Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Bjørn Ivar Nielsen

Continuous Authentication on an SSH Connection

Hovedoppgave i MIS4900
Veileder: Patrick Bours
Juni 2022

NTNU
Kunnskap for en bedre verden

Bjørn Ivar Nielsen

Continuous Authentication on an SSH Connection

Hovedoppgave i MIS4900
Veileder: Patrick Bours
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Abstract

With a shift to more remote-based work, that is only accelerated by the COVID19 pandemic, new ways of ensuring the identity of users of IT systems are important. The classical approach of username and password for authentication is vulnerable to stolen credentials. By stealing credentials of a user, an adversary can act on the system as the rightful owner of the credentials. Continuous authentication can be implemented to increase the chance of discovering an intruder, and secures the system by revoking access to the account suspected of not being controlled by the owner. The goal of this thesis has been to investigate whether analysis of keystroke dynamics on keystroke data captured at the server side of an SSH session can be used to identify the rightful owner of an account. A publicly available data set has been used as the source of data for our testing, and eBPF has been used to extract the decrypted SSH traffic on the server. The functionality of a commonly used distance measure has been compared on both sides of an SSH session. Different network stress has been applied to investigate the impact of network-introduced interference. A stability of the functionality of keystroke dynamics has been observed at the server side in normal network behavior, whereas a network under stress shows signs of heavily impacting the functionality of keystroke dynamics on the captured data of the SSH channel. With these observations we can state that it is possible to conduct continuous authentication on data capered on the server side of an SSH channel, but in unstable network condition the process will experience degradation.

Sammendrag

I nyere tid har mye arbeid beveget seg i retning av jobbing fra forskjellige lokasjoner. Dette har blitt veldig forsterket av hjemmekontor under COVID19 pandemien. I denne sammenheng er det viktig å utvikle hvordan brukere identifiseres mot IT systemer. Den vanligste tilnærmingen for å identifisere en bruker ved passord og brukernavn er åpen for utnytting enten ved at brukerinformasjon blir stjålet eller ved at sesjonen blir kapret. En ondsinnet aktør vil i dette tilfelle bli behandlet som en genuin bruker av systemet. Kontinuerlig autentisering kan i disse tilfellene bli brukt til å fange opp at brukeren av systemet ikke samsvarer med eieren av kontoen og stenge brukeren ute av systemet. Målet med dette masterprosjektet har vært å se på kontinuerlig autentisering av data fanget opp på serversiden av en SSH forbindelse. eBPF har blitt brukt til å hente ut den dekrypterte SSH trafikken fra en server. Forskjellig nettverksstress har blitt introdusert for å teste innvirkningen av støy har på den kontinuerlige autentiseringen. En stabilitet i resultatet av den kontinuerlige autentiseringen er blitt observert mellom server og klient når nettverket ikke introduserer mye støy. Ved en høyere mengde støy introdusert over nettverket har vi observert en tydelig forverring av funksjonaliteten til den kontinuerlige autentiseringen. Med disse observasjoenen er det konkludert med at kontinuerlig autentisering kan fungere på data fanget opp over en SSH session, ved mye netverks støy vil nøyaktigheten av autentiseringen svekkes.

Acknowledgement

I would like to express my gratitude and appreciation to everyone that have helped me in the process of this project. I will start of by thanking my colleagues at Telenor, they have been extremely helpfully in the technical part of my work the time saved trouble shooting scripts have been invaluable.

I would like to extend my sincere gratitude my supervisor Professor Patrick Bours that have been a great source of both insight into the theoretical field, as well as helped with keeping me on track when stress and doubt has crept in.

And lastly but not least I want to thank my fiancée Hedvig for all the support she have given during this project. Both with input on the project as well as the patience she has showed me during this stressful period.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Topic covered by the project

In today's society, almost everyone is interacting with online accounts daily, that being for online banking, work or e-commerce. To access these accounts, the system needs a way to authenticate that the user is the one they claim to be. In practise there are three ways a system can do this:

- Something you know: Password, PIN code.
- Something you have: Key card, authenticator.
- Something you are: Fingerprint, retina scan.

Due to the ease of implementation, passwords stay the most commonly used way of conducting authentication. Most systems conduct authentication once at the start of a session. This opens up for an unauthenticated user gaining access to an authenticated account and gaining access to parts of the system they should not have, also known as session hijacking. Continuous authentication is a technique that aims to prevent this by continuously reconfirming the identification of the user. This can be done in different ways. The project will focus on one of the approaches in the form of monitoring the behavior of the user. In this project the behavior that will be monitored is how the user types on a keyboard, also known as keystroke dynamics. Keystroke dynamics is one of the most common authentication mechanisms used in continuous authentication. Most research into the concept is based on the behavioral data being captured locally on the users machine. In this project however, an approach for capturing parts, or all, of the data at the server side of an SSH communication will be explored. This introduces some possible difficulties with the amount of data features being available for the authentication process. Another problem is the fact that the captured data is affected by noise from the network. At the same time, being able to conduct continuous authentication on the server side provides some advantages, in the form of not needing to manage the client, thereby making it more applicable in a "bring your own device" environment.

1.2 Keywords

Authentication, continuous authentication, periodic authentication, behavioral biometrics, keystroke dynamics, eBPF, SSH

1.3 Problem description

Much of the work done by IT administrators today is performed remotely, a fact that the pandemic has only proven more evident. Administrators often have access to large parts of IT systems, for example for user creation and access control settings. This makes an administrator account an attractive target for an adversary that wants to gain unauthorized access to a system. Logins often allow authentication solely with the use of a username and password. This leads to an adversary that has gotten access to an administrator account, by either stealing the credentials, or hijacking the session, being able to impersonate as the administrator unhindered until the breach is discovered. With continuous authentication, the goal is to discover if an account is being controlled by another person than the owner of the account. When it is discovered, the session should be reset and the locked out. This should preferably happen as soon as possible so the imposter gets the least amount of time to perform unwanted activity. Most of the research into continuous authentication using keystroke dynamics captures the data from the client of the user. In this project, we want to focus on capturing behavioural data on the server side of an SSH connection.

1.4 Justification, motivation and benefits

As we can see from the problem description, today's solution of mainly static authentication is not adapted to prevent an unauthorized user exploiting the system with stolen credentials. By implementing continuous authentication, what an unauthorized user is able to do on the system can be greatly reduced. Keystroke dynamics can be used to implement continuous authentication. With the increased use of remote working, not every IT department will have the luxury of managing all its end user clients, thus limiting the possibility of capturing client-side keystroke dynamics. An implementation of keystroke dynamics that can be conducted from data captured on the server side could reduce the consequence of unauthorized access.

This project is written based on a suggested topic from Telenor. Telenor is a telecommunication provider. They have proposed a topic looking into a use case of eBPF to extract data from the server side of SSH communication. eBPF is a technology that allows for running sandboxed programs in a privileged context, such as the system kernel, and has its origins in Linux. In the project we will cooperate with supervisors from Telenor.

1.5 Research questions

In this project the main research question we want to answer is the following:

Can continuous authentication be implemented using keystroke dynamics based on data features extracted on the server side of an SSH connection?

Sub-questions we need to satisfy to answer our main question:

1. How can keystroke data features be extracted from a server? To conduct keystroke dynamics from the server side data, a way of extracting it needs to be decided.
2. Is it feasible to implement continuous authentication using the data features captured on the server? In combination with the extraction of these features, an investigation into the usability of the features that can be extracted is needed.
3. How will network latency impact the accuracy of keystroke dynamics? A continuous authentication solution based on server side data features needs to work in variable network conditions.

1.6 Planned contributions

The goal of the master project will be to create a proof of concept of a continuous authentication scheme based on the data that can be gathered on the server side of a SSH connection with the use of eBPF. With this proof of concept, an evaluation will be performed on the functionality of a continuous authentication system. The evaluation will be based on the data features that are possible to extract on the server side of a client server SSH connection.

Chapter 2

Related work

This chapter will look at the general concept that lays the groundwork that is needed to answer our research questions. The chapter is going to look at some general facts about the topics **authentication**, **keystroke dynamics**,**eBPF**, **SSH** and **TCP** as well as presenting research relevant to this project.

2.1 Authentication

Authentication is the process of ensuring that the person or service requesting access to a resource is in fact who they present them self to be. Traditional authentication is done by one or more factors. The tree factors used in authentication are *something you know*, *something you have* or *something you are* also known as *biometrics*. To exemplify the three factors we can say that the most common form of the first is a password or pass code, for the second a token, and for the third fingerprint recognition. *Multi factor authentication* is the concept of requiring two or more of these factors for authentication. An example of this can be two factor authentication used for email login, where the user would need to authenticate with both a password as well as a time-based PIN generated from a shared secret in a mobile app. These methods can be categorized as a *static authentication*, where the system conducts authentication of the user once and creates a session.

There are a couple of ways this can be insecure: session hijacking [1] where a malicious actor uses an exploit to hijack an authenticated session, stolen credentials, or an unlocked session being left physically unattended by the owner and someone else accessing it.

Static authentication is not equipped to prevent these problems, and though multi factor authentication can decrease the likelihood of stolen credentials, the most secure systems might need additional authentication mechanisms. One option here is to re-authenticate the user either by *periodical authentication* or *continuous authentication*. In *periodical authentication* the system re-authenticates the user after a certain interval. This can often be experienced in online banking where you need to authenticate before conducting a transaction. *Continuous* *authentication* is the process of continuously re-authenticating the user. Continuous authentication schemes by nature need to be non-intrusive. This leads the main focus of continuous authentication to be on behavioral biometrics. This project will focus on the use of keystroke dynamics. We will now present keystroke dynamics before we show how this is used in continuous authentication and periodical authentication. In the literature periodical authentication is often described as continuous authentication. In this project we will be referring to periodical authentication as continuous authentication, and continuous authentication as true continuous authentication.

