Cybersecurity MSc: Identification, Authentication and Authorization 2020-21

Project (1st part): Zero-Knowledge password-based authentications with a cache of temporary, asymmetric credentials

April 21, 2021 Due date: May 30, 2021

Changelog

- v1.0 - Initial version.

1 Introduction

Password-based authentications are very convenient, but they suffer from many security problems. One of those problems is that most protocols reveal too much information to one of the participants. Thus, if one of the participators is a malicious one (Mallory hereafter), it can capture enough information to recover the password (e.g. using a dictionary attack). Note that the protocol can reveal too much information to the authenticator (and, in this case, Mallory will try to impersonate an authenticator), or the one being authenticated (and, in this case, Mallory will impersonate a legit client of the authenticator).

This is the case, for instance, with the UA Identity Provider (IdP) service. This service receives a password directly, which means that if it happens that Mallory is able to lure a client in order to make it contact a wrong Web server (i.e., a phony IdP), the client will give their password to Mallory.

Zero-knowledge Proofs (ZKP) are ideal to deal with this problem, since they allow both participants (the authenticator and the one being authenticated) to prove each other they share a secret (the same password) without revealing that secret to each other. We can consider ZKP authentication protocols to be the extreme case of challenge-response protocols.

Interactive ZKP is a class of ZKP where the participants exchange messages until they got convinced that the other knows the shared secret (the password). The protocol can go on forever until the authenticator got satisfied. This is what we will do in this project. However, we will use a twist: since Mallory can benefit from a ZKP protocol running for too long, since it receives more information that could be used to guess the password, we will force Mallory to also use ZKP to show its password knowledge. Since it cannot do so, it will reveal itself, though without knowing exactly when and how.

2 Homework

The work consists on using a personal helper application to perform an authentication dialog with an IdP (see Figure 1). For all practical purposes, the IdP will be a vanilla Web server, and the authentication is triggered by some Web service requiring the authentication of a browser user. To protect the authentication protocol from eavesdroppers, even considering a ZKP technique (which could nevertheless be vulnerable to a dictionary attack), we assume that a secure HTTPS session exists between the client browser and the IdP (though it does not need to be implemented).

The helper application should run locally to the client's browser and the IdP should get into contact

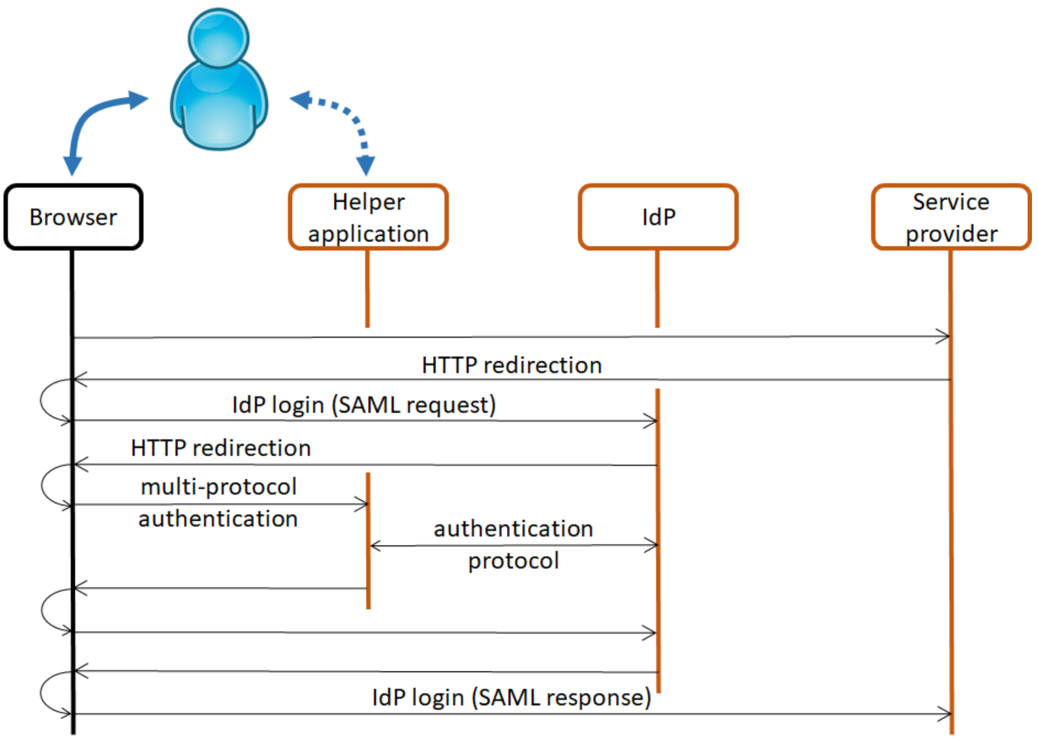

Figure 1: Architecture of the system to implement. The Helper application should run locally to the browser

with it using HTTP redirections (see [1]) or JavaScript calls (as with the Cartão de Cidadão plugin). The exact IP address of the helper application can be defined by a DNS name resolved differently by each client machine (e.g. `browser_helper_application` → 127.1.2.3), and the port can be 80 (the standard HTTP port). Note that in some browsers or host setups you may have to whitelist accesses to ports in the local host (i.e. ports associated with addresses 127.X.X.X.

