CPSC 329

David Ng

Winter 2017

Contents

1 January 9, 2017 **5**
1.1 Introduction 5

2 January 11, 2017 **5**
2.1 Information Security 5
2.2 Challenges to Computer Security 6
2.3 Computer Security Terminology 6
2.4 Threat Consequences 7

3 January 13, 2017 **9**
3.1 Computer System Assets 9
3.2 Network Attacks 9
3.3 Security Design Principles 10
3.4 Methods of Attack 10

4 January 16, 2017 **11**
4.1 Computer Security Strategy 11
4.2 Password Authentication 11
4.3 Guess-Verify Attack 13

5 January 18, 2017 **13**
5.1 Guess-Verify Attack Cont'd 13
5.2 Defending Against Guess-Verify Attacks 14
5.3 Keystroke Logging, Phishing, and Social Engineering 15

6 January 20, 2017 **15**
6.1 Password Protection 15

7 January 23, 2017 **17**
7.1 Graphical Passwords 17

8 January 25, 2017 **18**
8.1 Vulnerabilities . . . 18
8.2 Token Based Authentication . . . 19

9 January 27, 2017 **20**
9.1 Token Based Authentication Cont'd . . . 20

10 January 30, 2017 **21**
10.1 Token Based Authentication Cont'd . . . 21

11 February 1, 2017 **22**
11.1 Biometric Authentication . . . 22
11.2 Fingerprint and Iris Scanning . . . 24

12 February 3, 2017 **24**
12.1 Handwriting Biometrics . . . 24
12.2 Biometric Assessment . . . 25

13 February 6, 2017 **26**
13.1 Access Control . . . 26
13.2 Unix/Linux File System Security . . . 27

14 February 8, 2017 **29**
14.1 NTF . . . 29

15 February 10, 2017 **29**
15.1 Role Based Access Control (RBAC) . . . 29
15.2 Attribute Based Access Control (ABAC) . . . 31

16 February 13, 2017 **31**
16.1 Midterm Review . . . 31

17 February 27, 2017 **31**
17.1 Introduction to Cryptography . . . 31
17.2 Ciphers . . . 32

18 March 1, 2017 **33**
18.1 Ciphers Cont'd . . . 33

19 March 3, 2017 **35**
19.1 Measures of Security . . . 35
19.2 Types of Cryptography . . . 35
19.3 Cryptanalytic Attacks . . . 36

20 March 6, 2017 **36**
20.1 Cryptographic Hash Functions 36

21 March 8, 2017 **39**
21.1 Cryptographic Hash Function Applications 39

22 March 10, 2017 **40**
22.1 Symmetric Cryptography 40
22.2 Block Ciphers 41

23 March 13, 2017 **43**
23.1 Block Cipher Modes of Operation 43
23.2 Stream Ciphers 43
23.3 Confusion and Diffusion 44

24 March 15, 2017 **45**
24.1 Public Key Cryptography 45
24.2 Diffie-Hellman Key Exchange 46

25 March 17, 2017 **48**
25.1 Enveloped Public Key Encryption 48
25.2 RSA 49

26 March 20, 2017 **50**
26.1 Digital Certificates 50

27 March 24, 2017 **52**
27.1 SSL and TLS 52
27.2 SSL and TLS Protocols 53
27.3 Master Secret 55

28 March 27, 2017 **55**
28.1 Web Security 55
28.2 Server and Client 57

29 March 29, 2017 **57**
29.1 Browser Security 57

30 March 31, 2017 **59**
30.1 Web Authentication 59

31 April 3, 2017 **60**
31.1 Malware 60
31.2 Viruses 61

31.3 Worms 63

32 April 5, 2017 **64**
32.1 Worm Development 64
32.2 Trojan Horse 64
32.3 Botnets 65

33 April 7, 2017 **66**
33.1 Network Security 66
33.2 Packet Routing 67
33.3 Local Area Routing 67

34 April 10, 2017 **69**
34.1 Internet Routing 69
34.2 Network Attacks 69
34.3 Defense Against Network Flooding 70
34.4 Firewalls 71
34.5 Firewall Approaches 72

35 April 12, 2017 **72**
35.1 Firewall Types 72
35.2 Firewall Strengths and Weaknesses 73
35.3 Domain Name System 74
35.4 DNS Security 74
35.5 Virtual Private Network 75

1 January 9, 2017

1.1 Introduction

Information security and privacy is the practice of preventing unauthorized access, use, disclosure, disruption, modification, inspection, recording or destruction of information. It is a general term that can be used regardless of the form the data may take. **Computer security** is information security applied to technology. Several topics will be covered in this course, including authentication, access control, communication security, malware, cryptography, and the social aspects of security.

2 January 11, 2017

2.1 Information Security

In terms of information security, we desire for our private data to be secure from others. That is, for our data to remain unchanged. However, we also desire access to our data when we wish. Information security follows the principles of confidentiality, integrity, and availability. this is referred to as the **Security Requirements Triad**. Confidentiality concerns preserving authorized restrictions on information access and disclosure, including the means for protecting personal privacy and proprietary information. Integrity concerns guarding against improper information modification or destruction, including ensuring information nonrepudiation and authenticity. Availability ensures timely and reliable access and use of information.

For companies, there may be different levels of data access. In terms of confidentiality, companies may wish to protect stored data (employee data, project data, meetings, company secrets, user information) and communications (within the company, with sister companies, with users). Integrity involves securing all stored data and communications, and is thus a requirement for all data. Availability includes access to data, communication, and services (for employees, users), including access to the company's website, email, network, and all other operations.

Computer security is defined as the protection afforded to an automated information system in order to attain the applicable objectives of preserving he integrity, availability, and confidentiality of information system resources. This includes hardware, software, firmware, information/data, and telecommunications. The focus of computer security is on the following three fundamental questions:

1. What assets do we need to protect?
2. How are those assets threatened?
3. What can we do to counter those threats?

2.2 Challenges to Computer Security

In computer security, potential attacks on security features must be considered. Procedures that are used to provide particular services are often counterintuitive. Physical and logical placement needs to be determined, and multiple algorithms or protocols may be involved. It is important to note that all an attacker needs is to find one single weakness to infiltrate the system. The developer needs to find all weaknesses. Furthermore, users and system managers usually do not see the benefits of security until a failure occurs. Security requires regular and constant monitoring, but is often an afterthought to be incorporated into a system after the design is complete since it is thought of as an impediment to efficient and user-friendly operation.

2.3 Computer Security Terminology

We now introduce the following terminology to be used in the class:

1. **Adversary (Threat Agent)** - An entity that attacks, or is a threat to, a system.

2. **Attack** - An assault on system security that derives from an intelligent threat; a deliberate attempt to evade security services and violate security policy of a system. Attacks may be a passive or active attempt to alter/affect system resources, and may be performed by insiders or outsiders. An insider is someone authorized to use the system, while an outsider is someone not authorized to use the system.

3. **Countermeasure** - An action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken. Countermeasures are actions taken to prevent (through cryptography), detect/respond (intrusion detection allowing shut down or tracing of intruder), and recover (through backup). Residual vulnerabilities may result from new vulnerabilities through introduced countermeasures. The goal is to minimize residual vulnerabilities.

4. **Risk** - An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result.

5. **Security Policy** - A set of rules and practices that specify how a system or org provides security services to protect sensitive and critical system resources.

6. **System Resource (Asset)** - Data; a service provided by a system; a system capability; an item of system equipment; a facility that houses system operations and equipment.

7. **Threat** - A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. Threats are capable of exploiting vulnerabilities and represent a potential security risk.

8. **Vulnerability** - Flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy. Vulnerabilities may lead to corrupted (loss of integrity), leaky (loss of confidentiality), or unavailable or slow (loss of availability) files.

2.4 Threat Consequences

There are four kinds of threat consequences. **Unauthorized disclosure** is a circumstance or event whereby an entity gains access to data for which the entity is not authorized. Unauthorized disclosure is a threat to confidentiality. The following types of attacks can result in this threat consequence:

- **Exposure**: This can be deliberate, as when an insider intentionally releases sensitive information, such as credit card numbers, to an outsider. It can also be the result of a human, hardware, or software error, which results in an entity gaining unauthorized knowledge of sensitive data. There have been numerous instances of this, such as universities accidentally posting confidential information on the Web.

- **Interception**: Interception is a common attack in the context of communications. On a shared local area network (LAN), such as a wireless LAN or a broadcast Ethernet, any device attached to the LAN can receive a copy of packets intended for another device. On the Internet, a determined hacker can gain access to e-mail traffic and other data transfers. All of these situations create the potential for unauthorized access to data.

- **Inference**: An example of inference is known as traffic analysis, in which an adversary is able to gain information from observing the pattern of traffic on a network, such as the amount of traffic between particular pairs of hosts on the network. Another example is the inference of detailed information from a database by a user who has only limited access; this is accomplished by repeated queries whose combined results enable inference.

- **Intrusion**: An example of intrusion is an adversary gaining unauthorized access to sensitive data by overcoming the system?s access control protections.

Deception is a circumstance or event that may result in an authorized entity receiving false data and believing it to be true.

- **Masquerade**: Could be an attempt by an unauthorized user to gain access to a system by posing as an authorized user; Trojan horse. It could also be someone guessing a user's bad password to obtain unauthorized access.
- **Falsification**: Altering or replacing of valid data or the introduction of false data.
- **Repudiation**: To successfully deny an attack. In this case, a user either denies sending data or a user denies receiving or possessing the data.

Disruption is a circumstance or event that interrupts or prevents the correct operation of the system services and functions.

- **Incapacitation**: A result of physical destruction of or damage to system hardware. It is possible to damage hardware with software, such as by disabling fans.
- **Corruption**: System resources or services function in an unintended manner through unauthorized modification. For instance, installing backdoors. Malicious software in this context could operate in such a way that system resources or services function in an unintended manner. Or a user could gain unauthorized access to a system and modify some of its functions. An example of the latter is a user placing backdoor logic in the system to provide subsequent access to a system and its resources by other than the usual procedure.
- **Obstruction**: The overload of the system or interference with communications through intentionally filling up logs to drain disk space. This could be achieved by disabling communication links or altering communication control information. Another way is to overload the system by placing excess burden on communication traffic or processing resources.

Usurpation is a circumstance or event that results in control of system services or functions by an unauthorized entity.

- **Misappropriation**: This can include theft of service. An example is a distributed denial of service attack, when malicious software is installed on a number of hosts to be used as platforms to launch traffic at a target host. In this case, the malicious software makes unauthorized use of processor and operating system resources.
- **Misuse**: Misuse can occur by means of either malicious logic or a hacker that has gained unauthorized access to a system. In either case, security functions can be disabled or thwarted.

3 January 13, 2017

3.1 Computer System Assets

The assets of a computer system can be categorized as hardware, software, data, and communication lines and networks. In this subsection, we briefly describe these four categories and relate these to the concepts of integrity, confidentiality, and availability:

Hardware equipment can be stolen or disabled, thus denying services. This is an example of a threat to availability.

Software on the other hand, can be deleted. Thus, programs will not be available to users. Confidentiality could be at risk should an unauthorized copy of the software be made. Furthermore, the integrity of the software could be affected, since a working program could be modified to either cause it to fail during execution, or to cause it to perform an unintended task.

Data availability could be at risk, since files can be deleted, denying access to users. Confidentiality should also be considered in terms of data, since an unauthorized read of data could be performed. Furthermore, data integrity could be at risk since existing files could be modified, and new files could be fabricated.

Communication lines could be at risk to availability, since messages could be destroyed, and communication lines or networks could be rendered unavailable. Confidentiality is also of particular concern, since messages could be read and the traffic patters of those messages observed. Integrity should be considered as well, since messages could be modified, delayed, reordered, or duplicated. False messages could also be fabricated.

3.2 Network Attacks

Networks are susceptible to both passive and active attacks. **Passive attacks** attempt to learn or make use of information from the system, but does not affect system resources. This may include eavesdropping and monitoring transmissions. Passive attacks are difficult to detect, and the emphasis is on prevention rather than detection. Two types of passive attacks are the release of message contents and traffic analysis. We would like to prevent an opponent from learning the contents of these transmissions.

Active attacks involve modification of the data stream. Our goal is therefore to detect them, and then recover. The four categories of active attacks are masquerade, replay, modification of messages, and denial of service. A masquerade takes place when one entity pretends to be a different entity. Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect. Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. Lastly, the denial of service prevents or inhibits the normal use or management of

communications facilities.

3.3 Security Design Principles

We note that there is no foolproof design, but we do have guiding principles:

- **Economy of mechanism** states that the simplest design results in a simple testing procedure in terms of security.
- **Open design**, as opposed to security through obscurity, claims that the design should not be secret. With more people knowledgeable of the system, there is a better chance to spot problems.
- **Complete mediation** relates to checking every access point. It is related to caching permission checks. For instance, writing to files through permissions at intermediate steps.
- **Least privilege** states that the less the better. We should give as little privilege as necessary. An example would be root access provided by Ubuntu.
- **Fail-safe defaults** refers to the lack of access that is the default. For instance, newly created users should have strong default passwords.
- **Psychological acceptability** states that users should not consider security mechanisms an enemy. The security features should not interfere with work, and should be easy to comprehend.

Other design principles include isolation, encapsulation, and modularity.

3.4 Methods of Attack

An **attack surface** refers to reachable and exploitable vulnerabilities. For instance, unnecessary services attached to public ports, services on the inside of a firewall, buggy PDF or Flash applications, email clients that automatically open files, and employees susceptible to social engineering, especially those with high access level. Attack surfaces can be distinguished into three separate categories, with **network attack surface** concerning vulnerabilities in network protocols or flawed designs in services, **software attack surface** concerning vulnerabilities in applications and the operating system such as through an insecure web server, and **human attack surface** concerning social engineering, human error, or an insider attack.

An **attack tree** is a branching hierarchical data structure that represents a set of potential techniques for exploiting security vulnerabilities. It is a graphical description of a threat, possible attacks due to the threat, and possible risks. The root node is the goal of the attack, and the children are possible ways to achieve that goal. Leaf nodes represent possible ways to initiate the attack. Nodes with the

same parent are alternatives to accomplish the same goal. We can append additional information to the attack tree. For instance, we can consider whether the options are possible or impossible, legal or illegal, easy or hard, and cheap or expensive.

4 January 16, 2017

4.1 Computer Security Strategy

For an organization, a computer security strategy may include a security policy, implementation of security measures (prevention, detection, response, and recovery), and evaluation.

At the very minimum, the security policy should be an informal description of desired system behaviour. For instance, it may include sections on confidentiality, integrity, and availability. It is more useful when it is a formal statement of rules and practices relate to security. These may include a password policy, a privacy policy, an internet usage policy, and a policy for the disposal of old equipment.

To enforce these policies, we can employ a manual procedure whereby a person checks practices for compliance. For example, In Canada, the federal Personal Information Protection and Electronic Documents Act (PIPEDA) regulates the collection, use and disclosure of personal information in the private sector. An organization subject to the law uses a **privacy policy** as part of meeting its obligations. A **privacy officer** oversees all the activities related to the development, implementation, maintenance and adherence to the organization's privacy policies and procedures, whereas an **Office of Privacy Commissioner** (OPC) oversees the compliance. We also note that some policy enforcement can be automated, as is the case with firewalls that can block access to the internet.

4.2 Password Authentication

A **password** is a shared secret between a user and a computer system (server). The purpose of a password is to prove one's identity. Passwords are very widely used as a form of authentication. In picking a password, we want it to be difficult to guess, but easy to remember. In the process of password authentication, the user first supplies their user identification, alongside their password. This pair must match a password table on the system. The system compares the pair against the information stored on the system. Successful authentication allows the user to access different parts of the system. The reason we need user identification is to determine the user's privileges.

Error messages can reveal sensitive information. For instance, if the user-password pair failed, then the error could indicate that no such user was found. This could reveal information about whether there was such a user to begin with. It is advantageous on the other hand, since it notifies the user that they entered an invalid username or password. In the event that the error message allows one to

click if they forgot their password, this can reveal information about the existence of a user.

There are many different methods for finding other peoples' passwords. They are grouped into two categories, with **guess-verify attacks** simply guessing and verifying based on some leaked information about passwords, and **keylogging**, **phishing**, and **social engineering** obtaining the password directly. To avoid this, we need to ensure that the passwords are hard to guess. We can increase password strength to accomplish this. **Entropy** is used to measure the strength of passwords. The unit of entropy is a **bit**. A password with n-bit entropy requires 2^n guesses. That is,

$$Entropy = \log_2(\text{number of guesses required}).$$

In information theory, entropy is a measure of unpredictability of information content. If a set S has N elements, then the entropy of finding an element is $\log_2 N$.

Example. *Determine the entropy of the set of all three digit numbers,* $\{000, 001, 002, ..., 999\}$.

We note that the value at each position can be the numbers 0 through 9. Therefore, the size of the set is 10^3. Entropy is therefore $\log_2(1000) = 9.96$ bits.

Entropy can be though of as a guessing game of a person in the classroom, where any question that has a "yes/no" answer can be asked. The objective is to determine the person that the mediator is thinking of. The minimum number of questions required to be certain that one is correct is $\log_2$(the number of people). To accomplish this, we always ask a question that results in the elimination of half of the possibilities. Entropy is the least number of binary questions that is required to find the element, assuming that all elements are equally likely. If we consider sets S_1, S_2, and S_3, all with the same number of elements, then it is clear that all of these sets have the same entropy.

We note that bits are also used to represent a binary digit in computer systems. For instance, 8 bit ASCII code. To determine a password's entropy, we consider the case with 6 character passwords. If the 6 character password consisted of only lowercase letters, then we have and entropy of $\log_2\left(26^6\right) \approx 28$ bits. If we included 10 digits, we obtain an entropy of $\log_2\left((26+10)^6\right) \approx 31$ bits. Including 14 symbols as well, entropy becomes $\log_2\left((26+10+14)^6\right) \approx 34$ bits. However, passwords must be easy to remember, and thus usually contain meaningful words. If we consider the number of english words than an average person uses, this results in around 20000 to 70000 words, resulting in around $15-16$ bit entropy.

To create a strong password, one should avoid using full words and names, as "dictionary attacks" can be used to guess passwords. It is also advised that one creates passwords of twelve characters or more with mixed types of characters. One should also use different passwords for each website, and can also use a password manager such as LastPass or SplashID to organize and protect passwords. The password can also be tested for complexity with a password checker. These online tools can estimate password strength, but are reliable insofar as the underlying dataset and algorithms used.

