Hardening of a Continuous Behavior-based Authentication Distributed System

Julián González-Muñoz, Mario Casado, Daniel Garabato, Francisco J. Nóvoa, and Carlos Dafonte

Telematic Group, Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
CITIC Research Center, Computer Science and Information Technologies Department, Campus de Elviña, 15071 A Coruña, Spain
Correspondence: {j.gonzalezm, mario.diez, daniel.garabato, fjnovoa, carlos.dafonte}@udc.es

DOI: *https://doi.org/10.17979/spudc.000024.31*

Abstract: Password-based traditional authentication systems are increasingly insufficient when it comes to providing security and checking the identity of the authenticated user. What happens when the password of an user has been stolen or an active user is not the same user who authenticated firstly?
A distributed system using AI (Artificial Intelligence) acting as a second factor authentication method by analyzing user's mouse events has to provide confidentiality and integrity in order to protect against different attacks such as Man-In-The-Middle that allow sniffing or data tampering, resulting in an identity spoof.
In order to grant integrity and confidentiality, encryption and authentication must be implemented. Authentication is used to allow one node to produce or consume data from an existent message stream and encryption in order to avoid exposing these data to external agents.
PKI (Public Key Infrastructure) system is widely used over the internet, so it is a trusty authentication and encryption framework. By using PKI in this project, hardening is performed by creating with OpenSSL a trusted Certificate Authority that issues and signs the certificates used by each node in the distributed system. Trust in this Certificate Authority is implemented by creating keystores and truststores for each node with keytool. This project resulted in a secure communication system preventing data from being sniffed or tampered.

1 Introduction

Password-based traditional authentication systems are increasingly insufficient when it comes to providing security and checking the identity of the authenticated user. Biometric identification, physical keys or password policies came up as different solutions to manage these problems, but, even so, these solutions act as a first security barrier. Incident handling techniques or security controls try to increase trustness in traditional authentication systems by using security policies in order to force re-authentication periodically, dificulting identity spoofing of an authenticated user but falling again in the problem of using one isolated event as an identity checking system. At the same time, forcing an user to re-authenticate periodically can worsen the user experience or the user performance during a job.

The need to apply a new authentication system based on a different method of identity demonstration comes up after analyzing the weaknesses of the traditional authentication methods. A distributed system has been built in order to provide a continuous behavior-based

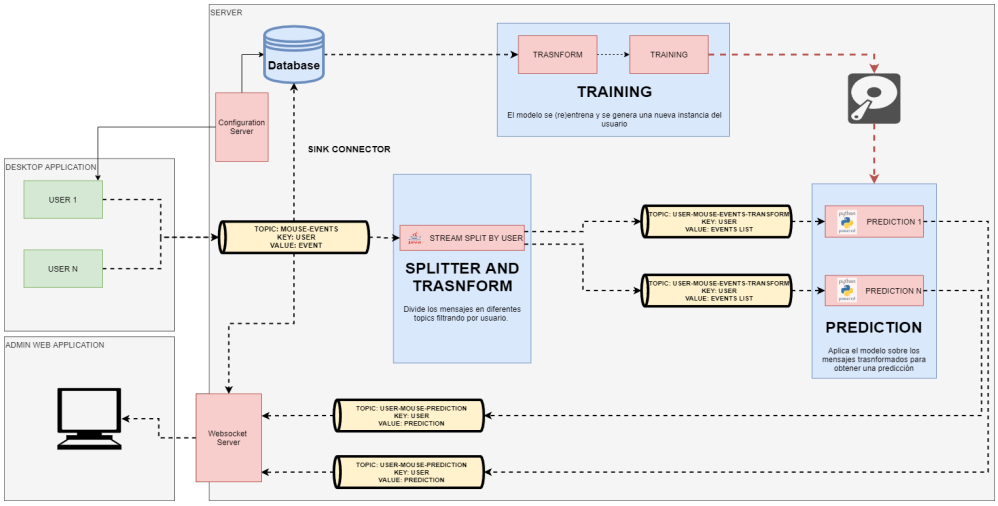


Figure 1: Distributed system deployed in this project

authentication method based on AI (Artificial Intelligence) acting as a second factor authentication method.

Once the system is working and providing authentication service, there is a need to apply security to its infrastructure and communications in order to avoid identity spoofing.

In Section 2 the architecture of the distributed system is explainend in order to contextualize the starting point. Next, in Section 3 the hardening methodology and PKI will be explained as well as the tools used to implement PKI. Finally, in Section 4 conclusions are discussed.

2 System Architecture

The distributed system used in this project uses Kafka messaging system to process all the information gathered from the user mouse movements [Narkhede et al. (2017)].

User mouse movements are sent via Kafka queues to the central node (broker) of the system. Once there, mouse events are splitted and transformed through Kafka Streams in order to use these transformed events as characteristics. The created characteristics are returned to the Kafka system via different queues splitted by user, these queues will be used by the final inference AI system to determine whether an active user is the one previously authenticated or not (Figure 1) [Silvelo et al. (2020)].

