1 Intro

Der iCTF (international Capture The Flag) wird jährlich von der University of California, Santa Barbara veranstaltet und ist der gröSSten Hackerwettbewerbe dieser Art. Der iCTF integriert Angriff und Verteidigung zugleich. Die Teilnehmer müssen gleichzeitig Flaggen von fremden Servicen klauen, sowie ihre eigenen Service patchen um diese unangreifbar zu machen. Dieses Jahr fand der Wettbewerb am 04.12. statt und ca. 40 Teams nahmen daran Teil. Dabei belegte unser Team (in23canation) den 22. Platz. Die Dauer des Wettbewerbs wurde mit 24 Stunden angekündigt jedoch kurz vor dem iCTF auf 8 Stunden reduziert. Die Stimmung beim Wettbewerb war durchgehend positiv und insgesamt war der CTF ein groSSer Erfolg. Durch etwas Werbung vorab war es möglich eine groSSe Teilnehmerzahl aus allen Semestern zu begeistern sich unserer Gruppe anzuschlieSSen. Schon bei den zwei vorangehenden Informationsveranstaltungen waren ungefähr 60 Studenten anwesend. Beim iCTF selber waren rund 25 Teilnehmer aktiv dabei.

1.1 Network-Setup

The setup of the iCTF 2015 is shown in the graphic 1.1. The Router has a private VPN running, tunneling all traffic through the THI-Network. Also the Router creates 2 sub-networks. The ”bad network”(shown in red) holds the iCTF-Router and vulnerable VM, as well as some attacker PCs, which will run the exploits against other teams. In the ”good network”(shown in green) are the participants, who have a local copy of the vulnerable Image and develop patches and exploit. It’s to mention, that the router has ip-tables-rules specified, so the ”bad network”can not communicate with the ”good network”. Also, the Attacker-PCs have to route their traffic through the CTF-Router. This should be solved differently in the upcoming CTFs, since the separation of the 2 networks needed a lot of time to be configured.

