Department of Mathematics and Computer Science

2IC80 - Lab on offensive computer security

Hue Hijacker

We hack hue.

Group 26

Authors:
William Trinh - 1440187 - w.trinh@student.tue.nl
Paul Vlaswinkel - 1430173 - p.r.vlaswinkel@student.tue.nl
Rinse Vlaswinkel - 1312529 - r.w.vlaswinkel@student.tue.nl

June, 2021

1 Introduction

With the increasingly popular Internet of Things (IoT) devices on the market, consumers acquire more of these so-called "smart devices". One of the popular devices is the Phillips Hue lamp. Using this lamp requires a direct Wi-Fi connection, or a Phillips Hue Bridge. Although the premise of the Hue lamp is only to control the colors of the lamp, it is rather vulnerable for attacks, which is relatively unknown to the public.

In this paper, we investigate the Hue control protocol, between the user and the Hue bridge, and how the Hue can be hijacked by an attacker. We present a web interface, where the attacker can essentially deceive the victim and gain control of the Hue interface.

The code for our attack can be found on our git repository: https://github.com/RinseV/hue-hijack
A video demonstrating our tool can be found here: https://www.youtube.com/watch?v=9GdWuC-OoZU

2 Attack description

2.1 Normal scenario

In a normal scenario, without an attacker, the end-user would use the Philips Hue app to send a SSDP discovery broadcast (more about this in chapter 5) to find the Hue Bridge the user wants to connect with. Once found, an authentication request is sent from the app to the bridge, then, the end-user has to physically press the "link" button on the bridge to finalize the authentication procedure. Once authorized, the end-user is able to interact with their Hue system and change the status of their lights and any other connected devices.

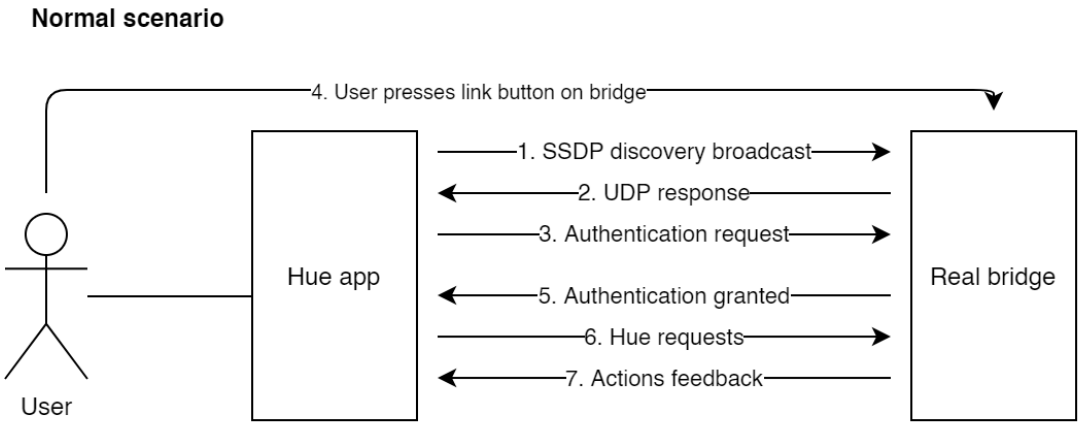

Figure 1: Normal scenario

2.2 Original plan

Our original plan was to perform a man-in-the-middle attack on a Philips Hue bridge. The end-user would be using the Philips Hue app to connect to their bridge, in an ideal scenario, we would have our own server emulating a bridge. When the user would then make a request to connect to their own bridge, our server would intercept this request and respond to the request of the user, making the user think they're connecting with their own bridge. When the user wants to authenticate with the bridge, the user will send an authentication request to the bridge (our server) which we can forward to the real bridge. Then, the user will press the "link" button on the real bridge, giving our server full access to the bridge, we can then authorize the end-user on our server by programmatically pressing our own link button. From then on, the user would send hue requests to our bridge which we *could* forward to the real bridge, but this is not necessary. We are now able to control both the real bridge of the user as well as block or modify any incoming requests from the end-user to the bridge.

