Real-Time Protocol Monitoring

The key idea behind our approach is monitoring the behavior of protocols, in real-time and long-term, to ensure their secure and reliable progression. Real-time monitoring is achieved through observing various parameter values and their temporal and spatial validity. For long term monitoring, we apply knowledge-based decision techniques such as data mining on data collected over a large number of sessions to detect anomalous events. Further predictive techniques will be used to expose hidden attack profiles.

**Protocol layers**

**MAC**

**Network**

**Transport**

**KDS**

 AD

Frames

 AD

Packets

RTPM

RTPM

 AD

Segments

RTPM

RTPM – Real Time Protocol Monitoring, AD – Anomaly Detection, KDS – Knowledge-based Decision System

Figure 2: Protocol Monitoring Framework

RTPM will be performed at various network layers as the protocols are designed with layer specific attributes. The results from a layer will be continuously communicated to the other layers for proper coordination and preemption of future attacks. Note that while RTPM tries to monitor protocols based on finer details of execution, KDS tries to capture a high level profile of the nodes. These approaches will complement each other in detecting security vulnerabilities at the system level.

During the run of a protocol, several pieces of information such as identities, one-time random values (nonce), time-stamps, and shared keys are exchanged between the two communicating parties. If the parties calculate an integrity check value (ICV) over these values using a pre-defined and mutually agreed upon function, the results would be identical at the two ends. The function must be irreversible, so that it is infeasible to generate a set of protocol messages that would result in that ICV. It should also be collision resistant such that different runs of the protocol must not result in the same ICV. However, there may be cases where the external intruding entity may be able to see all the pieces of information seen by legitimate participants. In such cases, simply demonstrating an ICV is not sufficient because the intruder also will be able to generate this value. This is the reason why signed messages are used in the monitoring channel to deliver validation data. The strong cryptography assumption based on the LSA architecture described previously ensures that an intruder will not be able to forge the signature of another entity.

## Meta-Authentication Framework

Based on the RTPM concept, we have developed a Meta-Authentication Framework. The term meta-authentication denotes an `encapsulating authentication' mechanism for general authentication protocols. This is achieved through a high level mechanism for validating the execution of underlying authentication protocols. Meta authentication operates in the context of a framework comprising an architecture and a high level validation protocol that together provide a distributed environment for monitoring and validating the execution of authentication protocols. Entities involved in an authentication session can ensure that the execution of the authentication protocol itself has been proper and devoid of any malicious tampering or manipulation. The framework does not make any assumptions on the underlying authentication protocol and is generic enough to support any protocol chosen by communicating entities. It is important to note that it is not a goal of meta-authentication to offer any assurance as to the correctness of beliefs established by the `encapsulated' protocol; this is still the responsibility of the encapsulated protocol. However, it does provide assurance that the encapsulated protocol itself runs correctly, protected from extraneous interference.

### The Trust Model

Meta authentication is based on the concept of trusted third parties. All communicating entities in a domain of trust utilize the services of the trusted entity. The use of trusted entities is not new in authentication; several authentication protocols that depend on trusted third parties have been proposed and used in the past (a familiar example is the Kerberos authentication system [10]). However, in almost all such protocols, the trusted third party is an integral part of the authentication process. Any weakness of the trusted party can seriously damage the security of numerous entities depending on its service. Moreover, in large systems, the trusted entity can quickly become a performance bottleneck, as it needs to be directly involved in all sessions initiated among the entities in its authentication domain. Inter-domain authentication (between two entities belonging to two different domains) is also a major issue in such protocols, because of the potentially limited trust that entities in one domain may be willing to have on the trusted entity belonging to another domain.

In the scheme proposed here, the role of trusted third parties is limited to only helping to monitor the execution of authentication sessions. They are neither directly involved in the execution of the protocol, nor do they play any role in establishing beliefs between communicating parties. Their service is needed only if the communicating entities wish to protect their authentication process through meta-authentication. Even without meta-authentication, they will still be able to operate any authentication protocol normally. The optional use of meta-authentication is designed to protect against security attacks, while incurring minimal overhead.

In meta-authentication, the entire user space is divided into trust domains. Each domain includes any number of ordinary communicating entities and a trusted meta-authentication server (henceforth called the meta server). A meta server establishes trust with every member in its domain, so as to function as an intermediary in intra-domain authentication. An entity need not trust any member even in its own domain, other than its meta server. Moreover, entities in one domain need not trust the meta server in another domain. However, meta-servers in different domains may establish and maintain trust on one another, so that their services can be extended to inter-domain authentication as well. As there is only one meta server in each domain, this trust is far easier to establish and manage as compared to maintaining trust among all entities in all domains, or between entities in one domain and meta servers in other domains, or even between all entities in the same domain. Essentially, meta authentication follows a hierarchical and transitive trust model. This means that, if there is trust between entity A and its meta server SA, between entity B (possibly in another domain) and its meta server SB, and between meta servers SA and SB, then eventually trust may be established between A and B.

