Abstract

This paper provides a Z specification for the Software Architectural Tactics of Authentication and Authorization for the Security Quality Attribute. A model of a system is created and each tactic is defined with respect to the model. Each tactic is independent however, the system encompasses all the required functionality for all the tactics.

1. Introduction

An important step in creating a system that meets the users functional and non-functional requirements is the design of the software architecture. To achieve system qualities (non-functional requirements), Bass, Clements and Kazman [2] recommend the use of software architecture design tactics. Software architecture design tactics are high level design decisions. Each design tactic will satisfy one or more quality attributes and may adversely affect others [2].

Security is one set of quality attributes which has three classes of tactics. The first class is Resisting Attacks which consists of: Authenticate Users, Authorize Users, Maintain Data Confidentiality, Maintain Data Integrity, Limit Exposure, and Limit Access. The first four tactics correspond to their associated intuitive definitions. The idea of limit exposure is to provide only a few services on each host. The idea of limit access is to restrict what external entities can access the system. Second, Detecting Attacks which consists of using Intrusion Detection Systems. Lastly, Recovering From Attacks which consists of Auditing and Restoration. However, when defining the security tactics of a system, there is currently no way to formally prove the implementation of the tactics. One way to establish confidence in the implementation of the tactics is to use formal methods to define the system [2].

This paper presents the definitions of the authentication and authorization security tactics with respect to a generic system. Formal specification of other security tactics can be found in a detailed document [9]. There is extensive use of the Z formal language, for a good introduction to Z see [8].

2. Background and Related Works

Since the work done by Shaw and Garlan [6] on formal specification of software architectural styles, significant advances have been made in Software Architectural Methodologies.

Security, as previously stated, has become a well-recognized system quality [2]. While formal methods have been associated with software architecture since the mid-nineties, formal methods have been associated with security for even longer. Since computer security became an issue, people have looked at formal models of the security of a system. This is in part because without a formal definition of security, there is no way to precisely determine if a system is secure or not [4].

Another avenue of research is specifying the properties of the security components of a system. See [1] and [5]. Song et al. [7] create a system model and prove that an intrusion detection system can detect unauthorized access to the password file. Using ACL2 (a formal language), a Unix like system model is defined and logging is explicitly added. Then the intrusion detection systems detection rules are defined and they are proved to be correct. The approach this paper uses is similar to the approach Song et al. took in proving the correctness of their system.

3. Formal specification of the design tactics

3.1. Construction of tactic specification

The specification for each design tactic is built upon a simple base system model which is explained in Section 3.2. The operations for each design tactic build upon the extended system model and modify the base system operations.

3.2. Base system model specification

The base system is a simple, high level definition of a software system. The system is described in terms of an operating system as this makes it easier to understand and provides a ready source of examples. The specification can be applied to applications or other systems by a simple mapping. The base system consists
of three elements: a userlist, a filelist, and a processlist as shown in Figure 1. USER, FILENAME, and FILE_PROPERTIES are undefined sets, they depend on the system to be implemented. The filelist is a partial function from a filename to a file property. Thus, each filename is unique in the system and filenames do not have to map to a file property.

The basis of the process is that it is a file run by a user. Figure 1 shows the process specification, it consists of the user running the process, the file being run, the current state of the process and the processes id. PROCESS_ID and PROCESS_STATE are undefined sets. The PROCESS_STATE represents the current state of the process. As specified in Figure 1, the system requires the process id is unique and that the file being executed and the user running the process both exist. This model does not consider the sharing of resources, each operation is considered atomic. If concurrency is required then the model would need to be extended to include a concurrency mechanism.

**Figure 1. Base system and process**

There are many operations on even this simple system: adding and deleting users; adding, deleting, reading, and changing files; starting, stopping and changing processes. A representative operation is discussed here: AddUser.

The AddUser operation, shown in Figure 2, consists of two possible cases: AddUser_OK - a user is successfully added to the set of users and AddUser_Exists - the user already exists so an error is returned.

**Figure 2. Add user**

This specification describes the parts of a system that are monitored or extended to provide the various security tactics. The system as defined here has no security properties whatsoever, so any entity can manipulate the system.

### 3.3. Authentication tactic model specification

Authentication is the first security tactic to be added to the base system. Authentication is the act of verifying a users identity [3]. In software systems, authentication is achieved by having the user provide their name and a piece of information that uniquely identifies them [3]. This unique information is provided through the AUTHENTICATION_CONTROL set. Thus, the base system is extended to include a partial function from USER to an AUTHENTICATION_CONTROL set as shown in Figure 3. Note that each user can have any number of authentication controls and thus there can be users in the userlist that are not authenticated. This means the creation of a user is a two step process, first the user is created, then the user is mapped to an authentication control. This allows the system to provide separation of concerns, which is one of the secure design principles [3].

**Figure 3. System including authentication**

There are three new operations for the authenticated system. The first, consists of two parts: Authenticate_User and Authenticate_User_Fail. Authenticate_User, shown in Figure 4, provides the authentication tactic. It verifies that a user is allowed to access the system by making sure the username, authentication_control pair is in the authenticated function. The second, Authenticate_User_Fail, also shown in Figure 4, returns the appropriate error message if the username, authentication_control pair does not exist.

