Introduction to Cybersecurity

UT Knoxville, Spring 2023, COSC 366

Alex Zhang

February 6, 2023

Contents

1 Security Concepts and Principles 2

1.1 Fundamental Goals of Computer Security 2

1.2 Computer security policies and attacks 3

1.3 Risk, risk assessment, and modeling expected losses 5

1.4 Adversary modeling and security analysis 6

1.5 Threat Modeling 7

1.6 Threat Model Gaps 8

1.7 Design Principles 8

1.8 Review 9

2 Cryptography 10

2.1 Introduction 10

2.2 Symmetric Encryption Model 11

2.3 Asymmetric Encryption Model 12

2.4 Digital Signatures 13

2.5 Cryptographic Hash Functions 14

2.6 Message Authentication Codes 15

2.7 Passwords 16

2.8 Password Authentication 17

Index 18

1

Security Concepts and Principles

1.1 Fundamental Goals of Computer Security

Definition 1.1.1 ▸ Computer Security

Computer security is the practice of protecting computer-related assets from unauthorized actions, either by preventing such actions or detecting and recovering from them.

The goal of Computer Security is to help users complete their desired task safely, without short or long term risks. To do this, we support computer-based services by providing essential security properties.

Definition 1.1.2 ▸ Confidentiality, Integrity, Availablity (CIA)

The ***CIA triad*** is a pneumonic device that includes three essential security properties:

- ***Confidentiality***: only authorized parties can access data; applies at rest and in motion (data being transmitted)
- ***Integrity***: data, software or hardware remaining unchanged, except by authorized parties
- ***Availability***: information, services and computing resources are available for authorized use

Definition 1.1.3 ▸ Principal, Privilege

A ***principal*** is an entity with a given identity, such as a user, service, or system process. A ***privilege*** defines what a principal can do, such as read/write/execute permissions.

In addition to the Confidentiality, Integrity, Availablity (CIA), we also have three security properties relating to principals.

Definition 1.1.4 ▸ Authentication, Authorization, Auditability (Golden Principles)

The ***Golden Principles*** are three security properties regarding principals:

- ***Authentication***: assurance that a principal is who they say they are
- ***Authorization***: proof that an entity has the necessary privilege to take the request

action; most commonly done by authenticating a principal, and lookup up its privileges

- ***Auditability***: ability to identify principals responsible for past actions

Auditability (or accountability) gives away two key things: who conducted the attack and the methods by which an attack was made.

Definition 1.1.5 ▶ Trustworthy, Trusted

Something is ***trustworthy*** if it deserves our confidence. Something is ***trusted*** if it has our confidence.

Definition 1.1.6 ▶ Privacy, Confidentiality

Privacy is a sense of being in control of access that others have to ourselves. It deals exclusively with people. ***Confidentiality*** is an extension of privacy to also include personally sensitive data.

1.2 Computer security policies and attacks

Definition 1.2.1 ▶ Asset

An ***asset*** is a resource we want to protect, such as information, software, hardware, or computing/communications services.

Note that asset can refer to any tangible or intangible resources.

Definition 1.2.2 ▶ Security Policy

A ***security policy*** specifies the design intent of a system's rules and practices (i.e. what the system is supposed to do and not do).

Definition 1.2.3 ▶ Adversary

An ***adversary*** is an entity who wants to violate a security policy to harm an asset.

Can also be called "threat agents" or "threat actors".

A formal security policy should precisely define each possible system state as either authorized (secure) or unauthorized (non-secure). Non-secure states raise the potential for attacks to happen.

Definition 1.2.4 ▸ Threat

A ***threat*** is any combination of circumstances and entities allow harm to assets or cause security violations.

Definition 1.2.5 ▸ Attack, Attack Vector

An ***attack*** is a deliberate attempt to cause a security violation. An ***attack vector*** is the specific methods and steps by which an attack was executed.

Definition 1.2.6 ▸ Mitigation

Mitigation describes countermeasures to reduce the chance of a threat being actualized or lessen the cost of a successful attack.

Mitigation includes operational and management processes, software controls, and other security mechanisms.

Example 1.2.1 ▸ House Security Policy

Consider this simple security policy for a house: no one is allowed in the house unless accompanied by a family member, and only family members are authorized to take things out of the house.

- The presence of someone who wants to steal an asset from our house is a **threat**.
- An unaccompanied stranger in the house is a **security violation**.
- An unlocked door is a **vulnerability**.
- A stranger entering through the unlocked door and stealing a television is an **attack**.
- Entry through the unlocked door is an **attack vector**.

