Development Against Security Vulnerabilities of Bluetooth Low Energy Beacons
and
Case Study: VipAssistant

Prepared by
Yavuz Selim Yeşilyurt
Gökhan San
Alper Kocaman

Middle East Technical University
Introduction to Computer Security

Pelin Angın

28 June 2020

Contents

List of Figures

List of Tables

Chapter 1

Problem Statement

1.1 Overview

Beacons are small wireless devices that broadcast signals using Bluetooth Low Energy [1] (BLE) technology. Mobile apps on compatible devices listen for signals from beacons placed in the physical world and then trigger an experience via a beacon-enabled app when it enters or exits a range.

Similar to any other technological advancement, beacon technology came with its own safety and security problems. Malicious parts -as always- try to exploit beacon based systems to gain access or expose the service they provide.

Security issues of beacons may cause severe social and business side harm. Beacon vulnerabilities may let attackers access private user data or business entities. Since the effects of these vulnerabilities to the real world are significant, IT experts try to develop solutions to destroy these vulnerabilities.

In this paper several types of vulnerabilities of BLE beacons will be discussed with a case study of a project that already uses beacons for indoor localization.

1.2 Types of Vulnerabilities

In this report, five specific vulnerabilities of Bluetooth Low Energy (BLE) Beacons are going to be examined and their corresponding solution approaches' with experiments are going to be discussed using their real-life applications on case study, namely the VipAssistant [2] Project.

1.2.1 Spoofing & Piggybacking

Among the other vulnerabilities of beacons, the most commonly-known ones are Spoofing and Piggybacking. Since they are kind of similar to each other, they are going to be taken into account under the same subproblem. As mentioned earlier, the Bluetooth Low Energy Beacons (or simply all the beacons that emit Bluetooth signals to their nearby) are devices such that they continuously broadcast advertisement packets which have some kind of identifying information in itself that could help the capturing party to discover the specific beacon that advertise the packet with a couple of additional fields that could also help the party to calculate the distance between the emitter and capturer, such as RSSI [3] value of the beacon. As also discussed earlier, there are several protocols to regulate mediums used by Bluetooth Low Energy Beacon technology. The most widely-used and important ones of them are iBeacon and Eddystone protocols and the contents of the advertised packets of these protocols are given in the table below.

iBeacon	**EddyStone**
Unique Beacon Proximity UUID Value	Eddystone-UID: A unique, static ID with a 10-byte Namespace component and a 6-byte Instance component.
Beacon Major Value: First independent number assigned to the iBeacon, in order to identify it with greater accuracy.	Eddystone-URL: A compressed URL that, once parsed and decompressed, is directly usable by the client.
Beacon Minor Value: Second independent number assigned to the iBeacon, in order to identify it with greater accuracy.	Eddystone-TLM: Beacon status data that is useful for beacon fleet maintenance, and powers Google Proximity Beacon API's diagnostics endpoint.
Constant Preamble Data	Eddystone-UID: A unique, static ID with a 10-byte Namespace component and a 6-byte Instance component.

Table 1.1: BLE Packet contents by iBeacon and Eddystone protocols respectively.

As can be seen from this table, both protocols use both unique UUID [4] values and some additional identifiers to accurately recognize the source Beacon from its advertisement packet. Furthermore, these advertised packets are traveling unencrypted in the air as a form of Bluetooth Signals and therefore they are publicly available for any of the capturing parties near the beacon that emits the packets. However, that is the normal working routine of a beacon device since the main benefit of these devices is, the independent mechanism in the system they belong can advertise the information to the target user group even in places that have restricted Internet access. But this benefit of beacons can be exploited by any malicious third-party as well. By using a sniffing tool or a similar tool, both the broadcasted specific beacon UUIDs and other identifying data can be captured and can be cloned to different broadcasting devices to spoof users of the application that is owned by the original owner of the beacons. This can result in very problematic complications for the users of the original application, potentially leading to breaking all the pre-programmed assumptions of the application and resulting in eventual failure. This vulnerability of beacons is called Spoofing.

On the other hand, knowing the identifying information that is advertised by a specific beacon, third-party applications can potentially use existing infrastructure of beacons to leverage their own applications without resorting to any kind of consent from the owner party of the beacons. As a result, this behaviour of the exploiting third-party constitutes an unethical situation where the original owner of the beacons potentially does not want to happen. This vulnerability of beacons is known as Piggybacking.

1.2.2 Hijacking

Hijacking is also another problem that needs specific attention and could result in unwanted, fatal situations if not tended properly. The term which originates from "an act of unlawfully seizing an aircraft, vehicle, or ship while in transit; a hijack." is closely related its general version in Computer Security, namely the type of network security attack in which the attacker takes control of the used communication medium and captures the sensitive information that is transmitted between the parties. In the beacon world, this security attack is also available if the beacon-utilizing party has its application communicating (end user transmitting data to the beacons) with the beacons and possibly transmitting sensitive or private information between. Since the transmission between the beacon devices and end users are unencrypted, attackers' job becomes very easy as there is not any kind of defense mechanism against such interceptions. However, applications that do not communicate with beacons by transmitting data but rather only gathering the packets advertised by the beacon are not directly vulnerable to this vulnerability of beacons.

1.2.3 Cracking

Another attack type to Bluetooth low energy devices is cracking. Up to this point, and in the following Denial of Service(DoS) subsection, we introduced remote attacks. For these attack types, detecting the Bluetooth Low Energy signal or contacting with beacon device is enough for the attacker, and all attack steps are done in the software domain.

Despite the other security vulnerabilities in this section, cracking of the beacon needs physical access to the beacon device. So, even if you have secured your beacon stack to remote attacks, you might suffer from this attack.

Well, how can an attacker crack beacons? S/he comes to the beacon point physically and removes the device from its environment, for example, removes from the wall. Afterward, opens the beacon up, means that attacker accesses to its circuitry. S/he can probe the beacon's memory and obtain all sensitive information like the beacon password. Even if your beacon device resists the remote attacks, the attacker directly captures it by accessing hardware at that point.

In this attack type, the attacker needs a lot of work. Thus, the possibility of occurring this attack is very low when compared to others. However, beacon devices sometimes used in critical systems. For example, these IoT devices can be used in the door lock system. In this case, if an attacker can reach the beacon controlling the door physically and probe the beacon memory, s/he can put his or her device mac address to the authorized list in the beacon memory. Therefore, if your beacon somehow controls a vital asset in your system like door example or there are concerns about human safety like our case study VipAssistant(see in section 3.1), developers of the system have to consider this cracking attack.

Finally, we would like to mention other problems when a malevolent person comes near the beacon device physically since this subsection is the only one that argues a physical attack type. Other issues affecting the system reliability and safety are beacon thievery, vandalistic acts to beacon device, and intentionally replacing the beacon position. Although these kinds of actions influence the system, they are not directly related to IoT security. That's why they were neither detailly examined nor explained.

In section 3.2.3, solution suggestions and analyzes of the current solution techniques were explained.

