Trust Management of XMPP Federation

Simon N. Foley and Wayne Mac Adams
Department of Computer Science,
University College Cork, Ireland
Email: s.foley@cs.ucc.ie, w.macadams@4c.ucc.ie

***Abstract*—Deploying an XMPP server requires system security configuration, including firewalls and XMPP security controls. Security threats include DNS spoofing, rogue servers, inadequate authentication/authorization, spambots, etc. An system administrator who understands the threats and their mitigation manages the server configuration. This administrator must also deal with routine requests to update the configuration in order to federate with new XMPP domains. In practice it is non-trivial, time-consuming and costly to get the configuration right.**

We describe the development of a configuration agent that can automate some of these system administration activities while ensuring that the server is correctly configured and available. The agent is used by individual XMPP servers to (autonomically) configure when and how they should federate to provide end-to-end services. The KeyNote Trust Management system is used by the agent to help manage the trust relationships across the federation and to decide when it is safe to admit a new domain.

I. Introduction

The Extensible Messaging and Presence Protocol (XMPP) [3] is an open standard originally created by the Jabber community for instant messaging and presence and has evolved to support a much wider range of other services such multi-party chat, voice and video calls. An XMPP server may be configured to support host-based security controls. For example, an IP address white-list may specify the other XMPP servers with which a server may be willing to federate. The XMPP protocol supports different authentication methods, including simple password-based, dial-back and TLS-based authentication. In addition to server-to-server configuration, server to client system connections can be configured. Individual XMPP servers also support, to varying degrees, specific configuration options for application-level security controls whereby mediation is based on application-level content. For example, the Openfire XMPP server packet-filter and content-filter plug-ins provide user-to-user access controls and message filtering of message content based on regular-expressions.

The XMPP server administrator is responsible for ensuring that their system configuration adequately defends against appropriate threats. For example, the administrator of a corporate XMPP server may require that clients and servers connect only via TLS in order to mitigate DNS spoofing, while dial-back server authentication is an adequate mitigation is considered adequate for a public/free service. An administrator might also use a white-list to ensure that only organizations with business agreements are permitted to federate and connect with the server. In deciding whether the XMPP server of another organization should be added to the white-list, the administrator may require that the server of the other organization is adequately configured; for example, that it has adequate IM SPAM (SPIM) and Denial of Service controls in place. The XMPP standards foundation [3] provide some recommended best practices for secure configuration, such as XEP-0205 (Denial of Service), XEP-0178 (authentication) XEP-0165 (JID Mimicking) and XEP-0159 (SPIM).

We are interested in providing a degree of autonomic configuration support for the XMPP administrator. Rather than operating directly on the configuration, the administrator creates a business policy that defines the conditions under which the XMPP configuration may be changed. An automated agent running on an XMPP server, on behalf of the administrator, accepts configuration change-requests, and if the request is permitted by the policy then the agent updates the configuration. In the simplest case, the request originates from the administrator and the agent ensures that the resulting configuration is at least consistent with the current business policy. In the more general case, the request may originate from some third party, for example, the user of another server requesting federation. In this case, the agent must determine whether the high-level policy indicates that it can trust the originator, for example, that a business-agreement is in place and may update the white-list. This request should be acted on by the automated agent without requiring intervention of the administrator.

This paper describes the design and implementation of an agent service that, subject to a policy, can update an XMPP server configuration based on third-party requests. The KeyNote Trust Management system [4] provides the policy framework and is used to manage the trust relationships between the administrator and third-parties. An advantage of taking a Trust Management approach is that it does not necessarily rely on a centralized authorization/policy service and the policy rules can be distributed across the network. The paper is structured as follows. Section 2 provides an overview of Trust Management. Section 3 outlines the architecture and design of the XMPP configuration agent. Sections 4 and 5 explores how the agent might be used in practice by presenting a series of sample policies. Section 6 outlines the current implementation of the agent in the Openfire XMPP server.

II. Trust Management Systems

Trust Management [6] is an approach to constructing and interpreting the trust relationships among public keys that are used to mediate security critical actions. Cryptographic credentials specify delegation of authorisation between public keys. When a request from an untrusted principle (key) is made to a networked application to execute a particular action, then, authentication notwithstanding, the application must determine whether the key(s) that made the request is authorised. Trust Management provides assistance to applications in making these decisions and facilitates decentralized policies: authorization may be determined without having to consult some central authorisation server, and users may further delegate their authority without reference to a Central Authority.

KeyNote [4] is a flexible trust management scheme that provides a simple credential notation for expressing both security policies and delegation. Authorisation comes in the form of digitally signed public key credentials that bind public keys to the authorisation to perform various actions. For example, Sally may hold a credential, signed by her manager's private key, binding her public key to her authorisation to capture deals up to a value of £100.00. Sally's public key signs an order request to the purchasing application; her credential provides proof of authorisation.