4.3 Guess-Verify Attack

An attacker's goal may be to target a specific user, or any user. It is important to evaluate security against an any user attack. In guess and verify attacks, the attacker can simply loop their guessing and verifying either offline or online. In **online guessing**, the attacker sends a guess and receives a response in real time. To counteract this, an account can be blocked after three tries. This is bad for a forgetful user, but it does slow down the attacker. One can also slow down the attacker by slowing the response after every time they enter incorrectly. While this can be effective, it is harder to block when an attacker commands a botnet. We can also counteract online password guessing by using CAPTCHA's (Completely Automated Public Turing Test to Tell Computers and Humans Apart) to detect bots. In **offline guessing**, the attacker obtains a password file/table and tries guesses against that file. If passwords are stored in cleartext (plaintext), then the attacker has already succeeded. Therefore, passwords should never be stored in cleartext.

5 January 18, 2017

5.1 Guess-Verify Attack Cont'd

We can **encrypt** the password file using some secret key. However, the problem remains that the secret key needs to be stored somewhere. If the attacker has obtained the password file, it is likely they will also have access to the key. Thus, we may attempt to use an unencrypted password file, but store the password has instead through storing the hash-function(password). The **hash function** should be difficult to invert. That is, given a hash, it should be difficult to find a matching password. A cryptographic hash function is a decent choice, although slower cryptographic hash functions provide a better alternative.

A hash function maps arbitrarily-sized data to fixed-length data (hash). It is generally quick, deterministic, and evenly distributed. Depending on its uses, it can require other properties (for example, a hash table). A **cryptographic hash function** is a hash function with extra features to ensure that it is difficult to crack. It must be infeasible to invert, infeasible to find two (x, y) such that $h(x) = h(y)$, and it should appear random by passing pseudorandom tests. Essentially, we want a brute-force search to be the only viable strategy.

Consider for instance, the hash function

$$h(x) = x \mod (10).$$

It is easy to find an x such that $h(x) = n$ for any arbitrary n. For example, $13, 23, 4543$. Now consider the hash function

$$h(x) = [\text{sum of digits of x}] \mod (10).$$

This is marginally better, but it is still easy to determine $x : h(x) = n$ for any n. It is difficult to define a good hash function.

In old UNIX systems, the password file was given by */etc/passwd*. It was world readable but only root can write. This password file contained user identification, hashed passwords, and other useful information about the user. This design makes it fairly easy to obtain a password, since any user has access to it. The UNIX shadow password system still stores the password file in */etc/passwd*. It is also world readable with only root capable of writing to it. While it similarly contains user identification, it does not contain actual passwords. The passwords are stored in a shadow file: */etc/shadow*, where only root can read/write. It offers a bit more security for little to no cost.

In offline dictionary attacks, the attacker first obtains a list of password hashes. The attacker then uses the guess-verify strategy with optimizations, such as precomputing the hash for every word in a dictionary and storing these results in a hash dictionary. While this is a slow process, it only needs to be done once. Cracking can now be very fast, and offers the ability to crack all users at the same time. This strategy of offline attack however, only works if passwords are dictionary words (or derived from leetspeak). One should recognize that dictionaries could contain common passwords, and the attacker could also be aided by wikipedia dumps, free/stolen textbook dumps, facebook dumps, etc. The hash dictionaries used include a **space-time tradeoff** to improve running time, since an algorithm can run faster if more memory is available. Dictionaries can become too big to be practical. The solution then is to decrease space, and increase running time slightly. We can also use rainbow-tables, which operate similar to a hash dictionary, with much less space requirement, but only slightly longer cracking running time.

5.2 Defending Against Guess-Verify Attacks

To compute a hash of a password, we can add some salt first, so that the password hash becomes

$$h(\text{salt}|\text{password}),$$

where | means concatenation. **Salt** is a string that is randomly generated when the password is created and when the password is changed. The salt is not secret (cannot be), and is stored in the password file together with the hash. Thus, even when two users have the same password, they will likely have different hashes, thus making offline dictionaries attacks much more difficult. When we need to verify the password, we simply need to give the slow hash function the salt and the password, to compare the result with the stored hashed password. To employ a dictionary attack, the attacker must now precompute the number of dictionary words multiplied by the number of possible salts. For instance, if the salt is only 12 bits (around 2 extra characters), the dictionary size needs to increase by a factor of 4096. Current best practices use a salt size around $32 - 64$ bytes. By using a secure random number

generator, the salt is unique to each user. Along with a good hash function that is slow and cyptographic based (such as argon2, scrypt, bcrypt, and pbkdf2), this provides the best defense against dictionary based attacks.

There are many password cracking programs. They can be used for attacks, or for password recovery. They include Brutus, RainbowCrack, THC Hydra, Cain and Abel, etc. A good password should not contain any words in the attacker's dictionary. It should also contain digits and special characters at an appropriate length (> 14 letters). The password should be close to random, and yet should be memorizable. It is therefore advisable to either memorize a 14 word sentence with letter substitutions, or use a password management program. To convince users to select good passwords, it is possible to enforce this by running a cracker on it and rejecting the password if it is cracked.

5.3 Keystroke Logging, Phishing, and Social Engineering

Keylogging can be done through hardware, such as USB keyloggers, electromagnetic emissions (wired keyboard), wireless sniffing (wireless keyboard), and acoustic keyloggers, or be accomplished though software, such as trojans, spyware, and virtual machine rootkits.

Phishing attacks aim to obtain sensitive information such as usernames and passwords by pretending to be a trusted entity via electronic communication. This is generally accomplished through emails. One must also be aware of **pharming**, since it is difficult to tell if the website is fake. It is therefore important to procure a good home router.

Social engineering is accomplished through psychologically manipulating people to disclose confidential information or to perform some actions. There are many forms, including phishing, pretexting (inventing scenarios or pretending to be of authority), quid pro quo (offer services), and baiting (through an infected USB).

6 January 20, 2017

6.1 Password Protection

There may be many ways for an attacker to obtain a password file. For instance, there could be weaknesses in the operating system as a result of not applying the latest security patches. The cause could also be accidental, as in the case of changing permissions of */etc/shadow* to world readable. The attacker could also obtain the password file by sniffing passwords through network traffic, or accessing it from a weak backup. To prevent this, we need to protect from phishing and other social engineering attacks. This starts by ensuring that the user is aware. This includes not clicking links in emails. We can also be protected by the server, through keeping our software updated and implementing security mechanisms. We can also use stronger authentication measures.

Two-factor authentication asks the user to supply a username and password along with another form of evidence regarding the user's identity. This could be a **one time password** (OTP). There are problems with this approach, such as synchronization issues, the loss/theft of a device, and the inconvenience associated. We can also protect from keylogging by removing the code-keyboard relationship. This can be accomplished through a virtual dynamic keyboard, whereby the user must use their mouse to enter the password. If a keyboard is used, the mapping changes every login. While this protects against keylogging, this only brings about the issue of mouse-logging and screen capture. We can make use of a scramble pad.

In the following network attacks, an attacker obtains sensitive information from a network between the user and the server. A **sniffing attack** is a passive attack, where the attacker captures the password while it is being transferred through a network, so they may apply a password cracker offline. In a **replay attack**, the attacker captures the password and later re-plays the same password through the network to the server. This is an active attack. Countermeasures to circumvent these attacks include the use of a one-time password. However, this requires extra hardware. We can also attempt to use a software only solution, employing a **challenge-response** based protocol that adds randomness to each authentication. It provides nonce to the procedure of authentication, as the password is not sent directly. In this case, a user would want to prove to the server that they know the password. However, they do not want to reveal it. In this challenge-response approach, the server provides the user with a unique challenge to be used as the salt for every authentication request. This must be a secure random number. The user then applies the cryptographic hash function to the salt concatenated with the password. This ensures that sniffing and replay attacks are not possible. Challenge-response systems are used by secure authentication systems and are usually based on cryptography. CAPTCHA is an example of a non-cryptographic challenge-response mechanism.

A **man-in-the-middle attack** (MITM) is an active real-time attack, and is performed when the attacker is placed in the middle of the communication between the user and the server. It is a very real problem, since it is fairly easy to execute in some environments, especially on public networks with low security. MITM can be accomplished through malicious software such as man-in-the-browser procedures. MITM are hard to detect and hard to prevent. We can limit its effectiveness by using *https*, considering certificate warnings, not downloading random programs, not clicking on attachments, obtaining a good home router, and using a VPN. Cryptography alone is often not enough to stop these kinds of attacks. **Out-of-band** authentication is usually required.

Password re-use is not a good idea, since an attack on a less secure site may be performed to use the password on a more secure one. One should be aware that passwords do not protect against attacks. It is therefore vital to use either strong passwords, or a password manager that securely stores passwords. It is important to note however, that password managers are not perfect, and do indeed introduce

new vulnerabilities. Online password recovery for when a user has forgotten their password also introduces a potential attack surface, since a weak recovery procedure means an easily obtainable password. Since much of one's personal information can be collected on the internet, it may be insecure for password recovery to provide secret hints, or ask secret questions where the answers could be easily found.

7 January 23, 2017

7.1 Graphical Passwords

While text passwords are the most widely used, it is also the methods with most techniques devised and most research dedicated. There are drawbacks however. If the password is easy to remember, then it is easy to guess. If it is hard to guess, then it is hard to remember. Users tend to pick simple passwords which are vulnerable to dictionary attacks. If they pick complex passwords, they either need to write them down or reuse them. To avoid these disadvantages, we can make use of **graphical user authentication** (GUA).

GUA can be separated into three main categories:

- **Recognition** based forms such as **Deja Vu** require users to memorize a portfolio of images during password creation and then requires users to identify their images from among decoys to authenticate. All images are random art generated from a seed. To create the password, a set of random art is selected and tested against decoys. During authentication, the system presents multiple sets with decoys. The user then has to select their image. However, this form of authentication takes longer to memorize and authenticate, and the server cannot store hashes of passwords. **Passfaces** use faces in place of random art. A **convex hull click scheme** is also a recognition based form of graphical password that attempts to reduce the **shoulder surfing** threat. Registration requires the memorization of around 10 icons, of which around 3 are shown with many decoys when the password is required. The user then selects the triangle formed by the location of the 3 icons. This is a lengthy authorization process.

- **Recall** based forms such as **Draw-a-Secret** require users to create or draw something during password creation for later reproduction during authentication. The creation of a password involves drawing a pattern. During the authentication process, the user to asked to redraw the pattern. Problems include shoulder surfing and the small search space.

- **Cued-Recall** based forms such as **Passpoints** is the same as recall, with additional cues to aid the recall process. Registration is performed by clicking on 5-8 items in an image. When the password is required, the user clicks in order the same remembered items.

GUA offers potential advantages. For instance, it a set of images is large, it could result in stronger passwords. It is also more difficult for users to write down the password. In terms of usability, GUA require a lengthy registration and authentication process that is not feasible for multiple passwords. There is also a trade-off between tolerance to account for user error and security. It is difficult to evaluate these forms of security, since we cannot really perform entropy measurements, and shoulder-surfing remains a big threat, with resistant approaches being more time consuming. There may also be technical issues, such as availability on mobile devices (due to screen size or touch screen instead of keyboard or mouse) and bandwidth or storage requirements.

8 January 25, 2017

8.1 Vulnerabilities

We note below the main forms of attacks on passwords:

1. **Offline dictionary attacks** are used when the attacker attempts to gain access to a password file. To prevent against it, one should protect the password file, detect theft of the file, and reissue new passwords.

2. **Specific account attacks** are used when the attacker guesses the password for a particular user. This can be prevented by timeouts and limited trials.

3. **Particular user attacks** occurs when the attacker uses social engineering or guesses the password for a particular user. This can be prevented through user training and the use of complex passwords.

4. **Popular password attacks** are used by attackers when they guess popular passwords to attack multiple accounts. One can stop this by disallowing popular passwords and detecting multiple submission patterns.

5. **Workstation hijacking** occurs when the attacker waits until the user walks away from their workstation. To prevent this, one can use automatic screen locking and advanced user behaviour detection.

6. **User mistake exploits** are used when the user writes down a password. The attacker can obtain access through this information. To prevent this, we can utilize user training or two-factor authentication.

7. **Password re-use** can be exploited by attackers, since their attack on one system can provide access to another. We utilize user training to guard against this.

8. **Monitoring** is when an attacker uses network sniffing, shoulder surfing, or keylogging. To protect passwords against this, one can encrypt communication and add nonce. It would also be beneficial to train users.

There are many different ways to authenticate a user. For instance, we can use something that the user knows (such as a password, PIN, and answers to prearranged questions), something that the user possesses (tokens such as barcodes, smartcards, electronic keycards, or physical keys), something that the user is (static biometrics such as fingerprints, retina, or face), or something that the user does (dynamic biometrics such as voice pattern, handwriting, typing rhythm, or gait).

8.2 Token Based Authentication

A **physical token** is something that the user physically has with them. This could include SecurID, and student ID card, a credit card, or a passport. This allows for quick identification and hence authentication. We want this physical token to be easy to use, cheap to make, but difficult to forge. Some common physical tokens include barcodes, magnetic stripes, smart cards, RFID, SIM cards, and smartphone authenticators.

Barcodes are printed labels that are used to improve efficiency in retail. It uses Automatic Identification and Data Capture (AIDC), and is scanned optically. The barcode reader scans the pattern and then converts this into ode. Barcodes see widespread usage for price tags, boarding passes, medical systems, and ID cards. The Universal Product Code (UPC) is a specific barcode system that most grocery stores use. 1D linear barcodes encode data in line widths and spacing. In contrast with the 1D barcodes which can store only limited data, 2D matrix barcodes come in many forms and hold greater data capacity. Modern phones can read common varieties, such as QR codes. **QR codes** are used for advertising, retail packaging, and virtual stores. They are easy to sync, and can also be used to facilitate two-factor authentication by providing a time-based one-time password.

However, since they are easy to read and easy to generate, QR codes may lead to information leakage and forgery respectively. Since QR codes often encode a URL, this can be exploited to send a user to a hacker's website. Phishing attacks on QR codes (QRishing), drive by download exploits such as towelroot, and browser exploits (such as access to camera and mic, access to browser data and emails, access to botnets and DDOS) can all be performed. QR codes can also be used to send SMS to premium numbers. Furthermore, cross-site scripting (XSS) can be exploited.

9 January 27, 2017

9.1 Token Based Authentication Cont'd

Barcodes may also be used for authentication, such as boarding passes that utilize 2D barcodes. These are created during check in and scanned before boarding. The main use of this technology is to improve efficiency. Originally, this had no cryptographic protection. In 2006, a computer science student created the Northwest Airlines Boarding Pass Generator that allowed users to fake boarding passes. This allowed access to secure areas of the airport, free upgrades to business class, and avoidance of the no-fly-list. The site was promptly taken down, and the DHS/TSA devised a **Boarding Pass Scanning System** (BPSS). The 2D barcodes now used on boarding passes included cryptographic protection. This includes a digital signature generated by the airline (private key) which the TSA can validate when verifying the signature by using the airline's public key. We note from this that barcodes alone do not provide secure authentication or confidentiality.

Magnetic stripe cards were developed in the 1960s and store information in an easily readable manner. Scanners can easy read debit/credit cards, driving licenses, and student IDs. These magnetic cards consist of a plastic card with magnetic tape that contained a plastic film, and was invented by an IBM engineer. These cards were standardized by the International Organization of Standardization (ISO), and affected the size of the cards, the location of the stripes, and the data format of the encoded information. Multiples tracks are used to store the card holder?s full name, the account number, the format information, the account number, the expiration date, and information to perform a parity check. In terms of security, they are similar to barcodes in that readers, writers, and blank cards are cheap to make and reproduce. To attain security through cryptography, we need additional information such as a PIN number.

Smart cards include a computer chip inside, with read only and read/write memory. the microprocessor reads and writes to memory and performs processes. The technology used is the same as in SIM cards, and is tamper resistant. These cards are capable of computation and communication, as they can execute protocols. Smart cards are often used with a PIN for two-factor authentication. They can be contact, contactless, hybrid, or dual, since these cards are application flexible, provide strong security and storage, and are resistant to magnetic fields.

There are still vulnerabilities with smart cards however. This can be system design flaws such as in protocol design and implementation, or through eavesdropping and communication tampering between the card and the reader, or between the reader and the backend. For instance, the card can be forged, the communication with the terminal can be intercepted, the terminal could tampered with or faked, and the communication between the terminal and the bank could be intercepted. EMV is a standard system for smart card payments where the PIN is stored on the card and encrypted. While this sounds good in theory, ATMs still need to generate

a nonce called an unpredictable number for each transaction. If this procedure is implemented poorly, attacks are still possible.

10 January 30, 2017

10.1 Token Based Authentication Cont'd

Smart cards also store **secret keys** and perform cryptographic operations. Extracting keys is difficult, but not impossible through reverse engineering. There are **intrusive methods** to achieve this, such as through destroying the card. This however, is expensive and time consuming. Each card requires the same amount of effort, but this works on any card, without prior knowledge of hardware or software. **Non-intrusive methods** on the other hand, leave the card intact. This can be time consuming for the first card, but once a method is discovered, it can be applied to multiple cards. One may benefit from prior knowledge of the hardware/software. **Side channel attacks** concern information gained from the physical implementation of a cryptographic system. For instance, measuring timing, power consumption, electromagnetic leaks, and even sound.

SIM cards (Subscriber Identity Module) are types of smart cards used in mobile devices and are issued by a network provider. It stores network specific information to identify and authenticate the user (including a cryptographic key). It also stores preferred roaming partners, emergency numbers, and the user's contact list. Each SIM card is uniquely tied to a record in the database of subscribers maintained by the network provider, through the following information:

- **ICCID** (Integrated Circuit Card ID) is a unique 20-22 digit hardware number
- **IMSI** (International Mobile Subscriber Identity) is a number unique to each user thaht identifies the owner?s country, network, and personal information.
- **Ki** is a 128-bit secret authentication key that is unique to each user. It is used to authenticate a phone to a mobile network.

