Flow - Register a goal conflict

Introduction

Any time the user visits a page that conflict the goals they have set, that conflict needs to be registered and communicated to the buddy. This page describes the different solution alternatives and concludes which one will be taken for implementation.

Design goals

The following design goals are defined:

- Private information about a user (buddies, goals, devices, messages from buddies, messages about goal conflicts, etc.) should be inaccessible to everyone, including those that have administrative access to the systems
- It should be impossible to build a quantitative understanding of how well a user acts against his own goals, even when having administrative access to the systems

Solution alternatives

Encrypted messaging

This approach relies on Public-key cryptography. Each user gets a public and private key assigned. Both keys are stored on the Yona system, the public key as plain text and the private key encoded based on a derivative of the password that is only available to the user. This derivative is not stored on the Yona system and cannot be calculated based on anything stored on it.

The system needs to store all URLs that conflict the user's objectives, as they have an option to disclose a URL in case they are convinced there is nothing wrong with it. The URLs need to be readable only for the buddy and the person who visited the site. This implies they need to be stored twice, once encrypted with the public key of the buddy and once with the public key of the visiting user.

Given that an attacker should not be able to build a quantitative understanding of the behavior of a user, we cannot simply relate the URLs to the buddy and the user. To mitigate that, we introduce the concept of a message group. The decision service maintains an in-memory mapping of account to message group ID. In case a message is to be sent to an account (whether a buddy or the visiting user), the system will see whether a message group is known. If not, a new ID will be generated and stored for the target user, encrypted by their public key. That way, an attacker will not be able to build an understanding of the number of messages targeted to a specific account. There will be a set of encrypted message group IDs, but the number of items in that set does not correspond to the number of messages but to the number of times the system is restarted. Given that the group ID is encrypted, an attacker cannot establish an association between the messages and the target account.

The below animated slide illustrates the encryption approach (Confluence wiki doesn't handle the animations properly, so you'll need to download it):

Encryption-based...vice design.pptx

Pros

- Most of the data privacy requirements are met

Cons

- Addition of a group ID implies that the person or their buddy missed a goal (it would be possible to mask that through dummy group IDs, but that implies additional complexity).
- If an attacker is able to attach a Java debugger to the Decision Server, they could link the group ID to the target, thus revealing a part of the messages
- Data in the queue between the classification server and the decision server is unencrypted and can be linked to people (this is only live data that passes by while the system is used).
- An attacker can see who has signed up, their goals and their buddies

Anonymous logging

In this approach, the user account that's linked to the VPN, cannot be linked to account details without access to the private key of the user. The heart of the solution is that the log messages are virtually anonymous: they mention an accessor ID, but that ID cannot be linked to anyone. When signing up a user, we'll create an account registration that includes information like the first and last name, the e-mail address, the goals, etc. This information is encrypted and stored, so only this particular user can access it. Besides this, we create a user ID for the VPN connection. This user ID is a synthetic ID (could be a UUID like de305d54-75b4-431b-adb2-eb6b9e546014) and it's named the accessor ID. This accessor ID is stored as part of the encrypted account information. We store the goals of the user, in plain text, related to the accessor ID.

Anytime the user access a website via Yona, the Smootwall classification engine will issue a record with information about the site that was visited, and the user visiting it. This user ID is taken from the VPN, so it resembles our accessor ID. The decision server verifies of the visited site conflicts the objectives of the user (those stored with their accessor ID). If so, it logs a visit record for this accessor ID.

When a user logs in to the system (through the app) to see their own visited URLs or those of their buddy, they'll decrypt their account information to find the relevant accessor ID and fetch the records belonging to it.

The following animated slides illustrate the approach (Confluence wiki doesn't handle the animations properly, so you'll need to download it):

Anonymous-logg...ce design.pptx

Pros

- All data can be encrypted or anonymized, even to the level that the administrators have no insight in what people are using the system.
- The data in the queue between the classification server and the decision server is anonymous
- No encrypt in the decision making process, only in the data retrieval.

Cons

- The anonymous data is hard to debug
- If one of the stored URLs contains identifying information (e.g. http://poker.com/?user=john@doe.com), then the anonymity of that user is broken.

Anonymous encrypted messaging

This approach is a merger of the previous two. Logging happens based on a anonymous user account, but the messages are now encrypted, so information in the message cannot be used to determine the identity related to the accessor ID. The plain text data related to the accessor ID already contains the goals of the user. In this approach, we add a list of target IDs and related public keys. Based on that, the decision server creates message a message for each target, encrypted with an public key that's specific for that target. The encrypted data of each user contains the private key of the key pair used by decision server.

When a user logs in to the system (through the app) to see their own visited URLs or those of their buddy, they'll decrypt their account information to find the relevant target ID and private key. They fetch the messages related to the target ID and decrypt them with the private key.

The following animated slides illustrate the approach (Confluence wiki doesn't handle the animations properly, so you'll need to download it):

Anonymous-encr...ce design.pptx

Note: As a simplification, the presentation uses the public key of the user account to encrypt the log messages. This makes it possible to associate the target ID to an account, which is unnecessarily revealing. Instead, we should have a separate key pair to encrypt the log messages. This key pair can be stored as part of the encrypted data of the user account.

Pros

- All data can be encrypted or anonymized, even to the level that the administrators have no insight in what people are using the system.
- The data in the queue between the classification server and the decision server is anonymous

Cons

- The anonymous data is hard to debug
- The decision making process contains encryption, which impacts its performance
- The data in the queue between the classification server and the decision server is not encrypted, so if an attacker can spy on the messages passing by, they might find a message that break the anonymity of the user. To reduce that risk, we could consider using a nonpersistent connection between the classification server and the decision server.

Conclusion

Anonymous encrypted messaging addresses all design goals, so this is the alternative of choice.