Security Education: The Next Generation

A Senior Project By

Dirk Cummings
Orion Miller

Advisor

Dr. Philip Nico
Cal Poly San Luis Obispo

December 5, 2011

Abstract

In the 2010-2011 academic year, the student run computer security club White Hat, with assistance from the Cal Poly Computer Science Department and funding support from Raytheon, opened a new security lab. Recognizing the need for an increased focus in teaching security, the new lab provides a place for students to work on security related projects in a safe environment. To help christen the new lab, we set out to host the first hacker Capture the Flag competitions at Cal Poly. What resulted was two competitions; one occurring in May 2011, and the other in November. The first modeled more typical hacker challenges while our second scenario consisted of custom services with a Star Trek theme. This paper discusses some of different types of CTFs, describes the design of each of our CTF scenarios and the problems incurred, suggests improvements to game design for the next competition, an analysis or our implementations' effectiveness to meet the desired objectives, and a comparison against other capture the flag competitions.

Contents

1 A *Hacker* Capture the Flag Competition

The idea of a *Hacker* Capture the Flag (CTF) competition plays off of the traditional child's game. A hacker CTF has a number of different versions with the most typical employing a number of servers to "challenge participants to attack and defend computing resources while solving complex technical problems" [6]. In a university setting a CTF provides a great opportunity "to teach defensive and offensive IT-security in a somewhat more realistic environment than the traditional classroom" [8]. Cal Poly's "Learn By Doing" motto provides a good platform for us to apply and expand our security knowledge without mounds of textbooks.

DEF CON The more well known competition is the annual DEF CON CTF in Las Vegas [4]. The format, theme, and scenario has changed over the years as the organizers change hands every few years.

Ghetto Hackers Is the group who ran the DEF CON CTF for DEF CON 10, 11, and 12 [5]. They took an interesting approach to score keeping especially in relation with their network topology. To prevent teams being able to directly connect the score keeping server by putting the score keeper behind the router which had bi-directional NAT forwarding so there was no way to distinguish an IP address from another team or the score keeper [5].

Hack Fortress This is a Hacker CTF that has been held at Schmoo Con [1]. It fuses Team Fortress 2 (i.e. a team based first person shooting video game) and a Hacker CTF similar the Hacking Olympics ([3], [9]). The game play is designed so those playing Team Fortress 2 have an impact on the progress of those playing the CTF and vice versa.

Educational CTFs There are a few hacking CTFs designed specifically for the course laboratory setting. Georgia Tech's graduate course ECS 6612, Computer Network Security, created the NetSecLab: a six week team project taking students with varying levels of Linux experience from the installation and configuration of an operating system, to securing and attacking common services [2]. Each team's server (also called a "box") typically runs six or seven standard services (e.g ssh, Apache, telnet, and MySQL) [2]. In addition to attacking other team's boxes, there are a few "victim boxes" running different versions of Redhat or Windows XP SP1 with seeded vulnerabilities for teams to search for and exploit [2]. Teams score points gaining access to other boxes, retrieving hash files, and mapping the network and services [2]. Points are lost when their box is compromised [2]. The NetSecLab is one of the more typical hacker CTFs modified to work in a classroom setting with students having little hacking experience. Competitions can become quite advanced as the experience level increases as seen at UC Santa Barbara International CTF.

Lexi Pimenidis describes how he's setup some CTFs in the past [8] and provides a good outline for those starting their first competitions. Much like the creators of the NetSecLab, Pimenidis notes the rules should be such to maximize learning and limit participants from destroying the game when designing competitions for inexperienced players or in educational settings. He also presents a "cookbook for custom services" which offers some design advice and pitfalls to avoid that we will be able to apply in one of our scenarios [8].

International CTF Starting nearly a decade ago, UC Santa Barbara's annual International Capture the Flag competition has expanded from a few dozen universities participating to nearly

70 groups competing for cash prizes [11]. Scenarios have ranged from hacking into an open source version of Windows [10], to fighting the rogue cybercrime supporting nation of Litya [11]. It's one of the more advanced and creative CTFs we've been able to find and is our long term goal.

Typed CTFs Conti and Babbitt [6] detail a number of other types of CTFs ranging from wireless to cryptanalysis, and hardware hacking to social engineering.

Hacking Olympics Lastly, there are even hacking competitions ([3], [9]) which don't require servers for participants to attack and defend. Instead they offer a web based CTF with a number of independent puzzle challenges to test a team's hacking knowledge and abilities.

2 Objectives

The computer science curriculum at Cal Poly has only recently increased the number of security course offerings, yet there is still a lack of security experience in the student body. This presents us with a difficulty when designing the competitions.

Competitions like the iCTF and DEF CON CTF can design their competitions to be as challenging as the designers can make it, we aren't allowed this "luxury". Other competitions will always have experienced people seeking them out and have a much larger pool of possible participants to pull from.

For now our focus is on designing competitions mainly for Cal Poly students. This means we must find a difficult balance between how hard and how easy we can make our competitions. They can't be so hard as to discourage people from participating, and yet can't be too easy such that there's no challenge for players. Competitions need to be playable, enjoyable, and challenging enough to expand a player's security knowledge.

To improve our conditions, we wanted our CTFs to

- See if there were other students interested in hacker capture the flags
- Generate an interest on campus for security and hacker capture the flag competitions
- Provide a fun way of hacking
- Introduce security concepts and skills to our peers
- Create CTFs that allow a wide range of skill levels to participate

Additional benefits of the Hacker CTFs include making use of the new Security Lab on campus and help develop a hacker and security culture with White Hat (i.e. the student run hacking and cyber-security club at Cal Poly).

3 The Duke Nukem Scenario

3.1 Objectives

Since this was our first CTF and because the lack of cyber security on campus, we felt that putting an emphasis on a wide array of challenges would increase participation and allow newcomers to have a good chance of doing well on game day.

