ELSEVIER

The Journal of Systems and Software 58 (2001) 107–118

The Journal of Systems and Software

www.elsevier.com/locate/jss

A proxy-based security architecture for Internet applications in an extranet environment

Andy Dowling *, John G. Keating

Department of Computer Science, National University of Ireland, Maynooth, Room 2.110 Callan Extension, Nui Maynooth Co., Kildare, Ireland

Received 4 February 2000; received in revised form 5 August 2000; accepted 18 September 2000

Abstract

Current Internet communications security is typically provided by the integration of secure transport functionality into client and server software. Two problems arise with this approach: Firstly, the use of integrated security services requires modification to the existing Internet applications, requiring re-development and re-deployment projects. Secondly, high-level security services such as authorisation are not provided by secure transport protocols, requiring applications to rely on customised (and often insecure) mechanisms for the provision of such services. We propose a platform-independent system that uses proxy applications to provide both secure transport and authorisation services transparently to existing Internet applications. We demonstrate that our approach requires no modification to existing applications, and that our security services are based on existing and widely used technologies. We discuss the merits of our architecture in the context of the intended deployment environment: an Internet-based heterogeneous private network such as an extranet or Virtual Private Network (VPN). We show that our approach achieves its goals at the expense of introducing a minor degree of performance loss into overall client–server communications, yet we maintain that this performance loss is a minor expense in relation to the advantages of the system as a whole.

Keywords: Internet; Extranet; VPN; Security; Proxy; Authorisation

1. Introduction

Internet-based client and server applications communicate with each other using a command and message format appropriate to the applications' protocol, which may be HTTP, FTP, SMTP, etc. By default, the application data is transmitted between client and server without any security mechanisms in place, leaving the client–server communications vulnerable to interception, modification and masquerading attacks.

1.1. Current practices

In order to defeat such attacks, the basic security services of privacy, integrity and authenticity are provided by an appropriate secure communications service. A popular approach is to import the services of a secure communications protocol such as SSL (Freier et al., 1996) or TLS (Dierks and Allen, 1999) into the applications. To avail of the secure communications support offered by SSL and other security protocols, both client and server applications must be modified to use this functionality, typically by making appropriate API calls to imported security libraries. Two major limitations with this conventional approach have been identified:

1. *Integrated security services*. Integrating security services into existing applications requires re-development and re-deployment of the applications. This integration process requires a separate re-development project for each application program used in a particular domain. In addition, the integration process may also inject new defects into the modified application in such a way as to cause a security breach under certain conditions. Moreover, if multiple applications have integrated security services, the amount of security-related administration increases linearly with the number of secured applications, since each application requires individual administration.
2. *Authorisation and access control*. The security requirements of a client–server environment are not entirely met by secure transport services such as SSL. Internet-based distributed environments such as extranets/VPNs often require secure authorisation and access control services in order to restrict access to

* Corresponding author. Tel.: +353-1-7086082; fax: +353-1-7083848.
E-mail addresses: andy.dowling@may.ie (A. Dowling), john.keating@may.ie (J.G. Keating).

0164-1212/01/$ - see front matter
PII: S0164-1212(01)00031-0

networked resources. Secure communications protocols such as SSL do not provide adequate authorisation and access control services, and consequently applications must implement their own services to complement any existing secure transport functionality. This leads to several problems:

(a) There are no security guarantees when applications implement their own authorisation and access control services. An example of this is apparent in web-based environments: a HTTP server may authenticate a client using SSL, but still controls access to web resources using an independent username/password combination, leaving the system vulnerable to password-stealing attacks. Whilst some applications may achieve security through using the underlying authenticated identity for controlling access, this is not guaranteed, and varies between applications.

(b) The lack of any generic binding between the authenticated identity and the associated set of privileges means that server applications tend to rely on authenticated identities (i.e. usernames) to control access to resources. If access is required to resources that require different privileges, the client must authenticate using a different identity. In the web-based example mentioned above, this results in the user managing a different username/password combination for each resource that they require access to.

(c) The administrative overhead involved in some custom access control mechanisms may be quite large. In the web-based example presented previously, Access Control Lists (ACLs) may be maintained on a web server site, each containing a list of users and/or hosts that are allowed access to a certain directory. As a site grows in size and as the access policy of the site becomes more complicated, these ACLs increase in both number and size, leading to large administrative overhead. Quantitatively, this overhead has an upper bound of $m * n$ ACL entries for a site with m restricted directories and n users.

(d) Application-dependent authorisation mechanisms take different approaches to representing and exchanging authorisation information. This leads to a breakdown in interoperability between applications that use customised authorisation functionality.

1.2. Solutions

Clearly, a different approach is required in order to overcome the problems introduced by integrated security services and application-dependent authorisation support. We take the approach of providing secure communications and access control using proxy applications.

1. *Proxy-based security services.* Instead of using integrated security services with existing applications, a more beneficial approach is to provide all security services separately to the applications by placing security proxy applications on the connection between client and server. Security proxies are deployed at both client and server sites, and networked applications have their communications transparently redirected via the proxies for security processing. A single proxy can be used by multiple applications (i.e. multiple clients or multiple servers), thus reducing the amount of security-related administration to a single application per site.
2. *Advanced authorisation and access control.* To overcome the many problems associated with application-specific authorisation services, a generic form of authorisation and access control service is required that is secure, highly interoperable, easy to administer and convenient to the end user. This can be achieved by using a combination of digital certificate technology and advanced access control models. Standardised digital certificate structures such as the X.509 Attribute Certificate (AC) (ITU, 1997 E; Farrell, 1999) may be used to securely bind a set of access privileges to a particular identity (such as an X.509 Public-Key Certificate (PKC) (ITU, 1997 E)) in a highly interoperable manner. The de-coupling of the identity from the access privileges means that the end user no longer manages multiple identities in order to access different resources. Instead, the end user authenticates once using their PKC, and the contents of the user's AC are used for making server-side access control decisions.

