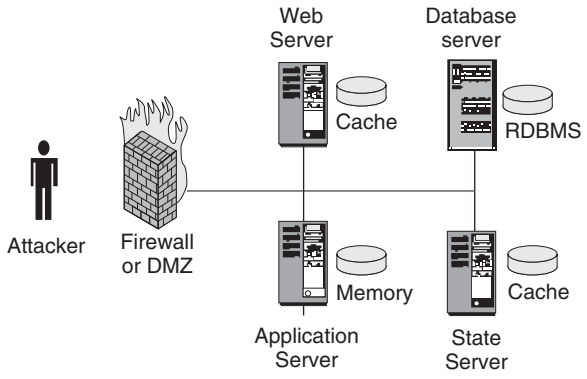
13.1 Information Obscurity

All systems are potentially liable to attack, whether from internal or external sources. If the information held by a system is sensitive, it should be protected. Part of this protection can take the form of obscuring the data itself, probably through some form of encryption, and obscuring information about the environment surrounding the data.

Example

A typical Internet technology system will use a combination of Web and application servers, together with a COMMON PERSISTENT STORE [Dys04], usually in the form of a common database, in which application data is stored. All these parts of the system will be protected from external attack by a firewall and possibly a DEMILITARIZED ZONE (449). However, this is no guarantee of security—what if the attacker breaches these external measures, or if an attack is internal to the organization?



Protection using a firewall or DMZ

The system will gather user information, such as credit card details, and store this in the database. The user information in the database is an obvious target for any attacker who wishes to steal or alter such information. Hence extra security measures may be put in place for the database. However, user information may also be retained temporarily by other parts of the system, in memory, in a cache, or in session state server, as shown in the figure on the previous page.

Application data can be protected by encrypting it, but such encryption is comparatively slow. Widespread use of encryption for all data in the system will impact system performance. Even then, there is no guarantee of security, as the system must

have access to the keys required to decrypt the data when it is needed by the application. This means that such keys are also vulnerable to attack. If the intruder can find and identify the encryption keys used for particular purposes, then all benefit from the encryption is lost. This can be addressed by designating one server to hold and distribute the keys. This server can then be specially protected. However, if an intruder can obtain credentials to access this server, then it too may be compromised, hence anywhere the application has access to such credentials (or equivalent privilege must also be protected).

Context

An APPLICATION SERVER ARCHITECTURE [Dys04] has been adopted to deliver Internet technology application servers together with a COMMON PERSISTENT STORE [Dys04]. The business logic and dynamic Web content generation of the application resides on application servers, while all static content is provided by Web servers that also act as a PROTECTION REVERSE PROXY (457) or an INTEGRATION REVERSE PROXY (465) for the dynamic Web content. The application gathers information on users and holds this in its database. The application is protected from external attack by a DEMILITARIZED ZONE (449).

Problem

How do we ensure that sensitive data gathered and stored by our system is protected from unauthorized access?

The solution to this problem must resolve the following forces:

- Much application data is non-sensitive, but the data that is sensitive needs to be protected in parts of the system that are vulnerable to attack. The degree of protection should be commensurate with the sensitivity of the data, and the data must still be readily accessible by the system itself.
- Encryption and decryption are comparatively slow and expensive in resource terms and so should be avoided unless necessary.
- To encrypt and decrypt information you need the appropriate encryption key. However, you must then guard this encryption key from unauthorized access.

Solution

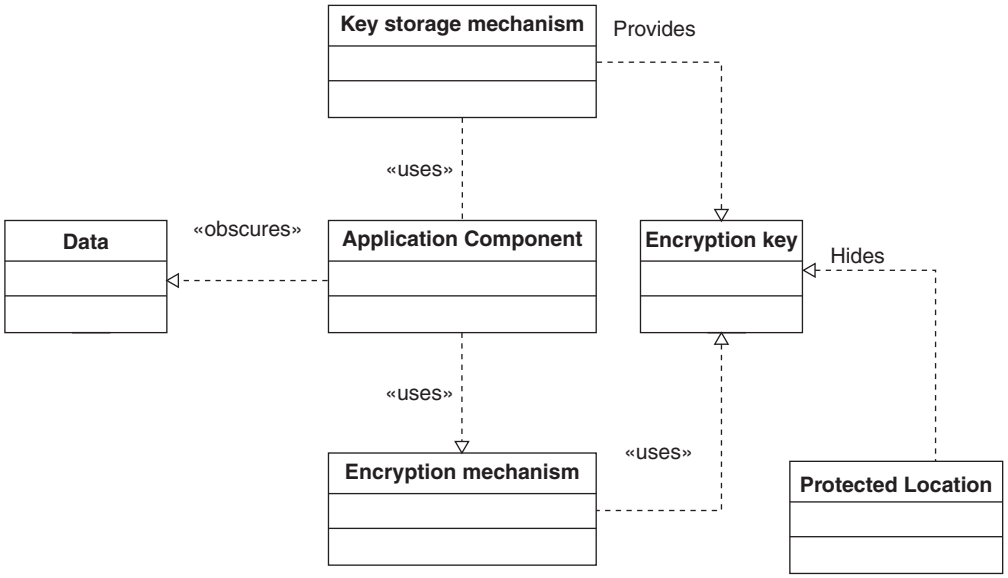
Grade the information held by the system for sensitivity. Obscure the more sensitive items of data using an encryption mechanism in situations in which it might be exposed to attack, while leaving the bulk of the application data unencrypted. Take appropriate measures to protect the encryption artifacts, such as encryption keys, from direct attack.