Security 101 To help our peers prepare for the CTF we teamed up with White Hat to teach them some basic security concepts. We did this by providing a security workshop each Saturday that covered a topic that would be a challenge in the CTF.

3.2 Design

3.2.1 Game Play

Players were given a website of a pseudo company that created was releasing the up coming 'Duke Nukem Forever' game. Their goal was to complete and progress through a variety of hacking tasks and eventually acquire a copy of the new Duke Nukem game.

To help prevent players from getting stuck we made each task have hints to multiple up coming tasks. This was to prevent a linear progression of tasks. It was to counter the lack of knowledge or technical ability a task from preventing players winning the competition. The flags for each task were consistent (alpha numeric key X number of chars long) and usually out of place to notify players they have accomplished the task.

3.2.2 Tasks

The planned vulnerabilities and seeded flags for each of the servers is listed and described below (note not all of the vulnerabilities were implemented by the start of the game):

1. Web Server
    - **Octobrain** - The first vulnerability to find. A PHP web page used a *system()* function to execute user input on the system through the *cowsay* command. With no input sanitation, users would be able to execute other command on the system. A simple listing of the current directory would list the file "FLAG.txt" which contained the first flag.
    - **Octobrain PHP** - Using the same vulnerablity as above, another file named "FLAG.php" also contained another flag. However, users needed to know how PHP is interpreted by the browser and note they couldn't get the flag using *cat*.
    - **Bad Employee Email Password** - The web server also had a link to the employee's web-mail login (a feature to allow employees to get their email while not in the office). Reading through a series of bio pages for each of the Duke Nukem developer team members, one could find enough information to guess a particular employee's email password.
    - **Plain Text Reset Passwords** - A new intern was hired to work in the Human Resource Department. She's a little ditzy and forgot her password and needed it reset. The new password was emailed to her in plain text. These credentials allowed access to the Human Resources server and databases.
    - **Steganography** - Three images available in plain sight on the web page contained hidden flags. One of the images had slight distortions and a simple search for "FLAG" in the raw data of the file would reveal a flag id. A flag hidden in another image was broken up and needed to be pieced together. The last image contained a hidden RAR compressed file which needed to be extracted in order to discover the flag.

2. Human Resources

    - **John Flagstaff** - Poorly designed database security has left all personal employee information in plan text in the database. To gain access to the database, players would have to find a reset password emailed to an employee in plain text that was never changed. The flag was listed under the employee John Flagstaff's social security number.
    - **Steganography** - Another flag hidden in a mass of shared Human Resource files.

3. FTP Server - Represents a FTP server for the Developer team to store their current iterations of the game.

    - **Steganography** - A flag was hidden in one of the saved beta versions of the Duke Nukem game.
    - **Encrypted Game Image** - A flag was hidden in an encrypted beta version of the game.

4. Dummy Box - Contains no useful company information or documents, but offers a space for a few extra hacking challenges

    - **High Port Telnet** - A quick port scan revealed a telnet server running which when connected to would reveal a flag.
    - **Packet Sniffing** - Periodic packets being sent out could be picked up to reveal a flag.

3.2.3 Score Keeping

Game Master The Game Master (GM) was written in Python and managed the scoring for the teams. It had a web front end that allowed teams to submit a flag to gain points for a completed task. The front end connected to a MySQL database to validate the flag and update the team's score accordingly. The database tracked the each team's score, every flag submission for each team, current flags for every task, and the value of each task. The back-end server of the Game Master would inform a Local Points Daemon, also written in Python, that a task was completed by a team so the Local Points Daemon could generate a new flag, update the task to contain the new flag, and then report back the new flag. Once the GM received a new flag from the Local Points Daemon it would update it in the MySQL database.

Local Points Daemon A Local Points Daemon (LPD) would run on each VM and would listen for the GM to tell the LPD that it needed to update the flag for a specific task. It would then call our flag generator program to get a new flag, update the flag for the specific task and then let the GM know the new flag for the specified task.

Flag Generation A program to create an unique alphanumeric key which would be used as a flag for a task.

MySQL Database This contained all the flag and team related data. It stored the score for each team and all the tasks and their submitted flags for that task. It also contained the current flag for each task.

3.2.4 Network

Our design originally called for each team to be on their own VLAN to prevent teams from attacking each other (and possibly their personal computers). A network diagram is show in Figure 1. The pseudo company's web server would be on all assigned VLANs so all teams have access.

Each server within the pseudo company would be on two networks (see Figure 2). One to act as a company Intranet, and the other for use by the Game Master. Access to the Game Master network was restricted and participants told not to interfere with the network (whenever they happened upon it). Both internal game networks were a simple shared virtual bridge between all virtual machine instances.

To prevent hazards on our network from leaking out onto the external Cal Poly network, the physical connection between the two was removed. Thus, anyone participating and connected to our network would not have Internet access. They were advised to either download and bring any tools they might need before hand, or safely use the campus WiFi network.