1.2.4 Denial of Service (DoS)

Denial-of-service attack (DoS attack) is a cyber-attack in which the attacker try to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with an extreme number of requests per second in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. In a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the target originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source.

BLE DoS attacks are no different from typical DoS attacks on a typical network. Attackers try to load the beacons with a high number of bluetooth packets to make the system unresponsive. The simplest form of this exploit use the following strategy: The attacker sends a connect request that stalls the communication on a slave device (target device) until the Link Layer Connection Supervision timer expires. This approach exploits a vulnerability on the Bluetooth Low Energy Protocol.

A novel way of DoS attacks on the BLE devices is quite different from the DoS attacks on other networks. Since BLE devices work with batteries instead of being plugged to an electricity source, Battery Drain DoS attacks can be easily applied to them. Strategy is similar again, send bluetooth packets until the device has no battery left to operate.

BLE chips are not particularly designed to be able to use the whitelisting methods and resolve the private address in real time This creates a severe weakness on the BLE Beacons and makes DoS attacks one of the most dangerous attacks that can be executed on a BLE device.

Unfortunately, DoS attacks are not always even done by a malicious part. A case study [5] showed that several bugs in the protocol implementation of iBeacons resulted in unavailable beacons.

Chapter 2

Related Work

Beacon security research is a fairly new area for researchers and developers. All the related work is from recent years and these work are heavily focusing on specific types of beacon vulnerabilities & how to exploit these vulnerabilities. Since exploitation of the vulnerabilities are also fairly new, defense mechanisms for possible attack schemes are yet to be developed.

The most significant part for developing defensive mechanisms based on found vulnerabilities are beacon vendors. Vendor companies try to provide both more secure hardware for their beacons and more secure software development kits for the product users. They equip beacons with security mechanisms such as Secure Shuffling, Secure Communication and Software Lock to provide more secure products from scratch.

In a nutshell, we benefit from papers [6] [7] while detecting beacon vulnerabilities and vendor sites while examining possible solutions. Also, we try to come up with new solutions if the current solution is not sufficient or applicable for our case study, the VipAssistant project.

Chapter 3

Proposed Approach

Before going deeper with the proposed approach to the specified problems with taking the VipAssistant project as our case study on the examination of evaluation of security vulnerabilities of Bluetooth Low Energy Beacons, we need to take a closer brief look into VipAssistant project and its internal mechanics.

3.1 VipAssistant

3.1.1 Overview

VipAssistant (Visually Impaired People Assistant) is a comprehensive project that is designed to overcome Indoor Navigation and Information problems of people and specialized in Visually Impaired Community. To be more specific, it provides an end product to its users that is able to locate, generate on demand shortest paths, navigate and support its clients with many more such functionalities via two special application modes (VIP and non-VIP mode) which are divided according to the platform's target user groups. In order to accomplish this objective, an indoor navigation engine with a world map and web backend support has been designed for the platform of the project along with an accessible and user-friendly interface.

3.1.2 Overall Architecture

VipAssistant has a clearly designed and kind of complex infrastructure behind. The Overall System's Architecture is given in figure below.

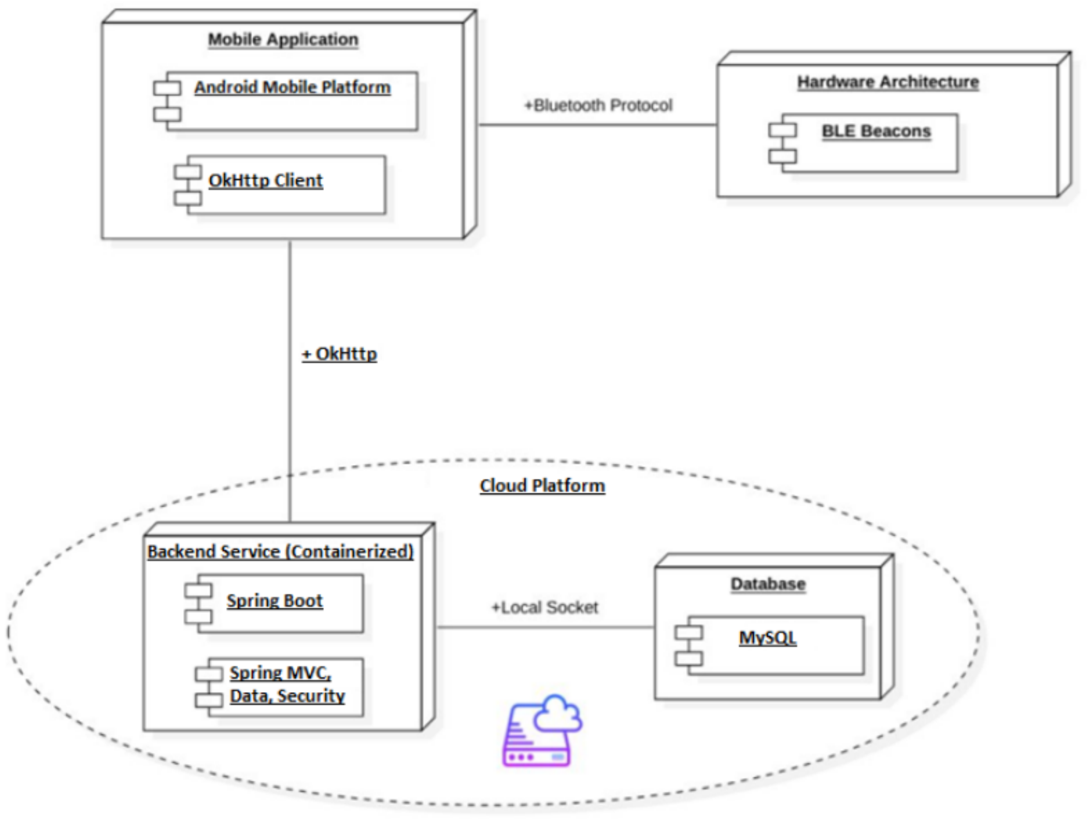


Figure 3.1: VipAssistant Overall Systems Architecture

As can be seen from the Overall Systems Architecture VipAssistant is composed of three main parts. Firstly, a Hardware Infrastructure that consists of decentralized Bluetooth Low Energy Beacons deployed in the buildings which is the main technology behind the real-time location determination module of the VipAssistant. Secondly there exists a Cloud Platform in a part of the architecture which is responsible for providing API and Database support into the functionalities of the platform that needs such requirements, for instance real-time Heatmap Builder module of the VipAssistant is one such end-product functionality that requires such an infrastructure. Finally, Mobile Application is the point where VipAssistant presents all the functionalities of the platform to its users.

3.1.3 Beacons & Security & VipAssistant

As mentioned in 3.1.2 VipAssistant has an Hardware Infrastructure consisting solely on Bluetooth Low Energy Beacons. This Infrastructure in VipAssistant project is used for its Indoor Localization mechanism inside the Indoor Navigation Engine. In the following figure 2 of VipAssistant's High-Level System Architecture, the relation between distinct mechanisms of the system architecture (especially the relation between Mobile Application and Beacon Infrastructure) can be seen easily.

