***Secure communication between Erlang nodes***

**Dissertation**

**By**

**Benjamin Adjei**

**This thesis is submitted in part of fulfilment of BSc Computer Science at Middlesex University, 2019**

Abstract

One of the major problems with Erlang is that is extremely difficult to secure personal data in Erlang. This invariably leads to problems with personal data being leaked or hacked. The problem to be solved is to secure personal data in Erlang so that it can not be leaked or hacked. The problem was addressed by creating 2 servers: a cloud server to hold the database with the user’s name, their secret password and their personal data, and an authentication server to check if the password a user types in matches the user’s password in the database. If it does, the user is allowed to access the data. If it does not, the user is shown an error message that says ‘error! Wrong password!’ The project was evaluated by someone pretending to be a hacker trying to hack my data. The conclusions that could be drawn from the work is that there is a way to secure data in Erlang so that it can not be leaked or hacked.

Table of Contents

Abstract1

Chapter 1: Introduction3-4

Chapter 2: Background and literature review5-10

Chapter 3: Requirements specification11

Chapter 4: Analysis and design12

Chapter 5: Implementation13-14

Chapter 6: Testing and results15

Chapter 7: Demonstration/Evaluation16-23

Chapter 8: Conclusions and further work24

Chapter 9: User guide25

Chapter 10: Bibliography26-27

Chapter 11: Appendix28

Chapter 1: Introduction

A clear definition of the problem to be addressed is that the data of patients could be shared which would violate the confidentiality of the data between doctors and patients. The aim of the project is to develop a prototype system in Erlang for a cloud-based patient information system which will help keep patient information secure, making sure their confidential data cannot be leaked or hacked. The objectives of the project are to further secure the data by having labels added to them. Last but not least, the implementation of nodes will be experimented on. The deliverables of this project are as follows:

(0) A literature review. This is “a search and evaluation of the available literature in your given subject or chosen topic area” (The Royal Literary Fund, 2018). This quote is the definition of a literature review from the Royal Literary Fund. This should take approximately 6 weeks to finish.

(1) An Erlang prototype with 3 processes to simulate the patient, the hospital and the cloud. This should take approximately a month to finish this. A prototype is “the first example of something, e.g. a machine or other industrial product, from which all later forms are developed” (Dictionary, 2018). This quote is the definition of a prototype according to the Cambridge English Dictionary. An Erlang process is “a completely independent unit of execution” (Staticshin.com, 2018). This quote is an explanation of an Erlang process from Akshat Jiwan Sharma.

(2) A more sophisticated implementation of adding security by labelling data. This should take approximately 6 weeks to finish. Implementation is “the act of putting a plan into action or of starting to use something” (Dictionary, 2018). This quote is a definition of implementation from the Cambridge English Dictionary.

(3) Using techniques like crypto protocols available in Erlang, e.g., SSL or Ipsec. This should take approximately a month to finish. SSL is “a standard security technology for establishing an encrypted link between a server and a client—typically a web server (website) and a browser or a mail server and a mail client.” (DigiCert, 2018) This quote is a definition of SSL from the Cambridge English Dictionary.

(4) Running and testing the implementation of different computers using Erlang nodes.

This should take approximately 6 weeks to finish. A computer is “a machine or device that performs processes, calculations and operations based on instructions provided by a software or hardware program.” (Techopedia.com, 2018) This quote is a definition of a computer from Technopedia.

In Chapter 2 of the report, you will find the definitions of background words and my literature review where I reviewed all the existing literature related to my project.

In Chapter 3 of this report, you will find the requirements specification (also known as a software requirements specification), which is a “comprehensive description of the intended purpose and environment for software under development. It fully describes what the software will do and how it will be excepted to perform” (SearchSoftwareQuality, 2019), for my project.

In Chapter 4 of the report, you will find the analysis and design for my project.

In Chapter 5 of this report, you will find the implementation of the software for my project.

In Chapter 6 of the report, you will find the testing and results of the testing of the software for my project.

In Chapter 7 of this report, you will find the documentation on the software and the evaluation of its performance.

In Chapter 8 of the report, you will find the conclusions to this report and further work.

In Chapter 9 of this report, you will find a guide on how to use the software.

In Chapter 10 of the report, you will find all the sources used for research.

In Chapter 11 of this report, you will find appendices.