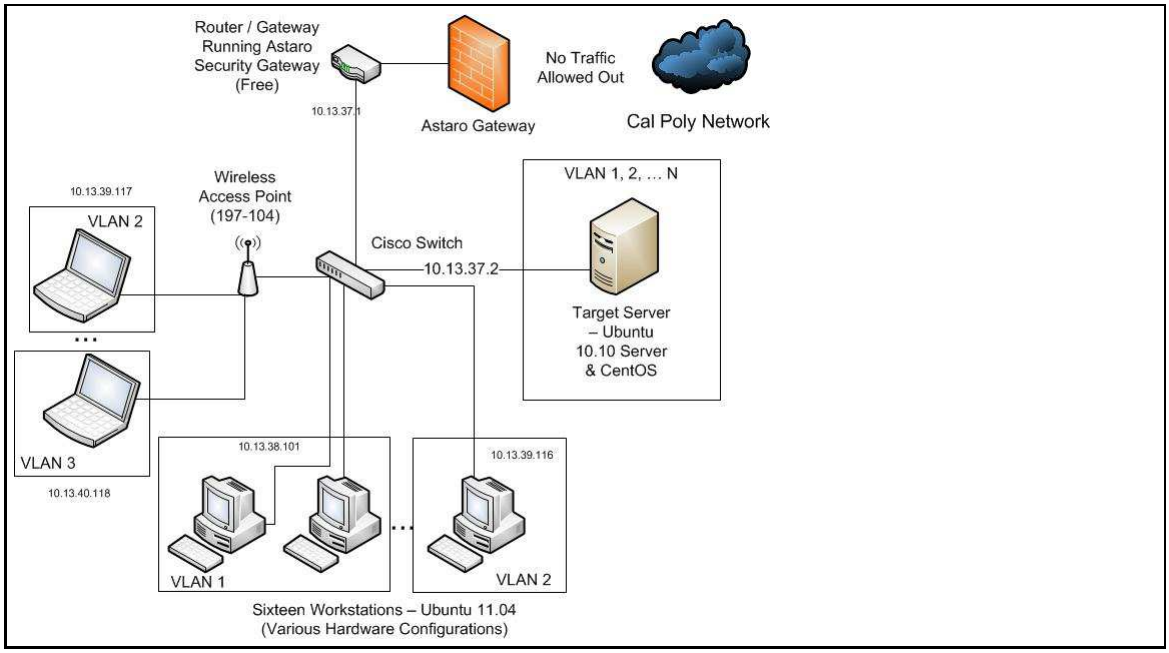

Figure 1: A planned network diagram for the Duke Nukem CTF Scenario. Note: as described in the problems and solutions sections, the planned network setup did not end up including any of the described VLANs and the network connection out of the lab was physically removed.

3.3 Game Day Results

Competition began on May 22nd, 2011 at 10:00 a.m. with approximately 20 people participating throughout the day. About a dozen people stayed for 10 hours until 7 p.m. when we called the game.

Many of the participants had been regular attendees of our Security 101 sessions and were able to find many of the flags directly related to the subjects we covered. Groups were originally limited

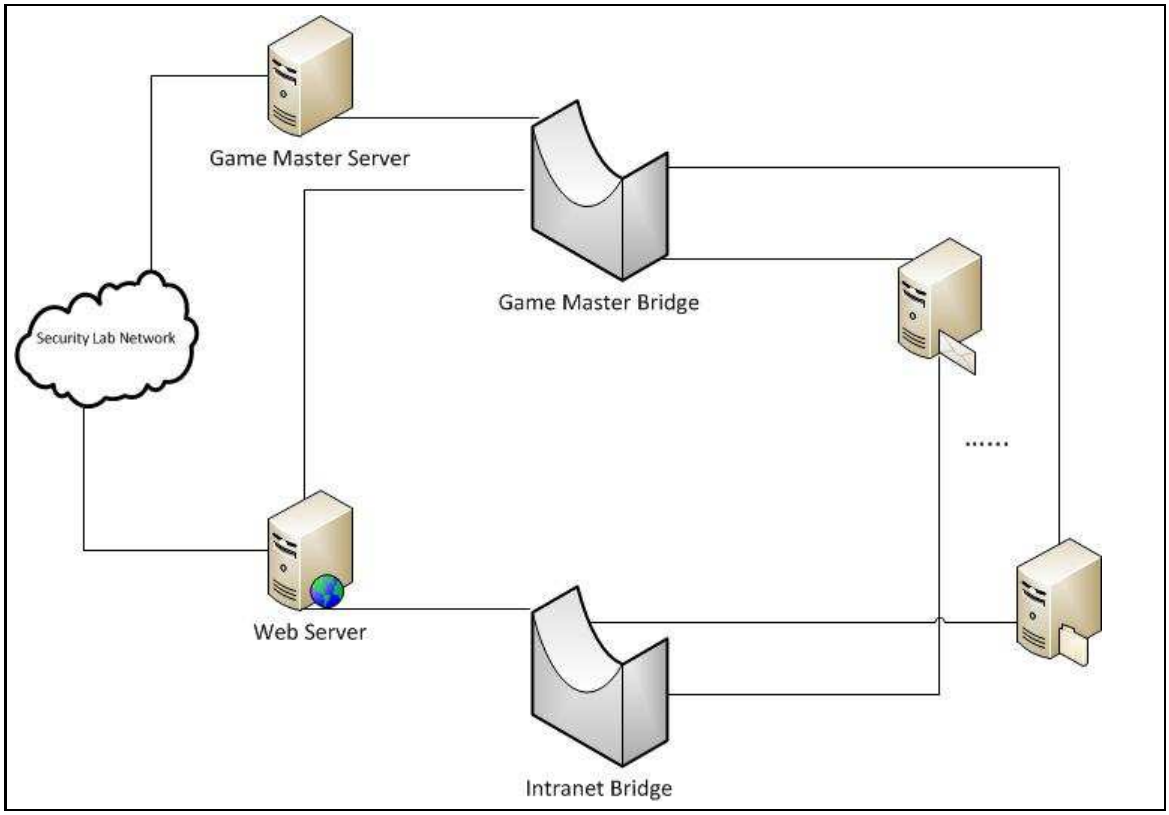

Figure 2: A network diagram for the pseudo company's Intranet and Game Master network.

to at most three people, but as the easier flags were all discovered participants began complaining about not finding anything else. A few hinted that outside of the Security 101 sessions they were very inexperienced and would like to join with the other groups finding some of the harder flags. After which, we allowed the teams to grow however large they liked.