Administration is reduced by restricting access in terms of the user's attributes rather than the user's identity. Attributes such as roles, groups, and access identities can be used to implement more flexible access control models such as Role-Based Access Control (RBAC) (Ferraiolo and Kuhn, 1992), which reduce the amount of security administration involved.

In this paper, we describe a proxy-based security architecture called YAPS (Yet Another Proxy System), that provides both secure communications and advanced authorisation support whilst requiring no modification to existing Internet applications. We discuss the YAPS architecture in the context of deployment within a VPN or extranet environment. We outline the YAPS architecture, discuss its operation, and we evaluate the system based on a number of criteria. We conclude with a statement of further research.

2. Related work

Claessens et al. (1998) describe and compare the application of various proxy systems to the problem of

weak SSL in exported US web browser applications. They propose a proxy-based model called "proxy and spoofing" to provide secure communications to web browser software without requiring any modifications to the browser. There are two limitations with this model. Firstly, this approach is limited to HTTP-based clients and cannot be applied to Internet applications in general. Secondly, the security enhancements are only made to on the client side, and do not apply to server applications. As a consequence, no authorisation and access control services are provided.

SESAME (Kaijser et al., 1994) is a security architecture based on Kerberos (Johl and Neumann, 1993) that provides strong communications security and advanced authorisation services to Internet applications. The SESAME architecture provides security components which can be used by both client and server applications. Authorisation support consists of an RBAC model that uses Privilege Attribute Certificates (PACs) (European Computer Manufacturers Association, 1996) for privilege representation. Whilst SESAME offers both secure communications and advanced access control, SESAME security must be integrated manually into applications. Furthermore, SESAME is considered a closed system as it relies on proprietary protocols rather than (de-facto) standard protocols such as SSL.

3. YAPS – Yet Another Proxy System

3.1. Overview

YAPS uses security proxies deployed at both client and server sites in a "double proxy" architecture (see Fig. 1). Connections to server applications protected by YAPS proxies are secured between YAPS client and server proxies using an arbitrary peer-to-peer secure transport protocol such as SSL or TLS. Proxy-to-proxy communications take place over the SOCKS V5 (Leech et al., 1996) protocol. X.509 PKCs are used for authentication. Privileges are encoded as attributes in X.509 ACs, and access to server resources is controlled using an attribute-based access control model. The access control model also encapsulates the policy-based model used in the Akenti (Johnston et al., 1998) project, permitting the distributed administration of access rights to resources. The YAPS architecture consists of four main components: certificate issuing authorities, the YAPS client proxy, the YAPS server proxy, and the certificate acquisition components.

3.2. Certificate issuing authorities

In order for YAPS to provide authentication and authorisation services, YAPS requires existing X.509 PKC and AC issuing authorities. These authorities are used in an offline manner, and are not illustrated in Fig. 1 for clarity. These offline authorities are trusted by the YAPS security components to issue certificates and CRLs. Certificate trust parameters are stored in local security databases maintained at the client and server proxy sites.

3.3. YAPS client proxy

The role of the YAPS client proxy is to accept connections from client applications and relay the connection between the client application and the destination server. If the destination server is reached via a YAPS server proxy, then a secure connection is established with that server proxy and the client request is tunnelled through the secure connection.

The YAPS client proxy resides on the client host and listens on a configured TCP port for connections. The SOCKS V5 protocol (with cleartext payload) is used to communicate with local client applications over the local network interface. In order for the processing to appear transparent to the client applications, calls to the

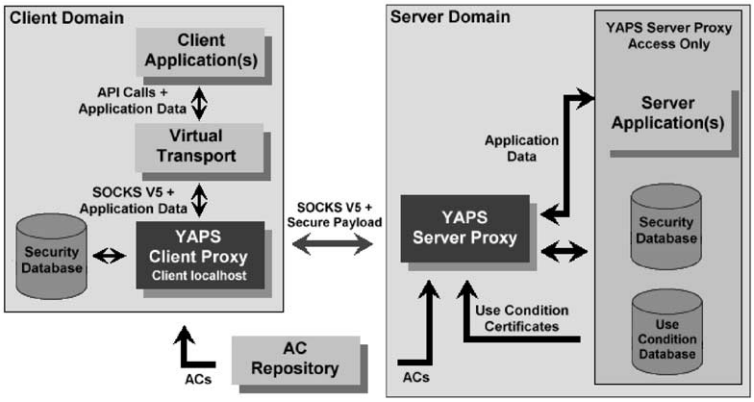

Fig. 1. YAPS architecture.

client's transport service (e.g. WinSock) are dynamically intercepted, translated into SOCKS V5 requests, and are redirected to the local client proxy. This is performed by an additional component residing on the client host.

3.4. YAPS server proxy

The YAPS server proxy is a SOCKS V5 proxy server that acts as a security gateway to server applications. The server proxy is deployed in such a way that all incoming connections to the server application must pass through the server proxy for security processing before an indirect connection is made to the destination server by the server proxy. To prevent direct connections to the server applications from clients, an arbitrary firewalling system is used.

3.5. Communications architecture

In the tradition of networking systems (e.g. OSI and TCP/IP, Tanenbaum, 1996), the communications architecture of the YAPS proxies is split up into a number of conceptual layers, with each layer handling data at a particular level of abstraction. Both client and server proxies handle data at three layers. Fig. 2 shows the architecture for the client proxy.

1. *Proxy layer*. The proxy layer uses the underlying TCP/IP service layer to establish connections to other YAPS proxies using the SOCKS protocol. SOCKS is used to negotiate connection parameters and to select a secure transport protocol for data exchange.
2. *Security layer*. The security layer is responsible for negotiating a secure connection with the security layer on the peer host. Data is accepted from the application layer, encapsulated in messages specific to the chosen security protocol, and is passed to the proxy layer for transmission as SOCKS payload data.
3. *Application layer*. The application layer is ultimately responsible for invoking data exchanges between YAPS proxies and remote applications. Data such as HTTP requests/replies and POP3 commands are handled at this layer. The services provided by the underlying layers are used to wrap the application data in a format appropriate for the connection type (i.e. to local application or remote YAPS proxy).