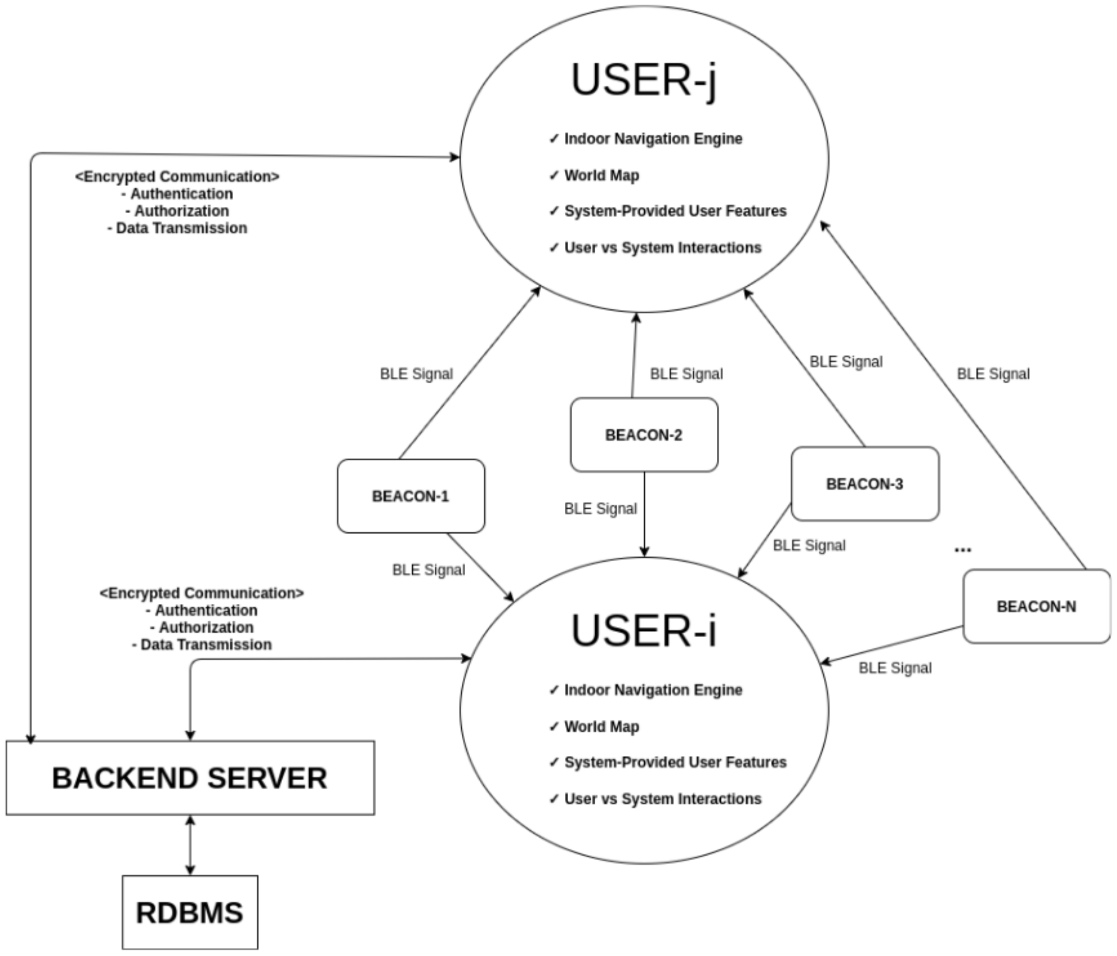

USER-j
✓ Indoor Navigation Engine
✓ World Map
✓ System-Provided User Features
✓ User vs System Interactions
<Encrypted Communication>
- Authentication
- Authorization
- Data Transmission
BLE Signal
BEACON-1
BEACON-2
BEACON-3
...
BEACON-N
USER-i
✓ Indoor Navigation Engine
✓ World Map
✓ System-Provided User Features
✓ User vs System Interactions
<Encrypted Communication>
- Authentication
- Authorization
- Data Transmission
BACKEND SERVER
RDBMS

Figure 3.2: VipAssistant High Level System Architecture

In the Decentralized Beacon Infrastructure of VipAssistant project, beacons are designed as kind of anchors that would continuously notify its surroundings about its very existence with Bluetooth Low Energy (BLE) signals. Then the users of the VipAssistant which have the mobile application could fetch all these signals emitted from the beacons which are nearby via a capture agent programmed on their applications. Afterwards user application is ready for processing all these gathered signals from the beacons to utilize in an algorithm called Multilateration [8] for indoor location calculation purposes. As can be seen, nothing is sent from any kind of application to any of the beacons, only a one-way signal emission is taking place in the architecture.

As can be guessed, due to very nature of the Localization mechanism of VipAssistant in its Indoor Navigation Engine, this project is also vulnerable to several Bluetooth Low Energy exploits mentioned in Problem Statement Part that could result in very catastrophic events for both the users and state of the system as the main targeted user group that VipAssistant specializes on is Visually Impaired Community and the data carried out in between different modules of the system is highly valuable as it contains user whereabouts respectively.

3.2 Solutions

In this part of the report, solution approaches to the problems introduced in the above sections, developed within the context of VipAssistant Project are going to be discussed, regarding the project as a case study and test environment for the proposed solutions. In each of the vulnerability item solutions, first its complication with the VipAssistant Project will be given then the solution against it will be shared.

3.2.1 Spoofing & PiggyBacking

The spoofing vulnerability of the beacons can have very fatal impacts in the VipAssistant architecture as it highly depends on the identifying data, namely the UUID values, advertised from the beacons to be able to resolve the beacon that emits the packet. The reason for resolving the advertising beacon is because for Localization Module inside Indoor Navigation Engine of VipAssistant to work, the system needs to know geo locations and distances of at least three distinct beacons since the Multilateration algorithm that is used for location calculation needs as such. The distance of the beacon is calculated by using the RSSI value that is transmitted with the packet and the measured power of the signal. For the geolocation of the beacon however, VipAssistant uses an internal UUID mapping (i.e. Beacon Table) that is stored in its database on a cloud platform and advertised to the end users of the project.

When such an indoor navigation system that utilizing beacons exists with the insufficient precautions taken against Spoofing Vulnerability of the beacons an attacker may capture the UUIDs of some beacons and clones them to some other advertising beacons that is deployed on somewhere that is completely different, the assumption-based indoor navigation system is going to be impersonated, its users are going to be mislead and the system will be broken eventually. Even worse, say the malicious attacker cloned the beacon UUIDs to another beacons such that when user wants to navigate a safe indoor place such as a classroom s/he will be navigated to somewhere very dangerous instead (say somewhere with high elevation and no precaution against falling down) and as the system's user could potentially be visually impaired, s/he may not be aware of her/his whereabouts and navigate to the dangerous place with relying on the system that s/he uses.

The solution approach against this vulnerability in VipAssistant has been developed using beacon UUID shuffling with cloud-fed seed. The below figure illustrates the architecture.