In practice, authorisation is achieved by a collection of credentials that exhibit the necessary trust relationships between their keys. For example, we may trust Sally's public key for orders up to £100.00 if her manager's public key is trusted to delegate orders up to £100.00 (or more), and so forth along a delegation chain that ends in a key that is known to be appropriately trusted. Given a policy (public keys, trusted in known ways), and a collection of credentials, a network application must determine whether a particular public key is authorised to request a particular operation. KeyNote has been used to provide trust management for applications including active networks [5] and web servers [1].

III. Managing XMPP configurations using KeyNote Trust Management

Figure 1 depicts use of the XMPP agent. The goal of the Federated Autonomic Configuration (FAC) agent is to provide an orchestrated security configuration service for a secure XMPP service. This includes host-level controls via XMPP server-server and server-client configuration, application-level controls, for example, packet and content filtering controls and other security controls such as firewalls whose policies must be aligned with the XMPP policies. The current prototype is limited to host-based XMPP configuration controls and is intended to provide a proof of concept for the more general approach, which is a topic for future research.

A. Agent Behavior

A *Target* server with IP address T and public key K_T hosts a FAC agent and an XMPP server. The target server has a KeyNote policy that specifies the conditions under which it is willing to trust a request. A *Requestor* with public key K_R sends a signed request (to federate with XMPP server at address R) to T. This is done by including the address of the server that hosts the requester XMPP service in the request

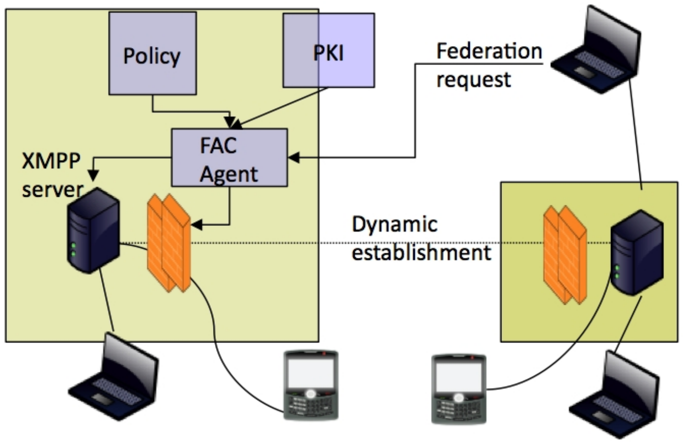

Fig. 1: XMPP Agent Application

along with KeyNote credentials that are intended to prove that the requester can be trusted for the request.

$$\begin{array}{lll} \text{Msg1}: & Requestor \rightarrow Target: & \{R, \ldots\}_{sK_R} \\ \text{Msg2}: & Requestor \rightarrow Target: & \text{KeyNote credentials ....} \end{array}$$

The target FAC agent confirms the signature on the message from the requester and, if valid, then the target queries (KeyNote) as to whether the public key K_R is trusted to carry out the action whereby [Server_IP_Address $\leftarrow R$]. If the query is successful then the Target FAC agent updates the *whitelist* entry on the Target XMPP configuration. If it is not successful then the request is rejected.

B. Target FAC Agent Policy

The target FAC agent has a policy defining the IP addresses it trusts, and thus willing add to the white-list, if requested to do so. For example, the KeyNote policy credential

```
Authorizer: POLICY
Licensee: TargetAdmin
Conditions:  App_domain="XMPP" &&
Server_IP_Address=~"^192\.168\.40\.5\d$"
Signature: ....
```

defines the conditions under which the identified `Licensee` can be trusted. For ease of exposition this is given as the identifier `TargetAdmin`; in practice it would be the public key K_T owned by the target administrator. The *Conditions* field uses a C-like expression syntax to specify the authority that the `Licensee` has been granted. In this case it is the authority to add address to the XMPP white-list that match the regular expression `"^192\.168\.40\.5\d$"`. Thus, the Target Administrator may use the agent to request that addresses in the range 192.168.40.50 - 192.168.40.59 be added to the white-list; any other requests are rejected.

C. Delegation Credentials

Suppose that the Target administrator trusts the administrator (with public key `KAlice`) of another XMPP server with address 192.168.40.55 and issues the following credential.

```
Authorizer: TargetAdmin
Licensee: KAlice
Conditions: App_domain="XMPP" &&
Server_IP_Address="192.168.40.55";
Signature: signed by target administrator
```

At some future point, when wishing to federate with the Target XMPP server, `KAlice` sends a signed request to the Target FAC Agent, along with the above credential. The FAC agent on the Target executes a KeyNote query as to whether the key `KAlice` can be trusted for [`Server_IP_Address` ← 192.168.40.55], given the policy credential and the above credential provided by `KAlice`. In this case a chain of trust exists from `POLICY` to `KAlice` that satisfies these conditions and the query is successful.

