A Security Framework for Service Overlay Networks: Access Control

Jinu Kurian
Dept. of Computer Science
University of Texas at Dallas
Richardson, Texas - 75080
Email: jinuk@student.utdallas.edu

Kamil Sarac
Dept. of Computer Science
University of Texas at Dallas
Richardson, Texas - 75080
Email: ksarac@utdallas.edu

***Abstract*—Service overlay networks (SONs) have recently been proposed to support various value-added services including multicast, resilient routing, QoS support, and DoS resistant communication in the Internet. Access control plays an important role for various SON applications yet most SON proposals do not consider access control or assume that it is a pre-existing service. The lack of a proper access control mechanism may introduce security or efficiency problems for various SON applications. In this paper, we present a scalable, distributed access control scheme with very low state information required to be maintained at the SON nodes. Using this service, a SON access node can decide if an end user's traffic should be accepted into the SON overlay for processing and forwarding towards its intended destination. We present our scheme and evaluate it via a combination of formal verification, security analysis, and an experimental evaluation work on its practicality.**

I. Introduction

Service overlay networks (SON) have garnered much interest in the research and industrial community. SONs have been proposed for applications ranging from multicast [1] and QoS support [2] to denial of service (DoS) defense [3], resilient routing [4] etc. Large scale SONs provide a wide array of added services and capabilities to users and service providers. They also help alleviate in a smaller scale some of the more difficult problems like QoS support and DoS defense which are not easy to solve in a global scale.

One of the major reasons for this interest in SONs is their ease of deployment in the Internet. This low incipient overhead allows for the testing and deployment of novel and possibly disruptive applications with minimal modifications into the network core. However, many SON proposals assume that several requisite functionalities including authentication, access control, secure routing, etc., are pre-existent or easily adaptable to the overlay context.

In this paper we consider sender-site access control (SSAC) as an important service in several SON applications. SSAC refers to an access-limiting service at the periphery of a SON overlay and is typically carried out by a SON *access node*. SSAC is necessary for proper SON functionality in many overlay-based applications. It is required in security overlays to filter out unauthorized users (as in SOS [3]) or in QoS overlays to provide network edge packet marking and prioritization (as in [2]). It may also be used in other SON applications including routing overlays and multicast overlays to limit the provided service only to registered users of the service.

SSAC should not be confused with client authentication that is used in many network applications. To illustrate, consider a financial institution, `BankA`, that uses a SON service to provide a DoS resistant communication service for its remote clients, `RClients`. In this context, a `RClient` has an account with `BankA` and has authentication credentials (e.g., a username and a password) to remotely login to `BankA`'s web site to do online transactions. In our work, SSAC is *not* a mechanism to replace this client authentication process. Instead, SSAC provides a sender-site (i.e., at `RClient` site) access control mechanism to decide if the SON overlay should admit and forward `RClient`'s packets toward the `BankA` web server or not. Please note that, without using such a sender-site access control mechanism, a malicious user can send unsolicited packets to the `BankA` web site over the SON overlay. Therefore, our main goal, from the point of view of this example application, is to build a scalable distributed SSAC mechanism which requires minimal state to be maintained at the SON *access node* for the *access node* to decide if a client's traffic should be forwarded on the SON overlay toward its *intended destination*.

There are two simple yet inefficient approaches to sender-site access control: (1) the sender-site overlay node (i.e., *access node*) can maintain a database of its domain-local users (e.g., local `RClients` in the above example) and their credentials or (2) the *access node* can forward the users' requests to the *intended destination* (e.g., `BankA` in the above example) to verify users' credentials and receive a 'grant-access' or 'deny-access' response from the destination [5], [6], [7]. The first approach has the advantage that the destination does not need to be involved on a per-client-per-session basis. However, it lacks mobility support which limits its practical utility. The second approach supports mobility but has all the disadvantages of a centralized solution.

In this paper, we build an access control mechanism that has the advantages of both of these approaches while avoiding their disadvantages. The crux of our solution relies on an access token which contains a new set of credentials (*SSAC credentials*) for the user possessing the token and allows for the verification of these credentials without maintaining per-user state at the SON *access node*. The only information required for a SON *access node* for token verification is its own public/private key pair. Note that *SSAC credentials* may be different from user's authentication credentials that the user may have for authenticating itself to a server at the destination site. We present our protocol and provide a detailed analysis on its accuracy, security strengths, and

the practical operational overhead through a combination of formal verification, security analysis, and an experimental evaluation on a testbed network environment.

The rest of the paper is organized as follows. Section II describes the access control problem that we address in this paper. Section III summarizes the related work in distributed access control. Section IV describes our solution and enumerates the protocol steps. Section V presents the evaluations of the protocol. Finally, Section VI concludes the paper.

II. Problem Description

In this section, we discuss the context of the SSAC problem in more detail and present a more formal definition. Our goal is to avoid possible confusion between the SSAC problem and the traditional client authentication problem. We use a generic SON model to define the context of the SSAC problem. We believe that our model applies to most SON applications.

