Title

By Dirk Cummings

Orion Miller

Senior Project

Dr. Philip Nico

December 2, 2011

Abstract

Abstract

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 version. 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" [5]. 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" [7]. 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 know competition is the annual DEF CON CTF in Las Vegas [3]. The format, theme, and scenario has changed over the years as the organizers change hands every few years.

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 which takes students with varying levels of Linux experience from basic the installation and configuration of a Linux operating system to securing and attacking common services [1]. Each team's server (also called a "box") typically runs six or seven standard services such as ssh, HTTP, telnet, and MySQL [1]. 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 [1]. Teams score points gaining access to other boxes, retrieving hash files, and mapping the network and services [1]. Points are lost when you box is compromised [1]. The NetSecLab is one of the more typical hacker CTFs modified to work in classroom settings on students with little hacking experience. Competitions can become quite advanced as the experience level increases as seen at UC Santa Barbara International CTF.

Pimenidis [7] describes how he's setup some CTFs in the past 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 [7].

International CTF Starting almost a decade ago, UC Santa Barbara's Annual International Capture the Flag competition has expanded from a few dozen universities participating to nearly 70 and cash prizes [10]. Scenarios have ranged from hacking into an open source version of Windows [9], to fighting the rogue cybercrime supporting nation of Litya [10]. 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 [5] 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 ([2], [8]) 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

3 The Duke Nukem Scenario

3.1 Objectives

3.2 Design

3.2.1 Game Play

3.2.2 Tasks

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

1. Web Server
    - **Octobrain** - The first exploit to find. A PHP web page used the *system()* function to send user input to *cowsay*. 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 exploit 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 webmail 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 would be able to find enough information to guess one 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.
    - **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" would find the flag id. The flag hidden in the other 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 an 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

3.2.4 Network

Our originally design 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.

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 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 to learn their techniques. After which, we allowed the teams to grow to 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 attempting to extract the message and find 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. Recall our originally 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 to the point 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 out 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.

The first scenario was good for getting the uninitiated starting in security, however, it did lack defense 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. Team formation in the second competition needed to move away from the standard method by which those who know each other tightly group together thus separating those who know from those who don't. This method tends to create groups with varying experience and a hording of knowledge.

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 [6], 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. The new approach also allowed us to make changes to the game play to focus on a more 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. Each team initially received identical copies of each service and tasked to fix and deploy on their servers (each a virtual machine running Ubuntu Server 11.10). Their goal to find and patch the vulnerabilities in their code and exploit them 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.

4.2.2 Score Keeping

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 build on top of. It is meant to represent the initial implementation of ship's services as made by freshman 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,
- listing 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 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

Shields

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 to be 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 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 returning from the request handlers of the 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 and the other did essentially the same thing by hardening access to their services no one not ssh'd into their server could 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 [4]. 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 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, an 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

6 The Next Ideal Solution

6.1 Game Play

Stick with the scenario [star trek] of large teams pitting against each other but create a bigger breadth of tasks and ways of winning similar to [scenario 1]. 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.

6.2 Network

6.2.0.1 VPN Create a VPN for the Sec Lab specifically for this event so people can participate any where in the world. Make the event a multi school event. Similar to the [ucsb ctf]. 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

6.3.0.2 Visualization Possibly add some form of graphical front-end end to game. Creating a merge between a hacking competition and video game. This would be used to help encourage new players to get involved with future CTFs or security.

6.3.0.3 Library 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.

6.3.0.4 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 [eric's paper] to see how robust the code is and compare it to scores that team received.

6.4 Score Keeper

6.4.0.5 Obfuscation The score keeper needs to be hidden better. Internally within the files and externally as the score keeping server.

Internally Encrypted packing of the base functionality. Would add extra prevention of people being able to reverse engineer the base functionality.

Communication We had encryption and some form of basic authentication by using a password. Needs to be better.

Externally Make your own virtual router to put your score keeper behind so that it makes it more difficult to directly connect tot he score keeper but also allows you to in a central point of score keeping see everything going on the network.

