جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الأمن السيبراني

Lecture: (7)

Access control software

Subject: authentication and access control

second Stage

Lecturer: Asst. Lecturer. Suha Alhussieny

Access control software

Many types of access control software and technology exist, and multiple components are often used together as part of a larger identity and access management (IAM) strategy. Software tools may be deployed on premises, in the cloud or both. They may focus primarily on a company's internal access management or outwardly on access management for customers. Types of access management software tools include the following:

- reporting and monitoring applications
- password management tools
- provisioning tools
- identity repositories
- security policy enforcement tools

Microsoft Active Directory is one example of software that includes most of the tools listed above in a single offering. Other IAM vendors with popular products include IBM, Idaptive and Okta.

Access Control Matrix

The classic view of authorization begins with Lampson's access control matrix. This matrix contains all of the relevant information needed by an operating system to make decisions about which users are allowed to do what with the various system

resources. We'll define a *subject* as a user of a system (not necessarily a human user) and an *object* as a system resource. Two fundamental constructs in the field of authorization are *access control lists,* or

ACLs, and *capabilities,* or C-lists. Both ACLs and C-lists are derived from Lampson's *access control matrix,* which has a row for every subject and a column for every object. Sensibly enough, the access allowed by subject *S* to object *O* is stored at the intersection of the row indexed by *S* and the column indexed by *O*

An example of an access control matrix appears in Table 8.1, where we use UNIX-style notation, that is, x, r, and w stand for execute, read, and write privileges, respectively.

Notice that in Table 8.1, the accounting program is treated as both an object and a subject. This is a useful fiction, since we can enforce the restriction that the accounting data is only modified by the accounting program. As discussed in, the intent here is to make corruption of the accounting data more difficult, since any changes to the accounting data must be done by software that, presumably, includes standard accounting checks and balances. However, this does not prevent all possible attacks, since the system administrator, Sam, could replace the accounting program with a faulty (or fraudulent) version and thereby break the protection. But this trick does allow Alice and Bob to access the accounting data without allowing them to corrupt it—either intentionally or unintentionally.

1. **ACLs and Capabilities**

Since all subjects and all objects appear in the access control matrix, it contains all of the relevant information on which authorization decisions can be based. However, there is a practical issue in managing a large access control matrix. A system could have hundreds of subjects (or more) and tens of thousands of objects (or more), in which case an access control matrix with millions of entries (or more) would need to be consulted before any operation by any subject on any object. Dealing with such a large matrix could impose a significant burden

on the system.

To obtain acceptable performance for authorization operations, the access control matrix can be partitioned into more manageable pieces. There are two obvious ways to split the access control matrix. First, we could split the matrix into its columns and store each column with its corresponding object. Then, whenever an object is accessed, its column of the access control matrix would be consulted to see whether the operation is allowed. These columns are known as access control lists, or ACLs. For example, the ACL corresponding to insurance data in Table 8.1 is (Bob, —), (Alice, rw), (Sam, rw), (accounting program, rw).

Alternatively, we could store the access control matrix by row, where each row is stored with its corresponding subject. Then, whenever a subject tries to files is required. This illustrates one of the inherent advantages of capabilities perform an operation, we can consult its row of the access control matrix to see if the

operation is allowed. This approach is know as capabilities, or C-lists. For example, Alice's C- list in Table 8.1 is

(OS, rx), (accounting program, rx), (accounting data, r), (insurance data, rw), (payroll data, rw).

It might seem that ACLs and C-lists are equivalent, since they simply provide different ways of storing the same information. However, there are some subtle differences between the two approaches. Consider the comparison of ACLs and capabilities illustrated in Figure 8.1. Note that the arrows in Figure 8.1 point in opposite directions, that is, for ACLs, the arrows point from the resources to the users, while for capabilities, the arrows point from the users to the resources. This seemingly trivial difference has real significance. In particular, with capabilities, the association between users and files is built into the system, while for an ACL-based system, a separate method for associating users.