A SON overlay is typically deployed across multiple ISP domains by a third-party overlay service provider (OSP) [2], [8]. The OSP uses the SON overlay to provide value-added communication services to interested networked application servers (NASs). Typical NASs include web portals, Internet radio/TV stations, and different types of corporations with online presence. As an example, `BankA` may have a web site for its clients to remotely login and do online transactions. If this web site is reachable through unicast only, it may be vulnerable to DoS attacks. To minimize the impact of such an attack on its clients, `BankA` may additionally use a SON overlay to provide an optional DoS resistant communication service for its remote clients. In this context, a remote client, `RClient`, may contact the `BankA` web site through unicast or through SON overlay. During normal operating circumstances, `RClient` may access the `BankA` web site via unicast and during a unicast-based DoS attack on the `BankA` web site, `RClient` can use the SON overlay to communicate with the `BankA` site. The SON based DoS resistant communication service ensures that `RClient` can access the `BankA` site without being affected by the ongoing unicast-based DoS attack [8].

In the above example, `RClient` should have necessary credentials (e.g., username and a password) to authenticate itself to the `BankA` web site. In addition to this, in order to use a SON based secure channel to the `BankA` site, `RClient` needs a new set of credentials that we call *SSAC credentials*. SSAC credentials are needed to satisfy two requirements: (1) *SON authorization* to verify that `RClient` is a legitimate/authorized user of the SON overlay and (2) *NAS access verification* to ensure that `RClient` is an authorized user for a particular NAS server. The former requirement is needed to block malicious users with no SON authorization credentials from accessing the SON overlay at all. The latter requirement is to ensure that the SON overlay provides services to multiple NASs while keeping the set of the remote clients of each NAS server separate from each other. Based on the above discussion about the requirements of the service, our problem statement is as below:

Problem Statement: Our objective is to design a distributed SON authentication and access control scheme where: (1) a user's SON authorization credentials can be verified by any SON *access node*, (2) a user's NAS access credentials can be verified before forwarding user's packets to the corresponding NAS site over SON, (3) NAS access verification procedure should differentiate between users of different NAS servers, (4) both SON authorization and NAS access verification services should be provided without the need to contact the NAS or a centralized authentication server on a per-session basis, and (5) SON *access nodes* should not be required to maintain state for individual users.

III. Related Work

Distributed authentication and access control have been studied extensively as classical security problems. A theory of distributed authentication and the basis of the Public-Key Infrastructure (PKI) systems were established by Lampson et al. [5]. Neuman [6] describes a proxy-based authentication, authorization, and accounting system that uses the concept of restricted proxies. A restricted proxy is a certificate that is bound to a principal and cannot be tampered with. The restricted proxy can be used for authentication and authorization of a principal from a centralized server or a hierarchy of authentication servers. Trostle and Nueman extend the concept of restricted proxies to co-exist with established Kerberos [9] based authorization systems. In both cases the solutions are centralized or partially centralized and hence are vulnerable to DoS attacks. Woo and Lam establish a generalized access control list (GACL) framework [7] to explicitly specify policy requirements and access restrictions of end users. An authentication server maintains the GACL and is contacted by end servers to provide authorization after initial authentication. Maintaining an ACL for every user can be expensive in large applications like SONs and the centralized authentication server is again vulnerable to DoS attacks.

The traditional Kerberos authentication protocol [9] is extended by Sirbu et al. [10] to provide distributed authentication. It achieves this by eliminating the centralized Key Distribution Center (KDC) and having the user contact the end server directly. In this manner, the session ticket can be obtained without the necessity of a centralized server to provide it. This approach is more scalable than the previously discussed approaches. However it requires the involvement of the end server on a per-session basis for all authentication requirements. In our approach we seek to create a framework which will require only a one-time involvement of the end server instead of the per-session involvement. This also allows the SON to provide DoS-resistant communication as a value-added service [8].

RADIUS [11] is a widely used protocol for remote authentication in network devices. It is used primarily in authenticating dial-up and DSL users. It consists of a centralized RADIUS server which is contacted by an access server to provide authentication and authorization to users for access to network resources. DIAMETER [12] is the proposed replacement for RADIUS. It improves on many aspects of the RADIUS protocol including security and reliability. Both approaches

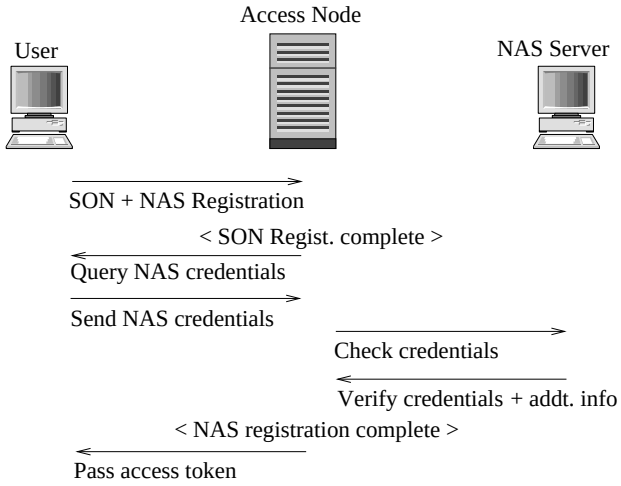


Fig. 1. User registration.

are centralized which can lead to DoS attacks using the authentication process.