Note that not all connection types require the services of all three layers. The YAPS client proxy uses a clear-text SOCKS payload to the local client applications, and consequently does not use the security layer. Similarly, the YAPS server proxy communicates directly with server applications, and bypasses both the security and proxy layers.

3.6. Access control services

In addition to the secure transport functionality offered by the YAPS server proxy, the server proxy also provides access control functionality. Access control is provided via a number of core components (Fig. 3).

3.6.1. Access Decision Functions

The Access Decision Functions (ADFs) provide a generic means of determining if access to a requested resource should be permitted or not. The YAPS ADFs accept and process a number of arguments in order to make an access control decision. Once the decision has been made, the result is either *true* or *false*, denoting access permitted or denied respectively. The arguments used by the YAPS ADFs are:

1. *Requested resource*. The resource requested by the client. Typical resources include web pages (HTTP), files (FTP), or e-mail boxes (POP3).
2. *Requested operation*. The operation that the client wishes to perform on the requested resource. Operations include web page retrieval (HTTP GET), web form posting (HTTP POST), or deleting an e-mail from a mailbox (POP3 DELE).
3. *Requestor's attributes*. The attributes associated with the (authenticated) client. These are obtained by

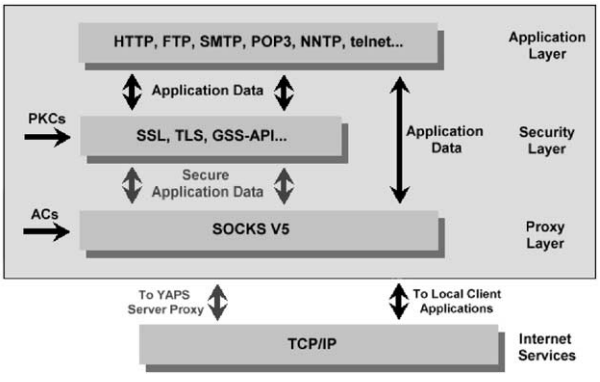

Fig. 2. YAPS layered internal communications architecture (YAPS client).

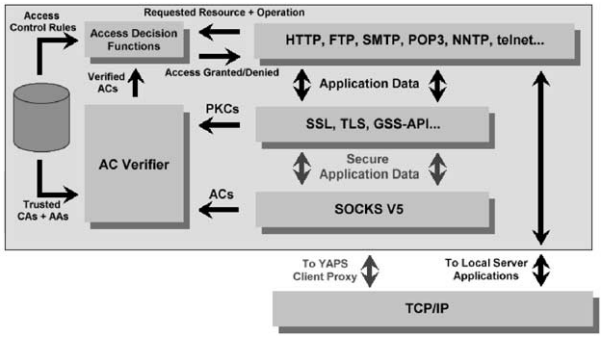

Fig. 3. YAPS server proxy access control architecture.

firstly acquiring the client's AC, passing it through the AC verifier and extracting the attributes from the verified AC.

4. *Access control configuration*. The access control configuration pertaining to all YAPS-protected server applications. This denotes which resources are restricted, and the requirements that must be satisfied by requesting clients in order for access to the resource to be permitted.

Application protocols use different means of representing resources and operations. In order to achieve application protocol independence in access decision making, the ADFs do not process the semantic values of resources and operations. Instead, ADFs compare the resource and operation values to the stored access control requirements, and make the access control decision based on this comparison. This notion of generic resource and operation representations is central to the making of protocol-independent access decisions.

3.6.2. AC verifier

The AC verifier is responsible for verifying the integrity and contents of X.509 ACs according to a set of validation rules. The YAPS AC verifier is compliant with the IETF AC verifier validation semantics, as defined in the IETF AC Internet authorisation profile (Farrell, 1999).

3.6.3. Application protocol handlers

In order to implement access control using application-specific resources, some application-specific processing is required by the server proxy. The YAPS server proxy uses application-level processing modules called "protocol handlers" to parse data at the application level (Fig. 4). The protocol handlers extract the application-specific data from the connection, translate it into a generic form used by the ADFs, and invoke the ADFs to determine if access should be granted or denied. If permitted, the protocol handlers continue to forward application data between the client proxy and the server application. If access is denied, an application-specific error message is formulated by the protocol handler and sent back to the client (e.g. HTTP Error 403 for a web-based system).

3.6.4. Access control list

The access control configuration for resources exported by application servers is stored in the YAPS ACL (Fig. 5). The ACL contains a series of rules, each rule containing a restricted resource and a series of access rules for that resource. The access rules map a "requirements expression" onto a set of operations that are permitted to be performed on the associated resource should the requestor satisfy the expression. A requirements expression is a boolean expression defined in terms of the attributes required in the requestor's AC in order for access to be granted. Note that resources are represented in a generic form (URL), and operations are represented "as-is" (i.e. strings as they appear in the application protocol exchanges).

YAPS requirements expressions can take one of two forms: attribute-based or policy-based.

1. *Attribute-based expressions*. Attribute-based expressions are used to implement attribute-based access control. This allows the encapsulation of role, group, and user-based access control models. Access requirements are expressed in terms of attribute-value pairs.
2. *Policy-based expressions*. Policy-based expressions add an extra level of indirection into the ACL by implementing the policy-based access control model used in the Akenti (Johnston et al., 1998) project. Policy-based expressions denote requirements in terms of the "stakeholders" (i.e. resource owners) who define the access requirements for the restricted resource. Policy-based expressions are used to allow stakeholders to administer the access requirements for their own resources. Access policies are stored locally as stakeholder-signed ACs on the YAPS server proxy as part of the security database (along with the ACL).

