Sheila Jimenez Morejon
Zhenghui Wang
CS568: Applied Cryptography
April 22, 2020

The Kerberos Authentication Protocol

Problem Overview

Hackers and malicious actors are often seen as an external individual or organization trying to gain entry to some computer or network system. To protect against outside threats, experts developed the concept of a firewall, a network security system that monitors and controls all external traffic on a network system. While firewalls protect against malicious servers on the outside, it does little to protect the network system from internal threats, which are extremely common ("About Firewalls").

This is a problem common with distributed systems, where a network of computers share resources and provide different computers, or servers, provide the user different capabilities. Users have different powers associated with their account in terms of services and information accessible to them. Proper proof of identification is vital to maintain confidentiality in users' data as well as preventing malicious manipulation of users' access. Consider a user with access to a client server. The client server can ask other servers for certain services, such as accessing their email or accessing the printers. However, without a proper authentication protocol, the email and printer servers have no idea who is sending the requests and are, hence, vulnerable to exploitation by other internal servers that wish to have access to the users' data and power in the network.

For example, let's say Alice wants to access her email when logging in to her school account. Alice can request access to her email, but if the email server does not require Alice to prove her identity, anyone can access her email and breach her privacy. Even if the authentication protocol for email access requires some password, the system is vulnerable. When Alice sends some unique identification or password to prove her identity, Mallory, who is listening over the network, now has Alice's information and can access Alice's email. This begs the question of how to authenticate oneself to a service without Mallory gaining the data needed to impersonate Alice.

Problem Details

We next define the abilities of the malicious party, otherwise known as Mallory. Suppose again that Alice wants to access her school account email. To do this, Alice must first prove to the email server that she is in fact Alice. Unfortunately for her, Mallory is able to observe any packets that Alice sends or receives over the network. This threat model is formally known as a passive eavesdropper attack. In this model, Mallory can only read the data over the network. She cannot modify or drop packets being sent over the network, nor can she send a new packet that did not originate from Alice or the server.

If Alice simply sends her username and password to the server, then Mallory gains access to Alice's email, rendering the authentication protocol useless. If Alice encrypts her username and password, then sends it to the server, Mallory learns nothing about Alice's information. However, the server will not be able to decrypt the data unless Alice and the server perform some expensive cryptographic techniques. For example, Alice and the server could have a shared key to encrypt their information with, but this results in an issue of deriving a shared secret key used to encrypt and decrypt. We will consider this problem to be solved if the server successfully authenticates Alice without Mallory gaining any new, sensitive knowledge regarding Alice.

We also consider the case of a replay attack, that Mallory intercepts the packets Alice sends to the server with important information and uses it later on to gain access to Alice's information once Alice is already authenticated. It is important to note the importance of the replay attack can be executed over and over, meaning Mallory has access to Alice's information for an indefinite amount of time ("What Is a Replay Attack?"). We will consider these problems solved if Mallory cannot execute a replay attack.

A more powerful Mallory has even more ways of compromising Alice's privacy. If Mallory can intercept and alter packets, she can execute a man-in-the-middle-attack. When Mallory impersonates a server, Alice believes she is communicating with the Kerberos server and may send sensitive information to Mallory without meaning to. This compromises the privacy and confidentiality of the connection between Alice and other non-malicious servers ("What Is a Man-in-the-Middle Attack?"). We will consider this problem solved if Mallory is unable to obtain information regarding Alice despite impersonating as the servers Alice attempts to communicate with.

Solution Overview

Researchers at MIT recognized this problem, and developed the Kerberos protocol to provide both security and authentication for computer systems. The protocol relies on the use of tickets to authenticate users instead of on sending passwords over a network as well as a trusted server. In this scenario, the Kerberos server, trusted by both the client Alice and other servers that provide HTTP services, has access to all the servers' passwords and can, hence, undertake all server authentication. The use of a third server acting as a middle man also prevents the user from gaining access to other servers' passwords and vice-versa. Alice and the HTTP server never communicate unless both parties have been safely authenticated, preventing a malicious observer, Mallory, from impersonating Alice.