Chapter 2: Background and literature review

**Background**

## Erlang

This is “a programming language that can run more than one process at a time for open distributed telecommunication (OTP) systems developed by [the] Ericsson corporation. It implements the actor paradigm by providing message passing as strategy for communication between several actors implemented as processes. Processes run fully parallel in Erlang. Each process has a mailbox where arriving messages are stored. The programmer can use pattern matching for message selection. Hence, the behaviour of an actor is controllable. If a process needs an answer, its process identifier (PID) must be passed through the message. Since memory sharing does not exist, neither locks nor mutexes are necessary. The code is grouped in modules which are referred to by their name. So modulename: functionname(args). starts a function from a specific module. A process is created by the spawn command supplying it with the process’ function and initial arguments. Erlang supports also named processes. Using register (Name, PID), the PID is registered in a global process registry and the process can be called by its name.

PID = spawn (Func, Args),

PID! Message,

Func(Args)…

receive

Pattern1 [when Guard1] 🡪 Expression1;

Pattern2 [when Guard2] 🡪 Expression2;

…

end.

Above, the basics of distribution in Erlang are shown. First, a new process which runs the function Func is started. Then, a Message to the new process is sent which is identified by PID. The function Func implements several patterns for incoming messages. Now, the system tries to match the arrived message against Pattern1 (and the guard if it exists). In case of success, Expression1 is evaluated. If the first pattern fails, the second will be used and so on. Another fundamental feature of Erlang is the single assignment, as in algebra, meaning that Erlang variables are immutable. The main data types are (untyped) lists, records, called tuples, for example {green, apple} and atoms which represent different non-numerical constant values. Any lower-case name is interpreted as an atom, any term that starts with a capital letter is a variable. In addition, there are modules for interoperability to other programming languages like C, Java or databases.” [4]

## Distributed programming

This is a “model in which components of a software system are shared among multiple computers to improve efficiency and performance.” (WhatIs.com, 2018) This quote is a definition of distributed programming from the WhatIs.com website.

## Security

This is either “the state of being free from danger or threat” or “a thing deposited or pledged as a guarantee of the fulfilment of an undertaking or the repayment of a loan, to be forfeited in case of default.”  (Oxford Dictionaries | English, 2018) This quote is a definition of security from the Oxford Dictionary.

## Privacy in healthcare applications

This is “respect for the confidential nature of the therapist-patient relationship.” (TheFreeDictionary.com, 2018) This quote is a definition of privacy in healthcare applications from The Free Dictionary website.

## Crypto protocols

These are “abstract or concrete protocols that perform security-related functions and apply cryptographic methods, often as sequences of cryptographic primitives. They are widely used for secure application-level data transport.” (En.wikipedia.org, 2018) This quote is a definition of crypto protocols from Wikipedia.

## Erlang nodes

These are “self-contained Erlang system containing a complete virtual machine with its own address space and own set of processes.” (node?, 2018) This quote is a definition of Erlang nodes from Stack Overflow.

## Kerberos protocol

This is a “network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography”. Anon, *Kerberos: The Network Authentication Protocol*. Available at: https://web.mit.edu/Kerberos/#what\_is [Accessed April 1, 2019]. This quote is the definition of the Kerberos protocol from web.mit.edu.

**Literature review**

The purpose of this literature review is to provide a critical evaluation of the existing literature that has been written on the topic of secure distributed programming in Erlang. The aim is to outline the development of academic theory and research that has been undertaken, in order to provide a context for the analysis of the subject. This should result in a precise definition of the problem to be investigated, which will provide structure and rationale for the research. The review will start with an overview of secure distributed programming in Erlang. It will then focus on the specific ways in which Erlang can be made secure.

The research question that the literature review helps to define is how patients’ confidential data cannot be leaked or hacked. This is being asked because people are interested in developing a prototype in Erlang for a cloud-based patient information system which will help keep patient information secure, making sure their confidential data cannot be leaked or hacked. Ioannis Papagiannis, Matteo Migiliavacca, David M. Eyers, Brian Shand, Jean Bacon and Peter Pietzuch have done something similar in their paper which is called ‘Enforcing User Privacy in Web Applications in Erlang’ (2019). In the paper, they implement information flow control to web applications in Erlang. They do this by using ‘Erlang’s lightweight process primitives to create fresh instances of components cheaply and thus to maintain isolation between components. This approach enables the building of massively concurrently web applications that enforce IFC constraints with a low performance impact.’ (Papagiannis et al., 2019).