By using Confluent Kafka, we can encrypt communications with SSL and authenticate them with the use of different authentication protocols such as SASL, mTLS or HTTP Basic Auth.

3 Hardening of the system

A lot of advantages exist when having a distributed system like the one used in the project (Figure 1). One of them is that it allows the possibility to split each node of the system in different places or servers. In addition, the client-side application will be always communicating through the Internet.

Considering these scenarios and the possibility of a malicious user trying to tamper the data, the communications among nodes of the system must be protected through different methods. The first method of protection is to encrypt communications in order to avoid sniffing and tam-

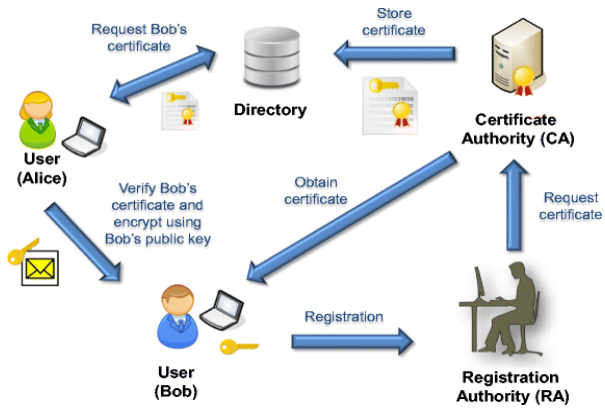


Figure 2: PKI simple schema [Tan et al. (2015)].

pering techniques. The other protection method consists of the authentication among nodes when consuming or producing data, this authentication would avoid malicious agents getting potentially confidential data from the system or producing and sending data to the system.

As mentioned in the previous chapter (Chapter 2) mTLS can be implemented in order to encrypt communications and perform authentication.

3.1 PKI

PKI is the framework that enables entities to securely exchange information using digital certificates. The components that form PKI include the hardware, software, policies, procedures and entities needed to safely distribute, verify and revoke certificates [Ellison (1999) Wood (2002)] (Figure 2).

Key elements of the PKI are:

- **Certificate Authority (CA):** Trusted party who provides certificates and authenticates their identity.
- **Registration Authority:** Party allowed by CA to issue certificates
- **Certificate store:** Enables programs running on the system to access stored certificates, certificate revocation lists (CRLs) and certificate trust lists (CTLs).
- **Certificate database:** This database stores information about issued certificates. In addition to the certificate itself, the database includes the validity period and status of each PKI certificate.

PKI has been used in this project by creating our own Root CA in order to issue all the needed digital certificates to encrypt and authenticate communications among nodes.

OpenSSL

OpenSSL is the world's most widely used implementation of the Transport Layer Security (TLS) protocol. It provides a set of command-line tools that serve a variety of purposes, including support for common PKI operations and TLS testing [Ristić (2023)].

The creation of a private CA is performed by using the OpenSSL command-line tool, creating a CA root certificate which is used to sign all the CSR (Certificate Signing Request) of the distributed system. Signed certificates of each node will be then stored in Keytool's keystores and the private CA will be trusted by all of the nodes by using Keytool's truststores.

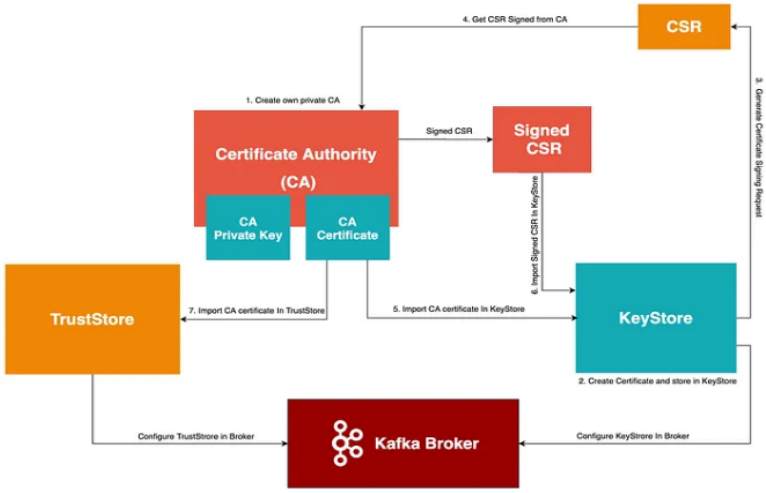


Figure 3: TLS configuration example in the Kafka Broker [Pandey (2020)]

Keytool

Keytool is a Java command-line tool which allows us to manage a keystore (database) of cryptographic keys, X.509 certificate chains, and trusted certificates [Oracle (2020)]. In this project, PKCS12 databases were created with Keytool to store signed certificates signed by the created CA (keystore) as well as the CA root certificate (truststore). Both keystores and truststores can contain more than one certificate.

