CSCD27 - Introduction to Computer Security

Introduction to Security

- Security issues are commonly caused by:
  - Bugs (buffer overflows, x-site scripting, etc.)
  - Insecure Configuration (improper auth, mediation, etc.)
  - No security by design

Security and Design

- **Safety:** For reasonable inputs, get reasonable outputs
- **Security** For unreasonable inputs, get reasonable outputs

CIA Properties of Security

Term	Idea
Confidentiality	Information is disclosed to legitimate users
Integrity	Information is created or modified by legitimate users
Availability	Information is accessible to legitimate users

- **Anonymity:** Do not record identity of user that performed action
- **Non-repudiation:** Someone cannot deny having done an action
- **Accountability:** Knowing that someone has done an action
- Security is often a compromise and engineered
  - **Risk Analysis:** Inferring what can go wrong with the system and creating a set of security goals
    - `You can't prevent, only lower risk`
  - **Risk Exposure:** Probability * impact

Cryptography

Classical Cryptography

- Communication has several threats,
  - **Interception:** (read messages) Confidentiality
  - **Modification:** (modify messages) Integrity
  - **Fabrication:** (inject messages) Integrity
  - **Interruption:** (stop/block messages) Availability

Defintions

- **Caesar Cipher:** One of the oldest cryptosystems, a substitution cipher
- **Plaintext:** Message in clear form

- **Ciphertext:** Message in ciphered/encrypted form
- **Encryption:** Transform plaintext to ciphertext
- **Decryption:** Transform ciphertext to plaintext
- **Cryptographic algorithm:** Method to do encryption/decryption
- **Cryptographic key:** An input variable used by algorithm above to do transformation
- **N-bit security entropy:** The number of bits necessary to encode the number of possible keys
    - i.e. Caesar cipher's key is v where v denotes character shift #. Since 26 == 0 for shift, there are 25 total possible keys < 25 = 32, so, Caesar cipher has 5-bit security entropy

Kerckhoff's Principle

- The enemy knows the system. That is, a cryptosystem should be secure even if everything about the system is known except for the key
    - (can't rely on them not knowing what type of encryption there is!)

Types of Cipher Attacks

Term	Idea
Exhaustive Search	try all possible keys
Ciphertext Only	you know one or several random ciphertexts
Known Plaintext	you know one or several random plaintext and corresponding ciphertexts
Chosen Plaintext	you know several pairs of chosen plaintext and corresponding ciphertexts
Chosen Ciphertext	you know one or several pairs of plaintext and their corresponding chosen ciphertext

- Example attack on Caesar
    - You can use statistical cryptanalysis, monoalphabetic ciphers do not change freq. of characters

---

Evolution of cryptography

```
> substitution > transposition > polyalphabetic > mechanization > public key
```

- **Substitution ciphers:** Mono-alphabetic cipher (Permutation of alphabet)
    - Like vigenere cipher, j
- **Transposition cipher:** Switch letters around a permutation (key being set of permutations)
    - Like XORing message with secret
- **Polyalphabetic:** Just add word (key) to message
    - Ex. Vigenere Cipher
- **One-time Pad:** Perfect cipher, very hard to use in practice
- **Mechanization:** Stuff like the Enigma Machine and the telegraph

Modern Cryptography

Three core methods of Cryptography

Term	Idea
Diffusion	Mixing-up symbols
Confusion	Replacing a symbol with another
Randomization	Repeated encryptions of the same text are different

Functional Requirements

- `Dk(Ek(m)) = m`
  - Decrypting an encrypted text of message M using the same key K yields the message M
- `Ek(m)` is easy to compute (polynomial/linear)
- `Dk(m)` is easy to compute
- `c = Ek(m)`
  - Finding m is very difficult without k (exponential)

Symmetric Encryption

Stream Cipher

- **Characteristics:** Earlier, faster, large volumes of data
- **Typical idea:** Sse IV, or Initialization Vector to act as an additional randomization factor

Examples

- **XOR Cipher:** Modern version of Vigenere, using XOR to combine message and key, but prone to known-plaintext
- **Mauborgne Cipher:** Uses a random stream as encryption key, problem is key-reused attack
- **Rivest Cipher 4 (RC4):** 8 cycles/byte (fast), 40-2048 bits key – BROKEN in 2015
  - WEP (wired equivalent privacy)
  - RC4_key = IV + SSID_password, transmitted in clear
  - 50% chance of same IV being used again after 5000 packets
- **Salsa20:** 4 cycles/byte (very fast), 128/256 bits key

Block Cipher

- **Characteristics:** Later, slower, more secure
- **Typical idea**: Combines confusion (substitution) and diffusion (permutation), not vulnerable to known-plaintext

Encryption Modes

- **Data Encryption Standard (DES):** 50 cycles/byte (slow), 56 bits key – withdrew as standard in 2004
  - Brute forced in 1998 in days, 250K, and hours in 2006, 10K$
  - 2DES is bad because you can make lookup tables
  - 3DES is actually very very good, used in PGP, TLS/SSL, etc.
    - Extremely slow

- **Advanced Encryption Standard (AES):** 18-20 cycles/bytes, 128/192/256 bits, adopted in 2001
  - **ECB (Electronic code book):** Each block is encrypted independently with the key
    - Blocks can be parallelized but same block is encrypted to same plaintext
  - **CBC (cipher block chaining):** Each block is encrypted using randomness from previous block, can't be parallelized
  - **CTR (Counter):** Randomness using a counter, there's high entropy and parallelism, but sensitive to key-reused