1. Confused Deputy

The *confused deputy* is a classic security problem that arises in many contexts. For our illustration of this problem, we consider a system with two resources, a compiler and a file named BILL that contains critical billing information, and one user, Alice. The compiler can write to any file, while Alice can invoke the compiler and she can provide a filename where debugging information will be written.

However, Alice is not allowed to write to the file BILL, since she might corrupt the billing information. The access control matrix for this scenario appears in Table 8.2.

Table 8.2: Access Control Matrix for Confused Deputy Example

Now suppose that Alice invokes the compiler, and she provides BILL as the debug filename. Alice does not have the privilege to access the file BILL, so this command should fail. However, the compiler, which is acting on Alice's behalf, does have the privilege to overwrite BILL. If the compiler acts with its privilege, then a side effect of Alice's command will be the trashing of the BILL file, as illustrated in Figure 8.2.

Why is this problem known as the confused deputy? The compiler is acting on Alice's behalf, so it is her deputy. The compiler is confused since it is acting based on its own privileges when it should be acting based on Alice's privileges. With ACLs, it's more difficult (but not impossible) to avoid the confused deputy. In contrast, with capabilities it's relatively easy to prevent this problem, since capabilities are easily delegated, while ACLs are not. In a capabilities-based system, when Alice invokes the compiler, she can simply give her C-list to the compiler.

the compiler then consults Alice's C-list when checking privileges before attempting to create the debug file.

Since Alice does not have the privilege to overwrite BILL, the situation in Figure 8.2 can be avoided. A comparison of the relative advantages of ACLs and capabilities is instructive. ACLs are preferable when users manage their own files and when protection is data oriented. With ACLs, it's also easy to change rights to a particular resource. On the other hand, with capabilities it's easy to delegate (and sub-delegate and sub-sub-delegate, and so on), and it's

easier to add or delete users. Due to the ability to delegate, it's easy to avoid the confused deputy when using capabilities. However, capabilities are more complex to implement and they have somewhat higher overhead—although it may not be obvious, many of the difficult issues inherent in distributed systems arise in the context of capabilities. For these reasons, ACLs are used in practice far more often than capabilities.

. **Multilevel Security Models**

In general, security models are descriptive, not proscriptive. That is, these models tell us what needs to be protected, but they don't answer the real question, that is, how to provide such protection. This is not a flaw in the models, as they are designed to set a framework for protection, but it is an inherent limitation on the practical utility of security modeling.

Multilevel security, or MLS, is familiar to all fans of spy novels, where classified information often figures prominently. In MLS, the subjects are the users

(generally, human) and the objects are the data to be protected (for example, documents). Furthermore, *classifications* apply to objects while *clearances* apply to subjects. The

U.S. Department of Defense, or DoD, employs four levels of classifications and clearances, which can be ordered as

TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED. (1)

For example, a subject with a SECRET clearance is allowed access to objects classified SECRET or lower but not to objects classified TOP SECRET. Apparently to make them more visible, security levels are generally rendered in upper case.

Let *O* be an object and *S* a subject. Then *O* has a classification and *S* has a clearance. The security *level* of *O* is denoted *L(0),* and the security level of *S* is similarly denoted *L(S).* In the DoD system, the four levels shown above in (1) are used for both clearances and classifications. Also, for a person to obtain a SECRET clearance, a more-or-less routine background check is required, while a TOP SECRET clearance requires an extensive background check, a polygraph exam, a psychological profile, etc.

Multilevel security is needed when subjects and objects at different levels use the same system resources. The purpose of an MLS system is to enforce a form of access control by restricting subjects so that they only access objects for which they have the necessary clearance. Military and government have long had an interest in MLS.

Today, there are many potential uses for MLS outside of its traditional classified government setting. For example, most businesses have information that is restricted to, say, senior management, and other information that is available to all management, while still other proprietary information is available to everyone within the company and, finally, some information is available to everyone, including the general public. If this information is stored on a single system, the company must deal with MLS issues, even if they don't realize it. Note that these categories correspond directly to the

TOPSECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED classifications

There is also interest in MLS in such applications as network firewalls.

