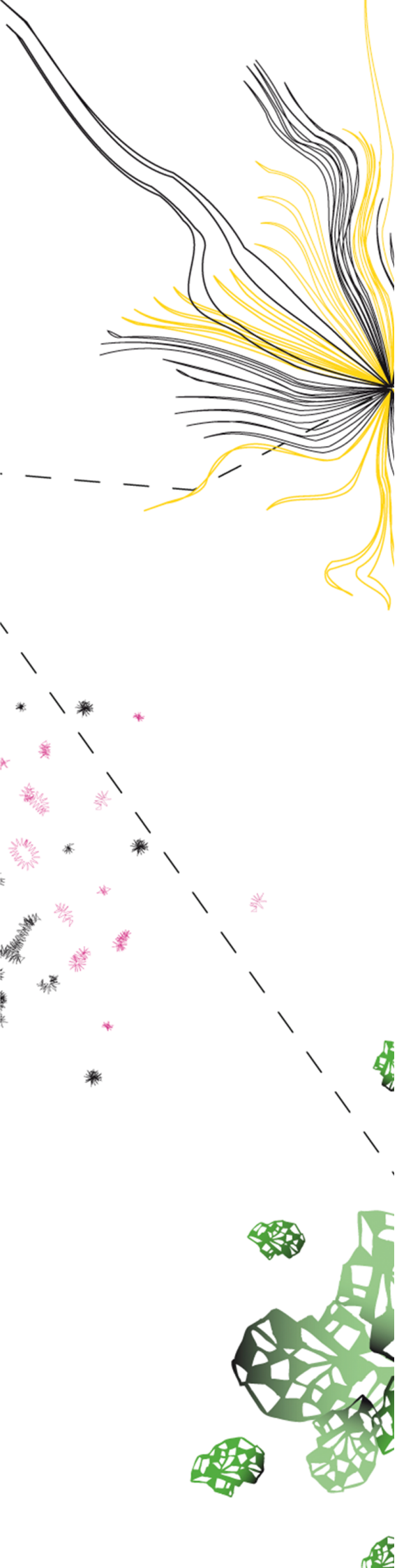

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

A threat model analysis of audio recording on mobile health care applications

Jochem Harmes

June 4, 2020

ChipSoft

Supervisors:
Prof. Dr. M. Stoelinga
Dr. Ing. E. Tews
Dr. T. van Dijk
Ing. J. Deege

Abstract

Mobile devices are getting used more and more in healthcare. In this report, we do a threat model analysis of recording audio on mobile devices and try to find out how attackers can obtain the privacy-sensitive audio data. We use multiple threat model methods, including STRIDE, attacker profiles, and attack trees. STRIDE is used to get an overview of vulnerabilities and threats and allowed to find 19 initial threats. We then define seven attacker profiles, found from literature and brainstorming sessions with experts. We select two of these profiles, the insider and criminals profiles, to create attack trees and find attack scenarios. Then we grade the nodes of the trees with an effort value to find which attack scenarios are most likely to happen. Last, we propose mitigations to counter these scenarios and try to see how they affect the probability. We found in both attack trees, that our mitigations would make it three times as hard to obtain the audio data. With the results, we choose three mitigations ChipSoft can apply.

ACKNOWLEDGEMENTS

I would like to thank my supervisors for their time and help. A lot of thanks to Prof. Dr. M. Stoelinga for her advice and bringing structure in the report, Dr. Ing. E. Tews for his expertise in security and ideas, Dr. T. van Dijk for helping me improve my writing and Ing. J. Deege for helping me day to day at Chipsoft. I would also like to thank Sander, Jordi, and Joris for their input and feedback on behalf of the security team from Chipsoft and Björn and Frank for their input on attacker profiles. Last, I would like to thank the multimedia team at Chipsoft for supporting me.

CONTENTS

PART I

OUTLINE

CHAPTER 1

INTRODUCTION

The mobile device is being used more and more in our daily lives. Not only are these devices used for our personal lives, but also in the workplace. Healthcare professionals can benefit from using mobile devices, including increased efficiency and enhanced productivity[1]. One company producing mobile apps for healthcare professionals is ChipSoft. They are the biggest provider of electronic medical records (EMR) software in the Netherlands[2].

Their leading software is called 'Healthcare Information X-change' or HiX[3]. At its core, it is an EMR system, but it also has a lot of extra modules. Some modules allow connectivity to other hospital devices, like heart monitors, others help to administer medication. HiX runs on desktops and laptops in a Windows environment. For some users, this can be cumbersome, because they do not always have direct access to a computer, like nurses and doctors doing their patient route. To fill this need, ChipSoft created multiple mobile apps to supplement (but not replace) the desktop system. One of those apps is for medical specialists. It allows them to see patient information on their mobile devices and to enter and save patient notes. In this app, doctors would like to see a new feature: the ability to record audio for their patients. Mainly they want to record summaries of consultations they had with their patients. Doctor consultations can be complicated and emotional for a patient, and he or she may not keep track of what the doctor says. These summaries would allow patients to listen to the doctor's points again at their convenience. In the future, this feature could be extended with speech-to-text software, so doctors can take notes by talking to their phones.

When creating software used in a hospital, developers should take extra security concerns into account. Most handled data will be sensitive information about patients and should be protected from unauthorized people. The software should also conform to (local) privacy laws and, most of the time, ISO standards before being allowed in a hospital. With these constraints, it should be known how adversaries could attack the software and what the consequences of these

attacks are. This knowledge can then be used to reason about how secure the software is and how it can be made more secure. The process of finding exploits and vulnerabilities in software is called threat modeling.

In this report, we apply threat modeling techniques to learn if implementing a secure audio-recording feature is feasible. In the first part, we will define research questions (chapter 2) and feature requirements (chapter 3). We conclude with chapter 4, where we look into threat model analysis methodologies, tools, and related works. In the second part, we apply multiple threat modeling techniques to answer the research questions. In chapter 5, we look into STRIDE to get an overview of possible vulnerabilities. Chapter 6 contains an analysis of possible attacker profiles. These techniques are combined in the next part 'Attack trees & Mitigations', where we create attack trees for two attack profiles: the insider (chapter 8) and the criminals (chapter 9). In these chapters, we also define mitigations and see how they affect the attack trees. In the last part, we discuss the results of the analysis (chapter 10), conclude (chapter 11), and look where to go from here (chapter 12).

CHAPTER 2

GOALS & METHODOLOGY

In the upcoming sections, we determine the goals of this report and which methodologies to use. We use questions to define our goals. We start by creating the main question. Then we define background and research questions to help answer the main question. For each of these subquestions, we think of methods to answer them. Table 2.1 shows an overview of the subquestions and the used methodologies. The questions define the scope and structure of this report.

2.1 MAIN QUESTION

Before researching the topic of secure audio recording on mobile devices, we define goals and how to reach them. The audio recorded contains sensitive patient data and should be protected. If not protected, the data could easily be used by people with bad intentions (i.e., attackers). For example, they could use the data to blackmail a (famous) patient, doctor, hospital, or ChipSoft. It is also possible criminals will try to obtain this data to sell to the highest bidder. Leaked data will also break the confidentiality between a doctor and a patient. This thesis will try to find out if this data can be protected and, if so, how. The main question therefore is:

> *How can a mobile app that records privacy-sensitive audio prevent attackers from obtaining said audio?*

This question should give ChipSoft an idea if it is feasible to record audio safely on a mobile device. If so, this report can be a starting point for the implementation or further research. The report will also help ChipSoft to comply with standards and regulations, as discussed in section 4.3.

From an academic point of view, it is interesting to know how safe to use mobile devices are in the medical field, especially in a world where more and

more devices are used in the medical world. The report will also show how good theory can be applied in practice.

We defined a multitude of sub-questions, which consist of background questions and research questions. The background questions help understand the context of the research. The research questions help answer the main question. Table 2.1 shows a summary of used methodologies for each question. The upcoming sections will go into detail of each question.

2.2 BACKGROUND QUESTIONS

BQ1 *What are the requirements for an audio recording feature in the specialist app?*

Requirements need to be defined first to answer the main question, as they will define the scope of this report. The requirements should be restricted to (basic) functional and security requirements. Requirements will be elicited from and by brainstorming with ChipSoft and discussed in chapter 3.

BQ2 *What are existing methods to find possible threats and attack scenarios?*

Answering this question will help to pick the right methodology and tools to use for our analysis. It will also help to better understand the context of the problem by finding related work. A literature study is performed to find the answer. Tools like Google Scholar[4] and Scopus[5] are used to find papers concerning this subject. Other information, like documentation of tools and attack libraries, is found by searching on the internet. This question will be discussed in chapter 4.

2.3 RESEARCH QUESTIONS

RQ1 *What are possible vulnerabilities and threats for an audio recording feature in a mobile app?*

This question will give a good overview of the system and were possible vulnerabilities may lie, which will help answer research question RQ3, because these vulnerabilities will define what scenarios are possible. The STRIDE methodology[6], discussed in more detail in chapter 4, is used to answer this question.

QUESTION		METHODOLOGY	CHAPTER
BQ1	Requirements	Elicit from and brainstorming with ChipSoft.	3
BQ2	Literature	Literature study, search on internet	4
RQ1	Threats & vulnerabilities	STRIDE methodology, Literature study, manuals from Android and iOS	5
RQ2	Attacker profiles	Use result BQ2, literature study, brainstorming with employees ChipSoft	6
RQ3	Attack scenarios	Use result RQ1, RQ2 and attack trees	8, 9
RQ4	Attack scenarios probability & impact	Attack trees	8, 9
RQ5	Attack scenarios mitigation	Use result RQ1, brainstorming, literature study	8, 9

Table 2.1: Overview of methods used to answer each research questions.

STRIDE gives us a global overview of vulnerabilities without going into too much (unneeded) detail. The threats can be found in chapter 5. In this chapter we also discuss some existing methods to mitigate found threats, to put them in context. These mitigations are also used to answer research question RQ5. Tools like Google Scholar[4] and Scopus[5] are used to find mitigations. Other valuable resources are the documentations from Android[7] and iOS[8].

RQ2 *What are possible attacker profiles?*

Attacker profiles can be used to find attack scenarios. They will define what skill and resources an attacker might have and how the attacker may attack the system. Literature found with background question BQ2, and brainstorming with ChipSoft employees will be used to find these profiles. They can be found in chapter 6.

RQ3 *What are possible attack scenarios?*

Possible attack scenarios will help understand how adversaries could obtain the audio data. Two attacker profiles from research question RQ2 are selected to create two attack trees. The attack trees, explained in section 4.1.2, will show what scenarios are possible. The results from research question RQ1 will help to create attack trees. By knowing the vulnerabilities, it is easier to think of attack scenarios and model them into an attack tree. The attack trees can be found in the third part of this thesis.

RQ4 *What are the probabilities of attack scenarios?*

Knowing what attack scenarios are most likely and their impact will decide whether the audio recording feature can be made safe enough. Maybe it is not possible to defend against an attack scenario, but is the risk so low, that ChipSoft decides to take that risk. So-called effort values are assigned to the nodes from the attack trees of research question RQ3. These values will give an insight into the probabilities of scenarios happening and can be found in the third part of this thesis.

RQ5 *How can the attack scenarios from research question RQ4 be mitigated?*

We suggest mitigations to lower the probability of attack scenarios happening to answer this question. The attack trees can be used to define the most effective mitigations by addressing weak spots. Mitigations are taken from literature, brainstorming, and the results from research question RQ1. We also try to calculate the effect of mitigations on the probability of attack scenarios. Calculating is done by updating the attack trees with new nodes and effort values, which simulate the mitigation being applied. The mitigations can be found in the same chapters as the attack trees, which are in the third part of this report.

2.4 CONCLUSION

In this chapter, we defined the main question, background questions, and research questions. We use these questions to define the structure of our report. The two background questions will help understand the context of the research, and the five research questions will help to answer the main question. We also explained which methods will be used to answer each of the questions, and table 2.1 shows in which chapter this happens. In the next chapter, we will answer the first background question by specifying the requirements.

CHAPTER 3

REQUIREMENTS

We will use this chapter to specify a (minimum) set of requirements for an audio recording feature. We elicit the requirements by looking at eight stakeholders and talking with employees of ChipSoft. First, we discuss the stakeholders, then we look at functional requirements, and lastly, we mention security requirements. Table 3.1 lists all requirements and which stakeholders they affect directly. These requirements determine the scope of the rest of this report. This chapter will answer our first background question BQ1, mentioned in the previous chapter:

> *What are the requirements for an audio recording feature in the specialist app?*

3.1 STAKEHOLDERS

Before requirements are defined, it is useful to see which parties have interests in creating a new audio recording feature. Below the main stakeholders and their interests are discussed. They can be divided into three groups: company (S1 to S3), hospital (S4 to S6), and university (S7 and S8) stakeholders. Some stakeholders have more interest in the process of the threat analysis, like the university stakeholders. Others are only interested in the feature itself, like the hospital stakeholders.

S1 *ChipSoft*

As a company, ChipSoft will want to make sure the feature is secure. A data leak could give much negative publicity and will cost the company money. Not only the confidentiality of data, but integrity is essential too. Lack of integrity may result in harming the patient, which leads to bad publicity. ChipSoft also needs to show that they researched security risks to comply with the standards

mentioned in section 4.3. The thesis should help to reach these goals, and the results should be useful for other features and projects across the company.

S2 *Developers*

The developers will benefit from secure, maintainable, and quality code. The thesis should help developers design this code. Developers would benefit most if it is clear what the next steps are after this report. The report should also give them confidence that a thorough security analysis is done. Also, like stakeholder ChipSoft, the reusability of the results is important for them.

S3 *Implementation and Support*

The implementation and support department from ChipSoft helps hospitals to install and use the company's software. They will benefit from a stable implementation, which is easy to use by the end-users. Stable software helps them ensure happy customers.

S4 *The hospital(s)*

Hospitals will use the software and they have to follow laws and regulations. As with ChipSoft, one of their priorities is to prevent data leaks and integrity of data. Other priorities are that the software saves money and improves the quality of care.

S5 *The medical specialist(s)*

The medical specialists will use the new audio recording feature. Their interests are that it is easy to use and that they can be sure their recordings get saved with the right patient. If recording a conversation is hard or unreliable, doctors probably will not use it.

S6 *The patient(s)*

Patients are also concerned with this new feature as the recordings will contain personal information about them. It is in their interest that the audio records are only available to those who are allowed to see their medical records. The availability of the records will also be important for them, but the focus in this report is on recording the audio.

S7 *University of Twente*

The university does not have a direct interest in the feature itself. Their interest lies in the academic value of creating the feature. Important is that this process reveals new information, which can be reused in other projects. They also need to see that the graduate can do this kind of project.

S8 *The graduate*

For the graduate, the main interest will be to complete the final project successfully. This interest can be supplemented with personal motivation, like discovering unknown solutions.

3.2 REQUIREMENTS

We use the stakeholders, the description of the feature in chapter 1, and input from ChipSoft to define requirements. First, we consider functional requirements, then security requirements. We try to keep the functional requirements simple to keep the focus on security. We will not discuss other types of requirements as they are out of scope and not crucial for this report. Table 3.1 shows an overview of all requirements and affected stakeholders.

3.2.1 FUNCTIONAL REQUIREMENTS

R1 *A medical specialist should be able to record audio on his or her mobile device.*

The doctor should be able to press some button to start to record audio. When the doctor is done, he or she should be able to stop the recording.

R2 *The audio records should be sent to and stored in HiX.*

The audio record should be stored in HiX so it can be easily retrieved. Doctors could then also access the audio record from their desktop.

R3 *Medical specialists should be able to choose the patient for which the audio record is meant.*

The doctor should be able to select a patient before he or she starts recording. Selecting the right patient should prevent mix-ups, where records are coupled to the wrong patient(s).

R4 *Medical specialists should be able to use their personal devices.*

Doctors tend to use their personal devices because they can receive calls about patients they are treating when they are not at work. Having a device from work would mean they would have to take two devices everywhere.

Because of the scope of this project, the functional requirements are kept as simple as possible. Requirements R1 to R3 are the minimal requirements to record audio for a patient. Requirement R4 is interesting because it has a significant impact on security.

3.2.2 SECURITY REQUIREMENTS

R5 *The audio record should not be stored on the device.*

Currently, no medical data is stored on the device to prevent data leaks when a doctor loses a device, or it gets stolen. Rules and regulations also do not allow data to be stored on devices. Therefore, audio data cannot be stored.

R6 *The audio record should be saved with the right electronic medical record (EMR).*

This requirement is needed to ensure patient safety. Storing an audio record with the wrong EMR could mean that a doctor gets the wrong information about the patient. It could also result in a data-leak when a patient can listen to a wrongly stored record from another patient.

R7 *Only authorized people should have access to the audio records.*

Audio records should only be accessible for authorized people, to prevent data leaks.

R8 *The feature should provide data integrity.*

A doctor (and patient) should be able to expect that the audio has integrity. Integrity is also essential for patient safety because tampered audio could contain false statements, which could lead to doctors making wrong decisions.

REQ.	DESCRIPTION	STAKEHOLDERS
R1	A medical specialist should be able to record audio on his or her mobile device	S4, S5, S6
R2	The audio records should be sent to and stored in HiX	S4, S5, S6
R3	Medical specialists should be able to choose the patient for which the audio record is meant	S4, S5, S6
R4	Medical specialists should be able to use their personal devices	S4, S5
R5	The audio record should not be stored on the device	S1, S4, S5, S6
R6	The audio record should be saved with the right electronic medical record (EMR)	S1, S4, S5, S6
R7	Only authorized people should have access to the audio records	S1, S4, S5, S6
R8	The feature should provide data integrity	S1, S4, S5, S6

Table 3.1: An overview of the requirements and which stakeholders they affect directly.

3.3 CONCLUSION

We found eight stakeholders, four functional requirements, and four security requirements. These requirements determine the scope for the rest of the report. The stakeholders help to understand the context and to define requirements. We kept the feature as simple as possible because we are interested in the security. If we look at the security requirements, it is hard, if not impossible, to ensure requirements R7 and R8. Nevertheless, the software should strive to reach these goals. This report should help understand if and to what extent this is possible.