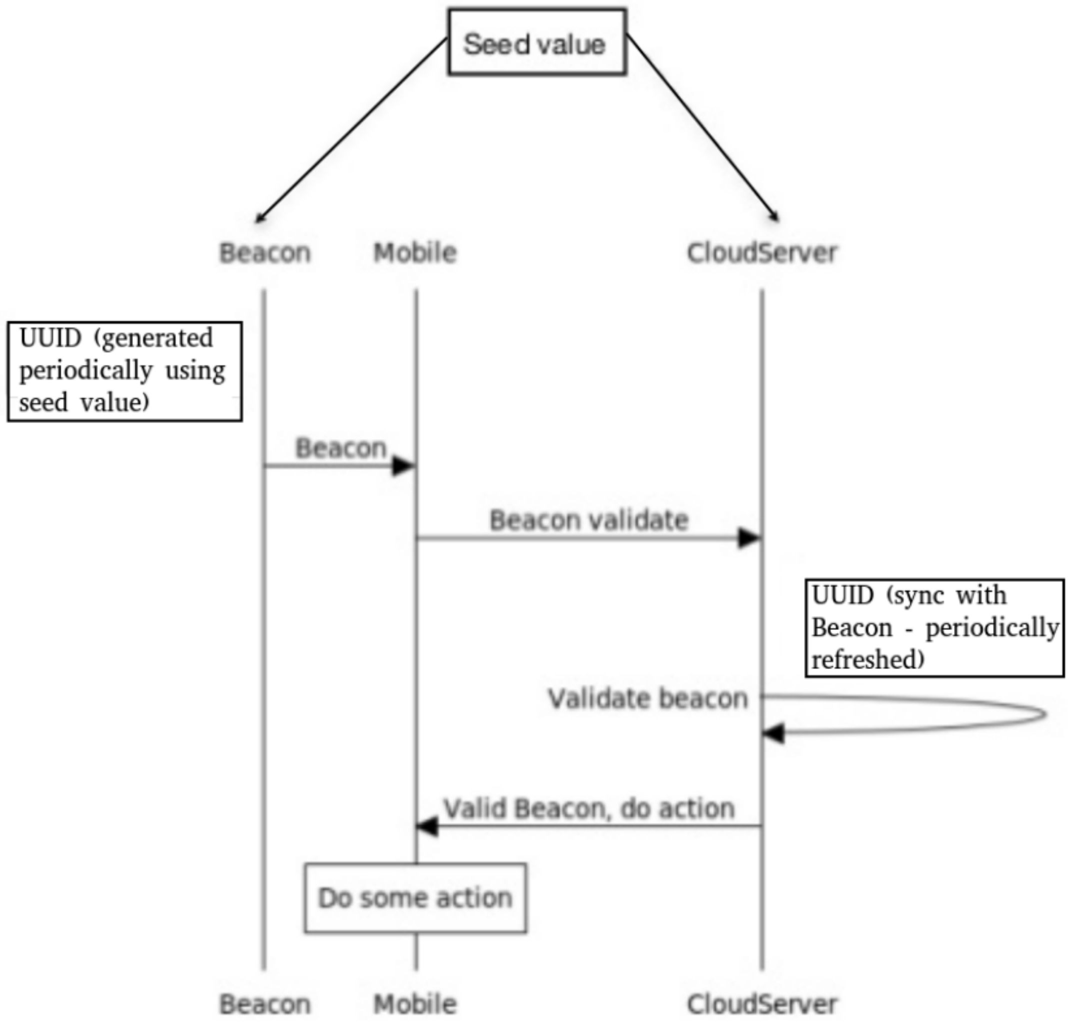


Figure 3.3: Cloud synchronized UUID architecture

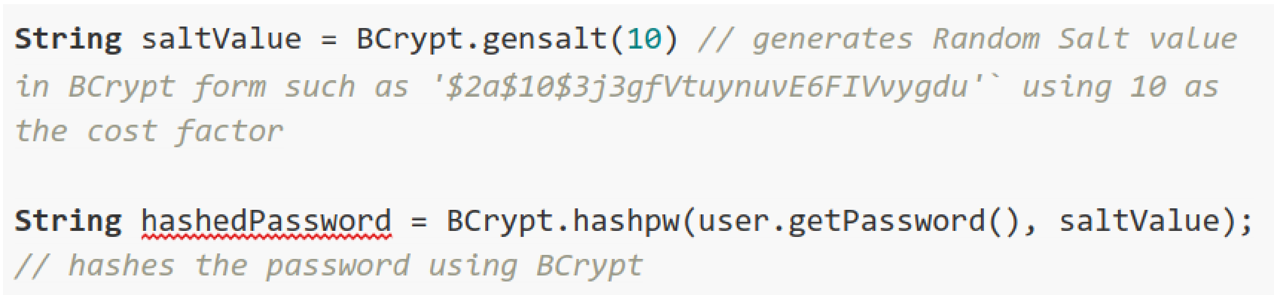
As can be seen in the figure, the solution is based-on changing the individual beacon's identifiers, actually only its UUID, continuously and with that making the sniffer third-party unable to use captured beacon UUIDs statically since they will be changing dynamically in real-time. Thanks to the vendor [9] (Kontakt.io) of the beacons we possess, we are able to change UUIDs of beacons as we wish. To be able to use this prevention against the vulnerability, VipAssistant end users also need to be synchronized with this changing Beacon UUIDs somehow. So in order to keep track of these UUIDs, VipAssistant keeps its records of beacons in its cloud platform as a beacon table which have UUID, geolocation and MAC Address fields for each individual beacon. Cloud platform also periodically (with the same period) shuffles the UUIDs of the beacon records in the database. The initial seed value used for the next UUID generation in Beacons is also kept in cloud platform, which lets the cloud platform generate the same UUIDs in a synchronized fashion and keep track of changing Beacon UUIDs in real-time. After a UUID shuffle in the cloud, end users are notified with the newly changed beacon table and their local beacon table gets updated synchronously as the beacon's UUIDs change. Therefore with this kind of a solution approach, VipAssistant system and users are protected from both spoofing and piggybacking kind of vulnerabilities as the third-party can not depend on the static UUID values advertised by the beacons as they change continuously.

3.2.2 Hijacking

As mentioned in the corresponding Problem Statement part of Hijacking, applications that communicate with beacons are directly vulnerable to this kind of attack since, by default, communication between an application and a beacon device is carried out unencrypted. But for the applications that do not communicate with beacons but rather only gather the packets advertised from the beacons are not directly vulnerable but they may be indirectly vulnerable. VipAssistant Project is also among the applications that indirectly gets affected by this vulnerability.

In the VipAssistant system, end users do not directly communicate with beacons, they gather the advertised packets and, as mentioned earlier, by using these gathered advertisement packets, VipAssistant's Indoor Navigation Engine is able to calculate the end user's location in real-time. The result of this operation is highly valuable, private and sensitive information as the user's location gets calculated in real-time with the consent of the user. On the other hand, there exists other extra functionalities of the VipAssistant platform that is also as much valuable as the location data such as Social Distancing Mode of VipAssistant, which is responsible for creating a safe social distancing radius and notifying users when there is a social distancing violation or someone that were nearby gets infected due to a pandemic such as Covid-19 [10]. If all these valuable information that is available thanks to beacons were to be taken over by any kind of malicious third-party, then lots of huge exploits such as User Profiling and User Tracking is possible for the attackers. Therefore VipAssistant Project also needs to tend to all possible security vulnerabilities that may result in attackers' exploit this highly valuable data generated in the system.