IV. The Protocol

A. User Registration with a SON

Each user who plans to use a SON to communicate with a remote NAS server needs to first register with the SON overlay. This is a one-time registration and establishes the user as an authorized user of the SON. Once SON registration is completed, the user's access credentials for its desired NAS server(s) need to be established. Note that the user is required to obtain access credentials for each such remote NAS server.

B. User Registration with a NAS

The goal of NAS registration is to create SSAC credentials for a SON registered user to access a remote NAS server over the SON. A user may have some pre-established relation with a remote NAS (i.e., the user may be a client of the remote NAS) or the user may need SON to facilitate the establishment of such a relation (i.e., the user may need SON to facilitate the establishment of a service contract between itself and the remote NAS). The NAS registration process slightly differs for these two cases as below.

For the former case, the user contacts its proximate SON access node, called *home access node*, to facilitate NAS registration. The user provides a proof of its pre-established relation to the *home access node* such as a secret key which is different from user's password with the NAS site. The *home access node* contacts the NAS server with the necessary identity and secret key information of the user and in return receives an 'grant access' or 'deny' response with some additional information to generate an *access token* for installation into user's machine (see Figure 2). For the latter case, the *home access node* also facilitates the initial service registration request of the user with a remote NAS server. Once the registration is completed, the above process is repeated for obtaining an *access token* for the user.

Generation and Structure of the Access Token:
Access token generation involves the user, the *home access node* and the remote NAS server. Initially, the user passes three pieces of information to the *home access node*. These include '`Domain`', '`Username`', and '`Secret`' corresponding to the NAS server to be contacted, username of the user with this NAS server, and a bit string that serves as a shared secret between the user and the NAS server respectively. The home access node passes these 3-tuple information to the NAS server and receives another 3-tuple response including '`Grant/Deny Access`' prompt, '`Info`', and '`Expr`' fields corresponding to result of the request (access/deny), a possibly empty field specifing additional info for the *home access node*, and an expiration date/time for the token respectively.

Upon arrival of a response with a '`Grant Access`' answer, the *home access node* creates an access token to be stored at the user's machine. The structure of the access token is shown in Figure 2. The '`Access Node Id`' field corresponds to the *home access node* installing this token. The '`Domain`' field corresponds to the NAS server that this token is valid for. '`Encryption Key (EKey)`' is a key that the home access node randomly selects and encrypts with its public key. `EKey` is used by the access node to verify the authenticity of the token later on. The '`Username Hash`' is the hash of the username initially provided by the user and is generated by the home access node using the '`EKey`'. '`Encrypted Secret (ESecret)`' is the '`Secret`' encrypted with '`EKey`' by the home access node. '`Expr`' is the expiration date and time of this token in clear text (Note: the token in its entirety will be sent over an secure channel for transmission, the Expr field is left in the clear since it is a low cost operation for the verifier to accept or reject the token based on its value). '`Info`' field is to optionally include information by the NAS server and/or *home access node*. The seven tuples so far makes up the `DATA` fields of the token. The '`Seal`' is a message digest field computed by the *home access node* over the `DATA` fields using the '`Key`'. The `Seal` is used to verify the authenticity of an access token. '`Certificate`' is the public key of the *home access node* signed by a certification authority (Note: We assume that a SON is controlled by a single OSP, therefore the necessity for the presence of a PKI infrastructure is avoided as a single certification authority will suffice). This field is used by an access node to obtain the public key of the home access node when required. We assume that a SON is controlled by a single OSP, therefore a single certificate authority rather than an PKI infrastructure will be sufficient. Finally, '`Signature`' field is a MAC of all the previous fields except the '`Seal`' field generated by the *home access node* using its private key.

The token has some important properties that make it secure and non-transferable. The '`Domain`' field binds the token to a single domain. Thus, a user cannot reuse a token for other sites it has not registered for. Even if the token is intercepted by an adversary, it is not useful without the '`Username`' and the '`Secret`' between the user and the NAS server. If required, the token can be as an added security mechanism bound to a

Access Node Id	Domain	EKey	Username Hash	ESecret	Expr	Info	Seal	Certificate	Signature

Fig. 2. Structure of the access token

single user machine (for example using a combination of fields like BIOS checksum, MAC address, operating system version, etc.) which ensures that it cannot be distributed to create an attack. Finally, the '`Seal`' field ensures that the token has not been altered in any manner by the user or an attacker. Since the '`Seal`' includes the '`Expiry`' and '`Access Node Id`' fields as part of the hash, the user cannot modify the token to extend its validity.