Table 3.1 also shows that not all stakeholders are directly affected by the requirements. Stakeholders S2 and S3, the developers, and the implementation

and support department are missing because requirements that directly affect them, like maintainability and usability requirements, are out of scope. Other stakeholders, like the university stakeholders (S8 and S7), are not affected by requirements because their focus is on the research.

With this chapter, we answered background question BQ1 and defined the scope of the feature. In the next chapter, we will take a look at threat modeling techniques and related work.

CHAPTER 4

BACKGROUND & RELATED WORK

There exist multiple techniques to do threat modeling. In this chapter, we start by explaining what threat modeling is. Then, we will look at six techniques: STRIDE, attack trees, attacker profiles, LINDDUN, CVSS, and attack libraries. Why these techniques are selected is explained in the next section. We use papers and books to understand the techniques. Learning about these techniques helps to decide which are appropriate to answer our research questions.

In section 4.2, we will discuss available tools that can help with the threat modeling process. Threat modeling can be a time-consuming process, so tools that can speed up that process are welcome. We found the tools by searching the internet and looking for tools used in the literature. After the tools, we will take a closer look at the laws, regulations, and standards which ChipSoft has to follow in section 4.3. We will get to know which ones affect this report.

Last, we look at other related work in section 4.4, where we try to find and discuss similar research. This research could help in defining and understanding our research. The upcoming sections should help us understand the context of this project. The chapter should answer background question BQ2:

> *What are existing methods to find possible threats and attack scenarios?*

4.1 THREAT MODELING

Wikipedia defines threat modeling as "*a process by which potential threats, such as structural vulnerabilities or the absence of appropriate safeguards, can be identified, enumerated, and mitigations can be prioritized*"[9]. There are many ways to do this process, and it depends on the context, which method is appropriate. What these methods have in common is that they allow identifying and prioritizing threats to a system. Threat modeling has multiple benefits[10]:

- Design issues with security are found before any line of code is written.
- It helps understand security requirements.
- It decreases the number of security bugs, making software more stable.
- It addresses software issues other techniques and tools cannot do.

Developers will have the most advantage of these benefits when the modeling is done before or during the design stage in the software development life cycle. However, threat modeling can be done during the whole development cycle and be useful. Doing it later will still find (the same) security issues, but it may cost more effort to resolve them.

Over the years, multiple approaches emerged to do threat modeling[11]. The goal of these approaches is to provide a systematic way to find threats. Methods can range from a brainstorming session to creating detailed models of possible attack scenarios. Five methodologies are selected and discussed in more detail. The STRIDE methodology is selected because it is mature, well-known, and supported by tools. At the moment of writing, ChipSoft also uses STRIDE to find security risks. We discuss attack trees because it is a well-known technique. Another reason is that the formal methods group of the University of Twente has a lot of expertise about them[12, 13]. Attacker profiles can support attack trees and give original attack scenarios, so we also selected this technique. We choose the LINDDUN technique because of the focus on privacy, which is a significant aspect of software in the medical field. The Common Vulnerability Scoring System is selected because it is one of the most popular prioritizing methods. Lastly, we discuss attack libraries because they are popular in web and app development.

4.1.1 STRIDE

One of the most mature methodologies is the STRIDE method[11]. It was introduced by Microsoft in 1999 when they started to document their threat modeling process[6]. The name comes from the six types of threats it defines: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege (hence STRIDE). Table 4.1 explains what these types mean, but also which software property gets violated by the type. These are all security properties. Discovering threats is done with the help of

data flow diagrams of the system. Data flow diagrams (DFDs) are build with the following elements[14]:

- **Processes** receive input data and produce output, for example, a server.
- **Data stores** visualize places where data is stored, like databases or a filesystem.
- **External entities** are used to display, for example, users of a system.
- **Data flows** are used to visualize exchanged data between elements.
- **Boundaries** show where data crosses a trust boundary. A trust-boundary is a boundary between parts of the system with a different level of trust[15]. An example is an internet boundary between a client and a server.

Note that boundaries are not part of 'normal' DFDs, but are used in threat modeling[6]. Data flow diagrams do not have to be detailed (although they can be): multiple 'levels' can be made of a system. For example, a 0-level DFD shows the whole system and how it interacts with external entities. This way, developers can create DFDs at the beginning or before the design stage of software when many details are still missing. Later on, more detailed (1- or 2-level) DFDs can be made. Having multiple levels of detail helps find threats on different scales.

When one or more data flow diagrams are made, the next step is to find actual threats, which is done by considering all threat types from STRIDE for each element. Table 4.1 shows which elements are typical victims of which threat types. These types should be considered guidelines. Sometimes it can be hard to categorize a threat under one type which should not matter, as threats should be found, not categorized.

The STRIDE method finds what vulnerabilities there are for a system, but not how these vulnerabilities can be exploited. This differs from, for example, attack trees, where a detailed model is created to show how attacks can be executed. These two methods can work complementary, where STRIDE is used to find the (initial) threats and attack trees are used to get a more detailed view. Finding all threats with STRIDE can be time-consuming by hand[11]. Luckily there exist tools to help with that, as is discussed in section 4.2.

There are also variants on STRIDE, like STRIDE-per-Element[6], STRIDE-per-Interaction[10] and DESIST[10]. This report will only focus on STRIDE.

THREAT	PROPERTY VIOLATED	THREAT DEFINITION	TYPICAL VICTIMS
Spoofing	Authentication	Pretending to be something or some-one other than yourself	Processes, external entities, people
Tampering	Integrity	Modifying something on disk, on a network, or in-memory	Data stores, data flows, processes
Repudiation	Non-Repudiation	Claiming that you did not do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?	Process
Information disclosure	Confidentiality	Providing information to someone not authorized to see it	Processes, data stores, data flows
Denial of service	Availability	Absorbing resources needed to provide services	Processes, data stores, data flows
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do	Process

Table 4.1: An overview of the different threat types in STRIDE. Taken from *Threat modeling: Designing for security*[10].

4.1.2 ATTACK TREES

The second methodology discussed are attack trees. Attack trees were proposed by Schneier in 1999[16]. Since then, they have become a popular aid in threat modeling.

In an attack tree, the nodes represent attacks/actions. The root node states the global goal of an attacker. The children of the root node represent ways to

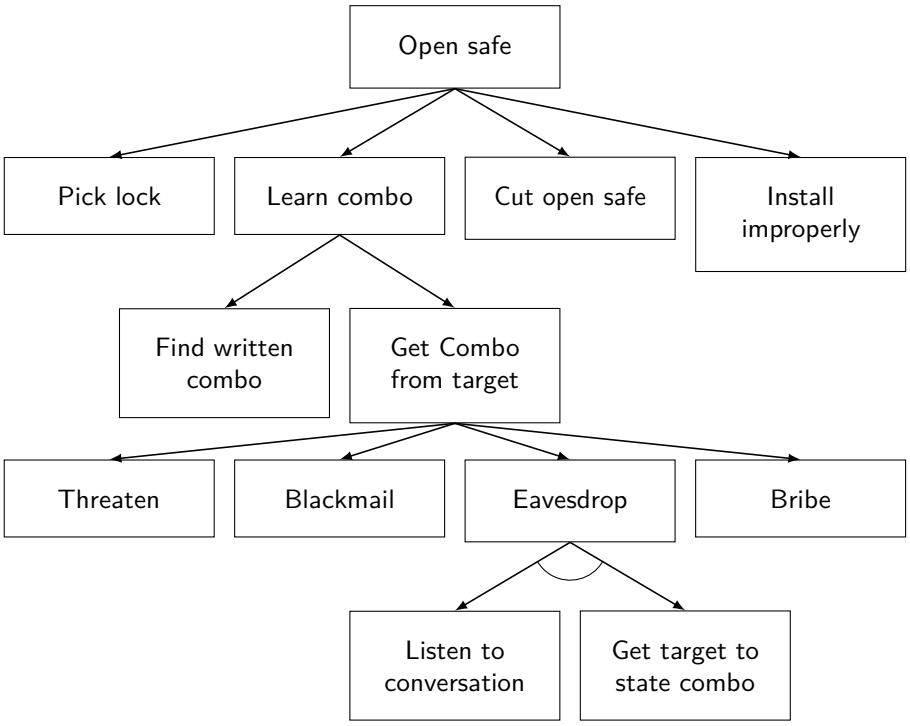

Figure 4.1: An example of an attack tree from Schneier[16]. Note the arc between the two arrows at the bottom, which represents an AND relationship.

achieve this goal. Each child can again have children with ways to achieve that child. This way, nodes get split up as far as needed.

Figure 4.1 shows an example of opening a safe. To open a safe, someone can either pick the lock, learn the combo, cut the safe open, or install it improperly. There is an OR relationship between these children. On the lowest level of the image, an arc can be seen between two arrows. The arc is used to visualize an AND relationship between the children. To be able to eavesdrop, an attacker needs to listen to the target, AND the target has to say the code.

Once an attack tree is created, attributes can be added to attack nodes to give more insight into attacks. Some attributes that can be used are:[17]

- The **Possibility** an attack happens, which can be expressed in percentage, possible or impossible, or categories like high, medium, low.
- The **Cost** to perform an attack, which can be costs to acquire tools or technologies, but can also be expressed as effort.
- The **Competency** required to perform the attack.

- The **Impact** the attack has. For example, an attack could make a company lose money, or intellectual property could be stolen.

Values can propagate up the tree. A node will have the same value as the lowest child when they have an OR relation. An AND relation will give the parent node the sum of the children. When one or more properties are assigned to nodes, attacks can be prioritized. Choices can be made whether it is worth defending against an attack. For example, a company may decide it is not worth to invest in mitigating an attack, which has almost no impact. Assigning these properties can be hard, especially if someone wants to give exact values. It can help to assign values relatively (e.g., an attack could be two times more likely than another attack in the tree).

Attack trees can also be combined with attacker profiles, discussed in the next section. Depending on the profile, attributes of the nodes can change. For example, the possibility of a node can depend on whether the attacker is a criminal or an insider. Different nodes may also lead to different nodes as possible actions differ for each profile.

As discussed earlier, attack trees provide a more detailed view on attacks than STRIDE (and other methods). Consequently, analysts need to have a good understanding of the system and a lot of cybersecurity expertise[11]. What also can be difficult is that attack trees can become large. Luckily they can be easily split into subtrees, where the root node is a leaf node from another parent tree. Common (sub)trees can also be re-used, like an attack on a TLS connection.

As mentioned earlier, attack trees are quite popular. Use-cases have been published, but also research on the method itself. Formal attack trees have been proposed by multiple researchers[18, 19]. Formalization of attack trees allows more reasoning about them, like soundness and completeness. Another interesting paper by Kordy et al. proposes nodes that represent defensive measures[20]. This extension allows more modeling capabilities and can reveal new threats that emerge from used defensive measures. The classic attack trees do not allow modeling temporal dependencies. Arnold et al. propose sequential AND (SAND) gates and OR (SOR) gates[12] for this. These can be used to model attack scenarios more precisely.

Attack trees have also been used in conjunction with STRIDE by making the STRIDE threats root nodes of attack trees[11, 17]. The STRIDE method is used to identify the threats, and the attack trees will show how these threats can be accomplished.

TYPE	DESCRIPTION	LITERATURE
System challengers	White hat or ethical hackers, thrill seekers or glory hunters, young or novice hackers	Novices, Ethical hackers, Browsers & cyber punks
Supporters	Non-technical support functions: mules, cash collectors, business functions such as recruitment, marketing or customer service	
Insiders	Banking employees, employees of third-party suppliers	Insiders
Ideologists	Hacktivists, online activists or cyber terrorists	Hacktivists
Officials	Nation states, sovereign countries, government or its agencies, military functions	Government agents
Professionals I: groups and gangs	Sophisticated large criminal groups or gangs and organised online crime syndicates, also termed as cyber mafia (members often professionally recruited)	Professional criminals
Professionals II: Small Groups and Individuals	Lone hackers and individual attackers, small criminal groups and gangs (can be relatives or friends rather than recruited)	Crackers & coders
Toolkit users	Users of attack toolkits (also called crime-in-a-box, exploit or crimeware kits), clients of criminal-to-criminal services (also named crimeware-as-a-service)	

Table 4.2: An overview of common attacker types found in 200 documents about attacks in the banking world and how they relate to types found in the literature by Moeckel[21].

4.1.3 ATTACKER PROFILES

A different and less formal method of finding threats is creating profiles of possible attackers. Creating these profiles helps to reason about the system from the attacker's point of view. This method can be used to find weaknesses in systems, which other methods may not find. Often, these profiles include motivation, a target, and resources[21, 22]. Resources can refer to a multitude of factors like skill, money, time, but also access to a system.

Moeckel created a list of common attack types found in the literature[21]. She also created eight profiles by examining 200 documents containing "information about digital banking fraud cases and the attackers involved". These profiles can be found in table 4.2. A benefit is that these profiles are based on data and, therefore, more reliable than profiles based on other literature. Unfortunately, this analysis has not been done for health-care applications, but the profiles are generic enough to be used in another domain. So, we could use them in the health-care domain, but we may miss domain-specific profiles.

Another interesting read is a report about the cybersecurity landscape in the Netherlands from the ministry of Justice and Security[23]. The report states that in 2019 the biggest threats are coming from foreign governments like China, Iran, and Russia. Script kiddies and vandals mainly did disruptive attacks by DDOSing. The threat of criminals stays big, mainly because cybercrime scales well. In 2019 there were also fewer attacks from insiders.

More detailed attacker profiles have been proposed by Cleland-Huang[24] called "Personae non grata". These are personas with a background story, motives, goals, skills, and how that person would try to attack the system. These extra details make it easier for developers to think like the attacker, but profiles are also very specific for a particular use-case and hard to re-use.

As mentioned in the previous section, attacker profiles can be used to create and grade nodes from attack trees. Thinking about how an attacker would attack the system can also help in making the attack trees themselves.

4.1.4 LINDDUN

A privacy-focused threat modeling methodology is LINDDUN[25]. Figure 4.2 shows an overview of the steps in LINDDUN. The first step is creating a data flow diagram (DFD) of the (sub)system. The second step is finding threats, the same way this is achieved with STRIDE, but with different types. The main difference is that these LINDDUN types are more privacy-focused: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, Non-compliance. During step three, the threats are examined in more detail with 'threat trees,' a variant of attack trees. In the solution space, step four, five, and

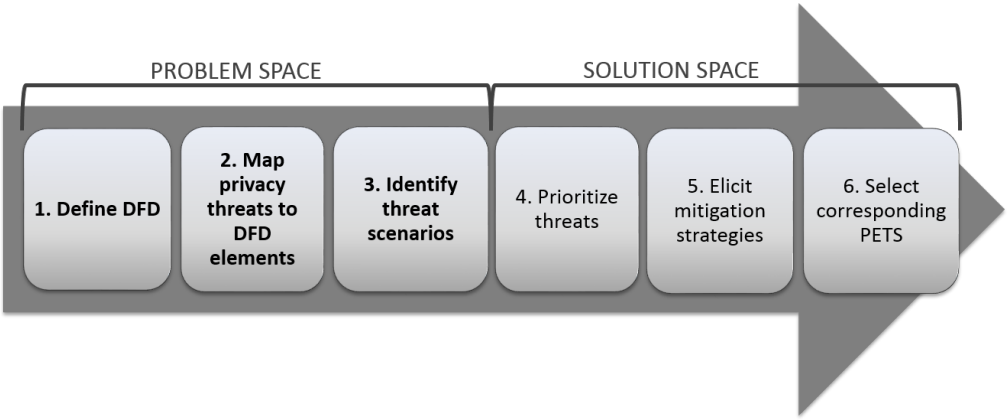

Figure 4.2: The LINDDUN methodology steps[26]

.

six, threats are prioritized, mitigation strategies elicited, and finally, solutions are chosen. LINDDUN does not provide a framework to prioritize threats but rather refers to established risk assessment techniques.

4.1.5 THE COMMON VULNERABILITY SCORING SYSTEM

On of the established risk assessment techniques is the Common Vulnerability Scoring System (CVSS)[27]. CVSS is maintained by the Forum of Incident Response and Security Teams (FIRST) and at the moment of writing at version 3.1[28]. The method can be used to score threats between zero (none) and ten (critical). It does so with the metrics shown in Figure 4.3. Three metric groups are defined. The base metric group represents metrics that are constant over time and across user environments. These metrics give a base score, which can be refined by the other two groups. The temporal metric group contains metrics that can change over time, but not across environments. The environmental metric group entails metrics that are specific to a particular user's environment.

Although CVSS is a public initiative[27], it is unclear how the score is exactly calculated from the metrics. Consequently, some people are critical about the method, but still, the method is widely used[11]. Another thing to note is that details about the attack(s) are needed (e.g., the complexity) to score threats. This requirement means threats found by methods as STRIDE cannot be scored with this technique, as they do not describe attacks. One can argue that CVSS is more a scoring system for attack scenarios than threats.

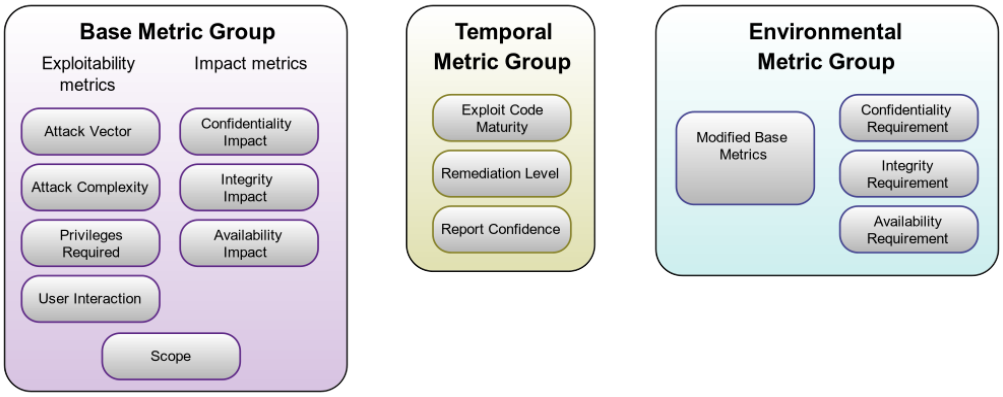

Figure 4.3: The metrics used by CVSS in version 3.1.[28]

.