D. Managing Other Configuration Attributes

In addition to server whitelist update (based on attribute `Server_IP_Address`), requests to update of the Openfire *Packet Filter* plugin are supported. Based on the request, the agent modifies packet filter rules that are used to block or admit packets involving specific users and/or groups.

Continuing the example, `KAlice` decides to delegate authority to make a federation request to `KBob`, a user of the XMPP server managed by `KAlice`. She signs the credential:

```
Authorizer: KAlice
Licensee: KBob
Conditions: App_domain=="XMPP" &&
Server_IP_Address=="192.168.40.55"
&& Action_Authorizers==KBob
&& Date >= 20100101 && Date <= 20100201;
&& UserName=="Bob" && PacketType=="message"
Signature: signed by alice
```

Bob signs a request to federate his (Alice's) XMPP server with the Target server, and submits credentials as proof of authorization. His request results in KeyNote query [`Server_IP_Address` ← 192.168.40.55; `UserName` ← `Bob`; `PacketType` ← `Message`], which evaluates to `true`, the agent adds `192.168.40.55` to the server whitelist and adds a packet filter rule permitting `Bob`'s `messages`. Note, Bob may not delegate his authority further (`Action_Authorizers==KBob`) and that the credential is valid only for the dates specified. In this way federation is provided only to those who have requested it, thus ensuring the principle of least privilege.

IV. Example: Threat Mitigation

Most XMPP server administrators require mitigation of certain threats prior to federation. An XMPP server is vulnerable to a variety of threats, including DNS Cache poisoning, Denial of Server (DOS) attacks, worms, spim, and eavesdropping. These threats can be mitigated, both within the XMPP service and its hosting server. For example, within the XMPP service, attributes such as number of connection attempts and concurrent logins are suggested for avoiding DOS attacks [2], while firewalls provide external controls. Ordinarily a server administrator checks that a requesting server satisfies his security policy by confirming the threat mitigation with the requester administrator. In the following example we illustrate how this policy can be encoded in terms of KeyNote credentials.

The `TargetServer` policy states that only servers that mitigate eavesdropping and DOS may be trusted. Eavesdropping mitigation requires TLS (Transport Layer Security) for server to server communication in addition to SRTP (Secure Real-time Transport Protocol) for any video communication. DoS mitigation requires either an XMPP stanza size limit of less than 30000 bytes or else have both XMPP multi-user chat disabled and firewall packet filtering in place.

```
Authorizer: POLICY
Licensee: TargetServer
Conditions: App_domain="XMPP" &&
(Eavesdropping_Mitigation=="true"  ->
  {XMPP_TLS_for_s2s=="required" &&
   XMPP_SRTP_for_video=="required"};)&&
(DOS_Mitigation=="true"  ->
  {XMPP_Sanza_byte_size_limit<30000 ||
  (XMPP_Multi_User_Chat=="disabled" &&
  Firewall_Packet_Filtering=="enabled")};  );
```

The Target administrator issues a credential stating that XMPP security audits from a `SecurityConsultant` are trusted:

```
Authorizer: TargetServer
Licensee: SecurityConsultant
Conditions: App_domain="XMPP";
Signature: ....
```

Suppose that the `SecurityConsultant` audits Alice's XMPP server (`192.168.40.55`) and issues a credential specifying the controls that are present on her server.

```
Authorizer: SecurityConsultant
Licensee: Alice
Conditions: App_domain="XMPP" &&
Server_IP_Address="192.168.40.55" &&
XMPP_TLS_for_s2s=="required" &&
XMPP_SRTP_for_video=="required" &&; &&
XMPP_Sanza_byte_size_limit==25000 &&
XMPP_Publish_Suscribe_Service=="disabled"
&& Ingress_IP_Filtering=="enabled" &&
Third_Party_DOS_Mitigation_OK=="true";
Signature: .... by security consultant
```

These controls imply that Alice's configuration provides both `DOS_Mitigation` and `Eavesdropping_Mitigation` (on the basis of the target policy). Finally, Alice trusts her user Bob for XMPP application requests.

```
Authorizer: Alice
Licensee: Bob
Conditions: App_domain="XMPP" &&
Server_IP_Address="192.168.40.55"
Signature: ....by Alice
```

Using these credentials, Bob can make a (successful) request to the Target FAC agent to federate with Bob's XMPP server.