Structure

INFORMATION OBSCURITY (426) requires the following elements:

- Encryption keys, to encrypt and decrypt sensitive data.
- A key storage mechanism, to store and possibly distribute the keys. This could be anything from a system registry to an off-the-shelf key management server.
- An encryption mechanism, used with the encryption key to obscure data.
- An application component or components obtain and use encryption keys to secure application data in various parts of the system.
- A protected location, a place to store encryption artefacts used by the system. This location should itself be defended by obscurity and/or other defence mechanisms.

The following relationships govern the encryption of data to obscure its contents:



INFORMATION OBSCURITY structure

The application component uses an encryption mechanism, seeded with an appropriate key, to obscure the data it uses.

Implementation

Only part of the data held needs to be obscured, as only part of it is sensitive. The first task is therefore to categorize the data held and used by the system. This process

of identification and classification is a form of SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS (89), in this case based on considerations such as:

- The impact should that data be accessed by an unauthorized third party, for the user, for the company, and for the relationship between the two: for example, a list of HIV-infected patients on a medical system.
- The incentive for a third party to find this data: for example, credit card details.
- The accessibility of the place where the data is stored: for example, in a cache file on disk.
- Whether this data can be used to compromise further data.: for example, an encryption key.
- The data protection rules governing the specific type of data.

The last point should be well noted, as in many countries there are legal requirements for organizations to take due care in the management and protection of information gathered from customers and clients. Failure to conform to the appropriate set of rules will not only be insecure, but also illegal.

> Discussion: password protection in operating systems.
>
> A suitable example of how to categorize and manage sensitive data is the way in which passwords are managed by the operating systems on which our applications run. An obvious piece of data to protect is a user's password. The first thing to note about the way that Unix and Windows handle user and password data is that they only encrypt a small part of the information. The rest of the information, for example the user name, home directory, shell, and so on in the Unix `/etc/passwd` file remains in clear text, which means that it is far easier and faster to manipulate it than if it were encrypted. Only the sensitive part—the password itself—is encrypted: the level of encryption for passwords has increased markedly over time in Unix, and even more markedly in Microsoft Windows, which was originally a single-user system. The second thing to note is that in both cases, a one-way algorithm is used to encrypt the sensitive information (the password) and the security subsystem never decrypts the password back into plain text. The password remains obscured throughout its use in the system—the only time that it is in plain text is when it is typed in by the user. In cases in which we are using a piece of data to authenticate a user—a password, pass phrase, or the ubiquitous 'mother's maiden name'—it may be quite sufficient to store it in obscured form without the ability to retrieve the original plaintext.

Because part of the sensitivity assessment is based on the location of the data in the system, and hence its exposure to attackers, this audit should be repeated whenever the system architecture changes in a major way as the system evolves—for example, the introduction of a DEMILITARIZED ZONE (449). Ideally you should make the decisions about the sensitivity of the data independently of the decision about

the obscurity mechanism to be applied. If you find that you have lots of data that needs to be obscured to a high level, then the project's sponsors should be persuaded to make the budget available to do this.

Most sensitive user data will still be stored in a database. Small amounts of information can be encrypted and stored in character- or byte-based fields, while larger amounts of ciphertext would be stored as BLOBs (binary large objects). Whether you store your encrypted data in the database or on the file system, you will need metadata to describe it in order to identify the user with which it is associated, used as a primary key in the database, or the file name on the file system. For custom software elements you can use the encryption APIs provided in the Java and .NET world to manipulate encrypted data, although you need to be aware of some limitations built into cryptographic products exported from the USA, which limit key lengths for 'foreign' implementations. Alternatively you can buy third-party cryptographic libraries from many places that achieve the same purpose.

If any part of your system is not enabled for encrypted data, you may need to build a custom adapter. One way of reducing the need for obscurity is to increase the number of the strength of the 'locks' through which a cracker must pass to be able to access the data. You might find that it is easier in overall terms to implement a stronger DEMILITARIZED ZONE (449) and use less encryption within the internal network than to make many parts of your application encryption-aware.

One thing to remember here is that INFORMATION OBSCURITY (426), when applied to data, is concerned with the protection of information inside the application. Once it moves outside the application, or even onto the network between elements in the application, this data is still potentially vulnerable. For this reason you often find INFORMATION OBSCURITY (426) used in combination with SECURE CHANNELS (434), so that data is protected both inside the system and in transit.