4.1.6 ATTACK LIBRARIES

An attack library can be a useful tool to elicit threats from a system. Attack libraries can consist of known attacks, tools/code to perform attacks, and models. These can be used as reference material to build secure software. When building a library, audience, detail versus abstraction, and scope should be considered[10]. A very detailed example is a checklist with SQL-injection attacks. The types of STRIDE could be seen as a very abstract attack library.

Creating an attack library can be a rather daunting task. Therefore, communities created some open libraries on the internet. An extensive general attack library is the Common Attack Pattern Enumeration and Classification (CAPEC) library[29]. As the name suggests, it contains all kinds of attack patterns with detailed execution flows. In the domain of mobile development, the *Mobile Security Testing Guide*[30] can be used. The guide discusses how to create secure apps and mitigate threats. It is a project of the Open Web Application Security Project (OWASP)[31]. OWASP contains more projects, like the *OWASP Cheat sheet series*[32], which contains a comprehensive list with security points for a lot of different (web)technologies. This information can be useful when building an app that communicates with a server.

4.2 TOOLS

Threat modeling can be done in any model drawing tool or with pen and paper. However, over the years, there emerged some tools to help with the process. Well-known free tools are Microsoft's Threat Modelling Tool[33] and Threat Dragon[34]

by OWASP. Both tools allow creating data flow diagrams with boundaries and add threats based on the STRIDE methodology. The tool from Microsoft will also generate threats while designing the diagrams, but it is only supported on Windows. Threat Dragon is supported on both Windows and Mac, but at the moment of writing, it is still an early version and not supported on Linux. It does not generate threats (yet), and the editor is less mature than Microsoft's tool.

A different approach is used by pytm[35]. Instead of creating a data flow diagram, the model is created in Python by creating objects (elements like processes) and connecting them (i.e., the data flows). The library then creates a data flow diagram, a sequence diagram, and a threat report. A downside of the tool is that it requires a lot of external tools to generate diagrams and threat reports. It also does not state from where generated threats originate.

A free tool to create attack-defense trees is ADTool[36] based on the formal presentation in "Foundations of attack–defense trees"[19]. The main features of the tool include the creation and editing of attack-defense trees but also quantitative bottom-up analysis of attack-defense scenarios.

Lastly, ThreatModeler[37] is a paid platform with multiple tools to find and keep track of threats over the whole project life cycle. It uses 'Visual, Agile, Simple Threat modeling' or VAST[38]. VAST is based on three pillars to support a scalable solution: automation, integration, and collaboration. Unfortunately, not a lot of (independent)literature can be found about how VAST exactly works. Threatmodeler includes tools to build threat models and attack trees. Users can also build their own centralized threat repository and keep track of those threats.

4.3 LAWS, REGULATIONS AND STANDARDS

Medical (and other) software need to comply with laws and regulations. Software created by ChipSoft should help hospitals to adhere to the "General Data Protection Regulation" (GDPR). In May 2018, the GDPR became applicable to all members of the European Union[39]. It aims to give citizens more control over their personal data. All businesses and organizations handling personal data should comply with the GDPR, thus hospitals too. ChipSoft can help hospitals to follow the GDPR by securing personal data but also keep track of how that data is processed[40].

ChipSoft's HiX is registered as a medical device[41]. Which ensures the software is safe to use in a hospital. Standards are used for the development processes[41], to guarantee quality software:

- **EN ISO 14971:2012**, for application of risk management to medical devices

- **EN-IEC 62366-1:2015 + EN-IEC 62366-1:2015/C11:2016**, to evaluate the usability as it relates to safety
- **EN-IEC 62304:2006+A1:2015**, which contains life cycle requirements for medical device software
- **Meddev 2.12/1 rev.8**, guidelines on a medical devices vigilance system
- **EN-ISO 15223-1:2016 (Cor. 2017-04)**, what symbols to use on device labels etc.
- **EN 1041:2008+A1:2013**, how to supply information with medical devices

When developing software for ChipSoft, these standards must be followed. This report will help follow the first two standards listed above. **EN ISO 14971:2012** will be followed as threat modeling is a risk management tool. The report gives an idea of the risks involved when creating an audio recording feature. It also shows how usability can relate to safety to some extent, supporting the second mentioned standard.

4.4 OTHER RELATED WORK

Research shows multiple cases where threat modeling has been applied to healthcare software. Abomhara, Gerdes, and Køien applied STRIDE to telehealth systems[42]. They were able, with the use of Microsoft's Threat Modelling Tool, to elicit threats and define countermeasures, such as hashing and signing data. Almulhem used attack trees with success on an electronic health record system[43]. These researches unfortunately only conclude that the methods worked. It would have been interesting to know if the type of software (i.e., healthcare) had any impact on threat modeling. A more detailed paper from Cagnazzo et al. used STRIDE with a risk assessment method DREAD, to find threats in a mobile health system[44]. They concluded that, although they found threats, this would only be a starting point, specific threat analysis should be done for each use-case. Mainly because of the heterogeneous nature of mobile health systems. They also argued that secure and usable authentication could be a challenge because mobile health devices could break and be switched a lot.

Kammueller did some interesting research where they proposed a formal modeling and analysis method to prove GDPR compliance[45]. They applied their model to an IoT healthcare system and proved it was compliant with the GDPR.

4.5 CONCLUSION

In the previous sections, we explained that threat modeling is a systematic way to find and prioritize threats. We looked at six different methods to do threat modeling. We talked about STRIDE, which is suitable for initial analysis to find possible vulnerabilities and threats. We discussed attack trees, which can help find actual attack scenarios and their probability of happening. Attacker profiles were mentioned, which provide a different angle leading to attack scenarios other methods may not find. They can also be combined with attack trees. For a more privacy-focused approach, we can use LINDDUN, and to prioritize threats, CVSS is a good option. Finally, we looked at attack libraries, which are a valuable resource for generic threats.

In section 4.2, we looked at tools to help improve the process of threat modeling. We ended with other related work, where threat modeling was applied to software in the medical field. In the upcoming part of the report, we will use STRIDE, attacker profiles, and attack trees to find threats.

PART II

ANALYSIS

CHAPTER 5

FINDING THREATS

In the upcoming sections, we will try to find the main threats and vulnerabilities of the system. We use STRIDE as it should give a good overview of all threats, without going into unnecessary detail. It is also supported by Microsoft's threat modeling tool and already used by ChipSoft.

First, we examine the architecture of the current system, which is needed to create a data flow diagram (DFD) of the feature. The DFD was created in Microsoft's threat modeler tool, and the generated threats are used as a starting point. The STRIDE method is then used to find any missing threats. We find 19 threats, which are discussed in section 5.3 and listed in table 5.1. We also take a look at existing mitigations for each threat.

This analysis will help us get a global view of vulnerabilities and how they could be mitigated. We can use this information when creating attack trees in part III. By knowing possible threats and vulnerabilities, we can think of scenarios (or nodes) an attack might use to obtain data. This chapter should answer research question RQ1:

> *What are possible vulnerabilities and threats for an audio recording feature in a mobile app?*

5.1 THE HIX PLATFORM

As discussed before, this report is about a new audio recording functionality in the existing mobile app for medical specialists. Therefore, it is important to understand how the current system is built. Figure 5.1 shows the (simplified) architecture of the system as a whole. The database and business logic layer is shared between multiple applications. These applications run in a trusted environment on servers and devices at the hospital. As mentioned before, HiX is the main EMR software and runs on desktops at the hospital. COMEZ is

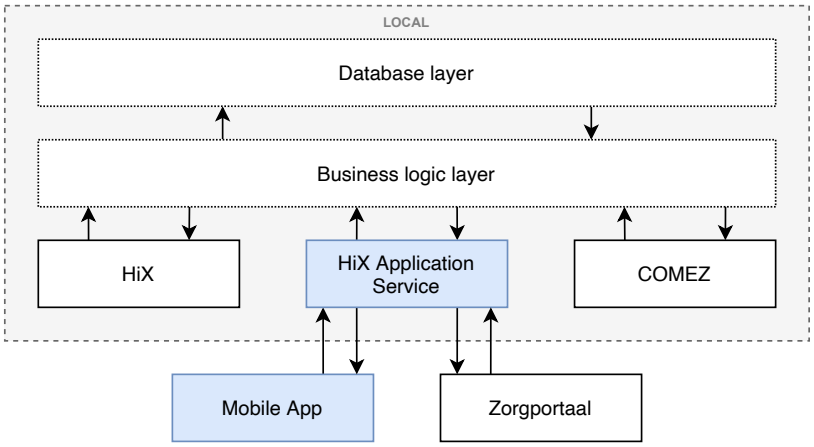

Figure 5.1: The (simplified) HiX architecture. The new feature will have an impact on the app and the HiX Application Service, shown in blue.

used to communicate with other third party software via industry standards. The HiX Application Service (HAS) allows access to data over the internet. There are more applications that use these layers, which are not shown here. These applications are connected to the same database(s) via the database layer. Creating stand-alone applications has multiple benefits:

- It increases availability and stability. If, for example, HiX is down, the HAS can still be up and running.
- It increases maintainability because there is a better separation of concerns. HiX should not be concerned with sharing data over the internet, making the code less complex.
- Each application can have its own (extra) security. The HAS will need other types of security because data is shared over the internet. It is also easier to fine-tune which data is shared this way.

The HAS is used by mobile apps and the ‘Zorgportaal’, an online dashboard where patients can get information about their medical record. Communication is done over HTTPS, and JSON is used to serialize data. The HAS limits which data can be requested and has its own authorization and authentication.

5.2 A DATA FLOW DIAGRAM FOR RECORDING AUDIO

We created a data flow diagram to find threats with STRIDE. This diagram can be found in appendix A. It shows the user starting the recording on the app, crossing the mobile device boundary. The mobile app request either sound from a microphone on the device or an external microphone. The selected microphone will receive sound from the environment, shown with the data flow arrow from the user to the microphone(s). The microphone will return this sound data to the mobile app. For the internal microphone, this will be a binary stream, for the external microphone, this depends on the device and how it is connected. Most external microphones will either be connected with Bluetooth or a cable. The audio data will at least be temporarily stored in local memory, which is part of the application sandbox. In theory, only the app, or a superuser, has access to this memory. Last, the app will send the audio to the HiX application service. The type of data and protocol of this flow will matter for how threats can be mitigated.

5.3 THREATS

The data flow diagram from fig. A.1 is also created in Microsoft's Threat Modelling Tool[33]. The tool has generated the first set of threats, which is rather general. Some of these threats are discarded because they are not applicable to our system. Other threats are added by brainstorming and examining the DFD. The threats are discussed below, ordered by type, and an overview can be found in table 5.1 at the end of this chapter. We also mention some generic mitigations against each threat.

5.3.1 SPOOFING

In theory, each process element in fig. A.1 could be spoofed by an adversary, which means a malicious process pretends to be a legit process in the hopes the system will use it. Therefore the following threats should be taken into consideration.

T1 *Spoofing the user*

The user itself could be 'spoofed' by an attacker, which could lead to unauthorized access. For example, an attacker could steal the device of an user and break into it. A more common situation is where the doctor lends their phone to their spouse, family, or friends. These could then (by accident) open and use the app.

MITIGATION

The device, on which the app is used, could be compromised. The app should ensure an authorized user uses it. A solution for this is already in place: doctors need to log in with their HiX credentials and need to use a device-specific pin code. Logging in is only done once, but the pin code needs to be entered every time the app is used. The code ensures only the doctor can use the app, even when family, friends, or an attacker uses his or her device.

Biometrics could also be used, but ChipSoft decided not to. One reason was that doctors could have multiple fingerprints on their device, and there is no way to enforce that these are all the doctor's fingerprints. People sometimes configure a fingerprint from their partner. Facial recognition has the same problem. It is pretty safe to assume that, when a pin code is entered, it is the doctor opening the app due to the pin code being set right after logging in for the first time. Biometric would work if it would also be set at this point (and limited to one fingerprint/faceID), which would require a custom and expensive implementation. Biometric would also work if you can enforce that they are only configured for the doctor, with tools like *Microsoft Intune*[46] and *Android Enterprise*[47]. Not all hospitals use these tools, so (for now) biometrics are not supported.

T2 *Spoofing the mobile app for the user*

The mobile app itself could be spoofed for the user. A spoofed app could mean the user (e.g., a doctor) could record a patient conversation, thinking it will be saved in the HiX system. However, in reality, the recording will be done with a malicious app and uploaded to a different server.

MITIGATION

The end-user should be sure the app is indeed the app from ChipSoft. Therefore, the app should only be installed via the official Google Play Store or App Store from Apple. The user should check the app is published by ChipSoft in the stores. ChipSoft themselves could check if there are any malicious copies in the store, but this would not be that effective. A malicious app could look different in the store, but when installed, have the same app icon.

T3 *Spoofing the mobile app for the web service*

The mobile app could be spoofed for the web service. The web service may receive 'fake' recordings (i.e., not made by an authorized user) and store them. These could, for example, lead to doctors misdiagnosing patients after listening to their recording, thinking it was real.

MITIGATION

With an authentication system in place, a spoofed app cannot just upload recordings. It needs the right credentials, only known by the end-user. The HiX application service may also use IP whitelisting to allow only requests from within the hospital, although this will not help against IP spoofing. Another security measure could be to create a client-side certificate for the app and check that on the backend. The app and backend also can have a shared secret, although this could be compromised with reverse engineering of the app[48].

T4 *Spoofing the webservice*

The app might be connected to a spoofed web service, meaning recordings can be sent to attackers. If performed well, the user will not know the difference.

MITIGATION

With the assumption the app is not tampered with, the used domain address for the service or IP is correct. With an https connection, some man in the middle attacks can be mitigated. If a domain address is used, an attacker could tamper with a DNS server on a local network. By only allowing communication on a safe and trusted network, it could be made harder for attackers to spoof the web-service.

T5 *Spoofing the (external) microphone*

The app might use a spoofed microphone to do the recording. For example, a malicious virtual microphone could be created, which relays a recording from a real microphone but also uploads this to another server.

MITIGATION

The possibilities of spoofing the internal microphone depends on which operating system is used. For Android, one can consider three methods[49].

The first one is to create a default recording app. To do this, attackers would create a (malicious) app, with an intent-filter, allowing other apps to use it for recording sound. The intent-filter should use action `android.provider.MediaStore.RECORD_SOUND`[50]. Apps using an external app to do their recordings could now use the malicious app. However, if multiple recorder apps are installed, the user will have to select the malicious app. A way to protect the app against this attack is not using an external app to do the recording but use the Android API directly.

The second approach requires tampering with the app itself, which will be discussed in more detail in subsubsection 5.3.2 and subsubsection 5.3.2. The last solution would be a kernel attack where attackers create a virtual microphone. This attack would require adversaries to install a tampered Android version on the device of a doctor. It will be hard to protect the app against this attack, as the whole device is compromised.

On iOS it will be harder to spoof the microphone. The os is not open source, so it will be harder to create a virtual microphone. iOS also does not allow choosing a default recording app.

5.3.2 TAMPERING

Attackers could tamper with the system: they could change code and data. Tampering gives the following threats to our system.

T6 *Tampering the app's runtime*

The app could be tampered with, during runtime, via code injection. Tools like Substrate[51], Frida[52], Xposed[53] or Magisk[54] can be used for injections. Consequently, the security mechanisms implemented in the app could be bypassed.

MITIGATION

There is no way to prevent an attacker from tampering with and using the app. Therefore, the app cannot be trusted to protect data or handle authentication and authorization. These should be the responsibility of the backend, which, currently, it already is. This way, an attacker will always need credentials, even if he or she has tampered with the app.

There can be a threat of users unknowingly using a device where the app is being tampered. Using an unrooted device should prevent tampering with the runtime for the most part, as many tools need root access to the device.

T7 *Tampering the app by binary patching*

The app can be tampered with by modifying the binary executables, tamper with resources or reverse engineering and recompiling the app[30]. Again, the security mechanisms implemented in the app could be bypassed, or malicious code added. The newly compiled app could be installed on a victim's device, and adversaries could steal information.

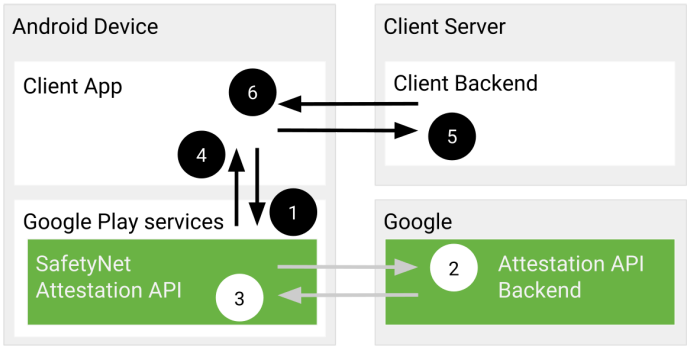

Figure 5.2: The workflow of Googles SafetyNet Attestation service[57]

.

MITIGATION

There is no way to prevent attackers from reverse engineering, tampering, and recompiling the app. It can be made harder by using obfuscating tools, like Swiftshield[55] for iOS or R8[56] for Android.

Another way to make it harder to tamper with the app is to check for the integrity of the app. On Android, SafetyNet can be used to check if the device and the app are tampered[57]. It provides a cryptographically-signed attestation. The workflow is shown in fig. 5.2 and is described in the documentation as follows[57]:

1. The SafetyNet Attestation API receives a call from the app. This call includes a nonce.
2. The SafetyNet Attestation service evaluates the runtime environment and requests a signed attestation of the assessment results from Google's servers.
3. Google's servers send the signed attestation to the SafetyNet Attestation service on the device.
4. The SafetyNet Attestation service returns this signed attestation to your app.
5. Your app forwards the signed attestation to your server.
6. This server validates the response and uses it for anti-abuse decisions. Your server communicates its findings to your app.

Note that these checks done by the app can be removed when an attacker rebuilds the app. It only increases the effort. A way to make it even harder

is to implement a custom attestation service and use that in conjunction. A custom service prevents that an attacker can use a general tool which cracks the SafetyNet Attestation service. He or she should now also crack the custom service and, although this service may be less robust, the effort for the attacker will be increased a lot.

