Security and Authentication

References

- The Basics of Information Security[1], chapters 1, 2, 3, 5, 10, 12
- Web Security[2]
- Web Application Security: A Beginner's Guide[3]
- The Open Web Application Security Project (OWASP)[4] has many guidelines for applications.
- Foundations of Security[5]
- Computer and Information Technology Handbook[6]
- Network and System Security[7]
- Principles of Computer Security[8]

Notes

In this section we collect a variety of information related to security, culminating in a description of how to properly access or offer access to protected resources.

Security in general

We start with a broad description of security principles. There are three key areas of concern when discussing security:

Confidentiality Our ability to protect the data from those not authorized to access it. A good example of this principle is the advice to not share with anyone what your password is. We try to preserve confidentiality when we for example make sure noone is looking at us as we type in our password, or if we try to hide the ATM screen when typing in our PIN.

Integrity Our ability to prevent change in our data in an unauthorized or undesirable manner. This of course includes preventing unauthorized access, but it also incorporates the ability to recover and reverse any undesirable changes. A common example of this is our use of hard drive or database backups, so that if something happens to our computer we don't lose all our data. Integrity is particularly important for example in medical information, as corrupted data may result in the wrong treatment given to a patient.

[1] http://learning.acm.org/books/book_detail.cfm?id=2742551&type=elsevier
[2] http://learning.acm.org/books/book_detail.cfm?id=2821217&type=24
[3] http://learning.acm.org/books/book_detail.cfm?id=2829333&type=24
[4] https://www.owasp.org/index.php/Main_Page
[5] http://learning.acm.org/books/book_detail.cfm?id=1214959&type=24
[6] http://learning.acm.org/books/book_detail.cfm?id=2505467&type=elsevier
[7] http://learning.acm.org/books/book_detail.cfm?id=1841673&type=elsevier
[8] http://learning.acm.org/books/book_detail.cfm?id=2988382&type=24

Availability Our ability to provide the data when needed to those authorized to access it. For example we can provide confidentiality and integrity by burying a hard drive inside a block of concrete, but then we also lose our ability to get to that data. Another example of loss of availability is the so-called Denial of Service (DoS) attacks.

Along with the aforementioned goals, we must also consider various attack methods. These fit broadly into four groups:

Interception attacks attempt to access the data without proper authorization.
Interruption attacks prevent our access to the data, temporarily or permanently.
Modification attacks aim to tamper with our assets/data. A hacker changing our bank account's balance would be a good example of that.
Fabrication attacks aim to produce manufactured data. A malware sending email from our account is a good example of this.

Group activity: Suppose that the "data" we want to protect is the contents of the desk in your room. Consider various approaches to "securing" this data, and their tradeoffs/vulnerabilities regarding the above three concepts. Start with considering the kinds of possible attacks on that data.

Identification and Authentication

At the core of any security system is the ability to correctly identify and authenticate individuals attempting to access the system. This naturally breaks into two steps.

Identification is the claim that of what someone or something is. For instance using our debit card in an ATM is an identification. Another example would be claiming to be over 21 in a bar.
Authentication is the process of establishing that the claimed identification is true. For instance asking for the cardholder to type in the PIN number for the debit card is an authentication. Showing our ID to confirm our age would be another example of authentication.

Group activity: Think of other practical examples of identification and authentication.

Group activity: Think of computer-related examples of identification and authentication.

One topic worthy of discussion is the difference between *identity verification* and *authentication*. In general, authentication is more secure. The difference can be seen in a bank giving someone access to their account. Simply presenting an ID to verify your identity would not be sufficient to authenticate you as an account holder. You will likely need to possibly know the account number or have a bank card or key.

Authentication methods naturally fall into categories, called **factors**. The standard factors are the following:

Something you know For example a password, PIN number, social security number, date of birth etc. By itself this is a somewhat weak form of security, as the moment this information is exposed it is no longer secure.