### An Architecture for Meta-Authentication

The meta authentication scheme uses public key cryptography to protect the integrity of sessions. During an authentication session, communicating members and the concerned meta servers exchange monitoring messages signed with their private keys. These signed messages, verifiable only with the respective public keys, deliver validating data to help the communicating members ascertain the integrity of the `encapsulated' run of protocol. It may however be noted that meta authentication exchanges are not confidential, i.e., exchanged messages are observable by anyone. This is because meta authentication relies only on the integrity of validation messages, not on their confidentiality. When two members of the same domain authenticate, this message exchange involves those two members and the domain's meta server. When the authentication is between two members belonging to different domains, the exchange involves the two members and the meta servers of both domains. As compared to other public key based systems, the meta authentication scheme imposes only minimal requirements, which are as follows.

1. Each member and meta server in any domain has a public/private key pair.
2. The private key of every member and meta server is kept strictly confidential, known only to the holder of the key (this is a requirement in all public key cryptographic systems).
3. All members in a domain know the public key of the meta server of the same domain.
4. A meta server knows the public keys of all members in its domain.
5. A meta server either knows the public keys of the meta servers in all other domains, or has access to a mechanism through which such keys can be obtained (for example, based on certificates issued by a higher trusted entity).

The scheme, however, does not impose any restrictions on the members. A member need not know the public key of any other member in its own, or another, domain. Further, a member need not know the public keys of any meta server in other domains. Similarly, a meta server does not need to know the public key of any member of other domains.

Thus, the scheme allows each domain to be separately administered. The only inter-domain knowledge is limited to meta servers who need to know the public keys of one another. This not only reduces the overhead but also reduces the chance of compromised key pairs being used. The basic schemes as would be used in intra-domain and inter-domain meta authentication are shown in Figure 4 (a) and Figure 3 (b), respectively.

In the figure 4, the public and private keys of any entity X are represented as Kx\_pub and Kx\_prv, respectively. The authentication protocol employed by the communicating members A and B is independent of the meta authentication scheme. Members may use any authentication protocol they choose. It is also irrelevant whether the encapsulated protocols are based on public key or shared key cryptography

 

**(b)**

**(a)**

Figure 4. Inter and Intra-domain Meta Authentication

### Protocol for Meta-Authentication

During the run of an authentication protocol, several pieces of information are exchanged between the two communicating entities such as identities, one-time random values (nonces), time-stamps, and shared keys. If the two entities calculate an integrity check value (ICV) over these values using a pre-defined and mutually agreed upon function, the results would be identical at the two ends. The function must be irreversible, so that it is infeasible to generate a set of protocol messages that would result in that ICV and it should be collision resistant, meaning, different runs of the protocol must not result in the same ICV. It is also possible to calculate the ICV using weighted functions resulting in different values at the two ends, which prevents replay type attacks on the meta authentication itself. By virtue of encryption mechanisms used in the authentication protocol itself, many of these values may be seen only by the communicating parties and not by any third party. Thus it will be infeasible for an external entity to deduce the ICV for a fresh instance of the protocol run. This indicates that in the presence of an attack, one or more pieces of information known (sent or received) to one entity in the session may not be the same as those seen by the entity at the other end. However, there may be cases where the external intruding entity may be able to see all the pieces of information seen by legitimate participants. In such cases, simply demonstrating an ICV is not sufficient because the intruder also will be able to generate this value. This is the reason why signed messages are used in the meta authentication framework to deliver validation data. The strong cryptography assumption ensures that an intruder will not be able to forge the signature of another entity.

### Security of Meta-Authentication

Consider a high level view of the transfer of validating information in the Meta authentication model. Take *x* and *y* as the integrity checkvalues generated locally by nodes A and B. Also *y’* is the value deduced by A and *x’* is the value deduced by B. Now the entity A accepts authentication protocol run if and only if *x’ =* *x*, and entity B accepts the authentication protocol run if and only if *y’ = y*. For given *x* or *y*, it is infeasible to generate a set of protocol messages that result in the same ICV values, because of the properties of *collision resistance* and *irreversibility*, processed by the ICV generating function.

 

Figure 6. Trusted Paths in Meta-Authentication

Under the given assumptions, a malicious entity cannot *forge* any of the signed messages, because the required private key will not be known to it. This follows from the fact the strong cryptography assumption. An intruder cannot *replay* any of the messages in an authentication session, because *x* and *y* are time dependent since their calculation uses time-stamps as one of the inputs. It cannot substitute a message in an authentication session with a similar message from a different run of the protocol, because it is infeasible to manipulate a protocol run such that it results in a known ICV (*x* or *y*). Therefore, if a message is delivered in an authentication session, the receiver is assured of the *data origin authenticity* and *data integrity.* However, a malicious entity capable of observing and manipulating protocol sessions may be able to intercept and possibly delete a message in an authentication session. However, the receiver will detect the message loss through a timeout and abort the session. From these observations, it is clear that an encapsulated (monitored) authentication protocol is secure (in terms of integrity) if the Meta channel between the authenticating entities is safe. Thus, the Meta authentication scheme provides a robust mechanism to ensure the integrity of authentication protocols.