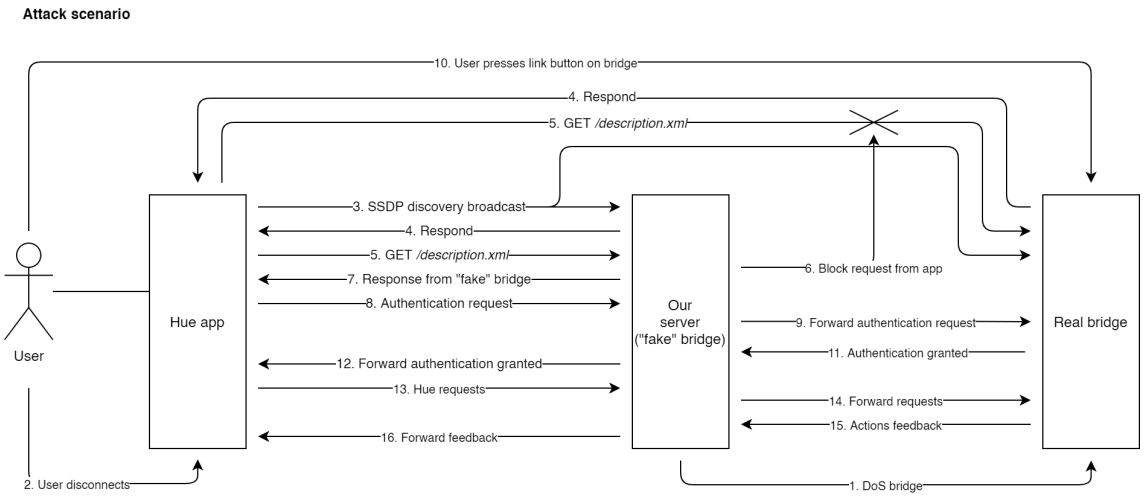

Figure 2: Original attack scenario

Of course, all of this requires the user to not be connected to their own bridge beforehand. Luckily for us, the Philips hue bridge is quite susceptible to DoS attacks meaning we can DoS the bridge for a few seconds, causing the Hue app to disconnect from the bridge. During this time, the end-user will most likely try to reconnect with their bridge, at which point we can start blocking discovery responses from the real bridge and start sending our own.

2.3 Adjusted plan

Unfortunately for us, at the start of the project, Philips had just updated their Hue app causing most known Hue bridge emulators to stop working. While we tried to work around this issue, the firmware of our own Philips Hue bridge was also updated to accommodate the new app, and reverting firmware updates is not something we wanted to do, so we opted for an alternative attack.

In our adjusted attack, we authenticate our own server to the bridge together with the end-user. Like the original attack plan, we first perform a DoS attack on the bridge to prevent the user from performing any actions on the bridge making them disconnect from the bridge. Once disconnected, we can start sending our own authentication requests to the bridge. Then, once the end-user will start the authentication process with the bridge, the user will have to press the "link" button again. While this is happening, our server is still sending authentication requests. Once the user presses the button, either the user or our server will be authenticated (more about this in chapter 5). Once authenticated, while unable to directly intercept the user's requests, we are able to change the state of the Hue devices ourselves, access all connected sensors and see all authenticated users.

