Project 2 - Authentication

Description

This assignment will focus on the implementation of robust authentication protocols. The objective is for students to develop a new protocol and a related Web application to explore it.

Authentication of people in Web applications

Passwords are still a commonplace in today's personal authentications. Regarding Web applications, authentications can be performed in two different protocol layers: on the HTTP layer and on the applicational layer. The former is usually explored in applications that use HTTP as a transport protocol, as it requires a specific manipulation at the level of the headers of HTTP requests and responses (the Authorization header). The latter explores the exchange of data within the payload of HTTP requests and responses. As such, it can be used by specific Wep applications or by generic Web clients, such as browsers. In this last case, the authentication is carried on with HTML forms, frequently mediated by Javascript, where users directly provide their credentials (e.g. username and password). This is the one we are interested in.

The authentication of people through HTML/JS (or even HTTP Basic Authentication), exposes the credentials directly to the authenticator and to network eavesdroppers. Because of this, they usually are performed within TLS (Transport Layer Secure) secure sessions. These sessions, that are automatically instantiated whenever a browser uses the schema `https'' instead of` http", authenticate the server side (authenticator) with an X.509 certificate, and encrypt the entire interaction to avoid eavesdropping attacks.

Nevertheless, people can be fooled by rogue servers (e.g. using DNS names that can hardly be distinguished from the intended one by a normal user, a problem known as typosquatting), and certificates may also be forged in extreme conditions (e.g. compromise of certification authorities). In such cases, personal authentications performed by means of forms provide user credentials directly to the attacker, without any possible defense.

Challenge-response authentication protocols

Challenge-response authentication protocols can tackle this problem. However, browsers do not implement them when data is provided in a Web page, and additional Javascript is required. Thus, in this project one of the goals is to implement and deploy such a protocol to authenticate people in Web applications, providing and higher level of security.

Still, challenge-response protocols can leak relevant information to rogue servers. In fact, such protocols normally produce a response given a secret and a challenge, and the challenge and response can be used to perform password discovery attacks (e.g. using dictionaries). Therefore, such responses should be only provided to trustworthy entities.

To that end, password-based authentication protocols can benefit from the fact that both parties (the one that gets authenticated and the one that authenticates) share some secret information: a password (or a transformation of it). This way, both parties can use that information to conduct an independent, though related authentication protocol. Or, in short, they can use personal credentials to perform a so-called mutual authentication. An example of such a protocol is MS-CHAP-V2.

Nevertheless, even when mutual authentication is used, one of the participants in the protocol has some advantage: the one that receives first a response computed from a challenge and from the shared password. Since both must do it, although with different parameters, one of them gets such response first, inevitably. And, in that case, if that participant is a malicious entity, it may recover information that may then be used to find the shared secret used to compute the response.

Typically, on a client-service interaction, the server is the participant that gets the first response (see MS-CHAP-V2). This approach avoids the exposure of user accounts to countless fake authentication operations with the sole interest of getting responses computed with their passwords. But it exposes such responses to rogue servers that managed to fool a user to use them, instead of the legit ones.

Enhanced challenge-response authentication protocol (E-CHAP)

This problem can be mitigated by running the authentication protocol N times, instead of a single one, while providing in each protocol run the minimum set of information (1 bit) in each response. With such short responses, both participants only get convinced about the benignity of the other after `N` correct answers, being $\frac{1}{2}^N$ the probability of an imposter to succeed in a personification attempt. Given this expression, the higher the N, the lower that probability is.

Furthermore, during a sequence of N protocol runs, each participant should not abort it after receiving a wrong response. Instead, it should continue, but providing random responses thereafter. This way, if the other one is an impostor, it will no longer benefit from the information received after the error detection, and will not also realize in

which step it stopped receiving genuine responses.

For this project, select a fixed, and reasonable, value for N.

User authentication application (UAP)

To implement E-CHAP you should rely on an application under he control of the user to implement it. This application, which will call the user authentication application will be similar to the plugin of the Cartão de Cidadão: it exposes a REST API, receives HTTP requests from a (local) browser to perform an authentication with a given server (DNS name) and runs a given authentication protocol with that server. Upon a successful completion, the control must return to the browser and service that required the authentication.

The UAP should implement the following functionalities:

- It may respond to HTTP redirections from the service or to JavaScript calls from a resource provided by the service. Furthermore, the UAP can interact directly with the service (via HTTP requests) or use HTTP redirections via the browser.
- It can have its own user interface or use the user browser for that purpose.
- It must allow the user to store several username-password pairs for each service DNS name).
- It must inform the user about the authentication protocol that is going to be run on its behalf.
- Regarding E-CHAP executions, it must inform the user upon an error in a response from the service, including the phase of the protocol where it occurred. Note, however, that E-CHAP should not be aborted because of that error.
- Since it is not part of the browser, it must be launched autonomously. Upon launching, it should ask for a password that is going to be used to encrypt the user's database of credentials (username-password pairs).
- All data stored should be encrypted. For instance, you can use a SQLite database with encrypted cell values, or a simple file where you store an encrypted JSON dictionary. For privacy reasons, service names should also be encrypted and equal usernames or passwords cannot be detected. Finally, all encrypted values must include some sort of random salt, to prevent leaking information from their size.

The UAP application can be implemented in any language, as a Web interface is all that is required.

Students should provide a version of their Web application developed in the previous project but now exploring e-CHAP. For a start, they can implement an existing challenge-response protocol with mutual authentication, such as MS-CHAP-V2, or they can devise a new one.

Ideally, the protocol should hide personal identities, but usually that is delegated to a secure transport with HTTPS. You can consider, for this purpose, that the interaction between the browser and the service uses HTTPS, so it can be used to exchange privacy-sensitive protocol messages.

A bonus of 10% can be provided if the project include some interesting feature related with the authentication handling or with the protection of users' credentials.

The project is expected to be implemented by **a group of 4 students**, and **MUST** reside in a private repository in the github/detiuaveiro organization, using the Github Classroom functionality (this is mandatory).

Project delivery

Delivery should consist of a git repository with at least three folders and a file:

- `app_auth`: contains the Web application with the extended authentication protocol, including instructions to run it. This represents the web page, running in a remote server.
- `uap`: contains the UAP, including instructions to run it. This component would run in the client computer.
- `README.md`: contains the project description and the authors;

Projects will be graded according to the implementation, the implementation of the code that implements the authentication and the protection of the related information, and the documentation produced.

This project is expected to be authored by the students enrolled in the course. The use of existing code snippets, applications, or any other external functional element without proper acknowledgement is strictly forbidden. Themes and python/php/javascript libraries can be used, as long as the vulnerabilities are created by the students. If any content lacking proper acknowledgment is found in other sources, the current rules regarding plagiarism will be followed.

References

- André Zúquete. Segurança em Redes Informáticas (6ª Edição Atualizada e Aumentada). FCA - Editora de Informática, Lda., Portugal, October 2021
- G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759, January 2000
- MySQL

2021