Cryptographic Hashing

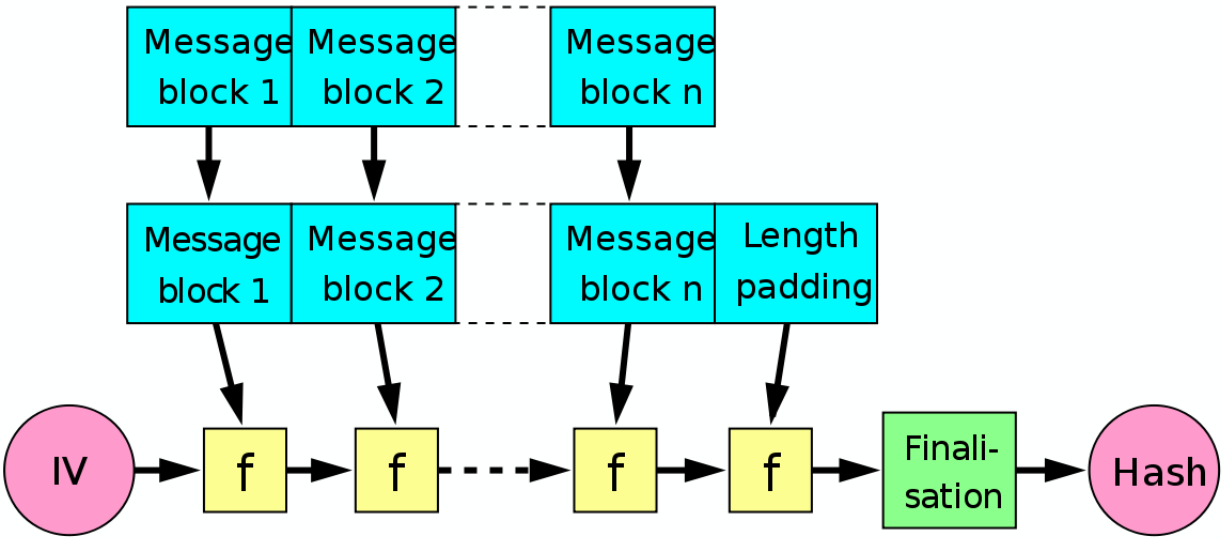
- `H(mn) = m'n'` is a hash function if:
  - H is a one way function
  - n (bit len) is unbounded
  - n' is short
- **2 types of hashing functions**
  - **Non keyed (IV is fixed)**
    - `H(mn) = m'n'`
  - **Keyed (IV is the key)**
    - `Hk(mn) = m'n'`

Characteristics

Term	Idea
PR - Pre-image Resistance	given H, x – hard fo find m, original message
2PR - Second Pre-image Resistance	given H, m, x, hard to find m' such that H(m) = H(m') = x
CR - Collision Resistance	given H, hard to find m and m' such that H(m) = H(m') = x

Merkle-Damgard construction

- A way to build CR hash functions from one-way CR compression functions. If H is CR, then M-D is CR



Message Authentication Code (MAC)

- Used to confirm message came from stated sender (authenticity)
    - Uses a keyed hash, `MACk(m) = Hk(m)`
    - Vulnerable to some Hash length extension attack
        - **Example:** `MACk(m || m') = H(MACk(m) || m')`
        - **Prevention:** Envelope method `MACk(m) = H(k || m || k)`, or padding method, `H((k XOR opad)||(k XOR ipad)||m)`
- We can ensure:
    - Confidentiality with Encryption (illegitimate users can't read)
    - Integrity with HMAC (hashed Message Authentication Code, stamp of approval)
- We can ensure both, with Authenticated Encryption
    - AEk(m) = Ek(m) || Hk(m)
- Basically appends the certificate/hmac to the end of the message

Replay Attacks

- Just uses the same message + HMAC as before, can lead to devastating results (e.g. withdraw $100 x 100)
    - Countered by:
        - Using a double nonce (random number)
        - Timestamps

Challenges

- How do they agree on the key?
    - **Key Distribution Center: ** When A/B want to talk, KDC can generate new keys and give it to them. It must be trusted, and it's a SSOF
- **Needham Schroeder Sym Key Protocol:** Gives a certain set of keys to Alice, Mallory, Bob. Can be broken and fixed (see slides)
- **Trust Models**
    - Decentralized Trust Model (web of trust, like GnuPG)
    - Centralized Trust Model (public key infrastructure, like TLS)
- **Web of Trust:** Alice should only trust Bob's key by fingerprint, either by Bob or by someone who already trusts Bob
- **Public Key Infra:** The browser should verify the certificate against certificate authorities (root, intermediary CAs)

Asymmetric Encryption

Characteristics

- Encoding, Decoding, and generating keys is trivial.
- Finding message or finding matching key is very hard.

Terms

- **Public Key:** Kp for encryption
- **Private Key:** Ks for decryption
- **Handshake:** Dks(Ekp(m)) = m

RSA

- Dependent on prime number theory
- **IDEA:** Use RSA to encrypt a shared key, use AES to encrypt message using that key
- **Other asymmetric schemes:** Diffie-Hellman, El-Gamal, Elliptic Curve Cryptography

Digital Signatures

- Use public cryptography to sign and verify things
- `m || SIGKsa(m)` where `SIGKsa(m) = EKsa(H(m))`
- This has the properties of integrity and nonrepudiation
- Transport Layer Security works this way. It provides confidentiality and integrity

Conclusion

- **Symmetric:** Fast, but needs key agreement
- **Asymmetric:** Slow, but doesn't need key agreement