Apple uses signed certificates for apps in iOS to prevent tampering[58]. To upload an app to the iOS app store, a user needs a developer account to get a so-called signing identity. This signing identity consists of a private and public key. They are part of a certificate provided by Apple. The public key is used to sign the app. Apple can use the private key to check if there has been any tampering.

T8 *Tampering the device's OS*

Attackers could tamper with the OS. Attackers could circumvent defenses mechanisms from the app, which depend on the OS. As Android is open source, a custom (tampered) ROM can be made and installed on a rooted device. There are also tools for iOS to change the OS like Substrate.[51]

MITIGATION

The aforementioned SafetyNet from Android[57] can also check (to a degree) if the used ROM has been customized. An attacker could make adjustments to Android to circumvent this service by faking the response as if the ROM was not changed.

The same can be said for iOS: there are ways to check if a device has been jailbroken, but if the device is jailbroken, it is also relatively easy to circumvent those checks.

T9 *Tampering an external microphone*

If an external microphone is used, attackers could tamper with the firmware of this microphone. Tampered firmware could spy on the user.

MITIGATION

Using an external microphone opens some new ways for attackers to record data. One of those is tampering with the microphone's firmware. It will be hard to detect this from within the app. There are a lot of different vendors and microphone types, so making a general way to check for tampering will probably be impossible. Using a wired microphone would be safer in general than one with Bluetooth.

Using Bluetooth creates a whole new attack surface for attackers. In 2017, Seri and Vishnepolsky showed multiple zero-day vulnerabilities in Bluetooth on Android and iOS devices[59]. Older phones may still have these vulnerabilities, and a tampered microphone may use this send data to the attacker. A tampered microphone may also work together with a malicious app by sending data to it. The app could then send this to the attacker. On Android, Vendor-specific AT commands[60] can be used, which allows custom Bluetooth commands. On iOS, the Core Bluetooth framework[61] can be used.

It should be considered if external microphones should be allowed because of the added risk. If they are not allowed, the app could check if one is connected. On Android, the AudioManager[62] can be used, and on iOS, the AVCaptureDevice[63] class.

5.3.3 REPUDIATION

As discussed, repudiation means that an entity can deny that something happened. In the medical field, it is essential to know who did what. The reason being that privacy-sensitive data is being processed. Repudiation gives us the following threat.

T10 *Data repudiation recording by end-user*

The end-user could deny that he or she recorded specific conversations.

MITIGATION

Currently, most actions done by users in the app already get logged. This way, the hospital can know which doctor reads which patient data. Logs are also needed for the created recordings. It should be clear which user made it and when.

5.3.4 INFORMATION DISCLOSURE

The audio that will be recorded will be private. The data must not get into the wrong hands, meaning there should be no information disclosure. Next are the threats of this happening.

T11 *An attacker may listen to the microphone*

Another app could be listening to the microphone in the background at the same time the app is recording.

MITIGATION

When the app is recording a conversation, it should make sure no other (malicious) app is recording at the same time. If attackers were able to do so, the privacy of the patient (and doctor) would be breached. How and the possibility to prevent this will depend on the platform. There may also be functionality in the operating system itself that uses the microphone. For example, when Siri is enabled on iOS, it will always listen for the phrase “Hey Siri” locally and send the sound coming after to be processed[64]. A user could trigger this by accident, sending (private) data to Apple without the user knowing. The Guardian also reported that Apple contractors hear Siri recordings to improve the product[65]. Google does the same with Google Home and Google Assistant. News outlet VRT NWS from Belgium got their hands on recordings from Google, used by employees to improve the products[66]. These also contained false positives, i.e., the assistant got triggered with no intention of the end-user.

The services listening to the microphone are a bigger privacy problem in general, especially in the medical field. The app should try to find out if other services or apps are also using the microphone and either block this or warn the user (and block the recording functionality). How this is achieved will differ from Android and iOS. Both are discussed next.

Android

Until now, it was not possible to share the microphone between two apps. If an app was already recording and another app was trying to record, an error would be returned, and the recording would not start. An exception was when an app (like Google Assistant or an accessibility service) with the permission `CAPTURE_AUDIO_HOTWORD` and audio source of type `HOTWORD` was recording. Then another app could start a recording, where the previously mentioned app would be terminated[67]. If Android is assumed safe, then we can also assume that no other app can listen to the microphone while the ChipSoft app is recording. From the app’s point of view, no extra mitigation is needed, except for making sure the used Android version is not rooted, and no-one tampered with it.

With the new Android Q, the behavior for audio sharing has changed. Android now distinguishes two kinds of apps: so-called ‘ordinary apps,’ installed by the user, and ‘privileged apps,’ which come pre-installed on the device, like Google Assistant and accessibility services. An app is also treated differently if it uses a ‘privacy-sensitive’ audio source, meaning one with type `CAMCORDER` or `VOICE_COMMUNICATION`. The rules for sharing audio are as follows[67]:

- Privileged apps have higher priority than ordinary apps.
- Apps with visible foreground UIs have higher priority than background apps.

- Apps capturing audio from a privacy-sensitive source have higher priority than apps that are not.
- Two ordinary apps can never capture audio at the same time.
- In some situations, a privileged app can share audio input with another app.
- If two background apps with the same priority are capturing audio, the last one started has higher priority.

What is interesting here is that in some situations, a privileged app can share audio with another app, something, which should be avoided. According to the Android documentation, this can only occur when either an Accessibility service UI is on top, or with Google Assistant and an app that does not listen to a privacy-sensitive audio source.

The app should use a privacy-sensitive audio source, so it has priority over other apps. If another (malicious) app also starts to listen to a privacy-sensitive audio source, it can silence the recording. The app should register to `AudioRecord.registerAudioRecordingCallback()` to be able to notice this happening. The user can then be warned that another app started recording, and the app could restart recording to take over the audio source again.

iOS

The official Apple documentation does not state if and how the microphone can be shared. Unfortunately, the emulator from XCode can not be used to test this. It only allows recording the microphone with AVAudioRecorder[68], which writes the recording to a file right away. As discussed in the next section, the audio should be returned as a buffer and not written to a file, for which AVCaptureSession[69] can be used.

T12 *The apps data could be stolen*

An attacker could try to steal the stored data by the app. The app will contain some audio data in local memory. An attacker could try to read this memory.

MITIGATION

As the app could be compromised by tampering, it is best to store the least possible amount of sensitive data on the device. For recording the conversation with a patient, this means the recording should be uploaded right away and not stored on the device. There always will be recording data, as a buffer, on the

device during the recording. However, this will be in RAM and only at the time of recording. Getting data from RAM is a lot harder. He or she cannot get the data at a later moment by rooting the (stolen) device. Still, the buffer on the device should be encrypted. In the future, the buffer may be stored on the device some time to allow recording audio without an internet connection.

T13 *An attacker could sniff the upload connection*

An attacker could get his hands on the recording by compromising the connection between the app and the HiX system.

MITIGATION

The connection between the app and the HiX application service should be secure. Assuming that no data may be lost, a TCP connection is probably the best option. An SSLSocket[70] could be used, which allows a connection over TCP, secured by SSL or TLS. It includes protection for integrity, authentication, and confidentiality. Both Android and iOS should be able to connect to a socket. Currently, the app connects to HiX with a RESTful API, so, for now, it may be easier to send the data in packets over HTTPS.

The send data itself could also be encrypted for an extra layer. For example, a stream cipher[71] could be used. The decision has to be made if this is worth it because it will cost more computational power (and subsequentially battery power) from the device. One scenario where this might be useful is when a doctor decides if the recording should be saved at the end. The app could upload the encrypted recording right away, but without the initialization vector. The encryption means the backend will not be able to read the recording. If the doctor decides to save the recording, the app could send the initialization vector, allowing the backend to read the recording. Privacy will be increased, as the doctor can now decide afterward, knowing the content if the recording should be available on the HiX system. The initialization vector should also be encrypted when sent, which can be done with asymmetric encryption.

T14 *An attacker could sniff the connection with an external microphone*

An attacker could sniff the connection between the device and an external microphone.

MITIGATION

As stated before, using an external microphone will introduce more vulnerabilities, primarily when used with Bluetooth. An attacker could try to listen in on a

Bluetooth connection. Encryption of Bluetooth traffic should at least be used as defined in *Bluetooth Core Specification*[72]. Unfortunately, there have been found multiple vulnerabilities[59] in Bluetooth, and some of them are publicly available. Updating Bluetooth headsets is hard or impossible, so security issues will (most of the time) not get fixed. No updates leave headsets vulnerable for attacks. Android devices have the same problem as they are only updated for a limited time. As doctors may use their own devices, it is hard to tell if they are up to date. A more secure approach would be using a headset that encrypts the data itself.

For now, it would be safer to block external microphones. As discussed before, this can be done with AudioManage[62] on Android and AVCaptureDevice[63] on iOS.

5.3.5 DENIAL OF SERVICE

Adversaries could do a denial of service attack, meaning they will block services by abusing resources. Attacking accessibility could be dangerous if a doctor needs information from a patient in a life-threatening situation. Three denial of service threats can be found.

T15 *The mobile app may be crashed or stopped*

The mobile app may be crashed or stopped. For example, the whole device can be crashed by an app using too many resources. The mobile's os might also give priority to other apps over the recording app, shutting it down.

MITIGATION

In general, keeping an app in the foreground will ensure it will not get closed. On Android, a foreground service[73] could be used to do the recording. It will not be killed when the os runs low on memory. So although the app itself may be killed, the feature of recording will not be killed.

On iOS, the background mode audio[74] should allow an app to record while it is in the background. It should be tested if this is true, as the (official) documentation is not clear on this.

T16 *The microphone may become unavailable*

Depending on the OS the app could lose access to the microphone, mainly because another app requests it.

MITIGATION

Which app gets the microphone is an OS decision, meaning the app could lose its access to another app.

As discussed in section 5.3.4 on Android, in general, the app will lose the microphone if another app decides to use it. Until Android 9, there is not a lot that can be done from the app, except warn the user. In Android 10, more things can be done, like using a privacy-sensitive audio source.

The official iOS documentation does not state how the microphone resource is assigned to which app. Tests will be needed to see how this works.

T17 *The web service may be crashed, overloaded or stopped*

The web service might be overloaded by DDos attacks or crashed by sending broken requests.

MITIGATION

DDos attacks should be prevented on the network level, and protection to broken requests should be done on the backend and or network. Mitigating this is considered out of the scope of this report.

5.3.6 ELEVATION OF PRIVILEGE

An attacker could use elevation of privilege, meaning he or she gets more authorization rights than allowed. Two elevation of privilege threats can be found.

T18 *An end user may gain additional privileges*

An end-user may get additional privileges, allowing a doctor to record for patients who are not under his or her care.

MITIGATION

A doctor could record data for patients where he or she should be able to access. Currently, the HiX application services already implement an authorization layer, so it is assumed this threat is covered.

T19 *An malicious app may gain additional privileges*

A malicious app may try to ‘break out’ of its application container. This app could then execute code as a superuser and, for example, read the memory of ChipSoft’s app.

MITIGATION

As discussed, a malicious app may try and succeed to get superuser rights on a device. It could then have access to all data on the device. Mitigating these attacks can only be done by the OS and not from an app. The only defense is making sure there is as least privacy-sensitive data as possible on the device.

5.4 CONCLUSION

In this chapter, we were able to create a data flow diagram of an audio recording feature. We used this diagram and the STRIDE types to find 19 different threats listed in table 5.1. We also looked at possible counter-measurements against these threats. Most threats can be mitigated with existing techniques. We can talk with more certainty about Android because the documentation is more comprehensive than that from iOS. For iOS, more (practical) testing needs to be done. However, the undocumented behavior of iOS may change without warning. These threats provide us with an overview of possible vulnerabilities. This valuable information helps us define the attack trees and mitigations in part III. In the upcoming chapter, we will take a look at attacker profiles, which we also use for attack trees.

THREAT	DESCRIPTION
T1	Spoofing the user
T2	Spoofing the mobile app for the user
T3	Spoofing the mobile app for the web service
T4	Spoofing the webservice
T5	Spoofing the (external) microphone
T6	Tampering the app's runtime
T7	Tampering the app by binary patching
T8	Tampering the device's OS
T9	Tampering an external microphone
T10	Data repudiation recording by end-user
T11	An attacker may listen to the microphone
T12	The apps data could be stolen
T13	An attacker could sniff the upload connection
T14	An attacker could sniff the connection with an external microphone
T15	The mobile app may be crashed or stopped
T16	The microphone may become unavailable
T17	The web service may be crashed, overloaded or stopped
T18	An end user may gain additional privileges
T19	An malicious app may gain additional privileges

Table 5.1: An overview of possible threats

CHAPTER 6

ATTACKER PROFILES

We will use this chapter to look at multiple possible attacker profiles. First, we explain how we found these profiles. We then discuss the seven profiles we found. We also look at attributes like skill, resources, and access. An overview of the profiles can be found in table 6.1. In the end, we select two profiles: the dissatisfied (ex)employee and the criminal hacker group. These profiles seem the most interesting for creating attack trees and assigning values to nodes. In section 6.3, we explain why. This chapter should answer research question RQ2:

What are possible attacker profiles?

6.1 METHODOLOGY

To find attacker profiles, we held brainstorming sessions with multiple employees. During these sessions, we asked who potential attacks could be and how they would attack an audio recording feature. The questions we used to get the brainstorming started can be found in appendix B.1. The employees were chosen for their role at ChipSoft: the team leaders of the core mobile and multimedia team, a member of ChipSoft's red-hat team, and a technical consultant with expertise in security. The different roles should give a broader view of possible profiles. We compared the suggested profiles with the proposed attacker profiles from section 4.1.3. We found that most profiles are comparable to a profile from the literature, especially those from table 4.2. Comparing them helps us define and grade attributes, as shown in table 6.1. These attributes will be useful for when we create attack trees and have to assign effort values to each node. The attributes we use are skill, resources, and access. The resources attribute is a mixture of money and amount of people.

6.2 POSSIBLE PROFILES

After brainstorm sessions, we found seven different profiles. Each profile can be linked to a profile found in the literature. Table 6.1 shows an overview of all profiles. For each profile, we scored the skill, resources, and access. These values are taken from the literature and brainstorming sessions. Next, we discuss each profile in more detail.

AP1 *Dissatisfied patient*

A patient may not be happy the way he or she is treated in the hospital, which may result in a conflict with a doctor, and trying to discredit this doctor. For example, a patient may try to steal their phone and try to get information. It is hard to say what the skill level of a patient precisely will be, but there is a high chance the patient will not have that much skill. The same can be said for resources. Both attributes are scored as 'low' in table 6.1. Access is scored as 'medium' because the patient will be in contact with the doctor. Without that much skill, the patient will probably hire someone, or try to use existing tools. Therefore, we can see this profile as a 'Toolkit user' from table 4.2.

AP2 *Competitor*

A competitor may want to discredit ChipSoft by exposing leaks in their software. Competitors can be considered to have a high skill level and many resources, while their access to ChipSoft's software is low (see table 6.1). They will try to attack the whole system, where other profiles might focus on a specific part. Attacks can range from getting people into a hospital to access HiX directly, to decompiling code to find leaks and steal intellectual property. A competitor will probably try to stay within the law, or at least in a 'gray' area. Looking at the skill and resources of this profile, it is best comparable to 'Professionals I: groups and gangs" or 'Professionals II: Small Groups and Individuals" from table 4.2.

AP3 *Dissatisfied (ex)employee*

ChipSoft may get in a conflict with an (ex)employee. This employee may try to discredit ChipSoft by exploiting or selling leaks. An (ex)employee will have much knowledge about the software and access to the code base. Therefore, we score attributes skill and access with 'high' in table 6.1. As an individual, an (ex)employee will not have many resources. An expected attack from this profile is creating a back door of some sort. This profile is the same as an 'Insider' from table 4.2.

AP4 *Criminal hacker group*

A criminal hacker group may try to obtain health care data to sell to the highest bidder or use it to blackmail someone. We assume a criminal group has at least some skill. Many criminals will be opportunistic, and use tools they found or bought to make as much money as possible. Using these tools does not require high skills, so we choose ‘medium’ in table 6.1. We also use ‘medium’ for resources because criminals want to make money and not spend too much on attacks. Access is scored as ‘low’ as a criminal group has no direct access. Depending on the group size, it can be compared to ‘Professionals I: groups and gangs” or ‘Professionals II: Small Groups and Individuals” from table 4.2.

AP5 *Hobbyists*

Hobbyists may try to hack the app for a bounty, the thrill, recognition from peers, or to make medical applications safer. We assume they have a high skill because they have a passion for what they do. This passion makes them try new ways to attack systems instead of using existing tools, which requires a high skill. We score resources and access both ‘low’ in table 6.1. We assume hobbyists are acting alone, or maybe in small groups, and they do not have any (in)direct access. They can be compared to the ‘System challengers’ or even ‘Ideologists’ from table 4.2.

AP6 *The Dutch Government*

The Dutch government may try to get their hands on medical data to suppress citizens, for example, by blackmailing political activists. In general, governments have a high amount of resources and people with high skills. The Dutch government has also access to the Dutch network infrastructure and could demand access to ChipSoft’s software. They will need legitimate reasons to do so, by law. Therefore, we score access as ‘medium’ in table 6.1. This profile exists in table 4.2 as ‘Officials.’

AP7 *Foreign Government*

Foreign governments may attack Dutch companies to get intellectual properties or destabilize the country. Again, as with the Dutch government, we can assume they have a high amount of skill and resources. We assume access will be low because they will operate from a different country. They will have to make extra effort to gain access to ChipSoft’s software. Table 4.2 shows the chosen values for the attributes. This profile is the same type as the ‘Officials’ profile in table 4.2.