1.3 Risk, risk assessment, and modeling expected losses

Definition 1.3.1 ▸ Risk

A ***risk*** is the expected loss of assets due to future attacks.

There are two ways we assess risk: ***quantitative*** and ***qualitative*** risk assessment.

- ***Quantitative*** risk assessment computes numerical estimate of risk
- ***Qualitative*** risk assessment compares risks relative to each other

Quantitative risk assessment is more suited for incidents that occur regularly, with historical data and stable statistics to generate probability estimates. However, computer security incidents occur so infrequently that any estimate of probability likely isn't precise. In contrast, qualitative risk assessment is usually more practical. For each asset or asset class, their relevant threats are categorized based on probability of happening and impact if it happened.

Precise estimates of risk are rarely possible in practice, so qualitative risk assessment is usually yields more informed decisions. A popular equation for modeling risk is:

$$R = T \cdot V \cdot C$$

where:

- T is the probability of an attack happening,
- V is the probability such a vulnerability exists, and
- C is cost of a successful attack, both tangible and intangible costs

C can encompass tangible losses like money or intangible losses like reputation. Whatever model we use, we can then model expected losses as such:

In risk assessment, we ask ourselves these questions:

1. What assets are most valuable, and what are their values?
2. What system vulnerabilities exist?
3.

In answering these questions, it becomes apparent we cannot employ strictly quantitative risk assessment. A popular model for qualitative risk assessment is the DREAD model:

Definition 1.3.2 ▸ DREAD

DREAD is a method of qualitative risk assessment using a subjective scaled rating system for five attributes.

Attribute	10	1
Damage Potential	data is extremely sensitive	data is worthless
Reproducibility	works every time	works only once
Exploitability	anyone can mount an attack	requires a nation state
Affected Users	91-100% of users	0% of users
Discoverability	threat is obviously apparent	threat is undetectable

The final DREAD score takes the average of all five attributes.

A common criticism of DREAD is that rating discoverability might reward security through obscurity. Often, people will omit discoverability, or simply assign it the maximum value all the time.

$$ALE = \sum_{i=1}^{n} F_i \cdot C_i$$

where:

- F_i is the estimated frequency of events of type i, and
- C_i is the average loss expected per occurence of an event of type i

1.4 Adversary modeling and security analysis

Definition 1.4.1 ▸ Adversary

An ***adversary*** is an entity that wants to violate a security policy in order to harm an asset.

- ***identity***: who are they?
- ***objectives***: what assets the adversary might try to harm
- ***methods***: the potential attack techniques or types of attacks
- ***capabilities***: skills, knowledge, personnel, and opportunity

In designing computer security mechanisms, it is important to identify traits about potential adversaries. Some attributes may include:

Different adversaries can have wildly different objectives, methods, and capabilities.

Definition 1.4.2 ▶ Security Analysis

Security analysis aims to identify vulnerabilities and overlooked threats, as well as ways to improve defense against such threats

Types of analysis:

- Formal security evaluation: standardized testing to ensure essential features and security
- Internal vulnerability testing: internal team trying to find vulnerabilities
- External penetration testing: third-party audits to find vulnerabilities; often the most effective

Security analysis is a heavily involved process that may employ a variety of methodologies. For example, manual source code review, review of design documents, and various penetration testing techniques. It's important to be aware of potential vulnerabilities as we are writing code.

1.5 Threat Modeling

A threat model will identify possible adversaries, threats, and attack vectors. We need to list a set of assumptions about the threats as well as clarify what is in and out of the scope of possibilities.

Diagram-Driven Threat Model Threat model diagrams include assets, infrastructure, and defenses/mitigations, making apparent the possible attack vectors. Popular examples include:

- ***Data Flow Diagrams*** – models all possible data routes
- ***User Workflow Diagram*** – models how users interact with the program, both frontend and backend
- ***Attack Trees*** – models all possible attack vectors like a flow chart; leaf nodes are initial actions

Definition 1.5.1 ▶ STRIDE

STRIDE is a model for identifying computer security threats.

- ***Spoofing*** – attacker impersonates another user, or malicious server posing as a legitimate server
- ***Tampering*** – altering data without proper authorization; can occur when data is stored, processed
- ***Repudiation*** – lying about past actions (e.g. denying/claiming that something happened)
- ***Information Disclosure*** – exposure of confidential information without authorization
- ***Denial of Service*** – render service unusable or unreliable for users
- ***Elevation of Privilege*** – an unprivileged user gains privileges

1.6 Threat Model Gaps

Often, wrong assumptions about risk or focus on wrong threats will lead to gaps between our threat model and what can realistically happen (e.g. if we don't account for something, but it does happen).