2.2 Keystroke dynamics

The earliest mention of the concept of keystroke dynamics, as far as we are able to tell, is in Gaines et al. [2] 1980 study. In this study the researchers examined the probability distribution of the time between letters in specific digraphs (combination of two letters). By focusing on certain digraphs, they suggested a "signature" that could be used to identify right-handed typists. Even though the study [2] is more than 40 years old, the research into the field did not take off before the early 2000s as can be seen in Teh et al. [3] 2013 survey paper into keystroke dynamics.

Mondal [4] states that most keystroke dynamics systems are implemented through software, even though some studies use specialized hardware on the client to gather extra input in the form of pressure and or sound. In these software solutions, raw data from the keyboard is recorded and can be used to extract timing information used in the authentication process. Banerjee and Woodard [5] define the input of the raw data as the keys pressed, consisting of the timing of the key press and the key release. In figure 2.1 it can be seen how different features can be extracted from this raw data in the form of latencies.

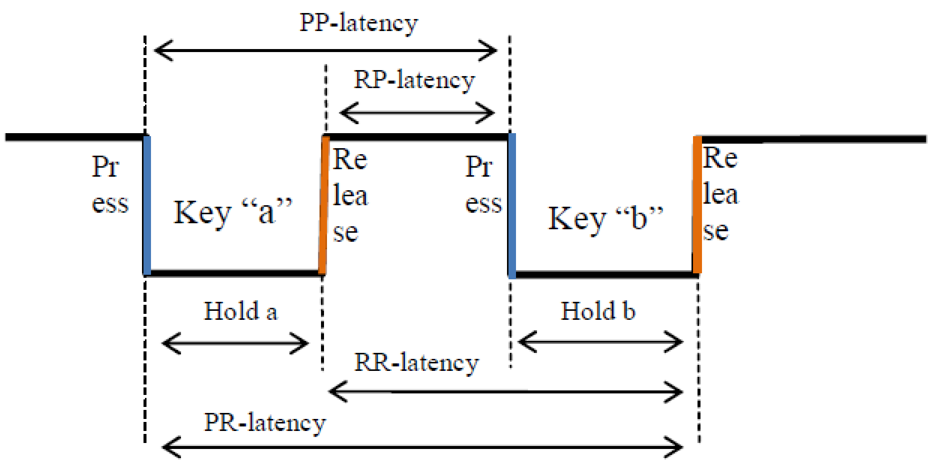

Figure 2.1: Key stroke features: image taken from [6]

With our suggested feature extraction from an SSH channel, we will be limited in the raw features available to us, the reason for this will be described in section 2.4. We will in particular be limited to the latency between different keys presses or the PP-latency from figure 2.1. Due to this we will have to look into the research that only rely on the latency timing. Gunetti, Picardi and Bergadano's research into keystroke analysis [7] and Gunetti and Picardi's further research into keystroke analyses of free text [8] focus on the timing between characters. They describe the rudimentary timing measures in keystroke dynamics based on measuring the time elapsed between when keys are pressed and the next key being pressed. With this information, they were able to calculate the latency between keystrokes. This again was used to calculate n-graphs of different lengths [8]. N-graphs are representation of the latency between n key press from the first key press to the n-th key press. A digraph would be the time elapsed between two characters while a trigraph is the time elapsed between three characters. Gunetti and Picardi investigated two different distance measures based on the sampled n-graphs, R and A-measures. The R-measure is based on the relative speed of n-graphs. This is based on the assumption that a tired typist will type everything slower and the relative speed between the n-graphs will stay constant over time. The A-measure is based on the absolute typing speed and builds on the assumption that a user will have the same typing speed of n-graphs over time. A number of studies build on Gunetti and Picardi's A and R-measures [8], some of these studies includes the following studies: [9–18]

2.2.1 R and A-measures

Gunetti and Picard proposed the *R-measure* in [8]. The R-measure can be described as follows.

R-measure is a way of comparing two different arrays of keystroke data by calculating a distance measure based on the disorder of the arrays. This is done by taking the two samples E1 and E2 and comparing the matching n-graphs of the two samples. Both arrays are ordered based on the timing of each n-graph. Then every element in both array is given the value 1-N where N is the number of matching n-graphs. With this ordered arrangement the distance of the positions of S1 and S2 is calculated where one of the arrays act as a reference, and the other as a probe. An example will illustrate this better. The example is taken from [8]. We use two samples consisting of the timing data of typing the words *authentication* and *theoretical*.

By selecting the matching n-graphs of the two samples we can calculate the R-measure for digraph and trigraphs, written as R_2(E1,E2) and R_3(E1,E2). We can also see that there is one shared 4-graph in the form of *tica*. Since there only is one shared 4-graph this will not give any useful information. In figure 2.2 we can see the corresponding digraphs and trigraphs and a visual representation of the calculation for R_2(E1,E2) and R_3(E1,E2).

Adding up the numbers for R_2(E1,E2) we get 8 and R_3(E1,E2) gives 4. This

Table 2.1: R-measure example: table taken from [8]

E1	Timing in ms	E2	Timing in ms
a	0	t	0
u	180	h	150
t	440	e	340
h	670	o	550
e	890	r	670
n	1140	e	990
t	1260	t	1230
i	1480	i	1550
c	1630	c	1770
a	1910	a	1970
t	2010	l	2100
o	2600		
n	2850		

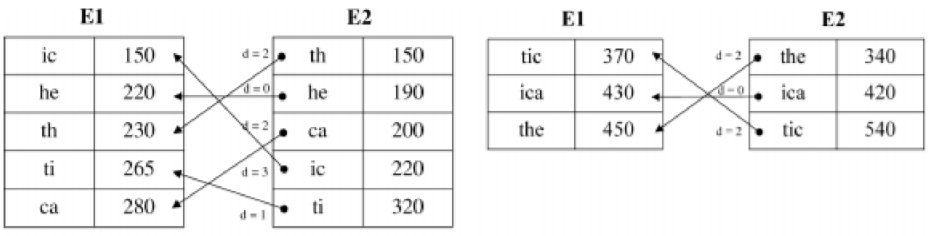

Figure 2.2: R-measure example: image taken from [8]

calculation is visualized in figure 2.2. To be able to compare R-measures with differing numbers of n-graphs, three values need to be normalized. This is done by multiplying them with a normalization factor. The normalizing factor for the R-measure is as follows: $\frac{N^2}{2}$ (if N is even); and $\frac{N^2-1}{2}$ (if N is odd). So in the example the normalized disorder would be $\frac{8}{\frac{5^2-1}{2}}$ or $8/12 = 0.666$ and $\frac{4}{\frac{3^2-1}{2}}$ or $4/4 = 1$. This example is mainly to show the procedure. A longer sample with more corresponding n-graphs is needed to be able to infer anything about whether or not the same user typed the sample and the probe.

A shortcoming of the R-measure becomes clear when calculations are performed between two persons with similar relative typing speed, e.g. they type the same n-graphs in the same relative order. They will have an R-measure of 0 even though the two have drastically different actual typing speed. Therefore Gunetti and Picardi [8] proposed the A-measure to be able to also differentiate based on the absolute typing speed and not just the relative typing speed.

The *A-measure* [8] is based on the premise that a user will be able to have the same or close to the same typing speed on the same n-graphs over time, and that the typing speed of a user will be more similar than that of another individual.

We can look at the similar example as before for the matching digraphs of our two samples.

Table 2.2: Example taken from [8]

digraphs	E1	E2	max/min	Match t=1.25
ca	280	200	1.400	N
he	220	190	1.157	Y
ic	150	220	1.466	N
th	230	150	1.533	N
ti	265	320	1.207	Y

As we can see from table 2.2, the A-measure is concerned with the ratio between the slowest and the fastest digraph. With a determined threshold we decide if the digraph is close enough to be deemed a match. In this example the threshold is set to 1.25, but this threshold can be adjusted to best fit a system. The A-measure will be the number of mismatches divided by the total numbers of shared digraphs. The A-measure can, like the R-measure, be done for n-graphs of varying lengths. For both measures Gunetti and Picardi [8] proposed a weighting that can be used to combine multiples of the same measure e.g. $R_{2,3}$ or $A_{2,3}$. Gunetti and Picardi's[8] approach to profiles were done by having multiple samples from each known user stored. When authenticating a user, the selected measure is calculated against each sample from the user to generate a mean distance between the stored profile and the provided sample. A decision on whether or not the typing was performed by the same user is done based on this distance.

2.3 True continuous authentication

Gunetti and Picardi [8] addressed authentication using R and A-measure in different combinations, but they did not look at it in a sense of true continuous authentication but rather by using full samples to authenticate against known users. Their approach can be described as continues authentication. In their test they used the whole sample instead of splitting into segments. To be able to move from this approach of comparing full samples, to an approach where every single keystroke is continuously used to conduct authentication an intermediate step need to be introduced. It is not enough information in a single digraph to conclude that a user is legitimate or not. One way to make a decision based on this incomplete data is to use a trust model. Trust model in continuous authentication was first introduced by Bours [19]. The trust model builds on the fact that a genuine user will more often than not act like what the system expects of the genuine user while an imposter will most of the time not act like the system expects from the user. A trust model consists of two stages, first it need to determine if the current user's typing complies the template storing the users expected typing pattern. Secondly the trust needs to be updated based on whether the user is deemed to comply or not. The trust score will vary based on the current typing of the active user. When the trust score falls below a given threshold the session is locked out. This increase and decrease in trust can be referred to as *reward* and *penalty* [19]. The action of a genuine user will never be totally inline with the profile, and the action of an imposter will at times match that of the genuine user. A well tuned trust model should preferably never lock out the genuine user, and should lock out the imposter as early as possible. This relies on a well set lockout threshold. It is also important that there is an upper bound for the trust score. This is to prevent the system from generating a very high trust score. Given a high trust score built up by a legitimate user, an imposter could stay active for a long time before the trust would decrease bellow the lockout threshold.

When it comes to the actual amount the score will be incremented and decreased, both [19] and [4] divide it into the two alternatives static and dynamic. Static has a predefined set of values the score will increase or decrease by. This means that there can be multiple different alternatives for reward and penalty values, but they are predefined. With the dynamic approach the increments and decrements are calculated based on the distance of the current action of the user and what is stored in the profile of the user.