Internet (In)security

Layer Vulnerabilities

- **Communication Protocol:** How communications should take place, usually defines data encoding, message sequence, etc.
- Established by IETF (Internet Engineering Task Force)

Internet Layers

Application Layer

- **BGP (Border Gateway protocol):** BGP is the protocol for establishing routes for Ip messages
  - **Route Hijacking:** Attacker can advertise fake routes
- **DNS (Domain Name server):** Translates domain names into IP addresses
  - **DNS Cache Poisoning:** Attackers can advertise fake DNS information

Transport Layer

- Collection of protocols to allow end-to-end connections
- Attacker can,
  - Determine open hosts by using 3-way handshake
  - Flood server by spawning new listeners using 3-way handshake
  - Guess current sequence number for existing connection and send reset flag to terminate connection
- **UDP (User Datagram Protocol)**
  - No acknowledgement, flow control, guarantee, etc. Used for media streamingprimarily
  - When UDP is received on non-opened port, host replies with ICMP – Destination Unreachable.
  - They can send large number of UDP to all ports, done in the Low orbit Ion cannon attack

Network Layer

- Collection of protocols to connect networks together, how messages are routed through networks based on different IP addresses
- **ICMP (Internet Control Message protocol)**
  - Exchange information about the network, error reporting, reachability
- Attacker can,
  - Scan entire network to find IP addresses of active hosts (nmap)
  - Generate raw IP packets with custom IP source fields
  - Split 64K payload and overflow a buffer
  - Overwhelm a host by sending multiple ICMP echo requests
- **ARP (Address Resolution Protocol)**
  - This is a mapping between MAC and IP addresses. Hosts broadcast IP/MAC to others to build table
  - **ARP Cache Poisoning:** Can broadcast fake IP-MAC mappings to the other hosts on the network

Link Layer

- Collection of protocols to connect hosts through a medium (copper, fibre, air)
- Media Access Control (MAC) Addresses are physical addresses - how hosts are connected to mediums
- Attacker can use network interface in promiscuous mode to capture (sniff) all traffic, even if its not to its MAC address (wireshark)

Internet Attacks

Common Attacks

- Scanning (surveying network and its hosts)
- Eavesdropping (reading messages)
- Spoofing (forge illegitimate messages)
- DOS (denial of service)
  - Asymmetric
  - Cheap for attacker, expensive for victim due to protocol amplification

Internet Protection

Transport layer security TLS

- Provides integrity and confidentiality
- 2-10x slower than insecure TCP connection
- Not used in practice to secure DNS/BGP

Preventing most attacks

Attack	Prevention
Packet Sniffing	using a switch to forward messages on specific ports

Attack	Prevention
ARP Spoofing	use static ARP tables (not practical), or authenticating ARP messages (not implemented)
IP Forgery	IPSec provides authentication and encryption of Ip traffic (rare in practice)
DNS Spoofing	DNSSec provides authentication, but not widely deployed, instead use DNS over HTTPs
Route Hijacking/BGP	use Bogon filterning, deny route advertised by hosts with spoofed addresses (used by ISPs)
TCP-syn flooding	use TCP-syn cookie to prevent needing to keep track of stuff, exchange this cookie
DoS/DDoS	network ingress filtering – deny access to network from spoofed addresses, ensure traffic is traceable
ICMP host discovery	limit ICMP or disable for non-same network hosts
Port Scanning	TCP connections can be rejected if they try to connect on too many multiple ports simultaneously

Protection terms

Item	Description
Firewall	logical defense parameter that acts as an access control between two networks (mainly packet filtering inbound traffic)
DMZ	demilitarized zone, exposes public servers like web, mail, databases, etc.
IDS	Intrusion detection systems (looks at headers, contents, fragmentation) and performs deep packet inspection in stealth mode
IPS	intrusion prevention system (IDS + firewall)
TOR	the Onion Router. The more nodes available the more secure it is. One type of anonymous VPN. Compromises in >3 nodes are fine due to limited knowledge between nodes. It prevents IP address identification but does not prevent application identity information (web tracking) to identify you

Human Authentication and Authorization

Identification — Assigning a set of data to a subject

Authentication — Making a safe link between a subject and one or more identities

Human Authorization Factors

- **Something known (password, PIN)** – Good as long as you can remember and not guessable

- **Something owned (IDs, keys, etc)** – Good as long as not damaged or lost and not duplicatable
- **Something you (fingerprint, biometrics, etc.)** – Robustness depends on quality/precision of this measure

Password Storage methods

- **Clear:** Bad
- **Hashed:** Bad because passwords that are the same have the same hash
- **Salted Hash:** Good, and easy to manage
- **Encrypted:** Best, but complex to manage

Authorization

- The system enables the subjects to use the resources
- The subjects are the active entities of the system
- The resources are made available by the system

- **Access-control matrix:** Who has what access All models implement this
- **Role-based lists:** Roles have access, people have roles Easier to manage
- **Complete mediation:** Every access to every object must be mediated
    - Incomplete mediation means that attackers can do something that policy cannot allow
- **Least privilege:** Do not grant subjects more rights than they need
    - Vulnerability that allows attacker to gain privileges that policy does not allow
- There are many access control models depending on application and policy (e.g. constraints, administration, etc.)

Attacks on Authorization

- **Incomplete mediation**
    - A misconfiguration in system allows attacker to do something the abstract policy does not allow
- **Privilege Escalation**
    - A vulnerability in the system allows an attacker to gain privileges that the abstract policy does not allow

Operating Systems & Program (in)security

Overview of an operating system

- Has a kernel, which acts as an API for interacting with the hardware
- We have the user-space which has the software that requires the hardware such as,
    - System calls
    - Applications
    - Services (Daemon)

What is a Daemon?