Information flow control ‘tracks how information flows through the program during execution to make sure the program handles the information securely. Secure information flow is comprised of 2 related aspects: information confidentiality and information integrity – intuitively pertaining to the reading and writing of the information. The prevailing basic semantic notion of secure information flow is non-interference, demanding independence of public (or, in the case of integrity, trusted) output from secret (or, in the case of integrity, untrusted) input. This document gives an account of state-of-the-art in confidentiality and integrity policies and their enforcement with a systematic formalisation of 4 dominant formulations of noninterference: termination-insensitive, termination-sensitive, progress-insensitive, and progress-sensitive, cast in the setting of 2 minimal while languages. ‘ (Hedin and Sabelfeld, 2019).

Information flow control is achieved by ‘associating data with tags. This taints components that receive it, potentially restricting future flows of data to and from these components’ (Lsds.doc.ic.ac.uk, 2019).

Kenji Rikitake and Koji Nakao have also done something similar in their paper which is called ‘Application Security of Erlang Concurrent System’. In this paper, they ‘first survey the current security functionality in the Erlang distributed virtual machine environment. They then evaluate and propose the possible further enhancements to add the protection against wide-area Internet attacks, while maintaining the current scalability of the Erlang-based system themselves’ (Rikitake and Nakao, 2019).

Andrei Sabelfeld and Heiko Mantel have also done something similar in their paper which is called ‘Securing Communications in a Concurrent Language’. In this article, they ‘investigate the interplay between, on the one side, public, encrypted, and private (or hidden) channels of communication and, on the other side, blocking and nonblocking communication primitives for a simple multi-threaded language’ (Sabelfeld and Mantel, 2019).

Hans Svensson and Lars-Åke Fredlund have also done something similar in their paper which is called ‘Programming distributed Erlang applications: pitfalls and recipes’. In their paper, they ‘discuss a number of such pitfalls, where the semantics of communicating processes differs significantly depending if the processes reside on the same node or not, they also provide some guidelines for safe programming of distributed systems’ (Svensson and Fredlund, 2019).

My research is relevant to practice in my field. What is already known or understood about this topic is how to enforce user privacy in web applications in Erlang. My research might add to this understanding or challenge existing theories and beliefs is that user privacy in Erlang can be implemented in ways other than information flow control.

Erlang has a lot of strength in secure programming for the following reasons: it has ‘no pointer assignment, once-and-only once variable assignment, it's based on message-passing so there is minimised sharing, there is restrictive access for input/output devices and the OTP supports secure communication modules.’ (Rikitake, 2019) . However, there are security flaws in Erlang. A way ‘nodes use to communicate is using the rpc:call function. Normally, this is not encrypted which gives an attacker the ability to take the data’ (E.D Williams, 2014, Page 6).

Erlang’s security model is ‘blank’ (Learnyousomeerlang.com, 2019). This is ‘a programming language that can run more than one process at a time for open distributed telecommunication (OTP) systems developed by [the] Ericsson corporation. It implements the actor paradigm by providing message passing as strategy for communication between several actors implemented as processes. Processes run fully parallel in Erlang. Each process has a mailbox where arriving messages are stored. The programmer can use pattern matching for message selection. Hence, the behaviour of an actor is controllable. If a process needs an answer, its process identifier (PID) must be passed through the message. Since memory sharing does not exist, neither locks nor mutexes are necessary. The code is grouped in modules which are referred to by their name. So modulename: functionname(args). starts a function from a specific module. A process is created by the spawn command supplying it with the process’ function and initial arguments. Erlang supports also named processes. Using register (Name, PID), the PID is registered in a global process registry and the process can be called by its name.

PID = spawn (Func, Args),

PID! Message,

Func(Args)…

receive

Pattern1 [when Guard1] 🡪 Expression1;

Pattern2 [when Guard2] 🡪 Expression2;

…

end.

Above, the basics of distribution in Erlang are shown. First, a new process which runs the function Func is started. Then, a Message to the new process is sent which is identified by PID. The function Func implements several patterns for incoming messages. Now, the system tries to match the arrived message against Pattern1 (and the guard if it exists). In case of success, Expression1 is evaluated. If the first pattern fails, the second will be used and so on. Another fundamental feature of Erlang is the single assignment, as in algebra, meaning that Erlang variables are immutable. The main data types are (untyped) lists, records, called tuples, for example {green, apple} and atoms which represent different non-numerical constant values. Any lower-case name is interpreted as an atom, any term that starts with a capital letter is a variable. In addition, there are modules for interoperability to other programming languages like C, Java or databases’ (Fleck and Kammüller, 2010). This quote is from the paper ‘Privacy by Flexible Parameterization with Erlang Active Objects’ by Andreas Fleck and Florian Kammüller.