Many cards also store a personal PIN that is required to access information on the card and a PUC (Personal Unlock Code). According to the GSM Challenge Response Protocol, to connect to a network through the SIM card, the phone first sends the IMSI to the base station. The base station then looks this up in its data base and sends back a challenge C, which is a random 128-bit number. The phone computes and sends back a response R where

$$R = A3(C, Ki),$$

where $A3$ is a proprietary encryption algorithm. Thus, R is an encrypted version of C. The base station then computes its own response R'. If $R = R'$, then the phone is authenticated. Communications between the subscriber and the base station can

be encrypted. GSM uses proprietary cryptographic algorithms such as $A5/1$, $A5/2$, and $A5/3$. However, they have all been reverse engineered and found to be weak. Security is obtained through obscurity, but this does not work.

Radio Frequency Identification (RFID) are small transponders that transmit identification information. They provide functionality similar to barcodes. They are composed of a chip that stores information and performs basic computations, and a coiled antenna that receives and transmits information while also supplying power (passive chip). It can also include a battery (active chip). The range is from a few centimeters to several meter. While barcodes are lighter, cheaper, and easy to create, RFID allow for potentially greater scan distance, for faster scans even outside of line of sight, greater security, increased durability, and increased data capacity. RFID can also update stored information.

Some uses of RFID include product tracking and tagging, animal tracking, car key fobs, book sorting in libraries, and electronic toll transponders. However, RFID are not without their share of privacy and security concerns. Furthermore, these problems are difficult to solve since one must consider disciplines of signal processing, hardware design, supply-chain logistics, privacy rights, and cryptography. Some attacks include sniffing, cloning, modifying, and disabling.

ePassports are modern passports that have an embedded RFID chip. The Canadian ePassport for instance contains a passive chip with a $10cm$ scanning distance. It stores personal information and a digitized picture, but does not yet contain biometrics. To prevent from tampering, it is equipped with **passive authentication** (PA). During production, data is digitally signed by the government, and during verification, the data signature is verified. To prevent unauthorized reading, it features **basic access control** (BAC) for a scan distance of at most 10 cm. The birth data, expiration date and passport number form a key which needs to be fed into the reader to access the data. To prevent cloning, it includes **active authentication** (AA) where a unique public/private key for each passport RFID chip is created. The private key cannot be read and therefore cannot be copied. Challenge-response protocol verifies that the passport has the correct private key, and the public key is signed by the government and included in the PA.

11 February 1, 2017

11.1 Biometric Authentication

Biometric authentication is an attempt to authenticate an individual based on unique physical characteristics and is generally based on pattern recognition. Compared to passwords and tokens, biometric authentication is both technically complex and expensive. While it is used in a number of specific applications, biometrics has yet to mature as a standard tool for user authentication to computer systems. This may be due to the varying levels of cost for a comparable level of accuracy in bio-

metric authentication. A number of different types of physical characteristics are either in use or under study for user authentication. This includes:

1. **Facial characteristics** are the most common means of human-to-human identification. Thus it is natural to consider them for identification by computer. The most common approach is to define characteristics based on relative location and shape of key facial features, such asthe eyes, eyebrows, nose, lips, and chin shape. An alternative approach is to use an infrared camera to produce a face thermogram that correlates with the underlying vascular system in the human face.

2. **Fingerprints** have been used as a means of identification for centuries, and the process has been systematized and automated particularly for law enforcement purposes. A fingerprint is the pattern of ridges and furrows on the surface of the fingertip. Fingerprints are believed to be unique across the entire human population. In practice, automated fingerprint recognition and matching system extract a number of features from the fingerprint for storage as a numerical surrogate for the full fingerprint pattern.

3. **Hand geometry** systems identify features of the hand including shape, lengths and widths of fingers.

4. **Retinal pattern** is the unique pattern formed by veins beneath the retinal surface is. A retinal biometric system obtains a digital image of the retinal pattern by projecting a low-intensity beam of visual or infrared light into the eye.

5. **Iris** structure is another unique physical characteristic that can be used for identification.

6. **Signatures** reflect the unique style of handwriting of the individual. However, multiple signature samples from a single individual will not be identical. This complicates the task of developing a computer representation of the signature that can be matched to future samples.

7. **Voice** patterns are more closely tied to the physical and anatomical characteristics of the speaker. Nevertheless, there is still a variation from sample to sample over time from the same speaker, complicating the biometric recognition task.

The cost vs. accuracy of these biometric systems can be compared. For instance, passwords are among the cheapest solution, and can range from low to high accuracy. Voice, face, signature, and hand offer low accuracy at increasing cost, fingerprint and retina offer medium accuracy at increasing cost, and iris offers high accuracy at a great cost. To achieve authentication, the system needs to compare whether the

user matches with the biometric data given. If there is a one-to-one match, then the user is authenticated. For identification, the system must determine to whom the biometric information belongs. This involves one-to-many matching.

For biometric applications, we generally require the characteristic possess universality (every person should have this characteristic), distinctiveness (each person should have a noticeable difference in their characteristic), permanence (the characteristic should not drastically change over time), and collectability (the characteristic should have the ability to be effectively determined and quantified).

11.2 Fingerprint and Iris Scanning

In **fingerprint scanning**, a reader takes a sample of one's biometric information and stores this into a feature vector of some form. A comparison algorithm is then used to determine whether or not this matches with a reference vector. In fingerprint scanning, minutia points are located to produce a minutia map that is converted into a data stream. The position and orientation of ridges on the finger are noted. In a typical minutiae-matching algorithm, the algorithm first uses local minutiae descriptors to coarsely align two fingerprints and then computes a global **similarity (match) score** based on minutiae correspondences. The final true or false decision of whether this fingerprint matches the user is based on comparing this similarity to some threshold. There can be two types of errors: **false match** and **false non-match**. The accuracy is determined by comparing either the false match ratio, and the false non-match ratio. The threshold can be adjusted to increase the accuracy of one at the cost of the other. Note then that an increased threshold can lead to greater security, but at the cost of decreased convenience. **Iris recognition** is accomplished by segmentation of the iris, then the normalization of this image. This is converted into data for the *iriscode*, which can later be compared.

It is relatively simple to forge a fingerprint out of a glue-like substance if a real finger is available. It is also theoretically possible to create a fake fingerprint from stored biometric data. The same is true of iris scans. To combat fake fingerprints, **liveness detection** can be used by monitoring measures of vital signs in the finger, pulse, perspiration, deformation, and odors. Similarly, irises can be checked for pupil dynamics.

12 February 3, 2017

12.1 Handwriting Biometrics

Biometrics can be distinguished as **physiological**, as in the case of fingerprints, iris, face, and hand geometry, and has **behavioural**. Behavioural is separated into **passive**, which includes typing pattern, walking gait, and signature, and **active**, as in response to a challenge, the playing of a game, or the act of writing.

Handwriting biometrics is advantageous because it is familiar, has historical acceptance (since it is similar to signatures), and its capture is not invasive. This process can be either static or dynamic, depending on whether a picture of the signature is taken or the entire process is taken. In both cases, a digitizer is required. **Static** methods measure the width, height, aspect ratio, and area. However, static handwriting is not secure. **Dynamic** methods on the other hand take into account pen timing (such as when the pen lifts), and is a promising area of research. However, this requires consistency on part of the user.

To initiate dynamic handwriting biometrics, the user writes sample phrases. Each sample is represented by a set of attributes such as shape (static analysis), and dynamics (pressure, timing, velocity, acceleration, positions). This is combined into a single feature vector that is representative of the user's handwriting style. During verification, the user is asked to write a phrase (challenge) where attributes are collected from a digitizer. This is matched against the stored feature vector.

12.2 Biometric Assessment

Note that there may also be multi-factor multimodal biometric systems that simultaneously use multiple biometrics to authenticate a user. Biometrics is advantageous since there is no need to memorize, or a need to carry a token. It could also potentially provide strong identification and cannot easily be shared. However, it cannot be changed, and there may be privacy concerns. There could be **covert collection** of publicly accessible data such as face scans, fingerprints, and voice. This violates the privacy principle that people should be informed when their personal information is being collected. **Secondary information** can also be of concern, since an iris scan can reveal a person's health, and a fingerprint can reveal a person's employability. This violates the principle that personal information should only be collected for a clearly identified purpose. **Cross-matching** is another cause of concern, since the data could be collected for one purpose, and later used for a different purpose. This goes against the principle that personal information should only be used for the purpose for which it was collected.

The **Privacy Impact Assessment** is a process intended to help organizations consider the impact that a new or substantially modified initiative can have on people's privacy, especially when personal information is being collected. The process is mandatory in the public sector. Federal institutions proposing a program, policy or service with implications for privacy are required to submit a Privacy Impact Assessment for review. Advice and recommendations for strengthening privacy safeguards are then provided to the institution.

13 February 6, 2017

13.1 Access Control

Information security is about preventing unwanted access, but it is often easier to define in terms of what we wish users to be able to access. For instance, a user Bob may create a file, edit this file, and delete their file. Another user Alice can read her own files, and the admin can delete Bob's files and log him out. Access takes the general form of

Subject can perform an **Action** on some **Object**.

Objects are any resources to be protected, such as files, memory, devices, and processes. **Subjects** are the users interacting with the system. Typical classes include the owner, group, or world. Processes interact with the system on the user's behalf. **Actions** or **Access Rights** are the possible interactions on objects by subjects. This can depend on the object types, and includes operations such as read/write on files and the ability to stop processes.

To implement access control, one may write a function *isAllowed* that takes as arguments a subject s, an object o, and an action a. There can then be some implementation such as a lookup in a database that returns true if s is permitted to perform a on o, and returns false otherwise. To complete this type of implementation, the access control language must be expressive, and the policy must be enforceable.

Access control policies fall under three main categories:

1. **Discretionary Access Control (DAC)** is based on ownership and is the most common policy. The owner can do anything, and has the right to provide or remove access rights. This is often accomplished using an **access matrix**. One dimension of this matrix consists of identified subjects that may attempt data access to the resources, while the other dimension lists the objects that may be accessed. Each entry in the matrix indicates the access rights of a particular subject for a particular object. This is not very practical since the matrix can get big yet sparse. Instead, we can use **access control lists** or **capability lists** to list the users and their permissions under each file, or list each file along with the permissions of the associated user respectively. An **authorization table** sorted by subject results in the capability list, whereas an authorization table sorted by object results in an access control list.

2. **Mandatory (MAC)** is based on clearances and sensitivity levels. This is popular in the military, with clearance labels such as high, low, and no, or top secret, secret, and public. These are automatically defined by the system, as opposed to DAC where the user defines the permissions.

3. **Role-Based (RBAC)** is based on roles. As opposed to assigning permissions to an individual, permissions are assigned based on roles that an individual may have.

13.2 Unix/Linux File System Security

This is an example of discretionary access control. There is an infrastructure of filesystem access control for files and directories, mounted filesystems ($/mnt/profs$), devices ($/dev/cdrom$, $/dev/usb$, $/dev/fb0$), sockets, pipes, and memory ($/sys$ and virtual filesystem mapping kernel subsystems). The Linux file system is a tree of directories, where each directory has links to zero or more files and directories. Every file is owned by a user and a group, with permissions often displayed in compact 10-character notation. That is, a d or a $-$ depending on whether we are dealing with a directory or file, followed by the 9 permission characters, followed by the owner, then the group owner.

Remark. The owner can always change the permissions.

Each unique user has an identification number (uid), where 0 is root (superuser). A member of a primary group is identified by a group ID. Every user belongs to a specific group, with the permissions offering 12 protection bits. 9 bits specify the read, **w**rite, and e**x**ecute permission for the owner of the file, members of the group, and all other users (world). The owner ID, group ID, and protection bits are part of the file's inode. We distinguish permission for files with that of directories in that directories are preceded by d, while files are preceded by $-$ before the 9 bits of permission for each entity are specified. For directories, permission bits are interpreted slightly differently. The **read** bit allows listing of file/directory names, the **write** bite allows for the creation and deletion of files in the directory, and the **execute** bit allows entering the directory and obtaining properties of files in the directory. Note that not all combinations make sense however, as in the case of reading without executing.

The permission check algorithm for a given $user$ and $filepath$ can be determined by first checking all parent directories in the path for execute permission. Then, we check whether the $user$ is the same as $file.owner$. If these match, then we use $file.userPermissions$. Otherwise, if $file.group$ is in $user.groups$, then we use $file.groupPermissions$. If these cases do no match, then we simply user $file.worldPermissions$.

In the Linux file system, Set User ID (**SetUID**) is only on executable files. The system temporarily uses rights of the file owner in addition to the real user?s rights when making access control decisions. This enables privileged programs to access files/resources not generally accessible. Set Group ID (**SetGID**) on executable files has a similar effect to SetUID, but for groups. On directories, new files/subdirectories will have the same group. The **Sticky Bit** (12th bit) when applied to a directory, specifies that only the owner of a file in the directory can

rename, move, or delete that file. This is usually set on */tmp* and */scratch* type directories. **Root** refers to the superuser with $UID = 0$. They are exempt from usual access control restrictions, and have system-wide access. This is dangerous, but necessary, and actually acceptable with good practices. Becoming root requires a password:

- **su** requires the root password. This command changes the home directory, PATH, and shell to root. This also leaves environment variables intact.
- **su -** command acts as if one is logged in as root.
- **su - user** allows one to become someone else $< user >$.
- **sudo command** requires the user password. This allows one to run a command as root. This is the recommended option, since it leaves an audit trail.

Permissions are changed with *chmod* or via a GUI. Only the file owner or root can change permissions. If a user owns a file, the user can use *chgrp* to set its group to any group of which the user is a member. root can change file ownership with *chown* (and can optionally change the group in the same command). *chown*, *chmod*, and *chgrp* can take the $-R$ option directly after the keyword to recursively apply changes through subdirectories. For *chmod*, any files for which the permissions are to be applied are specified at the end. If no files are specified, this means all files are affected. + followed by read, write, or execute indicates the addition of permission, while − indicates removal of permission. This is optionally preceded by u, g, or o to indicate whether this affects the owner, group, or world. If it is not preceded by any combination of the above, then this applies to everyone. *chown* is specified by naming the new owner of the following directory.

While the Unix standard/basic permissions are great, it is not perfect. It is limiting and not expressive enough. For example, user Bob cannot easily give user John read access to his files. Most Linux based operating systems support POSIX ACLs, which build on top of traditional Unix permissions. Several users and groups can be named in ACLs, each with different permissions. This allows for much finer-grained access control. Unix standard is also built such that each ACL is of the form $type : name : rwx$ where *type* is the user or group, *name* is the user name or group name, and rwx refers to the bits set. This means that *setuid*, *setgid* and *stickybits* are not possible.

The commands $getfacl$ lists the ACL for a file, while $setfacl$ assigns ACLs to a file/directory. Any number of users and groups can be associated with a file. This is accomplished by read, write, and execute bits. A file does not need to have an ACL. A directory can have an additional set of ACLs, called **default ACLs**. Default ACLs will be inherited by files and directories created inside of the main directory. Subdirectories inherit the parent directory's default ACLs as both their default and their regular ACLs. Files inherit the parent directory's default ACLs only as

their regular ACLs, since files have no default ACLs. A logical AND operation is performed on the inherited permissions for the user, group, and other classes with the traditional Unix permissions specified in the file creation procedure.

14 February 8, 2017

14.1 NTF

Each file and directory in this file system has an owner and zero or more **access control entities** (ACE). The ACE format is:

<principle><operation>(allow|deny),

where *principal* is the user or group, and *operation* is read, write, execute, full control, list, or modify. ACEs support inheritance, since the directory ACEs can propagate to children. This is similar to ACLs in UNIX, but NTFs support *deny* ACE entries, while UNIX only has *allow* ACL entries. UNIX is also limited in that only members of the group can have permissions different from world, and there is no way to specify permissions for a single user other than the owner. Furthermore, the NTF file permission algorithm checks only the ACEs of the file, while UNIX checks the entire path. While NTFs are more expressive, they are also more complicated.

15 February 10, 2017

15.1 Role Based Access Control (RBAC)

Role based access control is a modified DAC system. Instead of treating each subject individually, they are assigned roles. The permissions are then assigned to the roles. This is the basic principle behind RBAC. This system benefits from the fact that it is easier to manage a large number of users, and easier to align with company policies. RBAC models can be viewed as an evolution of the group-based permissions in file systems. It allows easier implementation of the principle of least privilege. An RBAC system is defined with respect to an organization. The organization has a set of resources such as documents, print services, and network services, and a set of users, such as employees, suppliers, and customers.

RBAC is comprised of the following components:

- **User**: A user is an entity that wishes to access resources of the organization to perform a task. Usually, users are actual human users, but a user can also be an application.
- **Role**: A role is defined as a collection of users with similar functions and responsibilities in the organization. Examples of roles in a university may include *student*, *alumni*, and *faculty*. A user may have multiple roles, but

often only one active role at a time. Roles and their functions are often specified in the written policies of the organization. The assignment of users to roles reflects the status of the user within an organization. Role re-assignment can occur, such as when an employee is getting hired/fired, or when a student is getting admitted/graduating.

- **Permission**: A permission describes an allowed method of access to a resource, For instance, reading a file, writing to a file, printing to a printer, etc. Each role can have multiple permissions associated with it.
- **Sessions** A session consists of a user and an active role. A user may have multiple sessions activated, and in each session a different active role.

Hierarchical RBAC refers to the structuring of roles in a hierarchy similar to an organization chart. A role R_1 can inherit role R_2, meaning that R_1 includes all permissions of R_2. This is denoted as

$$R_1 > R_2.$$

This implies a partial order among the roles. When $R_1 > R_2$, we say that R_1 is **senior** to role R_2, and that role R_2 is **junior** to role R_1. In a company for instance, the role *manager* inherits the role *employee*, and the role *president* inherits the role *manager*. Thus,

$$president > manager > employee.$$

Note that this hierarchical structure does not have to be linear. There can be multiple branches in either direction.

Constraints are used to provide a means to adapting RBAC to the specifics of administrative and security policies of an organization. Constraints are defined as a relationship among roles, or conditions related to roles. There are three general types:

1. **Mutually Exclusive Roles** means that a user can only be assigned to one role in a set (separatino of duties). A permission can be granted to only one role in the set.
2. **Cardinality** means that either there is a maximum number of permissions per role, or there is a maximum number of users per role.
3. **Prerequisite Roles** means that a user can be assigned to a particular role if the user is already assigned to a prerequisite role.