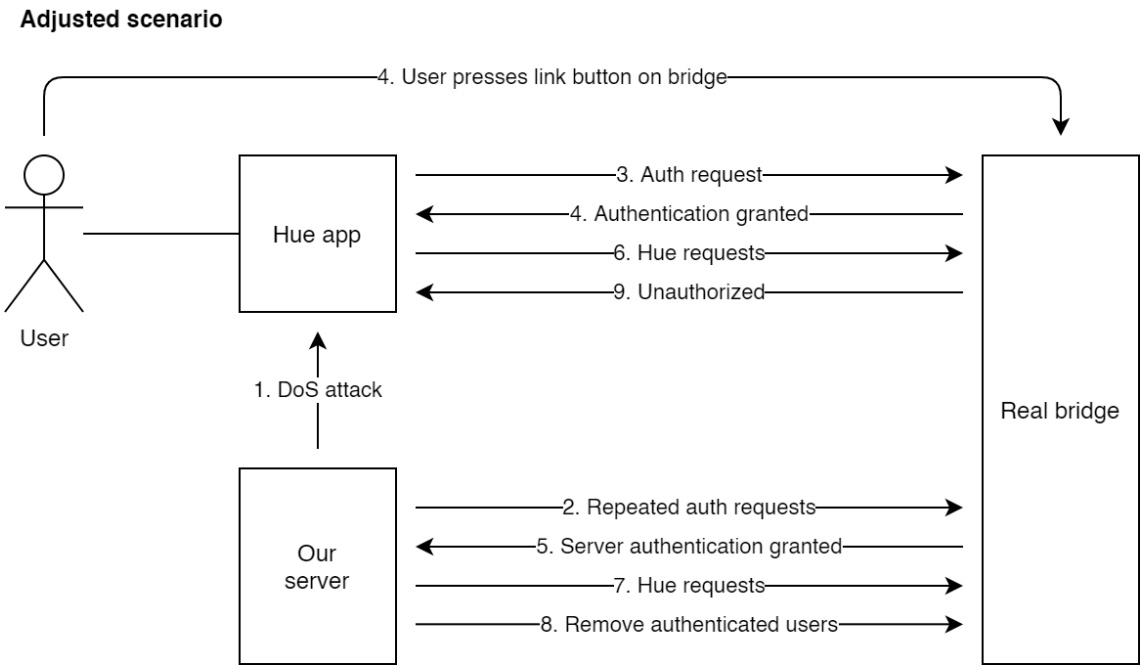

Figure 3: Adjusted attack scenario

3 Setup

For both the original attack plan and the adjusted attack plan, Node.js is used with a React [6] frontend. In the original plan, we also need to set up a server to listen and respond to the requests made by the Philips Hue app. For this, we use Express.js [3].

3.1 Original Plan

In the original attack plan, we create an Express server that acts like a Hue bridge. We can first flood the Hue Bridge with requests so that the user is prompted to reconnect to the bridge. When the user tries to reconnect, we block the discovery of the Hue bridge and make sure that we are discovered instead. We do this by ARP poisoning. After the app has discovered the services running, it checks an XML resource located at an endpoint in the service. We can block the app's request to the Hue bridge such that it never receives a response; since it never receives a response, it will not be seen as an active Hue bridge and will not show up in the app. We need to make sure our Express server shows up in the app; we do this by responding to the same discovery query and having our document, which it requests, available.

The user then tries to connect to the Express server through the standard authentication protocol. We create endpoints for the server such that we can imitate this authentication protocol. In the meantime, we also send an authentication request to the Hue bridge to set up our authentication. When the user presses the button on the bridge, we are authenticated. Moreover, we now also forward this authentication to the victim to ensure the victim does not know we have hijacked the Hue bridge. We can now set up the app's endpoints and forward them to the Hue bridge to make the app function normally. However, we can now also send requests ourselves and even read, change or block actions sent by the victim. Thus we now have complete control of the bridge. To make this easier to set up and control, we first have a setup page where we can fill in or find the IP of a Hue bridge and an optional username, which we can authenticate with when we already have been authenticated. There are two buttons, one to DoS the bridge and one to spam authentications. We can not do this simultaneously since the bridge cannot handle the authentication requests when it is under a DoS attack. Once we are authenticated, we move to an overview to see all connected lights with a button to see and change their power state. We can also see all sensors with corresponding values and all users who have access to the bridge. To further change the settings of the lights, we have another page. On this page, a light can be selected. First of all, we can see more information about the light, such as the model number or the last time the light was updated. Secondly, we can see and change the power status of the light again. Finally, we can change the color of the light.