V. Example: Business Level Federation

A FAC agent can mediate white-list modification based on any KeyNote policy. This policymay be based on organizational or business requirements. For example, a business has two types of business agreements: b2b roles, which correspond to business-to-business agreements and b2c roles which correspond to a business-to-customer agreement. The business has a policy that any individual involved in a b2b agreement may federate XMPP and ERP services, while b2c agreements are limited to XMPP services. This policy is specified as follows, where TargetAdmin corresponds to the public key of the system administrator of the business.

```
Authorizer: POLICY
Licensee: TargetAdmin
Conditions: role=="b2b"->
 {App_domain=="XMPP" ||
  App_domain =="ERP";};
role=="c2b"->
{App_domain =="XMPP";};
Signature: ....
```

The system administrator issues a credential to the director of the business indicating that he/she is unconditionally trusted.

```
Authorizer: TargetAdmin
Licensee: TargetDirector
Conditions:
Signature: ....
```

Upon aquiring a new client `Angela`, the `TargetDirector` establishes a b2c relationship and issues credential:

```
Authorizer: TargetDirector
Licensee: Angela
Conditions: role=="c2b";
Signature: ....
```

Suppose that `Angela`, the director, has at some point issued a credential trusting her administrator `Alice` to properly administrate the host at 192.168.40.55

```
Authorizer: Alice
Licensee: Angela
Conditions:
Server_IP_Address="192.168.40.55"
Signature: ....
```

And, as above, `Alice` trusts XMPP user `Bob`

```
Authorizer: Alice
Licensee: Bob
Conditions:App_domain="XMPP" &&
Server_IP_Address="192.168.40.55"
Signature: ....
```

VI. Implementation

The FAC agent has been implemented to support configuration management of an Openfire XMPP server and comprises three components: an Agent Server, an Openfire Agent plugin and an Agent Client plugin.

The *Agent Client Plugin* generates a request to a remote target FAC agent. Currently, it provides a simple user-interface that allows a user to manually select the request and credentials to be sent to the server. We are investigating how this can be done automatically whereby a failure response to an XMPP connection attempt is trapped and generates a request to the corresponding FAC agent at the target and is, in turn, followed by a repeated connection attempt.

The *Agent Server* sits alongside the XMPP server waiting for connections from Agent clients. A request from a client is interpreted as a KeyNote query and the result is passed back to the agent which takes the appropriate action. If the requester is authorised then the Agent will dispatch a configuration change to to Openfire FAC Agent plugin.

The *Openfire FAC Agent plugin* acts on (trusted) requests from the FAC Agent Server. The current prototype provides support for Openfire whitelist update and addition of packet filter rules.

Openfire configuration updates by a FAC Agent are temporary and expire after a `ValidityPeriod`, which can be specified in the policy and/or delegation credentials.

VII. Discussion

This paper describes an agent that automates configuration management of an XMPP server. The current implementation is limited to policies involving host and user based controls, that is, white-list configuration and user/group based packet filtering. We are investigating how this agent may be extended to include other policy attributes. For example, support for attributes: `Authentication` specifying server authentication configuration options (none, password, dialback, TLS, etc.), and `Content_Filter` specifying a regular-expression to be added to the the Openfire content filter rules for blocking text. The agent will be extended to coordinate configuration with other security controls, such as firewall configuration [7]. This goes beyond the firewall-centric approach described in [8].

The FAC agent currently operates independently of the Openfire server where requests are made by a user via a simple user interface. Future research will investigate how requests can be made transparently (to the requesting user): on being denied a connection to a target service, the requester (XMPP server) requests a configuration change to the target FAC.

Acknowledgements. This research has been supported by Science Foundation Ireland grant 08/SRC/11403.

References

[1] "Apache-ssl," Open source software distribution. http://www.apache.org.

[2] "XEP-0205: Best practices to discourage denial of service attacks," XMPP Standards Foundation, http://xmpp.org/extensions/xep-0205.html.

[3] "XMPP standards foundation," Webpage, http://xmpp.org.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The keynote trust-management system, version 2, IETF RFC 2704," September 1999.

[5] M. Blaze, J. Ioannidis, and A. Keromytis, "Trust management and network layer security protocols," in *Security Protocols International Workshop*. Springer Verlag LNCS, 1999.

[6] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance checking in the policymaker trust management system," in *FC '98: Proceedings of the Second International Conference on Financial Cryptography*. London, UK: Springer-Verlag, 1998, pp. 254–274.

[7] W. Fitzgerald and S. Foley, "Management of heterogeneous security access control configuration using an ontology engineering approach," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, ser. SafeConfig '10. ACM, 2010, pp. 27–36. [Online]. Available: http://doi.acm.org/10.1145/1866898.1866903

[8] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *ACM Conference on Computer and Communications Security*, 2000, pp. 190–199.