Before communicating with the service that Alice wants to access directly, Alice must first communicate with the trusted server. The trusted server consists of the authenticating server and the ticket granting server (TGS). First, Alice sends in plaintext a unique identifier (username or ID), the name of the requested service (TGS), her network address, and the requested lifetime of the Ticket Granting Ticket (TGT), to the authenticating server. The authenticating server will then check if Alice is a valid user, then sends back the TGT that is encrypted with the server's secret key, and TGS Session Key that is encrypted with client secret key derived from Alice's password. The TGT contains Alice's ID, a timestamp, Alice's network address, the lifetime of the TGT, and the TGS Session Key.

With this, Alice first decrypts the TGS Session Key, then communicates with the TGS. She prepares the Authenticator, which is encrypted with the TGS Session Key and consists of her ID and the timestamp, and sends the TGS this along with the TGT and a plaintext request specifying what service she is requesting for (school email) and the lifetime of said request. The TGS checks the validity of the service request, then decrypts the TGT with the server secret key to obtain the TGS Session Key. With that, the TGS decrypts the Authenticator, compares the information with the information in the TGT, and, if the Authenticator is valid and not already in the TGS's cache, generates a HTTP Service Session Key. The TGS then prepares the HTTP Service Ticket, which is encrypted with the HTTP Service Secret Key and contains Alice's ID, the HTTP Service name, Alice's network address, a timestamp, the lifetime of the ticket, and the HTTP Service Session Key. Alice receives the HTTP Service Ticket, along with another message that is encrypted with the TGS Session Key which contains the HTTP Service name, a timestamp, lifetime of the HTTP Service Ticket, and the HTTP Service Session Key.

Alice decrypts the second message with the TGS Session Key. She then prepares another Authenticator encrypted with the HTTP Service Session Key that she obtained by decryption. She sends that, along with the HTTP Service Ticket, to the HTTP service. The HTTP Service decrypts the HTTP Service Ticket with its secret key to see the HTTP Service Session Key, and uses that to decrypt the Authenticator and compares it with the information in the HTTP Service Ticket. If the Authenticator is valid and not already in the HTTP service's cache, the service sends Alice an Authenticator message containing the service's ID and a timestamp encrypted with the HTTP Service Session Key.

Alice decrypts the Authenticator from the service, and now Alice has been authenticated. All future requests to the HTTP service can be made by using the HTTP Service Ticket.(MIT)

Solution Details

The Kerberos protocol prevents fraudulent attempts to access servers in several ways. The backbone of this protocol is safe transmission of session keys between servers and the lifetime of authenticators. The distribution of tasks between several different servers allow the user and the Kerberos servers to safely share keys without the fear of an adversary gaining access to those keys. Each step in the protocol also has a type of intercalation of encryption. Each server requires different tickets to be encrypted with a specific key, so Mallory, even if she is able to forge some tickets, is unable to fully authenticate herself.

As a passive eavesdropper, Mallory can attempt to impersonate Alice to the Kerberos server and/or HTTP servers by sending copies of the packets Alice sends through her own network, but two measures taken in the Kerberos protocol prevents this. Firstly, Mallory is unable to decrypt much of the information sent back to Alice by the servers. Consider the first request to the authenticating server. Since Mallory has access to Alice's unique identifier and IP address, Mallory can make the first request to the authenticating server as Alice. However, when the authenticating server gives Mallory the session key, Mallory is unable to decrypt it as she does not have access to Alice's password. When Mallory attempts the same strategy with other servers by resending Alice's packets, Mallory will not be able to access information needed to connect with the next server; even if Mallory attempts to send Alice's authenticator and HTTP ticket to the HTTP server, the server will not be able to decrypt the authenticator and deem the request to be invalid.

The second measure actually even prevents the servers from sending anything back to Mallory at all. Servers keep track of which and what number of authenticators are received in their cache. When Mallory resends Alice's authenticator, the server identifies the authenticator as one of Alice's authenticators that has already been used and does not send back any information.

These scenarios demonstrate that Alice can prove her identity to all the servers without Mallory gaining new information to impersonate Alice. Furthermore, Mallory cannot forge authenticators and masquerade as Alice. The Kerberos protocol thus withstands the passive-eavesdropper threat model defined above.

Authenticators also prevent replay attacks. If Mallory intercepts one of Alice's authenticators and sends it at a later time, the server will accept the authenticator as it is the first time said authenticator has been used. However, the authenticator contains a timestamp, which the server compares to the timestamp of its corresponding ticket. If the timestamps differ enough, the server will not accept the authenticator as valid and not send back any information. The authenticator then prevents Mallory from intercepting Alice's packets and pretending to be Alice to the TGT and HTTP server. This solves the problem of replay attacks.