6.4.0.6 Scoring Metrics If you can generate a flag capturing means of qualifying for points it is a bit easier [scenario one - scoring] however if the game play is similar to [game 2] then there needs to be a better metric than just checking the up times and down times of those machines. They definitely could be a factor in the score but shouldn't really be the majority of the score. create a means of hacking the server having some form of potentially layered affect on their services so you can be more granulated or score and possibly modify points for scoring offensively and defensively

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 in 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 stressed to those warry 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 crashing and you'd rather be up all night during the competition rather than working on only a few hours of sleep and heading towards 56 hours without sleep.
- **Documentation** You're probably going to 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

9 Bibliography

References

[1] 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.

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

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

[4] 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

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

[6] 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/

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

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

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

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

[11] ——. (2011) The ucsb ictf. [Online]. Available: http://ictf.cs.ucsb.edu/

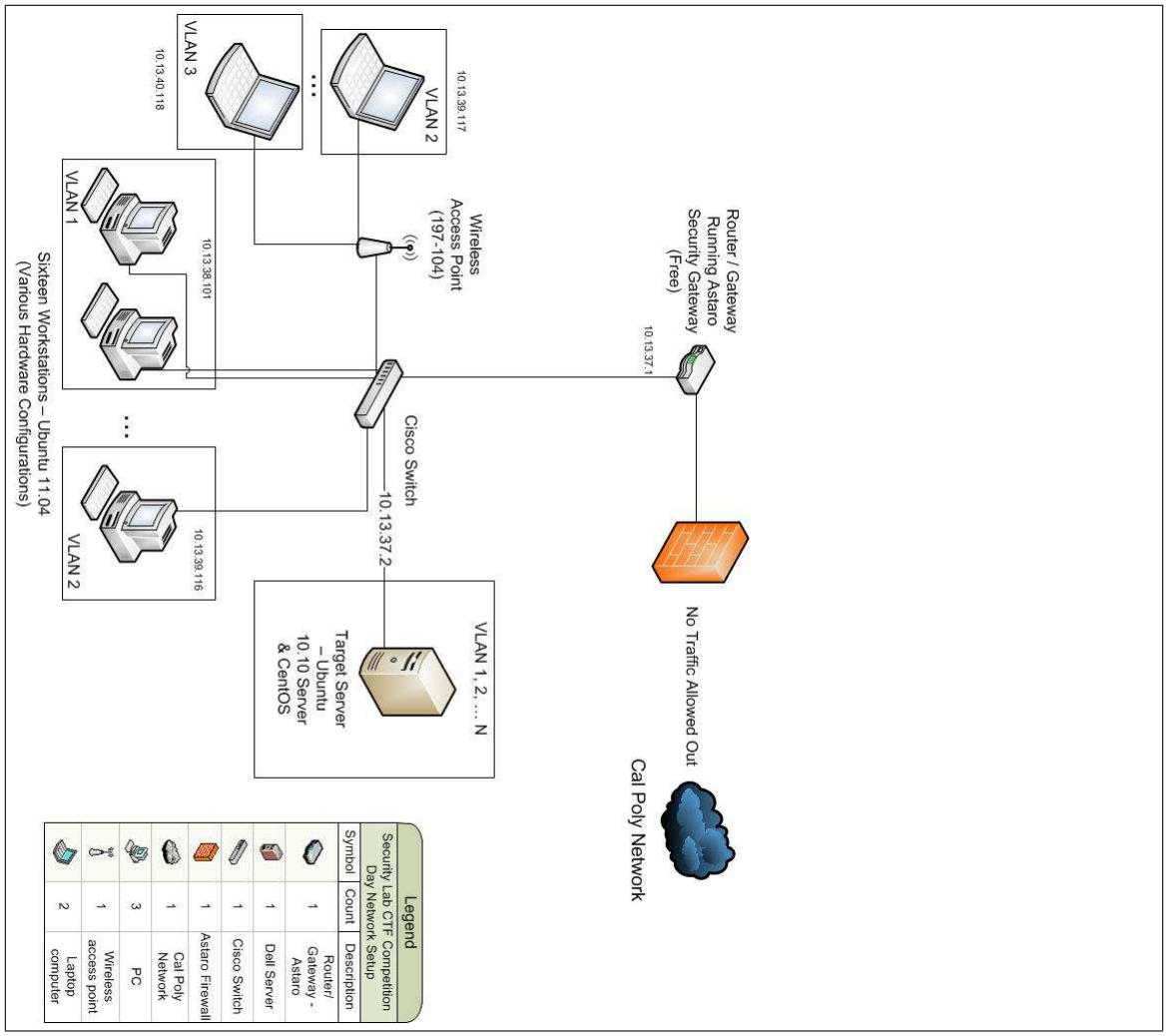

Figure 1: A planned network diagram for the Duke Nukem CTF Scenario. Note: as described in the problems and solutions sections, the planned network setup did not end up including any of the described VLANs and the network connection out of the lab was physically removed.

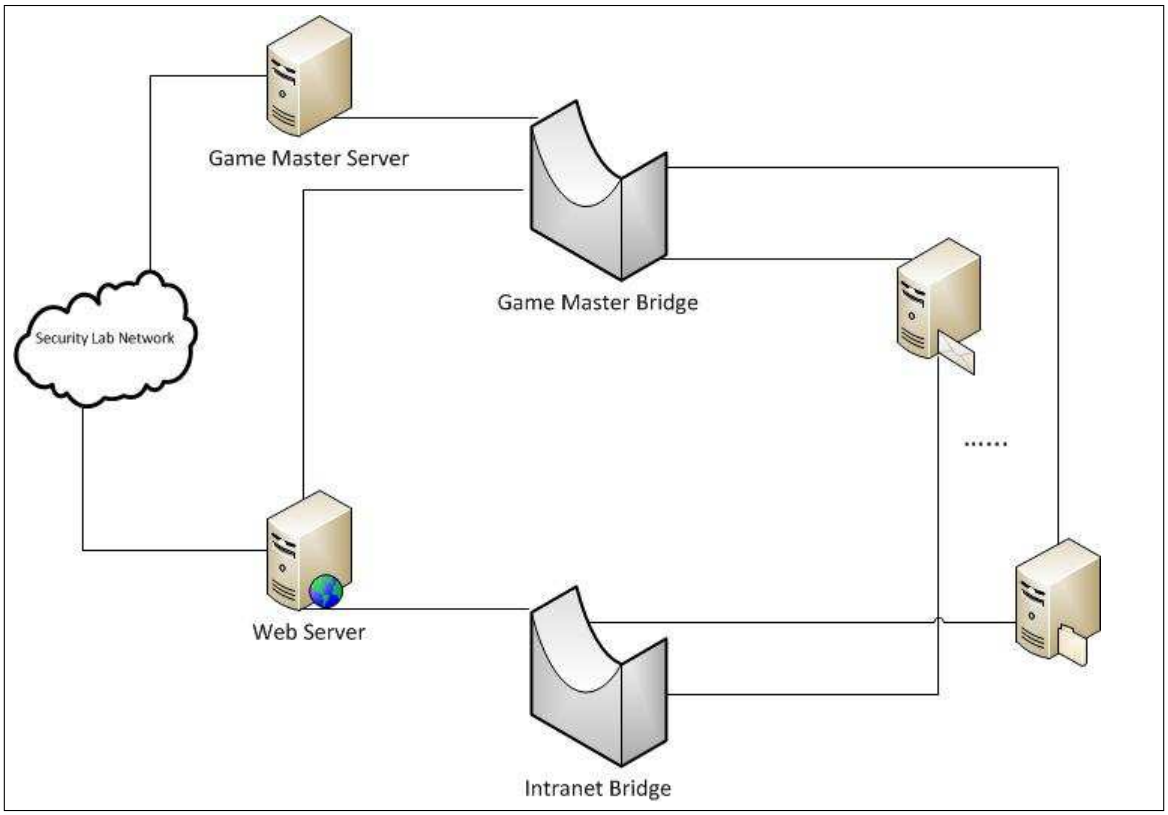

Figure 2: A network diagram for the pseudo company's Intranet and Game Master network.

```
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.