3.2 Adjusted Plan

The adjusted plan starts similarly; we first DoS the bridge to get the user to reset the connection with the app. However, instead of blocking the discovery of the bridge and impersonating a bridge, we now spam authentication requests to the bridge to be the first one to authenticate when the button is pressed. This way, we are not in the middle between the user and the bridge. However, we still have the same access as the bridge as in the original plan. We can read sensors, turn lights and other smart devices on and off and change the settings.

4 Attack analysis

Suppose the Hue is installed in a shared room, such as the living room and only one person can control that hue lamp. When a bystander wants to change the color of the Hue lamp, it needs to go through the owner of the lamp. Instead of asking the owner to adjust the Hue lamp, the bystander can use "the tool" to directly access the Hue. When using the tool, the bystander first performs a DoS on the owner, which leads to the owner "resetting" the Hue. In this step, the Hue forms a bridge in the connection between the owner and the Hue lamp. Then, the Man-In-The-Middle principle is applied, where the bridge now has full access to the Hue lamp.
After using the tool, both the owner and the bystander has access to the hue lamp. However, the bystander now has priority over the Hue, as the tool essentially is a Man-In-The-Middle bridge. The bystander can simply keep track of what the owner does with the Hue, or directly control the Hue lamp without interacting with the owner.

Another scenario is when a friend has a Hue lamp in his home, and you, as bystander, want to pull a prank on him. In the same way, the tool allows the bystander to gain full access on the Hue, without the friend noticing the Hue lamp has been hijacked. The friend can keep using the Hue as normally, but the bystander has priority control with the tool, where the bystander can let the Hue lamp behave "on its own".

5 Attack engineering

5.1 Philips Hue Discovery & Authentication

Before explaining the technical details of the attack it is important to know how the Philips Hue discovery and authentication system works. For discovery, there are a few options: UPnP, N-UPnP, IP scan, Manual IP and mDNS [5]. While officially deprecated, we intended to use the first option. This method relies on the Simple Service Discovery Protocol (SSDP) [1] to discover a Hue Bridge. For this to work, the client who wants to discover a Hue Bridge should send out a UDP broadcast with the following content:

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: ssdp:discover
MX: 10
ST: ssdp:all
```

It is important that the broadcast is send with a `HOST` of `239.255.255.250:1900` since this is the standard IP address to send broadcast messages to for IPv4. The bridge will then respond to the broadcast (once received) with the following message:

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=100
EXT:
LOCATION: http://192.168.0.21:80/description.xml
SERVER: FreeRTOS/6.0.5, UPnP/1.0, IpBridge/0.1
ST: upnp:rootdevice
USN: uuid:2fa00080-d000-11e1-9b23-001f80007bbe::upnp:rootdevice
```

While the client may get other responses from devices, one should send an HTTP GET request to the `/description.xml` endpoint to find the model name of the device. The client can then check for a matching model name (in our case this was "Philips hue bridge 2015") to see if the response came from a Philips Hue bridge. Once the client has found the bridge's IP- & MAC-address, the client can move onto authentication.

Once we have found the IP address of the Philips Hue bridge, authentication is pretty straight forward. First, an HTTP POST request should be send to {{`bridgeIp`}}`/api` with in the body an entry for `devicetype`. Then, if the link button is pressed, the bridge will send back a `success` response. If the link button is not pressed or the devicetype is invalid, the bridge will send back one or more `error` responses [4]. It is also important to note that the first authentication request the bridge receives, once the link button has been pressed, is authorized. Any follow up authentication requests are denied until the link button is pressed again. Once authenticated, one is able to access all the bridge's endpoints using their username they received from the `success` response. More information about these endpoints can be found on both the official Hue website as well as third party websites [7].

5.2 Original attack

For our original attack, there are several steps that need explaining:

5.2.1 DoS attack

For a DoS attack, we can simply send continuous requests to an endpoint on the bridge that does not require any authentication. If we send these requests very fast with a minimal delay between them (5 ms), the bridge will be unresponsive to user requests and the app will disconnect the user.

