Eso Overview

Josh Campbell

A tentative name Eso (from *esoteric* - intended for or likely to be understood by only a small number of people with a specialized interest) will be used to refer to the project to simplify this document as well as examples in this document.

When used in this document, *credentials* mean private/public keys, symmetric keys, certificate, and usernames/passwords.

Contents

1. Abstract
2. Application Objectives
3. Context / Operating Environment
4. Threat Model
5. Proposed Architecture

Abstract

As companies grow in size to thousands of machine spanning multiple continents, the task of credential distribution and management becomes cumbersome. It is no longer feasible to manually ensure that credentials are correctly being used when these machines have both overlapping and differing needs and permissions. Another problem that arises is ensuring that credentials do not exist in plaintext on any machine in the company so that a misplaced hard drive will not leak confidential credentials. Ideally credentials should only exist in the memory of the executing program, and even then for a brief amount of time. Furthermore, as time progresses, the very existence of credentials is a vulnerability; how does one enforce credential rotation when the very distribution of the credentials raises the uncertainty of whether all hosts will receive the updated credentials? In this document I consider the problem of allocating permissions for, securely managing and distributing, and preventing leakage of credentials in a distributed environment.

Application Objectives

The application objectives are:

1. Provide a centralized interface for managing credential permissions, creating/deleting new credentials, and deploying credentials to remote machines.

Permissions should consist of the set name, operation, and entity.

Set Name	Operation	Entity	Entity Type
com.joshuac.project1	encrypt, decrypt	joshuac	user
Com.joshuac.anotherproject	encrypt	uf	POSIX Group

By managing permissions, a user would be able specify a specific user or group to encrypt, decrypt, sign, verify, or retrieve a credential. Note that retrieving a credential allows the credential to exist in the calling program's memory, and should not be used casually.

Credentials should consist of the set name, version, type, and expiration.

Set Name	Version	Type	Expiration
com.joshuac.project.key	1	Public Key	2020-10-11
com.joshuac.project.creds	1	Credential Pair	2012-05-05
com.joshuac.project.creds	2	Credential Pair	2014-05-05

By creating a new credential through this interface, the user will not actually know the contents of the credential created. For example, the user could choose to create a new password, upon which the server would generate a ‘strong’ password and associate it with the user. Furthermore, the interface would only offer the user choices to create cryptographically secure credentials. The user would also be able to import exiting credentials, though this option is not recommended.

By deleting a credential, the user would be sure that the credential would no longer exist on any of the machines upon which is was previously deployed. This function would obviate the need to individually track down and remove credentials from machines, especially if the credentials were distributed globally.

By deploying a credential, the user securely pushes the credentials to the machine or machine groups specified. This differs from creating a credential in that when a credential is created, it is not actually pushed onto any machines until the user is certain that the permission specifications are correct.

2. Enable credentials to exist encrypted on machines

   Once credentials are deployed to a machine, they should exist encrypted in such a way that if the drive is removed from the machine, the credentials stored on the machine should remain safe.

3. Provide an API to allow developer services to access credentials existing on their machines. A couple examples follow:

```
// An example of using Eso to sign data
EsoLocal eso = EsoLocalClient.getService();
MaterialDescription material = new MaterialDescription();
Material.setMaterialName(keyname);
SignedResponse response = eso.sign(material, data.getBytes());

// An example of using Eso to bring a asymmetric key
// into memory
KeyPair keyPair =
   new EsoKeyPair(name,version)).getKeyPair();
```

```
// An example of using Eso to bring a username/password
// into memory
Credentials credentials =
        new EsoCredentials(credentialName).getCredentials();
```

Problems Eso Does Not Solve

1. The trust problem

   A malicious developer who knowingly attempts to decrypt or break Eso's credential database on his or her machine.

2. The permission problem

   Securing the machine from programs that might somehow arrive on the machine

Context / Operating Environment

Eso makes several assumptions about the operation environment:

1. A developer will primarily be using the machine – hence the need for an API for using the credentials
2. Machines exist in on a network where they have a static, unique IP address
3. Machines are running a UNIX variant
4. Machines have constant Internet access
5. A developer's username to access the machine uniquely identifies that developer in the company
6. Eso is assumed to be part of the standard distribution within a company, or easily available for download and installation

Threat Model

Eso may face various vectors of attack:

1. The central authority that distributes the credentials might be directly penetrated
2. As credentials are in flight to the end host, they might be intercepted
3. A malicious user might attempt to masquerade as the central authority and distribute fake credentials
4. The hard drive might be forcefully removed from the machine