In general there exists no problems in Indoor Navigation or general usage of the VipAssistant application since all the functionality is utilized Internet-free and everything is kept in local on end user devices (except beacon table refresh operations that is mentioned in 3.2.1). But for the platform functionalities that need to use the Internet for communication with cloud platforms needs the highest web security measures that can be taken otherwise sensitive user credentials data or location data and highly private data that can be exploited using Social Distancing Mode such as knowing user's whereabouts, pandemic background and user's nearby list remains insecure. For all these issues, VipAssistant uses the secure version of HTTP [11], namely the HTTPS [12], to ensure all the communication between end users and cloud platform is held encrypted. HTTPS is built on top of HTTP and creates a secure channel over an insecure network with encrypting all the entirety of the underlying HTTP protocol using an added encryption layer of TLS [13] to protect the ongoing traffic. Initially, the SSL certificate for the cloud platform of VipAssistant was to be generated using Certbot [14] and then issued using Let's Encrypt CA [15] but since we did not have sufficient authorization over the DNS records over the domain name of the web server, we instead created a self-signed SSL certificate using Oracle's keytool tool [16] with a key algorithm RSA [17] and keysize of 4096. Also to be able to authenticate and authorize the end users of VipAssistant correctly, an HTTP Basic [18] type of Authentication is developed using Spring Security Framework [19] in Cloud Platform. For password hashing operations BCrypt has been used in both end user applications and in cloud platform. In end user applications, before each request user's credentials gets packaged, his/her password is hashed by using a random salt value. Both operations are carried out using BCrypt in code pieces given on below figure.



```
String saltValue = BCrypt.gensalt(10) // generates Random Salt value
in BCrypt form such as '$2a$10$3j3gfVtuynuvE6FIVvygdu'` using 10 as
the cost factor

String hashedPassword = BCrypt.hashpw(user.getPassword(), saltValue);
// hashes the password using BCrypt
```

Figure 3.4: Cloud synchronized UUID architecture

Then this packaged credentials with BCrypt hashed password is put to the Authorization Header of the HTTP request. After the request is sent to the cloud platform using HTTPS, before accessing the endpoints of the cloud platform that is protected by Spring Security, this request gets intercepted by Spring Security to be able to perform authentication and then authorization checks according to the incoming credentials. After decryption of HTTPS packet, Spring Security opens up the Authorization Header of the request and checks whether the sent hashed password of the user with specified username matches the record in the database. If it does not, it denies the Access by responding with HTTP code UNAUTHORIZED-401. All the passwords in the database are kept with their BCrypt hashed versions which is also maintained by the implementations done using Spring Security Framework. Except register endpoint, all the endpoints are intercepted as such. Thus with these precautions taken in web security of VipAssistant, malicious third-parties that want to exploit valuable sensitive data calculated using beacon infrastructure, by actions such as eavesdropping [20] or Man-in-the-middle (MITM) [21] attack are prevented.

3.2.3 Cracking

As already mentioned in the corresponding problem definition subsection, the VipAssistant system should be reasonably reliable and safe since it is planned to be used by visually impaired people. Thus, we need to examine the system security from the point of cracking attack as well. When we investigate, our case VipAssistant project does not hold sensitive data in the beacon side. For example, beacons do not have authorized lists or control special assets. And from this point, the project can be seen as more secure than the projects whose beacons keep sensitive data. However, these beacons have settings, and one of them is crucial for us, the signal power of the beacon. The reason behind this setting's importance is that the project uses RSSI(received signal strength indicator) value of the incoming signals. As you might have guessed, obtained RSSI values from the beacon are varied if the transmission power levels altered in an unauthorized way. But an attacker can change this setting if s/he crack the beacon and probe the device. Hence, the system's reliability and safety are affected inevitably.

Up to here, the cracking attack effects on the VIPAssistant system is argued. From now on, we discuss solutions.

Firstly, we would like to explain the available solutions. Our beacons' vendor is the Kontakt IO firm, and there is a solution developed by the Kontakt IO for cracking attacks. The method is called "software lock". The beacons we use are Asset Tag beacons. In these beacons, there is a built-in microcontroller Nordic Semiconductor nRF52832. Other beacon devices of the firm generally use Nordic Semiconductor microcontrollers.

If an attacker tries to reach this Nordic chipset, the "software lock" wipes the system's volatile memory. After wiping the volatile memory, only This "software lock" is a proper solution technique for systems' that keep critical data inside the beacon. However, our system does not keep data, so this solution is not really beneficial for the VipAssistant.

Another built-in feature of the Kontakt IO beacons can be used for this task. There is an accelerometer in the Asset Tag beacon. The actual usage of this sensor is detecting an asset's movement by detecting movement in the beacon(beacon is attached to the asset). If you detect movement in the asset, you can trigger actions in the cloud with the help of mobile devices(or any device that can connect to the beacon and internet). This is a useful functionality for asset tracking, and the attackers should move the beacon in order to open it up. However, in this case, although movement detected by the beacon, it cannot be trigger cloud action at the same time since it does not have the internet connection. After then, the attacker tries to crack the beacon, and if s/he successful, beacon memory is wiped out, it doesn't have necessary URL data to trigger cloud actions. A costly solution can be placing internet connection capable devices nearby to the beacon. When there is a movement detected in the beacon, a cloud action is triggered at the same time. This action can remove the beacon's ID from the beacon table in the database, and users will not be influenced by the cracking of a beacon(localization error margin might increase).

A proper solution for the VIPAssistant system is that when a movement detected in the beacon position with the help of the accelerometer sensor, the beacon automatically shuts down itself. This solution not only solves the cracking problem but also addresses the replacement of beacons problem. As we mentioned in the corresponding vulnerability subsection, evil people can take beacons from its place and put it to another point. Also, beacons can fall to the ground. In this case, if the beacon continues to emit BLE signals, system reliability will be affected again. But, if this solution is applied, this problem also will be solved.

Lastly, we have to say that protecting beacons from physical effects is hard. Because there are evil people everywhere, and they can damage the system physically. As we mentioned, these people can steal or break beacons. But, a simple solution can be camouflage. If the beacon can camouflage on the wall, malignant people generally cannot see the beacon. Furthermore, the VipAssistant system can be used in big indoor environments such as airports, shopping malls and hospitals. These vandalistic acts cannot be taken easily because there are always cameras and security guards in these kinds of places.