To be able to evaluate how well an authentication system operates, some common metrics are used. In static authentication systems the most common metrics are *False Match Rate* (FMR), *False Non-Match Rate* (FNMR) and *Equal Error Rate* (EER). The FMR and FNMR are the rate of an imposter being accepted as the genuine user (FMR) and the chance the genuine user does not get recognised by the system (FNMR). And EER is the intersection where FMR and FNMR are equal [19]. For continuous and true continuous authentication we are more interested in knowing how long a user will have access to the system before being locked out. Bours [19] mentioned this metric, while Soumik and Bours [20] named it as *average number of impostor actions* (ANIA) and also introduced *average number of genuine actions* (ANGA). A system should have as low as possible ANIA and as high as possible ANGA.

2.4 SSH and TCP

We are not going into the deep inner workings of SSH in this project. Since we are going to look at keystroke dynamics over an SSH channel, we are going to present how it works and which part of the protocol impacts the keystroke timing. SSH or secure shell is defined in RFC 4253 [21] and is an application layer protocol that enables encryption, cryptographic host authentication, and integrity protection. In addition, higher level protocols can be utilised to achieve user authentication. SSH supports multiple functions, but the main functionality of the protocol we are going to utilize, is a secure way to communicate between a client and a server. The data being transmitted over the SSH channel is represented by UTF-8 encoding [22]. This will say that the key presses gets translated into UTF-8 on the sender side before transmission. This strips some latency information from the typing in the form of the different latency that can be measured from keystrokes, as presented in section 2.2 in the form of timing of keys being pressed and released. What we are left with are a timing of the character being received. SSH running in interactive mode has the effect of sending one package for every character [23], this is needed to circumvent latency that TCP (Transmission Control Protocol) introduce that would makes the typing jagged for the user, and is therfor needed in an interactive application like a text shell. This is a prerequisite to be able to conduct any form of keystroke dynamics on an online channel due to the fact that TCP normally waits for multiple character and send them in a bulk of characters which removes all information needed to conduct keystroke dynamics. Encrypted data also lacks information for keystroke dynamics since the information of which key being pressed is not available, there is statistic analysis that can be done on the encrypted traffic, as seen in [23]. For our use however this information is available after the decryption on the server side. The only latency left at this point is a form of RR or PP latency, in the form of timings between each character.

This latency can be monitored at the receiving side of an SSH channel, due to the transmission latency being variable, it will impact the timing of the received data compared to data being emulated and sent. SSH is most often transmitted over network using TCP. TCP was first defined in RFC 675 [24] i 1974 and in 1981 RFC 793 [25] defined TCP as a standalone protocol. TCP was designed as a reliable way of communication over an unreliable network. The IP protocol transporting TCP does not guarantee that each package reaches the destination, this is then something that has to be ensured by overlaying protocols, this is some of what TCP delivers. The internet is built up as a packet switched network structure using TCP/IP. A packet switched network is a communication network where the data is sent in small packages on a shared network, this differs from the earlier line switched network, where signals was sent on a designated circuit. This opens up for both more unforeseen variable delay, rearranging order of package and data loss of small segments of the data.

To ensure that the package is not lost TCP is designed so that the receiver needs to inform the sender that it has received the package. The protocol also needs to support the cases where the receiver never receives the package. This is done by a timeout mechanism, and acknowledgment. The timeout ensures that if no acknowledgment is received by the timeout timer, the package is re sent. To prevent problems from reordering during transmission sequence numbers are used, the sequence numbers are used as part of the acknowledgment mechanism.

Since TCP is operating on a shared network, mechanisms to prevent overloading the network is needed, to prevent degradation for both ones own communication as well as that of others. An important mechanism to ensure reliable transition as well as decrease the data load is *flow control*, flow control is mentioned in RFC 793 [25] it lets the receiver adjust the amount of data the sender can transmit. This is done to prevent the sender from overloading the receiver by sending more data than the receiver can handle, which again would lead to the data needing to be re transmitted and causing more load to the network. This is done by the receiver sending a "receiving window" which informs the sender of the sequence number that the sender is permitted to send, before getting a new permission. This will control that the sender never sends more data than the receiver can handle. This window will continuously be adjusted based on the receivers ability to process the incoming data. In practice both the client and server will act as both a sender and a receiver in the TCP transmission.

Another mechanism that TCP implement is congestion control which was introduced in RFC 896 [26]. Congestion control aims to prevent congestion in the network by adjusting the data rate based on the feedback from the network. This builds on the assumption that if a package is lost it is due to congestion in the network, and the assumption that if there is no package loss, the network can handle more data load. Congestion is based on a slow start, and a back off algorithm. An initial low data amount is chosen as the most data that can be handled by the network. When the sender receives acknowledgment an assumption that there is more capacity to use is made, TCP will then gradually increase the data amount until package loss is encountered. At that point, the sender will back off by decreasing the sender window and start increasing again. There exist different algorithms for this, but as an example the congestion control algorithm TCP Tahoe, will half the sending window and start the slow start again with a window size of half the size it was at when experiencing data loss. This implementation of the mechanism leads to a characteristic saw tooth pattern that can be seen in TCP traffic an example of this pattern can be seen in figure 2.3. It can be noted that this is mostly a problem when transferring large data loads where there is more data to send.

Another mechanism described in RFC 896 [26] that somewhat impacts SSH is the small-packet problem. The small-packet problem, consist of the amount of overhead being sent when sending a small amount of data per package. In SSH a single byte sent will be represented by a 36 bytes after encryption. The header for each sent package is 40 bytes, leading to over 50% overhead. In the example mentioned in the RFC telnet is used, which send 1 byte per byte, so a overhead of over 97,5%. By sending multiple bytes in the same package the overhead is reduced, this mechanism is implemented by an algorithm called Nagle's algorithm. Nagle's algorithem holds of sending a TCP package based on some parameters to decrease the amount of packages needed to be sent. This however impacts the timing of the packages in real time application. The interactive SSH mode that we mentioned in the introduction of this section disables Nagle's algorithm, which normally are a default part of TCP.

An example on how these mechanisms can impact our goal of monitoring timing of an SSH channel on the server. In particular how package loss can wipe out the timing information to be used in keystroke dynamics. We can visualise the timing effect this can have on keystroke timings as following. Given sending the string "Hello world" where each character is separated by 200ms. If the "o" in "word" is lost and the timeout and re-transition takes 400 ms the following two character will already have arrived before the "o" actually arrives. This will impact

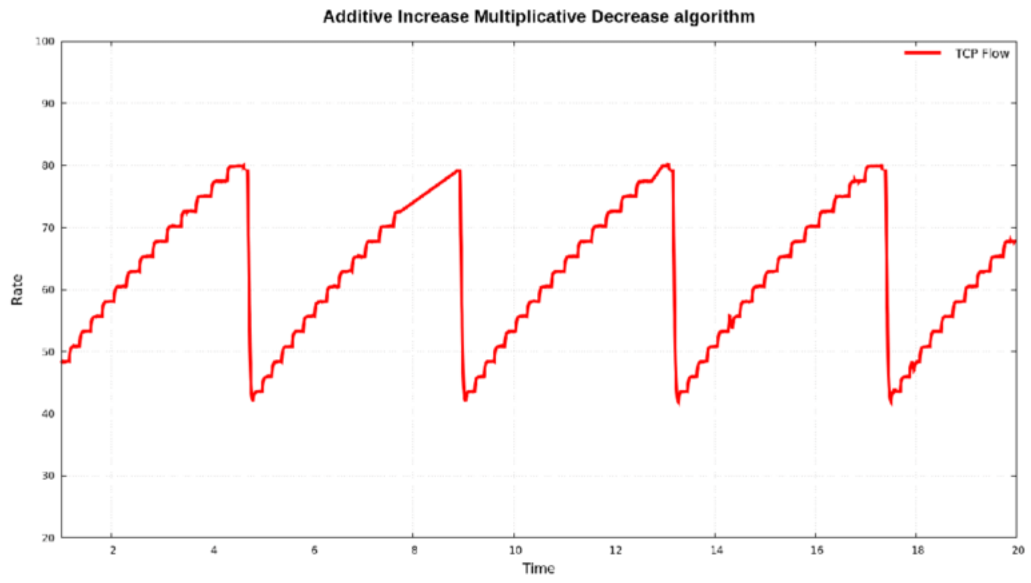

Figure 2.3: Characteristic TCP saw tooth pattern: image taken from [4]

the digraph "wo" "or" and "rl". The digraph timing on the sender and receiver can be seen int table 2.3. Here we can see that the timing of the 3 digraphs are affected by the loss of one package.

We have now presented aspects of TCP and SSH that can impact the latency between characters from what is sent to that receiver. We touched in on how a package switched network could lead to variable delay between package. This variable network delay is also known as *jitter* [27]. Other than our example where a packet loss is the cause of jitter, traffic where no package are lost can also experience jitter in the form of packages having slightly different travel time in the network. Claypool and Tanner [27] present the impact jitter has on a real time application in form of video, where QoS (Quality of service) is important, due to the user impact severe jitter can cause, voice communication is another place where jitter maters.

As mentioned in the introduction we want to explore the effect of capturing keystroke timing data from a server the user is connected to. In this regards network latency and jitter are important aspects. Network latency is the time it takes to transfer the input sent from the user to the server. In a perfect situation, this time should be the same for each package, but due to different factors in the network this time varies from package to package. This variance between the latency of single packages and the expected latency. This jitter introduces a difference in the delay between actions being monitored on the server compared to the real delay based on the users action. An example of how jitter can ruin other real time

Sender	
Digraph	Latency
He	200
el	200
ll	200
lo	200
o	200
w	200
wo	200
or	200
rl	200
ld	200

(a) subtable no. 1

Reciver	
Digraph	Latency
He	200
el	200
ll	200
lo	200
o	200
w	200
wo	600
or	0
rl	0
ld	200

(b) subtable no. 2

Table 2.3: Digraph timing sending and receiving side

application is voice, where jitter can cause interference of the sound leading to an unusable service. In our proposed setup the stable network delay should not pose a problem, but the jitter can cause degradation to the keystroke dynamic. We want to explore the impact of this depredation based on different network conditions.