The goal in such a case is to keep an intruder, Trudy, at a low level to limit the damage that she can inflict after she breaches the firewall.

1. **Bell-LaPadula**

The first security model that we'll consider is Bell-LaPadula, or BLP, which, believe it or not, was named after its inventors, Bell and LaPadula. The purpose of BLP is to capture the minimal requirements, with respect to confidentiality, that any MLS system must satisfy. BLP consists of the following **two statements:**

. **Simple Security Condition:** Subject *S* can read object *O* if and only if *L(0) <L(S).*

. ***-Property** (Star Property): Subject *S* can write object *O* if and only if *L(S)* < *L(0).*

The simple security condition merely states that Alice, for example, cannot read a document for which she lacks the appropriate clearance. This condition is clearly required of any MLS system.

The star property is somewhat less obvious. This property is designed to prevent, say, TOP

SECRET information from being written to, say, a SECRET document. This would break MLS security since a user with a SECRET clearance could then read TOP SECRET information. The writing could occur intentionally or, for example, as the result of a computer virus. In his groundbreaking work on viruses, Cohen mentions that viruses could be used to break MLS securit, and such attacks remain a very real threat to MLS systems today.

The simple security condition can be summarized as "no read up," while the star property implies "no write down." Consequently, BLP is sometimes succinctlystated as "no read up, no write down." It's difficult to imagine a security model that's any simpler.

In response to McLean's criticisms, Bell and LaPadula fortified BLP with a *tranquility property.* Actually, there are two versions of this property. The strong tranquility property states that security labels can never change. This removes

McLean's system Z from the BLP realm, but it's also impractical in the real world, since security labels must sometimes change. For example, the DoD regularly declassifies documents, which would be impossible under strict adherence to the strong tranquility property. For another example, it is often desirable to enforce *least privilege.* If a user has, say, a TOP SECRET clearance but is only browsing UNCLASSIFIED Web pages, it is desirable to only give the user an UNCLASSIFIED clearance, so as to avoid accidentally divulging classified information. If the user later needs a higher clearance, his active clearance can be upgraded. This is known as the *high water mark principle,* and we'll see it again when we discuss Biba's model, below.

Bell and Lapadula also offered a *weak tranquility property* in which a security label can change, provided such a change does not violate an "established security policy." Weak tranquility can defeat system Z, and it can allow for least privilege, but the property is so vague as to be nearly meaningless for analytic purposes. Unfortunately, BLP may be too simple to be of any practical benefit.

1. Biba's Model

Whereas BLP deals with confidentiality, Biba's model deals with integrity. In fact, Biba's model is essentially an integrity version of BLP. If we trust the integrity of

object *0*\ but not that of object O2, then if object *O* is composed of *0*\ and O2, we cannot trust the integrity of object *O.*

In other words, the integrity level of *O* is the minimum of the integrity of any object contained in *O.* Another way to say this is that for integrity, a low water mark principle holds. In contrast, for confidentiality, a high water mark principle applies. To state Biba's model formally, let *1(0)* denote the integrity of object *O* and *I(S)* the integrity of subject *S.* Biba's model is defined by the two statements:

. **Write Access Rule:** Subject *S* can write object *O* if and only if *1(0) < I(S).*

. **Biba's Model:** A subject *S* can read the object *O* if and only if *I(S) < I(0).*

The write access rule states that we don't trust anything that *S* writes any more than we trust *S.* Biba's model states that we can't trust *S* any more than the lowest integrity object that *S* has read. In essence, we are concerned that *S* will be "contaminated" by lower integrity objects, so *S* is forbidden from viewing such objects.

Biba's model is actually very restrictive, since it prevents *S* from ever viewing an object at a lower integrity level.

Figure below illustrates the difference between BLP and Biba's model. Of course the fundamental difference is that BLP is for confidentiality, which implies a high water mark principle, while Biba is for integrity, which implies a low water mark principle.