There were two teams with experienced players and in each team someone with a BackTrack Live CD. Since our servers were unpatched Ubuntu 10.10 installations, they were able to use the automated tools supplied in BackTrack to exploit known issues with Apache, MySQL, and PHP. These were not the intended attack vectors. Even though they were able to get a root shell, they weren't able to do much more. Eventually they realized they wouldn't be able to score any points for their team and backed off.

The competition was fraught with a number of issues at the beginning. We had not restricted user access to some potentially destructive commands or sensitive folders. For instance, an exploit which allowed participants to gain shell access also allowed them to delete every file in the web directory or emptying files by rewriting to them. In another instance, users were allowed to create massive files in the *tmp* directory. The virtual machine was limited to 15 Gigabytes of hard-drive space and eventually the *tmp* folder grew so large it prevented anyone else from downloading script they'd written to help them find flags. Eventually, the entire virtual machine was unusable and needed to be re-imaged.

By the end of the competition, only one team realized there were other servers in the network (even though they were informed at the start of the competition) and they were able to capture a few additional flags. When one of the more experienced teams was informed they missed the other servers, they reported a tool they used to find hidden content in images was reporting the strong likelihood of a message on one of the web pages. They had spent considerable time and resources attempting to extract the message and capture the flag. Upon our further investigation, it was discovered the image they were working on was not one of our seeded plants. Instead it was one of the stock images we used from the original website we copied.

3.4 Problems and Solutions

rm * -rf As described in the **Game Day Results**, a few individuals were able to shut down the game by attacking the server instead of finding the flags.

We were able to prevent the removal of files in the web directory by resetting access to the files and directories. We still wanted them to be able to create files to download their scripts, so we left one temporary folder open for them to use.

Free Virtual Machine Servers The server we were using to manage our virtual machines was free and thus limited in the scale of available features. Originally our plans called for a network setup such that the company's Web server was on all VLANs in the network (see Figure 2). The limited feature set of our Virtual Machine server did not include the ability to setup network cards for use with VLANs.

After two weeks of research and troubleshooting, we were not able to find a suitable solution and instead did away with all of the VLAN requirements. Additionally, we asked all participants to not attack each other's machines and backup their data or use a Live CD if they were worried about breaking their machines.

BackTrack One of our objectives was to test competitor's hacking knowledge and abilities and not their ability to use pre-built scripts and tools. Finding exploits in the services was not within the scope of this competition.

To prevent the use of tools which do all of the "hacking" for you, we decided to change the design of the next CTF such that they wouldn't be able to use something like BackTrack by building our own services no one had ever worked with before.

4 The Star Trek Scenario

4.1 Objectives

After the first CTF competition, we wanted to make our next focus to be on solving two problems:

- testing the individual's coding and hacking skills rather than their ability to find and use pre-built tools,
- and prevent the participants from breaking the servers (rather than the services) and ruining the game.

These would not only improve the overall game play and enjoyment, but would help eliminate game down time spent repairing and restoring virtual machine images. Additionally, by designing new services from the ground up we can give participants experience with the common cause of vulnerabilities: poor design.

Our first scenario was good for getting the uninitiated started in security, however, it did lack defensive objectives. While the first scenario focused mainly on attacking servers and their services, we needed to even the importance of defending to better align the game with a real world scenario.

Lastly, players of the first competition noted in their feedback they wanted more teamwork based cooperation. Specifically, those with little to no security experience had a tough time working out the solutions and would have liked to work with others outside of their team to strategize and learn techniques. The standard method of team formation usually results in those who know each other and have similar expertise tightly grouping together. This tends to vary experience by groups rather than the individuals in the group and creates a hording of knowledge in which the less experienced gain little from participating. For this scenario, we wanted to move away from this particular method.

4.2 Design

To ensure no participant would be able to use pre-built tools to exploit vulnerabilities, we chose to design services one might program during the first implementations of a ship from Star Trek. This design was chosen for two reasons: in the summer of 2011 Netflix began streaming every episode of every Star Trek series [7], and we couldn't find anyone who had created such services or written anything to exploit such services. A listing of these services and their designs can be found in **Game Play** and **Services** sections 4.2.1 and 4.2.3 respectively. Our new approach, described in the following sections, also allowed us to make changes to the game play to focus more on team based play.

4.2.1 Game Play

The new game design was made to resemble a WarGame style of play consisting of only two teams each with their own server to manage. We were even fortunate enough to provide both teams with their own rooms to work in secrecy. Both teams initially received identical copies of each service. Their goal was to find and patch vulnerabilities in their services then turn around and exploit the same vulnerabilities in the other team. If a team could keep their services responding and pass basic functionality tests they would score points. However, if their services become unresponsive or failed the tests, whether their own fault or the opposing team, they would stop receiving points while the opposing team would score additional points.

We also wanted to give those participants with little or no hacking (or even coding) experience something entertaining and contributory to do. Since these services should be fully functional and collectively represent a star ship, then it stands those who feel they can't "hack" could fly the ship around and score points for their team by shooting the other team's ship.

Team formation also changed. Now, no one knew which team they were playing on until they showed up to the event. Each new person walking through the door was automatically assigned to the opposing team assigned to the person before them.

4.2.2 Score Keeping

Since the game play was more complex than just doing tasks and getting points for capturing a flag, we needed some form of live understanding of how the game was being played and to create a scoring metric which could determine a winner and score. We also needed to account for them having admin access to their server and a way to keep them honest as they would be re-writing their source code and compiling their service.

We needed the score keeping to be more difficult to break than to play the game within the time frame of the competition. To do this we needed to make the score keeper as invisible as possible for the server and the score keeping partially handled with the base functionality given to the players to compile their service. Lastly we needed to prevent teams spoof score keeping traffic.

Scoring Schemes Our metric for scores:

- **Up-Time** - The main method for scoring points and represents the hacking side of the game. Scores are based on the accumulated time each service was responding to requests and passing the base-functionality tests.
- **Communications** - Secondary encryption challenges.
- **Damage Dealt** - Teams may gain points not by hacking but by playing the game and firing weapons as the opposing team's ship.