2.5 eBPF

eBPF no longer is an acronym for anything, though it was once called the "extended Berkeley Packet Filter". eBPF is a further development based on McCann and Jacobson 1992 paper into user-level packet capture [28]. McCann and Jacobs proposed BPF as an improvement on the original Unix packet filter. Packet filter in this sense is a way for users to monitor or tap network traffic. Network monitoring software needs to run in user space and due to this, every network packet needs to be copied over the kernel/user-space protection boundary. The improvements seen in BPF compared to earlier Unix packet filters comes in their pseudo-machine. This pseudo-machine is a virtual machine running in the kernel. The virtual machine is designed to be programmable and BPF programs can be deployed into the kernel. With the filtering taking place inside the kernel, BPF reduces the overhead of copying all unwanted data from kernel to user space saving on CPU usage. eBPF was released in 2014 in the Linux kernel 3.18 [29]. This extension to BPF improved on the efficiency of the BPF virtual machine, and opened for a broader use to other kernel component than network cards.

Chapter 3

Data and design

This chapter will describe the dataset used in this project to conduct our testing on continuous authentication as well as the lab setup and the data proseesing conducted. We will start off by describing the lab setup and the script we used to capture data, before presenting Clarkson's [30] keystroke dataset that we have used as the source for user data in this project. We will describe how we have processed the data and how it has been analyzed in regards to applicability for keystroke dynamics.

3.1 Lab setup

Figure 3.1 gives a high level overview of the lab setup. The lab setup consists of two virtual machines, one representing the client where the participants keystrokes get sent from, and one representing the server where the keystrokes get captured. On the client where the participants keystrokes were emulated using the Clarkson dataset [30]. An SSH channel was established between the client and the server. When the SSH channel was established then the capture points were initiated on the server in order to capture the keystroke data from the emulation transmitted over the SSH channel. Afterwards, the emulator was initiated to transmit the keystroke data from the client to the server over the SSH channel. The data captured on the server as well as the emulation timing from the client was used in our analysis, we have calculated the timing change introduced by the emulator and the network, as well as analyzed the impact this changes had on the A-measure and R-measure and A+R-measure when it comes to EER.

The two machines making up the lab setup were both Linux machines running Debian 10. Both of the machines are virtual machines located on geographical separated servers, with an expected network latency of around 7,5 ms each way. The servers are connected with Ethernet and the connection between the two locations is over dimensioned. This gave the setup a good baseline in order to have a low amount of interference to impact the data during transmission.

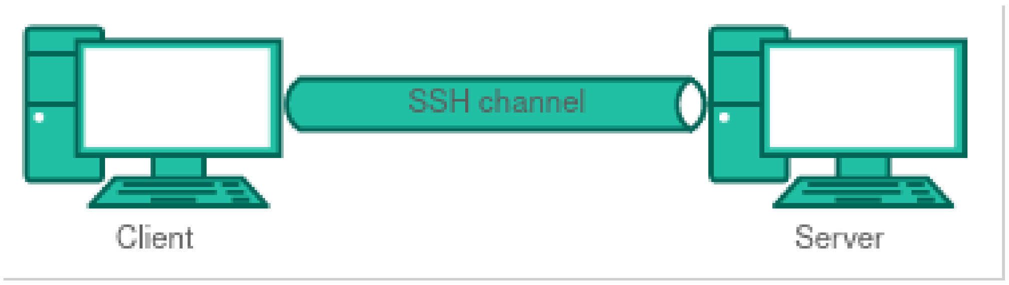

Figure 3.1: Experiment setup

3.2 Dataset

In this project we chose to use Clarkson's keystroke dataset as our data source. The alternative was to carry out our own data capture with live participants. We chose to use Clarkson's dataset due to the uncertainty and restrictions in the current corona situation, along with the flexibility this gave us in the form of replaying . The dataset was developed by Vural et al. [30] and published in 2014. The dataset was developed with the goal of creating a shared dataset to be used in the testing of keystroke authentication, due to the tendency in the literature of using proprietary private datasets. The Clarkson keystroke dataset consists of data collected from 39 subjects during two sessions for each subject. 34 of the users have conducted both sessions fully, while the remaining five participants did not fully completed the tasks, or did not conduct the second session. However in the version of the dataset we were supplied, there were 7 participant files missing data, leaving us with 32 participants who had fully completed the tasks. The sessions consisted of 72 tasks containing repeated password entering, free text and transcription tasks. in the form of password writing, free text survey question answering, and transcription of a speech by Steve Jobs. Both sessions consisted of the same tasks. In this project the free text survey questions were the ones that were used. The dataset consisted of one csv file per session, per participant. Each csv file was represented by a line for each task. Each line consisted of four tab separated values where the three first consisted of administrative information, and the forth is the keystroke data from the task. An example of keystroke data: "*0:61:1312903092233:1242719,...*" four values separated by colons representing:

1. Key down/ Key up
2. Virtual key code
3. Timestamp in ms
4. Elapsed time since last keystroke in ms

In the example 0 represent key press, 61 represent that the key pressed was numpad-1 while the two last values is timing values. Each keystroke is separated by a comma. The key code is presented in the format presented in this Microsoft documentation [31].

3.3 Emulator

In order to be able to use this dataset in our project, the participants keystrokes needed to be transmitted over an SSH channel to be recaptured, as described in section 3.1. To do this, we developed a keystroke emulator that parsed the free text tasks of the dataset to emulate the keystrokes as represented in the dataset. The emulator was ran on all the files in the dataset, continuously parsing the files, until all data was emulated. The action of the emulator was sent over the SSH channel. The script was written in Python and used the library Pyautogui, a python library that allows python to take control of keyboard and mouse input. This script parsed through the dataset and used the key code as well as the elapsed time since the last action to emulate the participants keystrokes. Some corrections where needed in files which did not properly end, to be able to parse them. These files ended in the middle of a keystroke and was corrected by removing everything after the last properly formatted keystroke. With everything formatted we were left with 32 participants with the desired data for our emulation. Some of the characters in the dataset could not be sent directly through our emulator due to them interfering with the sending terminal, eg. start opening a menu where the typing would continue. To circumvent this, these keys were represented by other characters that were possible to transmit. We timed the emulator to get a timing when each keystroke was pressed. This timing was used to calculate the time change introduced by the emulation script compared to the original dataset.

3.4 Data capture

In our project we were interested in looking at keystrokes on the receiving side of an SSH channel. To monitor the keystrokes on both sides of the SSH channel, as mentioned we timed the emulator to represent the sender side, while we on the server side used an eBPF script that monitored the Linux system call "vfs_write" against the process running SSH shell. This gave a data feed consisting of a timestamp in milliseconds and the character that was received, this together with the timing of the emulator was used in our analyses. In most keystroke dynamics research monograph timings *down up* and *down down* can be used. Due to SSH only transmitting UTF-8 encoded character, the timing of when the key was pressed and released was not accessible to us. We were only left with one timing for each character, represented by the time the character is written to the shell by vfs_write. With the timestamp of each key, the latency between each succeeding character was calculated to represent the digraph timing to be use in keystroke dynamics. Another effect of monitoring the SSH channel compared to the measurement used in most research was the ability to distinguish between keys. In the Clarkson dataset, we could distinguish a 3 and a numpad-3 as two different keys. On the other hand, in our data the capture UTF-8 character representation can not distinguish which key that has been used to write a character that can be written by multiple keys, eg numbers, symbols. There are also keys that can not be monitored in this data capture, for example shift, caps lock, and numb lock. These keys affect the behavior of the keyboard, but for the SSH keylogging we could not discern any of them. A capital letter "A" can be both an "A" written with the shift key down, or with caps lock enabled. As mentioned in the previous section, we chose to circumvent this by sending another key that could be discerned and that was not used in the Clarkson dataset, eg. shift was emulated as "*". This was done to simplify the problem due to the time aspect of this project. For future work however it is advised to look further into the data capture to look at only the actual received data.

The data gathered from the sender and receiver side gave us different data to be used in our analysis. The sender side data gave us the actual timings of when the keystrokes were made by the emulator, giving us a way of finding the delay introduced by our emulator as compared with the dataset. On the receiving side the described eBPF script was ran, this gave the timing of when the characters were written to the SSH shell on the server. When comparing the data captured on the sender side and the receiving side we were able to analyzed the impact of network delay had on our data, and for comparison the functionality of keystroke dynamics on both transmitted and transmitted data. From this data we were able to calculate multiple forms of introduced delay. From the client side data we were able to calculate the delay introduced by our emulator onto the data by comparing it with the original timings in the dataset. The difference between the client and server data gave a measure of the jitter introduced by the network channel.

3.5 Data processing

The captured data described in the last section consisted of a continuous data stream of all the data sent by the emulating script and the data captured by our eBPF script. The data contained all the free text tasks for each participant. To be able to separate the data in our data capture, some administrative data strings were sent together with the tasks by the emulator. This administrative data was used to split the captured data into one file per task. This left us with 16 files for each of the 32 participants that had completed all the free text tasks. The 16 files consisted of 2 occurrences of the 8 tasks answered by the participants. From the 16 files we created 8 reference profiles for each participant. We did a 8-fold cross validation of the A-measure and R-measure by taking the two sessions of one task for testing and training the profiles with the 2x7 remaining tasks. Our reasoning for keeping out data from two tasks and not just one task, was due to the similarity expected between the same tasks from the different sessions, due to them being the answer to the same question. This was done on both the sender side and the receiving side data.

The same profile creation was also conducted on the timing extracted directly from the Clarkson dataset. This was done to have a comparison to look for eventual degradation seen on the measures from the data to the recaptured data. For our recaptured data, we have compared the data with profile generated from the recaptured data as well as the profile generated timing from our emulation, we will refer to this profiles as *SP* or the sender side profile and *RP* or the receiver side profile. This left us with the three following comparison sets for each conducted comparison.

1. Comparing emulated timing with SP .
2. Comparing the recaptured data against RP.
3. Comparing the recaptured data against SP.