Both attribute-based and policy-based requirements expressions are similar to boolean expressions used in programming languages such as C and Java™, and make use of the logical operators "AND", "OR" and "NOT" for formulating compound requirements expressions. Parentheses may also be used to explicitly

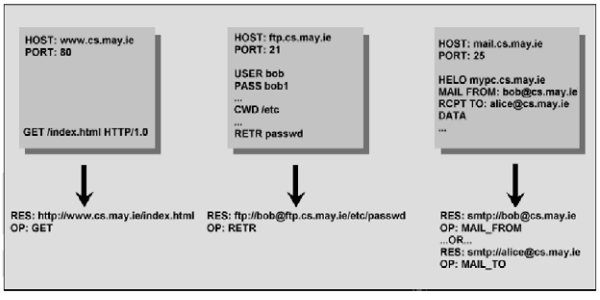

Fig. 4. Protocol-specific resource translation into generic ADF style.

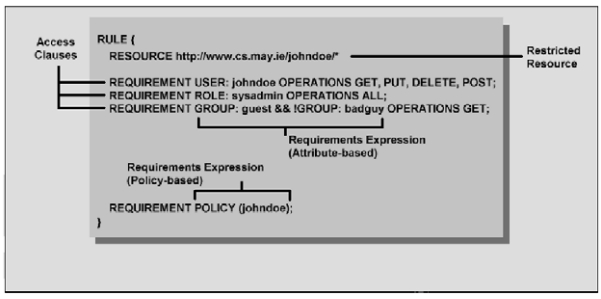

Fig. 5. YAPS ACL components.

specify evaluation precedence in requirements expressions, which defaults to left-to-right for all operators except logical negation. As a rule of thumb, the order of evaluation is the same as for the Java programming language, with parenthesis having the same effect on precedence as for Java.

3.7. Certificate acquisition mechanisms

In order to make an access control decision, the server proxy must acquire the client's AC. There are two mechanisms used by YAPS to accomplish this. The first model, known as the "pull" model, involves the server proxy retrieving the AC using an appropriate acquisition protocol. The second model, called the "push" model, specifies that the client retrieves its own AC (using the pull model), and sends it to the server proxy during proxy-to-proxy communication. Both models have relative advantages and disadvantages, and each is suited to a particular networked architecture. Hence, choosing an AC acquisition model is dependent on factors such as system architecture requirements and organisational security policy. For maximum compatibility with existing distributed system architectures, YAPS can use both the push and pull models.

3.7.1. AC pull using LAAP

YAPS uses the Limited Attribute certificate Acquisition Protocol (LAAP) (Farrell, 1999) for AC pull. LAAP is an experimental IETF protocol designed for the acquisition of ACs. Since ACs are relatively short-lived to PKCs, they are more likely to be generated "on the fly" by online AC server components rather than being stored to/acquired from a certificate directory. For this reason, LAAP is being developed as an AC acquisition protocol in preference to re-using existing protocols such as LDAP (Yeong et al., 1995). Technically, LAAP is a simple cleartext request-reply protocol, which may be encapsulated in SSL/TLS for security as required.

The main advantage of the pull model is that no modifications to the client–server communications protocol are required. Indeed, the client need not be "AC aware" at all. Consequently, AC pull can be used with any communications protocol between client and server proxies. However, the AC pull model imposes additional load on the server proxy, which must acquire the client's AC. This introduces some additional processing, network and database access overheads.

3.7.2. AC push using SOCKS V5

The YAPS implementation of the AC push model involves the transmission of the client's AC from client to server proxy during the SOCKS V5 negotiation. The AC push is similar to the work on AC push previously published in Internet drafts within the IETF TLS working group (Farrell, 1998). We prefer the implementation of AC push in the SOCKS negotiation for two reasons:

1. *Security protocol independence*. AC push takes place independently of the secure communications protocol used, since the security handshake takes place independently of the AC SOCKS exchange. This means that AC push can be used with any secure transport protocol.
2. *Reduced protocol modification*. There is no need to integrate AC push into existing security protocols (TLS and others) if the security protocols are encapsulated in this enhanced SOCKS protocol.

Our approach involves the introduction of a new SOCKS authentication method, which we refer to as the "AC Exchange" method in this text. All AC-related exchanges take place as a sub-negotiation after this method has been selected, and therefore the compatibility with existing SOCKS-based systems is not affected. When a SOCKS connection is established (see Fig. 6), the client sends a list of authentication methods that it supports (1). One of the methods in the list is the AC Exchange method identifier. Once the YAPS receives the list and recognises the AC Exchange identifier in the list, the server notifies the client that AC Exchange has been selected (2), followed by the selected authentication method (3). Authentication then takes place between client and server (4) using the selected method. Once authentication has completed successfully, the server sends an "AC Request" message, requesting the client to submit an AC to the server (5). The client sends an "AC Response" message (6) containing the appropriate status information, plus the client's AC, if available. To complete the handshake, the server sends an "AC Exchange Complete" message to the client (7). The method dependent sub-negotiation has now finished, and the remaining SOCKS exchanges proceed as normal (8).

In some cases, the server proxy may not require the client to push the AC at all. For performance reasons, the server proxy may request the client to push an AC

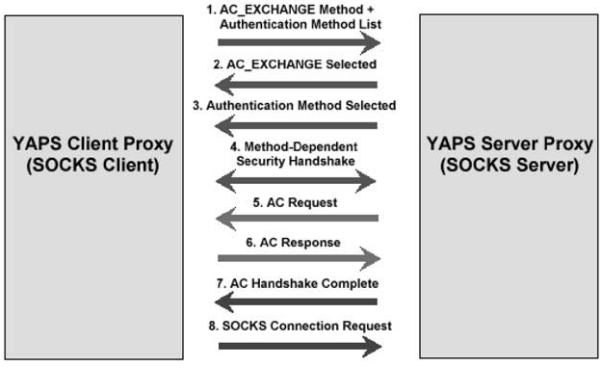

Fig. 6. AC push in SOCKS V5.

during the first connection request, and retrieve it from an AC cache for subsequent requests. To accommodate such situations, the AC request and AC reply messages in the above example are optional. The YAPS client proxy will only push an AC to the server proxy if an AC request message is received after the authentication phase. If the server proxy does not require an AC, an AC Exchange Complete message is sent to the client instead, and the client does not push an AC to the server.