Security of NAS Registration Step:

During the NAS registration process, a *home access node* is required to contact the remote NAS server sites to verify the authenticity of user-provided information. This step opens up a possibility of DoS attacks on the remote NAS server and therefore possible vulnerabilities need to be carefully examined. The attacker can try IP address spoofing to generate a large number of NAS registration request with the goal of flooding a NAS server. The attacker can also try using bots distributed across multiple domains to launch an attack against the NAS server. The NAS registration requests are forwarded on the SON overlay and the attacker needs to have authentication credentials to use the SON overlay. The SON authentication process can additionally be enhanced with turing tests to makes it difficult for the attacker to send NAS registration request floods over the SON. To limit the effect of these registration attacks, the remote NAS server and the *access nodes* can also be configured to give priority to NAS registered users over new user registration requests. This allows the NAS server to serve its established users even under an access request flooding attack.

C. User Authentication and Access Verification

In this section, we discuss the authentication process that we use to verify access credentials of a user before forwarding its traffic to the corresponding NAS server over the SON overlay. This process involves an *access node* to retrieve the access token and use it to authenticate the user carrying the token.

This process differs slightly depending on user's current location. If the user resides in his/her home domain, then we assume that the *access node* is the *home access node* who assisted the user with the collection of the token during user registration phase. If the user resides at some remote domain, then we assume that the *access node* is different from the *home access node*. We consider each case in turn. For both cases, the first step involves the user sending its access request for a remote NAS server along with its access token for this NAS server (the token can be installed as a cookie so that the browser fetches it automatically) to a local *access node*. When the user is at his home domain, i.e., the *access node* is the *home access node*, the *access node* receives the access token and performs several checks on the access token. Except for the '`Access Node Id`' check, failure in any of the other checks results in an authentication failure. The procedure is discussed below:

1) Collect the access token from the user
2) Check if `Access Node Id` is for itself
3) Check if the NAS server in request matches the `Domain` in the access token and check if the user is listed in a locally maintained black list for the `Domain`
4) Check for `Expiry` and recompute and compare the `Seal` value on the DATA fields of the access token
5) Prompt user for `Username` and `Secret` and compare them with their counterparts in the access token
6) If all checks succeed, install a session cookie on the user's machine for further access during this session.

When the user is at a remote domain, i.e., the *access node* is not the *home access node*, the local *access node* at the current domain first receives the access token from the user and verifies the expiry field in the token. It then uses the public key of the *home access node* in the `Certificate` field to verify the `Signature` in the access token. This ensures that the access token is not forged by the user. After this step, the *access node* forwards the access token to *home access node* for verification and the process follows as described above.

In both cases if user completes all these checks, the access token is deemed valid and the user presenting the access token is considered to be an authorized user of the SON based service with access permissions to the NAS domains requested in the access token. In both cases a temporary session cookie is created by the *access node* for the user which is retrieved and verified for further access during the session. The user's traffic is then routed through the SON overlay network towards the remote NAS server.

Protocol Details:

Figure 3 presents the details of the overall protocol message exchanges during this process. In the protocol in Figure 3, `A` is the user, `S` is the *home access node* and `B` is the NAS server requested by the user. `n1`, `n2`, and `n3` are nonces arbitrarily selected by the *home access node* to prevent replay attacks. `SKey(A)` is a shared key between `S` and `A` and `SKey(B)` is a shared key between `S` and `B`. `SKey(A)` can be created as a derivative of the `Secret` value. Note that `A` should know the `Secret` value prior to contacting `B`. After `A` passes the access token to `B` in step 0, `B` can extract this information from the access token and use it to compute `SKey(A)`. `SKey(B)` can be obtained by using authenticated Diffie-Hellman key exchange protocol between `S` and `B` over the SON overlay. `SKey(B)` is assumed to be a long term key (e.g., with a life time of a week or so) between `S` and `B`.

In Figure 3, at the end of step 1, the *home access node* has all the required information to authenticate the user and verify its access credentials to the NAS server. The remaining steps in the protocol is for the *access node* to create and distribute a shared session key `kab` for the user and the NAS server to use for secure communication over the SON overlay. In addition,

0 A → S : {Access token + NAS Domain}
1 A → S : SKey(A) {A, S, B, n1, Secret}
2 S → A : SKey(A) {S, A, B, n1, n3, kab}
3 S → B : SKey(B) {S, B, A, n2, n3, kab}
4 A → B : kab {A, B, S, n3}
5 B → A : kab {B, A, n3}

Fig. 3. Authentication and access control protocol.

step 2 indirectly helps authenticate the *home access node* to the user. Finally, for session maintenance between the user and the *access node*, the *access node* creates and installs a session cookie into the user's machine and this session cookie is piggybacked at each message sent from the user to the *access node*. The session cookies used are standard web cookies with an explicit expiration time [13]. These cookies are resistant to forging and manipulation and when combined with a nonce are resistant to replay and man-in-the-middle attacks. A more detailed analysis of the security and performance implications of using these cookies can be found in [13].