15.2 Attribute Based Access Control (ABAC)

Attribute based access control can be considered an evolution of RBAC. Instead of roles with permissions, ABAC has policies involving attributes. Subjects, actions and objects all have attributes. For instance, the user may have attributes *age* and *clearance*, file may have attributes *sensitivity*, and the environment can have the attribute *currentTime*. **Policies** are boolean statements expressing what is allowed and what is not allowed. For instance, a policy could check whether $user.clearance > file.sensitivity$. In the event that it is, the it will allow the usr to read access the file.

In comparing ABAC, RBAC, DAC, and MAC, note that RBAC is flexible enough to implement DAC or MAC. DAC with ACLs and groups can emulate with a simple RBAC. ABAC can implement RBAC (and thus implement DAC and MAC). ABAC is the most expressive of the four, and also the most complicated.

16 February 13, 2017

16.1 Midterm Review

Discussed midterm format and second assignment questions.

17 February 27, 2017

17.1 Introduction to Cryptography

So far, we have seen cryptography in passwords tables (hashing, encryption), in authentication challenge-response protocols (pseudorandom number generator, hashing), ePassports (encryption, digital signature), barcodes on boarding passes (encryption, digital signatures), sending card information over networks (encryption, digital signatures), and SIM cards (encryption, digital signatures). In everyday life, cryptography is used when making calls, using smartphones, installing software, using credit/debit cards, opening car or garage doors, online banking, online shopping, email, checking grades, and surfing the web.

However, cryptography does not solve all problems. Bad implementations, social engineering, malicious code, and denial of service attacks are some problems that we still have to deal with. Cryptography mostly deals with confidentiality and integrity, and lesser so with availability. While the advantage of cryptography is that it seems to be able to do things that look impossible, it can also provide a false sense of security. It is important to understand the limits of cryptography, and even the best designs can be poorly implemented. **Snake oil cryptography** is any cryptographic method that is considered to be fraudulent.

Cryptography, loosely translated as "secret writing", is the practice and study of techniques for secure communication in the presence of third parties called adver-

saries. **The Fundamental Tenet of Cryptography** states that if lots of smart people have failed to solve a problem, then it probably won't be solved soon. We make use of the following terminology:

- **Plaintext** is the original message (readable).
- **Ciphertext** is the encrypted message (gibberish).
- **Cipher** is an algorithm used to encrypt/decrypt a message (also sometimes spelled cypher).
- **Key** is a secret used by the cipher to encrypt/decrypt a message.
- **Cryptography** can be thought of as the inventing of new ciphers.
- **Cryptanalysis** is the breaking of ciphers.
- **Cryptology** consists of cryptography and cryptanalysis.

Encryption is used to convert plaintext into ciphertext. Decryption is used to convert ciphertext to plaintext. It is okay for a cipher to be public. In fact, it is recommended. More eyes will find weaknesses sooner. Algorithms tend to be reverse-engineered anyways. According to **Kerckhoff's principle**, which is the second rule of six on practical cipher design, the design should not always require secrecy, and it should not be a problem if it falls into enemy hands. While it is okay for the cipher to be public, it is not okay for the keys to be public. The keys need to be secret, since the key is needed to decrypt. If one does not have the key, it is difficult to break the system.

17.2 Ciphers

Ciphers should be fast in order to be practical. The security of a cryptographic method is measured by how hard or easy it is to break. Generally, slow does not mean that it is secure.

Suppose a combination lock needs 3 numbers, each in the range from 1 to 40 inclusive. The number of possible combinations is $40^3 = 64000$. At 5 seconds per attempt, the worst case is 3.7 days to break, the best case is 5 seconds to break, and the average case is 1.85 days to break. Now, suppose the same combination lock needs 4 numbers in the same range. The number of possible combinations is now $40^4 = 2560000$. At 6 seconds per attempt, the worst case is 178 days to break, the best case is 6 seconds to break, and the average case is 89 days to break. This is all assuming that brute force is the best attack.

The **Caesar cipher** is a type of shift cipher with $shift = 3$. The **shift cipher** is a subtype of the substitution cipher. For the shift cipher, the encryption function is

$$E_k(x) = (x + k) \mod 26,$$

while the decryption function is

$$D_k(x) = (x - k) \mod 26.$$

The secret key of the cipher is k. When we are asked to recover the plaintext without knowing the secret key, we can try a **brute force attack**. For **secure ciphers**, brute force is the best option available for recovering the plaintext. Brute force attacks cannot be prevent, but they can be made more difficult by increasing the number of keys.

A **monoalphabetic cipher** is an arbitrary mapping of a character to another character via a **substitution alphabet**. A substitution alphabet can be any permutation of an alphabet. For instance, a rearrangement of the alphabet would be one such substitution alphabet. Shift ciphers are a subset of monoalphabetic ciphers. There is always some mapping. The total number of mappings would be $26! \approx 4 * 10^{26}$. At 1000000000 tries per second, the correct mapping could be found in 12700000000 years. This sounds good, but by using **frequency analysis**, this breaks almost any substitution cipher. Frequency analysis uses statistical properties of a language to decrypt the ciphertext. Some letters and words are more common than others, and some words are very likely or even expected to appear in ciphertext (**cribs**). This can be used to break most classical ciphers.

Polyalphabetic ciphers substitute each letter with multiple different characters, and make use of multiple substitution alphabets, while **polygraphic ciphers** substitute groups of letters with other groups of letters, and make use of multiple substitution groups. These attempt to make statistical analysis more difficult. However, these are still breakable with more complicated statistical analysis. Monoalphabetic, polyalphabetic, and polygraphic ciphers are examples of substitution ciphers. Mechanical substitution ciphers include ENIGMA and SIGABA.

18 March 1, 2017

18.1 Ciphers Cont'd

Transposition ciphers, also known as **permutation ciphers** permute the plaintext characters. For instance, for a given plaintext, it can be arranged horizontally in a square. The ciphertext is then taken by vertically reading the letters of the square. By themselves, transposition ciphers are susceptible to statistical analysis. However, combining substitution and permutation ciphers can lead to very strong ciphers. Several rounds of repeated substitution and permutation can be very strong. This is the basis of many modern symmetric ciphers.

The **Vigenere cipher** is similar to the Caesar cipher, but uses multiple shift values based on position. The key is a string that is repeated for the length of the plaintext. The plaintext is then shifted by the corresponding value in the key to obtain the ciphertext. The Vigenere cipher is based on disguising the frequency

of letters, since the same letter maps to different letters. This is an example of a polyalphabetic cipher. The primary weakness is the repetition of the key during encryption and decryption, since the key needs to be stretched to be the same size as the plaintext. Given enough data, we can perform statistical analysis. That is, when the plaintext is very long (much longer than the key), or when we have many pairs of plaintext and ciphertext encrypted with the same key. Cryptanalysis then consists of two steps: figuring out the key length and then solving the multiple Caesar cipher problem.

If the key is a length of 1 (a single letter), then the Vigenere cipher is simply the Caesar cipher. If the key length is equal tot the length of the plaintext, then we have a fairly strong cipher unless we keep reusing the key for multiple encryptions. The attacker could obtain many pairs of (plaintext, ciphertext) to cryptanalyze in the case that the key is used multiple times. If the length of key and plaintext are the same and the key is only used once, this is very secure, but the Venona project where the US and UK decrypted Soviet communications to identify spies has shown that the key can still be guessed. In the case the key and plaintext length are the same, the key is random and only used once, then we have the most secure cipher known.

The **One-time pad cipher** (OTP) was invented in 1882 by Frank Miller, and then re-invented and patented by Gilbert Vernam in 1919. It is essentially a Vigenere cipher where the key length matches the text length, the key is only ever used once (it is never re-used since a new key is created for each plaintext), and the key is random. This can be implemented using modular addition as in Caesar and Vigenere ciphers. However, it is usually implemented using XOR since this is much faster. The encryption and decryption functions are

$$E_k(x) = \text{plaintext} \oplus \text{key},$$

$$D_k(x) = \text{ciphertext} \oplus \text{key},$$

where $\oplus$ is bitwise XOR. XOR is both the encryption and decryption algorithm. It works because

$$D_k(x) = \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext}.$$

The OTP cipher cannot be cracked. Given a ciphertext, any plaintext is equally likely since there exists a unique key for any guessed plaintext. The ciphertext provides no information to the cryptanalyst except the length of the message. Thus, the one-time pad provides **perfect secrecy**. It is **information-theoretically secure**, which is also sometimes referred to as **unconditionally secure**. It is immune to brute-force attacks and even with infinite computational power, it cannot be broken. The drawback of the OTP cipher is that it is impractical.

Although theoretically secure, there are many practical problems:

1. Since the key length has to be the same length as the plaintext, the key must be shared in secrecy. If used for communication, we might as well deliver the message in plaintext using the same channel as used for sharing the key.

2. The key must be truly random, so it cannot be predictable. However, due to its length, this can pose a significant problem.

3. The key also must be securely destroyed after use. This is not as easy as it sounds, since keys can be extracted from hard drives, RAM, SSD, and optical disks.

Despite these significant practical issues, OTP has real applications. For instance, in teaching cryptography. It is also quite easy to perform by hand, requiring no power or trusted computers. Most modern ciphers are not feasibly performed by hand. There are some uses when the key can be shared beforehand. Furthermore, this provides motivation for more advanced ciphers, such as stream ciphers.

19 March 3, 2017

19.1 Measures of Security

Unconditionally secure occurs when ciphertext does not contain enough information to deduce the plaintext. This is the ultimate goal of cryptography, since the ciphers are based on provably unbreakable algorithms. The **one-time pad** for instance is unconditionally secure. In fact, it is currently the only known cipher in this category. However, the one-time pad requires a **one-time secret key** with the key length being the same length as the message. Encryption and decryption can be performed (for instance, using binary XOR with the plaintext and key for encryption, and using binary XOR with the ciphertext and key for decryption). Cryptanalysis is impossible, but this is not practical. There are other candidates, but they have yet to be proven unconditionally secure.

Computationally secure occurs when the time required to break the cipher is too long. That is, it exceeds the useful lifetime of the information protected. The cost is therefore too high compared to the value gained by obtaining the information. This assumes hat the attacker has limited computational power and is based on algorithms that have been traditionally hard to break. There are some issues with this definition, such as how to estimate useful lifetime, how to estimate the value of information, and how to estimate how long it would take to break the cipher. There could be overestimates of lifetime and value, along with underestimate of how long it would take to break.

19.2 Types of Cryptography

The following are the main forms of cryptography:

1. **Cryptographic Hashing** requires no keys at all, since the cryptographic hash functions are simply computationally expensive.

2. **Secret Key** requires one key remain secret. The key is often shared for communication purposes, as in the case of symmetric keys. This is also known as **symmetric cryptography**.

3. **Public Key** requires two keys, where one is public and known to the world, while the other is private. The private key is secret, but we do not call it "secret" to avoid confusion. This is also known as **asymmetric cryptography**

19.3 Cryptanalytic Attacks

The attacker's goal is ultimately to recover the key. However, recovering the plaintext is often sufficient, and learning something about the key or plaintext allows for partial information extraction. The more information the attacker has, the more likely they are to succeed. Assuming that the attacker knows the ciphertext (or a set of ciphertexts), attacks can be based on the following forms:

- **Ciphertext only** is when the attacker knows nothing in addition to the ciphertext.
- **Known plaintext** is when the attacker also knows one or more plaintext-ciphertext pairs.
- **Chosen plaintext** is when the attacker also knows the plaintext-ciphertext pair, but for a plaintext chosen by the attacker.
- **Chosen ciphertext** is when the attacker also knows the plaintext recovered from a chosen ciphertext.
- **Chosen text** is when the attacker also knows the ciphertext for a chosen plaintext, and the plaintext for a chosen ciphertext.

20 March 6, 2017

20.1 Cryptographic Hash Functions

As we recall, a **hash function** is a function that maps data of arbitrary size to data of a fixed size. The output of a hash function is called a **hash value** or **hash** and is noted as

$$h(x) = y,$$

where x is the data and y is the hash. The hash function must be deterministic. That is,

$$x_1 = x_2 \implies h(x_1) = h(x_2),$$

$$h(x_1) \neq h(x_2) \implies x_1 \neq x_2.$$

However, this does not mean that

$$h(x_1) = h(x_2) \implies x_1 = x_2.$$

This is because **collisions** are possible and unavoidable. Other desirable properties include speed and a uniformly distributed hash. Depending on its uses, it can require other additional properties.

Cryptographic hash functions are hash functions with additional properties. The input is referred to as the **message** while the output is called the **message digest** (or **digest, fingerprint**, or even checksum). It should be infeasible to invert (brute force only), and a small change in the message should result in a large change in the hash. It should also be infeasible to find collisions, such as finding to (x, y) such that $h(x) = h(y)$. Furthermore, a cryptographic hash function should appear random by passing pseudo-random tests.

The main application is message integrity and authentication. The problem concerns the means by which the recipient may verify the integrity of the message. Before sending the message, we compute the digest of the message. We then send the message and include the digest (m, d). When the recipient receives the message and digest, they compute their own digest $d' = h(m)$. The recipient may then compare their digest with the received digest to verify that $d = d'$. However, this does not guarantee that the message has not been verified. For instance, a MITM attack could substitute both the message and the digest with (m', d').

Cryptographic hash functions must be immune to all known cryptanalytic attacks, such as pre-image attacks, second pre-image attacks, collision attacks, and additional attacks such as extension attacks and similar message or similar digest attacks. "Immune" is related to computational security. The higher the value of information, the more the attacker will spend to obtain that information. Hash values of 256 bits should be able to defend against most practical brute-force attacks.

Pre-image resistance is related to the fact that a cryptographic hash function should be **non-invertible**. For a given digest of y, it should be difficult to find any message x such that

$$h(x) = y,$$

where x is the pre-image of y. Difficult in this sense simply means that a brute force attacks is the only viable way to determine x from y. For most hash functions, brute force attacks are of the complexity of 2^n, where n is the bit length of y. For example, consider a 128 bit hash function. The digest is necessarily 128 bits long. The attacker would need to perform around $2^{128} \approx 3.4 * 10^{38}$ evaluations. Assuming one trillion evaluations per second, this would take around 10^{19} years. If the digest is 129 bits long, then the attacker would need to perform many more evaluations. **Second pre-image resistance** states that for a given message x, it should be difficult to determine y such that

$$h(x) = h(y).$$

If an attacker knows a message, it should be difficult for them to find another message that has the same digest.

Collision resistance states that it should be difficult to find any pair (x, y) such that

$$h(x) = h(y).$$

Furthermore, it should be difficult to find any two messages that have the same digest. Collision attacks alone are usually without practical applications, but they can signal a weakness in the hash function. Collision resistance implies second pre-image resistance, while second pre-image resistance does not guarantee pre-image resistance. Finding collisions via brute fore does not have a complexity of 2^n. For instance, consider the birthday problem. The number of hash values needed to find two collisions with probability p is given as

$$n(p, H) \approx \sqrt{2H \ln\left(\frac{1}{1-p}\right)},$$

where H is the number of possible hash values. For instance, if we want $p = .5$, then $n(0.5, H) \approx \sqrt{H}$. A hash function with an n bit digest will only provide $n/2$ bits of security for collision resistance assuming this is an ideal hash function. If we want 128 bits worth of security against collision attacks, we need a hash function with at least 256 bits of digest.

Message **checksum** requires that the integrity of the message is checked for accidental corruption. Usually, it is computed using a non-cryptographic hash function. For instance, this could be of the form

$$\text{checksum} = crc32(\text{message}).$$

Message **digest** is similar to checksum, but it is calculated using a cryptographic hash function, and could be of the form

$$\text{digest} = SHA3(\text{message}).$$

The digest detects malicious modifications. However, it does not protect against MITM if the message and digest are both susceptible to tampering. It is still useful for file checksums. The **message authentication code** (**MAC**) is similar to digest but can only be computed and verified using a secret. A possible implementation is

$$MAC = h(\text{secret}|\text{message}).$$

MAC confirms the integrity of the message and the authenticity of the message. In this way, it can protect against MITM attacks. MAC is used to compute and verify the **tag** of a message. A **keyed-hash message authentication code** (**HMAC**) is a particular implementation of MAC. It is defined in RFC 2104, and is currently the most popular MAC. It defends against length-extension attacks. It is of the form

$$HMAC = h(k_1|h(k_2|m)),$$

where m is the message, k_1 and k_2 are two different versions of the secret, and requires the application of $h()$ twice. For instance, this could be

$$HMAC(m,k) = h(XOR(k', opad)|h(XOR(k', ipad)|m)),$$

where h is the cryptographic hash function, m is the message, k is the key, k' is derived from k by making it the same length as input block size of $h()$ either by padding with 0's if short or by $k' = h(k)$ if it is too long, *opad* is $0x5c5c5c5c...5c$, and *ipad* is $0x36363636...36$. Just about any cryptographic hash function can be used as h. Note that SHA3 does not require HMAC, since a simple scheme digest of $sha3(k|m)$ is sufficient.

21 March 8, 2017

21.1 Cryptographic Hash Function Applications

The **Message-Digest Algorithm 5** (**MD5**) was developed by Ron Rivest in 1991. It uses 128-bit hash values. It is still widely used in legacy applications, even though it is generally considered insecure. Various sever vulnerabilities were discovered, with the first indications in 1993, and full collisions in 2004. Chosen prefix collision attacks were found by Marc Stevens, Arjen Lenstra, and Benne de Weger. To accomplish this, we start with two arbitrary plaintexts P and Q. One can compute suffixes S_1 and S_2 such that $P|S_1$ and $Q|S_2$ collide under MD5 by making 250 hash evaluations. Using this approach, a pair of different executable files or PDF documents with the same MD5 has can be computed.

The **Secure Hash Algorithm** (**SHA**) is a family of cryptographic hash functions published by NIST as a federal standard. **SHA-1** was developed by the NSA with 160 bit output. Although considered insecure, it is more secure than MD5 and is still used in legacy systems. Only theoretical attacks have been reported, since it would cost around $100000 to break a single hash. **SHA-2** was also developed by the NSA, and has 256 and 512 bit output. This is still considered secure, as no significant attacks have been reported. **SHA-3** was the winner of a public competition. It is very different from SHA-1 and SHA-2. It has output sizes of 224, 256, 384, and 512 bits.