The main advantage of the AC push model is that the client's AC is immediately available to the server proxy for verification, and no additional load is imposed on the server proxy. YAPS can use either AC push *or* pull, depending on the implementation. Server implementations that support AC pull will not use the "AC Exchange" SOCKS method, but will use whatever authentication method is selected to establish the identity of the client for use in the LAAP request during AC pull.

4. Implementation

We have developed a prototype implementation of YAPS to demonstrate the system in operation. The majority of the YAPS implementation was accomplished by developing a number of Java™ applications. Java was chosen as the development language for obvious reasons: portability, reduced development time, strong typing, and thread support. SSL, cryptographic, ASN.1/DER and X.509 functionality was provided by the IAIK JCE and iSaSiLk toolkits (Institute for applied information processing and communications), whilst ACL parsing functionality was provided by the JLex (Berk) and B/YACC (Jamison) tools.

Our implementation consists of:

1. *Certificate issuing applications*. The suite consists of four Java applications responsible for the generation of self-signed PKCs, CA-signed PKCs and CRLs, ACs, and use-condition certificates.
2. *YAPS proxies*. Both YAPS proxies are multi-threaded Java applications that run on the respective client/server host and provide the relevant services. Protocol handler components were implemented for performing protocol-specific access control, and were dynamically "plugged in" to the server proxy using the Java class-loader mechanism.
3. *LAAP server*. A simple multi-threaded LAAP server has been implemented to serve ACs to requesting clients (i.e. YAPS proxies).

Additional functionality such as AC generation/validation, SOCKS V5 support, and ADFs were implemented in Java and imported into the applications. The transparent interception of calls to the clients' WinSock service was accomplished by deploying the commercial SOCKSCap (NEC Networking Systems Laboratory) product on the client host.

Access control protocol handlers were implemented on the server proxy for the HTTP, SMTP and POP3 protocols. The prototype was tested with Netscape Communicator, Internet Explorer and Microsoft Outlook. Tests were conducted on the Windows 95™ platform, using Pentium II 350 MHz PCs for client, server, and AC server applications. Performance tests (HTTP) on the prototype resulted in a measurable access decision process of 0.1 s with server-proxy AC caching, 0.2 s for non-cached AC push, and 1.0 s for AC pull (cleartext LAAP).

5. Discussion and evaluation

The existing security architectures mentioned previously in this paper involve a trade-off between the available security services and the transparency to Internet applications. Systems such as SESAME (Kaijser et al., 1994) provide both secure communications and authorisation services to Internet applications, but require the applications to be modified in order to avail of the services. Proxy-based systems like SafePassage WebProxy (C2Net Software Inc.) and SafePassage SecureTunnel (C2Net Software Inc.) operate transparently to applications, but offer secure transport without flexible authorisation support. We now discuss how YAPS eliminates this trade-off using the double-proxy architecture and advanced access control.

5.1. Transparent security services

The YAPS architecture consists of security proxy applications deployed at both client and server sites. The YAPS proxies implement security processing on the communications channel between client and server applications, and consequently security processing takes place transparently and without modification to the communicating applications. Client requests are transparently intercepted by a virtual transport service running in the client host's operating system, and are redirected to the YAPS client proxy as SOCKS V5 requests. Since the interception of client requests takes place at the transport layer, the client application operates as if it were communicating directly with the server application. On the server side, the server proxy acts as a client to the server application. Therefore, the client and server applications operate as if they were communicating directly with each other, and are not aware of the intermediate security processing.

Communications between YAPS proxies are established using the de-facto standard SOCKS V5 protocol. The initial SOCKS messages are used to select a secure communications protocol and consequent exchanges are

encapsulated in the selected protocol as payload data. Our implementation uses SSL for secure transport, and hence the communications channel between proxies is protected against interception, masquerading and modification attacks.

5.2. Authorisation support

In addition to transport security services between proxies, the YAPS architecture was shown to provide a secure, generic and flexible authorisation service that reduces security administration and end-user management tasks.

5.2.1. Security

The security of any system is only as strong as its weakest component. The YAPS architecture is a combination of existing and custom-made components, and the security of these is crucial to the security of the system as a whole. We now evaluate the security of the system in four main areas with a view to gaining an overall view of the intended security level provided by YAPS:

1. *PKI security*. YAPS assumes the existence of a PKI with the appropriate CAs and certificate/CRL repositories in place. The YAPS proxies only accept validated PKCs issued from trusted authorities. PKC chains from SSL sessions and the PKCs of AC issuers that are handled by the YAPS proxies are validated in accordance with RFC2459 (Housley et al., 1999) before they are used. The private keys associated with YAPS applications are stored locally in PKCS12 (RSA Laboratories, 1999) format, and are protected with a PIN-derived symmetric key.
2. *Connection security*. Connections between YAPS proxies are secured with an arbitrary peer-to-peer transport security protocol that uses X.509 PKCs for both client and server authentication. Using SSL or TLS as the transport security protocol will result in a transport layer that provides authentication, data privacy and data integrity through strong cryptography. Connections to AC servers via LAAP may be protected by SSL (although this is not a requirement in all cases, it may be used where AC contents are not to be sent unencrypted over the wire).
3. *Authorisation security*. User attributes, like public keys, are protected by an underlying PKI. Access privileges are represented using X.509 ACs, which are securely bound to the PKC of an authenticated user. The resulting AC is protected against fabrication by a digital signature produced by the (explicitly trusted) AC issuing authority. Like PKCs, ACs undergo a series of rigorous syntactic and security checks by the AC verifier component in the server proxy before they are used in the making of an access decision.
4. *Architectural security*. Access to the application servers containing privileged data is restricted to the YAPS server proxies using an arbitrary firewalling technique. This may be done at the server configuration, by using separate TCP wrappers, or by a dedicated firewall. These measures, in conjunction with a correct network architecture (i.e. "sane" routing), will prevent the by-passing of the YAPS server proxy by incoming clients. The basic access policy of what security protocols to use with incoming clients (in the SOCKS V5 handshake) is configured at the server. Implementing an SSL-only policy (with SSL client authentication) will ensure that all incoming clients are strongly authenticated before any access decisions are invoked by the protocol handlers.