Both the communication and damage metrics exist to add other challenges to broaden the interest for the game.

Internal Score Keeping - Base Functionality Since the players would be compiling the service and had root on the server, we needed a way to gaurentee their service was running and keeping some form of necessary base functionality. We did this by creating base functionality requirements for each service that would report to the Score Keeping server the current state of the service. The results of the base functionality tests were reported to the Score Keeping server. If the service was down, then it would not be able to notify the server at all of the state of the service, so the Score Keeper would default the service as failing its tests. Tests were ran every 60 seconds.

Score Keeping Server Written in Python, the sole purpose of the server was to manage and update the scores for each team. It would listen for the state of each service and noting whether it was passing or failing its functionality tests. If a service did not report whether it was passing or failing its test cases it would default to the service being down and therefore not passing its test cases. It would also listen for damage from the Damage Manager and update scores accordingly. It managed and stored all information about the game by connecting to the Redis Database. Scores were updated every 60 seconds.

Damage Manager The Damage Manager was added after our design phase. It was on the Score Keeping Virtual Machine and provided the amount of damage a ship was dealt within a period of time. If a team shot its opponent's ship service, it would check to see if their opponent's ship was within range and if the shield for that service was down. If both of those cases were met it would record it as a hit. Damages were reported to the Score Keeping server every 60 seconds.

Redis Database This housed all the information related to the game. It contained the current scores, position, damage, and general information for each team.

The MySQL database was dropped from the last competition because the lack of documentation with PyMySQL to get the new code to work with the new Score Keeping server.

Secure Traffic and Score Verification Since the base functionality would be updating the state of the service to the Score Keeping server we needed to make sure that players could not spoof these validations. What we did to protect against this was wrapping the connection between the base functionality for a service and the Score Keeping server in an SSL connection.

This protected the information going across the network from being sniffed but it didn't prevent a team from connecting to the Score Keeping server and sending it bogus data. To prevent bogus data we required each service to send a pass phrase, which was specific for each team, to the score keeper to verify it was our code that was sending the status of a service.

Lastly to prevent a team from reverse engineering and just finding the pass phrase in memory we wrote a series of macros to form a 2 way hash so we could have the pass phrase obfuscated. The pass phrase was stored hashed in the provided object file and un-hashed just before the sending of any data.

4.2.3 Services

On competition day, there were three implemented services in production for the teams to fix with three additional services planned for release later in the competition. (Each of the planned services and a brief description of their functionality is listed at the end of this section).

Services are designed with two major parts: the built-in (or base) functionality, and the competitor's implementation.

The base functionality was implemented by us and provided in unlinked pre-compiled form for each team to build on top of. It is meant to represent the initial implementation of a ship's services as made by first year ship programmers. There are a few seeded vulnerabilities and even more hidden in poor and hastily made designs. Base implementation of a service is only responsible for:

- providing only the most basic level of support and implementation necessary for the service to run,
- setting up the network support required for running the service,
- listening on an assigned port, accept incoming connections, and pass on the connection to a handler which processes the incoming data,
- running base functionality tests on the competitor's implementation and pass on the results to the Score Keeper for scoring,
- and keeping a modest amount of internal book keeping specific to the service to ensure the honesty of a competitor's implementation.

While the exact functions for each service differ, every service's base implementation has functions (defined as externals in header files) for the competitor's code to call before finishing processing an incoming request. For each of a service's base function, there is a corresponding competitor's wrapper function to be called before the equivalent base function. The location of the wrapper

functions are stored in a structure as function pointers defined in a shared header file between both implementations. These wrapper functions allow the competitors the chance to perform their own book keeping and sanitize inputs from requests before calling the base functions. A diagram of the typical call stack and a snippet of the header files are available in Figure 3. This was one way for players to patch vulnerabilities and design flaws.

To communicate between and control other services, each service had a pre-defined packet structure mimicking an IP packet. Both teams started off with the same structures but were allowed to extend the packets to fulfill any future needs. The initial structure contained only enough fields to control each specific command of a service. Any security or error checking features were left up to the participants to develop and deploy.

Power The critical service of each ship responsible for managing and distributing a fixed amount of available ship power to all other running services. A service may only run while it has been allocated power. Should a service lose power, it shall be stopped and remain non-responsive until power is restored. If the power service itself is shutdown or crashed, all other services would be taken offline.

Engines Just like the engines in the star ships with support for both impulse and warp speeds. The engine service is only concerned with managing either engine's current power allocation and speed as well as ensuring their health. Each engine leaks varying amounts of radiation which if not dissipated could damage the engines to the point they are no longer functional. Scotty can fix engine damage, but he's not a bloody miracle worker; he can only work so fast!

Navigation Responsible for setting a ship's course, updating the ship's internal representation of the map, and managing the engines.

Communications Independent of all other services except for the power service, and instead of managing some aspect of a ship, presents each team with a number of cryptography puzzles to be decrypted, solved, and posted back to the Score Keeper for validation.

Weapons Shoots at a "service". If the shooter is within range and there are no shields protecting the targeted service the shooter receives points for damaging their opponent.

Shields Temporarily protects a service from opposing weapons fire.

4.3 Game Day Results

The competition started around noon on November 5th, 2011 with four participants and three of the services in production. Those services were the power, engines, and navigation. The others had not yet been implemented, but were planned for release some time at the end of the first day or during the start of the second day of competition.