Chapter 3: Requirements specification

* Functional requirements

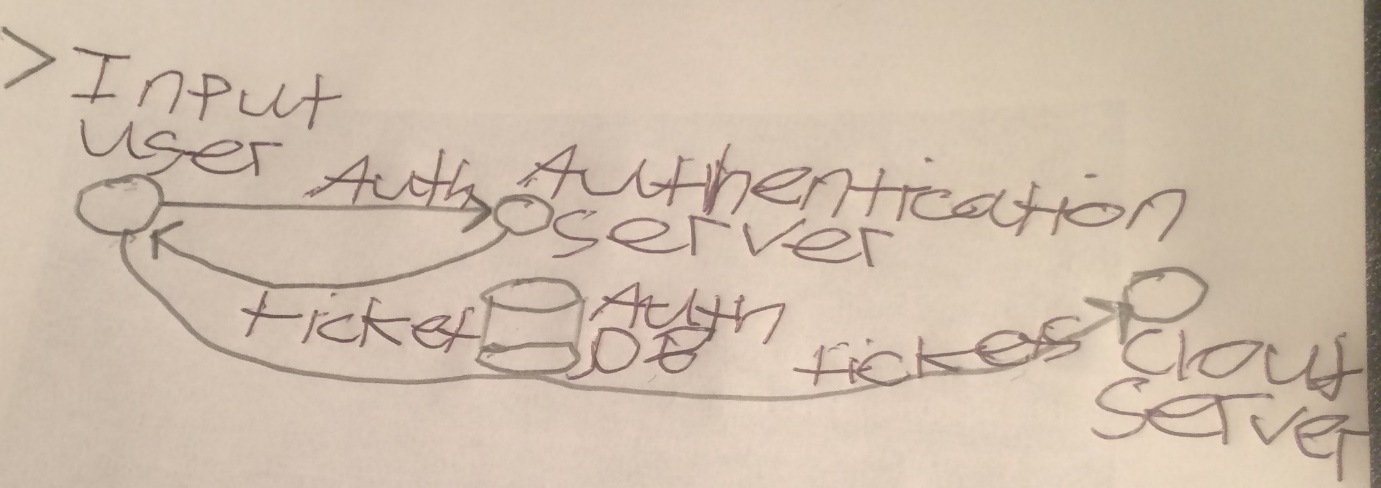
Store your name, secret password and personal data in a database by using a token to write it to the database and upload that to a cloud server.

* Read data from and write data to the database using a token
* Have an authentication server issue tokens from a tokenlist
* If the user enters the correct password, allow them to access their data using a token and delete that token from the tokenlist, otherwise tell the user that they have the wrong password
* Non-functional requirements

Stop hackers from accessing your data

Chapter 4: Analysis and design

**Server architecture**



The diagram shows a user entering their password in an authentication server. The password that the user entered matches the password stored in the authentication database and the user gets a ticket. Then, they use their ticket to access their data in the cloud server.

My 1st solution was to only store data in a single server but that wasn’t secure because a hacker can get access to a person’s data by typing their name in the Erlang shell so my 2nd solution was to improve the security by using 2 servers: one for authentication and one for storing the data. The effect of having 2 servers is that there is a layer of security between the users and their personal data. First, users enter their name and their personal data between the curly brackets in the start function of the program. Then, that data is uploaded to the cloud server using a token from the tokenlist and that same token is deleted from the tokenlist in the cloud server. Finally, if the user wants to access their data, they enter their secret password and if it matches the secret password stored in the database in the authentication server, the authentication server sends the user a ticket that allows them to access their data in the database in the cloud server. If the password that the user entered does not match the secret password stored in the database in the authentication server, an error message is displayed on the screen to the user that says, ‘wrong password!’. This is because a layer of security is added to prevent the personal data of users from being hacked or leaked.

Chapter 5: Implementation

cloud(LData, Tokenlist) ->

receive

{write, Owner, NewUpload, Token} ->

B = member(Token,Tokenlist),

if B ->

cloud([{Owner,NewUpload}|LData], del(Token,Tokenlist));

true -> cloud(LData, Tokenlist)

end;

{get, Name, Token, From} ->

B = member(Token, Tokenlist),

if B -> From ! {patient, searchval(Name,LData)},

cloud(LData, del(Token,Tokenlist));

true -> cloud(LData, Tokenlist)

end;

stop -> ok

end.

auth(UData,Tokenlist) ->

receive

{authenticate, User, Passw, From} ->

\_PW = searchval(User, UData),

if (\_PW == Passw) -> From ! {token, hd(Tokenlist)},

auth(UData, tl(Tokenlist));

true -> From ! {error,"wrong password"}

end;

stop -> ok

end.

start() ->

Server\_PID = spawn(erlangprototype\_v5, cloud,

[[{florian, 120},{ben, 80}], [42, 3, 7, 1]]),

register(server\_process, Server\_PID),

Auth\_PID = spawn(erlangprototype\_v5, auth,

[[{ben,1337},{florian,42}],[42, 3, 7, 1]]),

register(auth\_process, Auth\_PID).