Hash functions are often constructed from simpler **one-way compression functions**. Compression functions have the same general properties of a general hash function. Namely, they are fast, deterministic, uniformity, pre-image, second pre-image, and collision resistant. However, the input has a fixed length (input block size). For instance, it takes $2n$ bits of input to produce n bits of output. Compression functions are then combined to implement cryptographic hash functions that accept variable length input. **Merkle Damgard Hash Function Construction** is a method of building collision resistant cryptographic hash functions from simpler

one-way compression functions. If the underlying compression functions are collision resistant, then the constructed function is also collision resistant. Many cryptographic hash functions such as MD5, SHA1 and SHA2 are based on this method. It is important to note that padding is vital to many algorithms. Supposing that the message is not long enough, common padding is of the form

$$10000...0|\text{message length}.$$

It may be necessary to add a new block if the padding calls for it. It is also clear that cryptographic hash functions have a limited lifetime.

Remark. Note that we perform padding for every input given into the function.

22 March 10, 2017

22.1 Symmetric Cryptography

In **secret key cryptography** (**symmetric**), the key must remain secret from a potential adversary. This means that the key may have to be shared, such as through a two-way communication. This is a potential source of weakness. The **secret key** is used both for encryption and decryption. The encryption of a message m for a secret key k is denoted as

$$\text{ciphertext} = E_k(m),$$

while decryption of the ciphertext c is given as

$$\text{plaintext} = D_k(c).$$

This is in contrast to **public key cryptography** (**asymmetric**), which uses a **public key** that ideally everyone knows, and a **private key** that remains secret.

Secret key cryptography provides a means of secure communication since nobody can eavesdrop on the conversation. Replay attacks are possible however, and integrity is not preserved. This form of cryptography can also be used for secure storage on insecure media, such as the encryption of data before writing and the decryption before reading. **Mutual authentication** via challenge-response is possible, since one can prove the knowledge of a key without revealing it. Consider the following scenario where Alice and Bob want to prove to each other that they know the secret key k:

1. Alice creates and sends a unique challenge c_A.
2. Bob crates and sends a unique challenge c_B.
3. Alice computes and sends back $r_A = E_k(c_B)$.
4. Bob computes and sends back $r_B = E_k(c_A)$.
5. Alice verifies $D_k(r_B) = c_A$ and Bob verifies $D_k(r_A) = c_B$.

22.2 Block Ciphers

Most modern ciphers operate on fixed blocks of plaintext and ciphertext. Many ciphers require that plaintext and ciphertext blocks contain 128 bits. To encrypt messages of arbitrary size, we split the message into blocks. A plaintext of length n is partitioned into a sequence of m blocks where each block has a length of b, where the last block may require padding. The message is then encrypted or decrypted in terms of the blocks. The padding may be done by introducing extra bits.

Since block ciphers require the length of the plaintext be a multiple of the block size, padding needs to be performed. Padding the last block needs to be unambiguous. That is, one cannot simply add zeroes. When the block size and plaintext length are a multiple of 8, a common padding method is PKCS7. The padding in this method is a sequence of identical bytes, each indicating the length (in bytes) of the padding.

Example. *Suppose we have a block size of* $b = 128$*, which is 16 bytes. The plaintext message is "Roberto", which is 7 bytes long. Thus, we require* $16 - 7 = 9$ *bytes of padding. Determine the padded plaintext.*

Since we require 9 bytes of padding, this means that according to the PKCS7 padding method, we add a sequence of identical bytes each indicating the length in bytes of the padding. Thus, the padded plaintext becomes "Roberto999999999", which is now 16 bytes long.

Remark. We note that the last block always needs to be padded. Thus, if the message is a multiple of the block size, then an extra block will be added containing only the padding.

In practice, block ciphers have many applications. The **Data Encryption Standard** (**DES**) was developed by IBM and adopted by NIST in 1977 via competition. The NSA suggested some changes which IBM accepted. This was declared a federal standard for unclassified sensitive data. DES uses 64-bit blocks and 56-bit keys. The small key space makes exhaustive search attacks feasible since the late 90s. This standard is not secure at all today, but this is only because brute-force attacks have been made possible. The DES standard generated much interest in the cryptographic community. It has since been superseded by AES.

The **Triple DES** (**3DES**) was built to fix the DES weakness, namely its short key length. 3DES is a nested application of DES with three different keys k_A, k_B, and k_C. The effective key length is therefore 168 bits, making exhaustive search attacks infeasible. The encryption and decryption are

$$\text{ciphertext} = E_{k_C}(D_{k_B}(E_{k_A}(\text{plaintext}))),$$

$$\text{plaintext} = D_{k_A}(E_{k_B}(D_{k_C}(\text{ciphertext}))).$$

This is equivalent to DES when $k_A = k_B = k_C$, making it backwards compatible. 3DES is still used today and is generally considered secure.

DES uses **Feistel Network**, where the approach is to generate complexity by repeating s simple operation (**rounds**). Each round uses a **sub-key** k_1 derives from k. The encryption and decryption are the same algorithm, but using the sub-keys in the reverse order. Security depends on the function F called the Feistel function. In this function, a XOR operation is performed on the half block of 32 bits and the sub-key of 48 bits. Many substitutions are performed, with a final permutation performed on the collected data from the multiple substitutions. DES uses a half block of 32 bits in Expansion. The XOR operation is performed on this with a subkey of 48 bits. Substitution is then performed to separate it into 8 components, which are then combined for Permutation.

The **Advanced Encryption Standard** (**AES**) was selected by NIST in 2001 through open international competition and public discussion. AES is a subset of the Rijndael cipher, and is based on substitution-permutation network. It uses block sizes of 128 bits and key lengths of 128, 192, and 256 bits. Exhaustive search attacks are not currently possible. The **AES-256** is currently the symmetric encryption algorithm of choice. AES takes an input of 128 bits and a key of an appropriate length described above, to produce an output of 128 bits.

Instead of using a Feistel network, AES uses a **permutation-substitution network**. The AES encryption algorithm proceeds in n rounds,

- AES-128: 10 rounds.
- AES-192: 12 rounds.
- AES-256: 14 rounds.

Each round performs an invertible transformation on a 128-bit array called the state. The initial state X_0 is the XOR of the plaintext P with the key k, so that $X_0 = P \oplus k$. Round $1 \leq i \leq n$ receives state X_{i-1} as input and produces state X_i. The ciphertext C is the output of the final round, so $C = X_n$. Decryption is performed by reversing the rounds.

There are 4 steps per round in the AES standard. This consists of the following steps:

1. SubBytes step: An S-box substitution step.
2. ShiftRow step: A permutation step.
3. MixColumns step: A matrix multiplication step. This is skipped in the last round.
4. AddRoundKey step: A XOR step with a round key derived from the 128-bit encryption key.

23 March 13, 2017

23.1 Block Cipher Modes of Operation

A block cipher mode of operation describes the way a block cipher encrypts and decrypts a sequence of message blocks. Some standard modes include ECB, CBC, OFB, CFB, and CTR. The **Electronic Code Book Mode** (**ECB**) is the simplest. The plaintext block $P[i]$ is encrypted into the ciphertext block $C[i] = E_k(P[i])$, while the ciphertext block $C[i]$ is decrypted into the plaintext block $M[i] = D_k(C[i])$. ECB is simple, allows for parallel encryptions of the blocks of a plaintext, and can tolerate the loss or damage of a block. However, documents and images are not suitable for ECB encryption since patterns in the plaintext are repeated in the ciphertext.

The **Cipher Block Chaining Mode** (**CBC**) consists of the previous ciphertext block combine with the current plaintext block,

$$C[i] = E_k(C[i-1] \oplus P[i]).$$

$C[-1] = IV$ is a random block called an **initialization vector**. Decryption is performed by

$$P[i] = C[i-1] \oplus D_k(C[i]).$$

CBC does not show patterns in the plaintext and is relatively fast and simple. This is the most common mode, as decryption can be performed in parallel and even random access is possible. However, there is no integrity protection, and encryption cannot be performed in parallel. It is not suitable for applications that allow packet losses (such as music and video), and requires padding.

A block cipher in **Counter Mode** (**CTR**) can turn a block cipher into a stream cipher, such as through using AES-256. It uses a block cipher H with a block size of b (AES-256 implies $b = 128$ bits). The secret key pair (k, T), where k is the key and T is a counter is also a b-bit value. The (k, T) pair must be unique, so T is often a nonce. No padding is required. The ciphertext should be sent with HMAC, and the HMAC **tag** should be calculated after encryption (encrypt then authenticate).

$$\text{ciphertext}_i = H_k(\text{counter}_i) \oplus \text{plaintext}_i, \text{counter}_{i+1} = \text{counter}_i + 1,$$

$$\text{plaintext}_i = H_k(\text{counter}_i) \oplus \text{ciphertext}_i, \text{counter}_0 = T.$$

Essentially, a nonce is used with the counter, along with the key as input intot he block cipher encryption function. We then perform XOR on the resulting output with either the plaintext or ciphertext to obtain the corresponding ciphertext or plaintext respectively.

23.2 Stream Ciphers

A **stream cipher** is a symmetric key cipher where the plaintext digits are combined with a pseudo-random cipher digit stream called the **keystream**. Each plaintext

digit is encrypted one at a time with the corresponding digit of the keystream to give a digit of the ciphertext stream. The keystream is a pseudo-random sequence of bits $S = S[0], S[1], S[2], ...$ that can be generated online one bit (or byte) at a time. The stream cipher performs a XOR on the plaintext with the keystream so that $C[i] = S[i] \oplus P[i]$. The stream cipher is suitable for plaintext of arbitrary length generated live (for instance, in a media stream), and is loosely motivated by the one-time-pad cipher.

In a **synchronous stream cipher**, the key stream is obtained only from the secret key k. Both sides must be synced, and a sync loss could result in a problem. Recovery is still possible by attaching markers to the ciphertext. However, the random bit change is not a problem. In **self-synchronizing stream ciphers**, key streams are obtained from the secret key and n previous ciphertexts. The loss of packets cause a delay of n steps before decryption resumes. An example would be $RC4$, which was designed by Ron Rivest for RSA Security in 1987. It is also known as **Ron's Code**. This was a trade secret until 1994. It uses keys with up to 2048 bits and is a relatively simple algorithm.

23.3 Confusion and Diffusion

It is not easy to design a good cipher, as it requires a lot of experience. We do know some desirable properties of ciphers:

- It should be hard to figure out the plaintext from the ciphertext.
- It should be hard to figure out the key from the ciphertext.
- We assume that the attacker already knows the ciphertext, so the ciphertext should not be trivially similar to the plaintext.

For example, the attacker could know the frequency distribution of the plaintext, or know certain words or phrases that are used in the plaintext. This could be used to break a cryptographic algorithm. In 1945, Claude Shannon identified two important properties of ciphers, which are confusion and diffusion. If both are present, they make statistical analysis very difficult:

1. **Confusion**: The relationship between the key and the ciphertext is very complex. Every bit of ciphertext should depend on the entire key. Any change in the key should result in an unpredictable change in the ciphertext. The key cannot be deduced even from many different plaintext-ciphertext pairs.
2. **Diffusion**: The relationship between the plaintext and the ciphertext is very complex. Any change in the plaintext should change every bit of ciphertext with equal (50%) probability, and vice versa (**avalanche effect**).

Confusion is generally implemented by some form of substitution, while diffusion is generally implemented by some form of permutation. Confusion and diffusion are

generally implemented as a repeatable series of substitutions and permutations. What remains is mixing in the key between each round. Many modern ciphers are implemented using substitution-permutation networks. DES and AES are two such examples.

A **Meet-in-the-middle attack** is a space?time tradeoff attack on schemes using repeated encryption. For example, consider 2DES with two keys, $K1$ and $K2$. Encryption and decryption are given by

$$C = E_{K2}(E_{K1}(P)),$$

$$P = D_{K1}(D_{K2}(C)).$$

A brute force attack with $O(1)$ memory would require $2^{56+56} = 2^{112}$ attempts. With $O(2^{56})$ memory, we can execute a meet-in-the-middle attack. This requires precomputing all $E_{K1}(P)$ and all $D_{K2}(C)$ possibilities to identify key candidates. The time complexity is then $O(2^{57})$. 2DES, despite having 112 bit key size, only offers 57 bit security. 3DES is also vulnerable to meet-in-the-middle attacks. Although the key size is 168 bits, it only offers 112 bit security, provided $O(2^{56})$ is feasible. NIST predicts 3DES should remain safe until 2030.

24 March 15, 2017

24.1 Public Key Cryptography

If n people want to communicate using symmetric key, then every pair needs a separate symmetric key. Thus, they would need $O(n^2)$ keys. This is too many, and is difficult to manage. Thus, we either use public cryptography where we only need $O(n)$ keys, or we use centralized key management. The main uses of public key cryptography are for secure communication and digital signatures.

Consider the following procedure for communication between Alice and Bob using asymmetric cryptography:

1. Alice wants to send a message to Bob.
2. Alice has public and private keys k_A and k_A^{-1} (or kA and PkA).
3. Bob has public and private keys k_B and k_B^{-1} (or kB and PkB).
4. Alice encrypts the plaintext using Bob's public key k_B^{-1}, then sends the ciphertext to Bob. That is,
$$C = \{P\}_{kB}.$$
5. Bob decrypts the ciphertext using his private key, since
$$P = \{C\}_{PkB}.$$

6. If Bob wants to send a message to Alice, Bob encrypts using Alice's public key while Alice decrypts using her private key.

This serves to ensure confidentiality, and has nothing to do with integrity or non-repudiation. In this example, we assume that Alice and Bob know each other's public keys.

In public key communication where n people want to communicate, we need $O(n)$ public and private key pairs. It is still difficult to manage all of these public keys. The solution is to use a public key infrastructure or web of trust. Since public key ciphers are much slower than symmetric ciphers, these are often used together in **hybrid cryptosystems**. For instance, public key cryptography can be used in communication. First, the ender and receiver agree on a big random secret key. First, we exchange secret keys using public cryptography, and then use these secret keys for the rest of the communication. SSL and TLS (HTTPS) use a similar mechanism. Asymmetric cryptography can also be used to encrypt information stored on insecure media. First, a random secret key is generated, and then encrypted using asymmetric cryptography. Data can then be encrypted with the secret key using symmetric cryptography. We note then that asymmetric cryptography is used to securely transfer the secret key.

Digital signatures provide a mechanism for proving authenticity of a message (similar to standard signatures). The message arrives as a tuple $(message, signature)$. To compute the signature, a private key is used. Normally, this is applied only to a cryptographic hash of the message. The signatures is

$$signature = E_{private}(h(message)).$$

Only someone in possession of the private key can sign. To verify the signature, the public key is used. The cryptographic hash is computed on the received message. The received signature is decrypted using the public key and compared, where

$$D_{public}(signature) = h(message).$$

In this way, anyone with the public key can verify the signature. The signature can only be generated by someone in possession of the private key (and thus cannot be forged), but can be verified by anyone that has the pubic key. It is therefore ideal that the public key is widely known. Applications include authenticity, integrity (standard signatures do not have this), non-repudiation of origin, and certificates.

24.2 Diffie-Hellman Key Exchange

Suppose that Alice and Bob want to communicate securely. They have never met, and they do not have a secret key. The solution is to use the DHFE algorithm to establish a secret key. DHKE is used to generate a shared secret between two parties. This secret can then be used for symmetric encryption. The algorithm

relies on the assumption that the **discrete logarithm problem** is difficult. It is believed that DHKE is as hard as the DLP.

Let p be a prime number, and let a and b be non-zero integers. The discrete logarithm problem states that given a, b, and p, find x such that

$$a^x = b \mod (p).$$

DLP is believed to be an exponentially difficult problem, since the best running time is exponential with respect to $\log(p)$. The record in 2015 was a 232 digit prime number p, which was broken in 6600 core years ($p \approx 770$ bits). 2048 bits should be sufficient for the foreseeable future (until around 2030), while 4096 bits is even safer.

The DHKE algorithm is based on two people choosing a private number, and then calculating a public number to present the other. They then communicate the public numbers to each other. Each individual can then use the other's public number with their own private number to compute the shared secret. The eavesdropper cannot reproduce this. The steps of the DHKE are as follows:

1. Alice chooses a random prime number p, a secret number a, and g that is a primitive root modulo p.
2. Alice sends Bob$(g, p, g^a \mod (p))$.
3. Bob then generates a secret number b, and sends Alice $g^b \mod (p)$.
4. Bob computes $k = (g^a \mod (p))^b \mod (p)$.
5. Alice computes $k = \left(g^b \mod (p)\right)^a \mod (p)$.
6. Since the above two expressions are equal, they now both have the same number k, which is the newly created shared secret. They can therefore start communicating via symmetric cipher using the key k.
7. Eve the eavesdropper cannot compute $k = g^{ab} \mod (p)$ since she only as g, p, $g^a \mod (p)$, and $g_b \mod (p)$.

We note that if a number g is a **primitive root modulo** p, then it has the special property that g raised to the powers of $1, 2, ..., p-1 \mod (p)$ generates $\{1, 2, ..., p-1\}$. That is,

$$\{g^1 \mod (p), g^2 \mod (p), ..., g^{p-1} \mod (p)\} = \{1, 2, ..., p-1\}.$$

3 for instance is a primitive root modulo 7, while 2 is not. Consider the following example:

1. Alice and Bob agree to use $p = 23$ and $g = 5$. Note that 23 is a prime and 5 is a primitive root modulo 23.

2. Alice chooses a secret integer $a = 6$, then sends Bob

$$A = g^a \mod (p) = 5^6 \mod (23) = 8.$$

3. Bob chooses a secret integer $b = 15$, then sends Alice

$$B = g^b \mod (p) = 5^{15} \mod (23) = 19.$$

4. Alice then computes

$$s = B^a \mod (p) = 19^6 \mod (23) = 2.$$

5. Bob then computes

$$s = A^b \mod (p) = 8^{15} \mod (23) = 2.$$

6. Alice and Bob now share a secret s (the number 2).

However, DHKE is vulnerable to MITM attacks. After intercepting the first transmission to obtain g, p, and $g^a \mod (p)$, the attacker could send a false $g^c \mod (p)$ to both parties, along with g and p to the original receiver of the first message. To resolve this, we use public key encryption in the Authenticated DHKE to sign all communication with private keys. For instance, Alice signs everything with her private key, while Bob signs everything with his private key. Alice and Bob can now verify that MITM did not tamper with the exchange. We have now authenticated DHKE, so it is immune to MITM. In summary, the Authenticated DHKE requires that $g^a \mod (p)$ and $g^b \mod (p)$ are signed with the respective owner's private keys before they are sent. The signatures are then verified to ensure that the exchange has not been tampered with.