- These are programs that run in the background such as

    - System services
    - Network services (Servers)
    - Monitoring
    - Scheduled tasks

Security

- There are things called policies which stop certain users from interacting with certain things such as,
    - Alice being unable to access other users or kernel directly
    - Or Alice performing actions to the admin (root)

Hypothesis

- Programs are run by an authenticated user (Authentication)
- Resources are accessed through programs (Authorization)
- Every access is checked by the system (Complete Mediation)
- *Everything is "secured" are long as the system is well configured and programs behave as expected, but...*

Problems

Threats (How can security be compromised?)

- A program can crash or have undesirable behavior

Vulnerabilities

Malicious Program — Program designed to compromise security of the OS. User executes the malware

Vulnerable Program — Not designed to compromise the OS. User executes a legit program that executes the malware

- **Code Execution Vulnerability:** A vulnerability that can be exploited to execute a malicious program
- Malicious programs are software that is distributed to users to install, as vulnerable have malicious files

What happens when a bug occurs?

- Nothing, the program and/or OS are "fault tolerant"
- Program gives wrong result or crashes, but security of system is not compromised
- Resources are locked or OS crashes
- Program computes something that its not suppose to (Malicious code)

Timeline of a vulnerability

- The program is released with vulnerability
- Vulnerability is publicly disclosed (Common Vulnerabilities and Exposures (CVE) alert) (Most dangerous)
- Recommendation is issued
- Patch is released
- Patch is applied (All good)

Attacks

Buffer Overflow Attacks

- Inject wrong data input in a way that it will be interpreted as instructions
- This works because data and instructions are the same - binary values in memory
- Discovered as early as 1972, first severe attack in 1988

Stack execution

```
void func(char* str) {
    char buf[126];
    strcpy(buf, str);
}
```

Local vars	Pointer to previous frame	Return address	Arguments	Previous frame
buf	sfp	red addr	str	frame of calling func

- **SFP** - Stack Frame Pointer
- **EBP** - Base pointer
- **ESP** - Current Stack Pointer
- The top of the stack is on the right

Overstuffing buffer

- strcpy does notcheck for the length of *str. Therefore, if we put more than the buf size (126 characters), it would overwrite the stuff after it
- Ie if we add 4 bytes to skip over sfp, then 4 bytes for the address to the buffer. We can make it execute code from buffer.

TOCTOU Attacks (Time Of Check to Time Of Use)

- Also called race condition attack
- Idea is to swap the file that is about to be run by a program that requires higher privileges to open.
    - This targets concurrent programs with different privileges that use files to share data.
    - This however requires precise timing
- e.g. Attacker can use `symlink("/etc/passwd", "file");` to link important resource to file to be opened instead between authentication step and opening step

What is a secure system?

Some are ...	So ...
More deployed than others	More targetted by hackers
More complex than others	More points of failure

Some are ...	So ...
More open to third party code than others	More "amateur" codes

Security and Design

- **Safety:** For reasonable inputs, get reasonable outputs
- **Security** For unreasonable inputs, get reasonable outputs

What makes a good security metric? (Jonathan Nightingale)

Severity

- If directly exploitable or requires users to "cooperate"

Exposure Window

- How long are users exposed to vulnerability?

Complete Disclosure

- Do vendors disclose vulnerabilities found internally?

Discovering and Exploiting Vulnerabilities

- **Vulnerability Assessment:** Identify and quantify the vulnerabilities of a system
- **Penetration Testing:** Deliberate attack of a system with the intention of finding security weakness

Tools

Reconnaissance	NMAP (Network Mapping and Fingerprinting) - host discovery, OS detection, TCP/UDP scanning
Vulnerability Assessment	OpenVAS - Vulnerability Scanner
Penetration Testing	Metasploit - Exploit Framework

NMAP

- Host discovery, OS detection, Full TCP port scanning, Version detection, Export a full scan to file
- UDP Scan, Stealth Scan (to go through firewalls), Slow Scan (to avoid detection), Scripting engine (to exploit vulnerabilities)

OpenVAS

- Does a scan and gives a report of vulnerabilities

Metasploit

- Allows for the loading and execution of exploits (Basically a uniform automated UI)

Armitage

- Kinda like OpenVAS and Metasploit together, it finds exploits and allows you to use them at same place

Stack Smashing Defences

Canaries

- Compiler modifies every function's prologue and epilogue regions to place and check a value (canary) on the stack
- If overflow, then it gets overwritten. Therefore, it detects theres a problem
- Theres a few types such as random canaries or xor canaries
- Can disable the protection with `-fno-stack-protector`
- Can be bypassed with `Structured Exception Handling (SEH)` exploit that makes exception to point to own code

DEP/NX - Data Execution Prevention / No Execution

- Program marks important structures in memory as non-executable by generating hardware-level exception if executing from those regions
- Which makes normal stack buffer overflows that run shellcode impossible
- Can disable with `-z execstack`
- Can be bypassed with `Return-to-lib-c` exploit which makes a subroutine of lib C thats in the process's executable memory
    - Basically stitch some code out of code from libc

ASLR - Address space layout randomization

- The OS randomizes the location (random offset) where standard libraries and other elements are in memory
- Basically harder to guess addresses
- Can disable with `sysctl kernel.randomize_va_space=0`
- Can be bypassed with `Return-Oriented-Program (ROP)` or brute force (Less practical with 64bit machines)
    - Use instruction pieces of existing programs to weave the exploit