Mallory, however, can still continue her attempts to impersonate Alice with other servers. Once Alice has authenticated herself to the Kerberos server, Mallory can piggyback off this authentication and try to pretend she is Alice to other servers. Since Mallory knows the encrypted TGT and Alice's identifier, the only thing left to do is forge the authenticator and Alice's HTTP request. While the HTTP request is not encrypted and, hence, easily forged, the authenticator is more difficult to forge since it is encrypted with the TGS session key, which Mallory does not have access to. This means Mallory is unable to prove to the TGS that she is Alice. A similar scenario happens when Mallory pretends to be Alice to the HTTP server. While Mallory has access to the encrypted HTTP service ticket, she does not have access to the HTTP service session key, so she is unable to forge the authenticator sent to the HTTP service.

The Kerberos Protocol also includes authentication of the HTTP server, which prevents Mallory from impersonating as the HTTP service. Consider the situation of the man-in-the-middle attack where Mallory impersonates the HTTP Server. When Alice sends her authenticator and HTTP service ticket to what she believes is the HTTP Server, but is in fact Mallory, Mallory cannot decrypt Alice's authenticator and gain access to the HTTP session key. This means Mallory cannot send an authenticator encrypted with the proper key. This solves the problem of Mallory impersonating another server and gaining information from Alice.

Description of The Kerberos Authentication Protocol Demo (KAPD)

The demo uses object-oriented programming to represent different parts of the Kerberos Authentication protocol (KAPD). The three main files http.py, user.py, and kerberos.py represents the HTTP server, the client, and the Kerberos server respectively. The encryption.py contains the algorithms used to encrypt the information between the user, Kerberos and HTTP server. We note that only the elements of the lists sent between the “network” are encrypted rather than the packets being encrypted. Lastly, the main.py file connects the files listed above into one coherent program that mirrors a single request from a user to some HTTP service.

KAPD can be broken down into three main classes. The first is the user class. The user has methods to create an account, request some service, and process the replies from the Kerberos and HTTP server. Like in the real KAP, the user is unable to decrypt the TGT and the HTTP ticket as they do not have access to the other server’s keys stored in the KDC.

The second class is the Kerberos class, which is constructed out of two subclasses. Like in the abstraction of the Kerberos protocol, there exists a black-boxed third server (Kerberos server) which in reality is made up of the authenticating and TGT server. These are the two servers represented by objects in the kerberos.py file. These two objects are instantiated with a database object called the KDC (located in the KDC.py file). This provides access to the user and HTTP passwords to both the authenticating and the TGT server. After determining the server for which the request is for, the Kerberos class pushes the user’s request to the appropriate server. Both server objects check if the user is valid by comparing the name and the times between the authenticators and tickets the user sends. These measures allow for proper identification and authentication of the user. Each server object also has the power to send back tickets and other messages to the user.

The last class is the HTTP class. This is the simplest of the objects since the HTTP server only needs to authenticate the client and then authenticate itself to the client. Every HTTP_Server object is instantiated with a secret key, which is stored in the KDC object.

To protect the servers, all secret passwords are stored in the KDC. For ease of use and communication, the nonces used in the encryption are also stored in the KDC. This allows the servers to encrypt tickets and sensitive information without the need to send the keys over the network. The encryption methods are made available in the encryption.py file which uses the PyCa Cryptography library. In particular, this file uses CTR mode to prevent padding oracle attacks.

Link to github: https://github.com/jimenezh/Kerberos-Protocol-Demo

Works Cited

"About Firewalls." *About Firewalls*, Indiana University, kb.iu.edu/d/aoru.

Kerberos: The Network Authentication Protocol. MIT, https://web.mit.edu/Kerberos/. 22 April, 2000.

"What Is a Man-in-the-Middle Attack?" *What Is a Man-In-The-Middle Attack?*, Cloudflare, www.cloudflare.com/learning/security/threats/man-in-the-middle-attack/.

"What Is a Replay Attack?" *Kaspersky*, Kaspersky, www.kaspersky.com/resource-center/definitions/replay-attack.