Changing Times New adversaries, technologies, software, and controls means we need to constantly adjust our threat model.

1.7 Design Principles

1. Simplicity and necessity; complexity increases risks (KISS – keep it simple, stupid)
2. Safe Defaults – get security out of the box; users rarely change defaults
3. Open Design – don't rely on the secrecy of our code; it will leak
4. Complete Mediation – never assume access is safe; always check access is allowed
5. Least Privilege – don't give principals extraneous privileges; limit impact of compromise
6. Defense in depth – multiple layers of security; don't rely on only one security control
7. Security by design – think about security throughout development, not an afterthought
8. Design for evolution – allow for change for whenever it's needed

1.8 Review

CIA Triad, Golden principles, trusted vs. trustworthy, confidentialty vs. privacy.

TODO: add more review stuff from slides here; flesh out notes to reflect review

2

Cryptography

Cryptographic Rules to Remember

1. Do not design your own cryptographic protocols or algorithms.
2. ***Kerckhoff's Principle*** – security should only come from the secrecy of the crytographic key, NOT the algorithm or code.
3. Encryption only provides confidentiality, not integrity.

2.1 Introduction

Definition 2.1.1 ▶ Cipher, Plain Text, Cipher Text

A ***cipher*** is any encryption or decryption algorithm. A ***plaintext*** message is vulnerable, usually prior to cipher. A ***ciphertext*** message is secure, usually after cipher is applied.

Definition 2.1.2 ▶ Cryptographic Key

A ***cryptographic key*** is a value which is used to decrypt cipher text; should be relatively large and remain private.

The ***key space*** is the set of all possible cryptographic keys. It should be large enough such that it would be impractical to try every possible key. For example, AES-256 uses 256 bit keys. Even a thermodynamically perfect computer with the power of the sun could not even count through 2^{256} keys, much less guess and check them.

Some possible threats to encryption may be:

- Brute force attack – guess and check all keys
- Algorithmic break – some flaw in the algorithm

Some possible adversaries include:

- Passive – just listens to communications
- Active – can modify data

Definition 2.1.3 ▶ Information-Theoretic Security

We say information has ***information-theoretic security*** if given **unlimited** computer power and time, an attacker could not recover the plaintext from the ciphertext.

Definition 2.1.4 ▶ Computational Security

We say information has ***computational security*** if given **fixed** computer power and time, an attacker could not recover the plaintext from the ciphertext.

Definition 2.1.5 ▶ Stream Cipher, Block Cipher

A ***stream cipher*** encrypts information one bit at a time. A ***block cipher*** encrypts information in blocks.

2.2 Symmetric Encryption Model

Definition 2.2.1 ▶ Symmetric-Key Algorithm

A ***symmetric-key algorithm*** encrypts plaintext and decrypts ciphertext using the same cryptographic key.

Definition 2.2.2 ▶ One-Time Pad (OTP)

The ***one-time pad*** is a symmetric stream cipher that XOR's a message with a key. The resulting ciphertext is information theoretic so long as the following are met:

- Key must be as long as the plaintext
- Key must be random
- Key must never be reused
- Key must be kept completely secret

Many modern cryptographic algorithms work by taking a small cryptographic key and expanding it into a sufficiently large one-time pad key.

Definition 2.2.3 ▶ Advanced Encryption Standard (AES)

AES is the most common symmetric block cipher, providing computational security.

- Messages are split into 128-bit blocks (with padding)

- Cryptographic key is 128, 196, or 256 bits long
- Provides computational security
- A single bit changed should change (on average) 50% of the ciphertext

In addition, different block modes help to remove patterns among blocks.

2.3 Asymmetric Encryption Model

Definition 2.3.1 ▶ Asymmetric Encryption, Public Key, Private Key

Asymmetric encryption utilizes two cryptographic keys: a ***public key*** which encrypts plaintext, and a ***private key*** which decrypts ciphertext.

Unfortunately, public key encryption is orders of magnitude slower than symmetric key encryption. We can circumvent this through hybrid encryption.

