

Open Capture the Flag 6 Whitepaper

The Open CTF Group

September 3, 2010

1 The Game

oCTF consisted of two types of challenges, vulnerable services and forensics challenges. This format was chosen in an effort to expand the covered topics. The scoring ran on five minute intervals, with flags for each service updated at the end of each period. Vulnerable services were designed to be repeatedly exploitable so as to encourage teams to automate their attacks and score for each interval. Scoring for services was based on standard values for challenge difficulty rating, with a bonus of 10% given to the first team to exploit each challenge for a given scoring interval. Forensics challenges had a single flag hidden in each and were given one-time values based on the challenges relative difficulty. These values depreciated over time, with a 5% reduction from the initial maximum base value every 15 minutes until reaching a plateau of 50%.

2 The Infrastructure

The network was laid out to provide a robust infrastructure that could (hopefully) stand up to layer 2 attacks and stop teams from making the game unplayable to other teams as had occurred in prior games. The network consisted of 26 VLANs, including separate ones for game administration, the game servers, and one for up to 24 participating teams. These were used to separate the traffic between teams and minimize impact to those players newer to the world of Capture the Flag-style games and perhaps not ready or anticipating attacks from other teams. The main hardware used was not a Layer 3 switch, so a separate router was required to direct all the traffic across the VLANs to make the network actually function. A custom Live CD was developed based on Fedora 13, which served as the router for the game. This included safeguards such as read-only file system access and the ability to simply reboot in the event the machine was ever compromised. As a default rule all traffic was dropped; then white listing was done to allow traffic to be routed to specific destinations (the game VLAN). Teams were isolated and not allowed to communicate via the network with each other or the game administration VLAN. Only traffic originating from the game network was routed to the players. The game VLAN was configured to be reachable from anywhere, and the admin network was only exposed to the game VMs.

In order to minimize downtime and require as little physical equipment

possible, the decision was made to virtualize the entire game. Virtualization allowed us to snapshot machines in a known working state. This would allow images to simply be reverted to a snapshot in the case a machine was broken or compromised in a way that disrupted the game. For virtualization technology VMware ESXi 4 was chosen due to it being a relatively lightweight hypervisor-only system, which we had hoped would help performance and be one of the more secure options available.

Each virtual machine was a customized install of Slackware 13.1 which was chosen for its simplicity, security history, and vanilla packages. The virtual machines were kept as bare as possible to minimize the chance of someone having a working out-of-the-box exploit and to make local exploitation more difficult. Development tools were specifically left off requiring that an attacker compile (or build on Slackware 13.1) and upload their binaries rather than being able to readily build tools on the machines. Sane limits were placed in an effort to help prevent players from performing DoS attacks on the machines such as fork bombs. In order to make the systems more secure and push new versions of challenges more quickly, home directories were mounted as NFS shares with read-only access. A standard UID and GID scheme persisted across all game machines in order to ensure that permissions were kept and mounts were only allowed from systems within the actual game servers VLAN.

3 Services

2speed (20 pts.)

2Speed was a SUID binary, which would copy the value in the key file to a temporary file that was world readable and then remove it. This creates a race condition in that the file exists on the system, where it can be read by an attacker if they were to know the file name and read the file before it was removed. Players would have to reverse engineer the binary to figure out the naming scheme, which introduced a known value vulnerability as well. Since the random number generator is seeded with the current time, an attacker could predict the file name without having to simply checking all of the files within /tmp.

Blooper Surf (10 pts.)

Blooper Surf was designed to force players to write a port knocker, under the guise of racing an ascii art blooper around to collect 8 red coins. Teams were given a few clues as to what their objective was, including the last octet of their IP address encoded in octal placed into the head of their blooper. They were also given an octal number (with no indication of the encoding) that was to be the first location of an available red coin or port to knock on. The ports were chosen at random each time from a range of 51,000 to 61,000 ensuring that the encoded values were not valid base 10 port numbers. Ports were only opened for 3 seconds each, but only after each completed checkpoint was reached by a player sending their unique blooper to the requested port. After a successful run reaching the finish line port, a flag file was read and dumped as part of the output.

Cipher Block Speed Run (20 pts.)

Cipher Block Speed Run implemented the XECrypt encryption algorithm. Upon connection it presented players with a brief message followed by dump of encrypted data. In order to get the flag, the attackers needed to brute force a key from the cipher text and submit a response within ten seconds. To add difficulty to this challenge, the encrypted plaintext was chosen at random from one of six messages of different lengths. If the response was not within 10 seconds, the player received a taunt and the connection closed. If the key submitted evaluated as correct, the service would respond with the flag.