We did this comparison against multiple different data captures to represent different conditions both in the form of differing network conditions, and we used different size of probes to look at the trends.

3.6 A-measure and R-measure

As mentioned in the last section, the A and R-measure was conducted as a 8 fold cross validation. With each reference profile, the two tasks left out of the profile creation were used as probes. For each reference profile we are left with 32x2 probes, where 31x2 of them gives us an imposter score and the reminding two gives us a genuine score. This was repeated for all the 32 reference profiles leaving us with 32x2x8 or 512 genuine scores and 31x2x32x8 or 15872 imposter scores. We used the A-measure,R-measure and combinations to calculate the "Equal Error Rate"(EER). We mainly focused on the digraph, and calculated A_2, R_2 and A_2+R_2 for each of the comparisons mentioned in the last session. The calculations were conducted by python scripts that calculated the values based on the specified dataset. The results are presented in Chapter 4. We have conducted testing with different probe sizes. One with the probe being represented with the entire tasks, here the data amount can vary from approximately 500 characters to over 1300. We have also conducted a test with the probe being a limited block size, we have here looked at 100 and 250 digraphs. The first test consist with what the test done by Gunetti and Picard [8] where the entire data sample was used as the probe. While the latter test is more in accord with the literature on continues authentication.

3.7 Limitations and adjustments

Compared to a real world implementation of a server side keystroke dynamic system, we have added some limitation to better be able to directly compare with a traditional keystroke dynamics system. As mentioned earlier in this chapter, due to the scope of the dataset containing an array of keys that are not directly transmitted from the sending side to the receiving side over the SSH channel, some characters were sent as alternative characters by our emulator. Another adjustment was that some keystrokes in the dataset had a delay of an unnaturally long time, this will be when the participant was waiting or thinking which deviated from the normal typing rhythm. We chose to exclude every digraph with a latency over 500 ms and under 30 ms, the same as [30] used in their test of the dataset against A-measure and R-measure, in our emulator to reduce the time of the test, keystrokes with a delay over 600 milliseconds were reduced to 600 milliseconds. This was done both in the profile creation, and from the probe.

During the process of this project some adjustments have been conducted due to narrowing in what we are looking for, due to the result giving us more insight into or data. Some revision of our test end emulation have been conducted. We have chosen to not describe the full process done with both the original assumption, and with all the iterative changes we have done due to discoveries made during the process. So the content presented in this chapter describes the final approach of our data and design.

Chapter 4

Result and Discussion

In this chapter we will present the results we have gotten from our testing presented in chapter 3. We will present the test results one by one for each test scenario. The impact of the results when it comes to the functionality and applicability of continuous authentication will be discussed. We have conducted tests with different parameters. The results for these tests will be presented in their own section. Each section in this chapter consists of an introduction where we present the timing change experienced by the data during emulation and transmission. Each section also consists of two subsections, one for the results based on the full samples, and one looking at different block sizes. We have conducted tests to look at the impact of delay, jitter and data loss has on the functionality of R-measure, R-measure and A+R-measure. The following test cases have been conducted:

1. Baseline
2. Static delay
3. Jitter
4. Package loss
5. Jitter and package loss

4.1 Baseline

The first test is a baseline to see how the network behave with no added limitation or stress and how this impacts our measurements. To start of we will present some statistics on the timing changes we see from the original dataset during the emulation, as well as the network delay. We will present them in a split table consisting of: *Emulation time changes* representing the time difference between introduced by the emulation, and *Network time changes* representing the time changes introduced by the delay in the network.

With our limitation on which tasks to use, and which participants to include, as well as administrative data added, the baseline dataset consisted of 433486 digraph values. Both sides of the table consist of 3 columns each: *latency change*, *number* and *percentage*. The delay change is presented as a range, 0 is all the unchanged delays, 1 representing every digraph where there is one millisecond change, 2-5 for the digraphs where there are 2-5 millisecond changes and 6-10 represent the ones changing 6-10 millisecond and the >10 represent the once with more than 10 millisecond changes. The number column is the number of keystroke in the given range. While percentage is the percentage of all the characters in the range. The latency changes represent the absolute value of the change, in the emulation there are strictly speaking only positive changes, while the network time changes consist of both positive and negative time changes.

Emulator time change			Network time change		
Delay change	Number	Percentage	Delay change	Number	Percentage
0	5504	1,27%	0	270236	62,34%
1	257862	59,49%	1	150883	33,81%
2-5	168290	38,82%	2-5	11945	3,75%
6-10	1724	0.40%	6-10	359	0.09%
>10	106	0.02%	>10	63	0.01%

Table 4.1: Timing statistics:Baseline

From table 4.1 we can see that there is some time change introduced into our data from the emulation, for our testing this is seen as an acceptable delay, seeing that over 99,5% have a less than 6 ms change from the dataset and 99.98% with less than 11 ms. It is expected to see some delay introduced due to the processing time of the emulation script. In total the introduced delay from our emulation causes a 0.75% increase in the total time of the dataset. This introduces some change in how the A and R-measure behave with the data. Therefor we will use the timing after emulation for all our comparison. We also see that there is some network introduced time changes. In the baseline test this is at a very small scale, where 99,9% of the data have less than a 6 millisecond change, and over half the data experience no changes at all.

During our test period we have also looked at the capacity of the network connection used between the sender and receiver. The data load of the network link newer exceed 20% of the capacity of the network. We noticed some outages for the connection, but they were all during planed downtime's, no testing were conducted during these network outages.

4.1.1 Baseline full sample

Now we will present the tables representing the EER and standard deviation for the A-measure, R-measure and the A+R-measure when comparing the full sample with the different profiles. In the baseline test we have also presented the EER for the dataset. The EER is calculated for each of the eight profiles used in our eight fold cross validation, the EER is the mean of the false non match rate (FNMR) and the false match rate (FMR) at the threshold they are closest. The presented

EER is the mean of these 8 values. While the standard deviation is the standard deviation of the 8 values.

Dataset			
	A-measure	R-measure	A+R-measure
EER	6.4%	3.9%	3.8%
Std Dev	1.17	0.83	0.78

Table 4.2: EER of measures of Dataset

From table 4.2 we can see the EER expected from the original data we have used. By comparing these EER with what we see from our sending side data in table 4.3, we can ensure that there are no unexpected changes occurring in the emulation.

		A-measure	R-measure	A+R-measure
Sending	EER	6.4%	4.2%	3.5%
	Std dev	1.14	0.98	0.67
Receiving RP	EER	6.7%	4.1%	3.7 %
	Std dev	1.04	0.98	0.70
Receiving SP	EER	6.7%	4.1%	3.7%
	Std dev	1.05	0.98	0.75

Table 4.3: EER Baseline

When comparing the EER of the dataset from table 4.2 and the sending data from table 4.3 we can see there are some small changes of 0.3% to the EER for the R and the A+R-measures. Due to the time changes intruded by the emulation some changes are to be expected. From the standard deviation seen in the original dataset in table 4.2 the change is within a scale where the changes seen in our sending side data is well within half a standard deviation.

By comparing the *receiving RP* and *receiving SP* section of 4.3 with the *sending* section we can see that the small amount of network introduced delay from the network in this test, results in changes to the EER on the receiver side data as compared to the sending side. Also here the changes fall with in the observed variance of the data. When it comes to comparison with the different profiles this gives the same result with some minuscule changes in the standard deviation. From the data we have seen from this initial test it can seem like a low latency stable network connection does not greatly impact the functionality of the measures we are testing, and that keystroke dynamics can be conducted using network captured data, given the right conditions.

4.1.2 Baseline block size 100

Now we will present the tables representing the EER and standard deviation for the A-measure, R-measure and the A+R-measure when using a block sizes of 100 with the different profiles. In the baseline test we have also presented the EER for the dataset. A note to make here is that the EER is generally higher in this and the following subsection focusing on bloc sizes. This is due to the decreased data amount in the probes, the EER is expected to increase when reducing the block size. As mentioned earlier, we are not focusing on achieving the lowest EER, we are interested in the trends seen in the data when it comes to the impact the network has on the EER.

Dataset block size 100			
	A-measure	R-measure	A+R-measure
EER	18.4%	16.1%	12.5%
Std Dev	1.62	1.82	1.72

Table 4.4: EER of measures of Dataset block size 100

		A-measure	R-measure	A+R-measure
Sending	EER	18.4%	16.0%	12.4%
	Std dev	1.91	1.69	1.87
Receiving RP	EER	18.0%	16.3%	12.5%
	Std dev	1.93	1.88	1.93
Receiving SP	EER	18.1%	16.3%	12.5%
	Std dev	1.93	1.88	1.93

Table 4.5: EER Baseline block size 100

We can see that the sending side EER in table 4.5 is similar to what is seen in the corresponding dataset table 4.4. We see similar result in table 4.5as we saw in the test with the full text sample, where there are some small changes between the sending side and the receiving side EER of up to 0.4%, at this scale of the changes it seems like it is stable with no additional network stress.

4.1.3 Baseline block size 250

Now we will present the tables representing the EER and standard deviation for the A-measure, R-measure and the A+R-measure when using a block sizes of 250 with the different profiles. In the baseline test we have also presented the EER for the dataset.

We can see that the sending side EER in table 4.7 is similar to what is seen in the corresponding dataset table 4.6. Also at this block sizes we see similar result in table 4.7 as we saw in the test with the full text sample and with the block size

Dataset block size 250			
	A-measure	R-measure	A+R-measure
EER	11.6%	8.8%	7.7%
Std Dev	1.79	1.66	0.96

Table 4.6: EER of measures of Dataset block size 250

		A-measure	R-measure	A+R-measure
Sending	EER	11.8%	8.3%	7.7%
	Std dev	1.89	1.60	1.46
Receiving RP	EER	11.7%	8.7%	7.5%
	Std dev	1.96	1.49	1.25
Receiving SP	EER	11.7%	8.8%	7.5%
	Std dev	1.94	1.49	1.25

Table 4.7: EER Baseline block size 250

of 100, where there are some small changes between the sending side and the receiving side EER of up to 0.4%.