3.2.4 Denial Of Service (DOS)

The effects of a possible DoS for VIP Assistant is just like any other system of network. If the BLE Beacons become unavailable, all kinds of functionality becomes unavailable. Furthermore, the beacons' batteries would drain and a physical replacement for batteries would be required. The VipAssistant system has had several vulnerabilities against DoS attacks. An intruder would easily drop beacons one by one or as a whole to damage the infrastructure and availability of the system. This situation would likely cause very high costs in terms of effort and finance to make systems available again. In order to develop a solution, at first we have searched for the mechanisms that beacon vendors provide. The result of our study showed that there is no prevention mechanism that vendors provide. Also the Secure Shuffling mechanism does not prevent possible DoS attacks. The idea of removing DoS vulnerabilities branch into two main solutions. First one is whitelisting the trusted users, and the second one is self aware monitoring by user data. In general, whitelisting on BLE based devices is not possible on the device itself since BLE chips are not suitable for whitelisting operations. Diagram representation of the algorithm (called cracking algorithm) for such a solution is given in figure 5 below.

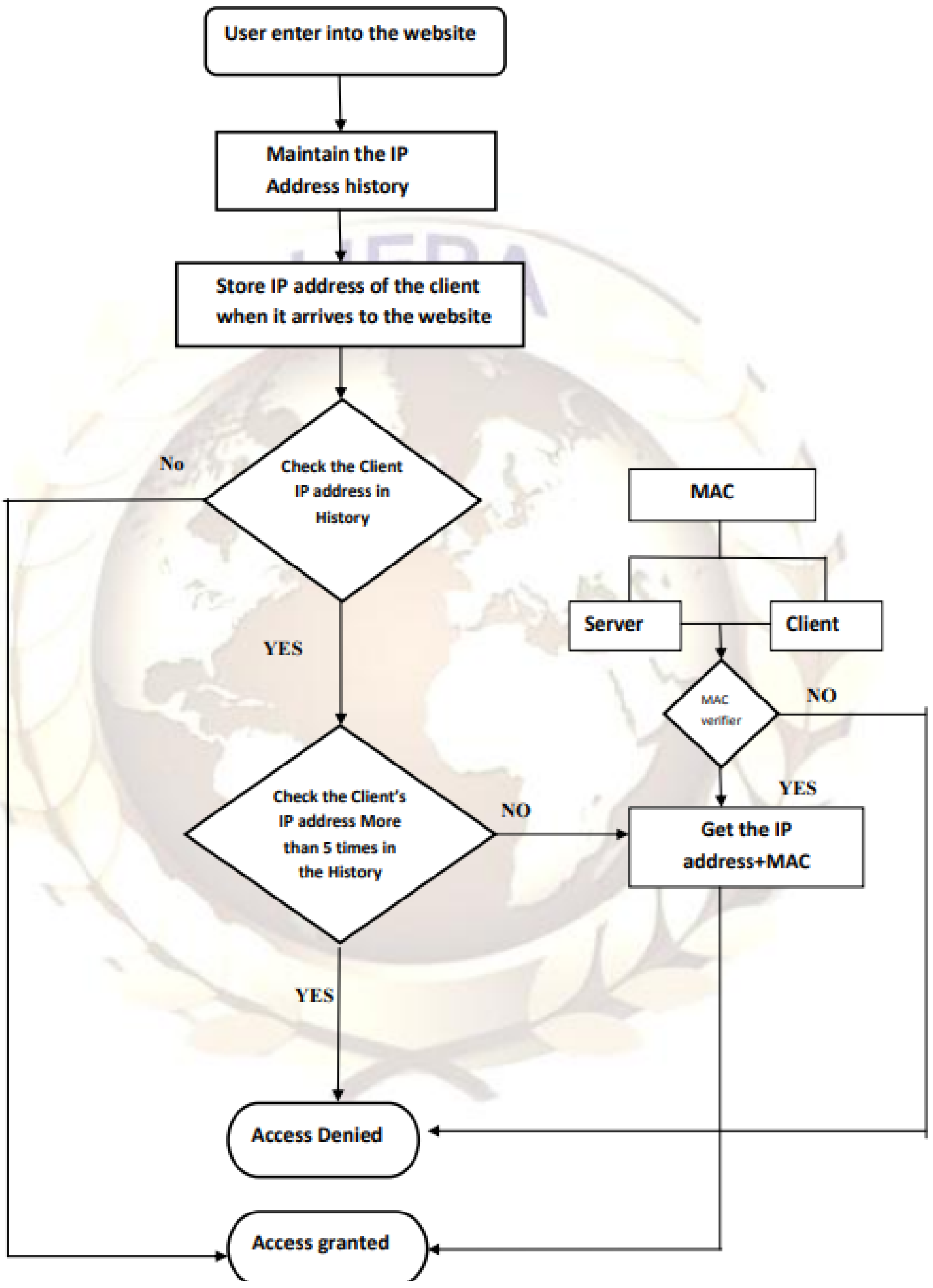

User enter into the website
Maintain the IP Address history
Store IP address of the client when it arrives to the website
Check the Client IP address in History
No
YES
Check the Client's IP address More than 5 times in the History
NO
YES
MAC
Server
Client
MAC verifier
NO
YES
Get the IP address+MAC
Access Denied
Access granted

Figure 3.5: Cracking algorithm diagram

```
Algorithm 1: Self-aware Monitoring Client Side
initialization;
while true do
    badBeaconCount=0;
    badBeaconList = [];
    for every nearby beacon do
        if no response or beacon RSSSI too low then
            badbeaconCount++;
            badBeaconList.add(beacon.UUID)
        end
    end
    if badBeaconCount > 0 then
        reportBadBeaconsToServer(badBeaconList);
        reportUnavailabilityToUser;
    else
        reportSuccessToServer;
        reportSuccessToUser;
    end
end
```

Figure 3.6: Self-aware Monitoring Client side pseudocode

```
Algorithm 2: Self-aware Monitoring Server Side
initialization;
unavailableBeacons = {};
brokenBeacons = [];
while reportFromClient do
    if reportFromClient == SUCCESS then
        continue;
    else
        bbl = reportFromClient.badBeaconList;
        for beaconUUID in bbl do
            if beaconUUID in unavailableBeacons and
             unavailableBeacons[beaconUUID] != -1 then
                unavailableBeacons.beaconUUID += 1;
            else
                unavailableBeacons.beaconUUID = 1;
            end
        end
        for prevUUID in unavailableBeacons do
            if prevUUID not in badBeaconList then
                unavailableBeacons.remove(prevUUID);
                continue;
            end
            if unavailableBeacons[prevUUID] > 10 then
                brokenBeacons.add(prevUUID);
                unavailableBeacons[prevUUID] = -1;
            end
        end
    end
end
```

Figure 3.7: Self-aware Monitoring Server side pseudocode

Chapter 4

Experiment Results

4.1 Spoofing & PiggyBacking

For the experiment parts of these similar Beacon Vulnerabilities that make also VipAssistant vulnerable regarding the issues they originate, we were able to perform experiments and trials for only the Piggybacking one. After the solution approach mentioned in 3.2.1 which tends to both of these vulnerabilities in VipAssistant, a designed system is tested to check if there exists any bugs or faults in newly implemented code pieces in both end user applications and cloud platform. Checking operation has been done by validating that UUIDs of the possessed beacon devices are shuffled periodically (with a period of 10 seconds) and their UUIDs' track is also synchronously kept via the mechanism that refreshes beacon UUIDs starting with the same initial seed value and notifies end users in cloud platform.