5.2.2 SSDP discovery

For SSDP discovery, we have setup our own SSDP server that is able to send and receive responses to broadcast messages. In our case, we want to send a Philips Hue bridge response to the end-user letting them know our server is a Philips Hue bridge. Using the SSDP protocol, we are able to send SSDP responses to any SSDP broadcast request.

5.2.3 Blocking description requests

For the end-user to only discover our bridge, we have to block the discovery of the actual bridge. We do this by blocking the app's request to the bridge to check if it is a Hue bridge. If this request does not get a response, the bridge will not be recognized and will not be shown in the app. To achieve this, we poison the ARP cache of the phone to link the IP of the actual bridge to another IP that does not respond. We do this by creating an ARP packet with the IP of the bridge and a non-existing MAC address. When the cache of the phone is poisoned, and it tries to send a request to the IP of the actual bridge, it sends this request to the non-existing MAC address. Thus it will not get a response, and it will not be shown in the app.

5.2.4 Authentication request forwarding

Once the user has discovered our "fake" Hue bridge, the user will want to authenticate themselves. We can mimick the actual bridge's endpoints and forward any request coming from the user to the actual bridge (which we have also discovered). After forwarding these requests, the user will press the "link" button on the actual bridge causing our forwarded request to be approved. Now we can also authenticate the user on our own bridge.

5.2.5 Forwarding Hue requests

Once the user is authenticated, they will start sending requests for changing the state of lights and such. We can again forward these requests to the actual bridge, giving the user the impression that they are connected to their own bridge. In reality, we are simply forwarding the requests from our own bridge to the actual bridge. With this, we are also able to block any requests coming from the user by simply not forwarding them to the actual bridge.

5.3 Adjusted attack

The adjusted attack relies on the same DoS attack principles of the original attack. However, the authentication request and hue requests are slightly different:

5.3.1 Authentication request

Instead of forwarding the user's authentication request, we will make our own authentication request at the same time as the user is making them. Once the user presses the link button to authorize their own request, our request will also be authorized, giving us full access to the Hue system.

5.3.2 Hue requests

While we are unable to block the user's incoming requests directly, we can still listen to any changes on the bridge and revert them. We are also able to remove all authorized users by simply requesting the list of all authorized users and deleting them one by one.

6 Future work

The first thing to improve would be the attack itself. While we were unable to execute the original attack successfully, we expect that popular Hue emulators (such as diyHue [2]) will be updated in the future to accomodate the newest version(s) of both the bridge firmware and Hue app. Once updated, these emulators can be used and modified to act as a fake bridge for an end-user to connect to. With a working emulator in-place, the original plan can be executed.

Another place for improvement would be the interface for interacting with the bridge and the user. Currently, we are running a very crude web-interface that only allows for the most basic operations such as launching a DoS attack to the bridge and authenticating on the bridge. If a working emulator is found, the web interface should be extended to automate the entire process of hijacking the Hue bridge.

Lastly, after authentication, we should be able to modify the states of all connected devices and remove any authenticated users, while we have a very basic version of this already working, the interface could of course be extended to include all functionality the Hue app itself also has.

Bibliography

[1] Yaron Y. Goland et al. *Simple Service Discovery Protocol/1.0*. https://datatracker.ietf.org/doc/html/draft-cai-ssdp-v1-03. Accessed: 28/06/2021.

[2] diyHue. *diyHue*. https://github.com/diyhue/diyHue. Accessed: 29/06/2021.

[3] *Express - Node.js web application framework*. URL: https://expressjs.com/.

[4] Philips. *Get Started*. https://developers.meethue.com/develop/get-started-2/#so-lets-get-started. Accessed: 28/06/2021.

[5] Philips. *Hue Bridge Discovery*. https://developers.meethue.com/develop/application-design-guidance/hue-bridge-discovery/. Accessed: 28/06/2021, requires developer account to access.

[6] *React – A JavaScript library for building user interfaces*. URL: https://reactjs.org/.

[7] tigoe. *Instructions on controlling the Philips Hue hub*. https://github.com/tigoe/hue-control. Accessed: 28/06/2021.