Protection

How to lower risk of security flaw resulting from bug

1. Build better programs
2. Build better operating systems

Better programs

- **Type-safe (Or memory safe)**
  - Pure Lisp, pure Java, ADA
- **Isolate potentially unsafe code**
  - Modula-3, Java with native methods, C#
- **Hopeless**
  - Assembly, C

Type-Safe Programs

- Cannot access arbitrary memory addresses
- Cannot corrupt own memory
- Do not crash

How to make better programs with unsafe languages

- **Defensive** — Good programming practices and being security aware
- **Proactive** — Use system libraries and penetration testing
- **Formal** — using formal methods to verify and generate a program

Defensive Programming Approach

1. Adopt good practices

- **Modularity**
  - Easier to security flaws
- **Encapsulation**
  - Avoid wrong usage
- **Information hiding**
  - Hide implementation (Doesn't improve security)

2. Be security aware

- Check inputs even between components (Mutual suspicion)
- Be "fault tolerant" by having consistent policy to handle failure (managing exceptions)
- Reuse known and widely used code via design patterns and existing libraries

Proactive Approach

1. Use security libraries

- For stack smashing, check if stack has not been altered when function returns
  - If altered, return seg fault
- **Examples**
  - Libsafe
  - Stackguard
  - ProPolice (gcc patches)
  - Microsoft's Data Execution Prevention

2. Perform peneratration testing

- **Test functionalities**
  - Unit test, Integration test, performance test, etc.
- **Test security**
  - Penetration test
  - Basically, trying to make software fail by pushing limits of a "normal" usage (ie. test what program is not suppose to do)

Formal Approach

1. Use formal methods to verify program

- Static analysis (Analyze the code to detect security flaws)
  - Control flow, analyzing sequence of instructions
  - Data flow, analyzing how the data is accessed
  - Data strcture, analyzing how data is organized
- Abstract interpretation
  - Basically we can't test everything, so we just have to make sure that it would vaguely be within range (An estimate)

2. Use formal methods to generate program

- Turn mathematical description of program into executable code or hardwre design
- We know that it works well by doing varies proofs of correctness and refinement
- **Examples**
  - VHDL, Verilog
    - Used by semi-conductor companies such as Intel
  - Critical embedded software (B/Z, Lustre/Esterel)
    - Urban Transportation, Aeronautics, Nuclear plants

Pros and Cons

- It's proven safe and can't possibly get better
- Takes alot of time, effort, and money to make
- Does not prevent specification bugs such as network protocols

Better operating systems

- Testing done in sandboxes, a tightly controlled set of resources for untrusted programs to run in
- Have different types such as servers (Virtual machines), programs (Chroo, sandbox, Metro App Sandboxing), and applets (Java/Flash for web)

Intrusion Detection/Prevention Systems (IDS/IPS)

- Based on signatures (well known programs) and behaviours (unknown programs)
- Example, Syslog and Systrace on Linux
- But vulnerable to malicious programs residing in kernel called "rootkits"

Security Assurance

- Basically a way validate how secure an organization or product/system is

Validating Organization (ISO/IEC 27k)

- **Objective:** Provide the best practice recommendations on information security management, risks and controls
    - Similar to ISO/IEC 9k for quality assurance

How to get certified?

1. Submit an evaluation plan to registrar
2. Registrar runs first audit and grant certification
3. Registrar keeps auditing to guarantee certification

What is inside?

- List of 133 candidate control objectives and controls
- Each control must be addressed one by one in evaluation plan

Governing principles

- Based on iterative program solving process (Deming's Wheel - PDCA)
    - **Plan:** Run risk analysis and define security policy
    - **Do:** Design & build security solutions (Called controls)
    - **Check:** Measure security solutions
    - **Act:** Improve the security assurance

What do the controls cover?

- Risk assessment (How to drive the risk analysis)
- Security policy
- Organization of information security (Governance)
- Asset Management (Inventory & classification of information assets)
- Human Resources protection (Security aspects for employees joining, moving, and leaving org)
- Physical and environmental security (Protection of computer facilities)
- Communications and operations management (Infrastructure supporting activity)
- Access Control (Access rights)
- Information systems acquisition, development, and maintenance (Result of activity)
- Information security incident management (CERT)
- Compliance (Ensuring conformance with security policies)

Validating Product/System (Common Criteria)

- **Objective:** Provide evaluation methodology of,
    - Defining security functionalities
    - Defining assurance requirements

    - Determining whether product meet requirements
    - Determing measure of evaluation results in Evaluation Assurance Level (EAL)
- Technical evaluation based on security assurance methods
    - Testing and penetration testing
    - Formal development and/or verification

TCSEC = "The Orange Book" (1983-1999)

- Used to evaluate and classify computer systems regarding storage, retrieving and processing of sensative data
    - By US department of defence in the 70s

Governing Principles

- **Introduce concept of policy**
    - Must be explicit and enforceable by computer system
    - Two kinds - DAC and MAC
- **Introduce concept of accountability**
    - Users must be identified and authenticated
    - Each access must be logged

Security Assurance Classes (1991-2001)

- **Class D:** Minimal protection
    - No security requirements
- **Class C:** Discretionary Security Protection
    - Multi-user environment and data with different sensitivity levels
- **Class B:** Mandatory Security Protection
    - Object labels, user clearance levels, and multilevel security policy
- **Class A:** Verified Protection
    - Formal design and verification

Common Criteria (Since 1998)

- **Protection Profile:** Functionalities and security requirements of product/system
    - Written by system consumer
- **Security Target:** Identifies security properties
    - Written by software designer in response to the protection profile

Evaluation Assurance Levels (EAL)

