Version 1.0

Cornelius Kölbel (corny@cornelinux.de)

May 2014

Everybody knows that a password - be it simple or even complex - is a potential vulnerability. Two factor authentication is the way to authenticate a user not only by verifying his password but additionally asking for the possession of a second factor - a hardware device. But nobody knows who can be trusted. This talk invites you to trust in two factor authentication and to trust in open source. This talk will give a short overview about possible ways to do two factor authentication with open source and finally describe the open source two factor system LinOTP, that can manage different kind of tokens like Yubikey, eTokenNG OTP and smartphone apps. Thus strengthen the security of VPNs, remote logins and hard disk encryption.

Two factor authentication – Open, trustworthy and enterprise ready

Two factor authentication

Two factor authentication is a combination of components of what only you know (like a password), what you have and what you are.

Authentication process

During an authentication process a person authenticates to a system as a user, to use the system or the service of the system. So in most cases the systems wants to identify the user – not necessarily the person.

What-you-know (password, passphrase, security questions) is specific to the user on the system.

What-you-have (OTP token, ssh key, client certificate, smart card) is also specific to the user on the system.

What-you-are (fingerprint, voice, retina, gait recognition) is specific to you as a human being and has nothing to do with the system.

What you are is easy to bring along, but is not limited to the system the user is authenticating to. The face or the gait can not only be used to authenticate to this very system but also to identify the person in the crowd for his hole lifetime. In contrast a password or a certificate is bound the system and can be changed or revoked in case it was compromised. Try to change you face, when it was compromised.

Thus I prefer to only combine *what-you-know* and *what-you-have* to do two factor authentication.

The aspect of compromise and uniqueness

Passwords

Passwords and passphrases (*what-you-know*) are easy to copy and the user will not recognize, that it was copied or stolen. If it is stolen, it is still there!

This is what makes the sole use of passwords so weak and why it should be combined with the second factor *what-you-have*.

Private keys

SSH-keys and client certificates are a good way to start and secure remote access, SSH logins, VPNs and web services. But also those keys can be copied without being noticed.

Smartcards

Smartcards store or even generate the private key on a crytographic device that does not allow to extract the private key, making the private key non-copyable and ensuring its uniqueness. If the private key is stolen, it needs to be stolen with the card and the user knows that his key is compromised. He is now able to revoke it.

To use smartcards many different drivers (for the reader, for the card) need be installed and the application also needs to support this.

One time password tokens

One time passwords are calculated using a symmetric algorithm with a secret key within the authentication device and in the server system. Applications do not need to be modified, since the way of authentication (entering a password) still looks the same to the application. Only the way the password is verified has changed.

There are many different one time password tokens. Today many smartphone apps offer the functionality of an OTP device. The best known might be the Google Authenticator[1]. As a modern computer system the smartphone is vulnerable to attacks on the symmetric secret key stored in the smartphone app. Thus such an authentication device can be compromised without easily being noticed.

The secret key in classical key fob authenticators like RSA SecureID or SafeNet eToken PASS can probably only be copied by destroying the device. Which will show the user, that his key was compromised. But those devices come with another problem. The secret key must also be known to the authenticating service. So the secret key is not only on the key fob but also on a CD or in file deliver by the vendor and shipped by the distributor and the reseller. This is a path of possible compromisation, that is not easily noticed.

Reprogrammable hardware devices guarantee best uniqueness of your secret key and are not easy to compromise. The SafeNet eToken NG, the eToken PASS using an additional programming device and the Yubico Yubikey are such devices. It must be noted that while during the programming additional driver software is needed all devices can be used without driver or software during authentication.

OTP Algorithms

Since an OTP value needs to be entered into a password entry field, an asymmetric algorithm can not be used. An asymmetric algorithm would create such a long passphrase that could not be entered manually into a password field.

This is why the usual OTP algorithms all use a symmetric secret key in the algorithm and a truncation function.

HOTP[2], TOTP[3] and OCRA[4] are open algorithms defined by the Open Authentication Initiative[5]. They are all based on the SHA-algorithm. The secret key is concatenated with a counter, the time or additional input data.

The motp[6] algorithm is an open MD5 based algorithm, that concatenates the secret key with a PIN and the time.

The Yubico Yubikeys use an AES based algorithm, where a counter and additional information are encrypted with the secret key and can only be decrypted by the server.

In addition there are proprietary algorithms by RSA, Vasco and kobil.

To sum up: All algorithms

- use a secret symmetric key,
- use a counter (either an event or the time)
- use an encryption or a hash function to generate the OTP value.

Still, the OTP algorithms have some limitation. While the OCRA algorithm can be used for signing data, none of them can be used to encrypt data, which make it hard to use OTP authentication for securing data encryption.

Implementations

There are many software implementations to authenticate users on the server side. `libpam-google-authenticator` and `libpam-yubico` may be well known.

Both of them only allow a standalone authentication. It needs to be installed on a single machine and the authentication device is also managed on this very machine. The same device can not easily be used to authenticate to other services. This is not manageable in enterprise environments.

Motp-AS[7], potato[8], the yubicloud[9] and the Yubico Validation Server[10] are examples, where the authentication devices are managed on a central system and many other applications or servers can authenticate against this central system. But all these systems only support a single device type.

LinOTP

LinOTP[11] is a modular authentication system, that supports many different authentication devices and different authentication protocols like RADIUS, http and PAM. Unfortunately only its core components are licensed under the AGPLv3 while necessary components for grabbing users from LDAP directories and storing audit trails to an SQL database are under a proprietary license by its vendor LSE[12] GmbH.

Nevertheless LinOTP is at the moment the most open and flexible two factor solution for the enterprise environment. Using RADIUS it can be used to secure existing VPN connections, SSH logins can be secured via RADIUS or PAM. There are also first steps to secure LUKS encrypted hard disks in a manageable way[13].

1 https://code.google.com/p/google-authenticator
2 https://tools.ietf.org/html/rfc4226
3 http://tools.ietf.org/html/rfc6238
4 https://tools.ietf.org/html/rfc6287
5 http://openauthentication.org/
6 http://motp.sourceforge.net
7 https://github.com/MOTP-AS/MOTP-AS
8 http://kelvin.nu/software/potato/
9 http://www.yubico.com/products/services-software/yubicloud/
10 http://www.yubico.com/products/services-software/validation-server-components/
11 http://linotp.org
12 https://lsexperts.de/enterprise-edition.html
13 https://github.com/cornelinux/yubikey-luks