Something you are This can consist of a variety of physical attributes, often known as biometrics. It would include your height, weight, eye color, iris/retina patterns, DNA, fingerprints. Some of this information is fairly transient (e.g. height or weight), while other parts like DNA can be fairly secure ways of identifying you. They are also expensive to set up.

Something you have This consists of the physical of electronic possession of some item or device. ATM cards and ID cards are good examples, as is your mobile phone or your email account when used in that fashion.

Something you do These are more complex aspects of who you are, for instance your gait, keyboard typing pattern or handwriting. These are difficult to falsify, but often result in false negatives when legitimate users may be incorrectly rejected.

The place you are For example certain servers are only accessible by terminals physically in the same room.

A common practice is the so-called **multi-factor authentication**, when we are using more than one factor to authenticate. A special case is the so-called two-factor authentication. For instance many online games now require you to download an "authenticator" to your phone, and when you want to log in to the game you need to both type your password (something you know) and type in the number shown on the authenticator (something you have).

Another example of this is using an ATM. You need to both have your debit card (something you have) and type in your PIN (something you know).

Group activity: Describe the authentication systems, if any, in place when using a credit card for a purchase, both physically at a store as well as online.

Group activity: Think of some single-factor systems and describe how we may turn them into multi-factor systems.

Authorization

After identification and authentication have been performed, the next step is authorization, namely providing access to only those assets that the authenticated party is supposed to be able to access. In other words, **authorization** determines what the authenticated party can and cannot do. These controls can be physical (a guard, gate etc), or logical (electronic lock).

A key principle when considering authorization is the following:

> **Principle of least priviledge**
>
> We should only allow the bare minimum of access to an authorized party, in order for it to perform the needed functionality.

There are many examples of this principle. For instance the staff working in the registrar's office should not have direct access to your business office information, and conversely someone working in the business office should not be able to sign you up for classes.

The system that we have in place right now in the college does not follow this principle. It would be nice if you could give your parents limited access to the "billing" part of your account, so they could monitor it and pay and so on, but the only way to really do that right now is to give them your password, which also gives them access to your email, classes, grades etc.

Another example is a web server. The process that is running the web server should be given access only to the files it needs to do its job, and not the rest of the system. Oftentimes however the process acting as the web server is a superuser of the system. This could allow an intruder to exploit a web vulnerability and execute instructions as a superuser/administrator.

Most systems implement a separate user account, often called "apache" in the case of the Apache web server, and all webpages get that user's more limited permissions.

Many problems on personal computers are caused by the (default) user account also being the administrator account. This means that any application or program downloaded from the internet will be executed with full administrator priviledges on the computer, and would thus be capable to do a lot of damage. Therefore it is a common guideline to always create separate "user" and "administrator" accounts.

Authorization is described most often in terms of **access control**. Every use case can be described in terms of the following basic operations:

Allowing access The most straightforward example of this is how a key gives you access to a locked room.

Denying access This should typically be the default. Placing a guard or a lock on a door is a typical example. Often times access is denied based on the time (for example campus buildings are locked after a certain time).

Limiting access It is often important to restrict the actions that can be taken. This is often described as running in a *sandbox* environment. A physical example would be that you may be allowed to access certain files in a room, but you may be forbidden from copying them or taking them out of the room. Another example would be access to a limited set of resources. For instance most faculty on campus have keys to their office building but not to other buildings, while some select personel have master keys that can open any door on campus.

Revoking access The system should be able to revoke the access it has given to someone, e.g. if they get fired.

There are fundamentally two ways in which one can provide access control.

Access Control Lists These are very common in many systems, from filesystems to web services. These list for any resource the identifiers of the parties that are allowed to access the resource, and what tasks they are allowed to perform. For

instance you may protect a file against writing, and only allow it to be read. Also we would prevent users from being able to access each other's files.