- **EAL 1** Functionally Tested
    - Requires documention of security function vouncing for minimum confidence regarding correctness, but threats are not as serious
- **EAL 2** Structurally Tested
    - Requires delivery of test procedures and results
- **EAL 3** Methodically Tested, and Checked
    - Requires developers to be aware of good software engineering practices

- **EAL 4** Methodically Designed, Tested, and Reviewed
  - Requires good commercial development methods to ensure good software engineering practices
- **EAL 5** Semi-formally Designed, and Tested
  - Requires rigorous commercial development practices supported by a security expert
- **EAL 6** Semi-formally Verified Design, and Tested
  - Requires rigorous development environment
- **EAL 7** Formally Verified Design, and Tested
  - Requires rigorous security-oriented development environment

Issues

1. Preparing documentation for evaluation takes alot of effort
   - Product is obsolete once certified
2. Processes such as evaluation is costly
   - Return on investment is not necessarily a more secure product
3. Evaluation is performed on documentation and not product itself
   - A good EAL does not prevent security flaws

Malware

Action

- Performs unasked for operations on the system
- **Rabbit:** Exhausts hardware resources of system until failure
- **Backdoor:** Allows attacker to take control of system bypassing authorization mechanisms (Also control type)
- **Spyware:** Collects information
- **Spamware:** Uses system to send spam
- **Ransomware:** Restricts access to data and resources, and demands ransom
- **Adware:** Renders unasked for advertisement

Dissimulation

- Avoid detection by anti-malware programs
- **Rootkit:** Hides the existence of malicious activities

Infection

- Penetrate a system and spread to others
- **Replication:** Copy itself to spread
  - **Virus:** Contaminates existing executable programs
  - **Worm:** Exploits service's vulnerability
- **Subterfuge:** Based on user's credulity
  - **Trojan Horse:** Tricks user to execute malicious code

Control

- Activate malicious code

- **Backdoor:** Communicates with *command & control servers* allowing attacker to control virus
- **Logic Bomb:** Activiates malicious code when certain conditions are met

History of malicious code

- **70s:** Era of first self-replicating programs
- **80s:** Era of maturity and first pandemics
- **90s:** Era of self-modifying virus and macro viruses
- **00s:** Era of Trojan horses and internet worms
- **10s:** Era pf cyber-warefare viruses

70s

- **ANIMAL** (Simple Joke)
  - Replication through the file system with no effect
- **Creeper/Reaper** (Disruptive)
  - Replication through modem and copied itself to remote system
  - Displays `I'M THE CREEPER : CATCH ME IF YOU CAN`
  - Reaper was made to hunt Creeper
- **Rabbit** (Destructive)
  - Replication through filesystem, which reduces system performance til crashing
- These are classified as viruses. There are two types of viruses
  - **Resident:** Remains in memory after infected programs terminates
  - **Non-resident:** Becomes inactive as soon as infect program terminates

80s

Apparition of boot sector viruses

- **Elk Cloner**
  - Displays short poem on every 50th boot on infected computer
- **Brain**
  - Disk label changed to "Brain" and advertisement text is written in boot sectors
  - Moves bootstrap loader elsewhere, puts virus code into boot sector. Therefore it runs before boot

Pandemics

- **Jerusalem** (MS-DOS)
  - Destroys all executable files on infected machines upon every occurence of Friday 13th
- **SCA** (Amiga)
  - Desplays a text every 15th boot
  - 40% of amiga owners were infected
- **Christmas Tree EXEC** (IBM/PC)
  - Displays a snow flow animation
  - Paralyzed several international computer networks in December 1987

Anti-virus softwares

- **Virus Scanner** (Detection)
    - **Signature based:** Using signature database of existing viruses
    - **Behavior based:** Looking for suspicious code patterns that can be used by viruses
- **Virus Removal Tools** (Sanitation)
    - Cleaning memory and filesystem

Avoiding Detection

- **Cascade**
    - Virus encrypts itself with cryptographic key and changes key when replicating
    - Each instance looks different
    - Emergence of polymorphic viruses

90s

The Chaeleon Family (Polymorphic Virus)

- **Ply**
    - DOS 16-bit based complicated polymorphic virus with built-in permutation engine

Anatomy of polymorphic virus

- Mutates when replicating, but keeps original algorithm. Does this by,
    - Using cryptography
    - Injecting garbage code
    - Doing permutations within certain instructions/blocks of instructions
    - Using code obfuscation techniques
- Can only be detected by detecting code patterns used for self-modification

Metamorphic Virus

- Virus that can reprogram itself by,
    - Using different instructions
    - Having different strategies to implement a functionality
- **Zmist:** First metamorphic virus
- **Simile:** First mutli-OS metamorphic virus

Macro Viruses

- Virus that is written in scripting languages used by some office applications (can be cross platform)
    - ie. Written in VBS, embedded in MS-office document which activities when document is open
- **Concept:** First Word macro virus that was also the most common. It did nothing tho.
- **Meliisa:** Shutdown email systems that got clogged with infect emails

00s

Trojan Horses

- Program disguised as legitimate program/file. Most cases replicated through emails

- **VBS/Loverletter ILOVEYOU:** Caused 5.5 to 10 billion dollars in damage
- **Sobig:** Sobig.F set a record in sheer volume of e-mails
- **MyDoom:** Broke record set by Sobig.F

Worms

- Explots a security flaw to infect machine and replicate itself through the network
    - Very fast (doesn't need user to be activated)
    - Has payload
- Has a few factors
    - Wide adoption of internet
    - Global network is good medium for virus pandemics
    - Multiplication of internet applications and services
    - Fast publication of program vulnerabilities
    - Slow release/adoption of corrective patches