Dr. Mario (10 pts.)

Dr. Mario was an example of poor input checking and resulting command injection vulnerability. At first connect the service prompted an attacker for the name of a patient. This input text would then be reversed and XORed against 0x45. The attacker would then be presented with the result asking if it was correct. Unrecognized responses or indicating the input was incorrect would each result in the respective taunt, and the connection dropped. If a reply in the affirmative were received, the service would check to see if the first part of converted input, delimited by semicolon, tried to match a name in the known list. Names were chosen based on common Mario characters and

were an easy guess. If the name existed, the service would hand off input to a program executed locally which would then cat the sanitized input string. Due to the very limited input checking an attacker could chain commands together to explore the local file system and eventually read the flag file.

Thumbd (20 pts.)

Thumbd was similar to a finger process. When connected, it would prompt an attacker for a user id. The service would look up the user id and respond with the user name associated with it. The get_input function was vulnerable to a simple stack-based buffer overflow from the use of the gets function to read input rather than one of the various bounds checking input functions. By overflowing the stack one could easily gain control of EIP and return into shellcode. ASLR was disabled allowing the attacker to easily use a generated reverse bind shell or any number of other payloads to obtain the flag.

TWSS (30 pts.)

TWSS unfortunately never worked in the live game due to networking issues, but was a hackback service disguised as an SSH client. By modifying the client and servers authentication tokens and version, the normal ssh client was rendered useless and unable to connect. The client, which was distributed, was stripped and statically compiled to allow all teams to be able to run it, regardless of their *nix flavor and library versions. The point of the challenge was to reverse engineer the client and create a working one which didnt have a reverse bind shell in it. A separate program called broadcaster was running for each scoring interval changing the authentication credentials and sending out a broadcast packet with the new password in it. Teams would have to listen on the wire for the packet and pull the password out for each scoring interval to log in and grab the flag.

Warp Zone (30 pts.)

Warp Zone was designed as a reversing challenge that contained a simple format string exploit. Teams were given a stripped binary which contained a main function susceptible to format string exploitation, and an unused statically declared function that would cat the flag file had it been called.

Reversing was eased by minimizing functionality of the service, network connections and input/output pipes were handled by a neutral program. Unfortunately due to a compiler optimization that showed up in the game binary which skipped dtors if no dtors functions were declared, this program was never exploited as intended, and although a corrected binary was eventually created once the problem had been identified, the game ended before live exploitation could take place.

4 Forensics

Forensics 100 (500 pts.)

Forensics 100 was a gif concatenated with a zip file. Inspection of the file with strings revealed a fairly obvious clue that this was the case. Due to the preservation of gif file format information in the header and the zip information in the footer, the zip file could easily be extracted. The zip contained a tiff image that included the flag value.

Forensics 200 (1000 pts.)

Forensics 200 was a bitmap image. By zooming into Marios mouth a 1 pixel high strip of colors could be seen on the tongue. Converting each pixels hexadecimal color code to ASCII characters revealed the flag.

Forensics 300 (1500 pts.)

Forensics 300 was a discolored image from the original Super Mario Bros. If closely inspected text could be seen in the bottom right corner. By viewing only the alpha channel the flag became clear and readable.

Forensics 400 (2000 pts.)

Forensics 400 was an OS X .dmg image which contained a deleted jpeg file. This could be recovered by inspecting the image with any number of well known forensics tools. In the jpeg there was EXIF data, which contained a base64 encoded string as the Author value. Decoding this resulted in flag.

5 Conclusion

Throughout the game a number of attacks were performed that we believe are worthy of mention, some of which resulted in bonus points. The attack which resulted in the largest impact on the game was from team Neg 9, who was able to successfully gain root access to one of the game VMs through exploiting the 2speed challenge. A write-up explaining their attack can be found at <https://neg9.org/wiki/DC18-oCTF-2speed-local-root>. The second attack which drew our attention was from team Vand, who would have been able to gain full read access to any of the home directories by using a reverse SSH tunnel through one of the game nodes, had we not properly secured our NFS server.

Finally, we would like to thank everyone for playing this year; we hope that the game was enjoyable and challenging. Also, we are grateful for everyones patience throughout the first day as the network and scoring system was fixed and the game brought to a playable state. We look forward to hosting an even better game next year with more challenges, space for more teams, and a more robust backend.