However, if we already have public and private keys already, then why do we bother with DHKE? We do so to achieve **forward secrecy**, which is also known as **perfect forward secrecy**. Forward secrecy is the property of secure communication in which the compromise of a long-term secret key does not compromise past session keys. That is, someone who learns of Alice or Bob's private keys would not permit them to decrypt past conversations. Communication based on the Diffie-Hellman Key Exchange have this property if each communication session uses a different prime number.

25 March 17, 2017

25.1 Enveloped Public Key Encryption

EPKE is a method of using public key cryptography to ensure confidentiality, integrity, non-repudiation. The sender first signs a message with their own private

key, then encrypts this with the receiver's public key. When this is sent, the message cannot be observed or tampered with. When one receives the message, they decrypt the received message using their own private key. They can then verify the signature using the sender's public key.

25.2 RSA

RSA (**Rivest, Shamir, Adleman**) is one of the first practical public-key cryptosystems and is widely used for secure data transmission. It was created in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. Clifford Cocks had developed a similar system in 1973 for the UK intelligence agency GCHQ, but this was not declassified until 1997. The public key consists of the pair of numbers (e, n), while the private key consists of the pair of numbers (d, n). Note that only d needs to be kept secret. Encryption and decryption are respectively,

$$c = m^e \mod (n),$$

$$m = c^d \mod (n),$$

where c is the ciphertext and m is the plaintext. In this case, we have a maximum length for c and m, since it would be dictated by n. **Optimal asymmetric encryption padding** (**OAEP**) provides a means to add an element of randomness to convert a deterministic encryption scheme (such as traditional RSA) into a probabilistic scheme. This is especially useful when the message m is small. It also prevents partial decryption of ciphertexts by ensuring that an adversary cannot recover any portion of the plaintext without being able to find the inverse of the encryption function.

For RSA to be useful, we need to be able to find (e, d, n) such that

$$(m^e \mod (n))^d \mod (n) = m^{ed} \mod (n) = m,$$

for any $m < n$. We also need to be able to quickly come up with many different version of (e, d, n). Encryption and decryption should be fast, and it should also be difficult to decrypt a ciphertext if only (e, n) are known. The RSA process is described below:

1. Two distinct prime numbers p and q are selected. These should be chosen randomly, and should be similar in magnitude but differ in length by a few digits to make factoring harder. This can be accomplished using cryptographically strong random number generation and a probabilistic primality test (such as Miller-Rabin).

2. Calculate $n = p \cdot q$. $\log_2(n) \approx 4096$ bits is the key length.

3. Calculate $\phi(n) = (p-1) \cdot (q-1)$ or $\lambda(n) = lcm(p-1, q-1) < \phi(n)$, where $\phi(n)$ is Euler's totient function, and $\lambda(n)$ is Carmichael's totient function. This value is kept secret.

4. Select e such that e is relatively prime to $\phi(n)$ or $\lambda(n)$. That is, the greatest common denominator of e with the other value is 1. Usually, e can be small ($1 < e < \phi(n)/\lambda(n)$), so that it starts at 65537, which is 10000000000000001 in binary.

5. Next, we find d such that $e \cdot d \mod (\phi(n)/\lambda(n)) = 1$. That is, d is the multiplicative inverse of e modulo $\phi(n)$ or $\lambda(n)$. This can be calculated quickly using the extended euclidean algorithm. e is released as the public key exponent while d is kept as the private key exponent.

RSA is secure under the assumption that integer factorization is difficult. Integer factorization involves finding all the prime factors of n. There is currently no known algorithm that can accomplish this in polynomial time. The second assumption is that the strength of the RSA cipher is related to the factorization problem. To break RSA, we want to recover m from $c = m^e \mod (n)$. The best way we know to accomplish this is to find d from (e, n) where d is the multiplicative inverse of e modulo the function used. However, to find d, we need the corresponding $\phi(n)$ or $\lambda(n)$. These functions are used as **trapdoors**. However, to find these functions, we need (p, q), which is the same as solving integer factorization.

RSA works because

$$m^{e \cdot d} = m \mod (n).$$

We know that $n = pq$, where p and q are both prime. Thus, if we can show that $m^{e \cdot d} = m \mod (p)$ and $m^{e \cdot d} = m \mod (q)$, then $m^{e \cdot d} = m \mod (n)$ since $n = pq$. First, we consider the case that m is a multiple of p. Then $m = 0 \mod (p)$, so $m^{e \cdot d} = m \mod (p)$. In the second case that m is not a multiple of p, we note that we chose e and d such that they equal 1 modulo $\phi(n)$. So, $ed = 1 \mod ((p-1) \cdot (q-1))$. Thus, $ed = 1 \mod (p-1)$. This means that $ed = k \cdot (p-1) + 1$ for some integer k. Then, $m^{e \cdot d} = \left(m^{p-1}\right)^k m$. By Fermat's little theorem, $m^{p-1} = 1 \mod (p)$, so $m^{e \cdot d} = 1^k m \mod (p)$. The proof for q is identical.

26 March 20, 2017

26.1 Digital Certificates

A practical problem with public key cryptography is that one needs to know another entity's public key to encrypt. However, it is difficult to ensure that one obtains a legitimate key in the presence of MITM. Both entities need to send and retrieve public keys through a mechanism that can verify this key. This mechanism that is to be trusted by both parties may come from a centralized service that would verify both public keys. However, this first solution could be impractical or susceptible to MITM attacks. The second solution is to employ a trusted third party. This is the basis for digital certificates:

1. Alice and Bob get their public keys signed by Trent.
2. Alice contacts Bob.
3. Bob sends to Alice his public key together with Trent's signature (as one document).
4. Alice verifies the document signature.
5. If the signature matches, then Alice has good reason to believe she is talking to Bob.
6. Alice also sends to Bob her public key, together with Trent's signature.
7. The cycle repeats.

The signed document is called a **digital certificate** or a **public key certificate**. In this scenario, Trent plays the role of the **certificate authority** (**CA**). Before Alice can talk to Bob, she needs to obtain (somehow) the certificate authority's public key and install in on her computer. Alice is now ready to talk to Bob, so long as Bob uses the same CA. Alice can trust anyone who has their public key signed by the CA. Before Bob can respond, Bob asks the certificate authority to sign his public key. The CA verifies Bob's identity and charges a fee. The CA then signs Bob's key using the CA's private key. The CA then gives Bob the certificate, which includes Bob's public key and the CA's signature.

A **digital certificate** is a signed document containing an identity and a public key with some additional information. It is used to prove the ownership of a public key by binding an identity and a public key. The trust must start somewhere, so we generally trust the CA's signature and their public key. The CA's public key (trust anchor) must be obtained out-of-band. This usually ships with a browser. Inside the certificate is the public key, subject, issuer, and signature. Technical information, such as the signature algorithm used and a unique serial number are also on the certificate. The certificate contains a valid from/to date, outlines the purposes for the key, and provides identifying information such as the company name and location. The issuer does not have to be the root CA, as it could be an intermediate CA. Thus, certificates can contain a chain of certificates.

Most CAs are tree-structured. The **root certificate authority** endorses the **intermediate certificate authorities**, forming a **hierarchy**. This creates a chain of trust, as a certificate includes certificates of all intermediate CAs up to the root. There are multiple root CAs, each with their own hierarchies. Browser are generally pre-installed with **root certificates**. Some website certificates can be checked using the command line or using online services, such as "https://www.geocerts.com/ssl˙checker".

Keys associated with certificates can be compromised. One possible solution is to use **certificate revocation lists** (**CRLs**). These can get very large. They are a good thing when certificates expire, and browsers consult these periodically.

Another solution is to use **online certificate status protocol** (**OCSP**). These are web-services that check the validity of a certificate by8 serial number. There can be security implications (MITM) and a high load on the CA. An improvement is **OCSP stapling**, which is similar to kerberos tickets in that the certificate holder requests time-stamped validation. Certificate revocation is a serious problem.

The addition of SSL/TSL over HTTP is HTTPS, and is indicated by the green lock icon in the address bar. Communication is authenticated using certificates. Some third party verifies that the site belongs to the entity claiming to be the owner. Communication is encrypted, and all content on the site is also protected (images, scripts, etc). The green icon anywhere else means nothing. Generally, when we use a browser, we trust the browser developers. Many certificate authorities do not have many warranties, and disclaim all liability.

PGP's Web of Trust is a solid privacy encryption software that is used primarily for email encryption. PGP implements a different style of public key infrastructure that is based on a **web of trust** instead of the traditional certificate hierarchy. The certificates can form an arbitrarily-complex graph. Users can verify paths to as many trusted anchors as they wish. Users sign each other's certificates at signing parties. Other alternative trust models rely on commercial identity-based CAs. One obtains certificates from parties they know directly, and issue certificates to their own users. Let's Encrypt is a free, automated, and open certificate authority, "https://letsencrypt.org/".

X.509 and **PKIX** are standards for public key certificates. X.509 is an international standard from the International Telecommunications Union's Standardization sector (ITU-T), while PKIX is a practical subset of X.509 used on the internet. Parsing is not trivial, so OpenSSL does most of this for us.

27 March 24, 2017

27.1 SSL and TLS

Transport Layer Security and its predecessor Secure Sockets Layer are cryptographic protocols used to establish secure communications over an insecure network. This means that it is immune to both eavesdropping and tampering. It provides confidentiality through symmetric encryption, with keys generated uniquely for each connection (perfect forward secrecy). The algorithms and other technical details to accomplish this are negotiated by both ends. Since it provides a message authentication code (MAC) for every message, it protects the integrity of the data. Lastly, it verifies authentication of both parties through certificates and trusted CAs. In client/server scenarios, it is usually only the server that authenticates itself to the client.

All versions of SSL were created by Netscape. SSL 1.0 was motivated by the need for secure HTTP, but it was not ultimately released. SSL 2.0 was released in 1995

and used for HTTPS, but contained serious flaws. In 1996, SSL 3.0 was released, and has since been broken in 2014 through a poodle attack. TLS 1.0 (RFC 2246) was based on SSL 3.0, and was released in 1999. Small but important differences exist between the two, so they are not compatible. TLS 1.1 (RFC 4346) was released in 2006 with some minor changes including fixes to CBC block cipher mode. In 2008, TLS 1.2 (RFC 5246) was released with many major changes, such as the inclusion of AES, and the replacement of many instances of MD5 with SHA256. This was later refined to disable a SSL 2.0 downgrade. As of 2017, TLS 1.3 is currently a working draft.

Recall that the green lock icon in the address bar of a web browser indicates a secure connection. The browser is communicating in HTTPS, which is HTTP over the SSL/TLS protocol. This form of communication is secure. SSL and TLS are applied at the 7th layer in the application level protocol, between the **transmission control protocol** (**TCP**) and application. They were designed to be usable by any application (as opposed to OS support). Support can be added to the client and server applications through existing libraries. Many libraries are available, with multiple languages supported (for instance, OpenSSL). Its most common uses are for HTTPS (for browsing), VPN (for extending private networks), email, instant messaging, and voice over internet protocol (VOIP) such as Skype.

27.2 SSL and TLS Protocols

The underlying principle of SSL and TLS is a three step procedure:

1. **Handshake** involves establishing parameters for secure communication. This is done using public key encryption.

2. **Change Cipher** involves agreeing to switch to a newly negotiated cipher. There is confirmation that all parameters were exchanged correctly.

3. **Talk** involves talking securely using the parameters established during the handshake. This is done using symmetric cryptography, and is called **application data protocol**.

The **application data protocol** is where the two applications get to talk to each other. For instance, in HTTP, the browser will finally be able to tell the web server which page it wants, and the server then sends the contents of said page. It uses cryptographic primitives based on the parameters established during the handshake. Long messages are split into smaller fragments. For each fragment, it optimally compresses the data, calculates HMAC, encrypts all the data using the shared secret through a symmetric cipher, adds a header, and then transmits.

The **handshaking protocol** is used to negotiate the version of the protocol and the set of ciphers. It is needed to increase the chances of older software talking to newer software. Nonces are then exchanged. An agreement is made with regards

to a pre-master secret. Certificates and related information are exchanged for purposes of authentication. A master secret is then created from the pre-master secret and nonces. It can then verify that each side has the same parameters and that the handshake was not tampered with by MITM. It then switches to symmetric encryption.

The handshaking protocol is a complex process. **Client hello** consists of a random 4 bytes. The session ID is null for a new session or a new connection, and otherwise indicates a request to open a new connection on an existing session. This includes the client SSL version, supported ciphers ordered by preference, and supported compression algorithms ordered by preference. Some examples of supported ciphers include

$$TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA,$$

$$TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256,$$

$$TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256.$$

Server Hello consists of a random 32 bytes. The session ID is new or reused (if still in cache). It includes the chosen SSL version (which is the minimum of the client version and server version), the chosen cipher, and the chosen compression algorithm.

In the event that the chosen cipher requires a public key, the server sends a **certificate**. This is most commonly the case, since RSA, Fixed DH (Diffie-Hellman), and Ephemeral DH require pubic keys. Anonymous DH does not require a public key. The server actually sends an X.509 certificate chain. The **server key exchange** is not needed for some ciphers, but for others, provides extra information needed for the cipher to function. These are signed by the server. This is followed by a **certificate request** that is sent if the server also wants the client to authenticate. The server supplies allowed CA roots. However, this step is not common. **Server hello done** indicates that the client can now talk.

The client then sends the **certificate** if this was requested by the server. If the client has a valid certificate chain, it is sent. Otherwise, an empty list is sent. The **client key exchange** depends on the type of key exchange agreed on. If it was RSA, then the client generates a random pre-master secret and encrypts this with the server's public key. If DHKE was used, then the client sends enough information for both parties to then use DHKE to arrive a the same pre-master value. Lastly, **certificate verify** occurs since the client must send this if they are sending a client certificate. This contains a digital signature of all previous communication, and serves as proof of knowing the private key corresponding to the public key in the certificate.

The **change cipher spec protocol** is considered a separate protocol that occurs during the last stage of the handshake. **Change cipher spec** is a single byte denoting a switch to the newly negotiated cipher. **Finished** is sent under the new

encrypted cipher. The message is a hash of all previous messages and the master secret. The change cipher spec protocol is sent by both the client and the server.

27.3 Master Secret

The pre-master secret is 48 bytes, and is computed for each session during the handshake. It is derived differently for different ciphers, and is only used to compute the master secret. This means that it id deleted immediately afterwards. The **master secret** is also 48 bytes, and is always computed using the same algorithm. The master secret is computed by applying a **pseudorandom function** (**PRF**) to the pre-master secret, the client hello random, and the server hello random. The PRF is a deterministic function that generates random looking output of any desired length. In SSL, it is implemented by repeated application of HMAC.

Recall that the master secret is different for each session. The master secret is cached for faster session resumption. Both the client and server contribute to its creation. The master secret is used for generating encryption keys, HMAC secrets, and initialization vectors. Thus, the master secret, server random, and client random are used by the PRF once again to generate the required items listed above. If the master key is compromised, only one session will be affected, as in accordance with the principles of forward secrecy.

28 March 27, 2017

28.1 Web Security

All websites should use HTTPS. However, cryptography alone is not sufficient. We are often concerned with what the server really knows about the client, and what the client really knows about the server. Breaking SSL/TLS is quite difficult. The underlying cryptography, protocol, and implementation are generally very strong. These may be occasionally broken, but they are fixed quickly.

Some attacks take advantage of unsuspecting users or poorly designed software. **Typosquatting** or **URL hijacking** is a type of cybersquatting. It takes advantage of common typing mistakes. By registering a fake domain, one can create a look-alike site. They can then wait and harvest usernames and passwords. This can be through mistyped addresses, or phishing attacks containing links to a fake website.

Network attacks sniff unencrypted connections on existing WiFi or wired hubs. This can be accomplished by setting up free WiFi hotspots in popular locations. The attacker can then eavesdrop on unencrypted connections and harvest usernames and passwords. **ARP** or **DNS poisoning** can also be used. The Domain Name System servers translate domain names into IP addresses. If an attacker subverts the DNS server to report a fake IP address, then the attacker can install a website clone on the fake IP address. Alternatively, they may launch a MITM attack through a cloned site. By implementing a login screen, they can harvest credentials. They

can also report that the site is under maintenance for all other requests. If the user recalls that the site usually has a secure green icon, the attacker could

- Place a green icon somewhere on the page.
- Hack into one of the root CAs to sign their own certificate.
- Hack into the user's computer to install a fake root CA on the user's machine.

MITM attacks are possible even with HTTPS present. The attacker can tamper with traffic to a select site, or tamper with all traffic (to all websites). For MITM attacks with hybrid HTTP/HTTPS, the could

- Eavesdrop and record HTTP requests from the browser.
- Forward the requests to the HTTPS version of the real site.
- Obtain replies from the real site via HTTPS.
- Forward the results back to the browser over HTTP.

An example implementation can be found at "https://moxie.org/software/sslstrip/". An **Extended Validation Certificate** costs more money, but provides the user with an extra chance to spot a fake website.

Back to the MITM attack, we note that the attacker records HTTP requests. This means that the user is attempting to use HTTP to connect to a site running HTTPS. To navigate to a website securely, we should use an HTTPS bookmark, or type in the address bar with HTTPS. Some sites redirect HTTP requests typed in the address bar to HTTPS. This redirect happens via a message sent from the server to the browser. The attacker can easily intercept this message. Some sites are implemented using only partial HTTPS, which is cheaper, so users must browse most of the site in HTTP. Checkout and login are performed via HTTPS. However, it could be possible for an attacker to intercept the checkout button on the HTTP to supply their own page. It is clear that a user may not notice. Many smaller sites also outsource payment processes.

HSTS, the **HTTP Strict Transport Security** defines a mechanism enabling web sites to declare themselves accessible only via secure connections and/or for users to be able to direct their user agent(s) to interact with given sites only over secure connections. The policy is declared by web sites via the Strict-Transport-Security HTTP response header field and/or by other means, such as user agent configuration, for example. This prevents many MITM attacks, but a site must be accessible only through HTTPS. Furthermore, the protection is only activated on the second visit. It works best with the HSTS preload list, where browsers come pre-installed with HSTS lists, or through opt-in lists.