4.2 Static Delay

To investigate that a static delay do not impact the result of the EER any more than the baseline test, we have conducted a test where we have introduced a static delay to all network traffic. This is done by adding a 150 millisecond delay on all outgoing network traffic from the sender this is approximately a 20 times longer delay than the data experience in the network. From table 4.8 we can see that the emulation delay is consistent with what we saw from our baseline test, with approximately 99,9% of the data having less than 6ms delay and only some minor changes as compared to the baseline test. The same holds true for the network time changes as can be seen from table 4.8.

Emulator time change			Network time change		
Delay change	Number	Percentage	Delay change	Number	Percentage
0	5882	1,36%	0	255640	58,97%
1	256883	60,62%	1	159092	95,67%
2-5	168527	99,49%	2-5	18266	99,89%
6-10	2059	99,99%	6-10	451	99,99%
>10	135	100%	>10	36	100%

Table 4.8: Time change statistics: Static delay

4.2.1 Static delay full sample

In this subsection we present the EER from the full sample test with introduced 150 ms static delay.

		A-measure	R-measure	A+R-measure
Sending	EER	6.5%	4.1%	3.7%
	Std dev	1.09	0.98	0.75
Receiving RP	EER	6.6%	4.0%	3.8%
	Std dev	1.25	0.81	0.81
Receiving SP	EER	6.5%	4.0%	3.8%
	Std dev	1.14	0.80	0.78

Table 4.9: EER with introduced static delay

From the EER in this test as can be seen in table 4.9, we can see a small differences of 0.1% consistent with what we saw from the baseline test. This is in line with what we expect from the theory, where a flat delay should not impact the EER in any way. In a real world network a longer delay that comes with a larger distance from the receiver, would naturally introduce some more jitter, but from Wondernetworks ping statistics [32] we can see that it is not directly correlated to distance, and seem to be more tied to the stability of the network between the nodes. With a longer distance between the sender and the receiver, it is a larger probability of encountering a link experiencing congestion or other problems that might effect the inter package timing.

4.2.2 Static delay block size 100

In this subsection we present the EER from the block size test of 100 characters with 150 ms static delay introduced.

		A-measure	R-measure	A+R-measure
Sending	EER	18.6%	15.9%	12.3%
	Std dev	1.09	1.88	2.03
Receiving RP	EER	18.0%	16.4%	12.5%
	Std dev	1.31	1.91	2.05
Receiving SP	EER	18.1%	16.2%	12.4%
	Std dev	1.08	2.11	1.90

Table 4.10: EER with introduced static delay block size 100

In table 4.10 we can see a similar EER change as we saw in the baseline test instead of an up to 0.4% change as seen in the baseline test we see an up to 0.6% change in the EER for the 100 character test but still well within the standard deviation of the sending side data.

4.2.3 Static delay block size 250

In this subsection we present the EER from the block size test of 250 characters with 150 ms static delay introduced.

		A-measure	R-measure	A+R-measure
Sending	EER	11.6%	8.8%	7.7%
	Std dev	1.65	1.80	1.09
Receiving RP	EER	11.8%	8.7%	7.3%
	Std dev	1.83	1.72	0.99
Receiving SP	EER	11.9%	8.5%	7.3%
	Std dev	1.87	1.74	1.01

Table 4.11: EER with introduced static delay block size 250

In table 4.10 we can see a similar EER change as we saw in the baseline test with some small differences from the sender side to the receiving side, but no changes to any large extent.

4.3 Jitter

We have conducted testing where we have introduced jitter. This test was conducted to view the effect of jitter on the A, R and A+R-measure. In the following test we have introduced jitter by introduced a 25 millisecond delay, with a +-25 millisecond variance. This gives the chance of 50 millisecond jitter if one package have the maximal delay, followed by one with the minimum delay. The chosen jitter is based on the maximum jitter Cisco deems voice and video should be able to handle [33], to see if continuous authentication using A and R-measures can handle the same extreme. This gave the timing statistics that can be seen in table 4.20.

Emulator time change		
Delay change	Number	Percentage
0	6213	1,43%
1	263528	62,22%
2-5	161868	99,56%
6-10	1765	99,99%
>10	112	100%

Network time change		
Delay change	Number	Percentage
0	8704	2,01%
1	17510	6,05%
2-5	64563	20,94%
6-10	72753	37,73%
11-25	166185	76,06%
26-50	103750	99,99%
>50	20	100%

Table 4.12: Time change statistics with introduced jitter

We can see from table 4.20 that the emulation timing is approximately the same as for the earlier tests, and that as we expect the changes are due to the jitter. From the timing changes introduced by the network as seen in table 4.20 we can see that there is a large shift from what we saw in the baseline and the static delay test. In the baseline mostly all the packages were within 0, 1 or 5 ms bracket amounting to 99,9% as can be seen in table 4.1 while in this test we only got one fifth of the data falling within this range. Where as two thirds of the data falls within the 10, 25 or 50 ms brackets in this test. As to how this impacts the A and R-measures we can see the result from this in the tables presented in the next two subsections. We have also added the >50ms bracket to show that only 20 characters are effected more than the maximum introduced jitter.

4.3.1 Jitter full sample

In this subsection we present the EER from the full sample test with 50 ms jitter introduced.

		A-measure	R-measure	A+R-measure
Sending	EER	6.5%	4.1%	3.7%
	Std dev	1.13	0.98	0.73
Receiving RP	EER	7.7%	5.1%	4.3%
	Std dev	1.22	1.06	0.67
Receiving SP	EER	7.3%	5.0%	3.9%
	Std dev	0.95	0.57	0.81

Table 4.13: EER with introduced jitter

The distance measurements on the sending side in this test as seen in table 4.13 give us the same result as we saw in the baseline test falling in line with the values seen in the EER against the dataset. When we look at the receiving side data however, we start to see some degradation to the EER. For the A-measurement we see between 0.8-1.3% increase, depending on the profile we are comparing with, the change to the R-measure is between 0.9-1.0% while the A+R-measure experience a 0.2-0.6% change. There are some signs that the degradation is somewhat reduced when comparing against the SP, compared with the RP. Especially the A+R-measure only experience a 0.2% increase representing under 1/3 of a standard deviation from the mean as seen in the sender in the sender side data.

4.3.2 Jitter block size 100

In this subsection we present the EER from the 100 character block size test with 50 ms jitter introduced.

In table 4.14 the sending data is consistent with what we have seen in the earlier tests. In the other sections of the table however we see degradation in the form of increased EER. In table 4.14 we see the A-measure increasing 2% from sending side to receiving side when compared with both profiles, the R-measure see a between 1.6-1.9% increase while the A+R-measure experience a 2.2-2.8%

		A-measure	R-measure	A+R-measure
Sending	EER	18.3%	16.1%	12.4%
	Std dev	1.21	1.66	1.78
Receiving RP	EER	20.3%	18.0%	14.6%
	Std dev	1.60	2.18	2.09
Receiving SP	EER	20.3%	17.7%	15.2%
	Std dev	1.91	1.37	2.20

Table 4.14: EER with introduced jitter block size 100

increase. When looking at this increases with regards to the standard deviation of the sending section the the A-measure and the A+R-measure SP is the most impacted bought having an EER change of more than 1.6 standard deviation from the mean.

4.3.3 Jitter block size 250

In this subsection we present the EER from the 250 character block size test with 50 ms jitter introduced.

		A-measure	R-measure	A+R-measure
Sending	EER	11.4%	8.8%	7.4%
	Std dev	2.13	1.55	1.24
Receiving RP	EER	15.7%	11.3%	9.8%
	Std dev	1.58	1.58	1.54
Receiving SP	EER	15.1%	11.3%	9.8%
	Std dev	2.18	2.00	1.58

Table 4.15: EER with introduced jitter block size 250

In table 4.15 the sending data is consistent with what we have seen in the earlier tests. When looking at the 250 character test in table 4.15 we also see large changes in the EER from the sending side to the receiving side. For the A-measure the increase is 3.7-4.3% the R-measure increase 2.5% while the A+R-measure increase 2.4%. The largest increases in the 250 character test is more significant at around 2 standard deviations from what is seen in the sending side.

4.4 Package loss

Another aspect Cisco describes that impacts QoS applications are package loss [33], they mention a 1% package loss as the limit. We have preformed one test to see the impact 1% package loss got on our measures.

In table 4.16 we see a similar distribution for the emulation timer as that we have experienced from our other tests. On the network timing we can however see

Emulator time change			Network time change		
Delay change	Number	Percentage	Delay change	Number	Percentage
0	5883	1,36%	0	262445	60,54%
1	259735	61,28%	1	149550	95,04%
2-5	165842	99,98%	2-5	12207	97,86%
6-10	1922	99,99%	6-10	345	97,94%
>10	104	100%	>10	8938	100%

Table 4.16: Time change statistics with introduced package loss

that approximately 2% of the data is at over 10 ms changed delay. This correspond with what we visualized in our example in section 2.4 in table 2.3 given that 1% of the packages is lost and only one character gets sent before the re-transition. Th. We observed a pattern in the data that corresponds with this where there are groups of two characters with high timing change. By looking at these characters we have observed that the first character have an added delay , due to being lost and having to wait for the time out, while the second have gotten received during this wait time and have therefor 0 delay to the first character. Since we are looking at the timing between characters, the difference of the timing of the second character in the group is (zero-original latency).

4.4.1 Package loss full sample

In this subsection we present the EER from the full sample test with a 1% package loss.

		A-measure	R-measure	A+R-measure
Sending	EER	6.8%	4.2%	3.8%
	Std dev	1.10	0.95	0.73
Receiving RP	EER	7.1%	4.1%	4.1%
	Std dev	1.48	0.73	0.79
Receiving SP	EER	6.5%	3.9%	3.9%
	Std dev	1.13	0.71	0.69

Table 4.17: EER with introduced package loss

From the EER in table 4.17 we see no irregularities in the EER on the sending side. We can see that this seem to not impact the EER any more than what we saw in the baseline and the static delay tests. This might be due to the fact that half of the digraphs experiencing high change as seen in the timing table, will be excluded as an outlier due to being less than 30 ms. This might minimize the impact of data loss on the result. This might also explain why we see a degradation when compared the received data against the receiving side profile, that we do not see when comparing the receiving data against the sender side profile. In the

RP comparison, the data with altered timing impacts the profiles as well as the comparison, while in the SP comparison the impacted data will only alter the comparison and not affect the profile.