As noted earlier, it is not just user data that needs to be protected, but also the configuration information used by the application. To be flexible, information used by the application for its own purposes is often held externally, for example in configuration files. However, some of this configuration information is in itself sensitive information. An example of such application configuration data would be an encryption key. The application needs the encryption key to access encrypted user data, but you do not want an intruder to obtain it easily. Information-based security artefacts such as encryption keys are particularly sensitive, as they can just be stolen—copied—without your knowledge if you don't spot the intrusion.

To secure this type of data, you could secure your external configuration file from unauthorized access. In addition, you could use an obscured name to identify the key in the configuration file. This makes it more difficult for an attacker to identify their target information. If you are still not happy with the level of security—for example, the file could be accessed over the network if the system is configured incorrectly—you could move the sensitive data into a location that is only accessible to local principals, such as the Windows system registry. Alternatively, you can embed the information in a binary artefact such as a compiled class or resource component to make

it more difficult to retrieve. In a late-bound environment such as Java or .NET, you might even want to obfuscate your bytecode or intermediate language to make it even less obvious which bit of data is the key.

Obviously, most of the considerations for the encryption key relate to the strength of the 'lock' protecting it. In the case of other sensitive information, such as a database connection string containing credentials for the database, encryption can be used to obscure the contents of the string to help prevent the discovery and use of the embedded credentials. This encrypted information can then be placed in a suitable location, as discussed above for encryption keys.

Once you have decided what is to be encrypted, you need to consider the impact on the rest of the system. The main issue with encryption is speed. Encryption and decryption on general purpose computer systems requires resource-intensive cryptographic algorithms to be run using the standard processor and memory. Although these resources are suitable for general application server usage, they are quite slow compared to what you would ideally want for cryptographic purposes. If you only require a small amount of cryptographic processing, this is usually acceptable. However, the more cryptographic processing you require, the more impact is caused by running it on sub-optimal hardware.

One solution would be to upgrade all the systems to have faster processors, for example, more on-board cache and faster memory. However, this would increase the cost of each system noticeably. The alternative is to buy dedicated hardware that performs the encryption and decryption. Depending on the level and type of encryption required, this would probably be cheaper than upgrading the processor and memory. It would almost certainly be faster.

One final aspect to consider is infrastructure security, as application security can be undermined by an insecurely-configured infrastructure. To address this, INFORMATION OBSCURITY (426) can also be used to help to improve the security of the infrastructure. Some parts of the system already use obscurity, for example when storing passwords. However, this can be undermined if a suitable password policy is not enforced. Other steps can be taken to make a system less vulnerable to attack, such as using obscure host names rather than, say, 'dataserver,' 'kerberos1' or 'keymanager.'

Example Resolved

All public data, such as catalog information held in caches and in memory on the Web servers, is held in plain text. However, any credit card details are held in encrypted form. The only place in the system where such details appear in plaintext is in memory on the application server as it is delivering this information to the credit card processing agency.

After weighing the possible consequences of data disclosure against the risk of intrusion, it is considered that the system contains other data worth encrypting explicitly—customer information. The passwords used by customers for personalization are encrypted anyway by the personalization and customization engine, but

their personal details however are not. The encryption used for the customer information is not too strong, as we don't want to impact system performance too much. The main intention is to make it difficult for any intruder to break this encryption casually.

One point to note is that there is a single encryption key used for all customer information, not one per customer. There is little benefit (and much complexity) in the use of multiple keys, as the intention is that the application server software is authorized to view this data, as it has the decryption key. The authentication and authorization of each customer is a separate matter —see KNOWN PARTNERS (442).

Known Uses

Web application components that cache sensitive data on the Web server will obscure the data in those caches. User authentication mechanisms apply INFORMATION OBSCURITY (426) to the data they need to maintain by only encrypting the user's password.

Consequences

The following benefits may be expected from applying this pattern:

- Security is improved by data obscurity because, even in the event of an attack during which the attacker gains access to the file system, system memory and application database, sensitive data is not usable by the attacker.
- The impact on system performance is minimized, because only a small percentage of the application and system data is typically encrypted in order to deliver a reasonable level of security.
- Security is also improved by configuration obscurity, because any attacker will find it more difficult to obtain the information they need to crack the system.

The following potential liabilities may arise from applying this pattern:

- Performance is impacted if an obscurity mechanism is introduced, due to the processing overhead associated with the mechanism. This is particularly true of complex encryption algorithms with long key lengths.
- Manageability is impacted, as additional configuration will be needed for any encryption mechanism, such as key management.
- Components that use obscured data may need to encrypt and decrypt that data themselves, so adding to the cost and effort of developing them.

- Cost is probably increased, as the extra requirements for encryption may require either additional general capability to support software encryption, or dedicated encryption hardware. You may also need to buy additional encryption software, depending on what comes with your existing platforms and tools.