Over the next three or four hours both teams were still reading through the documentation and asking questions trying to figure out the structure of the game and the code. At least we thought they were reading the documentation. Both teams would read a bit of the documentation and then start asking a dozen or so questions each. Almost all of which could be answered if they continued

```
extern struct PowerFuncs {

   void *(*request_handler) (void *confd);

      The following functions are your wrappers around the built-in / main
        Power service functions. They will be used for testing your service
          implementation and will not be called until your write your own
          implementation. At the end of each of your implementations, you must
          call the external equivalent functions. A diagram of the setup is
          bellow along with an example of the order of the function calls.

      Diagram:

      ------------------------------------------------------------
      |  request_handler                                         |
      |                                                          |
      |     ---------------------------------------------------|
      |     |  additional_function                               |
      |     |     (if you want to add additional                 |
      |     |     checking or functionality to add security)     |
      |     |                                                    |
      |     |     -----------------------------------------------|
      |     |     |  wrapper_function                            |
      |     |     |     (for your book-keeping)                  |
      |     |     |     EX) pow_funcs.wadd_power(int, int)       |
      |     |     |     -----------------------------------------|
      |     |     |     |  built-in_function                     |
      |     |     |     |     EX) add_power(int, int)            |
      |     |     |     |                                        |
      ------------------------------------------------------------

      Function Call Example:

      --> Incoming Connection Request
            |--> Connection Accepted
            |--> pow_funcs.request_handler(connection_file_descriptor)
               |--> additional_function(...)
                  |--> pow_funcs.wadd_power(int, int)
                     |--> add_power(int, int)
                  |--> send_packet(struct PowerHeader, socket_fd)

   int (*wregister_service) (int, int);
   int (*wunregister_service) (int);
   int (*wadd_power) (int, int);
} pow_funcs;
```

Figure 3: A snippet of the provided header file for the Power service. It describes the structure and use of the wrapper functions as well as demonstrating the data flow of incoming requests.

reading. It became quite irritating quite quickly especially as the questions started to move towards exploiting the game rather than playing it. Very little code was actually being written, but design flaws and vulnerabilities were slowly being realized.

One or two members of Team A had previous experience in UNIX System Administration and immediately began setting up monitoring tools to track incoming requests and log the commands executed by each service. After which they moved their focus to the operating system running their services and attempted to "secure" it instead of their services. A few hours later and they abandoned this approach. Eventually they settled on simply not processing any incoming requests in their engine and navigation services. They effectively produced a reverse denial of service.

Team B on the other hand, situated in the neighboring conference room decided to start by developing tools to help them manage the services. (The initial implementation provided no user friendly means to control a service. A member would have to format a packet, open a connection to the service, send the byte stream through the connection, and process any returned information.) Through this management tool the team was able to track command requests between services and drop unexpected incoming requests. Given the sequence of events for each service's commands, the team could select the authentic commands and dump any malicious ones. This was a great start for the team as they were able to reap quick results.

Up until this point Team A had been setting up connections and sending "random" bytes of data to Team B's services. Since the original implementation was faulty, these connections and incorrect data often crashed Team B's services, causing them to fail tests and costing them points. When the new filter went into place, these malicious attacks were almost completely eliminated.

Later on that night Team B started on their design for a basic level of encryption.

4.4 Problems and Solutions

Lack of Experience Among Players No one here really knows how to exploit code, and those that do didn't participate.

For the first event we offered Security 101 sessions every Saturday leading up to the competition to help prepare players and get them inspired. Because of our lack of availability we weren't able to get a similar series setup. We don't want to suggest correlation implies causation, but there was an obviously lower attendance for the second competition.

Stagnation While one team outright dumped all incoming requests, the other did essentially the same thing by hardening access to their services such that anyone not remotely logged-in into their server could not gain control. Ultimately a good achievement, however, game play began to stagnate.

An easy fix would be to require all services to have an open public interface. Instead of giving teams control of the server and the ability to lock others out, anyone should be able to control a Team's services. This does bring up the possibility of each team's ship experiencing erratic behavior (such as rapid navigation changes and little to no movement across the game's map) but this can be redefined as desired behavior. Such behavior can serve as motivation for a team to work on ways to prevent the attacks or increase the attacks on another team to bring their ship down first.

Short On Time One quarter to design, implement, and test completely custom services and wrap them up into a game is not enough time for only two people. You say you're going to work on it over the break, but good luck keeping priorities.

At least two more people and one more quarter would help prevent slipping and bad code (see next problem).

Bad Tests We started performing our implementation tests one and a half weeks prior to the competition. While were able to catch and solve a few big problems before the last minute, we weren't able to perform a "dry-run" of game and iron out some other major issues between the Score Keeper and the services. A fare and functional Score Keeper is key to a successful CTF [5]. Without the dry-run, we weren't able to find logic and design flaws in our service tests and reporting which severely affected game play.

Ideally, all code (services, Score Keeper, helpers, etc.) should begin undergoing final implementation tests and debugging four weeks out from the competition and integration tests done two full weeks out culminating in a complete dry-run of the game. All services and Score Keeping should be fully implemented and tested before integration testing begins. Additionally, at least one virtual machine image should be created and saved for each of the following (and will not be used for the competition):

- a clean (without any game code) and fully configured server
- for each team, a working (with all game code included) instance which also contains and passes all implementation and integration tests
- for each team, the final working copy distributed at the start of competition

Code Management Each team's code had slightly different configurations which had to be changed by hand within a single code base. With 10 to 12 changes to make in each service's code, insuring all changes were properly made before each update was difficult under pressure. The design of our Score Keeper (see Section 4.2.2) required all service specific configurations to be contained within the pre-compiled source code provided to each time. On a few occasions configuration settings for one team made it into the compiled version of the other team. Resulting in services not starting up, not communicating properly with other services, or communicating incorrect results to the score keeper.

Ideally the services would have been written such that no updates were ever needed to be pushed to servers during game play, however, other problems inhibited this solution.

Because of the way we decided to authenticate service status updates with the proper team (as described in Section 4.2.2), we would not be able to keep all configurations in a common header file. Instead we should have either created two code bases (one for each team) with configurations which don't change, or use a simple script to write all unique configurations, compile the service, and push the update to the team's server. The former would introduce the problem of ensuring both code bases were synchronized with the proper updates. A simple solution, but one just as error prone and no less irritating. The latter solution would take little time to implement and cut update hassles down to a simple command line execution.

