Title

By Dirk Cummings

Orion Miller

Senior Project

Dr. Philip Nico

December 2, 2011

Abstract

Abstract

Contents

1 Research of other CTFs

2 Objectives

3 The Doom Scenario

3.1 Objectives

3.2 Design

3.3 Game Day Results

3.4 Problems and Solutions

4 The Star Trek Scenario

4.1 Objectives

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

- testing the individuals 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 town time spent repairing and restoring the virtual machine servers images. Additionally, by designing new services from the ground up we can demonstrate to participants and give them experience with the common cause of vulnerabilities: poor design.

The first scenario was good for getting the uninitiated starting in security, however, it did lacked 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 learn techniques and practices. 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 dont. 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 [**?**], and we couldnt 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 the

Game Play and Services sections. 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 their own server to manage. We were even fortunate enough to provide both teams with their own rooms to work in together. Each team would initially receive the same source code of the services 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 represent a star ship, then it stands those who feel they cant contribute could fly the ship around and score points for their team by shooting the other teams ship.

4.2.2 Score Keeping

4.2.3 Services

On competition day, there were three implemented services in production for teams to work on with three additional services planned to be released 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 competitors implementation.

The base functionality is implemented by us and provided in unlinked pre-compiled form for each team build from. It is meant to represent the initial implementation of ships services programmers. There are a few seeded vulnerabilities and even more poor and hastily made designs. Base implementation of a service is responsible only for:

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

While the exact functions for each service differ, every services base implementation has functions (defined externally in header files) for the competitors code to call before finishing processing

an incoming request. For each of a services base function, there is a corresponding competitors 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 is available in Diagram XX.XX. This is one way in which the players can 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 structure but were allowed to add on to fulfill any future needs. The initial structure contained only enough feilds to control each specific command of a service. Any security or checksum features were left up to the participants to develope and deploy.

4.2.3.1 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.

4.2.3.2 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 engines 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 hes not a bloody miracle worker; he can only work so fast.

4.2.3.3 Navigation Responsible for setting a ships course direction, managing the ships internal representation of the map, and managing the engines.

4.2.3.4 Communicatiosn The communications service is independent of all other services except for the power service, and instead of managing some aspect of a ship, presents a team with a number of cryptography puzzles to be decrypted, solved, and posted back to the Score Keeper for validation.

4.2.3.5 Weapons

4.2.3.6 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. These were the power, engines, and navigation services. 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. Very little code

was actaully being written, but design flaws and vulnerabilities were slowly being realized. One or two members of Team A had previous experience in UNIX System Admnistration and immediately began setting up monitoring tools to track incomming 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.

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.)

4.4 Problems and Solutions

5 Comparing Scenarios

6 The Next Ideal Solution

6.1 Game Play

6.2 Network

6.3 Services

6.4 Score Keeper

7 Conclusion

8 Bibliography