Another example of ACLs is the firewall that we use at the college, which blocks access to many resources from out of campus, while allowing emails or web requests through. In this instance the "identifier" is the specific IP address of the incoming request and "port" that it comes at. Email and web requests use specific ports, so the firewall blocks all other ports.

Capabilities These are less common, and their goal is to provide access based on a *token* rather than an identification of the individual. The holder of the token is given access, and thus for example a user can obtain a token for a specific action, then give that token to someone else. That other party can then use the token to obtain access to the resource. A good example of this is our college card system. If you give someone your card, then they can use it to for instance open a door, make photocopies, etc. But their access is mostly limited to those actions that the card enables. They still can't check your email for example, as they do not have your password.

Another example of this would be a prescription for a medicine. Anyone can pick up the medicine as long as they have the filled out prescription (and possibly an ID).

Group activity: Think of other authorization situations from your experience, and describe them according to the aforementioned groups.

Cryptographic Tools

When it comes to electronic security, Cryptography has given us a powerful set of tools. We will very briefly discuss some of these tools in this section. The full topic would go far beyond the scope of the course.

Let us clarify some terms used in a cryptographic system

message A piece of information that needs to be communicated between two parties.
ciphertext An encrypted version of the message.
(secret) key A piece of information that in combination with the message produces the ciphertext
cipher The specific algorithm describing how the message and the key are to be combined to produce the ciphertext.
sender The sender wants to send a message. They instead compute the ciphertext, a process called **encryption**, and send that.
receiver The receiver receives the ciphertext, and reverses the process to obtain the message. This is called **decryption**.
attacker The attacker attempts to determine the message based on the ciphertext. In various settings the attacker has varying capabilities, including asking the sender to encode various messages or trying to send various texts to the receiver and seeing what the receiver does with them.

A fundamental tenet of modern cryptographic systems is that everything else about the process is known to an attacker, except for the message and the secret key. The attacker gets to see the ciphertext and also has full knowledge of how the message and the secret key were processed to obtain that ciphertext. The attacker then wants to learn the message or the key, and if the system is secure then they should not be able to do so. The converse approach, trying to secure something by hiding the details, is called "security through obscurity", and is considered a bad and brittle approach.

The remarkable fact is that mathematics provides us with the means of producing these secure ciphers. Even though an attacker knows perfectly well what the ciphertext is and how it was obtained, they have no way at all to determine the message and key that gave rise to the ciphertext.

Symmetric Key Cryptography There are various kinds of ciphers. A first fundamental example is that of *symmetric key ciphers*, also known as *private key* ciphers. In this situation, there is a *common key* between the sender and the receiver, somehow agreed upon in advance. This same key is then used for both encryption and decryption.

These ciphers come in two flavors. The **stream ciphers** operate on one bit of the message at a time. The **block ciphers** on the other hand operate on blocks of bits, typically 64 bit, as a group. Most symmetric key ciphers in use nowadays are block ciphers.

Popular examples of symmetric key ciphers are DES, AES, RC5 and Blowfish.

Asymmetric Key Cryptography The asymmetric key ciphers differ from symmetric key ciphers in that they use two keys. The sender is given a **public key** which they use to encrypt the message, while the receiver has a **private key** that they use to descrypt the ciphertext.

The advantage of these ciphers is that they do not need for the parties to have had any prior "conversation" (in the symmetric key case the two parties must already share a secret key). The sender simply asks the receiver for their public key, and uses it to send the message.

Asymmetric ciphers are often used during the "handshake" portion of a client-server interaction, to establish a common secret key that the two parties can use for further information exchange via a symmetric cipher. For instance the first time your computer (the client) tries to connect to your bank's web server, they might initiate such an exchange. It might go something like that:

1. The client asks the server for their public key.
2. The server sends the public key back.
3. The client then creates a message M that consists of say 256 random bits.
4. The client uses the public key to encrypt that message and sends it over to the server.
5. The server uses their private key to decrypt the ciphertext and also learn the message M.