Open-BSD During the first competition, we had a number of people trying to attack the server's services rather than finding the vulnerabilities we created. Servers in the first competition were unpatched Ubuntu Server 10.10 instances with a number of open network services. Since this go around we didn't need things like PHP, Apache, MySQL, etc., a stripped down operating system

capable of compiling code and running our network processes was ideal. After the initial configuration, users would not have sudo-er privileges or be able to install new packages.

However, during our integration testing, we ran into a number of problems getting the services to start up and behave properly. All code was POSIX compliant but services had difficulties opening and binding sockets as well as sending and receiving data. Limited remaining time (see **Short On Time** and **Bad Tests**), we were not able to solve the problem and instead switched to fully patched Ubuntu Server 11.10 instances with no sudo-er privileges and without Apache, PHP, or MySQL installed.

5 Scenario Analysis

5.1 Duke Nukem Scenario

Challenges Not Met We tried to meet all tasks listed in Section 3.2.2, however we were unable to finish the 'Steganography' and 'Encrypted Game Image' tasks for the FTP Server as well as the 'Packet Sniffing' task for the Dummy Box. The absence of these tasks did not effect game play as no team made it far enough for these tasks to become necessary.

5.2 Duke Nukem Scenario Compared to other CTFs

Our challenge was similar to other traditional CTFs except it did not have a defense aspect to the game play. The Duke Nukem Scenario was most similar to the Hacker Olympics discussed in Section 1. We also strayed away from any challenges requiring knowledge of having to write an exploit since this was outside the scope of knowledge and technical ability for almost all participants.

Players Reaction Roughly half of our attendants were lost on how to complete most of the CTF tasks but those who consistently attended our 'Security 101' sessions appeared to be well equipped to participate in the CTF.

Everyone seemed engaged in the CTF, and curious to find ways to solve the given tasks.

We intended for attendees to play in teams. We provided a team sign up sheet prior and encouraged joining teams ad hoc. However most were reluctant to work in teams.

At the end of the competition everyone was excited for our up coming CTF and appeared to have a serious interest in security.

Future Uses This game format would be perfect for casual play over a long duration for students. By adding more tasks, more features to the Score Keeper, and some form of administrative system, players could start and stop as they wish. Since they could work at their own pace and improve their skills on there own time it could be added as an extended platform of instruction for the 'Security 101' sessions.

5.3 Star Trek Scenario

Services Not Met We were unable to finish all the services planned in section 4. The 'Communications', 'Weapons', and 'Shields' services were not completed by the start of the competition. This impeded some of the video game play aspects of game. However it did not stop the game from proceeding.

5.4 Star Trek Scenario Compared to other CTFs

The Star Trek scenario was similar to classic CTFs (e.g. DEF CON CTF [4]) however the key difference was that our scoring was based on up-times not by teams acquiring and submitting flags.

Players Reaction In this competition even though the players stayed for a longer time, most players appeared to be ill-equipped to participate in this competition. More importantly they were reluctant to try and mess around with the services to find security holes. Instead of being curious and creative it felt like that they were expecting us to tell them exactly what to do. Some complained that they be given the source code to the base functionality rather than working on securing what they were given.

Having a 'Security 101' series leading up to this CTF would have probably proven very useful to all the players.

After the competition people were genuinely interested in continuing this game, having future CTFs, and possibly starting a more official CTF team.

6 The Next Ideal Solution

6.1 Game Play

Stick with the Star Trek Scenario game play (section 4.2.1) style of large teams pitted against each other but create a bigger breadth of tasks and ways of winning similar to the first scenario. In addition create a visual front end to the game for the video game aspect. By doing this you could create diversity in the game to help encourage newcomers and retain them for longer.

Visualization Adding visualization to services (e.g. navigation, weapons, and shields) would merge the game between a hacking competition and video game similar to Hack Fortress [1]. We feel by adding this to the Star Trek Scenario (section 4) would not only help encourage new players to get involved but help retain newcomers by having an easier way to socially interact with those who have more CTF experience.

6.2 Network

6.2.1 VPN

Create a VPN for the Sec Lab specifically for this event so people can participate any where in the world. Making the compitition a multi school event similar to the iCTF. This would also allow students to practice when the security lab is closed. This also could be used for the design of long term (e.g. quarter long) CTFs.

6.3 Services

Adding Depth The custom services provided to players needs to have added functionality to add depth to the service and making the service not as straight forward to secure.

Denial of Service One of the vital issues with the services of Star Trek Scenario (section 4) is the score keeping and over arching design was not able to prevent a teams denial of service from other teams. This could be fixed by modifying the score keeper to be integrated with a router [5], or wrapping all send and receive socket calls which would be required to be used by the teams so that the data is sent to Score Keeper to make sure packets are being accepted and responded to accordingly for all teams.

Service Creation Libraries Create libraries to make it easier to create new services. Possibly make a generic base functionality. This way it can be easier to create new services and so more time can be spent on adding depth to the functionality of the service. This has been started but not completed.

Game Time Compile and Submit Add a way for players to automatically compile their code and the snapshot of that code to the a repository. This would allow for a post game analysis of code. Possibly using code Covarity to see how robust the code is and compare it to scores that team received.

6.4 Score Keeper

6.4.1 Obfuscation

The score keeper needs to be more surreptitious, internally (i.e. Base Functionality) and externally (i.e. score keeping server).

Internally To better deter players from reverse engineering the base functionality a polymorphic engine could be added. This would not prevent the base functionality from being reverse engineered, but it would act as large deterrent and make it easier for players to focus on playing the game rather than breaking the scoring system.