**Figure 4. Authenticate user**

The other two new operations added are AddAuth and DeleteAuth. They control how the authentication set is modified. Figure 5 shows AddAuth. AddAuth_OK confirms the user exists before adding the user, authentication_control pair to the authenticated set. If the user doesn’t exist, AddAuth_NotExists returns an error. AddAuth checks the user adding the authentication pair is authenticated, then attempts the add. DeleteAuth is similar.

**Figure 5. Add authentication**
AddAuth shows how all the operations on the system need to be modified to add authentication, the user performing the task must first be authorized, then the operation is performed or the authorization fails. Figure 6 shows how AddUser has been modified.

**Figure 6. Add user including authentication**

There are several important features that this specification emphasizes. First, the user is authenticated and then the action is performed, this prevents covert channels where different responses from the system can be used to determine properties of the system. Second, the system as specified performs complete mediation. Complete mediation, where every action is authenticated, is one of the secure design principles [3]. Third, the system, while requiring complete mediation, allows for a Kerberos style authentication scheme where the user is provided with a ticket or authentication control. This scheme allows the user a simpler access control, instead of having to provide their main authentication control for every action. Fourth, the authentication scheme is only as good as the authentication control, if the authentication control is easily compromised, then the system will not be secure. This is a feature of all authentication schemes and this specification make the association explicit. As this system model is refined, the designer will have to determine what constitutes a sufficiently strong authentication control for their system. For example, a system for the military would need stronger authentication than a system for a university. Finally, the system does not provide authorization. If an entity knows a correct user, authentication_control pair then that entity can perform every operation in the system.

### 3.4. Authorization tactic model specification

Authorization is the determination of whether an entity is allowed to perform an action or not. The access control matrix, which describes the rights every user has with respect to every object in the system [3], is one way to implement the authorization tactic. For this system, the access control matrix (ACM) consists of four separate sets, one for each type of object in the base system: userlist, filelist and processlist and one set that controls access to the ACM itself.

Each access control set is a mapping from the user performing the operation and the object being operated on to a set of rights. The rights are defined to coincide with all possible actions on the system. Access to the ACM must also be controlled. Access control for the ACM is defined for each of the four sets, the user is either granted access to a set in the ACM or not. This level of granularity was chosen for the specification because additional granularity, such as controlling whether a user can only add or delete entities from a given set in the ACM makes the specification more verbose without adding meaningfully to the specification. Figure 7 shows the new system and additional types.

**Figure 7. System including authorization**

The domains of the four parts of the ACM are subsets of the userlist, a user that doesn’t exist does not have rights within the system. However, a user that doesn’t exist can be an object in the ACM, in fact, users and files that don’t exist must be objects in the ACM, otherwise creation of objects would not be controlled by the ACM.

**Figure 8. Add user ACM**

The system has four new elements, the sets in the ACM and each element needs to be modifiable, therefore there are eight new operations, an add and a delete for each set in the ACM. Access to these operations is authorized through the acmACM set of the ACM. Figure 8 shows AddUACM. AddUACM takes three users, the first user is the user modifying the uACM, the second user is the user being granted the right and the third user is the object that the right is
being granted for. For example, AddUACM_OK(u1, u2, u3, {Add_User}) would, assuming u1 has the right to modify the uACM, allow u2 to add u3 to the userlist. The other operations for adding and deleting entries in the uACM, fACM, pACM and acmACM are similar.

Most of the revised operations of the base system are modified in a similar way to authentication, except each operation requires a check that the user has the correct rights, not that the user has the correct authentication_control. Figure 9 shows the modified AddUser.

![Figure 9. Add user including authorization](image)

The operations on processes have been modified to include a runAs user. The idea behind this is a user might want to start a process that has fewer rights than they do, for example, root on a Unix system starting the mailer daemon should not run the mailer daemon as itself.

The authorization of a system operates similarly to the authentication of a system, it wraps the existing system so a user must be authorized before they can try and perform any action. There are several features of authorization that this specification emphasizes. First, it does not perform authentication, which is a separate tactic. This does mean that if a system only implements authentication, if someone can imitate a user, they would immediately have that users rights. This is like having a Unix system where all the passwords are empty. In practice, authentication and authorization will almost always be used together. Second, like authentication, this specification performs complete mediation; every operation is checked to make sure the user attempting the operation is allowed to perform that action. Third, the ACM must be created in an appropriate manner, if a user is given rights that they should not have, the information within the system could be compromised. Finally, the ACM is capable of representing any style of access control: Roll Based Access Control, Discretionary Access Control or Mandatory Access Control by defining constraints on how the ACM is modified.

4. Discussion

This paper provides Z specifications of architectural tactics that can be used to achieve the security goals of a system. A base system model, with no security is specified and the authentication and authorization tactics are defined as constraints on the extended base system model.

This paper also demonstrates the power of creating Z specifications of architectural tactics and formal languages capacity to capture high-level abstract concepts.

Formal specifications of software systems provide a myriads of benefits [6]. The formal specification of the design tactics for the security quality attribute is no exception. First, it provides a template or building block for creating systems that make use of the various tactics. Second, the formal specifications of the tactics clarify the requirements of each tactic. Third, a formally defined system can also be rigorously analyzed. Forth, the tactic specifications create a framework in which to analyze specific security mechanisms.

There are several avenues of additional research. First, it is necessary to formally specify the other quality attributes. Second, as many of the tactics require policies such as an intrusion detection or logging policy, there is an area of additional work to create templates for the aforementioned policies.

For a detailed discussion of the results and future work, see [9].

5. References