The intended security level of YAPS is strong. The strength of the security is dependent of the strength of the underlying PKI. A PKI that is weak in either configuration, management, or implementation will result in that weakness being propagated to YAPS and any other security architectures dependent on that PKI.

The machines that host the YAPS proxies are assumed to be secure in the context that only the YAPS server proxy administrator can access them. The consequences of an attacker gaining full access to a machine running a YAPS server proxy would be disastrous. Completely preventing a hacker who has gained access to a server proxy from compromising YAPS very difficult, if not impossible. Our implementation stores private key information in an encrypted form (using PKCS12), and both the ACL and security database are digitally signed. In a future implementation, one could take this a step further and use tamper-resistant hardware to store private key information.

5.2.2. Interoperability

The use of ACs for YAPS authorisation services conforms to the current IETF Internet authorisation profile. This, in combination with the X.509 standard ASN.1/DER AC encoding, provides a basis for highly interoperable privilege representation and promotes interoperability between YAPS and other security architectures. Since privileges are encoded in the AC data structure, their use is not limited to a particular application or application protocol, but may be exported for use in third-party open systems.

5.2.3. YAPS management issues

The management and administration tasks required by YAPS can be broken down into a number of sections:

1. *PKI management*. YAPS is built upon an existing PKI. The management tasks associated with a PKI exist regardless of whether or not YAPS has been deployed. In a "real-world" PKI deployment, these tasks include CA management, PKC (and possibly

key) generation, directory management, PKC and CRL publishing.

2. *Proxy management*. Each YAPS server proxy maintains its own configuration and operates in its own environment, requiring individual administration. The way in which YAPS is deployed in front of application servers, (i.e. single proxy, multiple servers; or multiple proxies, multiple servers) determines the amount of management required. In practice, each proxy that is deployed will require separate management. At worst, the number of applications that require management doubles (multiple proxies, multiple servers). At best, management of one additional proxy is required (single proxy, multiple servers). Either way, the management of security is de-coupled from the actual server application and is done at the proxy(ies), providing a consistent and uniform security management interface.
3. *Application server management*. Security and access control management tasks are delegated to the YAPS proxy associated with a particular application server. All other management tasks remain with the application server. By migrating security-related tasks to the proxy, the attribute and policy-based access control models can be exploited to significantly reduce the amount of access control administration across all server applications.
4. *Privilege management*. YAPS introduces the requirement for an AA and an AC server (which, in practice, are the same application) to produce ACs upon request. Identity to attribute mappings are managed at the AC server, (i.e. role and group membership are assigned to users here) along with the configuration of the AC server itself. If policy-based access control is used, the management tasks involved in defining access control requirements for particular resources are delegated to the resource owners.

In addition to the management of the underlying PKI and the server applications themselves, YAPS requires management of the proxies and the AC server. Whilst this is an obvious burden on the system administrator, YAPS compensates by reducing the workload involved in access control maintenance. YAPS also provides a consistent interface for security management across multiple application servers.

5.2.4. Ease of use

The reduction in security administration on the server side is mirrored on the client side. By using the YAPS client proxy to handle security, the end-user does not have to manage numerous security configurations and passwords across various client applications. The user need only remember a single pass-phrase in order to unlock their private key for the client proxy, and use their applications with a single YAPS configuration to securely access remote YAPS-protected servers.

5.3. YAPS limitations and workarounds

The YAPS architecture is by no means a perfect security solution. Whilst YAPS does offer more in terms of security and flexibility than the existing systems mentioned earlier, there are a number of minor drawbacks associated with the YAPS architecture. We now discuss these limitations, and we outline possible solutions and workarounds to these issues.

5.3.1. Routing

In order for the YAPS client proxy to determine whether a destination server is accessed via a YAPS server proxy, it must maintain a lookup table mapping destination addresses to YAPS server proxy "gateway" addresses. This is analogous to the concept of IP network routing (Tanenbaum, 1996). In environments where the number of "secure domains" protected by YAPS server proxies is small, this is a trivial issue. However, for large networks with many separate networks accessed via YAPS server proxies, maintenance and configuration issues regarding the YAPS lookup tables become more prominent. The responsibility for maintaining the correctness of the routing tables falls on the end-user, contradicting the goal of reduced end-user management tasks in the first place.

An interesting solution to this problem involves remote configuration. Upon startup, the YAPS client could connect to a local LAAP server and download a lookup table encoded in an AC issued by the local system administrator. In addition to removing the responsibility of routing table management from the end-user, the routing table configuration is protected against fabrication.

Another routing-related problem arises when "bind" requests are made by the client application. When this occurs, the client proxy cannot always determine whether to bind to a local port, or to relay the request to a server proxy. Moreover, if multiple routing entries exist for multiple server proxies, the client cannot determine which proxy the bind request must be relayed to. One possible solution to this problem is for the client proxy to anticipate whether or not to forward the bind request based on the state of existing connections opened by the same application. For example, examination of run-time logs produced by SOCKSCap has shown that, when FTP clients make a bind request to accept file data from a download request, SOCKSCap recognises the existing FTP control connection and consequently relays the request to the appropriate proxy so that the FTP data channel can be secured. This solution only works for protocols that firstly establish a connection with a remote server application using a "connect" request, and would not work for more complicated protocols that may not use an initial outgoing connection.