The user types their name and the data they want to be secured, which uses a token from a tokenlist, where each token con only be used a certain number of times, in a database which is uploaded to the cloud server. An authentication server checks if the password the user has typed in to access their data matches their password in the authentication server. If the password they have typed in matches the password in the authentication server, it gives them a ticket which they can use to access their data in the cloud server. If they enter the wrong password, it gives them an error that says, ‘Wrong password!’ .Chapter 6: Testing and results

I tested my 1st version of my Erlang prototype by opening the Erlang shell, and where the prompt is, typing the letters e,r, and l to invoke the Erlang running system or emulator. Next, I pressed the Return key on the keyboard. Then, I made sure that I was in the directory that the program was stored in by typing the command ‘pwd(), where the prompt is,’ to display the directory that I was in on the laptop screen. I was not in the directory where the program is stored so I changed directory at run time in the emulator by using the command or built in function cd(Dir). I replaced the word ‘Dir’ with the address where the program is stored which is "C:/Users/Benjamin Adjei/Documents/Subjects and exercises year 3/Computer Science project". I replaced the forward slashes in the address wit backward slashes to change directory. Next, I compiled the Erlang prototypes by typing ‘c(modulename).’ where the prompt is. I replaced the word ‘modulename’ with the module name of my program. The module name of my first program is ‘erlangprototype’. I typed c(erlangprototype) where the prompt was and pressed the Enter key on the keyboard to compile the program. The system returned ‘{ok, erlangprototype}’ because the compilation was successful. I evaluated the exported functions in the module by entering the following expression at the Erlang shell prompt: ‘erlangprototype:start’, followed by a full stop and a carriage return. Erlang printed the result ‘true’. Then, I made 4 more versions of the program where I added more code to each version, tested each version, and fixed the error messages that were displayed on the computer screen with help from my project supervisor. The only error messages that I had was that some of my functions in my programs took more arguments than I wrote for the function and when I compiled the program with the errors in it, the compilation was not successful and an error message was printed on the compiler in the following format: modulename.erl: line number: error message.

Chapter 7: Documentation / Evaluation

-module(erlangprototype\_v4).

-export([cloud/1, start/0, stop/0, write/2, read/1, searchval/2]).

searchval(\_,[]) -> 0;

searchval(Name, [{Name,Val}|\_]) -> Val;

searchval(Name, [\_|L]) -> searchval(Name,L).

cloud(LData) ->

receive

{write, Owner, NewUpload} ->

cloud([{Owner,NewUpload}|LData]);

{get, Name, From} ->

From ! {patient, searchval(Name,LData)},

cloud(LData);

stop -> ok

end.

start() ->

Server\_PID = spawn(erlangprototype\_v4, cloud,

[[{florian,120},{ben,80}]]),

register(server\_process, Server\_PID).

stop() ->

server\_process ! stop,

unregister(server\_process).

write(Name,String2) ->

server\_process ! {write, Name, String2}.

read(Name) ->

server\_process ! {get, Name, self()},

receive

{patient, Data} -> Data

end.

Code for my 4th version of my Erlang prototype with a cloud server

The above code uses only 1 server to secure the personal data.

In line 1, I wrote the module name which is ‘erlangprototype\_v4’. This is the same as the filename without the extension ‘.erl’. In line 2, I exported the functions with the names ‘cloud/1’, ‘start/0;, ‘stop/0’, ‘write/2’, ‘read/1’ and ‘searchval/2’. The number on the right side of the forward slash is the number of parameters the function takes.

In the lines 4-6, I wrote a recursive function with the name ‘searchval’, which takes an element and a list for its parameters, to search a list of pairs of name and values for a certain name.

The function has 3 clauses. The 1st clause says that if the name is not in the list, the function should return the number 0. Next, the 2nd clause says that if the name, along with the value, is the 1st element of the list, the value is returned. Finally, the 3rd clause says that, if the name is not the 1st element of the list , the function has to call itself recursively to search the rest of the list for the name.

In the lines 8-16, I wrote the cloud server that holds a database of name-value pairs that are written to it. If a user wants to access their data, the database is searched for their name using the ‘searchval’ function and their data is returned from the database if their name is found in the database because a message is sent to return the data from the database.