V. Security analysis

A. Formal verification

In this section we use formal verification mechanisms to evaluate the correctness of our proposed protocol. Specifically, we model the interactions between the three participating entities as communications between a set of sequential processes. This allows us to verify that the shared secrets (Secret, nonce, session keys) are shared only between the participants of the protocol and also that the user and the NAS server are authenticated to each other at the end of the protocol run.

The language of Communicating Sequential Processes (CSP) [14] can be used to describe any system with agents that communicate by passing messages between each other. Among other applications such as software engineering, parallel processing and distributed systems, CSP allows for the formal description of entities in an authentication protocol and the messages used in the protocol. CSP has been used extensively in the analysis of security protocols including the breaking and fixing of the Needham-Schroeder protocol [15], analyzing the TMN protocol [16], RADIUS [17] and many others [18].

Since developing and analyzing CSP specifications manually is a tedious and error prone process, over the years various tools have been developed to ease both these processes. Casper [19] is one of the tools available to derive CSP definitions of authentication protocols. It accepts a description of an authentication protocol in a syntax similar to the widely used security protocol notation and converts it to a CSP description. This CSP description can then be analyzed using the Failures Divergence Refinement (FDR) [20] model checker. FDR accepts protocols described in CSP and checks to see if the implementation is a *refinement* of the specification, i.e., if the properties described in the protocol specification are met by the implementation.

To evaluate the correctness of our authentication protocol, we modeled the protocol in CSP using a Casper derivation (see [21]). The CSP model was then examined using the FDR2 model checker to verify if the protocol implementation refines the specification. We give a brief overview of some of the important features in the modeling and evaluation of the protocol below.

The first requirement for Casper is a set of specifications. These specifications are the properties that the protocol is required to maintain at the end of its run. In our protocol, the final result of the protocol run should be that A has successfully authenticated itself to B and vice-versa. Additionally, the protocol also needs to ensure that the session key kab and the seed nonce n3 are both kept secret and shared only between A, B and S. These requirements necessitate six specification statements as below:

1) Secret(B,n3,[S,A])
2) Secret(A,n3,[S,B])
3) Secret(A,kab,[B,S])
4) Secret(B,kab,[A,S])
5) Agreement(B,A,[n3,kab])
6) Agreement(A,B,[n3,kab])

Secret(B,n3,[S,A]) specifies that B believes at protocol completion that the value of n3 is a secret shared between itself, S and A only. Agreement(B,A,[n3,kab]) specifies that B has authenticated itself to A and they have agreed on the values of n3 and kab. These six statements completely specify all the desired requirements of the protocol.

The next requirement is to model the intruder and the knowledge it possesses. From an intruder I's perspective, she would know the participants in one of the protocol runs, i.e. it knows A the user, B the NAS server and S the *access node*. Additionally, she might also be an authorized user in the system, i.e. she shares a secret key with the *access node* S. We assume that A's secret and her shared key SKey(A) are unknown to intruder I. Additionally, SKey(B) is also unknown to I. The statements below formally specifies these requirements.

Intruder = I
IntruderKnowledge = {A,B,I,S,SKey(I)}

This indicates that the intruder I has a knowledge of all three agents in the protocol, the *access node* S and the two agents A and B. Additionally it can act as an authorized user of the system due to its possession of a shared key SKey(I) with the *access node* S.

The next requirement is to specify working systems to validate the protocol specification. We consider six different systems to validate the protocol specification (see [21]). These systems represent actual protocol runs between the different agents. It also specifies the knowledge that each agent possesses at the start of the protocol run. For example, a basic system would have A acting as an initiator, S acting as a *access node* and B acting as the responder. In this case A (or the attacker) is allowed only to act as either the user or the NAS server or the *access node*. A more practical system

is one in which the attacker can act both as initiator and responder in a single protocol run, or it can have multiple sessions concurrently or sequentially. The specification below shows an example of such a system.

```
INITIATOR(A,S,n1,Secret)
INITIATOR(A,S,n1,Secret)
RESPONDER(B,S,n2)
SERVER(S,kab,n2,n3)
```

In this case the intruder `I` can initiate multiple sessions with the *access node* `S`, or respond to a request from `A` masquerading as the *access node*, masquerade as `S` and initiate sessions with `B` or pretend to be `B` and respond to messages from `S`. We consider six such systems with different combinations of agents acting as initiator or responder. Note that there are infinitely many such systems that can be created with different combinations of agents. However, it is accepted in the formal verification community [19] that these six systems will find nearly all attacks.

We evaluate the correctness of our protocol for each of the six systems and for all six specifications using FDR. However, for the sixth specification, `Agreement(A,B,[n3,kab])`, FDR returns a possible attack for the system. The sequence of events can be found in [21]. In this attack the intruder masquerades all three agents `A`, `S` and `B`, and intercepts the messages sent from each side to forward to the other side. However this attack can be prevented if `S` and `B` maintain a database of previously used nonce values. Additionally, none of the secrets are revealed to the intruder at any stage. Hence, this attack is not useful to the intruder. All the other systems check correctly for all specifications.