Examples

- **Code-Red**
    - Exploits security flaw (buffer overflow) of Microsoft IIS web server patched one month later
- **Nimda**
    - Explots another MS-IIS security flaw and is most widespread worm so far
- **Klez**
    - Exploits security flaw of IE layout engine used by Outlook and IE
    - Infection through email attachment and user doesn't need to open the attachment to get infected
- **SQL-Slammer (Also called Sapphire)**
    - Exploits flaw in MS-SQL serers that got a patch six months later
    - Caused DOS and dramatically slowed global internet traffic
- **Sasser**
    - Exploits buffer overflow of Microsoft LSASS on Windows 2000 and XP
- **Blaster (Also called Lovesan)**
    - Exploits flaw in DCOM-RPC services on Windows 2000 and XP
    - SYN flood against port 80 of windowsupdated.com
- **Welchia (Also called Nachia)**
    - Exploits same flaw as Blaster
    - Used to correct security flaw by patching system (Counters Blaster)
- **Conficker**
    - Explits flaw in NetBIOS, disables auto-update and adds dictionary password cracker and backdoor to turn machine into bot
    - Believed to be orignated from Ukraine and/or Russia

Web Worms

- **Santy**
    - Exploits vulnerability in phpBB and uses Google to find new targets
    - Infected 40k sites before Google filtered search query used by worm

XSS worms

- Exploits a cross site scripting within website
- **Samy** - Target MySpace
- **JTV.worm** - Target Justiin.tv
- **Twitter.worm** - Target Twitter

10s

Cyber-warfare Virus

- **W32.Dozor**
  - Virus that created a botnet dedicated to perform DDoS attack on South Korea and US government website
  - Believed to originate from China and/or North Korea
- **Stuxnet**
  - Sophisticated virus that targets SCADA systems (Supervisory control and data acquisition)
  - Believed to have taken down 4000 nuclear centrifuges in Iran
  - Beleived to originate from USA and Israel
- **Flame (Also called Skywiper)**
  - Espionage virus that embeds sophisticated spywares believed to be from US (Olympic Games defence program)

Ransomware

- **Reveton**
  - Displays message from law enforcement agency saying you have pirated software and child porn
  - Ask to pay fine using prepaid cash service
- **CryptoLocker**
  - Encrypts specific files on machine with 2048 RSA key
  - Ask to pay ransom with Bit coins
- **WannaCry and Petya**
  - Use vulnerability found in NSA hacking toolkit leak
  - Researcher found "kill switch"
  - Paralyzed hospital in UK and trains in Germany

IoT malware and Cryptominers

- **Mirai**
  - Infects IoT devices, and most powerful DDoS attacks to date
- **Coinhive**
  - JS in website and popular malware as well

Hoax Viruses (Really Dumb)

- Gives you the method to detect and remove virus and ask you to transfer this email to your contacts
- Almost harmless and do nothing by themselves (But users do)

Modern Malicious Code

- Exploded around 2000s (144% between 2012 and 2013)
- Why?
    - There's money for malicious software
    - Easy to hire hacker or get cutting-edge hacking tools online
    - **In conclusion, making a new malware is as simple as assembling pieces available online**

How to create new malware

1. Create malware's payload (a.k.a building a RAT)
2. Make malware undetectable (a.k.a packing a malware)
3. Spread the malware

What malware do

- Take control of victim's device turning it into a zombie/bot
- Act as spam relay or DDoS relay
- Steal personal information like passwords, bank info
- Clickbot for traffic
- ...

1. Remote Administration Tool (RAT)

- Basically remote admin tool with
    - Stealth features
    - Specific functionalities such as camera controller, hardware destroyer, password loggers, etc.

DIY RATs

- **Pro:** Free and Personalized
- **Con:** Time consuming and requires good expertise of targetted system

Commerical Off-The-Shelf RATs

- **Zeus:** Initially $700, but open source
- **DarkComet:** Open source
- **BlackShades:** Can be purchased from official company
- Basically has menu with options, and even has options for you to troll people

2. Make malware undetectable

Detection methods

- **Static Analysis**
    - Scan program comparing it to a collection of signatures
    - Bypassed with encryption and code obfuscation
- **Dynamic Analysis**

    - Run program in sandbox and infer from its behavior
    - Bypassed by detecting environment and employ trigger based behaviors

DIY packing

- **Pro:** Free and Personalized
- **Con:** Requires good expertise of cryptography, code obfuscation, and execution environment

Commerical Off-The-Shelf Crypter

- **Byte Crypter**
- **Datascrambler**
- **BlackShades Crypter**
- Functionalities include
    - Start malware on startup
    - Block sandbox from monitoring
    - Kill other bots
    - Protect from botkiller
    - Delay for dynamic analysis
    - Persistence and binder

3. Spread the malware

Via Social Engineering

- Trick people to download and install malware. Some ways,
    - Tutorial on hacking that makes you install malware
    - Video/chat player to exclusive content or people
    - Pirated software on P2P
- **Pro:** Free
- **Con:** Dfficult to get cautious people infect and limited impact

Via webpage

- Exploit browser/plugin vulnerability to automatically download and install malware on victim device
- **Pro:** Everyone with vulnerable browser can be infected, can be used for massive infections and targetted ones
- **Con:** Requires good expertise of target browser, it's vulnerabilities, and how to exploit them

Buy Exploit Bundle/Kit and services

- **Blackhole:** 19 CVEs mainly targetting Java and Adobe products
- **Redkit:** 4 CVEs mainly targeting Java

Types of services

- **Exploit Bundle:** Program to embed into website
- **Bulletproof host:** Hosting service to bypass any kind of IP filtering, anti-spam, anti-virus, anti-malware, law enforcement, etc.

