RCCTF Setup/Challenge Guide

RCCTF is a remote-controlled car driven by a Raspberry Pi. The car's computer comes with several beginner-level cybersecurity challenges. Solve the challenge to gain control of the car!

Initial Setup

The entire car is powered by the Raspberry Pi. Simply connect the Pi to a power source (i.e. the portable charger/battery) via the USB-C port on the board to power it on. Please allow at least 30 seconds for the system to boot properly.

Once fully booted, the Pi will serve as a wireless access point. Using your own computer, connect to the WiFi network corresponding to the car you are using. For example: `RCCTF-Red146`

Your computer and the car are now able to communicate with each other over the network.

To begin solving challenges, first connect to the Pi via SSH. Enter the following command in the terminal of your computer: `ssh pi@10.3.146.1`

IP address will vary depending on which car is being used. By default, the password is `ash whole dirty protest`

Once you have successfully logged in, change directories into the `CarPlatform` folder using the `cd` command. From here, you can change directories into the folder that corresponds to the challenge you wish to play. Simply use the following command to begin the challenge: `bash docker_run.sh`

See "Challenge Solutions" for instructions on how to conquer each individual challenge.

Network Configuration

RCCTF's network is controlled by a service called RaspAP. To access the configuration page, connect to the car on port 8080 via a web browser. Formatting in the address bar is as follows: `10.3.146.1:8080`

Login using the username 'admin' and password 'AdAstraPerWifi'.

From here you can change settings such as SSID, IP address, DHCP settings, etc. Everything should already be set up in the optimal configuration, so tinkering around in here is not recommended.

Challenge Solutions

- **WebApp Exploitation**

  In a web browser, type the IP address of the car into the address bar. You will be greeted with a website containing four arrow buttons which presumably control the car. Unfortunately, clicking the buttons returns the following error message: “You're not authorized to drive this vehicle…”

  Using “Inspect Element” or by simply navigating to your browser’s developer tools, you can view the web page’s HTML source code. There are two sections of particular interest. The page is running some Javascript code, as indicated by the <script> tags. The first script seems to store a certain value in your browser’s Local Storage. The second would appear to check that value and use it to decide whether you should be allowed to drive the car.

  Using your browser’s developer tools, access the local storage. There is a key “authorized_to_drive” with the value set to “False”. Change this value to “True”. Now, on the webpage, try using the arrow buttons again.

  Success!

- **Reverse Engineering**

  Using a program such as telnet or netcat, connect to the car on port 1337 like so:
  `telnet 10.3.146.1 1337`
  You are given a prompt that reads “Please authenticate with your valid driver license >>>”. Submitting an incorrect license causes the program to close the connection, and you’ll have to reconnect.

  The basis of this challenge is a program simply called “car”. Depending on how you have chosen to run the challenge, you may either examine the source code directly or disassemble the compiled binary with a program like BinaryNinja or Ghidra. Either way, you will gain insight into how the license is verified.

  A good place to start in any reverse engineering challenge is by analyzing the function `main()`. It can clearly be seen that the only thing this function does is call another function called `authenticate()`. Looking at this function, we can see that this is where the prompt is displayed. More importantly, this function seems to return a different response depending on the result of another function, called `license_check()`, which is only called if the license is contains exactly 8 characters.

  This function can be divided into three parts. The first part is a simple `if` statement which checks to make sure that the first two characters of the license

are “FL”. Next, there is a `for` loop which iterates through the rest of the remaining characters. On each iteration, the bits representing the value of the `res` variable are shifted to the left four times, then the value of the current character in the license is added to that variable. After the `for` loop is completed, there is one more `if-else` statement that will return true if and only if the value of `res` is `54946352`.

Now that we understand how the program works, we need to find a way to perform these operations in reverse and find a valid license.

One method of solving this is with a symbolic execution engine such as Z3. Writing a simple python script which creates a Z3 solver with all of the necessary constraints easily gives us the valid license: FL133700

- **Buffer Overflow**

  Using telnet or netcat, connect to the car on port 1337: `nc 10.3.146.1 1337`. You are given a message that reads “We are sorry, but our car service is not available at this time. Please provide your contact info and we'll get back to you as soon as possible >>>”

  Analyze the program either by using a program like BinaryNinja or Ghidra, or by simply reading the source code. The main function calls another function called `authenticate()`. This function will only return true if the value of the `access` variable is “GRANTED”. Unfortunately, there doesn’t appear to be a way to change that value within the program.

  Thankfully, you’re a hacker! There is a very simple way to break this program. If you look closely at this function, you can see that the `contact` variable has been allocated 16 bytes of data, but the call to `scanf()` that reads our input will accept 24 bytes of data. This means that the program is vulnerable to a buffer overflow. If we get the program to send more than 16 bytes of data to `contact`, the remaining bytes will overwrite whatever is on the next level of the stack in memory. We can see that `access` is declared immediately after `contact`, meaning `access` would come immediately after it on the stack. Therefore, any excess data sent to `contact` will begin to overwrite the value of `access`.

  We know that we need at least 16 bytes to reach the variable we want to change, so we will pad our input with 16 random characters. After the padding, we add the word GRANTED. Now that we have all of the info we need, our final payload should look something like this: `AAAAAAAAAAAAAAAAGRANTED`

  Submit the payload, and you’ve got control of the car!