#	WHO	GOAL	TYPE	SKILL	RESOURCES	ACCESS
AP1	Dissatisfied patient	Discredit hospital/doctor	Toolkit user	+	+	++
AP2	Competitor	Discredit ChipSoft	Professionals I	+++	+++	+
AP3	Dissatisfied (ex)employee	Discredit ChipSoft	Insiders	+++	+	+++
AP4	Criminal hacker group	Obtaining medical data	Professionals I, Professionals II	++	++	+
AP5	Hobbyists	Bounty, thrill, activist	System challengers, Ideologists	+++	+	+
AP6	The Dutch Government	Political influence	Officials	+++	+++	+++
AP7	Foreign Government	Political influence, disruption	Officials	+++	+++	+

Table 6.1: Summary of attacker profiles proposed by ChipSoft and how they relate to the attacker types of Moeckel[21]. On the right, skill, resources and access are shown as low (+), medium (++) or high (+++).

6.3 SELECTING TWO PROFILES

We select two profiles to create attack trees in the next part. The number two is chosen because of time constraints, but in the future, ChipSoft might want to research more profiles. We try to select the two most interesting profiles, with the help of security experts from ChipSoft.

The first selected profile is the dissatisfied (ex)employee (AP3). Attacks by insiders have potentially a big impact, making it an interesting choice. The experts at Chipsoft also have had experiences in hospitals where employees blocked resources by changing passwords, making it realistic that an employee

could attack ChipSoft. Looking at this profile will also provide a critical look at the software design process.

The second selected profile is the criminal hacker group (AP4). As mentioned in section 4.1.3, criminals represent a big part of possible attackers[23]. ChipSoft also has had experience with criminals trying to get into their system. So it is reasonable to expect that criminals will also attack the app.

These two attacker profiles will also give different types of attack scenarios. Insiders will probably try to get malicious code in the code base, while criminals will use (existing) tools to obtain data. Together they give a broad view of possible scenarios.

6.4 CONCLUSION

In this chapter, we discussed seven possible attacker profiles, listed in table 6.1. We used profiles found in the literature and brainstorm sessions to elicit these profiles. We selected two profiles to use for attack trees. The dissatisfied (ex)employee, or insider, is chosen because of the possible impact this attacker can have. The criminal hacker group is selected because they represent a big part of possible attackers. In the next part, we will use attack trees to find attack scenarios, the possibilities of them happening, and how effective mitigations may be. However, first, we will explain how we made the attack trees and assigned effort values to nodes.

PART III

ATTACK TREES & MITIGATIONS

CHAPTER 7

CREATING ATTACK TREES

In this part of the report, we will create attack trees. Attack trees, as we explained in more detail in section 4.1.2, will help us discover possible attack scenarios and the possibilities of them happening. In this chapter, we will explain our process, which consists of six steps:

1. Create nodes of attack trees, with help of STRIDE results and attack profiles
2. Grade leaf nodes with effort values
3. Calculate the effort values from other nodes
4. Propose mitigations, considering the weak points of the attack tree
5. Update the effort values, assuming mitigations are in place
6. Compare attack trees to see the effect of mitigations

In the upcoming sections, we will look at each step in more detail and how they will help answer research questions RQ3 to RQ5. In the next two chapters, we will use this process for the insider and criminals attacker profiles.

7.1 CREATING NODES

We start our process by creating the nodes of the attack trees. We begin with the root node ‘Obtain audio data.’ After all, obtaining audio data is what we are researching with the main question of this report. The tree is then extended with the information about threats and vulnerabilities from chapter 5, brainstorming, and more literature research. During brainstorming, having the attacker profiles helps to think up new ways to attack the system.

We create the attack trees in the desktop version of draw.io[75], a program to draw diagrams. The trees have OR and SAND (sequential AND) gates, shown

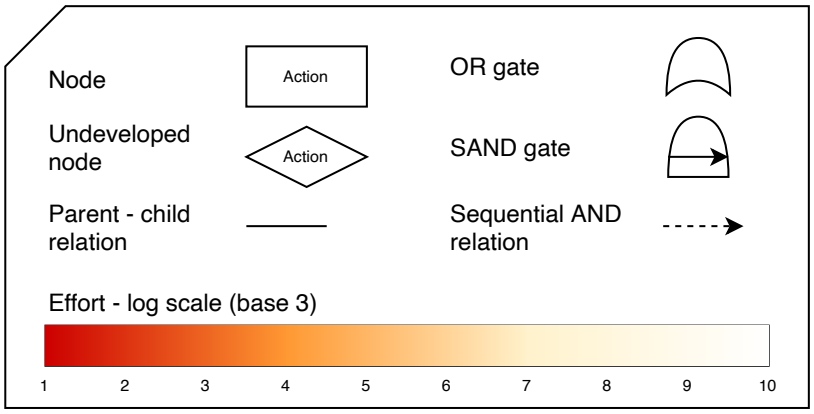

Figure 7.1: The components and there meaning of the created attack trees.

in fig. 7.1, to present these relations between child nodes. They also contain so-called ‘undeveloped’ nodes. These nodes are ignored because they are out of scope and could be extended in future work. The final tree will give a good insight into possible attack scenarios. This answers research question RQ3:

What are possible attack scenarios?

7.2 GRADING LEAF NODES

The next step is to assign values to the leaf nodes. We choose to call this the effort value, meaning the effort an attacker would have to make to perform an attack. The value is a combination of skill and resources. It is hard to give exact values to nodes, so we try to grade nodes with relative values. For example, node A costs three times more effort than node B. We mark nodes from one to ten. One means a node costs little to no effort and ten an impossible amount. We use a logarithmic scale, to increase the total range while keeping the scoring system simple. We chose base number three, which is a trade-off between granularity and range. Choosing a too small number will give us a too small range, which may result in an impossible brute force costing only ten times more effort then decompiling an app, for example. In real life, this would not be the case. If we choose the base number too high, nodes may end up with the same number (remember we rate with round numbers), which in reality costs a different amount of effort. For example, with base number ten, if node A cost five more times effort than node B, they still will have the same effort number. This way, we lose (important) granularity.

The values we assign are educated guesses. To make sure they are realistic, they were discussed with security experts from Chipsoft and the university. We also based the grades more on Android than iOS. The documentation from Android is more extensive, making it easier to grade nodes more precisely. Unfortunately, the official documentation from iOS is less comprehensive, making it hard to know the effort value for a node.

7.3 CALCULATING EFFORT VALUES

The next step is calculating the effort values for each node from the leaf nodes. Draw.io[75] stores its diagrams as XML, which we update with a script to calculate the values automatically. If a node has child nodes with an OR gate, that node gets the value of the lowest child node. After all, to perform the parent node, we can choose any of the OR child nodes, so we only need the lowest amount of effort. If a node has child nodes with a SAND gate, the parent node gets the sum of the child nodes as effort value. In this scenario, an attacker will have to do all child nodes, so we need the sum of the children. The script also colors all nodes, depending on the effort value. The color scale can be seen in fig. 7.1. The colors help us identify higher and lower effort values more easily. With the values calculated, we can say which attack scenarios are more probable than others, and we answered research question RQ4:

What are the probabilities of attack scenarios?

7.4 PROPOSING MITIGATIONS

With all effort values calculated, we can propose mitigations that focus on the weak parts. The color-coding helps us find these points more easily. To think of mitigations, we use the mitigations mentioned in chapter 5 as a resource. For each mitigation, we look at which stakeholder from section 3.1 can help achieve it. This way, we know which mitigations ChipSoft could do themselves and which need to be done by external parties, like the hospitals.

7.5 UPDATE ATTACK TREES

The next step is to update the effort values of the attack trees for each mitigation. We do this by assuming the mitigation is in place. This will lead to higher effort values on certain nodes or newly added nodes. If nodes are added, we also add them to the original attack tree with value 0. We do this to keep all trees consistent, which prevents errors with missing nodes and makes them more manageable. Technically, the zero in our scale ends up being one as a value, but this should not affect our values too much. We use our script to recalculate the effort values on all nodes. Besides calculating the effect of every single mitigation, we also calculate the effort values when we apply all mitigations to the tree. Again the new effort values are educated guesses and checked with experts.

7.6 COMPARE ATTACK TREES

As the final step, we compare the effort values from all trees. We discuss what mitigations are most effective. We look at the effort values of the root node and some high-level nodes. These results will help us answer research question RQ5:

What are possible attack scenarios?

In the next two chapters, we will follow this process for the insider and criminals profiles.

CHAPTER 8

THE INSIDER

Attack trees will help us understand what attack scenarios are possible and which are most likely. Knowing which attacks are most likely will help to implement mitigation strategies effectively. In this chapter, we will create an attack tree for the insider profile, which can be found in appendix C. We follow the steps discussed in the previous chapter. In section 8.1, we discuss the created attack tree, which is the result of the first three steps. We take a look at some interesting nodes in the tree and their effort values. In section 8.2, we use the attack tree to propose mitigations, which are listed in table 8.1. For each mitigation, we look at how it would change the tree and which stakeholders are responsible. This section corresponds to step four and five from the previous chapter. Finally, in section 8.3, we compare the effort values of the root and some high-level nodes after applying the effects of mitigations. The results are listed in table 8.2. We find that applying mitigations can make it three times harder for an insider to obtain the audio data.

8.1 ATTACK TREE

The full attack tree has been added to the appendix (fig. C.1) and can be seen on digital devices by zooming in. As explained in the previous chapter, the tree contains nodes with SAND and OR gates, has numbers representing the effort values, and is color-coded. The diamond ‘undeveloped’ nodes are ignored because they are out of scope for this report. The root node is obtaining the sensitive audio data, which has four child nodes. The most left is obtaining data via the upload connection, which is discussed in the next section. The second is getting data from the server and is out of scope for this report. The third and fourth are ‘getting data from the app’ and ‘tamper with the app’ respectively.’ Table 8.2 shows the effort values of top-level nodes and the root node. The effort values are educated guesses and have been reviewed with the security experts

from ChipSoft to see if they are realistic. This section covers steps one to three, discussed in the previous chapter. Next, parts of the attack tree are discussed in more detail.

8.1.1 ADDING MALICIOUS CODE

One of the weakest nodes in the attack tree is adding malicious code, as can be seen in fig. 8.1. Currently, Microsoft's Team Foundation Version Control (TFVC)[76] is used as a central versioning system. Developers can check-in code directly in the main branch of their team. After the check-in, an automatic build process checks if the code still builds and if all unit tests are still passing. If not, the developer gets notified and has to fix or rollback their code as soon as possible. So as long a malicious developer ensures these two requirements, he or she can add code without ringing any bells. Therefore, we consider that it is easy to circumvent the review process and add (malicious) code to the code base.

Code gets also reviewed by other developers. Reviewing is done by searching check-ins with a specific issue number. So if a developer does not add an issue number to the check-in, this will not come up during code reviews. Another approach could be using an issue number from an issue that is already reviewed or a non-existing number. Although here, there is a more significant risk a developer may notice. An approach which cost more effort is to get a 'fake' review by finding an accomplice or using the account of a colleague.

Check-ins will also show up in the history of a file, but would probably be hard to spot. Developers can also check-in code into code of other teams, except the 'core' teams, which are responsible for the framework the system is built on and general code. This check-in will be easy to spot because in the history of a file as the name of a developer will show up that is not part of the team. Of course, an attacker could try to get the credentials of another colleague, to cover up themselves or get access to core team code.

8.1.2 THE UPLOAD CONNECTION

One way to obtain data is by getting it from the data flow between the app and the server. In our attack tree, we are focusing on two paths: active sniffing, by a man in the middle attack, and passive sniffing.

Figure 8.2 shows the man in the middle attack node with children. An insider would need to control a node and send traffic to it. An attacker could tamper with the network that hosts the back-end, but we assume this will be hard to do because the employee has no direct access to these networks. How this can

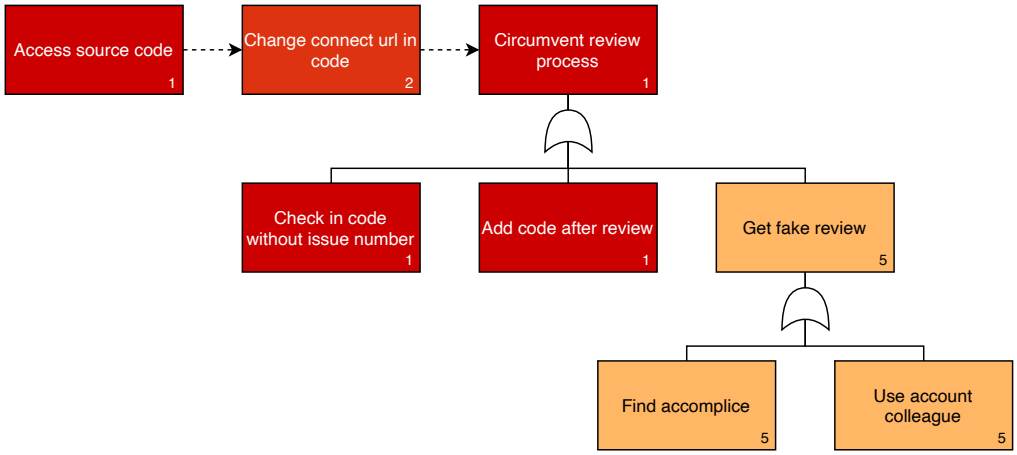

Figure 8.1: It is relatively easy for an insider to add malicious code.

be done is considered part of the back-end and out of scope. Besides, it would be easier to change the code of the app and route the traffic to the attacker's node. An attacker might not only change the URL, as this can be too obvious. An approach would be to overwrite the URL when certain conditions are met, like when it is running on a doctor's phone. This approach could help hide it during testing and development.

The full attack tree shows that using passive sniffing will be harder than a type of man in the middle attack. Obtaining the private keys used by the back-end and app to establish a secure connection would require the least amount of effort. He or she would then be able to decrypt sniffed traffic.

Another approach would be to force security protocols with known vulnerabilities like SSL3.0 and the POODLE[77] vulnerability. Forcing SSL3.0 can be hard because both the device and the back-end server need to support it. Android stopped supporting this since API level 26 (Android 8)[78], and iOS also does not allow apps to use it anymore. Another problem with using vulnerabilities in older protocols is that most of these vulnerabilities are not useful to decrypt a whole data stream. Most exploits are focused on obtaining data like passwords, which is a relatively small part of the whole stream.

8.1.3 GETTING DATA FROM THE APP ITSELF

The second general approach is getting data from the app itself. The attack tree has three sub-branches: access the microphone remotely (via the app), send the data from the app, or store the data on the phone and obtain it at another time. The most cost-effective way is to send data to a server set up by the attacker,

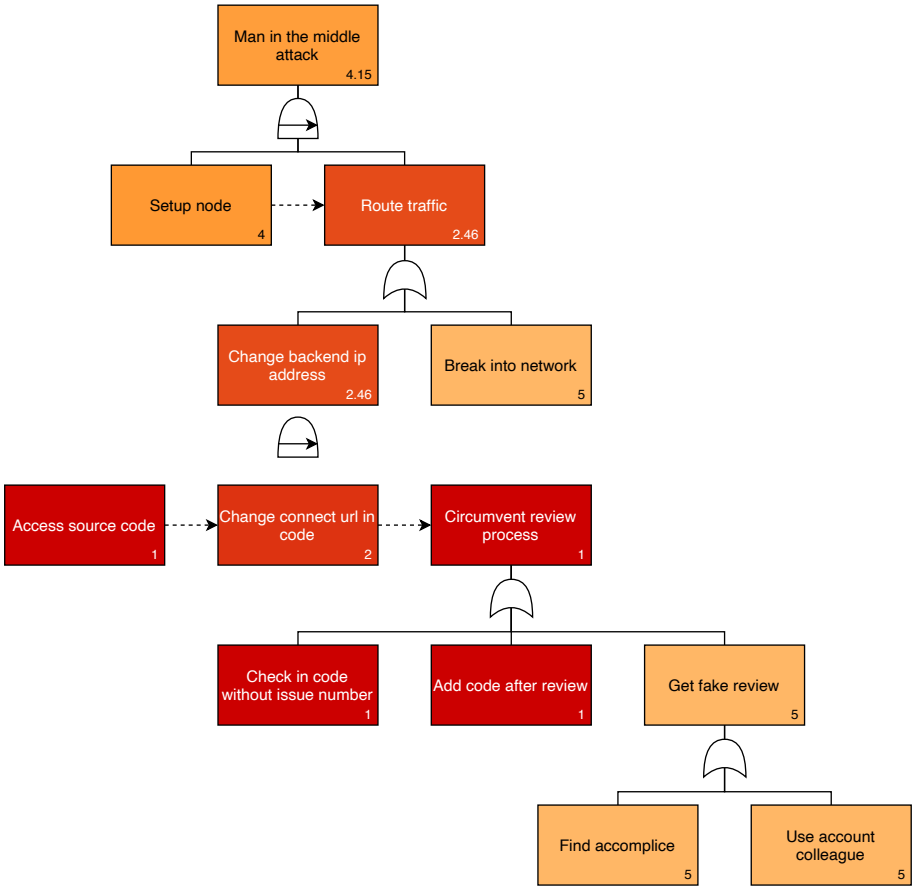

Figure 8.2: Possible ways for an insider to perform a man in the middle attack.

shown in fig. 8.3. Again, this depends on the insider adding malicious code. An attacker could record and send data any time the app is used, but a doctor could notice this because it will ask for microphone permissions. Less obvious would be to send the data when a doctor wants to record or when recording is stopped. The latter approach would need a way to store the data on the phone so that it can be sent later on.

The second most cost-effective is injecting malicious code to access the microphone remotely. The benefit for an attacker can be, that this is less obvious than the previous approach because the app will only send data when the attacker chooses. The app would need a back door to connect to, like a socket server. The infected device would also need to let the attacker know their IP address, so he or she can connect to it.