4.4.2 Package loss block size 100

In this subsection we present the EER from the 100 character block size test with a 1% package loss.

		A-measure	R-measure	A+R-measure
Sending	EER	18.4%	16.0%	12.6%
	Std dev	1.31	1.69	1.53
Receiving RP	EER	19.1%	16.5%	13.5%
	Std dev	2.29	1.76	1.33
Receiving SP	EER	18.5%	16.8%	13.2%
	Std dev	1.67	1.76	1.67

Table 4.18: EER with introduced package loss block size 100

Table 4.18 seem to have a sending side EER consistent with what we are expecting from our previous tests. The most visible pattern in the data from this test is an increase in the EER for the A-mesure on the receiving side compared to the RP is higher than that of the EER when comparing with the SP. In general the values on the receiving side seems to be higher than what we saw in both the baseline and the static delay data, this is a change that was not that visible in 4.4.1, this might be due to the fact that there is less data left to average out the error in each probe.

4.4.3 Package loss block size 250

In this subsection we present the EER from the 250 character block size test with a 1% package loss.

		A-measure	R-measure	A+R-measure
Sending	EER	11.6%	8.6%	7.5%
	Std dev	1.99	1.53	1.26
Receiving RP	EER	13.6%	9.1%	8.1%
	Std dev	2.15	1.29	1.14
Receiving SP	EER	12.2%	9.2%	8.2%
	Std dev	2.00	1.41	1.00

Table 4.19: EER with introduced package loss block size 250

Table 4.19 seem to have a sending side EER consistent with what we are expecting from our previous tests. We see a similar pattern here as we diid in our

100 character sample, where A-measure on the receiving side compared with RP, an a general increase across the board.

4.5 Package loss and jitter

The last test we have conducted is one where we have introduced both jitter and package loss, we have used the same amount of package loss and jitter as we have done in the earlier tests. From the timing statistics of the test we can see that the timing change from the network looks like a combination of the one we saw in our package loss test and jitter, where most of the data is shifted from >10 ms to between 10-50 ms, the difference from the jitter only test is that we hare see 2% of the data with a >50 ms delay change.

Emulator time change		
Delay change	Number	Percentage
0	6619	1,53%
1	261461	61,84%
2-5	163449	99,55%
6-10	1848	99,98%
>10	109	100%

Network time change		
Delay change	Number	Percentage
0	8589	1,98%
1	16559	5,80%
2-5	62797	20,29%
6-10	71116	36,69%
11-25	163632	74,44%
26-50	102156	98,01%
>50	8636	100%

Table 4.20: Time change statistics with introduced package loss and jitter

4.5.1 Package loss and jitter full sample

In this subsection we present the EER from the full sample test with both 50 millisecond jitter and a 1% package loss.

		A-measure	R-measure	A+R-measure
Sending	EER	6.8%	4.1%	3.6%
	Std dev	1.17	0.75	0.93
Receiving RP	EER	8.9%	5.8%	5.2%
	Std dev	0.71	1.04	1.23
Receiving SP	EER	7.5%	5.6%	4.5%
	Std dev	1.39	1.02	1.09

Table 4.21: EER with introduced package loss and jitter

This test is where we have introduced most interference, and from the results we can see that it also has the highest impact on our EER. In table **??** we can see an almost two standard deviation change in all the measures on the receiving side from what is seen on the sender side. The R-measure experiencing the highest change in the form of 2.26 standard deviations from the mean on the sender side to the receiver. When comparing the two different profiles on the receiving data, we can see the same pattern that we have earlier observed. Where the EER when compared with the SP is noticeably lower than what we saw from the RP. The change seems to be largest when looking at the A-measure and the A + R-measure, while the R-measure is pretty stable. We can see a similar pattern for the EER in section 4.3. Signs of the pattern can also be seen in section 4.4, in in this section it is however somewhat more uncertain, since we see a distance between receiver side RP and SP EER comparison, however they are approximately the same distance from the sender side EER.

4.5.2 Package loss and jitter block size 100

In this subsection we present the EER from the 100 character block size test with both 50 millisecond jitter and a 1% package loss.

		A-measure	R-measure	A+R-measure
Sending	EER	18.2%	16.0%	12.9%
	Std dev	1.29	1.78	1.96
Receiving RP	EER	22.8%	18.9%	16.0%
	Std dev	2.8	2.7	0.9
Receiving SP	EER	20.9%	18.6%	15.4%
	Std dev	2.26	2.71	1.47

Table 4.22: EER with introduced package loss and jitter block size 100

In this test we can see that there is a high impact on the EER across all receiving side measures as seen in table 4.5. A pattern that seem to emerge here that was not as distinct in 4.5.1 is that the EER is somewhat lower when using the SP as compared to the RP. As suggested in subsection 4.4.1 this could be the fact that when comparing with RP bought the profile and the probe is impacted, will for the SP only the probe is impacted.

4.5.3 Package loss and jitter block size 250

In this subsection we present the EER from the 250 character block size test with both 50 millisecond jitter and a 1% package loss.

In this test we can see that there is a high impact on the EER across all receiving side measures as seen in table 4.7. The same pattern is visible here as saw in the 100 character test data, where the EER is generally lower in the SP comparison then the RP.

		A-measure	R-measure	A+R-measure
Sending	EER	11.4%	8.9%	7.5%
	Std dev	2.05	1.61	1.02
Receiving RP	EER	15.4%	12.8%	10.9%
	Std dev	1.73	1.00	1.09
Receiving SP	EER	14.9%	11.2%	10.2%
	Std dev	1.25	1.57	1.32

Table 4.23: EER with introduced package loss and jitter block size 250

4.6 Sampling error

Something we observed from our testing, was the fact that there was some irregularity of the timing of the captured data compared to the presented data. This was especially present when testing the capturing script by holding one key down, where sometimes multiple keys were registered simultaneously, and sometime it would be 100 ms delay between them. This also held true when looking at locally recaptured data. When trying to find documentation on the sampling rate used by bpftrace we could not find this specified. We have conducted a test to look at the sampling by sending 100 keys at each timing in the interval 0ms to 50ms to see if we could see a pattern of irregularities with low latency key presses. There was no sign of the scale of irregularities as we observed when holding down a key. This might be tied to either the emulation delay spacing the keys out enough for the problem to disperse, or that the keystroke when holding down a key is sent at different timings based on factors in the operating system. By the result it seem that the sampling rate seem to be somewhere between 1-2 milliseconds. We have observed some instances of simultaneous keys in our data that seem to be caused by network jitter lining up keys with a low inter key latency.

4.7 General

We have in all our result presented two different comparisons of the measures on the receiving side. One where the reference profiles were generated from the SP, and one where the reference profiles were generated from the RP. In our results we see a trend toward comparison with SP giving slightly better EER in cases where we add interference to the network channel, while the profiles are comparable with low to no interference. In our testing the profiles have been generated on both sides for each test, while in a real life scenario a choice on where to capture the users profile would have to be made. An impotent thing to considerate there is that the profile should be applicable both in ideal conditions, as well as variable different network environment that could impact the traffic. Using an approach where the participants profile are generated onsite will give the most accurate measurement, however if the participant is mainly connecting via a network connection that introduces some latency, the recaptured data might give a more accurate view of the normal typing pattern the server will observe. From our data it seems like it would be a good approach to generate new profile by getting the user's profile generated locally from the computer instead of over network. This approach is also where the profile is most accurate to the actual typing of the user, with the least chance of interference from any sources, and in a real life scenario without any emulation delay this will be the real typing time of the user.

In our test we have excluded outliers where the digraph latency was less than 30 ms or over 500 ms. This works for our testing where there is no one trying to misuse the system, however given a real implementation such an outlier exclusion policy will lead to a potential security flaw. If an imposter were able to recognise this mechanism, they could circumvent the system by typing with a slow typing speed. At this point mechanisms to look for high amount of outliers could help prevent miss use.

4.8 Discussion

In our research we have utilized Clarckson's keystroke database as our source for user input. This gives a good source for free text that works well in keystroke dynamics in general. A thing to keep in mind is that this dataset and the content of it is not in line with normal content seen in an SSH session. SSH use in system administration is often used to conduct changes on a server's configuration. This will mostly consist of shorter predefined cli(Command-line interface) commands typed repeatedly. This might open for giving extra weighting to some specific digraph of commands a user actively uses. By looking at commands that can be written in different ways a heavier emphasis could be placed on deviation from the users normal behavior. Another point to bring up is that a lot of changes can be done to a server with a small number of characters. An example is that a server could be shut down given the right permission with less than 20 characters. With in our testing we have tested using 100, 250 as well as the total test consisting of between 500 and 1300 character per probe. As mentioned earlier our research have not aimed at creating the most accurate EER, somthing we can see from our research however is that a increase in the probe size increase the accuracy of the measures. Another momentum is that a lot of work done over SSH consists of following procedures, where a user will copy the text from the procedure, and past it into the SSH session. This kind of use will not give us any timings that can be used for keystroke dynamics. This makes our test a way of seeing that an SSH session is not interfering too much with the keystrokes in general. However we can not guarantee the applicability of keystroke dynamics with real life traffic from an SSH connection. A thing to keep in mind when bringing up this is the fact that we have not looked to optimize the EER in our project. Previous research point at a combination of A and R- measure of multiple n-graphs being more accurate than just the digraph that have been focused on in this project.

From our results we can see that a low latency network connection dose not impact the ability to conduct continues authentication using the data. A trend that seem to emerge is how the more data used in the measure the more stable it seem to be. By viewing the tables in this chapter is that the standard deviation seem to change most in the 100 block size test. A less obvious trend that seem to also point at this is how the A-measure seem to be the measure where we see the highest spread, while he A+R-measure is the one experience the most stable standard deviation. This might hint to the fact that applying a highly accurate measure might give an increased robustness against interference.