HPKP, the **Public Key Pinning Extension for HTTP** defines a new HTTP header that allows web host operators to instruct user agents to remember ("pin")

the hosts' cryptographic identities over a period of time. Pinning may reduce the incidence of man-in-the-middle attacks due to compromised Certification Authorities. PKP is meant to be used in conjunction with HSTS, but this is not necessary. It acts to prevent compromised CAs and fake certificates.

28.2 Server and Client

We recall that the server only knows about the client what has been relayed by SSL. If client-side certificates are used, then the server knows the full identity of the client. This can be used instead of a username/password, and is supported by many browsers. if client-side certificates are not used, then the server knows absolutely nothing about the client. SSL provides only a secure pipe, so while communication is confidential and integrity is protected, the user on the other side could be anyone.

The client on the other hand, receives the server's certificate. Someone is vouching for the binding of some name and attributes to a public key. Many browsers ship with multiple CA root certificates. If even a single one is compromised, security is lost. For instance, if someone convinces one of them to sign and sell a certificate for www.bank-of-montreal.com, a user may not notice that the real site is www.bmo.com.

While there is nothing wrong with SSL/TLS cryptography, the human factor is the issue. Most users do not know what a certificate is, or what to do about the warnings. Even if a user knows what a certificate is and how to verify it, it is difficult to discern what it should say in any given context. There is no bulletproof way of deciding whether to trust any given CA.

29 March 29, 2017

29.1 Browser Security

The attacker's attackers' goals are to steal information (passwords, credit card numbers) and to turn computers into bots. Since browsers are big applications, they act as a considerable weakness, since many lines of code corresponds to many bugs, leading to many vulnerabilities. Additionally, browsers serve **active content**, which includes Java, Flash, ActiveX, and other types of plugins. JavaScript is a source of many security holes. There is no well defined security model, and it is a crucial component in cross-site scripting attacks. JavaScript can be used to:

- Interact with arbitrary web sites and servers to send or receive data.
- Interact with other pages in the same browser.
- Interact with other scripts.
- Interact with the page elements.

- Detect key presses and mouse gestures.
- Access the camera and microphone.

AJAX is **Asynchronous JavaScript and XHTML**, and allows interactive web pages, or even a nearly desktop-like experience. It allows browsers to interact with the servers without a page reload. The user might not even be aware that communication is going on. This can be misused by adversaries. **Websockets** is a communications protocol that provides similar functions.

Overall, JavaScript contains cross-site vulnerabilities. This is ultimately based on misplaced trust, as the server developers trust their own client code, also trust other people's code (libraries). Browser bugs exist, and can be due to buffer overflows causing arbitrary code execution, API bugs causing the logging of keystrokes, or sandbox bugs.

Cross-Site Scripting or (**XSS**) occurs when sites do not sanitize user input to strip HTML. It exploits the user's trust in the site. There are two main types of XSS attacks:

- **Reflected XSS** requires some social engineering. Search site uses GET requests. The search site displays the searched term and all items found that match. An attacker may notice that the searched string was not sanitized, so they craft a URL with a script embedded in it. If the user clicks on the link while logged in, their credentials will be stolen. The submitted short script can also load other scripts.
- **Persistent XSS** may take advantage of a site allowing users to enter comments. These comments can include JavaScript code. These comments are then saved on the server, and are served to the other users along with the injected JavaScript. The injected JavaScript code can now access the protected resources. That is, it runs with the same privileges as legitimate JS code. The injected JavaScript can then transmit user?s authentication cookies to some other site without the affected user ever noticing. This is one of the most popular attacks on the web due to the user's tendency to reuse passwords.

To defend against these attacker, users can disable JavaScript. This is not feasible however, as many websites depend on it. We can convince developers to use POST instead of GET requests to ward off reflected XSS attacks, and to sanitize input properly (the best choice). Input sanitization is not trivial, especially with Unicode. White-listing is preferable to black-listing, as accepting something is better than denying a certain script. To defend in general, developers need to realize that their server code should not trust everything sent by the client.

Cross-Site Request Forgery (**CSRF**) exploits the website's trust in users. It involves nearly unnoticeable social engineering. The user needs to visit a malicious website while logged in to another site. This attacks requires a target site that

relies on a user's identity that is stored and sent by the browser. The website uses HTTP requests that have side effects. For example, to delete all files the user clicks a button, which executes:

$$action = deleteAllFiles$$

If a logged in user visits a page on a different website where an image for instance is linked to that website code (posted on a blog site for example), the user will notice that all their files are gone.

Servers are also tempting targets for defacement, distribution of malware to unsuspecting clients, and data theft. The defenses that the server has available are limited to not trusting the client, checking all inputs, and securing the server. Most websites use server-side scripts such as PHP, Python, Ruby, and JavaScript. Each such script is technically a separate network service. For a website to be secure, all of its scripts must be secure. Scripts must have limits on the damage they could cause. One must consider the security context in which the scripts are run, the protection of sensitive files from malfunctioning scripts, and whether users can run their own server side scripts. A partial defense would be some server side sandboxing, such as Apache's suexec, or the VM based Docker and LXC.

Injection attacks result from not sanitizing inputs. A buffer overflow bug in a script may permit an attacker execution of arbitrary code. Code and command injections may come from server code that executes system calls or external commands based on user inputs, or from malicious user input. In **SQL injection**, an attacker could execute arbitrary database queries. The type of result/error reporting to the browser can lead to blind SQL injection. A series of true/false SQL statements can incrementally reveal information in the database. An attacker can "fingerprint the relational database management system (RDBMS)".

30 March 31, 2017

30.1 Web Authentication

Generally, we have three options for website authentication:

1. **Client-Side Certificates** use SSL. These are handled by the browser/server directly. However, storing and protecting the private key is difficult. We need to determine where the key lives, and how it is to be moved between machines.

2. **Site/Application Specific** usually consists of a custom login screen. This is the most common form of web authentication. It is often implemented by amateurs.

3. **HTTP Authentication** is a part of HTTP. The two types are **basic** and **digest**. These are usually used on top of SSL. Both result in a browser dialog asking the user to enter their username and password. It is considered

unaesthetic. **Basic HTTP authentication** occurs when the server sends a challenge, and the user replies with their username and password in plaintext. **Digest HTTP authentication** occurs when the server sends a nonce, and the user replies with

$$hash(username, password, nonce, url)$$

to ensure that it is immune to replay attacks. Password storage is another distinction between the two forms of HTTP authentication. With basic authentication, UNIX style hashed passwords are used. In digest authentication, passwords must be stored in plaintext (this is true of most challenge response protocols). Thus, password files can be stolen. All of these apply to any custom authentication systems as well.

Generally, HTTP authentication is not used, as there is no fancy login screen (it looks different on different browsers), no easy recovery from authentication failure, and no password retrieval. Generally, it is only used by low-end websites. After the initial authentication is performed by password, we also generally require continuing authentication. Users do not want to enter their password for every action, so the solution is to authenticate a session. The two main ways to accomplish this are through cookies and custom solutions. The browser sends a token with each request, so the server decides what that token is, and how to interpret it. The fundamental issue however, is that tokens are sent by untrusted clients, so they could be forged. The solutions to this is to use server-side storage or cryptographic sealing. We recall that SSL is used throughout the entire process.

Server-Side Storage provides the client with a nonce. The server stores $(nonce, userID)$ or $(nonce, sessionID)$ in database. When the client sends back the nonce, the server looks up the identity. The server makes certain that nonces are not guessable or findable by exhaustive search while the clients make sure nonces are not (easily) stolen. However, server-side storage can be exhausted and sessions must have a limited duration. **Cryptographic Sealing** occurs after the user logs in. The server creates a token containing $userID$ or $sessionID$. The token is then encrypted and MAC'd using the secret key. For added security, a timestamp and IP address can be included. This is similar to Kerberos TGT. The token is sent to the client to remember. The client then sends back the token to server with each request. Only server can decrypt the token and verify the hash. The server then looks up the JSON Web Token.

31 April 3, 2017

31.1 Malware

Malware is short for malicious software. It is defined as software that is designed to perform unwanted actions on a computer system. This could include disrupt-

ing computer operations, gathering sensitive information, displaying advertising, stealing data, stealing CPU, disrupting operations, etc. There are many types of malware such as viruses, worms, spyware, adware, etc. They are generally categorized based on how they spread, for instance worms vs. viruses, and on what they do (payload), such as for annoyance or erasing a filesystem. Other categories depend on whether the malware requires a host program. That is, whether it is parasitic or self-contained. In the list below, the first three are distinguished by the means through which they spread, while the last four are distinguished by their payload:

- **Viruses** infect other programs and require user interaction to spread.
- **Worms** are similar to viruses, but spread automatically. They are also self-contained.
- **Trojan Horses** are disguised malware since they pretend to be useful software.
- **Ransomware** disables systems and demand payment for repair. This could be accomplished by encrypting files for instance.
- **Spyware** gathers and transmits information without consent. This would include keyloggers.
- **Adware** delivers unwanted advertising.
- **Scareware** scares the user into buying security services.

31.2 Viruses

A **computer virus** is simply a program that includes code for self-replication, includes a malicious **payload**, and includes a trigger. It **replicates** itself by modifying other programs. It accomplishes this by inserting itself (fully or partially) into other programs. The inserted code can then further replicate. It can be distinguished from other forms of malware due to its self-replication that requires some form of user assistance. For instance, the user has to run the infected software, insert a USB drive, or open an email attachment.

In 1986, Brain was the first virus to infect personal computers. It was developed by two brothers to protect their commercial software from piracy. The payload was to slow down floppy disk access. Many users did not realize they were affected. The virus came complete with the brothers' address, three phone numbers, and a message that told the user that their machine was infected and to call them for inoculation.

The three main phases of a virus are **propagation**, **trigger**, and **action**. A virus program would include code that infects a program by inserting the virus through self replication. Once the trigger condition is met, the virus would perform

a malicious action through execution of the payload. For instance, this could be erasing the filesystem or displaying an advertisement. This is all done before the original program is allowed to run. The virus can acts according to the following phases:

1. **Dormant phase**: The virus is doing nothing. It is simply laying low and trying to avoid detection. Not all viruses have this phase.

2. **Propagation phase**: The virus is replicating. It is finding and infecting new files, avoiding re-infection.

3. **Triggering phase**: Some logical condition causes the virus to transition to the action phase. This could be based on a date, user action, a propagation counter, etc.

4. **Action phase**: The malicious action is executed (payload).

Some infection types include **overwriting**, where part of the original code is destroyed, **pre-pending**, where the original code is retained (possibly compressed), **infection of libraries/system**, where the virus resides in memory, and **macro viruses**, where documents are infected. The virus may be inserted as a contiguous block, or separated into small chunks and stored throughout the infected program.

Virus developers often try to hide or conceal their viruses from detection. An **encrypted virus** uses a decryption engine and an encrypted body. A randomly generated encryption key is used, and detection looks for the decryption engine. Encryption is often a very simple algorithm, such as XOR. A **polymorphic virus** is an encrypted virus with random variations of the decryption engine (for example, padding code). Signatures are useless, so detection is performed using a CPU emulator and decryption or pattern analysis. A **metamorphic virus** uses different virus bodies. An approach for detection includes code permutation and instruction replacement. This type of virus is difficult to detect.

Macro and scripting viruses take advantage of the **active content** in documents. The most common example is Microsoft Word and Excel macros, or PDF documents. These were very common in the mid-1990s. This is because they are platform independent, they infect documents (instead of programs), often install in the main document template, and spread easily. These types of viruses can exploit the macro capability of Microsoft Office applications. More recent releases of MS Office have included built-in protection mechanisms. Various antivirus programs have been developed, so these are no longer a predominant virus threat.

In 1987, Fred Cohen demonstrated that there is no algorithm that can perfectly detect all possible viruses, since this is an undecidable problem, "In order to determine that a given program 'P' is a virus, it must be determined that P infects other programs. This is undecidable since P could invoke any proposed decision procedure 'D' and infect other programs if and only if D determines that P is not a virus. We

conclude that a program that precisely discerns a virus from any other program by examining its appearance is infeasible." Thus, our best form of protection against viruses is to prevent viruses from entering the system in the first place. This includes user education and technological measures. Prevention may fail, so we need another plan. **Antivirus software** is responsible for the detection and removal of viruses:

- **Signature based detection** revolves around a unique binary string in an executable (fingerprint). This is difficult to perform with advanced viruses.
- **File integrity check based detection** uses save file digests in a database. It periodically scans and compares the results.
- **Heuristic scanning** performs detection from behaviour. This includes emulation (running the tested program in a sandbox environment). It is advantageous because it allows for the detection f unknown malware. However, scanning and analysis take time, so this slows down the system considerably. False positives are also a problem. Antivirus programs will often quarantine the suspected file, instead of deleting it.

31.3 Worms

Worms are a form of malware that spreads copies of itself, usually over a network, without the need to inject itself in other programs. This is often accomplished without human interaction. The act of spreading is often the most harmful effect of the worm, since this exhausts network resources. Worms usually carry a malicious payload, such as deleting files or installing backdoors. The first worm was built in the labs of John Shock and Jon Hepps at Xerox PARC in the early 80s. Christmas Tree written in REXX, released in December 1987, targeted IBM VM/CMS systems, and was the first worm to use email service. The first internet worm was the Morris Worm, which was released on November 2, 1988 and written by Cornell student Robert Tappan Morris. Some notable worms are presented below:

- Melissa (1998): E-mail worm. First to include virus, worm and Trojan in one package.
- Code Red (2001): Exploited Microsoft IIS bug and probes random IP addresses. It consumed significant Internet capacity when active.
- Code Red II (2001): Also targeted Microsoft IIS, installs a backdoor for access
- Nimda (2001): Had worm, virus and mobile code characteristics. It spread using e-mail, Windows shares, Web servers, Web clients, and backdoors.
- SQL Slammer (2003): Exploited a buffer overflow vulnerability in SQL server. Compact and spread rapidly.

- Sobig.F (2003): Exploited open proxy servers to turn infected machines into spam engines.
- Mydoom (2004): Mass-mailing e-mail worm that installed a backdoor in infected machines.
- Warezov (2006): Creates executables in system directories. The worm sends itself as an e-mail attachment. It can disable security related programs.
- Conficker (2008): Exploits a Windows buffer overflow vulnerability. It was the most widespread infection since SQL Slammer
- Stuxnet (2010): Restricted rate of spread to reduce chance of detection. It targeted industrial control systems.

Unlike viruses, worms propagate by finding and infecting vulnerable hosts. They need a way to tell if a host is vulnerable, and a way to determine whether the host is already infected. Initially, there is slow initial growth. This phase does not last long, as it soon gives way to exponential growth after the initial phase. When there is nothing left to infect, it slows down. It follows logistic growth.

32 April 5, 2017

32.1 Worm Development

To develop a worm, vulnerabilities are first identified. This includes software bugs and vulnerabilities in trusted patterns (email addresses). Code is then written to exploit the vulnerabilities. A large target list is then generated. This includes some aspect of unpredictability and randomness. Often, they target local computers. The next step involves installation and execution of the payload. The payload is often downloaded later, and activated synchronously. The next step is querying and reporting whether the host has already been infected. The last step is stealth mode.

32.2 Trojan Horse

A **Trojan horse** (or **Trojan**) is a malware program that appears to perform some useful task, but which also carries a malicious payload. It uses social engineering for spreading. It tricks the user into assisting in the compromise of their own system. Trojan horses are often installed by a user or administrator, either deliberately or accidentally. Examples include mobile phone trojans.

To defend against malware, we follow these general steps:

- Keep the operating system updated.
- Keep other software updated.

- Install antivirus software.
- Update antivirus software.
- User education.
- Deploy firewalls.
- Deploy intrusion detection systems.

32.3 Botnets

Apart from ransomware (disabling systems in demand of payment for repair), spyware (gathering and transmitting data), adware (delivering advertisements), and scareware (scaring users into buying their security service), some other forms of malware payloads include **backdoors**, which provide the attacker access to the system, and **bots** or **zombies**, which allows the host to be remotely controlled and join a botnet.

A **botnet** is an interconnected network of computers infected with malware without the user's knowledge and controlled by cybercriminals. They are typically used to send spam emails and to perform distributed denial of service attacks. Botnets can be built via payloads of other malware such as viruses, worms, and trojans. Drive-by downloads utilizing bugs in browsers can result in software installed on a machine just by visiting a website. It includes automated exploits to penetrate machines. One can buy or rent botnets, and even have them custom built. It is also possible to steal them as several attackers attempt to steal each other's bots - some bots will patch security holes on host systems to minimize this. Many modern bots are remotely upgradable. That is, they download new versions, fix bugs, change encryption parameters, and download new payloads (DDoS, email spam, scanning). The primary uses of botnets are for financial gain, email spam, distributed denial of service attacks, extortion, hactivism, revenge, spyware, click-fraud, further infection, bitcoin mining, combined attacks, diversion, or interruption of security mechanisms.

Distributed Denial of Service (**DDoS**) attacks were first seen in 1999. There was evidence of a big one planned for Y2K, but this never happened. Today they are the most common form of DoS attack. Most of them exhaust network bandwidth by using a large number of compromised computers (called zombies or bots), usually in the range of 10000 to 30000000, connected through a network. A control node is used to issue commands to the bots, while the bots do the actual work. Common communication channels include IRC and P2P networks. These types of attacks are hard to trace back.

A **denial of service** attack is a cyber attack on a system with the aim to deny service to legitimate users, and to disrupt normal operations. Typically, they are executed over a network by flooding a server with illegitimate requests. These are usually performed against high-profile organizations or companies. DoS targets

network bandwidth by clogging up the network. This is made possible when sending messages is possible at higher rates than the ability to received them. DoS may also target the CPU. For instance, by making the receiver try to perform expensive decryption or signature checks. Memory can also be targeted, as it could be forced to keep many connections in open state. Note that this does not necessarily mean all of physical memory. Disk space may be attacked by filling up logs. Any finite resource can be exhausted.

33 April 7, 2017

33.1 Network Security

Computers communicate through interconnected hubs, switches, and routers. This is performed through a combination of circuit and packet switching:

1. **Circuit switching** is used in legacy phone networks. It provides a single route through a sequence of hardware devices, established when two nodes start communication. Data is sent along the route, and the route is maintained until communication ends.

2. **Packet switching** occurs when data is split into **packets** before it is sent. Packets are transported independently through the network. Each packet is handled on a best efforts basis. Packets may follow different routes, but are eventually re-assembled on delivery.

