-driven game playback, (which was implemented to facilitate development and automated testing.)

The UI component is implemented within the Unity3D framework. This currently consists of about 2400 lines of C# code within 30 scripting files, representing the primary UI abstractions within the game, e.g., computers, objectives, configuration dialogs, tool-tips, etc. These files provide the basic scaffolding for most of the UI functions, with most of the remaining work being UI layout and the look and feel.

UI Dialogs

The following enumerates UI dialogs within the game. Those dialogs currently implemented using Java, (and having XML interfaces to the network security simulation) are noted with "***".

UI dialogs related to network components

- computer configuration
    - procedural checklists
    - configuration checklists
    - passwords (length, change frequency, complexity)
    - authorizations (sub dialogs for user IDs, authentication servers)
    - ACLs on local assets
    - Network link access controls (including MAC labels)
- Network filters (subdialogs for exception lists) *** (complex dialog & data access methods)
- VPN clients & gateways, connection profiles, with subdialogs for ***
    - testing connections
    - selecting PKI roots
- PKI ***
    - CA certificate signing, root selection

    - email settings
    - SSL settings
- IDS ***
- Link encryptors
- Network /Port scan and discovery ***
- Wireshark-like dialogs (multi-pane) ***
- Purchase of components
    - servers
    - workstations
    - network devices
    - ID devices (e.g., iris scanner)
- Network connections (attach/detach to lan segments)
- Software purchase, removal
- Virus scan, reimaging

Other dialogs

- multiple choice questions ***
- yes/no dialogs ***
- information messages ***
- User clearance background checks
- User status display (goals, training)
- Phases and Objectives
- Briefings
- Zone security settings
    - user access permission

    - physical settings (checklist, guard hiring)
    -

- Attack log ***

Interactions between the network security simulation (NSS) and the UI

Use sockets between UI and NSS with a simple protocol:

4 byte length packet followed by data. Data is typically a message type and a message, separated by a colon. The protocol is implemented in the uiSocket.cpp source code file within the simulation engine. On the UI component side, it is implemented by IPCManagerScript.cs.

From UI to NSS:

- Component, zone and user configuration settings.
- Connection & disconnection of networks to components
- Purchase & placement of components
- Movement and scrapping of components
- Reassign user to new workspace
- Hire/fire IT & security staff
- Pause / Play
- Save / Load games
- Animated character position intersects with
    - checkpoint
    - specific component
    - specific other character

From NSS to UI:

- Time passing
- Penalty / Bonus / Monthly costs / cash on hand

- User goal/productivity/happiness status
- Attack log events
- User
    - thoughts
    - speaks (support withdrawal, naming)
    - animation selection (stand, sit, type, walk, idle)
    - walk destination
- Phase & Objective status
- Malware on components
- Physical computer theft
- Encyclopedia page selection
- Popup dialogs
    - multiple choice questions
    - yes/no and information dialogs
- Tickers, start, withdraw (must include naming)
- Fires, start/stop

Appendix A: Implementing the U.S. Navy "Virtual Worlds" as the UI Component

The U.S. Navy is developing training and education content packaged within a "Virtual Worlds" environment implemented using the OpenSimulation multi-user 3D application server, which provides an environment similar to Second Life™. The joint project with NAVSEA included investigation and prototyping the use of OpenSimulator as the UI component of CyberCIEGE.

The preliminary work performed to separate the NSS from the UI component was reused in this effort, along with some of the design and software from the UI component that had been developed for Unity3D. OpenSimulator is quite different from the Unity3D game engine, both in structure and intended use. However, OpenSimulator does include support for application extensions written in C#, allowing us to repurpose some of the UI scaffolding, though not the UI elements or interfaces.

OpenSimulator "Region Modules" were used to implement the UI scaffolding and to define the objects that would populate the virtual world, and the objects that are consolidated into interactive UI components, (Heads Up Displays, or "HUDs" in the parlance of OpenSimulator). From the game development perspective, a substantial difference between Unity3D and OpenSimulator is the latter provides no user interface widgets, buttons or whatever. These all must be created out of textured 3D objects which are then collapsed to 2D when made part of a HUD. User input handling is largely limited to detection of a mouse click, (with no distinction between buttons). For example, to create a toggle, the developer must create the visual representation of the two toggle states, and implement the entire toggle function based only on a name of an object that has been clicked.

We developed crude appearing, but fully functional, buttons, toggles and toggle groups for use in this project. These were organized into crude HUDs and were able to provide a UI that allows a user to change configuration settings on a CyberCIEGE computer. The proof of concept also includes a UI via which the user can select different CyberCIEGE campaigns, and different scenarios within those campaigns.

A typical OpenSimulator or Second-life implementation includes a set of functionally rich scripts associated within in-game objects. While that scripting strategy serves the primary intended us of those platforms, the use of object-specific scripts to manage interactions with the network security simulation would require significant complexity. This led us to implement most of UI component functions within Region Modules, which run within the core OpenSimulator engine. The object-specific scripting is limited to click detection and invocation of the associated Region Module function.

Conceptually, the work required to implement a UI component within OpenSimulator is similar to that required to implement it within Unity3D. However, the actual amount of effort required for OpenSimulator would be substantially higher due to is lack of integrated user interface functions and its limited documentation resources. The following summarizes work required to complete a port to OpenSimulator.

- Importing models into the OpenSimulator environment would require converting the existing Maya 3D models to Blender, which appears to be a format understood by OpenSimulator. It remains unclear (and unlikely) whether the animated character models can be imported into OpenSimulator. Game characters would likely have to be implemented as “Non Player Characters (NPCs). These have limited animations but should be sufficient to represent in-game users.
- Create graphically presentable toggle, button, pulldown and text-presentation objects. OpenSimulator text presentation is limited to fixed-sized text that hovers just above an associated object. Any control over font size, placement or layout of text requires that the text be rendered into a texture.
- Create camera control logic that corresponds to game requirements.
- Design and implement a means for the user to navigate the virtual world and select/configure component based solely on mouse clicks. This would likely be based on a series of pull-down menus (the foundations of which would have to be implemented from scratch). The OpenSimulator/Second-life menu system is not usable for other than the simplest interactions.
- Tickers, help-tips, how to invoke and view the encyclopedia.