Kerberos Authentication of X Connections

Tom Yu

Abstract

Traditional Kerberos client-server authentication requires that each application have a secret server key. In the Athena environment at MIT, public workstations have a well-known root password, and the secure storage of a secret server key would not be feasible. Hence, extensions to X11 for using Kerberos authentication have not been undertaken. With the advent of Kerberos Version 5, user-to-user authentication allows for authentication without the use of a secret server key on the server, provided that the server has access to a user's ticket-granting ticket. The X server on a public Athena workstation runs as root, and therefore it is possible for it to obtain the user's ticket-granting ticket and authenticate an X connection without requiring the storage of a secret server key on the server.

Implementation of this authorization scheme was a challenging task, since the nature of the scheme used in the X protocol prior to X11R6 only provided for a single exchange of packets before the actual X connection became established. In the old protocol, the client sends a single packet to the server, which then uses the data to determine whether the client is authorized to connect to it. The server then sends back either a rejection or a success notification to the client.

Due to the extra packet exchanges needed for user-to-user authentication, this existing minor exchange of packets is insufficient, and consequently we implemented an extension to the connection opening protocol. We split the connection opening sequence into two parts: the first calls the authorization protocol and the second sends the success or failure notice to the client. The first part of the connection opening sequence will usually call the second, except for when protocols such as our Kerberos V5 extension are used. Splitting the connection code in this way allows authorization protocols to have multiple packet exchanges before completing the connection with the client.

Tom Yu is a student at the Massachusetts Institute of Technology.

The Public Workstation Dilemma

Athena, MIT's distributed computing environment, consists of a relatively large number of public workstations, in addition to private workstations and server machines. Public workstations have a well-known root password, and are not physically secure. Because of these shortcomings, it is not desirable for a public workstation to store a secret key, which it would need in order to provide a secure network service. In the past, Kerberos authentication has relied on the server's keeping a secret key with which to decrypt a client's service ticket for the server.

Client-Server Kerberos Authentication

The mechanics of Kerberos Authentication revolve around encrypted packets of data known as tickets. A Kerberos ticket, in the most general form, consists of the identities of both the client and the server, along with timestamps and other information to prevent spoofing. In addition, a randomly generated session key is part of the ticket. The entire ticket is usually encrypted in the secret key of the service, known only to itself and the Key Distribution Center (KDC). A Ticket-Granting Ticket (TGT) is special because the server is the KDC, and the key in which it is encrypted is a secret key known only to the KDC. The service which the KDC provides is also called the Ticket Granting Service (TGS), and all TGTs are encrypted in the TGS secret key.

In the original client-server model of Kerberos, the client requesting a service asks the KDC for a TGT. When the KDC receives a request for a TGT from a client, it constructs the TGT and encrypts it and the session key for the TGT, using the secret key of the client. Normally, the secret key of the client corresponds to the password of the user who is the client.

The client then decrypts the response from the KDC, using its secret key. To then use a Kerberos authenticated network service, the client then makes another request to the KDC, including the TGT and a time-stamped authenticator encrypted with the session key to the TGT. The request also includes the name of the service which the clients wants to access. The KDC receives the request, decrypts the ticket with its TGS key in order to obtain the session key sealed within the ticket, and decrypts the authenticator in order to verify that the client actually did originate the request.

The KDC then issues a ticket for the service. The KDC encrypts the service ticket with the secret key of the service. It then encrypts the service ticket along with the session key for the service ticket in the client's TGT key. The client, upon receiving this packet, can then use its copy of its TGT session key to decrypt the service ticket and session key. The client then can send a request to the service, accompanied with the service ticket and an authenticator encrypted with the service ticket's session key. The server can then follow steps similar to those followed by the KDC in order to authenticate the client. For further details about the Kerberos protocol, see [1].

User-to-User Kerberos Authentication

Since it is not possible for a public workstation to securely store a secret key, another approach is needed to allow for public workstations to provide Kerberos authenticated services. A possible solution to this problem is to use the TGT session key of the user logged into the workstation. The client begins the authentication process by making a request to the server. The server sends to the client in cleartext a copy of its TGT, but without the session key to the ticket. The client then receives the server's TGT and sends it, along with its own TGT and an authenticator constructed from its TGT session key, to the KDC.

The KDC receives this special request from the client, and decrypts the client's TGT with the TGS key to obtain the client's TGT session key. It then verifies the authenticator with this session key. The KDC now decrypts the server's TGT with the TGS key and assembles a special user-to-user ticket, consisting of the identities of the client and server like any ordinary ticket, but encrypted in the TGT session key of the server. The KDC has access to the server's TGT session key because this key is sealed in the server's TGT. The KDC then encrypts this new service ticket, along with its session key, in the TGT session key of the client.