In the lines 18-21, I wrote the ‘start’ function which creates and initialises a new process and stores the initial name-value pairs of ‘florian,120’ and ‘ben,80’ in the database. Finally, it registers a process with the registered name of ‘server\_process’ and the bound process id of ’Server\_PID’ in the system.

In the lines 23-25, I wrote the ‘stop’ function which sends a stop message to the server process and then unregisters the server process.

In the lines 27-28, I wrote the write function, which takes a name and string for its parameters, and it sends a message to the server process to write the name and the string to the database.

Finally, in lines 30-34, I wrote the read function which takes a name as its parameter and sends a message to the server process to receive a patient’s name and data from the database

1 -module(erlangprototype\_v5).

2 -export([cloud/2, auth/2, start/0, stop/0, write/3, read/2, searchval/2,member/2, del/2, authenticate/2, reduce\_token/2]).

3

4 searchval(\_,[]) -> 0;

5 searchval(Name, [{Name,Val}|\_]) -> Val;

6 searchval(Name, [\_|L]) -> searchval(Name,L).

7

8 member(\_,[]) -> false;

9 member(X,[X|\_]) -> true;

10 member(X,[\_|L]) -> member(X,L).

11

12 del(\_,[]) -> [];

13 del(X,[X|L]) -> L;

14 del(X,[Y|L]) -> [Y|del(X,L)].

15

16 cloud(LData, Tokenlist) ->

17 receive

18 {write, Owner, NewUpload, Token} ->

19 B = member(Token,Tokenlist),

20 if B -> cloud([{Owner,NewUpload}|LData], del(Token,Tokenlist));

21 true -> cloud(LData, Tokenlist)

22 end;

23 {get, Name, Token, From} -> B = member(Token, Tokenlist),

24 if B -> From ! {patient, searchval(Name,LData)},

25 cloud(LData, del(Token,Tokenlist));

26 true -> cloud(LData, Tokenlist)

27 end;

28 stop -> ok

29 end.

30

31 auth(UData,Tokenlist) ->

32 receive

33 {authenticate, User, Passw, From} ->

34 \_PW = searchval(User, UData),

35 if (\_PW == Passw) -> From ! {token, hd(Tokenlist)},

36 auth(UData, tl(Tokenlist));

37 true -> From ! {error,"wrong password"}

38 end;

39 stop -> ok

40 end.

41

42 reduce\_token(Number, Tokenlist) ->

43 if Number == 0 -> del(token, Tokenlist);

44 true -> Tokenlist

45 end.

46

47 start() ->

48 Server\_PID = spawn(erlangprototype\_v5, cloud,

49 [[{florian, 120},{ben, 80}], [{42,5},{3,7},{7,2},{1,30}]]),

50 register(server\_process, Server\_PID),

51 Auth\_PID = spawn(erlangprototype\_v5, auth,

52 [[{ben,1337},{florian,42}],[{42,5},{3,7},{7,2},{1,30}]]),

53 register(auth\_process,Auth\_PID).

54

55 stop() ->

56 server\_process ! stop,

57 unregister(server\_process),

58 auth\_process ! stop,

59 unregister(auth\_process).

60

61 authenticate(Name,PW) -> auth\_process ! {authenticate,Name,PW,self()},

62 receive

63 {token, Token} -> Token

64 end.

65

66 write(Name,String2, Token) ->

67 server\_process ! {write, Name, String2, Token}.

68

69 read(Name, Token) ->

70 server\_process ! {get, Name, Token, self()},

71 receive

72 {patient, Data} -> Data

73 end.

Code for 5th version of Erlang prototype

The code above uses 2 servers: a cloud server and an authentication server.

In line 1, I wrote the module name which is ‘erlangprototype\_v5’. This is the same as the filename without the extension ‘.erl’. In line 2, I exported the functions with the names ‘cloud/2’, ‘auth/2’, ‘start/0’, ‘stop/0’, ‘write/3’, ‘read/2’ , ‘searchval/2’, ‘member/2’, ‘del/2’, ‘authenticate/2’ and ‘reduce\_token/2’. The number on the right side of the forward slash is the number of parameters the function takes.

In the lines 4-6, I wrote a recursive function with the name ‘searchval’, which takes an element and a list for its parameters, to search a list of pairs of name and values for a certain name. The function has 3 clauses. The 1st clause says that if the name is not in the list, the function should return the number 0. Next, the 2nd clause says that if the name, along with the value, is the 1st element of the list, the value is returned. Finally, the 3rd clause says that, if the name is not the 1st element of the list , the function has to call itself recursively to search the rest of the list for the name.