As can be seen from the above discussion, the protocol implementation is verified to be secure against almost all known attacks in ensuring the secrecy of all desired values and allows for the authentication of all the three participating entities to each other at the end of the protocol run. In the next section we take a different approach to analyze the security of the protocol.

B. Threat Analysis

In this section we further analyze the ability of the system to handle various known attacks.

Common attacks: Common attacks include eavesdropping on the session establishment, recording the data transferred for replay attacks and man-in-the-middle attacks. These types of attacks are checked for and verified as impossible by the formal verification described in the previous section.

Attacks on access token and session cookie distribution: As we discussed in Section IV-C, the protocol relies on an access token stored in the user's machine for user access and a session cookie for session maintenance. If an attacker compromises a legitimate user's system or if a legitimate user is malicious, he can distribute the access token across several machines to launch a distributed attack. However, the token (because of the `Access Node Id` field) is valid only with a single or a small set of *access nodes* within a single domain ensuring that it cannot be distributed across multiple domains. Additionally, each *access node* can impose a restriction on the number of sessions allowed with a single access token. Attacks are however possible using a distributed token and the remote access procedure. We will discuss these later in the section. The cookie provided after protocol completion is similarly restricted to a single domain and the number of sessions made with a single cookie to a NAS can be limited at the *access node* itself.

Dictionary attacks: In order to gain information about the keys used by a *home access node* (for creating the access token), the attacker can attempt to create a dictionary of `(plaintext,ciphertext)` pairs by registering many times with different NAS servers. In this manner it obtains several tokens with the `EKey` field for its database. However the plaintext value of the `Key` is randomly chosen by the *access node* and is not an easily recognizable value for the attacker. Since all the other fields in the token are encrypted using the `Key` in the `EKey` field, it is difficult for the attacker to obtain sufficient values to create a database.

Compromised overlay nodes and server database reading attacks: Unlike a typical KDC, compromising a single overlay node does not provide the attacker with a complete knowledge of the system. Since no user information is stored in any overlay node, the server database reading attack exposes limited information. The attacker will have to compromise multiple overlay nodes to obtain their keys. Compromised overlays can however selectively drop packets, admit unauthorized users, sniff traffic or cooperate with other compromised nodes to flood a victim NAS server. Such attacks are generally more difficult to defend against and will be part of our future work.

Blacklist poisoning: Each *access node* is required to maintain a blacklist of users whose authentication tokens have been revoked. This blacklist may also be updated if the user fails multiple login attempts. An attacker can try to blacklist legitimate users with numerous failed login attempts. However, only *after* the user presents a valid token and fails the authentication test multiple times will it be blacklisted. The attacker will not be able to attempt its logins without possessing the user's token which by itself can be encrypted during transmission to prevent the attacker from recording it during session establishment.

DoS attacks through remote access: Remote access in the system requires that the user's access request be forwarded to its *home access node* for verification. If the attacker distributes a legitimate token to multiple remote domains, it can force a DoS attack through the overlay itself using the verification procedure. This can be prevented by restricting the distribution of the token to a large number of hosts. For example, by tying the token to specific system by a combination of MAC address, BIOS checksum, operating system version and similar fields the distribution of the token can be limited. This cannot however completely defend against attacks in which the attacker has a large number of such tokens. Note that this is an attack on a *home access node* generating an access token for a user. Such an attack would cause service degradation for the users sharing the attacked *home access node*. Therefore, the impact of the attack is localized and the user causing this

attack (i.e., the owner of the access token) can be identified for corrective actions.

DoS attacks through user verification process: We discussed these attacks in Section IV-C under security of the pre-registration step. Hence we do not repeat the discussion here.

DoS attacks on overlay nodes: An attacker can attempt to bring down the security framework by launching DoS attacks against overlay nodes themselves. For the attack to succeed, the attacker will have to target a large number of overlay nodes. If we assume an attacker with limited resources, this severely limits the attacker capabilities in launching an attack to bring down the entire network. Additionally, overlay nodes can be configured to receive traffic only from its neighboring overlay nodes [8].

C. Experimental Evaluations

1) Motivation: In this section, we present our experimental evaluation results on the overhead of the presented SSAC protocol. An authoritative evaluation of the protocol overhead is difficult due to the multitude of deployment combinations possible for various factors including the processing power of NAS servers or *access nodes*, and number and location of SON overlay nodes among others. Therefore, in our experiments, we have chosen to evaluate one example scenario that we believe is suitably representative of an application utilizing the SSAC protocol. We consider only flooding based DoS attacks in our experiments. The resistance of the protocol to other attacks like intruders have been covered in previous sections.

The SON application that we consider is DoS resistant communication service as in the Secure Overlay Services (SOS) [3] study. In this context, we use SON *access nodes* to perform a remote and distributed authentication service (SSAC) and compare the overhead of this service to a server site user authentication service (SSAS).