5.3.2. SOCKS V5

YAPS relies heavily on SOCKS V5 and is therefore restricted by limitations imposed by the SOCKS protocol itself. One such limitation is the inability of SOCKS V5 to proxy incoming UDP datagrams. Another limitation is the fact that SOCKS can only proxy a single incoming TCP connection per request. As previously mentioned, IETF work is underway with the aim of resolving these issues in a next-generation SOCKS protocol (IETF Authenticated Firewall Traversal Working Group, 1999). The fact that SOCKS V5 is used by YAPS for inter-proxy communications also means that the security functionality used by the YAPS client proxy cannot be used to secure communications with, say, publicly-accessible web servers that rely on SSL (i.e. without SOCKS) for secure communications. This is effectively an implementation issue, and could be resolved by implementing the client proxy such that it will provide direct SSL-enabled communications to destination servers that have well-known SSL port numbers.

If advanced access control was required by existing web servers without using SOCKS, a YAPS proxy on the server side could be deployed, which would not use SOCKS V5 at all, but would use HTTP over SSL and only perform AC pull to obtain client ACs.

5.3.3. Reliability and load balancing

The use of a single proxy gateway to server applications introduces both performance and reliability factors.

If a YAPS server proxy was to encounter a failure, the server applications would not be reachable until the problem is rectified. To minimise service disruption, backup mechanisms can be put in place in the event of a server proxy failure. Secondary YAPS servers can be deployed at the appropriate locations, and their services can be invoked by client proxies if the primary server proxy does not respond in a certain amount of time.

Using a single server proxy as a gateway to multiple server applications has an impact on the performance of YAPS, as the single server proxy effectively becomes a performance bottleneck. To alleviate this, YAPS can be distributed to balance the load across multiple servers. This can be done using either of two techniques:

1. *Proxy per server*. This is the most straightforward way to implement load balancing with YAPS. Instead of maintaining a single YAPS server proxy for all server applications, each server application is protected by a dedicated YAPS server proxy, and this distribution is visible to incoming clients. For example, two web servers: *webserv1.cs.may.ie port 80* and *webserv2.cs.may.ie port 80* could be protected by two YAPS proxies residing on *proxy1.cs.may.ie port 80* and *proxy2.cs.may.ie port 80*. Distributing YAPS in this manner eliminates the bottleneck created by a single server proxy for both server applications. The downside to distributing YAPS in this manner means that the amount of configuration administration increases linearly for each server application, since each YAPS server proxy maintains its own security database, ACL and system configuration. In addition, the routing configuration of the client must be updated to route requests to the web servers via the client and server proxies.
2. *Proxy per server with dispatching*. In some cases, a single server application may have its load distributed across multiple physical machines, each running a replicated copy of the server application (i.e. a "web farm"). Distributing YAPS in this manner is a complex task. One cannot simply apply round-robin DNS or some other form of network-layer load balancing solution, as this will not work with session-based protocols such as SSL or with application protocols that maintain state information (i.e. HTTP and cookies). Applying off-the-shelf application-layer distribution products such as Arrowpoint CS-800 (Arrowpoint Communications) will not work as they would require SOCKS compatibility and logic to handle the AC push.

A possible solution in this case is to implement a YAPS "dispatcher" server-side proxy that will select which of the replicated proxies is to deal with a particular request. Once selected, the dispatcher would act as a transport-level gateway passing encrypted traffic back and forth between client and server proxies. In order to keep application-level sessions intact across subsequent connections, the dispatcher can keep a record of application servers and any associated sessions that may be been established. The only application protocol independent session information available to the dispatcher is available from the security layer (i.e. the SSL session ID). This assumes that the transport security protocol is session-based. In addition, distribution based on the content of the application data is not possible since the dispatcher cannot decrypt it. application data.

The provision of a reliable YAPS service follows the same pattern as the dispatching service discussed above. In addition to the relaying of connections between incoming clients and actual YAPS server proxies, the dispatcher could be used to periodically probe all of the server proxies in the farm, and would redirect incoming connections to a working server proxy in the event of a failure.

5.3.4. Resource and operation abstractions

In order to achieve a high degree of application protocol independence, the YAPS ADFs use the notion of abstract resource and operation pairs in both the API exported to the protocol handlers and the server proxy ACL. Whilst abstract resources are handled using URLs (which, by their definition, are protocol independent), the notion of the abstract operation is still an issue that

YAPS has left open. YAPS does not try to make an abstraction of operations, but uses application-protocol specific operations in the ACL such as GET, POST, and PUT (for HTTP), which are compared to the protocol-specific operations passed to the ADFs from the protocol handlers. This requires an intimate knowledge of application protocols by YAPS server proxy ACL administrators. Simple abstractions such as *read* and *write* may not be sufficient as application protocols become more complex, and the abstract operation concept needs more study.

5.3.5. Performance

The fact that communications are diverted and routed through numerous proxy applications means that extra propagation delays are introduced into client–server communications. These propagation delays, in addition with the secure communications and authorisation overhead, contribute to an overall decrease in client–server communications performance. However, the results obtained from our experiments (see also, Dowling, 1999) have shown that this impact is by no means severe. Nonetheless, minimising the performance degradation is an ongoing task, and one can only aim to improve performance via the use of more efficient software techniques, load balancing and accelerated hardware. Whilst YAPS introduces some performance degradation, it does provide much in return for the extra communications overhead. YAPS has been shown to provide secure transport, secure authorisation, fine-grained access control with reduced administration, application-protocol independence and high interoperability potential.

5.3.6. Evaluation summary

The routing issues, in addition to the dependency on SOCKS V5, restricts the use of YAPS to confined environments. Whilst YAPS, in its present form, cannot be used to strengthen communications to publicly-accessible SSL-enabled servers, it is ideally suited to the security needs of an extranet/VPN environment.

However, before being deployed in a real-world extranet environment, further YAPS development is required in the area of use-condition certificate distribution. The current system uses a local use-condition database. A more realistic approach would be to have users publish use-conditions to a directory and to integrate directory access logic into the YAPS server proxy for this purpose. To improve system performance and reliability, a load balancing solution can be put in place.

6. Conclusion and further research

We have designed and implemented the platform-independent YAPS architecture to transparently provide secure transport and authorisation services to existing Internet applications in a way that requires no modification to the applications. We have described YAPS in terms of its architecture and we have discussed its operation in the context of an intranet/extranet environment. In summary, the YAPS project has achieved its goals at the expense of introducing some new issues regarding routing, performance and the proxying of UDP-based applications.