6. Now both client and server share this secret message M. They can now use it as the secret key for a symmetric cipher, to continue the rest of their conversation.

The reason you may want to do that is that asymmetric ciphers are a lot slower than symmetric ciphers.

Asymmetric ciphers often rely on deep mathematical conjectures. For instance the RSA cipher relies on our belief that for two very large prime numbers, if we are only given their product then we cannot recover the numbers.

Some popular asymmetric ciphers are the RSA, which is used to implement the SSL protocol used for all secure web exchanges (e.g. HTTPS), ElGamal and Diffie-Hellman.

This system relies on a level of trust when the server sends its public key to the rest of the world. *Certificates* are used to keep this kind of information somewhat protected. But this is a more complicated topic.

It's worth pointing out that in the past many of these algorithms were considered state secrets. For instance a popular cipher called PGP was on its release considered a *munition*, and its creator who released it to the world spent considerable time being accused of arms trafficking violations.

Cryptographic Hash Functions Cryptographic Hash functions are another tool in our arsenal. They are key-less, in the sense that they do not employ a secret key, and are one-way, in the sense we will describe in a moment.

A cryptographic hash function (briefly a hash function) turns a message into a much shorter **message digest** or **hash**. The resulting hash has a number of useful properties:

- It is extremely unlikely for two messages to produce the same hash. The hash can therefore act as a verification for the validity of the message.
- Even small changes in the original message result in vastly different hashes.
- It is infeasible to recover any part of the message from the hash.
- Given a hash, it is infeasible to produce a message that would result in that hash.

NOTE: Hash functions are also used to construct hash tables and dictionaries in many programming languages. These functions do NOT have all the good properties described above.

Some popular hash functions are MD5, SHA2 and SHA3.

MD5 in particular is extensively used to validate file downloads. Though it is slowly supplanted by SHA in recent instances. A company that has released a file/executable may provide you with multiple mirrors/locations from which you can download it. But to prevent the possibility that someone might change that executable and corrupt your system they also provide you with the MD5 hash for the executable. You can then run this hash on the downloaded file before executing it, to make sure it matches what the company gave you.

Digital Signatures Hash functions along with asymmetric key encryptions can be used to "digitally sign" a document. The server is here the one who wants to send a signed document to the client. The steps are as follows:

1. The server computes a hash of the document they want to send.
2. The server uses their private key to encrypt the hash, then appends the encrypted version to the document.
3. The server then sends that document to the client.
4. The client takes apart the portion of the document that is the encrypted hash.
5. The client uses the public key of the server to decrypt that encrypted hash, and therefore recover the hash that the server had computed.
6. The client separately computes the hash of the document that the server sent.
7. If the two hashes don't match, that means that someone tampered with the document in transit.

Digital Signatures thus assure the integrity of the message. If confidentiality is also needed, then we can add a symmetric encryption phase to ensure that as well, though it is important to get the details right and we will not discuss this further here.

Digital Certificates Digital Certificates can be used to link a public key to a particular individual or company. Therefore when we need to obtain someone's key, we instead obtain their certificate instead of directly asking them. This prevents so-called man-in-the-middle attacks.

Digital Certificates are issued by a **Certificate Authority (CA)**, which is a trusted party that digitally signs the certificates using their key. This way we only need to trust the certificate authority, rather than trust each individual or company to give us their true public keys. VeriSign is one such widely used CA.

There is a larger infrastructure that makes working with public keys effective. It is called a **Public Key Infrastructure** (PKI).

Web server/service Authentication

In this section we will discuss how to implement web-server authentication. The most standard system in place is a username-password system, with session information preserved. Typically this consists of the following:

1. Users are able to register for accounts by providing a username and password (and often an email address for verification and password recovery).
2. To access the site, users provide their username and password.
3. A session is created when users successfully log in. This session is somehow maintained so users don't have to provide their password on every single page.
4. Users have the option to log out of that session at any time.
5. Sessions typically time out after a while.