Externally Make a similar network design as the Ghetto Hackers did [5] where the score keeper sits behind a router with bi-direction NAT forwarding. This would be another project in itself however it would have enormous benefits to the robustness of score keeping as stated in section 1.

Communication Even though the communication between base functionality and the Score Keeper used SSL to transmit data across the network securely, services still had to authenticate themselves by providing a password. It would be an improvement for the password authentication of a service to be more automatic such that each service for each team has a password. Even if a team discovers a password for their service it would help minimize the affect a team could take advantage on scoring.

6.4.2 Scoring Metrics

Keeping the CTF to the same format as the Star Trek Scenario (section 4) it is hard to have a fair and balanced scoring scheme. If you can generate a flag capturing means of qualifying for points it is becomes easier. However, if the game play is similar then there needs to be a better metric than just checking service up times. It would definitely be a factor in the score but shouldn't really be the majority of the score.

7 Other Suggestions And Things to Avoid

For things specific to our and possible future CTFs at Cal Poly, the following is a list of suggestions and things which should be avoided that may not have been covered in a previous section or are important enough to be reiterated:

- **Teaching** The competitions held in the immediate future should be expected to still have a large base of limited experience participants. Thus, the security workshop series should continue leading up to competition to cover topics which may appear in the scenario. It should not only help increase the number of participants, but also increase the level of play, enjoyment, and learning through the competition.
- **Advertise** Get competition details out every week (usually in conjunction with the 101 sessions). Be sure to get into the various department mailing lists (CSC, CPE, etc.). Posters should be up no later than two weeks prior to the competition.
- **No Experience Necessary** We've tried using it while advertising for both competitions, but no matter what, it didn't seem to help get the crowds we expected. There may be varying understandings of the phrase, but something along the same lines should be stressed to those wary to participate.
- **Room Reservations** Reserve Bonderson rooms WELL in advance and confirm the reservation at least a month out. Don't forget to double check who has card swipe access.
- **Timing** Being on a quarter system presents a number of difficulties while selecting a date to host the competition. We found the end of week six might be the ideal time to avoid conflicts with midterms. However, use best fit (least number of conflicts) when picking a date knowing no weekend you chose is ever going to be perfect. Mind the holiday weekends, midterms, sporting events, or other club functions.
- **No all Nighters**. It should go without saying. You're mainly going to be thrashing and you'd rather be up all night during the competition than working on only a few hours of sleep and heading towards 56 hours without sleep.
- **Documentation** You're probably going to be writing a report (maybe even one like this) and documenting before you begin work (ideally) or while you're working will greatly help in the future.

8 Conclusion

There was not enough time between the two of us to complete everything for both competitions. We were not able to write perfect code as suggested in 'Hosting a Hacking Challenge' [8] and some of our base designs fell apart. Creating a more thought out design and robust CTF while meeting our main objectives (section 2) would require a larger development team and having at least two quarters for thorough testing and development. With the current lack of security classes within the Computer Science Department it is highly advised to provide security workshops related to the upcoming CTFs. We were able to meet our over arching objectives by generating interest amongst our peers to participate in CTFs and we may potentially have enough interest to start an official Cal Poly CTF team in the near future.

9 Bibliography

References

[1] (2011) Hack fortress. [Online]. Available: http://www.shmoocon.org/hack_fortress

[2] K. F. J. C. Christopher Lee, Arif Selcuk Uluagac, "The design of netseclab," IEEE Transactions on Education, vol. 54, no. 1, pp. 148–155, February 2011.

[3] Cipher. (2010) Ctf-style hacking challenges. [Online]. Available: http://www.cipher-ctf.org/CaptureTheFlag.php

[4] D. C. Communications. (2011) Def con capture the flag archives. [Online]. Available: http://www.defcon.org/html/links/dc-ctf.html

[5] R. C. Eller, "Capture the flag games: Measuring skill with hacking contests," 2004, audio and PDF also available at http://www.blackhat.com/html/bh-media-archives/bh-archives-2004.html. [Online]. Available: http://www.blackhat.com/presentations/bh-asia-04/bh-jp-04-eller/bh-jp-04-eller.ppt

[6] J. N. Gregory Conti, Thomas Babbit, "Hacking competitions and their untapped potential for security education," IEEE Computer and Reliability Societies, pp. 56–59, June 2011.

[7] A. Pascale, "Netflix begins streaming star trek tos, tng, voyager and enterprise," July 2011. [Online]. Available: http://trekmovie.com/2011/07/01/netflix-begins-streaming-star-trek-tos-tng-voyager-enterprise/

[8] L. Pimenidis, "Hosting a hacking challenge," in 22nd Chaos Communication Congress, None, Ed., December 2005.

[9] smp. (2010) smpctf 2010 hacker olympics. [Online]. Available: http://www.smpctf.com/aboutSMP.php

[10] G. Vigna. (2003) ictf 2003. [Online]. Available: http://ictf.cs.ucsb.edu/archive/iCTF_2003/index.html

[11] ——, "The 2010 international capture the flag competition," IEEE Computer and Reliability Societies, pp. 12–14, January 2011.

[12] ——. (2011) The ucsb ictf. [Online]. Available: http://ictf.cs.ucsb.edu/

Work Distribution

Scenario 1

Game Master	Orion
Local Game Daemons	Dirk & Orion
Octobrain	Orion
Octobrain PHP	Dirk
Bad Employee Password	Dirk
Plain Text Password Reset	Dirk
Web Site Steganography	Orion
John Flagstaff	Dirk
HR Steganography	Dirk
FTP Steganography	Orion
FTP Encrypted Image	Orion
Telnet	Orion
Packet Sniffing	Orion
Network	Dirk

Scenario 2

Score Keeper	Orion
Power	Dirk
Engines	Dirk
Navigation	Dirk
Shields	Orion
Weapons	Orion
Communications	Orion
Damage Manager	Orion