InnoMetrics

Client-Side encryption

Implemented by **Muhammad Mavlyutov**
m.mavlyutov@innopolis.ru

Alexey Zhuchkov
a.zhuchkov@innopolis.ru

The encryption processes are involved in both sender applications (data collectors) and viewer applications (data viewers). The system is designed in such a way, that data can not be accessed without the user's password, which is never sent to the server. Instead, the server stores a bcrypt hash of a PBKDF2 hashed (10k iterations) user password. The server also stores user's public_key and private_key, the last one is encrypted according to PKCS#8 standard, using hashed user's password as a passphrase. Each activity packet contains a random enc_key, that gives access to the sensitive data inside (AES cipher). This enc_key is also encrypted using RSA OAEP, by the generated user's public_key. Only encrypted enc_key is stored on the server.

Collector

1. Prompt email and password.
2. Hash the password using `PBKDF2(salt = email, iterations = 10000, key_length = 64)`. The result of this hashing is to be converted to **`hex`** (by some `bytes->hex string` function, must be **lowercase**). Async hashing is recommended.
3. Send the email and hashed password to the server.
4. The server will return a **`public_key`** cookie. This is a base64 encoded PEM-packed public key.

5. Using features of an RSA provider library, import this public key and create **`RSA_OAEP`** cipher.
6. Collect data and put it in a packet.
7. Once ready to send the packet, generate **`enc_key`** (a random bytes string of length a multiple of 16 bytes. Recommended length - **32** bytes).
8. Encrypt `enc_key` using the created RSA cipher. Include this encrypted `enc_key` in a packet body (in **`hex`** encoding, must be **lowercase**) as field **`enc_key_h`**. DO **NOT** include plain text `enc_key`.
9. Generate **`Initial Vector`** (a random bytes string of length 16 bytes). Include it in a packet body (in **`hex`** encoding, must be **lowercase**) as field **`iv`**. DO **NOT** encrypt it.
10. Using the generated `Initial Vector` and generated `enc_key` (unencrypted), create **`AES_CBC`** cipher.
11. Encrypt the fields that contain sensitive data using the created AES cipher. Fields `'executable_name', 'browser_url', 'browser_title', 'ip_address', 'mac_address', 'activity_type', 'project'` are expected to be encrypted.
12. Send the packet to the server.

Viewer

1. Prompt email and password.
2. Hash the password using **`PBKDF2`**`(salt = email, iterations = 10000, key_length = 64)`. The result of this hashing is to be converted to **`hex`** (by some `bytes->hex string` function). Async hashing is recommended.
3. Send the email and hashed password to the server. If login successful, save the hashed password in local storage (e.g. as **`password_h`**)
4. The server will return a **`private_key_h`** in JSON. This is an encrypted PEM-packed private key.
5. Using features of an RSA provider library, import this private key and create **`RSA_OAEP`** cipher, using **`password_h`** (in **`hex`** encoding, must be **lowercase**) as the passphrase (i.e. `private_key` is exported in PEM format and encrypted using `password_h`, which is encoded as **`hex`**. You just need to reverse these steps.)
6. Request an encrypted packet from the server.

7. Convert **`enc_key_h`** from the received packet from **`hex`** string to **`bytes`** string. Decrypt **`enc_key_h`** using the created RSA cipher and keep it as **`enc_key`** for processing this packet.
8. Convert the **`iv`** from the received packet from **`hex`** string to **`bytes`** string. Using the converted **`iv`** and decrypted **`enc_key`**, create **`AES_CBC`** cipher.
9. Decrypt the packet fields that contain sensitive information using the created AES cipher.
10. Show the packet to the user.

Password changing

There is an end-point in backend for changing the password (not implemented in the frontend yet). The documentation for the end-point is available in **`documentation.yaml`**

Reference

https://habr.com/ru/company/yandex/blog/344382/