We will discuss how such a system can be implemented, and various gotchas along the way. Here are some things to watch out for:

1. If someone is monitoring the HTTP traffic, they should not be able to see the user's password.
2. Ideally, someone gaining access to the database should not be able to see the users' passwords.
3. Session IDs should not contain any sensitive information about the session, and the user should not be able to manipulate them.
4. An attacker should not be able to obtain a user's session ID and access that user's information.

HTTPS One first key tool is the use of HTTPS instead of HTTP. This adds a number of steps in the process:

1. A "secure transport" system is used, either SSL or TLS, to ensure the confidentiality and integrity of all the information sent back and forth between client and server. This ensures that anyone monitoring the HTTP traffic does not get to see any of the actual content, except more or less for the client and server addresses. You should never have any password submission page that is not HTTPS.
2. In order for that system to work, at least one of the sides must be using a digital certificate. This is often the server, so HTTPS protocols require a little bit more setup on the server side.
3. An asymmetric encryption system is used to establish a common secret key, and a symmetric encryption system based on that secret key is then used for all subsequent communications between client and server. This is all negotiated during what is known as the **handshake** phase.

While in some instances HTTPS is used only for the login page, it is preferable to instead use it throughout the site to better protect the session information.

Password Management Another key component is how passwords are being managed.

1. While we could simply store passwords in a database, this is not advisable. Any vulnerability to the database can expose the user's passwords to an attacker.

2. Instead, we store *hashes* of the passwords using an algorithm like md5. When a user attempts to log in and send us their password, we then compute its hash and compare it to the stored value.

3. As hashing is one-way, the database never stores a user's password. This is why whenever you report that you have forgotten your password, a new one is generated instead of telling you your password; because the server literally does not know your password.

4. As most people's passwords tend to have small "entropy" (i.e. the number of different possibilities is not all that big), they are often subject to what **dictionary attacks** or **rainbow table attacks**. To combat this, we often use what is known as a **salt**.

   The salt is a randomly generated sequence of fixed length (say 20 bytes) that we append to the password before hashing. We generate the salt when the user first creates their account, and we store it in the database alongside the username and hashed password. When the user tries to log in later, we recover the salt from the database, combine it with the password the user provided, compute the hash and compare it to the stored hash.

   This adds a bit of randomness to each account. Even if two users have the exact same password, they will have different salts and therefore will produce different hashes.

Session management Session aim to maintain user information across multiple HTTP(S) requests. This is what allows us to not have to log on every single page of a site. In order to achieve that, sessions are created.

1. When a user first logs in, a session is created. We can think of the session as a dictionary of useful and relevant information that should be available to the web page as the user navigate the site. It may include their username, role, expiration time of the session, temporary "shopping cart" type information etc.
2. A random session ID is generated. That ID is then to be used as the session identity. We store it in a database, alongside the actual data that comprise that session.
3. We effectively send the session ID back to the user, and when the user moves to the "next page", that ID is sent back to the server so that the server knows what the state of the system is.
4. This random ID is preferable to storing the various session information on the client side. That approach would potentially make it easier for the user to "change" that information to serve their purposes. But it has the downside that we must maintain a database of active sessions. (This can however be done on an in-memory database like Redis)
5. In order to prevent a user from trying to type in a session ID and potentially use another user's settings, the server **digitally signs** the session ID. Recall that this entails computing the ID's hash, then encrypting that hash with the server's private key and appending it to the ID before sending the ID back to the client. The server would then only accept IDs that are accompanied by this extra hash bit, which the server can decrypt and compare to the session ID's hash to make sure noone tried to change the hash.
6. Typically **cookies** are used to communicate the session ID information back and forth. It is important that the cookies are set up as **secure** and **httpOnly**. This ensures that the browser only sends the session information over an HTTP**S** connection, and also does not expose the cookie to the Javascript code running on the page.