- **Traffic:** Attract peopel to visit the infected webpage

Buying installs of malware

- **Pro:** Easy and can be selected about geolocation of the host
- **Cons:** Pricy

Web Security

Architecture

- Separated into client (Web browser) and server side (Web sever & database)
- Uses the HTTP, a network protocol for requesting/receiving data on the web
    - Standard TCP protocol on port 80
    - Uses different URI/URL to specify resources and different methods for actions

Anatomy of URL

Protocol	Server	Path	Query String	Resource	GET Params
http://	whitehat.local	/	index.php	?filter=	hello

User Authentication Process

1. Ask user for login and password (Sent to server over HTTP/POST)
2. Verify login/password (based on information on server usually in db)
3. Start a session (once authenticated)
4. Grant access to resources (according to session)

What is a session?

- A session is created via a session id (token) between browser and web app
- This should be unique and unforgeable long random number or hash stored in cookie
- The id is bind to key/value pairs data on server
- The id can be created/modified/deleted by user in cookie
    - But cant access key/value pairs in server

Transport Layer Issues

- To steal user credentials, you steal user password or session ID
- **Threats**
    1. Attacker can eavesdrop messages
        - Confidentiality
    2. Attack can tamper with messages
        - Integrity
- To address those issues, we have HTTPS (HTTP + TLS)
    - Provides end-to-end secure channel (Confidentiality) and authentication handshake (Integrity)

    - This however fails if there is mixed content from elements served with HTTP on HTTPS page, or control transfer to another HTTP page of same domain
        - This results in authentiction cookie being sent over HTTP for the pickings
    - There's also limitations as it only protects the channel, not the client/server

How to protect cookie

- **Secure** Flag
    - Makes it so that cookie will be sent over HTTPS only
    - Prevents leaking in case of mixed-content
- **HttpOnly** Flag
    - Makes cookie not reabable/writeable from frontend
    - Prevents cookie from being leaked when XSS attack occurs

Ways to steal password

From Client

- Social engineering (Phishing)
- Keyloggers (Keystroke logging)
- Data mining (Emails, logs)
- **Hack the client's code**

From Server

- Hack the server
- **Hack the server's side code**

Vulnerabilities

Front-end

Content Spoofing

- Basically inject HTML into website via data put into the database
- Can be resolved by validating data inserted in the DOM

Cross-Site Scripting (XSS)

- Inject JS code into website via data put into the database
- Can do stuff like
    - Add illegitimate content (Same as content spoofing)
    - Add illegitimate HTTP requests through Ajax (Same as CSRF)
    - Steal Session ID from cookie
    - Steal user login/password by modifiy page to forge scam
- You can also make worms which spread

Types of XSS Attacks

- **Reflected XSS**
  - Malicious data sent to backend is meediately sent back to frontend to be inserted into DOM
- **Stored XSS**
  - Malicious data is stored in backend, and later sent back to be inserted
- **DOM-based Attack**
  - Malicious data is manipulated in JS and inserted
  - Can be resolved by validating data before inserting into DOM

Cross-site Request Forgery

- Basically make a request to target site from malicious site.
  - This makes cookies associated to that target to be attached and used
- Solution is to use the **Same origin policy**
  - Resources must come from same domain (protocol, host, port)
  - This covers Ajax reuqests and form actions
  - But not JS scripts, CSS, images, video, sound, plugins
  - Can be relaxed iframes, cross-origin resource sharing (CORS), or JSONP

Problem

- An attacker can execute unwanted but authenticated actions on web app by,
  - Setting up malicious website with cross-origin requests
  - Injecting malicious urls into page
- Solution is to add a CSRF token
  - A unique, secret, unpredicatable value generated by server for next HTTP request
  - Basically its a nonce
- Another solution is to use SameSite

Back-end

Incomplete Mediation

- Basically server doesn't check requests, hence you can make your own to do stuff
- Hence, don't trust frontend data and sensitive operations must be done on backend

Information Leakage

- From database dumps or just hacking into system

SQL Injection

- Inject SQL/NoSQL code to get/add/modify/delete information, or bypass authentication
- **SQL Example**

```
db.run("SELECT * FROM users WHERE USERNAME='${ username }' AND PASSWORD='${
password }'");
```

```
```

  - We can put username as `alice`, and password as `blah' OR '1' = '1'`
  - This would result in password always true. Hence access as alice!

- **NoSQL Example**

```
db.find({ username, password });
```

  - We can put username as `alice` and password as `{ gt: "" }`
  - Same effect as SQL example

Web Penetration Testing Tools

- Proxy mapper
- Vulnerability scanner
- Replay HTTP requests
- (Exploit tool)

Social Engineering and Information Diving

Social Engineering

- The act of manipulating people into performing actions or divulging confidential information, than than by breaking in or using technical cracking techniques
  - Basically to get someone to "willingly" give information
- Kevin Mitnick, most wanted hacker in history who did alot of phishing

Information diving

- The practice of recovering technical data from discarded material

Phishing

- Criminally fraudulent process of attempting to acquire senitive information by masquerading as a trustworthy entity in an electronic communication
- Can be bought as services

Spear Phishing

- Combines Social Engineering with Phishing

Security Questions

- Kinda bad as some people can actually answer them. Also you can combine some to get full informations

Google Hacking

- Using Google search to find security holes in configurations and computer code that websites use