Whilst YAPS uses proxies to provide secure transport and authorisation services, the proxy architecture could be exploited further to provide additional high-level security services. The YAPS proxies could also be applied to the provision of content-filtering services. The protocol handler components could be used at both the client and server sites to filter application-specific data for inappropriate content, application-level attacks, and/or virus infection.

There is scope for further work in the area of trust policy and security configuration. Whilst YAPS uses ACs to representing access privileges in a standardised manner, the YAPS security policy is maintained using custom representations for ACLs, trusted CAs and trusted AAs. A standardised mechanism for representing security policy would provide even greater interoperability benefits. Corporate security policies could be formally specified using an appropriate syntax by the responsible authorities and the correct security configuration could be generated from the policy encoding.

Further work regarding the standardisation of authorisation frameworks is underway in IETF working groups (Farrell et al., 1999). Work is also underway regarding the development of a generic authorisation API for applications (Ryutov and Neuman, 1999; DASCOM). This work will play a positive role in the further development of the YAPS architecture. A standardised authorisation framework will allow the communication and externalisation of access rights between YAPS proxies and third party applications, whilst a generic authorisation API will provide a platform for developers to write YAPS protocol handlers for both standard and proprietary application protocols.

References

Arrowpoint Communications. CS-800 Content Smart Web Switch. http://www.arrowpoint.com/index.html.

Berk, E. JLex: A Lexical Analyzer Generator for Java. Version 1.2.3. http://www.cs.princeton.edu/~appel/modern/java/JLex.

C2Net Software Inc. SafePassage SecureTunnel. http://www.c2net.com.

C2Net Software Inc. SafePassage WebProxy. http://www.c2net.com.

Claessens, J., Vandenwauver, M., Preneel, B., Vandenwalle, J., 1998. Setting up a secure web server and clients on an Intranet. In IEEE: Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford University, CAlifornia, June, pp. 295–300.

Dierks, T., Allen, C., 1999. The TLS protocol version 1.0. IETF Transport Layer Security Working Group RFC 2246, January.

DASCOM. IntraVerse Authorisation API. http://www.dascom.com.

Dowling, A., 1999. A proxy-based security architecture for internet applications. MSc. Thesis (Mode I), National University of Ireland, Maynooth, July.

European Computer Manufacturers Association (ECMA), 1996. Standard ecma-219. Authentication and privilege attribute security application with related key distribution functions – Part 1, 2, and 3, March.

Farrell, S., 1998. TLS extensions to AttributeCertificate based authorization. IETF Transport Layer Security Working Group Internet Draft, August.

Farrell, S., 1999. An internet AttributeCertificate profile for authorization. IETF Public Key Infrastructure (PKIX) Working Group Internet Draft, October.

Ferraiolo, D.F., Kuhn, R., 1992. Role based access controls. In: 15th NIST-NSA National Computer Security Conference, Baltimore, MD, October, pp. 554–563.

Freier, O., Karlton, P., Kocher, P., 1996. The SSL protocol version 3.0. IETF Internet Draft, November.

Farrell, S., Vollbrecht, J., Calhoun, P., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., Spence D., 1999. AAA authorisation requirements. IETF Authentication, Authorisation and Accounting Working Group Internet Draft, October.

Housley, R., Ford, W., Polk, W., Solo, D., 1999. Internet X.509 public key infrastructure certificate and CRL profile. IETF Network Working Group RFC 2459, January.

IETF Authenticated Firewall Traversal Working Group, 1999. http://www.socks.nec.com/mail/aft/index.html.

Institute for applied information processing and communications (IAIK), Graz University of Technology, Austria. Java Security Libraries: JCE v2.5.1 and iSaSilK v2.5. http://jcewww.iaik.tu-graz.ac.at.

ITU Recommendation X.509, 1997 E. Information Technology – Open Systems Interconnection – The Directory – Authentication Framework, June 1997.

Jamison, B. BYACC/JAVA version 0.91.http://www.lincom-asg.com/~rjamison/byacc/.

Johnston, W., Mudumbai, S., Thompson, M., 1998. Authorization and attribute certificates for widely distributed access control. In: IEEE: Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford University, CA, June.

Johl, J., Neumann, C., 1993. The Kerberos network authentication service (V5). IETF Common Authentication Technology Working Group RFC 1510, September.

Kaijser, P., Parker, T., Pinkas, D., 1994. SESAME – the solution to security for open distributed systems. Comput. Commun. 17 (7), 501–518.

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L., 1996. SOCKS protocol version 5. IETF Authenticated Firewall Traversal Working Group RFC 1928, April.

NEC Networking Systems Laboratory. SOCKSCap. http://www.socks.nec.com.

RSA Laboratories, 1999. PKCS 12 v1.0 – Personal Information Exchange Standard, June.

Ryutov, T., Neuman, C., 1999. Generic authorization and access control application program interface C-bindings. IETF Common Authentication Technology Working Group Internet Draft, June.

Tanenbaum, A.S., 1996. Computer Networks, third ed. Prentice-Hall, Englewood Cliffs, NJ.

Yeong, W., Howes, T., Kille, S., 1995. Lightweight directory access protocol. IETF Network Working Group RFC 1777, March.

Mr. Andy Dowling is a lecturer in the Department of Computer Science, National University of Ireland, Maynooth, Ireland. He completed his B.Sc. and M.Sc. degrees in 1998 and 1999, respectively, both in the area of Computer Science. His main interest is in the field of information and application security. Andy has worked on information security projects for several multi-national companies, including Siemens and Baltimore Technologies.

Dr. John Keating is a senior lecturer in the Department of Computer Science, National University of Ireland, Maynooth, Ireland. He completed his B.Sc. and Ph.D. degrees in 1986 and 1990, respectively. He has co-ordinated several funded national and international projects, and has published research papers mainly in the areas of Neural Networks and Information Processing.