The ZKP authentication protocol should run a minimum number of iterations, say N, on both directions. So, at the end, the protocol will have at least $2N$ messages. On each message there should be a random challenge and a result bit r. The value of r should be computed from the all the challenges sent and received so far (C) and from the shared password P. If

$$R_i = f\left(C_i, P\right)$$

where R_i is the response computed from all the challenges exchanged until iteration i, r can be the parity of R or one of its bits on a predefined position (e.g. $i \bmod \text{length}(R)$). If, at some point, the received result bit r is wrong, the receiver will know that the other participant is not legit, but will nevertheless continue the protocol, though sending purely random results. This way, the malicious participant will not receive genuine responses after a wrong response, which prevents it from getting enough data to run a successful guessing attack against the unknown password. Furthermore, a malicious participant will not have a clue about which of its responses was the first one that was wrong.

In the definition of the ZKP protocol, assume that each part can contribute to define N. In principle, the higher the N, the more secure is the protocol, but also more resources it may consume (namely to the authenticator). Find a solution that is able to balance the quality of the authentication with DoS prevention.

2.1 Speeding up authentications with temporary asymmetric credentials

Upon a successful outcome of the ZKP protocol, which can be slow, the entity being authenticated should generated a key pair for a future authentication. The private key should be stored, protected with a random key (recoverable by the helper application), and the public key should be uploaded to the authenticator (the IdP). The upload should be authenticated with a common key derived from the current authentication protocol (e.g. a key formed by bits extracted from R values, but other than the responses r). Upon the upload, conduct a challenge-response protocol that assures the overall correctness of the authentication of both participants using the novel credentials.

Note that the public key of the new key pair can be used by the client, hereafter, to also authenticate the authenticator. Therefore, the upload of the public key should be confidential. Also, you should not reuse the same key pair for different authenticators to avoid tracking.

Since the same person can use the same password with the ZKP to get authenticated in different terminals, each of those terminals will generate an asymmetric key pair, which should be distinguished. Therefore, for each public key uploaded to the authenticator, the helper application must use a random identifier to identify a particular public key belonging to a person. The authenticator must define, for each uploaded public key, the lifetime of that public key (a simple method to implement garbage collection). This lifetime must also be communicated to the owner of the corresponding private key, so that it could know until when it can use it.

These temporary asymmetric credentials should be used instead of ZKP authentications whenever possible, in order to speed up authentications.

2.2 Secure tunneling

The communication between the helper application and the IdP can be performed using consecutive HTTP redirections. This strategy can benefit from an HTTPS connection between the browser and the IdP, relying on the certificate verification performed by the browser during the setup of the HTTPS session with the IdP.

Alternatively, the helper application can directly address the IdP, as illustrated in Figure 1, acting as a direct HTTP client. In this case, the helper application must encrypt and authenticate the communication, which can be implemented using key material provided by the IdP to the helper application through the HTTPS session (e.g. a shared, secret key uploaded by the IdP on its initial call of the helper application).

In both cases, the secure tunneling of the authentication protocol is assured, which is beneficial for preventing the eavesdropping of the authentication protocol messages, which could be used to implement off-line password guessing attacks.

2.3 The Helper application as a browser-independent login manager

The helper application should act as a browser's password manager for a user. However, it should be independent of any browser, i.e., it must be an independent application, and not a browser plug-in.

In this sense, it makes sense to consider that the helper application implements a login interface for collecting the user credentials to be used on a given authenticator (the IdP, in this case). Therefore, the helper application should present the identity of the authenticator (e.g. its URL) and should keep, for that authenticator, one or more user profiles, each one bound to an internal name (aka username). Note that HTTP redirections natively provide the identity of the redirection requester in the `referer` HTTP header field.

The helper application can implement an autonomous GUI or can be headless, using the client browser for providing a GUI.

2.4 The overall system

You should implement a system with all the four components presented in Figure 1, which should work with any browser. The service provider should provide a very simple service given the identity of the client (e.g. a photo repository). The IdP should manage a database with personal profiles, which must contain a password and may also contain some temporary authentication tokens, such as the public keys above referred. The insertion or modification of passwords in the database can be manual and off-line.

The dialog between the service provider and the IdP should be encapsulated in SAML messages, following the SAML Web Browser Single Sign-On Profile[1]. For this project those SAML messages do not need to be authenticated.

You can use any language and framework to implement your system.

3 Delivery

Send your code to the course teachers through Elearning (a submission link will be provided). Include a report, with no more than 30 pages, describing the authentication protocols defined and implemented, the overall sequence of operations considered, the interfaces used and their parameters, some relevant implementation details (not complete copies of the code!) and the results achieved, including evaluations of delays imposed by ZKP protocols.

4 Evaluation

This 1st part of the project will be evaluated as follows:

- Implementation of the architecture with IdP and a service provider using SAML: 20%;
- ZKP authentication protocol: 20%;
- Authentication with the asymmetric key pairs set up to replace ZKP authentications: 20%;
- Helper application interface: 20%;
- Written report, with a complete explanations of the strategies followed and the results achieved: 20%

References

[1] André Zúquete, Hélder Gomes, and Cláudio Teixeira. Personal Identification in the Web Using Electronic Identity Cards and a Personal Identity Provider. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things*, pages 160–169, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[1]Section 4.1 of `https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf`.