The client receives this packet from the KDC and proceeds to decrypt the new service ticket and session key. The client can now form a service request to the server with the service key and form an authenticator with the session key. The server receives the request, decrypts the ticket with its TGT session key because it has no secret key, and can then decrypt the authenticator with the session key it obtains from the ticket. Further details about the origin of this user-to-user protocol can be found in [2]. The user-to-user protocol did not exist in Version 4 of the Kerberos protocol, but was added to Version 5. For more details about the differences between Kerberos Version 4 and Kerberos Version 5, see [3].

Application to X Authentication

The X server on a public workstation in the Athena environment typically runs as root. Thus, it has access to the Kerberos credentials of the user logged in. The X server can then use the session key associated with the user's TGT to authenticate incoming X connections. Clients wishing to connect to the X server can make use of the user-to-user protocol to establish an authenticated connection.

The X Authorization Scheme

The original connection setup protocol specification for X11R5 specifies that the client, in establishing a connection with the server, send an authorization protocol name along with authorization data to be verified by the server. The server then sends back a packet denoting either success or failure [4]. This exchange would suffice for Kerberos authentication if the X server had access to a secret service key. Unfortunately, the X server on a public workstation has no such secret key, and therefore must resort to the user-to-user protocol.

Limitations of the Previous Scheme

The previous X connection authorization scheme has some shortcomings. There is no way for the X server to send its TGT to the client without adding some extra packets to the exchange. There are several possible solutions to this problem. The X server can unconditionally accept any connection which indicates that it will use the Kerberos user-to-user authorization protocol, and use additional opcodes to continue the user-to-user authentication. This has the disadvantage of decreasing the number of available X protocol opcodes. The other possible solution was the one actually implemented in X11R6.

Instead of sending back a success or failure packet to the client, the X server sends its principal name and its TGT to the client, tagged with a special opcode to indicate that it is a continuation of the authorization protocol and not a success or failure packet. The X server and the client can continue to exchange packets as necessary to complete the authentication and authorization, at which point the X server can send success packet to the client.

Solution

Passing packets back and forth between the X server and the client before the connection is "officially" established allows the necessary exchange for user-to-user authentication to take place. The sequence is basically as follows:

- client sends connection setup request to server indicating Kerberos user-to-user protocol
- server sends packet to client including server's principal name and TGT
- client obtains user-to-user ticket from KDC and uses it to generate a request and authenticator for the server
- server sends authenticated reply to client in order to mutually authenticate
- client sends back acknowledgment to server
- server sends connection setup information to client

At any time during this exchange, if the server detects a problem with the authentication, it can send back a failure packet and disconnect from the errant client.

Implementation

Server Side

A new authorization protocol, `KERBEROS-V5-1`, was registered in the list of protocols known to the server. The function which normally takes responsibility for calling the authorization protocol needed to be changed. We moved the code in `ProcEstablishConnection()` which was responsible for transmitting the connection setup information into a separate function, `SendConnSetup()`, while leaving the code which called `ClientAuthorized()` in the function `ProcEstablishConnection`.

The implementation of additional packet exchanges in an authorization protocol prompted the creation of a new dispatch vector. Now, `ProcEstablishConnection()` will call the authorization protocol and check the `client->requestVector`, the dispatch vector associated with the client, afterwards. If the authorization protocol needs additional packet exchanges with the client, it modifies `client->requestVector` to point to its own dispatch table. `ProcEstablishConnection()` will recognize this, and fall back through to the dispatcher in order to allow other clients to be serviced. If the authorization protocol does not require extra packet exchanges, `ProcEstablishConnection()` will call `SendConnSetup()` to continue the connection setup as normal. If the authorization protocol does require additional packet exchanges, it is responsible for calling `SendConnSetup()` to indicate either success or failure as appropriate.

Another change which we needed to make to the X server internals related to the function `CheckAuthorization()`, which now needs to pass the `client` pointer to the authorization protocol in the event that the authorization protocol needs to communicate with the client. The

Kerberos user-to-user authorization protocol needs to exchange additional packets with the client, so it needs to receive the `client` pointer.