In the lines 8-10, I wrote a recursive function with the name ‘member’, which takes an element and a list for its parameters, to find out if an element is in a list. The function has 3 clauses. The 1st clause says that if the element is not in the list, the function should return false. Next, the 2nd clause says that if the element is the 1st element of the list, the function should return true. Finally, the 3rd clause says that, if the element is not the 1st element of the list , the function has to call itself recursively to see if the element is present in the rest of the list.

In the lines 12-14, I wrote a recursive function with the name ‘del’, which takes an element and a list for its parameters, to remove element from a list. The function has 3 clauses. The 1st clause says that if the element is not in the list, the function should return an empty list. Next, the 2nd clause says that if the element is the 1st element of the list, the function should return he list. Finally, the 3rd clause says that, if the element is not the 1st element of the list , the function has to call itself recursively to delete the element from the rest of the list if it is in the rest of the list.

In the lines 16-29, I wrote the cloud server that holds a database of name-value pairs that are written to it and a tokenlist. If a user wants to access their data, the database is searched for their name using the ‘searchval’ function(this uses a token and deletes it from the tokenlist) and their data is returned from the database if their name is found in the database because a message is sent to return the data from the database. If the name is not found, the token is not used.

In the lines 31-40, I wrote the auth server that holds a database of pairs of names and values that are written to it and a tokenlist. If a user wants to access their data, it checks the if the password they entered matches the password for accessing the database. If it does, the user gets a ticket/token from the tokenlist. If there’s no ticket at the head of the tokenlist, the function calls itself recursively to find a ticket in the tokenlist by searching it. If the user entered the wrong password, they get an error that says “wrong password”.

In the lines 42-45, I wrote the ‘reduce\_token’ function which deletes a token from the tokenlist if it can nott be used anymore(each token in the tokenlist can only be used a certain number of times.

In the lines 47-53, I wrote the ‘start’ function which creates and initialises a new process and stores the initial name-value pairs of ‘florian,120’ and ‘ben,80’ in the database. It also stores the tokenlist and the number of times each token can be used in pairs.. Finally, it registers a process with the registered name of ‘server\_process’ and the bound process id of ’Server\_PID’ in the system. I do the same thing for the authentication server but replace the word ‘server’ with the word ‘auth’.

In the lines 55-59, I wrote the stop function which sends a stop message to the server process and then unregisters the server process. It does the same with the auth process.

In the lines 61-64, I wrote a function with the name ‘authenticate’, which takes a name and password for its parameters. It gives the user a token if they entered the correct password.

In the lines 66-67, I wrote a function with the name ‘write’, which takes a name and a string for its parameters, to write data to a database. It sends a message which has the name, the data to write and the token to be used, to the server process

Finally, in lines 69-73, I wrote the read function which takes a name as its parameter and sends a message to the server process to receive a patient’s name and data from the database.

The improvements I tried to make are that the token should have a lifetime so that it can be used for more than 1 action because a user may want to write data to the database and then read data from the database and I should use proper encryption to “translate the data into a secret code because this is the most effective way to achieve data security” Webopediacom. 2019. Webopediacom. [Online]. [21 April 2019]. Available from: <https://www.webopedia.com/TERM/E/encryption.html>. There are “algorithms for public key encryption and decryption in Erlang”. Erlangorg. 2019. Erlangorg. [Online]. [21 April 2019]. Available from: <http://erlang.org/doc/man/crypto.html> Examples of encryption algorithms in Erlang are AES and DES .

Chapter 8: Conclusions and further work

The result of the work that I came up with is that I have managed to secure personal data in Erlang. The process of how I did this is that first I made an Erlang prototype with 3 processes to simulate the patient, the hospital and the cloud. Then, I made a more sophisticated implementation of adding security by labelling the personal data with the owner of that personal data. Next, I did a literature review where I researched and reviewed all the existing literature in my field. I did a literature review for the following reasons: to find out what research question I was asking, why I was asking it, if anyone else had done anything similar, if my research was relevant to research or theory or.practice in my field, what is already known or understood about this topic and how might my research add to this understanding or challenge existing theories and beliefs. The main points that have emerged are that there are ways to secure data in Erlang to prevent it from being leaked or hacked and for my field, this means that people can send data in Erlang securely without it being leaked or hacked. They are related to my project because there is a paper with the title ‘Enforcing User Privacy in Web Applications using Erlang’ by Ioannis Papagiannis, Matteo Migliavacca, David M. Eyers, Brian Shand, Jean Bacon and Peter Pietzuch. In it, they implement information flow control in Erlang and they follow the same goals as what I am trying to achieve in my project. To further secure the users’ personal data, I could give a lifetime to the tokenlist so that each token can only be used a certain number of times and to use proper encryption to encrypt the personal data of the users so that it is jumbled, and a hacker would not be able to read it.

