Research Directions

CPAL - Protocol Definition and Evaluation

An obvious way for a Legion programmer to ensure the privacy, integrity, and delivery of messages to or from remote objects is to utilize cryptographic protocols. While many Legion programmers will opt for well-known protocols from the literature, others may wish to construct express protocols of their own. Designing and implementing cryptographic protocols has proven to be difficult for experts, and we could expect Legion programmers to make mistakes as well. For this reason, we feel it is necessary to provide those who will be creating Legion objects with a suite of tools that assists them in designing the security mechanisms for their objects, examining the mechanism for weaknesses, and automatically translating the design into usable Legion code.

Towards this end, we are developing the Cryptographic Protocol Analysis Language and Evaluation System (CPAL-ES), an automated tool that is intended to assist designers in evaluating and implementing their protocols. CPAL-ES searches for both active and passive attacks on a protocol, alerts the user of any weaknesses it finds, and will translate the protocol into valid Legion code at the request of the user.

For more information see:

- Alec Yasinsac "A Formal Semantics for Evaluating Cryptographic Protocols," PhD Dissertation, University of Virginia, 1995.

Distributed Key Generation

In a public key cryptographic system, the authenticity of the keys are essential to the success of the system. Traditionally, a single, centralized key distribution/certification server is used to authenticate public keys. This approach requires a distinguished entity which could potentially become a point of failure or penetration in a distributed environment. What is needed is a distributed key generation technique; a scheme that ensures the uniqueness and authenticity of the keys without the help of a single centralized trusted server.

Suppose we want a resulting n-bit public key for object A, a unique n/2-bit number M is assigned to each key generator. In Legion, this n/2-bit number contains A’s class ID, and some number that uniquely identifies A. Our key generation process works in such a way that the lower n/2-bits of the resulting n-bit key will be the assigned M. The resulting public key is used as A’s object ID. In this way, the key generation process is completely distributed. No public file is required to store the public keys since the object IDs are their public keys. An agent needs only know the ID of his/her communication partner to communicate with others.

For more information, see the white paper on this topic.

Formal Specification and Analysis

Computer security is the protection of the resources of a computer against events that can result in loss of availability, unauthorized access, or modification of data. The techniques devised to protect computers from these attacks are general purpose in nature and therefore cannot directly enforce security that has no universal definition. The high degree of assurance in security properties of systems used in security-critical areas such as military and commerce is usually achieved by verification. In this thesis we develop a framework for carrying out this verification in a formal and systematic manner.

Security verification in information systems consists of - specification of a security policy, specification of the system that must enforce the policy, and verification of policy against system specification. We develop a comprehensive framework for performing all these tasks.

We propose a temporal logic based language that can be used to specify the security policies in a succinct and unambiguous manner. We show that the protection features of information systems can be specified using the specification language Z and develop a method to concisely describe the set of traces generated by such a specification. We develop a set of rules, which can be applied inductively, to verify that the set of traces generated by a specification have the properties specified by the policy. These rules can also be used for verifying that the functionality of a system is not restricted more than required to enforce a given security policy.

For further information, see:

- Ramesh Peri, "Specification and Verification of Security Policies," PhD Dissertation, University of Virginia, 1995.

In a related vein, we are currently involved in developing a formal specification of the Legion security model. This effort has already proved worthwhile, by bringing to light questions and possible ambiguities in the informal description of the model. In the future, the formal specification will serve as a basis for formal analyses, permitting a user's security requirements to be precisely decomposed into those elements that Legion can be relied on to provide, and those that the user must explicitly account for. Finally, the formal specification provides a basis for certification of Legion implementation compliance, neatly sidestepping many of the problems that arise when one relies on an operational model in that capacity.

Object Replication

In many ways, the integrity elements of security and conventional fault-tolerance are very similar. Security is concerned with ensuring that a deliberately malfunctioning host cannot cause computations to reach incorrect results, while fault-tolerance is traditionally concerned with accidentally malfunctioning hosts. While there are obvious differences about assumptions one can make regarding failure modes, it is clear that security can leverage some ideas from fault tolerance and vice-versa.

Borrowing from fault tolerance, object duplication is proposed as a technique that can provide some assurance that a computation object run on an untrusted host will be permitted to execute without tampering. Under this approach, in order to determine whether an untrusted host has executed a request faithfully, a second copy of the object running on a trusted host can be used as an oracle. More generally, a second copy of the object can be run on another machine where the probability of both the original host and the second host collaborating to disrupt the object execution in an identical manner is considered acceptable. This scheme raises a number of interesting problems and questions and is rife with research potential.

Dynamic Security Compliance Checking

We are currently developing a test object: an object that can be used to stress-test an implementation of the Legion security mechanism. This object will check that the security features are correctly implemented, recording details about the frequency and order in which its security functions are invoked. In conjunction with other objects placing requests on it, the access, licensing, and caching mechanisms can be tested with any number of degenerate test cases. While such an object will be of considerable utility in implementing an instance of Legion, it can also be used to certify or license Legion implementations.

Perhaps the greatest use of such an object would come from the ability to spot-check hosts and sites for security violations. At present, we are actively exploring the questions surrounding such an approach, such as how to ensure that the site in question cannot tell that it is being checked and what sort of confidence metrics can be placed on such run-time checks.

legion@Virginia.edu
http://www.cs.virginia.edu/~legion/