For spoofing we could not make any experiments due to our lack of ability to clone an arbitrary UUID to another Beacon device directly. We were able to provide a seed value for the shuffling of the UUIDs thanks to the vendor of the beacons we possess but for a direct UUID cloning, we do not have the type of the beacon of the vendor that allows such direct operations.

For piggybacking we made our experiment by developing a simple arbitrary mobile application that also utilizes beacon advertisement packets to leverage its own services which kind of represent the third party that applies piggybacking on the beacon devices. The application simply runs in background when triggered and listens for some specific beacon UUIDs. When the application captures the advertised UUID it was searching for, it notifies the user (by pushing a notification in the background), telling that user is now nearby a specific place (since the UUID advertised by the beacon is anchored to some constant place). In the demo application, it is assumed that the third-party did know the initial UUID that belongs to the target beacon earlier, therefore this known UUID is given statically in the demo application's code.

For the experiment, this demo application and VipAssistant are both installed in the same testing mobile device and the tester first triggered the demo app then started to use VipAssistant's end user application. While navigating with VipAssistant, the tester approached the target beacon before the initial UUID refresh period has passed, a push notification has appeared on the user's mobile device from the demo application (represents the abused case). But afterwards when user got out of the perimeter of the target beacon and then got back again, the demo application did not seem to push another notification saying that user is near the targeted beacon, it instead failed to find such UUID and it will not be able to find another again thanks to the solution developed in VipAssistant, since beacon's UUID has been shuffled with the same seed used in its cloud platform and end user application has been notified with the changed value after the one refresh period has passed.

4.2 Hijacking

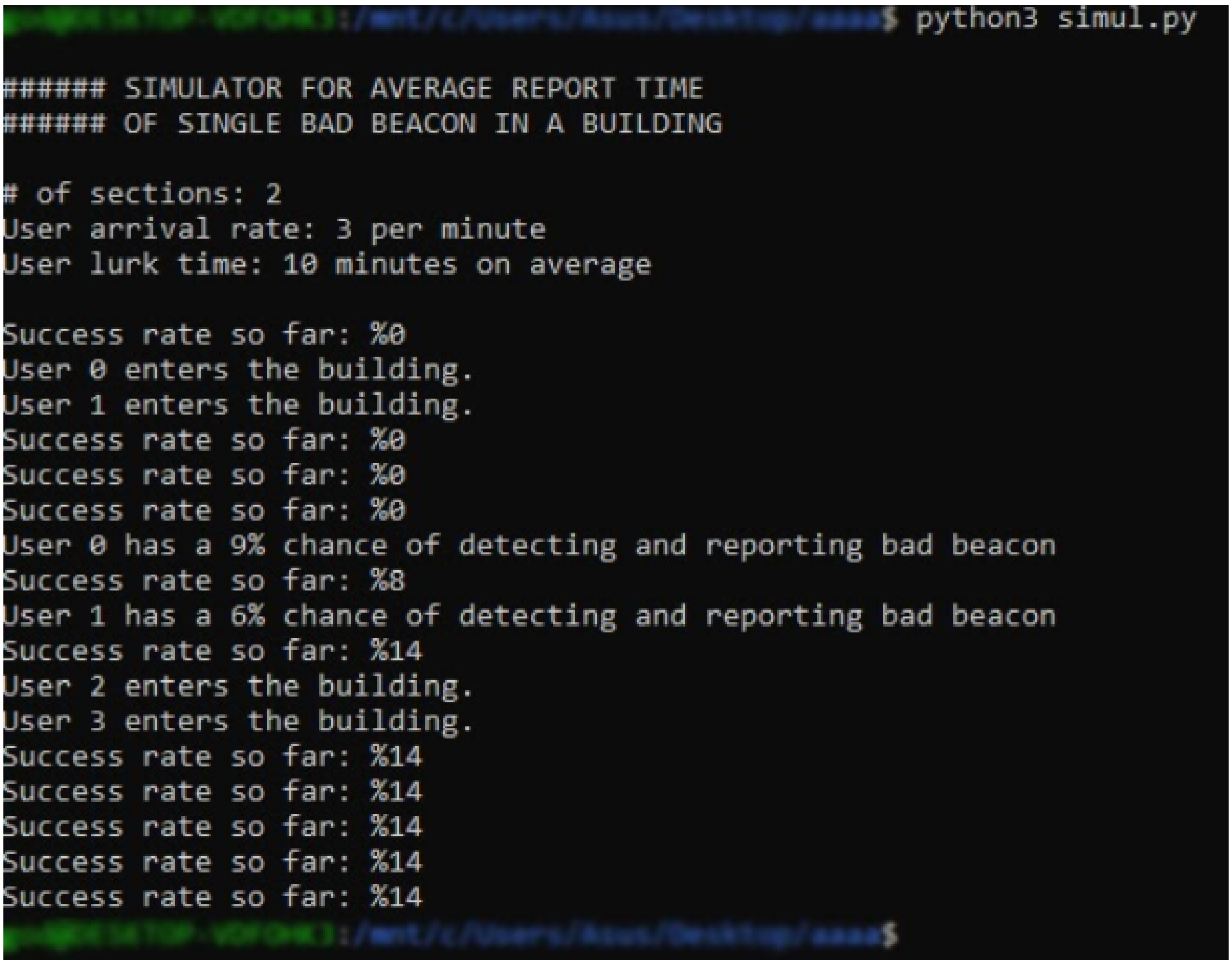
For the solution approach that is developed against Hijacking vulnerability in VipAssistant (discussed in 3.2.2) we did not perform any additional experiment effort regarding beacon device configurations or implementations since VipAssistant was not a direct victim of this vulnerability but rather it is indirectly vulnerable to this issue due to the high value data that is generated using beacon infrastructure such as real-time location information, was being carried across the web (to the cloud platform of VipAssistant more precisely) for the platform functionalities that require Internet connection. For the experiments regarding the implementation done against the web security vulnerabilities of VipAssistant we simply relied on the proven security measures of the technologies we used, namely the HTTPS which is built on top of the HTTP using an added encryption layer that consists of TLS/SSL layer. For the Authentication and Authorization part made numerous arbitrary both unauthenticated and unauthorized requests to our endpoints that are secured by the HTTP Basic Authentication powered by Spring Security Framework, using the Postman [22] tool. In all of them, the server responded to us with HTTP-401 response code. For the requests that has their credentials set correctly, we got HTTP-200 responses from the server.

4.3 Cracking

Since VipAssistant does not keep any data on the beacons, experiments for this part were not needed. Thus, we did not have any experiment for cracking solutions. Also, since we cannot open the beacons up or change their vendor's software, we cannot implement our proposed solution approach.