2) Experimental Setup: Our experimental setup consists of three Cisco 3600 routers, 12 Linux PCs and a Sunfire V880 server. Using this equipment, we built a network configuration as presented in Figure 4. In this configuration, Linux PCs function as end users and/or SON *access nodes* and Sunfire V880 server functions as a high end NAS server running an Apache2 web server.

The experiments are conducted from the perspective of a set of users attempting to establish a secure session with the NAS server. These users (henceforth referred to as legitimate users) generate a constant 100 requests/sec traffic for all the experiments. Their performance is then measured under varying levels of additional user requests.

In the experiments, we implemented the operation of our SSAC protocol in C++ at *access nodes*. After the access verification is completed, users use httperf [22] to generate HTTP GET requests to the Apache2 web server. For SSAS, we use SSL-enabled httperf at the user site and use `mod_ssl` module with OpenSSL in the Apache2 web server to implement user authentication service at the NAS server site. We also ensure that we use the same crypto suite (i.e., RSA, DES, and HMAC) for the implementation of both cases.

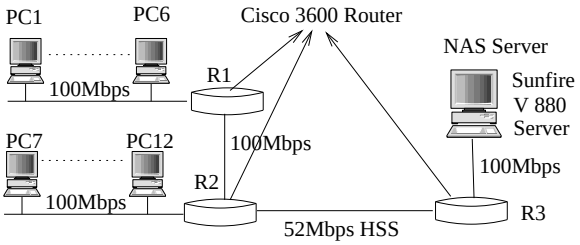


Fig. 4. Evaluation topology.

3) Metrics: We evaluate the SSAC protocol by comparing its overhead with that of the SSAS service with SSL support operating under similar conditions. The metrics used are:

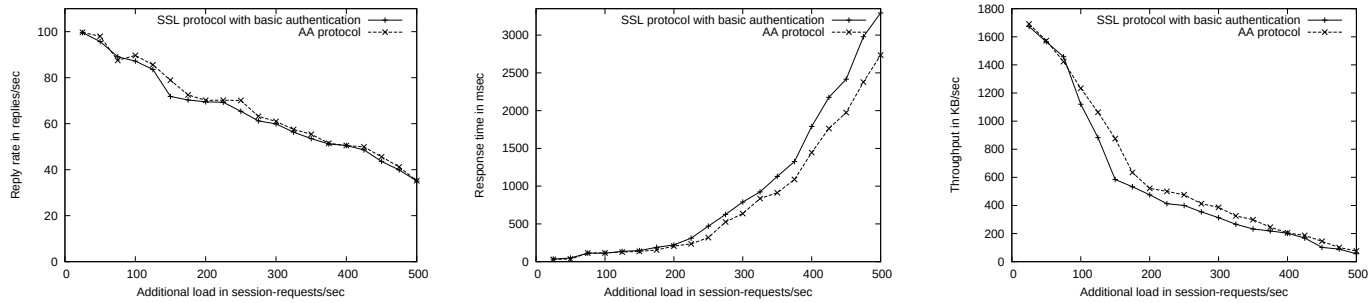
(a) Average reply rate: Average reply rate refers to the average rate of replies over the entire experiment to HTTP GET requests from the legitimate users. This metric is a direct indicator of the processing overhead introduced by the respective user authentication protocols.

(b) Average response time: Average response time refers to the average time taken by an HTTP GET request to be processed by the web server. This is an important metric as it directly reflects the user-perceived overhead of the protocol. If the response times are too high, they are not suitable for practical deployment.

(c) Average throughput: Average throughput refers to number of bytes per second delivered to all legitimate client applications. Throughput is a direct indicator of the overhead to the applications using these protocols for adequately serving user requirements.

4) Experimental Details: For the SSAS case, we use httperf to generate a total of 100 requests/sec from 100 legitimate users and a varying number of additional users (from 25 to 500) each generating one service request/sec to introduce additional load on the web server. Each session starts with an SSL session establishment phase and a two-message username/password authentication phase after the session is established. The user then issues an HTTP GET request to fetch a page from the web server. During this transaction, we measure the average response time (from the start of the SSL session establishment to the end of page delivery), the average number of replies/sec to session requests, and the average throughput for the legitimate users.

For the SSAC case, we use four Linux PCs as SON *access nodes* and eight Linux PCs as legitimate users. In this case, *access nodes* help users to perform SSAC based remote authentication. The legitimate user request rate is fixed at 100 requests/sec through the first *access node*. The additional load request traffic is uniformly divided among the other three *access nodes* so that each of them receives a maximum of 168 requests/sec. In the simulations, we assume that all the requests arriving at the *access nodes* are legitimate and hence approved. Once the authentication protocol is completed, httperf is used to generate HTTP GET requests. The overall average response time (from the start of the authentication protocol), average number of replies/sec and average throughput of the legitimate users are measured.