The last considered way to get data from the app is by storing the data on the

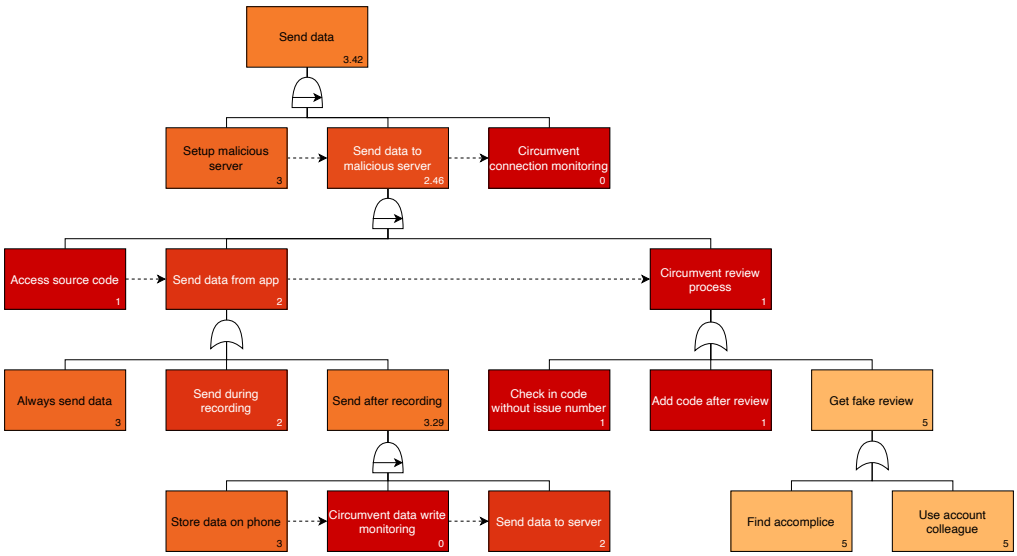

Figure 8.3: Possible ways for an insider to send data to a malicious server.

phone and accessing it. This approach requires to store the data on the phone and to obtain said phone. This method will not be that effective for the insider as it will be hard to obtain a significant amount of data, as he or she needs to steal a phone each time.

8.1.4 TAMPER WITH THE APP

The last considered option is to tamper with the app and make the doctor install this version. The first step here is to obtain the source code. This action should not be a big issue for an insider as developers at ChipSoft can access all source code. A harder approach would be reverse engineering the app. After obtaining the code, the attacker could add malicious code and use any of the techniques already shown in other parts of the attack tree. The hardest part will be to get the app on the device of the doctor. Installing can be done manually by the attacker, but will require access to the device, which may be hard for an insider. Another approach may be to add the (tampered) app to the app store and hoping doctors will install and use it. The attacker will have no control over this.

#	DESCRIPTION	STAKEHOLDERS*
M1	Secure process of adding code	S1,S2
M2	Decrease access to (security-sensitive) code	S1,S2
M3	Monitor external connections	S1,S2,S4
M4	Monitor stored data	S1,S2,S4
M5	Add extra encryption layer	S1,S2
M6	Check signature app	S1,S2

* S1: ChipSoft, S2: Developers, S4: The hospital(s)

Table 8.1: The proposed mitigations and which stakeholders are or can be responsible for them.

8.2 MITIGATION

Mitigations should help increase the effort for an insider to perform the attacks from the attack tree (fig. C.1). We propose six mitigations, listed in table 8.2. This table also shows which stakeholders are or can be responsible for them. As can be seen, the stakeholders ChipSoft and developers are responsible for all our proposed mitigations. This result can be explained by the fact that our attack tree focuses on an insider at ChipSoft. Therefore, our mitigations also focus on taking counter-measures at ChipSoft.

For each mitigation, we look at how it changes the attack tree when it would be applied. We can then recalculate the effort values of the attack tree. Table 8.2 shows the effect of all mitigations separately and together. A mitigation can change either the values of nodes or introduce new nodes. The newly added nodes are also added in the original attack tree with effort value 0 to keep all versions of the attack tree consistent. With consistent attack trees, it is easy to notice if one of the trees is missing a node when the original attack tree is updated, reducing errors. This section covers steps four and five of the previous chapter. We will now look at each mitigation in more detail.

M1 *Secure process of adding code*

One of the weakest points in the attack tree is circumventing the code review process (section 8.1.1), as can be seen in fig. 8.1. Making it harder to add code unnoticed will help mitigate these types of attacks. One way to solve this is to make sure other developers review all check-ins. There still would be a chance the reviewer(s) would help the attacker and add malicious code, but the chance would be a lot smaller. Each check-in should be linked to an open issue to ensure code reviews Issues should only be closed when all code is reviewed and approved by another developer. A system could check all check-ins and warn teams when it is not linked to an issue.

This approach still has a downside that the code is already in the main branch. If a warning is ignored or forgotten, code will still be added unnoticed. It would be better if code can only be added after it is reviewed and approved by other developers. Unfortunately, Team Foundation Version Control has no straightforward way to do this. One technique would be to disable direct check-ins on the main branch and only merge feature-branches into the main branch. However, in TFVC, creating a branch is a 'heavy' operation and mostly can only be done by certain (privileged) developers[79].

A better solution would be to use Git[80] in combination with the existing Team Foundation Server (TFS). Git allows only to add code to the main branches via pull requests. A pull request is a request to merge a (feature) branch into the main branch. Before the merge can happen, it is possible to have 'guards.' For example, that another developer reviewed and approved the request, or that the code can be built by an automatic build system. Consequently, no merge can happen if the new code is not approved or will break the build. Using this process ensures all code is approved by another developer. Using Git is also more future-proof as it is now the default versioning system for TFS[79], and Microsoft has chosen Git over TFVC[81].

There are two stakeholders responsible for this mitigation: ChipSoft and the developers. ChipSoft should enforce this mitigation by making it a company-wide policy. Developers should follow this policy and make sure their colleagues do as well. If we update the attack tree with this mitigation, the 'check-in code without issue number' and 'add code after review' (see fig. 8.1) will be updated to 10 (i.e., impossible). With this mitigation, these actions will not work anymore to circumvent the review process. Therefore, an insider will now need to find other ways to circumvent the review process. Table 8.2 shows how this mitigation affects the effort values.

M2 *Decrease access to (security-sensitive) code*

Currently, developers have read access to all source code. This access helps when fixing bugs and developing new features as it allows to use the debugger on other team's code. It also makes it easier to understand how code from other teams works when (up to date) documentation is missing. The downside and significant risk are that an attacker can search for weaknesses in the system or create a malicious copy.

Ideally, developers would only need access to the modules they are working on, but this is not realistic. A concession could be made by decreasing access to specific modules with security-sensitive code. This way, development time will not be affected too much, but the security increased. An attacker could still decompile the modules, and an extra precaution can be using stubs. These could even decrease development time as these stubs can skip compute-heavy operations.

Teams also should not be able to check-in code in modules of other teams. Currently, this is an unwritten rule, but it should be enforced with user rights. It is already enforced for code from the core teams, so it should be relatively easy to apply this to other teams as well. Again, as with the previous mitigation, ChipSoft and the developers will be responsible for this mitigation.

Decreasing the access to code will change the effort value from the 'access source code' node. The effort value will now depend on which code needs to be updated: accessible code or not accessible. It is hard to assign a value without exactly knowing which code becomes inaccessible. We assume attackers can make most back-doors in their own team's accessible codebase. However, it will take more time because they will have to work around inaccessible security code. Therefore, we choose the new effort value 3 for now. The 'steal source code' in 'the tamper with app' branch will be updated from 1 to 5 as this attack will require all source code. We assume getting all code is as hard as getting things as private keys.

M3 *Monitor external connections*

A good way to know if any data is leaking is monitoring for suspicious connections. This would mitigate approaches where an attacker adds malicious code to send data to their server. Monitoring could be done all the time by code in the app itself. For successful monitoring, this code should not be easily accessible (mitigation M2). If so, an attacker knows how to circumvent it. The benefit of this approach is that it can run in production.

Another way would be to monitor this during integration tests. This approach would allow us to check the whole device and may find traffic that cannot be

detected from the app. A downside is that an attacker could circumvent this check by only activating malicious code when it is running on a doctor's device.

These two approaches should, again, be applied by ChipSoft and the developers. A third approach could be made by the hospital: monitoring with other software on devices. If a hospital already uses device management software, they could extend it with this type of monitoring.

A node is added to the attack tree called 'Circumvent connection monitoring' to see the effect of this mitigation. The effort value of this node depends on how the monitoring tool will and can be implemented. We assume that an attacker needs to know how the monitoring works, to circumvent it. We also assume that the monitoring is not accessible to all developers. Therefore, we give this node effort value 5, which is like stealing a private key.

M4 *Monitor stored data*

Requirement R5 in section 3.2.2 states that no data should be stored on the device, as this increases the possibility data gets leaked. The attack tree shows that some attack scenarios require data to be stored on the device. Monitoring if the app stores data could be a way to mitigate any (accidental) storage of data. The same tactics could be used as with mitigation M3: monitor in a production environment, during the integration test-phase or with another external tool on the device.

A node is added to the attack tree to represent this mitigation. The effort value is dependent on how the monitoring will be implemented. We assume the attacker needs to steal the monitoring code to circumvent it, just like with mitigation M3. Therefore, we give the node effort value 5.

M5 *Add extra encryption layer*

Another layer of encryption could be added to counter a compromised upload connection. Symmetric encryption would probably be best from a performance point of view. The app and back-end could use a key known beforehand, use a new calculated key or share a new random key. The insider will probably compromise the connection by stealing private https keys, so stealing the encryption key will not be that much extra effort. If a key is calculated each time, the insider could find out how this is done in the code and still decrypt the data. Mitigation M2 could help prevent this, but it still means keys can be predicted. Using a new shared key will also be known by the attacker as he or she already has access to the connection. In conclusion, it is questionable if it will add that much extra security. On the other hand, encryption can make it harder and is relatively easy to implement.

This mitigation will allow doctors to decide if a recording should be saved at the end of a recording, by sending the key after. Data would be stored encrypted on the back-end, where the system cannot access it (yet). A doctor could still cancel it, and the recording can be safely deleted, without any other entity accessing it.

Again, this mitigation needs to be applied by ChipSoft and its developers. An extra node is added to the attack tree to calculate the effect of this mitigation. With the assumption that if an insider can break the https encryption, this extra encryption will be relatively easy, the effort value given is 3.

M6 *Check signature app*

As mentioned before, in section 5.3.2, preventing attackers from tampering with the app is not possible. Though, it can be made harder to do, which may deter some attackers. One way would be to make use of the fact that apps need to be signed to be installed. It is possible to check the signature in the app code and compare that with an expected signature. This can be done on the app itself, but a better solution would be to do this on a server. The aforementioned Google's SafetyNet Attestation service[57] could be helpful with this.

ChipSoft and the developers will be responsible for this mitigation. An extra node is added to calculate the effects. As this check will only slow down the attacker, the effort value 3 is given to this new node.

8.3 RESULT MITIGATIONS

By updating the attack tree with the effects of the mitigations, we can see how effective the mitigations will be. The updated attack trees can be seen in appendix C.1. Table 8.2 shows the result of the newly calculated attack tree for the top-level nodes and root-node. Mitigation M1 is very effective. It increases the root node the most and affects the second-most nodes. It also has the highest increase of individual nodes. Mitigation M2 is most overall effective but does not increase nodes that much. Mitigation M3 increases the root node with the same amount as mitigation M1, but affects fewer nodes. The last three mitigations have no effect on the root node but do have the highest effect on specific child nodes. Mitigation M4 has the highest, but still small, effect on accessing data on the phone. Mitigation M5 has only a small effect but is the only one affecting the 'collecting data connection' node. Mitigation M6 has a small, but highest, effect on the 'tamper with app' node. Looking at all mitigations combined it is interesting to see that all nodes are increased. The 'tamper with app' is now the

DESCRIPTION	NONE	M1	M2	M3	M4	M5	M6	ALL
Man in the middle attack	4.15	+1.14	+0.21					5.63 (+1.21)
Collect connection data	5.1					+0.08		5.18 (+0.08)
Access microphone remotely	3.74	+1.99	+0.53	+1.46				6.12 (+2.38)
Send data	3.42	+1.72	+0.41	+1.73				5,74 (+2.32)
Access data on phone	5.31	+0.48	+0.06		+0.49			6.14 (+0.83)
Tamper with app	4.14		+0.08				0.22	4.42 (+0.28)
Root node	**3.42**	**+0.72**	**+0.41**	**+0.72**				**4.42 (+1.00)**

Table 8.2: Some high-level nodes and the root node of the criminals attack tree and their effort value per applied mitigation. Higher numbers mean more effort and are therefore better.

lowest node. The other nodes are now all above 5. The root node is now 1.00 higher, meaning it will be three times harder to get the data for an insider.

8.4 CONCLUSION

In this chapter, we looked at an attack tree created for the insider attacker profile. We then proposed mitigations and looked at the effects. We used the steps discussed in the previous chapter to do so. The attack tree had some interesting nodes, like the way an insider could add malicious code to the code base. In general, it does not cost much effort for an insider to act maliciously. The six mitigations we proposed try to increase this effort. ChipSoft itself can do all mitigations. Comparing the effects of these mitigations with the original tree showed that it could make it three times as hard to obtain the audio data.

Securing the process of adding code to the code base (mitigation M1) is the most effective mitigation. It may cost ChipSoft effort to implement this mitigation, but it can then be used for other projects as well. If Git[80] is used, broken builds

on the main branch can also be avoided. Consequently, developers will not be blocked by broken builds, which will lead to higher productivity. So there is a multitude of reasons, besides security, to apply this mitigation, and it is the first one we advise to do. We also advise applying the second mitigation: restricting access to security-sensitive code. It is an effective mitigation and is relatively easy to do, assuming security-sensitive code is not spread through the whole codebase. Both of these mitigations have an impact on the development process, which ChipSoft might consider too costly. In that scenario, we advise to at least implement the third and fourth mitigation: monitoring for external connections and saving data. These mitigations should help find malicious code and can be used for the whole app, not only the audio recording part. The last two mitigations do not have that much effect but are cheap to implement.

With this chapter, we learned about possible attack scenarios, their probability, and how we could mitigate them, which answers research questions RQ3 to RQ5 for the insider profile. In the next chapter, we will repeat the process for the criminals profile.

CHAPTER 9

CRIMINALS

In this chapter, we will look at the attack tree and mitigation for the criminal profile (AP4). We used the same steps as in the previous chapter and discussed in more detail in chapter 7. We start by discussing the created attack tree for the criminal profile in section 9.1. This tree is the result of the first three steps mentioned in chapter 7. We will also take a closer look at some interesting parts of the tree. Next, we propose seven mitigations in section 9.2. An overview of these mitigations is listed in table 9.1. For each mitigation, we will discuss how it affects the effort values of the attack tree. Finally, we will (re)calculate the effort values and compare these with the original ones in section 9.3. As with the previous chapter, this chapter should help find attack scenarios, their probability, and how we can mitigate them. These findings will answer research questions RQ3 to RQ5.

9.1 ATTACK TREE

Figure C.9 in the appendix shows the created tree. As with the previous attack trees, the tree contains SAND and OR gates. Each node has a logarithmic effort value with base number three and is colored to reflect that number. The root node is again obtaining the audio data. Compared to the insider attack tree, the effort values are higher. This tree again has the ‘get data from upload connection’ and ‘tamper with app’ nodes. New high-level nodes are ‘using a spying app,’ ‘virtual microphone,’ and ‘tampered external microphone.’ In the upcoming sections, we will look to the top-level nodes in more detail.

9.1.1 THE UPLOAD CONNECTION

Getting data from the upload connection has two child nodes: via a man in the middle attack (MITM) or by sniffing data. The MITM can be achieved by breaking

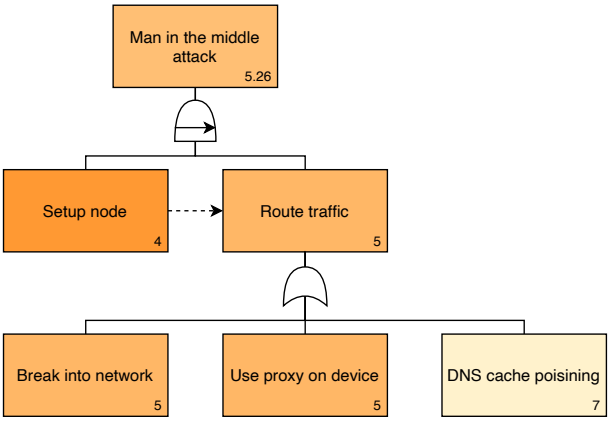

Figure 9.1: The effort values for a man in the middle attack

into the network, getting a malicious proxy on the device, or techniques like DNS cache poisoning. It will take criminals some effort to perform this attack. From the app, there is not a lot we can do to mitigate this. The most effective mitigations are done on the network itself, which is out of scope for this report. The other technique, sniffing data, will be harder to accomplish, assuming TLS is used.

9.1.2 TAMPER WITH APP

Tampering with the app is the weakest point of the attack tree. One of the reasons is that it is not possible to prevent tampering. Criminals would have to do the same steps as the insider in section 8.1.4. The easiest way to obtain the source code would be by decompiling a downloaded version from the store (fig. 9.2). Then they could tamper with it and inject malicious code to steal sensitive data. The hardest part will be to get the tampered app installed on the device of a doctor. The attacker could try to get physical access to the device to install it, but a more effortless way would be to trick the doctor in installing it from the app store. One way would be to create a look-a-like app in the store, with (almost) the same name and icon, and hope the doctor installs it. Another way could be to create a different app and get the doctor to install that. The criminal group could then update this app with the tampered app, which looks the same as the official app. The doctor may use the app of the criminal if he or she does not notice the two identical apps.

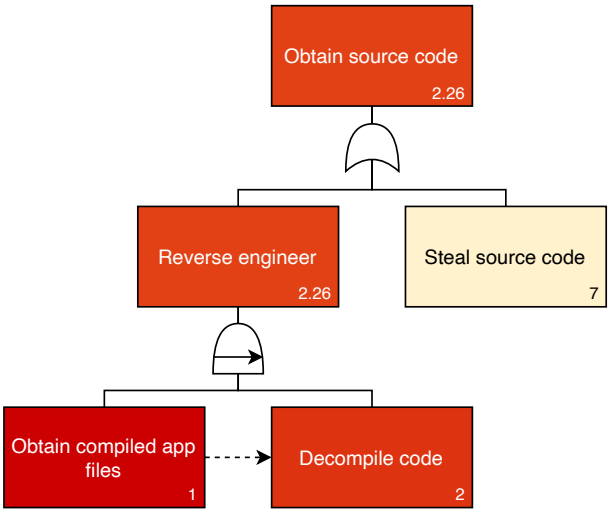

Figure 9.2: Obtaining the source code

9.1.3 USE A SPYING APP

Criminals may decide to use a spying app to obtain the audio recording data. This type of attack also allows them to obtain data continuously. The first step is obtaining an app to spy on the doctor. Figure 9.3 shows how this can be achieved. Criminals can create the app or buy it. The buy node is the lowest because we assume criminals will know how to obtain that type of software. There is also a big chance they already bought or created it. In the attack tree, this is considered part of the buy node.