4.4 Denial Of Service (DOS)

Whole experiment session for our DoS (or any unavailability issue in general) protection was run by a simulation code written in Python. The reason for this was that since the application is not on commercial use yet, we do not have real data about user bad beacon report statistics. Thus, such an experiment is only viable through a simulated environment for this work. The main purpose of the simulator is to detect under how much time a bad beacon can be detected. The simulation environment is assumed to be a building or place consisting of n sections. Average user lurk time is determined as m minutes on average inside the place and there is an average time between entrance of each user as p minutes visitors incoming to the place. These parameters can be found on the source can and can be changed easily to simulate different environments. Each user has a random chance of discovering a bad beacon and reporting it to the server, thus adding it to the suspicious beacons list for the implementations given on the solutions segment of the documentation. The percentage possibility that a user can detect a bad beacon is randomly decided between a predetermined interval and this interval would be calculated based on building structure, beacon frequency of the place, crowd and other factors that can affect beacon signal spread. Sample output of the simulation env is given below.



```
[redacted]$ python3 simul.py

###### SIMULATOR FOR AVERAGE REPORT TIME
###### OF SINGLE BAD BEACON IN A BUILDING

# of sections: 2
User arrival rate: 3 per minute
User lurk time: 10 minutes on average

Success rate so far: %0
User 0 enters the building.
User 1 enters the building.
Success rate so far: %0
Success rate so far: %0
Success rate so far: %0
User 0 has a 9% chance of detecting and reporting bad beacon
Success rate so far: %8
User 1 has a 6% chance of detecting and reporting bad beacon
Success rate so far: %14
User 2 enters the building.
User 3 enters the building.
Success rate so far: %14
Success rate so far: %14
Success rate so far: %14
Success rate so far: %14
Success rate so far: %14
[redacted]
```

Figure 4.1: Experiment results of a faulty beacon detection run

The simulation was run with different parameters 30 times each. Graphic representation of the parameter versus success rate statistics can be found below.

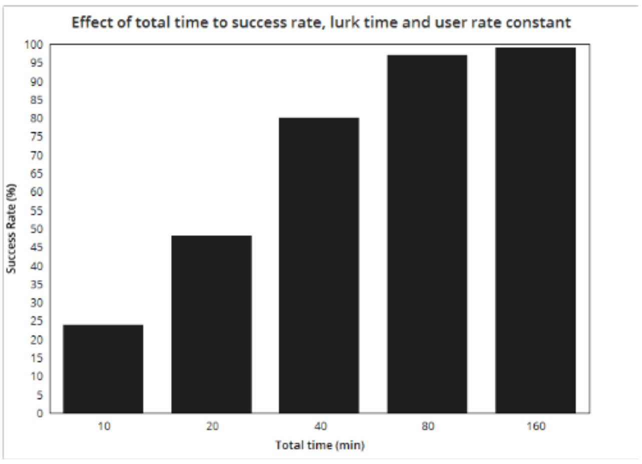


Figure 4.2: Chart of “Total Time x Success Rate”

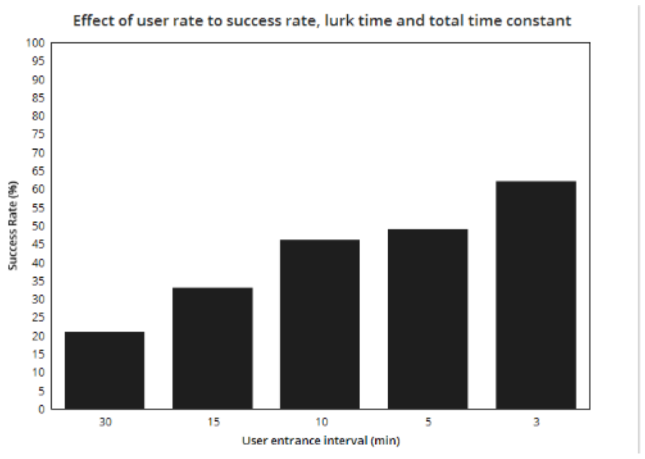


Figure 4.3: Chart of “User interval x Success Rate”

Chapter 5

Conclusion

Beacon technology continues to expand its reach. Those who plan to utilize this technology within cities, stores, or other businesses should take the time to understand and test the complete ecosystem of the product they are looking to deploy in order to maintain a safe and secure environment for all users.

This work was a study over well-known beacon security vulnerabilities, and possible solutions for most common attacks that exploit these vulnerabilities with a case study on VIP Assistant -our own beacon based navigation application-. Experiment results for developed or simulated solutions were shared and analyzed.

Chapter 6

References

[1] “Bluetooth low energy,” 2020. [Online]. Available: https://www.wikiwand.com/en/Bluetooth_Low_Energy

[2] “Official vipassistant website,” 2020. [Online]. Available: http://senior.ceng.metu.edu.tr/2020/vipassistant/

[3] “Received signal strengthindication,” 2020. [Online]. Available: https://www.wikiwand.com/en/Received_signal_strength_indication

[4] “Universally unique identifier,” 2020. [Online]. Available: https://www.wikiwand.com/en/Universally_unique_identifier

[5] A. Sathyabama, C. Nalayini, and S. Priyadharshini, “Earnest access of divulging and aversion of ddos attack,” *International Journal of Computer Applications*, vol. 123, pp. 18–21, 08 2015.

[6] C. Kolias, L. Copi, F. Zhang, and A. Stavrou, “Breaking ble beacons for fun but mostly profit,” 04 2017, pp. 1–6.

[7] K. E. Jeon, J. She, P. Soonsawad, and P. Ng, “Ble beacons for internet of things applications: Survey, challenges and opportunities,” *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 04 2018.

[8] “Multilateration,” 2020. [Online]. Available: https://www.wikiwand.com/en/Multilateration

[9] “Kontakt io official website,” 2020. [Online]. Available: https://kontakt.io/

[10] “Covid19 disease,” 2020. [Online]. Available: https://www.wikiwand.com/en/Coronavirus_disease_2019

[11] “Http protocol,” 2020. [Online]. Available: https://www.wikiwand.com/en/Hypertext_Transfer_Protocol

[12] “Https,” 2020. [Online]. Available: https://www.wikiwand.com/en/HTTPS

[13] “Transport layer security,” 2020. [Online]. Available: https://www.wikiwand.com/tr/Transport_Layer_Security

[14] “Certbot open source software tool,” 2020. [Online]. Available: https://certbot.eff.org/

[15] “Lets encrypt,” 2020. [Online]. Available: https://letsencrypt.org/

[16] “Java 8 se documents,” 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

[17] “Rsa encryption system,” 2020. [Online]. Available: https://www.wikiwand.com/en/RSA_(cryptosystem)

[18] “Basic access authentication,” 2020. [Online]. Available: https://www.wikiwand.com/en/Basic_access_authentication

[19] “Spring security,” 2020. [Online]. Available: https://spring.io/projects/spring-security

[20] “Eavesdropping,” 2020. [Online]. Available: https://www.wikiwand.com/en/Eavesdropping

[21] “Man in the middle attack,” 2020. [Online]. Available: https://www.wikiwand.com/en/Man-in-the-middle_attack

[22] “Postman,” 2020. [Online]. Available: https://www.postman.com/