Definition 2.3.2 ▶ Hybrid Encryption, Key Wrap

Hybrid encryption incorporates both symmetric key and public key encryption. We encrypt our plaintext using a symmetric key, and we send both the encrypted ciphertext and the symmetric key encrypted using the other party's public key. This encrypted key is called a ***key wrap***.

Definition 2.3.3 ▶ Padding

Many encryption algorithms ***pad*** messages to:

- ensure the message is properly formatted
- prevent attacks by adding random noise

Suppose we only need to send a single integer. While the key space is large, the message space is small. Thus, an attacker could guess and check all possible messages until it matches the encrypted message.

There are many ways to pad a message. Generally, we want to use the optimal asymmetric encryption padding (OAEP).

Definition 2.3.4 ▶ RSA

RSA is a common and mathematically simple cryptographic algorithm that provides computational security.

Code Snippet 2.3.1 ▶ Basic RSA Encryption using C#

```
var message = "Hello World!";
var plaintext = Encoding.ASCII.GetString(ciphertext);
var rsa = RSA.Create(2048);
var public_key = rsa.ExportRSAPublicKeyPem();
var private_key = rsa.exportRSAPrivateKeyPem();

var ciphertext = rsa.Encrypt(plaintext, RSAEncryptionPadding.OaepSHA256);

var decrypted = rsa.Decrypt(ciphertext, RSAEncryptionPadding.OaepSHA256);
var decrypted_plaintext = Encoding.ASCII.GetString(decrypted);
```

2.4 Digital Signatures

Definition 2.4.1 ▶ Digital Signature

We can ***digitally sign*** information to verify that we made it. In this context, we use the ***private key*** to sign data, and others use the ***public key*** to verify a signature.

Digital signatures provide three security properties:

1. Data origin authentication – the data comes from the owner of the private key
2. Data integrity – the data has not changed since it was sent
3. Non-repudiation – the sender cannot later claim to have not sent the data

We avoid using the same key for signing and for cryptography because:

1. If our private key gets stolen, someone can then decrypt our data **and** forge digital signatures.

2. There can be mathematical interactions between encryption and signing that can reveal information about our key.

Similar to encryption algorithms, there are signing algorithms that create digital signatures and verification algorithms that verify digital signatures. In addition, we still pad our messages before signing to ensure the message is formatted and to prevent some (relatively esoteric) attacks.

Data is hashed before signed, and different key pairs should be used for encryption/decryption and signing/verification.

2.5 Cryptographic Hash Functions

Definition 2.5.1 ▶ Hash Function

A ***hash function*** takes an arbitrary length string and outputs a unique fixed output length string.

$$H : 2^* \to 2^n$$

Usage:

- H can be applied to data of any size
- H outputs a fixed length data
- H is fast to compute

Security properties:

- ***One-way property***: Theoretically impossible to find the original message from the hash; one hashed string maps to an infinite amount of input strings
- ***Weak collision resistance***: Given an input, it's computationally impossible to find another input that produces the same hash.
- ***Strong collision resistance***: It's computationally impossible to find any pair of distinct inputs that produce the same hash.

Definition 2.5.2 ▶ SHA

SHA is a common hash function.

Example 2.5.1 ▸ Data Integrity

- When sending data, first hash it and send the hash along with the data
- When receiving the data, first hash it and check that the calculated hash matches the one that was received
- TODO: finish from slides

Example 2.5.2 ▸ Password Storage

Instead of storing passwords in plaintext, the server stores the hashed passwords.

Hashing at client and then sending to server is flawed!

- If an attacker steals the hashed passwords, an attack can simply log in using the hashed passwords.

2.6 Message Authentication Codes

- Symetric-key equivalent to digital signatures
- Provides data origin authentication and data integrity (Only for two people)
- Does NOT provide non-repudiation (at least two people will have the symmetric key)

Definition 2.6.1 ▸ Tag

Definition 2.6.2 ▸ Hash-Based Message Authentication Code (HMAC)

Can be used with any cryptographic hash function, only use with safe functions (e.g. HMAC-SHA256)

Definition 2.6.3 ▸ CMAC

Based on symmetric encryption using CBC mode

Definition 2.6.4 ▸ UMAC

Based on universal hashing; pick a hash function based on the key, then encrypt the digest