Next, the criminals will need to find a way to get the app on the doctor's device. The nodes here are similar to getting a tampered app on the device. The part of doing this via the app store differs a bit because the spy app does not look like the app from ChipSoft. Criminals could try to create a legit useful app with the malicious code embedded or create a look-a-like of a popular app. In both scenarios, they should hope a doctor installs it.

9.1.4 VIRTUAL MICROPHONE

In section 5.3.1, we talked about the possibility of spoofing the microphone. Spoofing can be done by creating a malicious virtual microphone. The app can then use this microphone to record audio, which would also send the data to a malicious server. Figure 9.4 shows how an attacker can achieve this. This approach has the downside that it only works for one doctor. It also needs a

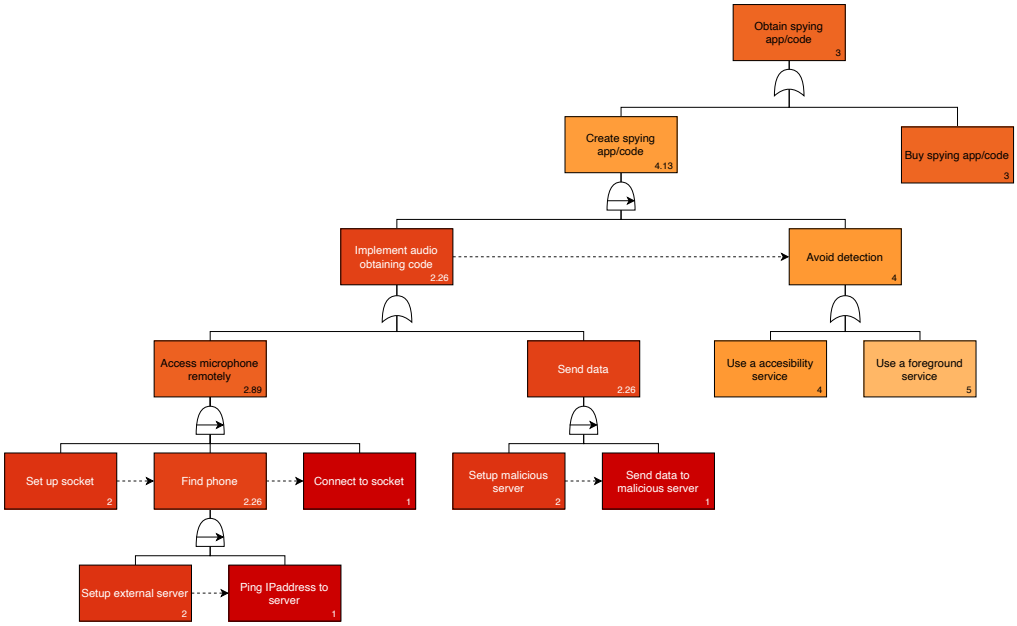

Figure 9.3: Obtaining an app to spy on doctors

tampered OS, which can be hard to install, while keeping existing files and apps on the device. If criminals install a tampered OS, they probably will not create a virtual microphone but use the microphone to send data all the time. After all, this approach would give more data.

9.1.5 TAMPERED EXTERNAL MICROPHONE

The latter approach in the attack tree is to tamper with the firmware of an external microphone. For this to succeed, the criminals would first need to tamper with the firmware and be able to let it send audio data. They could let it send data to a planted Bluetooth receiver, assuming it supports Bluetooth. The other way would be sending it via the device, but then some companion app is probably needed. The attackers will also have to find a way to let the doctors use the microphone. They could try to sell it relatively cheaply to doctors or switch a doctor's microphone with a tampered one.

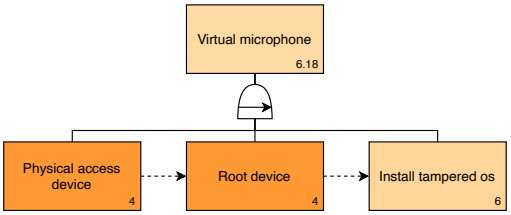

Figure 9.4: Creating a malicious virtual microphone

9.2 MITIGATIONS

Again, we propose mitigations to increase the effort values of the criminals attack tree (fig. C.9). Table 9.1 shows all mitigations we propose in this chapter and which stakeholders can or are responsible for them. This section will cover steps four and five mentioned in chapter 7. We will re-use mitigation M5 *'Add extra encryption layer'* and mitigation M6 *'Check signature app'* of chapter 8. Mitigations M1 to M4 do not apply to this tree. They are focused on securing the development process, and no node in the criminals attack tree makes use of that.

Mitigation M5 can be used in the 'collect connection data' part of the tree, where a node is added to reflect the encryption. As in chapter 8, we assume that if criminals have broken the https connection, they probably have done this by stealing keys. If they were able to get those keys, there is a high chance they can get keys from the custom encryption. So the value assigned to the node is 3.

Mitigation M6 can again be used to make it harder to tamper with the app. As with the insider attack tree, a node is added to reflect this mitigation with effort value 3.

M7 *Obfuscate code*

As mentioned before, in section 5.3.2, preventing attackers from tampering with the app is not possible. It can be made harder to do, which may deter some attackers. One relatively easy way is to obfuscate code. Obfuscating code makes it harder for the attacker to understand the decompiled code, which could slow them down in understanding the app and making adjustments. ChipSoft already did some tests with obfuscating and found it did not increase the effort to decompile a lot. Therefore, the node 'decompile code' is updated from effort value 2 to 3. Stakeholders ChipSoft and the developers are responsible for this mitigation.

#	DESCRIPTION	STAKEHOLDERS*
M5	Add extra encryption layer	S1,S2
M6	Check signature app	S1,S2
M7	Obfuscate code	S1,S2
M8	Block other apps from using the microphone	S1,S2
M9	Use a whitelist for apps	S4,S5
M10	Block external microphones	S1,S2,S4,S5
M11	Use an encrypted external microphone	S1,S2,S4,S5

* S1: ChipSoft, S2: Developers, S4: The hospital(s), S5: The medical specialist(s)

Table 9.1: The proposed mitigations and which stakeholders are or can be responsible for them.

M8 *Block other apps from using the microphone*

In chapter 5, we talked about the use of a microphone by two apps on Android and iOS. If the app could prevent other apps from using the microphone at the same time, spying apps could be blocked. If blocking is possible will depend on how advanced the spying app is. On Android, if it uses the microphone in a 'normal' way, using it in ChipSoft's app would effectively block it. If the spy app uses an accessibility service or some other trick, this will not work. For iOS, it is hard to say for sure if the microphone can be blocked. Again, stakeholders ChipSoft and the developers are responsible for this mitigation.

For now, we assume that we can block other apps from using the microphone in most scenarios, and a spying app will have to put more effort to circumvent this. A new node is added called 'Keep access to the microphone.' The effort value is hard to estimate. It will depend on how the spying app is accessing the microphone, which we do not know. We do a rough estimate and score this value on 4, which seems appropriate compared to the values of other nodes.

M9 *Use a whitelist for apps*

Many nodes in the attack tree rely on the fact that a doctor (unknowingly) installs a malicious app. An effective way to prevent this is by using a whitelist of apps a doctor is allowed to install. Tools mentioned before, like *Microsoft Intune*[46] or *Android Enterprise*[47], can be used for this. A whitelist will put an extra barrier for criminals to get their malicious code on the doctor's device. The downside is that doctor can get annoyed when they cannot install apps which are not on the list, which can be especially the case when doctors use a personal device. The responsibility for this mitigation will be mainly the hospital stakeholder, which should make sure mobile management tools are used. There could be scenarios where hospitals do not have full control over devices because personal devices are used. In that scenario, doctors can also be seen as a responsible stakeholder as they will have to comply with the whitelist.

A node is added called 'Circumvent whitelist' to parts of the attack tree where an app needs to be installed, to see the effect of this mitigation. The effort value assigned is 4, with the assumption that there may be tools to circumvent these types of whitelists, and criminals may have access to them. Again this is a rough estimate after looking at the values of other nodes.

M10 *Block external microphones*

Section 9.1.5 talks about criminals tampering with the firmware of external microphones and obtaining sensitive data. This approach can be blocked by not allowing external microphones to be used in the app.

There is no straightforward way to do this in Android. It only allows us to choose an input type[82] like a microphone, but Android will decide which device is used. Another approach will be to check if any devices are connected via Bluetooth. If so, the app could block recording audio.

On iOS, we can use the internal microphone by setting the inputDataSource of the AVAudioSession[83] to the internal microphone. We can also check which inputDataSources are available to check if any external microphone is connected and block recording in the app.

These two approaches can be executed by stakeholders Chipsoft and developers. Another way to achieve this is by making it a hospital policy not to use external microphones. In this scenario, the hospital and doctors are the responsible stakeholders.

This mitigation will decrease the usability of the app, so ChipSoft should consider what is more important. A node is added called 'Let app use the tampered microphone' with value 7, to see the effects on the attack tree. The reason for this high value is that it will be hard to connect an external microphone

over Bluetooth but hide it in the list of devices. We also assume here that the tampered microphone is connected with Bluetooth.

If the upcoming mitigation M11 is also used, we assume this mitigation blocks all microphones except the encrypted microphones. The effort value of the added node will be 6 instead of 7 as the criminals now have the chance to tamper the encrypted microphones. Using both mitigations also adds an extra node with value 6 at the virtual microphone part of the three, because attackers must find a way to let the doctor use their virtual microphone. They can do this by making the virtual microphone appear as a whitelisted microphone, but a general virtual microphone will not work anymore.

M11 *Use an encrypted external microphone*

A defense that could help against tampering and a spying app is using a headset with encryption like proposed by Boruchinkin[84]. The idea is that the headset has a chip to encrypt the audio data, and the phone merely passes it to the server. The phone will not be able to decrypt the data which protects from attacks via a compromised phone. If criminals managed to create a tampered app, they will now also have to ensure that another microphone is used instead of the encrypted one. A spying app will not be able to use the data from the microphone as it is encrypted.

A node is added with value 3 to the tampering part of the attack tree. This node represents the effort needed to prevent the use of the encrypted microphone and instead use another (unencrypted) microphone. One way to achieve this would be to disable Bluetooth, forcing the use of an internal microphone. Doing this would be relatively easy, but criminals will need to know that it is needed, so 3 seems appropriate. With this mitigation, we automatically also implement mitigation M5, so these effects also apply.

This mitigation requires the cooperation of multiple stakeholders. ChipSoft and its developers need to ensure the use of this microphone in the code. Hospitals will have to supply the microphone, and doctors will have to use it.

This mitigation is not compatible with mitigation M10 where we block external microphones. Here we want to force the use of an external microphone but only a specific one. Therefore, the effects of Mitigation M10 differ when this mitigation is also applied. In the previous section, we already mentioned what these effects are.

DESCRIPTION	NONE	M5	M6	M7	M8	M9	M10	M11	ALL
Man in the middle attack	5.26								5.26 (+0.00)
Collect connection data	7.01	+0.01						+0.01	7.02 (+0.01)
Tamper with app	4.24		+0.20	+0.14		+0.51		+0.20	5.04 (+0.80)
Use spying app	3.68				+0.80	+0.80		+0.34	5.00 (+1.32)
Virtual microphone	6.26								6.77 (+0.51)
Tampered external microphone	6.34						+1.02		6.81 (+0.46)
Root node	**3.68**				**+0.56**	**+0.80**		**+0.34**	**5.00 (+1.32)**

Table 9.2: Some high-level nodes and the root node of the criminals attack tree and their effort value per applied mitigation. Higher numbers mean more effort and are therefore better.

9.3 RESULT MITIGATIONS

Table 9.2 shows the effects of each mitigation on the effort values in the attack tree. The first thing to note is that the starting values are relatively high, except for ‘tampering with the app’ and ‘using a spying app.’ The effects of the mitigations are all below one except one, which can be expected because a logarithmic scale is used: the higher the starting value, the more effect a mitigation needs to have to increase the value in the attack tree.

The table also shows that the mitigations do not affect the ‘man in the middle attack’ node. There are only so many possibilities to counter this from within the app, and counter-measures implemented on the network and back-end server will have more effect. Investigating these counter-measures is out of scope of this report. The ‘Virtual microphone’ node only shows an increase when all mitigations are applied. The reason being that mitigation M10 and mitigation M11 are applied at the same time. The app will force to use a specific external microphone, which makes it less easy for attackers to let the doctor use a malicious virtual internal microphone.

The mitigations have the most effect on the two weakest nodes, which was the primary goal. Both are now above 5, which increases the root node with 1.32. Meaning the criminals will have to make more than three times the effort to get the sensitive data.

9.4 CONCLUSION

In this chapter, we discussed the attack tree for the criminal hacker group profile. We noticed that its weak points were mainly tampering of the app and using a spying app. We proposed seven mitigations, re-using two from the last chapter. ChipSoft and its developers can apply most of these mitigations. Applying these mitigations allowed us to increase the effort value with more than one, meaning it will be more than three times harder for criminals to obtain the audio data. The most effective mitigations are blocking the use of the microphone by other apps (M8) and using a whitelist for apps(M9). Therefore, we advise to applying these two first. Blocking apps from using the microphone can be done by ChipSoft and is relatively cheap. Whitelisting will be the responsibility of the hospital and its specialists. Mitigations M5 to M7 can be done because they are relatively cheap, but their effect is limited. The same goes for mitigation M10, blocking external microphones, which should be cheap to implement but will only help against tampered external microphones while having an impact on usability. Mitigation M11, using an encrypted microphone, also has a decent effect, but will cost much money for the hospital and also decreases usability.

With the last two chapters, we were able to answer research questions RQ3 to RQ5. In the next part, we will discuss the results, conclude the report, and look at future work.

PART IV

CONCLUSION

CHAPTER 10

DISCUSSION

In this report, we used multiple threat modeling analysis methods to answer the research questions in chapter 2. In this chapter, we review the results. In each section we discuss one threat modeling technique and we end with the proposed mitigations. This discussion aims to help understand our results better, what are strong points and what could be improved.

10.1 STRIDE

In chapter 5, we used STRIDE to answer research question RQ1. The research question was ‘What are possible vulnerabilities and threats for an audio recording feature in a mobile app?’. We did find threats and vulnerabilities, but not all. The number of threats found depends mainly on two factors. The first is how correct the data flow diagram is, and the second, if threats fall in one of the STRIDE categories. The data flow diagram was reviewed by the security experts from ChipSoft and the supervisors. Therefore, we think that it is a realistic diagram and should expose most threats. Still, we only created a diagram of the system, so threats that are not directly linked to components in the system are missed. An example is the threat that employees will add malicious code to the code base, which we later found with attack trees. We can conclude that the results cover most threats, but not all. The threats that we did find were a valuable resource when creating the attack trees in chapters 8 and 9. They helped understand how attackers could achieve their goals.

10.2 ATTACKER PROFILES

Research question RQ2, ‘What are possible attacker profiles?’, was answered in chapter 6. We found seven profiles with the help of brainstorming sessions and literature. The literature ensured we got at least the more generic profiles and the brainstorm session helped to get more specific profiles. Still, the profiles could be improved with data of real life attacks. This data can be obtained from attacks done to ChipSoft and hospitals. Data can help prioritize profiles and understand more about their motives, skills and resources. The found profiles can be considered a good starting point, and helped in assigning values to nodes in the attack trees of chapters 8 and 9.

10.3 ATTACK TREES

We used attack trees to find attack scenarios and the probabilities of them happening to answer research questions RQ3 and RQ4 in chapters 8 and 9. We found some initial results, but the trees can be improved. The current trees do show some common attack approaches but are not complete. In general, attack trees will never be complete as they model a world with unlimited ways to attack a system. Attack trees should also be maintained and updated as the system changes, and new ways to attack it are discovered.

It is also important to be aware that the assigned effort values are educated guesses. Improvements can be made by trying to use data to find these values. It will be challenging to find this data and to combine it into an effort value that is based on multiple attributes. Choices have to be made on how much weight each attribute has. So even with data, it will be no exact science. We assigned values relative to get better results. With this technique, we could find at least the weakest points reasonably accurate, even if the absolute values from nodes are not that accurate. In chapter 7, we mentioned effort values are mainly based on Android as iOS’s documentation was less comprehensive. Still, the current trees try to support both platforms. An improvement would be to create separate trees or at least different effort values for each platform.

When applying the mitigations to the trees, we also discovered that attack-defense trees might be more appropriate to model this. We also did not look at the impact of attacks with our trees. For example, an insider adding a backdoor to the app will have way more impact than one hacking the phone of one doctor. The impact should be considered by ChipSoft when prioritizing mitigations.

10.4 MITIGATIONS

The results from STRIDE and the attack trees helped us define mitigations. The effects of these mitigations on the attack trees are educated guesses, as with the effort values in general. In future work, it may be interesting to use attack-defense trees, instead of updating the original attack tree. These can incorporate our mitigations, which may lead to new insights. The advice for which mitigations to apply first are based on the found effectiveness. Before deciding which mitigations to apply, ChipSoft should also look at the impact and costs of mitigations. The impact of the attacks a mitigation helps to defend should also be considered.

CHAPTER 11

CONCLUSION

In this report, we did a threat model analysis of an audio recording on mobile healthcare applications. During our research, we considered iOS and Android. Unfortunately, the documentation of iOS was not as extensive as that from Android. Therefore, the results are better applicable to Android, while iOS could use more research.

We devided our report in four parts. After the introduction in the first part, we started by defining the scope and goals of this thesis in chapter 2. Methodologies were selected to answer each of the research questions. We then defined the requirements for designing an audio recording feature in the mobile app from ChipSoft in chapter 3. We looked for functional but also security requirements to answer background question BQ1. In chapter 4, we did an extensive literature study to learn about threat model analysis methodologies, tools, and related work. We found a multitude of methods, from which we could use a few to answer our research questions. With this chapter, we answered background question BQ2.