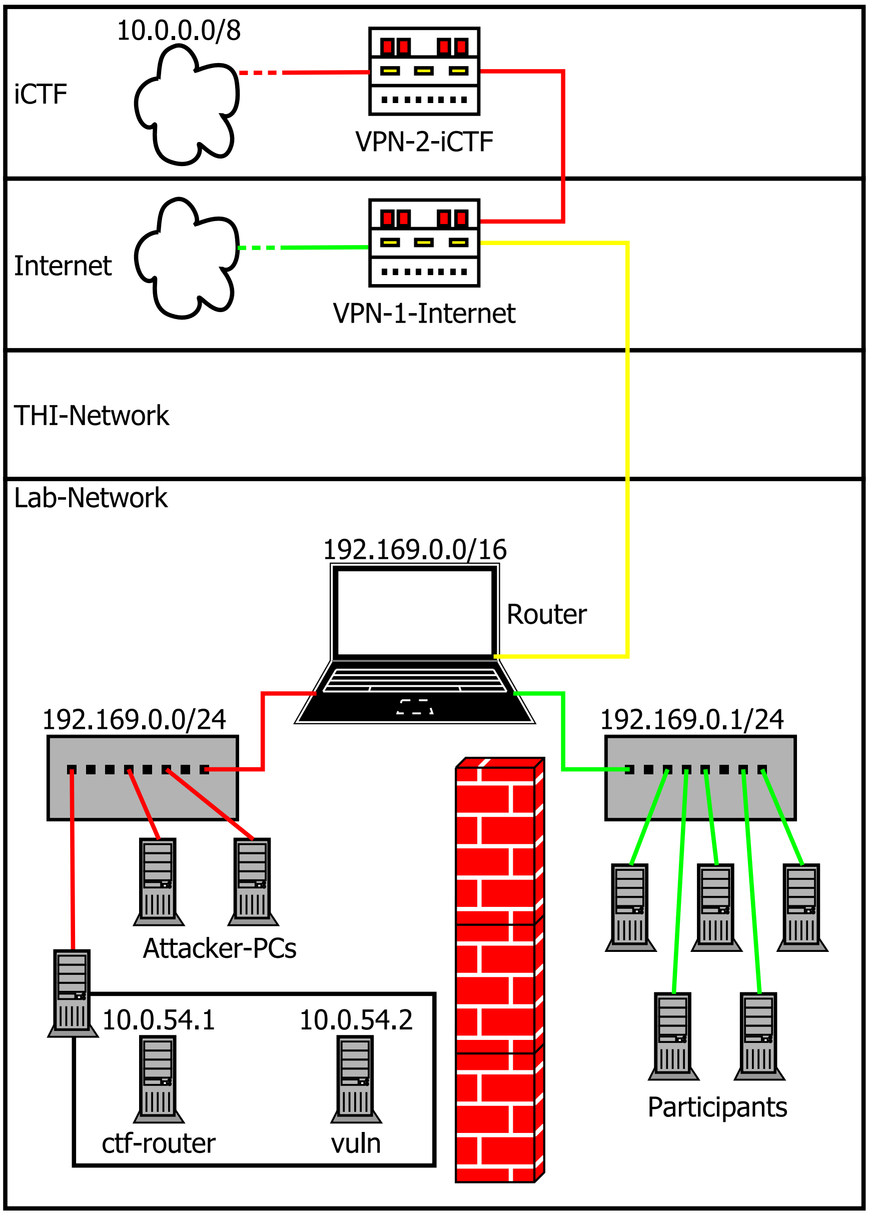

Abbildung 1.1: Network-Setup iCTF 2015

2 Dungeon

The Dungeon service was a consoleservice which presented a litte game. The player needs to run through the dungeon to finally kill a dragon in the end. To do this, two items are needed. Firstly a golden sabre and second a magic shield. If either of the items is missing the player will die to the dragon.

To get the sabre the player needs to find a secret room and in there has to get the name of an airplane picture right. If he does he will get the sabre.

Now the player will get in a room with a dwarf. He says if you get the number i guess you will get my shield. But the number he has is random generated and nearly impossible to get right. Also if you get the number right he will tell you that he gives you the shield but the variable HAVE_SHIELD will not change. To achive the shield the player has to go to a room with a gnome which asks the player's name and then says Hi! playername. In this function there is an adress returned with the variable HAVE_SHIELD. So the player need to type in anycharacter than %n so that the adress will be overwritten with 0x1.

If the player encounters the dragon with both items he can slay the dragon with the command "kill dragon". After that he will be asked what his name is. Here comes the tricky part. With a Bufferoverflow it is possible to alter the return adress so that the function will jump to the secret treasure room where the flag can be optained.

To do this the player have to type 88 random characters followed by 0xDEADBEEF in ASCII than 8 random characters again and then the adress to the storage room. The 0xDEADBEEF is needed because a value is checked for change. If you overwrite it with the bufferoverflow the service will disconnect the player. Both 0xDEADBEEF and the return adress have to be turned because little Endian is used.

3 Tweety_bird

The service tweety_bird was a console service, which saved a secret in a file and stored a password for it in another file. This was done by selecting write(“X“) and entering “<filename><secret><password>“.

The secret could be retrieved by selecting read(“R“) and entering “<filename><password>“. The flag was the stored secret. After decompiling of the c-code and checking it, it was determined, that the password was written in an array of size 50. After that the password is compared with the right password or if its not empty.

In case of a bufferoverflow the compare value for empty will be overwritten and any password will be accepted. A first try with the Passwort 51 times “A“ already was accepted and the secred was retrieved because the files were unencrypted. Closing the found security gap was not tried, because of the missing original c-code and time issues.

4 Text_File_Store

If you open the site you can see a form where you can login with a username and password or you can enter a short message you want to store. After storing some text you get the username and the password to look for that text later. The username and the password are calculated at runtime of the service.

Every parameter of the service are passed in the URL via GET. In the viewFile.php file you can find a parameter called ”file”. This parameter is used to access a file in the directory where all textfiles are stored.

But if you modify this parameter you can read the content of the access.log file without any verification. In this file you can see every connection to this service. At last you have to filter for the flags and read them. Also it is possible to inject a command with an adjusted url call. It generates a file on the server with the filenames in the service-directory. With this information it is possible to open the files containing the flags.

5 ATM_Machine

The ATM_Machine Service is a bank automate service in which the user has three options. It is possible to create an account with a ping, to check the balance of an account and to withdraw money from an account. If the user tries to withdraw more money than he has, the service will close the connection.

At First the ATM.jar File was Decompiled using the JAD-Decompiler. After that a simple string concatenation was found which leadd to the possibility of a SQL-Injection. After that the structure of the Database was analized and with the SQLite editor test injection were tried.

After some testing the solution was to connect to the service, than choose option 1 to check the account Balance. After that the service ask for an account number here the user can type in any char. Now the service needs the pin and here we can use the SQL injection. The statment we need ist UNION ALL SELECT acnum, cash, password FROM login WHERE acnum = "“ + self.flagID. Now the service returns the password for the falgID which is the flag itself.

For the statement we need the UNION ALL SELECT to unite the existing statment result with the injected one. We only need the password from the database but because the service only returns the thid collumn we need two previous ones. Thats why acnum, cash, password was used with password in the third collumn. “FROM login WHERE acnum = "“ + self.flagID“ was used to read the password from the correct table using the given account number which was the flagID given from the provider.