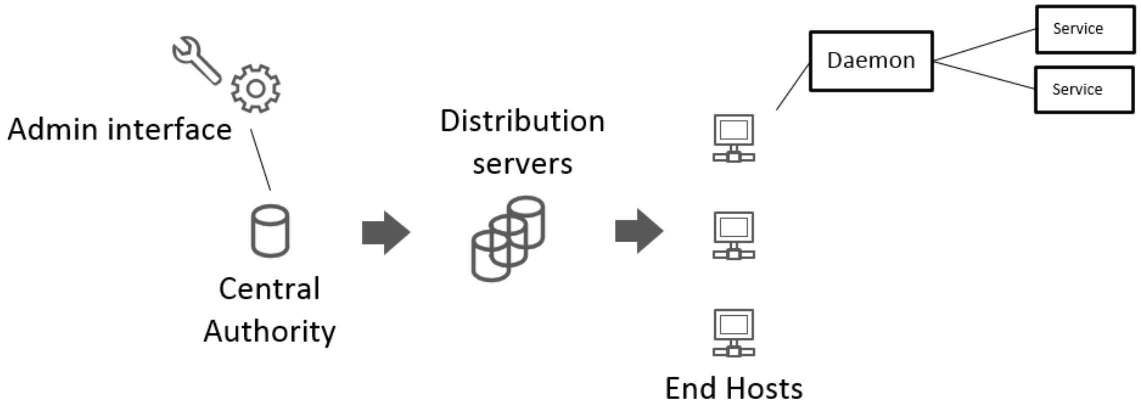
5. The integrity of the credentials on the user's machine might be violated
6. Users might attempt to repeatedly access credentials they shouldn't
7. A malicious user might masquerade as a process and attempt to pull credentials from the central authority
8. Caching and paging might cause the credentials to exist vulnerable for a brief amount of time

Eso also faces various failure threats:

1. The central authority might crash and be rendered useless – along with all the credentials and permission it is keeping track of
2. The end host might never receive an updated credential

Proposed Architecture

The proposed architecture aims to solve the problems highlighted in the threat model.



Logging will be used at all levels in order to determine if a user continuously tries to access credentials he or she does not have permission to. Furthermore, all levels will use TCP or the company's internal equivalent.

- Central Authority

  The central authority provides an admin interface that owners of credentials can use to manage where keys go, what permissions the keys have, and what types the keys are. The central authority also maintains a database of all the credentials. It is assumed that central authority is located is a safe location. Both the distribution servers with have some form of authentication in order to identify who is requesting credentials from it. Both the central authority and distribution servers will use envelope encryption (using the recipient's public key) in order to safely transfer credentials to the end host.

- Distribution Servers

  The distributions servers are basically datacenter-level caches that provide central authority to end host support. In a global company, the distribution servers can be positioned strategically in order to decrease latency from the central authority to the end hosts. The distribution servers can be used to rebuild the central authority in case of the central authority failure. The actual credentials on distribution servers may also be a subset of the whole corpus of credentials for legal or security reasons.

- Daemon

  A daemon runs locally on each end host. The daemon's job is to manage cryptographic operations on the machine, and is the interface services talk to. The daemon provides cryptographic functionality to services, stores credentials and secures them in a database on the end host, and enforces permissions. When the daemon boots and after a period of time, the daemon polls a distribution server to see if it missed any updates.

  If the daemon does not have the credentials requested by the service, the daemon uses dynamic pulls to request credentials from a local distribution server. The credentials will be cached locally after the pull in an encrypted database maintained by the daemon.

  When the daemon stores or a retrieves a credential from the local database, it encrypts/decrypts them which a key generated from the hardware profile of the machine. If the disk is separated from the machine, it would be impossible to break the database without as NSA-class machine. To maintain the integrity of credentials in the database, the daemon can use a cryptographic hash.

  In order to keep the daemon from crashing and dumping core, as the daemon pulls a credential from the local database, it decrypts the key, uses it, and then zeros the memory. This decreases the chance that the daemon will dump core while the credential is in memory. Because of the uncertainty of garbage collection, languages like Java are not appropriate for the daemon. Instead, C or C++ will be used.

There will be no measure to prevent another process on the box from generating a key from the hardware profile and decrypting the daemon’s local database. As mentioned previously, Eso is not meant to solve the permission problem of accessing the machine – keeping people or programs off the machine another program’s problem.

(**TENTATIVE**) In order to determine the actual owner of the service requesting credentials from the daemon, the daemon will take the actual socket handle and find the owner in the kernel for that file handle. Users will not be able to assert identity to the daemon. The daemon will not bind to remote connections.

To verify that the credentials actually come from the central authority or distribution server, the admin will generate a cryptographic hash and sign the data it sends. The daemon will have a certificate to verify.

Caching and paging are still open problems.

The daemon can be kept continuously running by registering a cron job that periodically checks whether the daemon is running.