Thus, the main difference is that circuit switching sends all the data through a single channel, while packet switching sends the split data through different channels before it is reassembled at its destination.

A **network protocol** defines the rules for communication between computers. Protocols are broadly classified as **connectionless** and **connection oriented**:

1. **Connectionless protocol** send data out as soon as there is enough data to be transmitted. It is advantageous because there is low overhead, but can result in data loss, duplication, and deliveries that are out of sequence. Some examples include user datagram protocol (UDP) and Internet Protocol (IP).

2. **Connection oriented protocol** provide a reliable connection stream between two nodes. It consists of set up, transmission, and tear down phases. It emulates a circuit-switched network. Its advantages and disadvantages are the reverse of connectionless protocol. An example would be transmission control protocol (TCP), a protocol that is implemented on top of IP.

A **network packet** typically consists of **control information** for addressing the packet, either in the header or sometimes in the footer/trailer. This can include the origin address, destination address, and other information. A **payload** is a

actual data that is sent. Visually, a packet can be thought of as consisting of a frame header, an IP header, a TCP header, the data, followed by the frame trailer.

Network models are typically implemented as a stack of layers. Higher layers use the services of lower layers via encapsulation. A layer can be implemented in either hardware or software, with the bottom-most layer often implemented in hardware. A network device may implement several layers. The communication channel between two nodes is established for each layer. The actual exchange occurs through a channel at the bottom layer, with virtual channels at higher layers.

The internet for instance, consists of virtual channels between the server, through routers, to your computer. The application layer, transport layer, network layer, and link are the highest to lowest software levels in the server and client. The server connects to the physical layer through the link. The ethernet of the physical layer connects this to the link of a router with only the link and network layers. Fiber optics in the physical layer transfer data to another router with similar layers, which is then received by the client through Wi-Fi in the physical layer. The physical layers always connect to the link layer in software in each component.

33.2 Packet Routing

To deliver a packet, we need route finding mechanisms. This is implemented as route finding protocols. There are different protocols for LANs and WANs.

- In a **local area network**, these packets are routed using switches and a **MAC** address (media access control). a MAC address is a 48 bit number usually represented in hex. For instance,

  $$00 - 1A - 92 - D4 - BF - 86.$$

  The routing protocol used is the **Address Resolution Protocol** (**ARP**).

- Over the internet, these packets are routed using routers along with an **IP address** and **port numbers**. IP addresses are 32 bit numbers, such as

  $$136.159.7.7.$$

  Some routing protocols used are **Border Gateway Proctcols** (**BGP**) and **Open Shortest Path First** (**OSPF**).

33.3 Local Area Routing

A switch is a common network device that operates at the link layer. It has multiple physical ports, each connected to a computer. The switch operates by first learning the MAC address of each computer connected to it. It then forwards frames only to the destination computer. Older hubs simply forwarded traffic to all connected hubs. Switches can be combined by arranging them into a tree structure. In this

case, each port learns the MAC addresses of the machines in the segment (subtree) connected to it. Fragments to unknown MAC addresses are broadcast, while frames to MAC addresses in the same segment as the sender are ignored.

MAC address filtering occurs when a switch is configured to provide service only to machines with specific MAC addresses. Allowed MAC addresses need to be registered. This can be done manually by a network administrator, or by an automated system (username and password). A **MAC spoofing attack** impersonates another machine. It does this by finding the MAC address of the target machine, then reconfiguring the MAC address of a rogue machine (this can be done easily in Linux and Windows). The target machine is then turned off or unplugged. To prevent against these spoofing attacks, one can block part of the switch when a machine is turned off or unplugged. One can also monitor and disable duplicate MAC addresses.

ARP stands for **Address Resolution Protocol**. It connects the network layer to the data layer by converting IP addresses to MAC addresses. ARP works by broadcasting requests and **caching** responses for future use. The protocol begins with a computer broadcasting a message asking the entity with a certain IP address to tell this to an entity with another IP address. When the machine with the particular IP address receives the message, it broadcasts a response, claiming that it has a certain MAC address. The requestor's IP address is contained in the packet header. The Linux and Windows command

```
arp -a
```

displays the ARP table. This includes the Internet Address (IP), the Physical Address (MAC), and the Type (many are dynamic).

The ARP table is updated whenever an ARP response is received. Requests are not tracked, and responses are not authenticated. Thus, the ARP protocol assumes that machines trust each other. As a result, a rogue machine can spoof other machines. This is known as **ARP spoofing** or **ARP cache poisoning**. Almost all ARP implementations are stateless. Therefore, the ARP cache updates every time it receives a reply, even if it did not send any ARP request! It is possible to poison an ARP cache by sending gratuitous ARP replies. These unsolicited replies can be useful at times, such as by dynamically replacing a failed server. However, an attacker can easily convince other machines to send their traffic to the attacker instead.

Under normal routing conditions, the LAN user connects to the hub or switch, which then connects to the LAN gateway before connecting to the internet. Routing that is subject to ARP cache poisoning now contains the additional connection between the hub/switch to a malicious user, thus permitting interference.

34 April 10, 2017

34.1 Internet Routing

The **Internet protocol** (**IP**) is a communication protocol for relaying packets across network boundaries, thus enabling internet. Networks are connected using routers. A **router** is a networking device that forwards packets to another router. This is somewhat similar to a switch, but interconnects networks. IP uses numeric addresses for routing. These are 32 bit numbers such as 136.159.7.7. There are typically several hops in the route. The path that the packets follow is decided using BGP/OSPF protocols. The following are some examples of IP vulnerabilities:

- **Plaintext transmission** means that **eavesdropping** is possible at any intermediate host during routing. This concerns confidentiality.
- **No source authentication** means that the sender can **spoof the source address**, thereby making it difficult to trace an attack.
- **No integrity checking** means that the entire packet (header and data) can be modified, allowing MITM attacks.
- **No bandwidth constraints** means that a large number of packets can be injected into the network to launch **denial-of-service** attacks.

The **Transmission Control Protocol** (**TCP**) complements the IP protocol. It guarantees reliable data transfer, and automates error detection, retransmission, packet splitting and reassembly. It also implements congestion control, adds port numbers, etc. Most popular application protocols are built on top of TCP, including HTTP, SSH (Secure Shell), SSL/TLS, and FTP (File Transfer Protocol). When a TCP packet is received, the sender gets an ACK (Acknowledgement) receipt, thus making the protocol reliable. A TCP packet has a sequence number that is used to order the packets and discard duplicates. TCP uses a checksum to detect errors in transmission. However, there is very little security built into TCP. To establish a TCP connection, we need to go through a three way handshake:

1. The client requests a connects by sending a SYN packet.
2. The server responds by sending a SYN/ACK packet.
3. The client sends an ACK packet.

34.2 Network Attacks

A **SYN flood attack** was an early internet DoS attack that was first discovered in 1994. The attacker sends many SYN packets to the server. This type of attack uses forged source addresses. The SYN-ACK reply has nowhere to go, or goes

unanswered. The server never receives the final ACK. Thus, the server keeps the connection in a half-open state, so the server's memory eventually fills up.

ICMP - Internet Control Message Protocol - is used for error messages and diagnostics. An example would be a PING message that is used to test the reachability and round-trip time of a host on the internet. A **PING flood attack** (**ICMP flood attack**) aims to overwhelm the network of a target organization. The attacker sends many PING messages with spoofed source IP addresses. The victim will be busy replying to non-existent senders. This is not a common attack anymore, as it is quite costly for the attacker and requires that the attacker's bandwidth is greater than the victim's bandwidth.

DDoS PING flooding is a type of **Smurf attack**. It is a DDoS attack that uses a broadcast mechanism built into IPv4. Packets sent to a special address are distributed to all hosts in an IP range. For example, a packet sent to IP address 136.159.55.255 could be sent to all hosts 136.159.55.*, where the last three digits could change. An attacker can therefore send a PING request to a router allowing broadcast. The attacker spoofs the source address, thus aiming it at the victim.

34.3 Defense Against Network Flooding

Flooding attacks are usually classified based on the network protocol that is used. The intent is generally to overload the network capacity. Almost any type of network packet can be used. Currently, there is no perfect solution that could offer complete protection against all attacks. Some current strategies include filtering traffic using firewalls (block any PING requests from outside) and using cryptographic solutions (such as SSL/TLS and IPsec).

Cryptographic solutions are applied above the network layer, such as SSL/TLS and SSH. This protects against connection hijacking, data injection, and eavesdropping. However, it does not protect against DoS attacks by spoofed packets. At the network layer, **Internet Protocol Security**, (**IPsec**) also protects against IP address spoofing. IPsec is essentially the IP protocol along with additional cryptography. It is similar to SSL/TLS, but works at network layer. Each packet is authenticated and encrypted, including data and headers. IPSEC therefore provides data integrity, mutual authentication, replay protection, and confidentiality. It does not prevent DDoS attacks directly, but makes them more difficult.

DDoS defense is still an active area of research on both ends. So far, there is no comprehensive solution. Some heuristics are starting to emerge, but most defenses are on a case by case basis:

- **Over Provisioning**: Use bigger pipes than needed, in order to ride out the attack.
- **Black-Hole Routing**: Use a router at the **Internet Service Provider** (**ISP**) that can direct the traffic of an attacked server to nowhere. The service goes offline for a while, but the rest of network is unaffected.

- **Anomaly Filtering**: Detect anomalous patterns, then instruct a router upstream (ISP) to filter traffic based on the anomaly.
- **Rate Limiting**: If router output is overloaded, then inspect the input links and tell the upstream routers to rate limit connections. This process is repeat recursively through a pushback mechanism. Thus, for routes where there is heavy traffic flow directed to the server, a pushback message is sent from the server back through the paths with heavy traffic.

34.4 Firewalls

A **firewall** is an integrated collection of security measures designed to prevent unauthorized electronic access to a networked computer system. It act as a barrier between a private network and a public network to limit communication with the outside world. It can be deployed as software or dedicated hardware. Firewalls offer a good amount of security for relatively low cost. Typical firewall setup is described below:

- For a dedicated hardware firewall, a local network is connected to a main hub or switch. This then connects to the internet after passing the firewall.
- For a dedicated firewall/router, the computers are directly connected to router and firewall. This then permits connection with the internet.
- For firewalls running as software, each device with the software firewall is directly connected to the internet.

Firewalls follow a set of rules that are usually arranged in a table. The Direction, Src IP, Dst IP, Src Port, Dst Port, and State are listed, along with an Action to take (such as Reject, Accept, or Drop). The first rule that matches is applied, and the specific action is taken.

We use firewalls because most software contains bugs. Most software has security vulnerabilities, and most computers have security holes. Thus, computers accessible from outside networks are security risks. Computers connected to the outside world should be protected. Firewalls are based on much less code, and hence have fewer bugs. They can be centrally or professionally administered, and are a great place to do more monitoring and logging. They can partition a network into separate security domains, with each domain having it's own security policy.

Typical firewall deployment in an organization assumes that everyone on the LAN is a good guy, and that bad guys live on the WAN. The **demilitarized zone** (**DMZ**) contains necessary servers that are potentially dangerous, such as mail and web servers. The hosts on LAN can access DMZ, but the DMZ has limited access to LAN. DMZ is protected from the outside via firewall. Thus, computers connected to a LAN Switch must communicate to the DMZ network connected to another

LAN switch through an internal firewall. This other LAN switch is connected to an external firewall, which passes a boundary router before reaching the internet.

34.5 Firewall Approaches

Firewalls are used to enforce policy. These policies often reflect administrative boundaries. This can take the form of internal firewalls between domains, such as administrative, research, and students.

- Firewalls can be designed to block only dangerous traffic. This is a less disruptive option, but not a very good approach in general as it requires the system administrator be smarter than the attacker.
- Another approach is to use firewalls that block everything by default, only allow necessary traffic through. This tends to be more disruptive, but is also much more secure.

Many organizations permit all outbound traffic, while some organizations decide to limit outbound traffic. This could be for libraries for child-friendly internet, regulatory requirements, or cutting down on social media activities. Firewalls can limit outgoing traffic as effectively as inbound traffic.

35 April 12, 2017

35.1 Firewall Types

There are many types of firewalls, some of which are listed below. Many firewalls are a combination of these types:

- **Simple packet filters** were the original firewalls. They are very cheap, as individual packets are inspected and then either accepted or rejected. Rejection means that they either silently ignore (drop), or send an error response (reject). Packet header fields inspected for source and destination address, source and destination ports, and protocol and TCP flags. Simple packet filters mostly works at the network (IP) layer, with a little bit of peeking into the transport (TCP) layer (ports and flags). There is no notion of state or sessions, so it does not work well with protocols like File Transfer Protocol (FTP and Remote Procedure Call (RPC). In this scheme, we are concerned with how to handle outgoing connections without state. If we want to permit outgoing connections, we have to permit reply packets, but we need to do this without managing states. Recall the TCP handshake: SYN, SYN-ACK, ACK. After that, all packets have an ACK-flag set in the header. The solution is to allow all inbound packets with the ACK-flag set.

- **Stateful packet filters** are the most common type of packet filter today. The firewall maintains per-connection state (requires some amount of RAM), where the states may be **new** or **established** connections. When a packet is sent out, the firewall records it in an internal state table. When an inbound packet arrives, it can be looked up and associated with a state. Most firewalls have limited memory, so this can be used for DoS attacks. Stateful packet filters solves many problems of simple packet filters, as they can handle User Datagram Protocol (UDP) query/response, and associate ICMP packets with a particular connection. However, they are still not able to handle RPC. Many firewalls come with extra tools (helpers) to handle more complex protocols
- **Application layer firewalls** are not discussed in much detail.
- **Circuit level gateway** are not discussed in much detail.
- **Personal and distributed firewalls** are not discussed in much detail.

35.2 Firewall Strengths and Weaknesses

Many firewalls have **network address translators** (**NAT**) functionality built in. This translates the source address and port numbers. Thus, some or all hosts behind the firewall can have unroutable IP addresses (private range). The primary purpose of NAT is coping with the limited number of global IP addresses. It also adds extra security, as the addresses behind the firewall can be hidden from the attacker. Network reconnaissance is made more difficult.

Many draconian administrators and organizations only allow HTTP traffic. As a result, a lot of newer software can be configured to run over HTTP ports. Since HTTP usually gets through most firewalls, firewalls are increasingly less effective but not yet useless. They still offer a good amount of security for not much additional cost.

Firewalls are effective, since they define a single point for monitoring and blocking traffic. A single firewall can provide some protection for an entire local network. It is also a fairly inexpensive solution that solves many problems (but not all problems). However, firewalls also come with many inherent limitations. Firewalls are not some magical solution to all network security related problems. Firewalls are a response to the fact that we do not know how to write software that is secure, correct, and easy to administer. Better network protocols will not eliminate the need for firewalls. Firewalls cannot protect against attacks bypassing the firewall, against insider attacks (for instance, someone installing WiFi on the inside), or against devices that are already infected.

35.3 Domain Name System

TCP/IP operate on numeric IP addresses, but humans remember words easier than numbers. For instance, users prefer to type in "www.google.com" instead of "172.217.23.228". However, the browser needs to know the IP address, because it uses TCP to communicate. To resolve this, your browser allows you to type in the url, but it then contacts a DNS service to get the corresponding IP address. **Domain Name System** (**DNS**) is system for mapping domain names to IP addresses. The client contacts the DNS server requesting the IP address for a particular website.

Domain names are two or more labels, separated by dots. For example, "http://www.harvard.edu." or "http://www.seas.harvard.edu." are domain names. The rightmost label (ca.) is the **top-level domain** (**TLD**). The DNS system is organized into a hierarchy of name servers based on domain names. The root server (.) connects to the TLD server (.edu), which connects to the domain server (harvard.edu.), followed by the subdomain server (.seas.harvard.edu.)... Servers communicate with each other (up or down).

Since it is impractical for a single DNS server to know the entire internet, these DNS servers are organized into a hierarchy (tree structure). There are thirteen logical root servers, with hundreds of physical root servers (for performance and redundancy). A resolution method is required when the answer is not in the cache. The client asks the ISP DNS server. This then asks the root name server, which responds by directing the ISP DNS sever to ask the top level domain. The ISP DNS server repeatedly asks each successive subdomain until it obtains the numeric IP address.

35.4 DNS Security

DNS Hijacking involves changing the IP address of a DNS server on a computer either via trojan horse or via hacking an insecure router. An attacker can redirect traffic to a spoofed site (pharming), or execute a MITM attack. To defend against this:

- Do not download random software.
- Install antivirus.
- Keep your router updated.
- Switch to Google's public DNS service.
- Switch to Google's public DNS-over-HTTPS service. This can be found at "https://github.com/behrooza/dnsd"

There would be too much network traffic if the DNS tree is traversed for each query. The root zone would be rapidly overloaded. Thus, the DNS servers cache

results for a specified amount of time. Operating systems and browsers also cache DNS results. Unfortunately, DNS communication was not designed to be secure. **DNS cache poisoning** occurs when the DNS servers are sent false records to cache. DNS security uses a 16 bit request identifier to pair queries with answers. The cache may be poisoned when a name server disregards identifiers, has predictable identifiers, or accepts unsolicited DNS records. To prevent this from happening, we can either fix the above problems, or deploy DNSSEC.

DNS Security Extension (**DNSSEC**) is DNS combined with digital signatures. It guarantees DNS reply origin authentication, integrity of reply, and authenticity of denial of existence. DNSSEC accomplishes this by digitally signing the DNS replies at each step of the way through a mechanism similar to root certificates. It is still not fully deployed, even though the initial **Request for Comments** (**RFC**) was in 1997, due to critical mass, politics, and cost. In 2016, the resolvers performing exclusive DNSSEC are about 15%. The Google Public DNS implements DNSSEC validation by default.

35.5 Virtual Private Network

A **virtual private network** (**VPN**) allows private networks to be safely extended over the internet. It creates virtual point-to-point connections, usually through encrypted tunnels. Many implementations are based on SSL/TLS. If encrypted, it provides data confidentiality, integrity, and authentication. When connected via VPN, remote resources are available the same way as if connected locally. VPN are implemented on many routers. There are also many open source implementations, as it does not require special application support. VPNs can be categorized as:

- **Point-to-Point** allows a single host to connect remotely to a private network. An example would be an employee accessing company resources from home. Another example would be a student accessing the university library's resources while on vacation.
- **Site-to-Site** provides a secure bridge between two or more physically distant networks. An example would be making two offices appear to be on the same local network.