Chapter 9: User guide

The 1st step is to type your name and the personal data you want to secure between the curly brackets in the start function of the program on the line below the line starting with ‘’Server\_PID’. The 2nd step is to type your name and your secret password between the curly brackets below the line starting with ‘Auth\_PID’. The 3rd step is to open the Erlang shell, and where the prompt is, type the letters e,r, and l to invoke the Erlang running system or emulator. The 4th step is to press the Return key on the keyboard. The 5th step is to make sure that you are in the directory that the program is stored in by typing the command ‘pwd(), where the prompt is,’ to display the directory that you are in on the screen. If you are not in the directory where the program is stored, change directory at run time in the emulator by using the command or built in function cd(Dir) and replace the word ‘Dir’ with the address where the program is stored which is "C:/Users/Benjamin Adjei/Documents/Subjects and exercises year 3/Computer Science project". Replace the forward slashes with back slashes. The 6th step is to compile the Erlang prototype by typing ‘c(modulename).’ where the prompt is but replace the word ‘modulename’ with the module name of the program. The module name of the program is ‘erlangprototype\_v4’ so type c(erlangprototype\_v4) where the prompt was and press the Enter key on the keyboard to compile the program. The system should return ‘{ok, erlangprototype\_v4}’ because the compilation was successful. The final step is to evaluate the exported functions in the module by entering the following expression at the Erlang shell prompt: ‘erlangprototype\_v4:start’, followed by a full stop and a carriage return.

Chapter 10: Bibliography

# Bibliography

Rikitake, K. and Nakao, K. (2019). [online] Available at: https://www.researchgate.net/publication/228732159\_Application\_security\_of\_erlang\_concurrent\_system [Accessed 18 Jan. 2019].

Sabelfeld, A. and Mantel, H. (2019). *Securing Communication in a Concurrent Language*. [online] Dl.acm.org. Available at: https://dl.acm.org/citation.cfm?id=716091 [Accessed 18 Jan. 2019].

Svensson, H. and Fredlund, L. (2019). [online] Available at: https://www.researchgate.net/publication/221211336\_Programming\_distributed\_Erlang\_applications\_pitfalls\_and\_recipes [Accessed 18 Jan. 2019].

D Williams, E. (2014). *Erlang Security 101*. [ebook] p.6. Available at: http://file:///C:/Users/Benjamin%20Adjei/Downloads/erlang\_security\_101\_v1-0.pdf [Accessed 17 Jan. 2019].

Papagiannis, I., Migliavacca, M., Eyers, D., Shand, B., Bacon, J. and Pietzuch, P. (2019). [online] Citeseerx.ist.psu.edu. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.6325&rep=rep1&type=pdf [Accessed 24 Jan. 2019].

Hedin, D. and Sabelfeld, A. (2019). [online] Cse.chalmers.se. Available at: http://www.cse.chalmers.se/~andrei/mod11.pdf [Accessed 24 Jan. 2019].

Rikitake, K. (2019). [online] Erlang-factory.com. Available at: http://www.erlang-factory.com/upload/presentations/214/ErlangFactorySFBay2010-KenjiRikitake.pdf [Accessed 24 Jan. 2019].

Learnyousomeerlang.com. (2019). *Distribunomicon | Learn You Some Erlang for Great Good!*. [online] Available at: https://learnyousomeerlang.com/distribunomicon [Accessed 26 Jan. 2019].

Fleck, A. and Kammuller, F. (2010). *Privacy by Flexible Parameterization with Erlang Active Objects*. p.464.

SearchSoftwareQuality. (2019). *What is software requirements specification (SRS)? - Definition from WhatIs.com*. [online] Available at: https://searchsoftwarequality.techtarget.com/definition/software-requirements-specification [Accessed 21 Mar. 2019].

Anon, *Kerberos: The Network Authentication Protocol*. Available at: https://web.mit.edu/Kerberos/#what\_is [Accessed April 1, 2019].

Webopediacom. 2019. Webopediacom. [Online]. [21 April 2019]. Available from: https://www.webopedia.com/TERM/E/encryption.html

Erlangorg. 2019. Erlangorg. [Online]. [21 April 2019]. Available from: http://erlang.org/doc/man/crypto.html

Chapter 11: Appendix

The code can be found in the zip file.