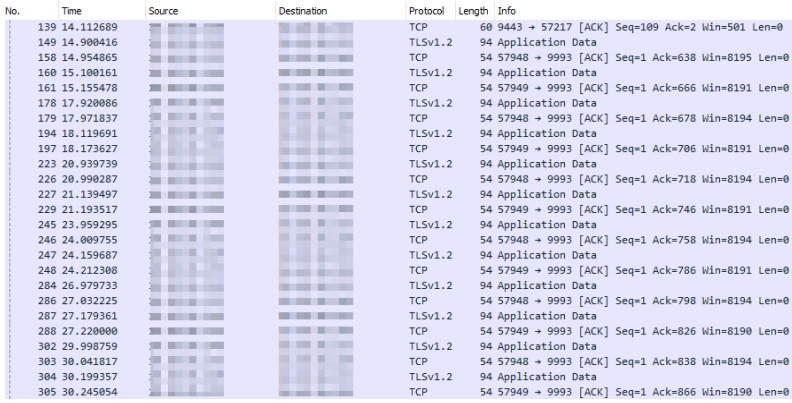
4 Conclusions

The creation of a private CA with OpenSSL (Section 3.1) allows to sign the CSR of each node thus creating each node's signed certificate and storing it in their Keytool's keystores. These signed certificates will be trusted by using Keytool's truststores where the CA root certificate is stored. This implementation can be shown in the Figure 3 where the structure of the mTLS communication configuration is represented.

The implementation of continuous authentication as a second factor method is a need nowadays, it provides an extra layer of security in the systems but it is not enough if it is not properly hardened. By using PKI, data flows in the distributed system are encrypted and authenticated, preventing identity spoofing from malicious actors (Figure 4).

Acknowledgements

This work was funded by the Spanish MCIN/AEI/10.13039/501100011033 and European Union Next Generation EU/PRTR through grant TED2021-130492B-C21 and the Galician Regional Government, Xunta de Galicia, through grants ED431B 2021/36 and ED431G 2019/01



No.	Time	Source	Destination	Protocol	Length	Info
139	14.112689	[redacted]	[redacted]	TCP	60	9443 → 57217 [ACK] Seq=109 Ack=2 Win=501 Len=0
149	14.900416	[redacted]	[redacted]	TLSv1.2	94	Application Data
158	14.954865	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=638 Win=8195 Len=0
160	15.100161	[redacted]	[redacted]	TLSv1.2	94	Application Data
161	15.155478	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=666 Win=8191 Len=0
178	17.920086	[redacted]	[redacted]	TLSv1.2	94	Application Data
179	17.971837	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=678 Win=8194 Len=0
194	18.119691	[redacted]	[redacted]	TLSv1.2	94	Application Data
197	18.173627	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=706 Win=8191 Len=0
223	20.939739	[redacted]	[redacted]	TLSv1.2	94	Application Data
226	20.990287	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=718 Win=8194 Len=0
227	21.139497	[redacted]	[redacted]	TLSv1.2	94	Application Data
229	21.193517	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=746 Win=8191 Len=0
245	23.959295	[redacted]	[redacted]	TLSv1.2	94	Application Data
246	24.009755	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=758 Win=8194 Len=0
247	24.159687	[redacted]	[redacted]	TLSv1.2	94	Application Data
248	24.212308	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=786 Win=8191 Len=0
284	26.979733	[redacted]	[redacted]	TLSv1.2	94	Application Data
286	27.032225	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=798 Win=8194 Len=0
287	27.179361	[redacted]	[redacted]	TLSv1.2	94	Application Data
288	27.220000	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=826 Win=8190 Len=0
302	29.998759	[redacted]	[redacted]	TLSv1.2	94	Application Data
303	30.041817	[redacted]	[redacted]	TCP	54	57948 → 9993 [ACK] Seq=1 Ack=838 Win=8194 Len=0
304	30.199357	[redacted]	[redacted]	TLSv1.2	94	Application Data
305	30.245054	[redacted]	[redacted]	TCP	54	57949 → 9993 [ACK] Seq=1 Ack=866 Win=8190 Len=0

Figure 4: Encrypted communications between desktop client and broker

Bibliography

C. M. Ellison. The nature of a useable pki. *Computer Networks*, 31(8):823–830, 1999.

N. Narkhede, G. Shapira, and T. Palino. *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.", 2017.

Oracle. keytool — docs.oracle.com. *https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html*, 2020.

M. Pandey. Kafka SSL: Setup with self signed certificate—Part 1 — medium.com. *https://medium.com/jinternals/kafka-ssl-setup-with-self-signed-certificate-part-1-c2679a57e16c*, 2020.

I. Ristić. Library: OpenSSL Cookbook 3ed — Feisty Duck — feistyduck.com. *https://www.feistyduck.com/library/openssl-cookbook/*, 2023.

A. Silvelo, D. Garabato, R. Santoveña, and C. Dafonte. A first approach to authentication based on artificial intelligence for touch-screen devices. *Proceedings*, 54(1), 2020.

S.-Y. Tan, W.-C. Yau, and B.-H. Lim. An implementation of enhanced public key infrastructure. *Multimedia Tools and Applications*, 74(16):6481–6495, Aug 2015.

D. Wood. Pki, the what, the why, and the how. *GSEC–SANS Institute*, 2002.