krb-agent: Safer Kerberos on Untrusted Machines

Miriam Gershenson, Gurtej Kanwar, David Lawrence, Jordan Moldow

Motivation

Kerberos is an authentication system which allows a single password to grant access to many different services. Unfortunately, this means that an attacker who eavesdrops on a user authenticating to a single service may then impersonate that user to any other Kerberized service. This is especially concerning in the context of an Athena cluster computers: an adversary could easily gain physical access to an Athena machine and compromise the local software installations. We aimed to minimize the damage caused by an attacker with such control while still allowing users to authenticate to services using Kerberos.

Architecture

We decided to split the Kerberos authentication system into two components: a “client”, which is untrusted but capable (e.g. an Athena workstation), and an “agent”, which is trusted but limited (e.g. an Android phone). When a user authenticates with Kerberos, they type their password into the agent; the agent holds their credentials (session key and ticket-granting ticket) in escrow. When the user wishes to use a Kerberized service on the client, the agent obtains a ticket for that service on the client’s behalf and forwards only the service ticket to the client. Thus an adversary who has compromised the client will only be able to impersonate a user to the services which they actually used on the client.

The Client

Kerberos stores a user’s credentials in a data structure called a *credential cache*. The credential cache is user-specific and normally includes a user’s session key, their ticket-granting ticket, and cached copies of any service tickets that they have requested. Tickets in the credential cache are stored there until they expire or are explicitly destroyed.

Kerberos is extensible in that it can support different credential cache types. All types must implement the credential cache API given by *struct krb5_cc_ops*, but are otherwise free to store credentials however they choose. When a credential cache is not specified by the client, the file type is used, and a user-specific file (the default is /tmp/krb5cc_[uid]) is used to store the user’s credentials on disk. Standard Kerberos builds also include the DIR type (a directory storing multiple file credential caches) and MEM type (credentials are stored in memory). The credential cache location may be set using the KRB5CCNAME environment variable.

We defined a new credential cache type, instantiated as REMOTE:[hostname]:[port]. It is a wrapper around the file type: each remote credential cache has a local backing file, and most remote credential cache API functions are wrappers around the corresponding functions for the backing file. However, functions that interact with a password, TGT, or session key are

delegated to the agent. The agent and client communicate over a text channel, which in our case is a network socket on the agent.

The extensibility of the Kerberos credential cache system allowed us to implement our protocol without making substantial changes to the base Kerberos libraries. Our client changes consisted only of writing a new C file for the remote credential cache implementation, registering the remote credential cache type with the Kerberos client, and modifying the kinit program to avoid requesting a ticket-granting ticket when it initializes a remote credential cache. As such, our modifications are easily mergeable into the master Kerberos repository.

The Agent

We implemented the initial agent as a Python daemon that stores the client's session key, ticket-granting ticket, and service tickets in a file credential cache. The agent listens for requests from the client and sends responses using a simple protocol over a network socket. The three types of requests are "kinit", to request that the agent authenticate a user, "ticket", to get a service ticket for the client, and "kdestroy", which destroys the agent's credential cache.

We secondarily developed an Android version of the agent, which communicates over a TCP socket. This agent implemented the same protocol, but was written in the framework of a downloadable Android application. The Android agent more closely models our expected use case: a portable trusted device which can act as an agent to untrusted workstation clients.

Source code

We forked the Kerberos repository and developed our protocol at <`https://github.com/jmoldow/krb5`>. Our Python agent and ticket serialization routines were developed in the top-level `/util` directory, and our credential cache type definition lives in `/src/lib/krb5/credential cache/cc_remote.c`. The Android agent started as a fork of the kerberos-android-ndk Github project and is in a Git submodule under the main repository.

Analysis

Our implementation successfully isolated the most sensitive components of the Kerberos system -- the ticket-granting ticket, the session key, and the user's password -- from the client side of the system. An attacker with control of the client would only gain access to short-lived tickets for individual services. (It is not possible to do better than this while remaining transparent to client applications.)

However, there are a couple of new concerns raised by our system. The agent sends service tickets unencrypted to the client, so although the user's session key and ticket-granting ticket are protected it may become easier for an adversary to eavesdrop on individual service tickets.send the tickets in the clear from the agent to the client, so other unapproved workstations or machines could steal the tickets by network monitoring. If an Android phone serves as the agent, the security implications of a user losing their cell phone become much more severe.

Future Expansion

Full Android Application

Though we only developed a proof-of-concept Android agent, this could be reasonably developed into a more polished user-facing application. In terms of real-world use, this is the most likely candidate for a reasonably secure but portable agent, and we could imagine this becoming the primary authentication method for all users who have Android-based phones available.

Graphical Workstation Login

At MIT, there are three major use cases for Kerberos authentication: on MIT affiliates' personal computers, on remote login machines such as `linerva.mit.edu`, and on Athena cluster workstations. Athena cluster workstations are the most complicated case, because Kerberos is only one component of a more complicated login system that includes a display manager and the Hesiod lookup service. Some additional work would be required to achieve seamless integration between the login screen and an Android agent.

Merging into krb5 Repository

We hope to be able to make a pull request of our code into the Kerberos master branch, so that future versions of Kerberos can benefit from the additional privilege separation our system provides. This will require us to build up test coverage of our code and make the network socket communications much more robust.