Client Side

We made modifications to the `XOpenDisplay()` code to allow for the use of the Kerberos user-to-user authorization protocol. If the Kerberos authorization protocol is to be used in the connection setup, then `XOpenDisplay()` will call `k5_clientauth()` to take care of it. If `k5_clientauth()` receives a failure packet from the X server before the entire authorization protocol can complete, it returns to `XOpenDisplay()` with a non-zero value, indicating that a connection setup prefix has already been read, in order to allow `XOpenDisplay()` to read the remainder of the failure packet properly.

Client-Server Additions

We implemented the traditional client-server authentication, in addition to the user-to-user authentication, for the sake of completeness. The protocol exchange follows the user-to-user protocol fairly closely. The only major difference is that X server does not send a TGT to the client; it only sends its principal name. The client takes the principal name and requests a normal service ticket from the KDC, and sends a request to the X server using that ticket. The rest of the protocol is essentially identical to the user-to-user authorization protocol.

Other Changes

We made a number of other changes in order to simply operation of the authorization scheme. The format of hostnames used by `xhost` and in the configuration file `/etc/X0.hosts`, for example, had to change in order to accommodate a new address family. We added the address family `FamilyKrb5Principal`, which could be used to add Kerberos principals to the list of authorized "hosts." Because the '@' character appears in SunRPC network principal names as well as in the string representation of Kerberos principal names, we decided to mostly abandon the previous scheme of determining the address family of a particular address by searching for special characters. Instead, `xhost` and `/etc/X0.hosts` use case-insensitive prefixes to specify the address family. The prefixes are as follows:

- `LOCAL:` non-network local transport, *e.g.* shared memory, Unix domain sockets
- `INET:` Internet hostname
- `DNET:` DECnet node name
- `NIS:` SunRPC principal name
- `KRB:` Kerberos Version 5 principal name

In addition, in the Athena environment, there is the question of how the X server accepts connections when there is no user logged in to provide a TGT session key with which to authenticate connections. We came up with a solution to this problem. The X server now checks its authority file each time it receives a connection opening request. If the file has changed, it rereads it and reloads the authorization protocols. This way, the X server can start up without using authorization protocols, and later load the Kerberos user-to-user authorization protocol when a user logs in. The `xlogin` program, which obtains Kerberos credentials for a user who is logging into Athena, can easily be modified to update the authority file to point to the correct Kerberos credentials cache.

Security Issues

It is a common vulnerability that Unix kernels can be made to believe that an IP packet which has the local system's address on it actually originated from the local system. In the past, there has been no way to switch access from local, non-network transport mechanisms on without also enabling network connections from the local system. An idle public Athena workstation needs to be able to accept local X connections without authorization protocols, since it has no user's Kerberos credentials from which to get a session key. There is, however, no need to open up such a workstation to attacks from anywhere on the network. We modified the code which allows non-authenticated access to the X server so that it is now possible to control access from non-network local transports, such as Unix domain sockets, and added a new address family, `FamilyLocalHost`, to accommodate this new option.

Future Directions

We gave some thought to the concept of including an option for end-to-end encryption of X connections. After some examination of the network code of Xlib and the X server, we determined that, while it might be relatively simple to add end-to-end encryption to the server, it would be rather difficult to add it to the client library. The X server maintains one queue for incoming packets from a client, and one queue for outgoing packets to a client. This makes it somewhat easy to adapt the X server to encrypt all data flowing to and from a particular client. Xlib, on the other hand, maintains separate queues for Event notifications, Reply packets, and Error packets. Adding support for end-to-end encryption here would require too many changes to too many segments of code. Pressed for time, we did not take this challenge.

Acknowledgments

Don Davis and Ralph Swick provided the original demonstration of the need for user-to-user authentication in the Athena environment, as well as the proposed solution [2] which became part of Kerberos Version 5. Ted Ts'o, Bob Scheifler, and Stephen Gildea assisted with many details of design and implementation. Abbe Cohen and Albert Dvornik read and suggested changes to earlier versions of this paper.

References

[1] J. Kohl and C. Neuman, "The Kerberos Authentication Service (V5)," RFC1510, University of Southern California, Information Sciences Institute (September 1993).

[2] D. Davis and R. Swick, "Workstation Services and Kerberos Authentication at Project Athena," MIT Project Athena (March 1989).

[3] J. T. Kohl, B. C. Neuman, and T. Ts'o, "The Evolution of the *Kerberos* Authentication Service," MIT Project Athena (June 1992).

[4] R. W. Scheifler, “X Window System Protocol,” Version 11 Release 5, MIT Laboratory for Computer Science (1988).

Author Information

Tom Yu may be reached at MIT Room E40-342, 77 Massachusetts Avenue, Cambridge, MA 02139, USA. Email: tlyu@MIT.EDU.