In the next part, we used the STRIDE methodology to get an overview of threats and vulnerabilities of the system to answer research question RQ1 in chapter 5. We created a data flow diagram of the system, and we considered all the STRIDE types for each component of the diagram. We found 19 threats and discussed existing mitigations to counter those threats. We also discovered that STRIDE is useful to find threats that can be coupled to specific parts of the system but does not find threats in the development process. Still, the result provided a good overview and resource for the other threat modeling techniques.

In chapter 6, we found attacker profiles, which are needed to create attack trees, and answered research question RQ2. We used brainstorming sessions, and the literature to elicit them. We looked at multiple attributes for each profile, including skill, resources, and access. With the help of the security experts from ChipSoft, we selected two profiles, which seemed interesting, to use for our attack trees. The selected profiles were the dissatisfied (ex)employee or insider and the criminals group.

#	DESCRIPTION
M1	Secure process of adding code
M2	Decrease access to (security-sensitive) code
M8	Block other apps from using the microphone

Table 11.1: The mitigations we advice to do first, as they are most effective and can be executed by ChipSoft.

In the third part, we created two attack trees and proposed mitigations. The attack trees showed possible attack scenarios for the two selected attacker profiles, answering research question RQ3. The insider attack tree showed it was relatively easy to insert malicious code in the code-base. The criminals attack tree had tampering with the app as the weakest point. The trees provide a good start but should be maintained and extended as the system evolves. We graded each node with an effort value to find the scenarios with the highest probability to answer research question RQ4. We tried to score the nodes relative to each other, as the values are educated guesses. Scoring them relative ensures we can find accurate weak points in the trees, although the value itself may not be that precise. We validated the trees with security experts from ChipSoft to ensure they were realistic.

With the help of chapter 5 we proposed mitigations to counter weak nodes in the attack trees, answering research question RQ5. We updated the attack trees with the mitigations to see what the effects would be on the probabilities of attack scenarios. With mitigations, we were able to make it three times as hard for an insider to obtain the audio data. For criminals, it became more than three times as hard. We advised applying three mitigations first, as they are most effective and can be realized by ChipSoft. Table 11.1 lists them. Our advice is based on the results of calculating the effects with our attack trees. We did not look at the costs of mitigations, which should be known before ChipSoft can decide on which mitigation to apply.

Using multiple complementary threat model analysis methods, gave us a broad view of possible security issues. This broad view is needed because security affects multiple aspects of software design, including the development process, implementing the code, and running the application in production. We showed how we could prevent adversaries from obtaining privacy-sensitive audio data and answered our main question. This report should help ChipSoft complying to the standards mentioned in section 4.3. In the next section, we discuss what can be done in the future.

CHAPTER 12

FUTURE WORK

As mentioned before, in chapter 10, the threat analysis can be improved. In this chapter, we propose ideas that ChipSoft could do in the future. Many of these ideas focus on improving the threat analysis. ChipSoft could also decide the analysis suffices and implement the audio feature, discussed in the last section.

12.1 INCREASE THE SCOPE OF THE ANALYSIS

In this report, we focused on the audio recording in the app. However, attackers may use multiple attack vectors in different systems. These attack scenarios will only be found if the scope of the analysis is increased. For example, the analysis could also include back-end services. Another reason to increase the scope of the analysis is that many threats and mitigations are applicable on a bigger scope. Many of the proposed mitigations are also useful for the whole app, not only the audio recording feature. Many found threats and attack scenarios are also not exclusive to this feature. Therefore, we can reuse the results from this report for a bigger scope.

12.2 DO A DATA ANALYSIS OF SECURITY INCIDENTS

ChipSoft could research security incidents at hospitals and the company itself. Data should be acquired from hospitals, which could then be researched like Moeckel[21] has done (explained in section 4.1.3). For each incident, researchers could look at what type of attack was done and who did it. Knowing which attacks were done will show which types have the highest probability. ChipSoft could use that information to apply mitigations. Knowing these types can also be used to improve and create attack trees. Knowing who made the attacks will help create realistic attacker profiles for the healthcare domain. Again, this information can

help improve attack trees. Even without attack trees, this information can help ChipSoft understand where to focus their effort on securing the system.

12.3 IMPROVE ATTACK TREES

The created attack trees can still be improved and updated. In a changing environment, attack trees are never finished: new attack scenarios may arise as software changes. Effort values may also change as attackers create new tools. The following points can be improved:

- **Extend nodes**: the trees can be extended with more attack scenarios.
- **Improve effort values**: the effort values are educated guesses and could be made more accurate. Data should be used to do so.
- **Use attack defense trees**: the trees may be upgraded to attack defense trees, mentioned in section 4.1.2. These types of trees may be a better fit, especially when proposing mitigations.
- **Consider impact**: in this report, we did not consider the impact an attack has. An insider adding malicious code will have a bigger impact than a criminal infecting one device. This information can be imported to decide which mitigations have priority.

Attack trees can be hard to maintain because they can get quite large. It would be advisable to create or buy software that supports this process.

12.4 RESEARCH MITIGATIONS

This report gives an idea of which mitigations are effective. However, the updates we applied to the attack trees in chapters 8 and 9 were educated guesses, as were all effort values. As proposed in the previous section for the attack tree, these values could be made more accurate. Ideally, data can be used for this.

Another thing that can be improved to the proposed mitigations is researching how high the costs and impact are. A mitigation may be too costly to be feasible, for example, giving every doctor an encrypted microphone. Other mitigations may have a big impact on usability or software development. Knowing these two properties of mitigations will help to prioritize them.

12.5 IMPLEMENT THE AUDIO RECORDING FEATURE

The report should give a good idea of possible threats and vulnerabilities and how to mitigate them. ChipSoft could decide this is sufficient to start implementing the audio record feature. The information from the STRIDE analysis in chapter 5 and the proposed mitigations in part III can be used to design the feature. If the information in these chapters is not enough, it should at least point in the right direction. This design could then be implemented and tested on security by the experts from ChipSoft.

BIBLIOGRAPHY

[1] C Lee Ventola. “Mobile devices and apps for health care professionals: uses and benefits”. In: *Pharmacy and Therapeutics* 39.5 (2014), p. 356.

[2] Marieke van Twillert. *ChipSoft marktleider van ziekenhuis-epd’s*. Accessed: 4 november 2019. Apr. 2018. URL: https://www.medischcontact.nl/nieuws/laatste-nieuws/artikel/chipsoft-marktleider-van-ziekenhuis-epds.htm.

[3] *HiX*. Accessed: 27 august 2019. URL: https://www.chipsoft.nl/oplossingen/1/HiX.

[4] *Google Scholar*. Accessed: 30 januari 2020. URL: https://scholar.google.com/.

[5] *Scopus*. Accessed: 30 januari 2020. URL: https://www.scopus.com/.

[6] Adam Shostack. “Experiences Threat Modeling at Microsoft.” In: *MODSEC@ MoDELS*. 2008.

[7] *Documentation for app developers*. Accessed: 30 january 2020. URL: https://developer.android.com/docs.

[8] *Apple Developer Documentation*. Accessed: 30 januari 2020. URL: https://developer.apple.com/documentation.

[9] *Threat model*. Accessed: 5 november 2019. Oct. 2019. URL: https://en.wikipedia.org/wiki/Threat_model.

[10] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[11] Nataliya Shevchenko et al. “Threat Modeling: a Summary of Available Methods”. In: *no. July* (2018).

[12] Florian Arnold et al. “Sequential and parallel attack tree modelling”. In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2014, pp. 291–299.

[13] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. “Quantitative attack tree analysis via priced timed automata”. In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2015, pp. 156–171.

[14] *Data-flow diagram*. Accessed: 6 november 2019. Sept. 2019. URL: https://en.wikipedia.org/wiki/Data-flow_diagram.

[15] Peter Stavroulakis and Mark Stamp. *Handbook of information and communication security*. Springer Science & Business Media, 2010, p. 13.

[16] Bruce Schneier. *Academic: Attack Trees - Schneier on Security*. Accessed: 28 august 2019. 1999. URL: https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

[17] Sriram Krishnan. “A Hybrid Approach to Threat Modelling”. In: (Feb. 2017).

[18] Sjouke Mauw and Martijn Oostdijk. “Foundations of attack trees”. In: *International Conference on Information Security and Cryptology*. Springer. 2005, pp. 186–198.

[19] Barbara Kordy et al. “Foundations of attack–defense trees”. In: *International Workshop on Formal Aspects in Security and Trust*. Springer. 2010, pp. 80–95.

[20] Barbara Kordy et al. “Attack–defense trees”. In: *Journal of Logic and Computation* 24.1 (2014), pp. 55–87.

[21] Caroline Moeckel. “Examining and Constructing Attacker Categorisations: an Experimental Typology for Digital Banking”. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. 2019, pp. 1–6.

[22] Larisa April Long and Egan Hadsell. “Profiling hackers”. In: *SANS Institute Reading Room* 26 (2012).

[23] Nationaal Coördinator Terrorismebestrijding en Veiligheid. *Cybersecuritybeeld Nederland 2019*. 2019. URL: https://www.ncsc.nl/binaries/ncsc/documenten/publicaties/2019/juni/12/cybersecuritybeeld-nederland-2019/CSBN2019.pdf.

[24] Jane Cleland-Huang. “How well do you know your personae non gratae?” In: *IEEE software* 31.4 (2014), pp. 28–31.

[25] Kim Wuyts, Riccardo Scandariato, and Wouter Joosen. “Empirical evaluation of a privacy-focused threat modeling methodology”. In: *Journal of Systems and Software* 96 (2014), pp. 122–138. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2014.05.075. URL: http://www.sciencedirect.com/science/article/pii/S016412121400137X.

[26] Kim Wuyts and Wouter Joosen. “LINDDUN privacy threat modeling: a tutorial”. In: *CW Reports* (2015).

[27] Peter Mell, Karen Scarfone, and Sasha Romanosky. “Common vulnerability scoring system”. In: *IEEE Security & Privacy* 4.6 (2006), pp. 85–89.

[28] *CVSS v3.1 Specification Document*. Accessed: 8 november 2019. June 2019. URL: https://www.first.org/cvss/v3.1/specification-document.

[29] *Common Attack Pattern Enumeration and Classification*. Accessed: 8 november 2019. URL: https://capec.mitre.org/.

[30] Bernard Mueller, Sven Schleier, and Jeroen Willemsen. *Mobile Security Testing Guide*. OWASP, 2019. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/.

[31] *OWASP*. Accessed: 8 november 2019. URL: https://www.owasp.org/index.php/Main_Page.

[32] OWASP. *OWASP Cheat sheet series*. 2019. URL: https://cheatsheetseries.owasp.org/.

[33] Microsoft. *Microsoft Threat Modeling Tool*. Version 7.1.60702.1. July 2, 2019. URL: https://docs.microsoft.com/en-gb/azure/security/develop/threat-modeling-tool.

[34] OWASP. *Threat Dragon*. Version v0.1.26. May 17, 2019. URL: https://threatdragon.org.

[35] Izar. *izar/pytm*. Accessed: 3 september 2019. June 2019. URL: https://github.com/izar/pytm.

[36] Barbara Kordy et al. “ADTool: security analysis with attack–defense trees”. In: *International conference on quantitative evaluation of systems*. Springer. 2013, pp. 173–176.

[37] *ThreatModeler Software Inc*. Accessed: 8 november 2019. URL: https://threatmodeler.com/.

[38] *Threat Modeling Methodologies: What is VAST?: ThreatModeler*. Accessed: 8 november 2019. May 2019. URL: https://threatmodeler.com/threat-modeling-methodologies-vast/.

[39] *EU data protection rules*. Accessed: 12 november 2019. July 2019. URL: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules/eu-data-protection-rules_en.

[40] *Seven steps for businesses to get ready for the General Data Protection Regulation*. Publications Office, 2018. URL: https://ec.europa.eu/commission/sites/beta-political/files/ds-02-18-544-en-n.pdf.

[41] Accessed: 12 november 2019. URL: https://www.chipsoft.nl/hix-abc/artikel/16/Certificering.

[42] Mohamed Abomhara, Martin Gerdes, and Geir M Køien. "A stride-based threat model for telehealth systems". In: *Norsk informasjonssikkerhetskonferanse (NISK)* 8.1 (2015), pp. 82–96.

[43] Ahmad Almulhem. "Threat modeling for electronic health record systems". In: *Journal of medical systems* 36.5 (2012), pp. 2921–2926.

[44] Matteo Cagnazzo et al. "Threat modeling for mobile health systems". In: *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE. 2018, pp. 314–319.

[45] Florian Kammueller. "Formal modeling and analysis of data protection for gdpr compliance of iot healthcare systems". In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2018, pp. 3319–3324.

[46] *Microsoft Intune*. Accessed: 9 november 2019. URL: https://www.microsoft.com/en-us/microsoft-365/enterprise-mobility-security/microsoft-intune.

[47] *Android Enterprise*. Accessed: 9 november 2019. URL: https://www.android.com/enterprise/.

[48] Skip Hovsmith. *Mobile API Security Techniques*. Accessed: 4 september 2019. Jan. 2017. URL: https://hackernoon.com/mobile-api-security-techniques-682a5da4fe10.

[49] Bulat Saifullin et al. "Analysis of Android Camera Spoofing Techniques". In: *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel / Distributed Computing (SNPD)*. IEEE. 2018, pp. 10–14.

[50] *MediaStore.Audio.Media*. Accessed: 6 september 2019. URL: https://developer.android.com/reference/android/provider/MediaStore.Audio.Media.html#RECORD_SOUND_ACTION.

[51] Saurik. *Cydia Substrate*. URL: http://www.cydiasubstrate.com/.

[52] Ole André V. Ravnås. *Frida*. Version 12.6. URL: https://www.frida.re/.

[53] rovo89. *Xposed*. URL: https://repo.xposed.info/.

[54] John Wu. *Magisk*. URL: https://github.com/topjohnwu/Magisk.

[55] Rockbruno. *Swiftshield*. May 2019. URL: https://github.com/rockbruno/swiftshield.

[56] *Shrink, obfuscate, and optimize your app*. Accessed: 9 september 2019. URL: https://developer.android.com/studio/build/shrink-code#obfuscate.

[57] *SafetyNet Attestation API*. Accessed: 9 september 2019. URL: https://developer.android.com/training/safetynet/attestation.html.

[58] *iOS Security*. Accessed: 9 september 2019. URL: https://www.apple.com/business/docs/site/iOS_Security_Guide.pdf.

[59] Ben Seri and Gregory Vishnepolsky. "The dangers of bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern bluetooth stacks". In: *Armis, Tech. Rep.* (2017).

[60] *Vendor-specific AT commands*. Accessed: 18 december 2019. URL: https://developer.android.com/guide/topics/connectivity/bluetooth#AT-Commands.

[61] *Core Bluetooth*. Accessed: 18 december 2019. URL: https://developer.apple.com/documentation/corebluetooth.

[62] *AudioManager*. Accessed: 9 november 2019. URL: https://developer.android.com/reference/android/media/AudioManager.html.

[63] *AVCaptureDevice*. Accessed: 22 september 2019. URL: https://developer.apple.com/documentation/avfoundation/avcapturedevice.

[64] Lisa Vaas. *Siri is listening to you, but she's NOT spying, says Apple*. Aug. 2018. URL: https://nakedsecurity.sophos.com/2018/08/13/siri-is-listening-to-you-but-shes-not-spying-says-apple/.

[65] Alex Hern. *Apple contractors 'regularly hear confidential details' on Siri recordings*. July 2019. URL: https://www.theguardian.com/technology/2019/jul/26/apple-contractors-regularly-hear-confidential-details-on-siri-recordings.

[66] Lente Van Hee et al. *Google-medewerkers luisteren mee naar uw gesprekken, ook in uw huiskamer*. July 2019. URL: https://www.vrt.be/vrtnws/nl/2019/07/10/google-luistert-mee/.

[67] *Sharing audio input : Android Developers*. Accessed: 19 august 2019. URL: https://developer.android.com/preview/features/sharing-audio-input.

[68] *AVAudioRecorder*. Accessed: 22 september 2019. URL: https://developer.apple.com/documentation/avfoundation/avaudiorecorder.

[69] *AVCaptureSession*. Accessed: 22 september 2019. URL: https://developer.apple.com/documentation/avfoundation/avcapturesession.

[70] *SSLSocket*. Accessed: 16 september 2019. URL: https://docs.oracle.com/javase/7/docs/api/javax/net/ssl/SSLSocket.html.

[71] Matthew JB Robshaw. “Stream ciphers”. In: *RSA Laboratories Technical Report* (1995).

[72] *Bluetooth Core Specification*. Jan. 2019. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/.

[73] *Running a service in the foreground*. Accessed: 16 september 2019. URL: https://developer.android.com/guide/components/services.html#Foreground.

[74] *UIBackgroundModes*. Accessed: 16 september 2019. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/uibackgroundmodes.

[75] JGraph. *DrawIO dekstop*. Version 13.0.3. Apr. 29, 2020. URL: https://github.com/jgraph/drawio-desktop.

[76] *Team Foundation Version Control*. URL: https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview?view=azure-devops#team-foundation-version-control.

[77] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. “This POODLE bites: exploiting the SSL 3.0 fallback”. In: *Security Advisory* (2014).

[78] *SSLSocket*. Accessed: 8 april 2020. URL: https://developer.android.com/reference/javax/net/ssl/SSLSocket.

[79] *Choosing the right version control for your project*. URL: https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/comparison-git-tfvc?view=azure-devops.

[80] *Git*. URL: https://git-scm.com/.

[81] *How We Use Git at Microsoft*. URL: https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/use-git-microsoft.

[82] *MediaRecorder*. Accessed: 1 may 2020. URL: https://developer.android.com/reference/android/media/MediaRecorder.

[83] *AVAudioSession*. Accessed: 1 may 2019. URL: https://developer.apple.com/documentation/avfoundation/avaudiosession.

[84] A Yu Boruchinkin. “Secure voice communication system with hardware encryption of data on hands-free headset”. In: *Proceedings of the 8th International Conference on Security of Information and Networks*. 2015, pp. 76–79.