(a) Avg reply rate (b) Avg response time in msec. (c) Average session throughput in KB/sec

Fig. 5. Experimental evaluation results on protocol overhead.

In both cases above, each experiment run (i.e. with a specific additional load request rate) is conducted for a duration of 15 minutes and each experiment is repeated three times. The averages of all three experiments are presented in Figure 5.

5) Results: Figure 5(a) shows the average reply rate for 100 requests/sec from the legitimate users. According to the figures, both the proposed SSAC approach and the SSAS approach degrade as the load introduced by the user increases. We observe similar results in the average response time (Figure 5(b)) and average throughput (Figure 5(c)). It is seen that in all the cases the proposed SSAC protocol provides a performance that is comparable to that of SSAS approach.

The results of our experiments suggest that the proposed SSAC protocol can be deployed and used for practical SON applications in the Internet. The overhead introduced by our protocol is comparable with the existing SSAS protocols under normal operational environment. On the other hand, in the case of an attack scenario on the NAS server, the overhead of the SSAS protocol will increase to a level to cause a DoS for all remote legitimate users of the NAS server. The proposed SSAC approach however will maintain its load level due to its distributed and relatively stateless nature. We note that our experiments are of a small scale and are merely indicative and not a complete evaluation of the protocol. Real life attacks will be of a much higher order of magnitude, but comparably a real life implementation of the architecture would have a much larger number of overlay nodes.

VI. Conclusion

In this paper we have studied distributed access control problem in service overlay networks. We have presented a scalable and distributed solution to the problem with minimal state dissemination to the overlay nodes required. We have examined the correctness of our solution using formal verification methods; presented a detailed analysis on its security features; and presented experimental evaluation results to show its operational overhead as it is perceived by end users. Our evaluation results indicate that our solution provides an effective and efficient mechanism to solve the considered distributed access control problem for SON overlays.

References

[1] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, Santa Clara, CA, USA, June 2000.

[2] Z. Duan, Z.-L. Zhang, and Y. T. Hou, "Service overlay networks: SLAs, QoS, and bandwidth provisioning," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 870 – 883, December 2003.

[3] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: An architecture for mitigating DDoS attacks," *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Service Overlay Networks*, vol. 22, no. 1, pp. 176– 188, January 2004.

[4] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of 18th ACM SOSP*, Banff, Canada, October 2001.

[5] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 285–310, November 1992.

[6] J. T. Trostle and B. C. Neuman, "A flexible distributed authorization protocol," in *Proceedings of the Symposium on Network and Distributed System Security*, Reston, VA, USA, April 1996.

[7] T. Woo and S. Lam, "A framework for authorization in distributed systems," in *Proceedings of Conference on Computer and Communications Security*, Fairfax, VA, USA, May 1993.

[8] J. Kurian and K. Sarac, "Provider provisioned overlay networks and their utility in dos defense," in *Proceedings of IEEE Globecomm*, Washington D.C, USA, November 2007.

[9] C. Neuman and T. Tso, "Kerberos: An authentication service for computer networks," *IEEE Communications*, vol. 32, no. 9, pp. 33–38, September 1994.

[10] M. A. Sirbu and J. C.-I. Chuang, "Distributed authentication in kerberos using public key cryptography," in *Symposium on Network and Distributed System Security*, San Diego, CA, USA, February 1997.

[11] C. Rigney, S. Willens, and A. R. andW. Simpson, "Remote authentication dial in user service (radius)," June 2000, iETF RFC 2865.

[12] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter base protocol," September 2003, IETF RFC 3588.

[13] K. Fu, E. Sit, K. Smith, and N. Feamster, "Do's and Dont's of client authentication on the web," in *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, USA, August 2001.

[14] T. Hoare, *Communicating Sequential Processes*. Prentice Hall, 2004.

[15] G. Love, "Breaking and fixing the needham-schroeder public-key protocol using csp and fdr," *Lecture Notes in Computer Science*, vol. 1055, pp. 147–166, 1996.

[16] G. Lowe and A. Roscoe, "Using csp to detect errors in the tmn protocol," *IEEE transactions on Software Engineering*, vol. 23, 1997.

[17] I.-G. Kim and J.-Y. Choi, "Model checking of radius protocol in wireless networks," *IEICE Transactions on Communications*, pp. 397–408.

[18] S. Schneider, "Verifying security protocols: an application of csp," in *Proceedings of 25 Years of CSP*, London, England, July 2004.

[19] P. Ryan and S. Schneider, *Modelling and Analysis of Security Protocols*, 1st ed. Addison-Wesley, August 2001.

[20] Formal Systems (Europe) Ltd., "Fdr2 user manual," available at http://www.fsel.com/documentation/fdr2/html/index.html.

[21] J. Kurian and K. Sarac, "A security framework for service overlay networks: Access control," University of Texas at Dallas, Tech. Rep., January 2008.

[22] HP Labs, http://www.hpl.hp.com/research/linux/httperf/.