Again, don’t reuse the same key for encryption and decryption

Code Snippet 2.6.1 ▸ Simple HMAC in C#

```
using System.Security.Cryptography;
using System.Text;

var message = "Hello World!";
var plaintext = Encoding.ASCII.GetBytes(message);

var digest = SHA256.HashData(plaintext);
var digest2 = SHA256.HashData(plaintext);

// Verify the two digests are the same
digest.SequenceEqual(digest2)

var receivedtext = "Gello World!";
var key = RandomNumberGenerator.GetBytes(512);
var calculatedMAC = HMACSHA256.HashData(key, plaintext);
var receivedMAC = HMACSHA256.HashData(key, receivedtext);

// This will be false
calculatedMAC.SequenceEqual(receivedMAC);
```

2.7 Passwords

In regard to passwords, we usually have an associated account with a username, public or secret. The password is a string of enterable characters, most commonly ASCII. We are more concerned with the bits of the password, not necessarily the characters. As such, we need to have a deterministic way of converting characters to bits, whether it be ASCII or UTF-8.

Advantages of passwords:

- Simple and easy to learn
- Free
- No physical load to carry
- Usually easy to recover lost access

- Easily delegated (e.g. sharing Netflix passwords)

Disadvantages:

- Hard to create random passwords (theoretical key space is large, but practical key space is small)
- Hard to remember good passwords
- Easy to reuse passwords

Definition 2.7.1 ▶ Password Composition Policies (PCP)

Specifies minimum and maximum password length; specifies character requirements and restrictions

Although intended to improve password strength, PCPs often backfire.

Definition 2.7.2 ▶ Passkey

Passwords can be changed into cryptographic keys called ***passkeys***.

PBKDF2 is a common function. Modern algorithms like Argon2id can also be used.

2.8 Password Authentication

Example 2.8.1 ▶ Most Basic Password Authentication

Registration:

- User sends a username and password to the website
- Website stores the info

Authentication:

- User sends a username and password to the website
- Website compares the sent values with the stored values

Some flaws:

- Data can be intercepted when sending to the server; need to encrypt communication between user and server
- Phishing: attacker tricks user into giving the attacker their info; hard to mitigate,

could use password managers or hardware security tokens

- Online password guessing; have to rate limit authentication attempts, can also require strong passwords
- Password database theft

Example 2.8.2 ▸ Most Basic Password Authentication

Registration:

- User sends a username and password to the website
- Website hashes the info, then stores the info

Authentication:

- User sends a username and password to the website
- Website compares the sent values with the stored values

Index

Definitions

1.1.1 Computer Security . . . 2
1.1.2 Confidentiality, Integrity, Availablity (CIA) . . . 2
1.1.3 Principal, Privilege . . . 2
1.1.4 Authentication, Authorization, Auditability (Golden Principles) . . . 2
1.1.5 Trustworthy, Trusted . . . 3
1.1.6 Privacy, Confidentiality . . . 3
1.2.1 Asset . . . 3
1.2.2 Security Policy . . . 3
1.2.3 Adversary . . . 3
1.2.4 Threat . . . 4
1.2.5 Attack, Attack Vector . . . 4
1.2.6 Mitigation . . . 4
1.3.1 Risk . . . 5
1.3.2 DREAD . . . 6
1.4.1 Adversary . . . 6
1.4.2 Security Analysis . . . 7
1.5.1 STRIDE . . . 8
2.1.1 Cipher, Plain Text, Cipher Text . . . 10
2.1.2 Cryptographic Key . . . 10
2.1.3 Information-Theoretic Security . . . 11
2.1.4 Computational Security . . . 11
2.1.5 Stream Cipher, Block Cipher . . . 11
2.2.1 Symmetric-Key Algorithm . . . 11
2.2.2 One-Time Pad (OTP) . . . 11
2.2.3 Advanced Encryption Standard (AES) . . . 11
2.3.1 Asymmetric Encryption, Public Key, Private Key . . . 12
2.3.2 Hybrid Encryption, Key Wrap . . . 12
2.3.3 Padding . . . 12
2.3.4 RSA . . . 13
2.4.1 Digital Signature . . . 13

2.5.1 Hash Function 14
2.5.2 SHA 14
2.6.1 Tag 15
2.6.2 Hash-Based Message Authentication Code (HMAC) 15
2.6.3 CMAC 15
2.6.4 UMAC 15
2.7.1 Password Composition Policies (PCP) 17
2.7.2 Passkey 17

Examples

1.2.1 House Security Policy 4
2.5.1 Data Integrity 15
2.5.2 Password Storage 15
2.8.1 Most Basic Password Authentication 17
2.8.2 Most Basic Password Authentication 18

Code Snippets

2.3.1 Basic RSA Encryption using C# 13
2.6.1 Simple HMAC in C# 16