An intention when starting this project was to look at true continuous authentication as well as continuous authentication. After the implementation and the testing of the R and A-measure this however showed to be infeasible due to the time constraint of the project. A contribution we can make in this regard however is that the data size for each participant need to be larger than what we get from the Clarkson dataset. We looked at the statistics of a trust model with the data in the dataset consisting of both the free text and the transcription tasks. This gave approximately 20000 characters per participant. Applying a trust model approach as described in chapter 2 with 80% of the data going to training and 20% going to testing, did not give a significant enough difference between the imposters and the genuine user. With a similar amount of participants a full scale data collection with double this amount would run for around 10 days. This would lead to us needing to focus all our effort on this to be able to gather similar data.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The results of our test gives an indication that keystroke dynamics can be applied on keystroke latency's gathered from the server side of an SSH connection on free text data. This result however mostly points to the functionality of keystroke dynamics on data gathered with network delay. The dataset used in this project is somewhat atypical of the normal typing seen over an SSH channel. In this project we have seen that there is a sufficient amount of data that can be extracted from an SSH session to conduct continuous authentication. The reduced amount of data that is expected from an SSH session will put a higher strain on the authentication system. We have seen that a large amount of jitter on the network traffic will lead to degradation of the functionality of continuous authentication. If an implementation of continuous authentication or true continuous authentication on an SSH channel was to be introduced, a decision of how to capture data to train the reference profiles will have to be made. From what we have seen in our test an approach of locally captured data seems to be more suitable than capturing on a server. Another alternative that seems to work is to do the data capture remotely using a server given that it can be confirmed that the data has not experienced any noticeable interference over the network, this however might lead to data needing to be discarded, and the user having to input more data to compensate. The problem here is to determine what data should be discarded without directly knowing which part of the data is impacted by network interference. With a more realistic user pattern for SSH the data amount will in most cases be more limited both in the form of a reduced character set as well as reduced length. In the testing conducted in this project the probes have been approximately 500 digraphs long. With the decreased amount of data to work on in a real life scenario, an authentication mechanism that can take an authentication decision with a lower amount of data is needed for an functional implementation of a network based continuous authentication system.

5.2 Further work

In this project we have utilized a publicly available keystroke dataset from Clarkson University, and utilized the free text data found in it. A shortcoming of this is that the typing data of the dataset is not consisted with what would normally be seen in an SSH session. In the dataset we see structured sentences answering questions, while a more normal SSH interaction to a large extent is consisting of different CLI commands and paths. In future research into this topic we would suggest looking at conducting data collection of actual SSH traffic. An alternate perspective to look at here is to gather the profile from normal typing tasks, and the probes from CLI tasks, to see if the typing pattern are consistent enough to be used in combination.

We have seen that the data amount we utilized in this project is to limited to generate a profile to be used in a trust model approach for continuous authenticating. We suggest looking at an extended data amount in combination with network captured keystroke dynamics, to be able to test the impact of network delay on a trust model based continuous authentication system. In our mind, free text data gathered from open tasks would fit better to generate large amount of data compared with CLI tasks. So also here the interaction between a typical SSH session's traffic, with a more generally gathered profile would be interesting.

We have in this section described SSH specific tasks since this has been our focus. The suggestion hold true for all network communication that sends characters one by one.

Bibliography

[1] M. Johns, ‘Session hijacking attacks,’ in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 1189–1190. DOI: 10.1007/978-1-4419-5906-5_661.

[2] R. S. Gaines, W. Lisowski, J. P. S and N. Shapiro, ‘Authentication by keystroke timing: Some preliminary results,’ *RAND report, pp. 123-132*, May 1980.

[3] P. S. Teh, A. Teoh and S. Yue, ‘A survey of keystroke dynamics biometrics,’ *The Scientific World Journal*, 2013. DOI: 10.1155/2013/408280.

[4] S. Mondal, ‘Continuous user authentication and identification: Combination of security forensics,’ Ph.D. dissertation, 2016. DOI: 10.13140/RG.2.1.1152.0882.

[5] S. Banerjee and D. Woodard, ‘Biometric authentication and identification using keystroke dynamics: A survey,’ vol. 7, pp. 116–139, 2012.

[6] A. Morales, M. Falanga, J. Fierrez, C. Sansone and J. Ortega-Garcia, ‘Keystroke dynamics recognition based on personal data: A comparative experimental evaluation implementing reproducible research,’ 2015. DOI: 10.1109/BTAS.2015.7358772.

[7] F. Bergadano, D. Gunetti and C. Picardi, ‘User authentication through keystroke dynamics,’ *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 367–397, Nov. 2002. DOI: 10.1145/581271.581272.

[8] D. Gunetti and C. Picardi, ‘Keystroke analysis of free text,’ *ACM transactions on information and system security*, vol. 8, no. 3, pp. 312–347, 2005.

[9] H. Davoudi and E. Kabir, ‘A new distance measure for free text keystroke authentication,’ *2009 14th International CSI Computer Conference*, DOI: 10.1109/csicc.2009.5349640.

[10] H. Davofudi and E. Kabir, ‘Modification of the relative distance for free text keystroke authentication,’ *2010 5th International Symposium on Telecommunications*, DOI: 10.1109/ISTEL.2010.5734085.

[11] H. Ferreira J. Santos, ‘Keystroke dynamics for continuous access control enforcement,’ *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2012. DOI: 10.1109/CyberC.2012.43.

[12] J. Hu, D. Gingrich and A. Sentosa, 'A k-nearest neighbor approach for user authentication through biometric keystroke dynamics,' in *2008 IEEE International Conference on Communications*, 2008, pp. 1556–1560. DOI: `10.1109/ICC.2008.301`.

[13] J. Huang, D. Hou and S. Schuckers, 'A practical evaluation of free-text keystroke dynamics,' in *2017 IEEE International Conference on Identity, Security and Behavior Analysis (ISBA)*. DOI: `10.1109/ISBA.2017.7947695`.

[14] A. Kolakowska, 'User authentication based on keystroke dynamics analysis,' *Computer Recognition Systems 4.*, 2011. DOI: `10.1007/978-3-642-20320-6_68`.

[15] P. Kang and S. Cho, 'Keystroke dynamics-based user authentication using long and free text strings from various input devices,' *Information Sciences*, vol. 308, pp. 72–93, 2015. DOI: `https://doi.org/10.1016/j.ins.2014.08.070`.

[16] A. Messerman, T. Mustafić, S. A. Camtepe and S. Albayrak, 'Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics,' in *International Joint Conference on Biometrics (IJCB)*, 2011. DOI: `10.1109/IJCB.2011.6117552`.

[17] P. Pinto, B. Patrão and H. Santos, 'Free typed text using keystroke dynamics for continuous authentication,' in *Communications and Multimedia Security*, B. De Decker and A. Zúquete, Eds., 2014.

[18] K. A. Rahman, K. S. Balagani and V. V. Phoha, 'Making impostor pass rates meaningless: A case of snoop-forge-replay attack on continuous cyber-behavioral verification with keystrokes,' in *CVPR 2011 WORKSHOPS*, 2011. DOI: `10.1109/CVPRW.2011.5981729`.

[19] P. Bours, 'Continuous keystroke dynamics: A different perspective towards biometric evaluation,' *Information Security Technical Report*, vol. 17, no. 1-2, pp. 36–43, 2012.

[20] S. Mondal and P. Bours, 'Performance evaluation of continuous authentication systems,' *IET Biometrics*, 2015. DOI: `10.1049/iet-bmt.2014.0070`.

[21] T. Ylonen and C. Lonvick, 'The secure shell (ssh) transport layer protocol,' RFC Editor, RFC 4253, Jan. 2006. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc4253.txt`.

[22] T. Ylonen and C. Lonvick, 'The secure shell (ssh) protocol architecture,' RFC Editor, RFC 4251, Jan. 2006. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc4251.txt`.

[23] D. X. Song, D. Wagner and X. Tian, 'Timing analysis of keystrokes and timing attacks on ssh,' in *Proceedings of the 10th Conference on USENIX Security Symposium*, USA, 2001.

[24] V. Cerf, Y. Dalal and C. Sunshine, 'Specification of internet transmission control program,' RFC Editor, RFC 675, Dec. 1974. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc675.txt`.

[25] J. Postel, 'Transmission control protocol,' RFC Editor, STD 7, Sep. 1981. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc793.txt`.

[26] J. Nagle, 'Congestion control in ip/tcp internetworks,' RFC Editor, RFC 896, Jan. 1984. [Online]. Available: `https://www.rfc-editor.org/rfc/rfc896.txt`.

[27] 'The effects of jitter on the peceptual quality of video,' *Proceedings of the seventh ACM international conference on Multimedia (Part 2), author=Claypool, Mark and Tanner, Jonathan, year=1999*, DOI: `10.1145/319878.319909`.

[28] S. McCanne and V. Jacobson, 'The bsd packet filter: A new architecture for user-level packet capture,' USENIX'93,San Diego, California: USENIX Association, 1993, p. 2.

[29] F. Ellis, *Ebpf, part 1: Past, present, and future, 2017*, `https://www.ferrisellis.com/content/ebpf_past_present_future/`, Accessed: 2022-04-18.

[30] E. Vural, J. Huang, D. Hou and S. Schuckers, 'Shared research dataset to support development of keystroke authentication,' *IEEE International Joint Conference on Biometrics*, 2014. DOI: `10.1109/btas.2014.6996259`.

[31] Unknown, *Virtual-key codes (winuser.h) - win32 apps*, Accessed: 2021-10-18. [Online]. Available: `https://docs.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes`.

[32] *Ping time between oslo and other cities*, Accessed: 2022-04-20. [Online]. Available: `https://wondernetwork.com/pings/Oslo`.

[33] *Video quality of service (qos) tutorial*, Accessed: 2022-03-10, Sep. 2017. [Online]. Available: `https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-video/212134-Video-Quality-of-Service-QOS-Tutorial.html`.

NTNU

Kunnskap for en bedre verden