Topics:

1. Malware.
2. Intrusion detection.
3. Assurance and evaluation.
4. Denial of service.
5. Firewalls.
6. Passwords biometrics and identity.
7. OWASP and software security.
8. Buffer overflow.
9. OS security.
10. Access control.
11. Database security.
12. Cryptography.
13. Logging and auditing.
14. Information flow.

Malwares:

1. **Trojan**: Spreads by users transfers, may infect file, needs user to spread.
2. **Worm**: Spreads by network, doesn't infect file, doesn't need user to spread.
3. **Virus**: Spreads when a file is executed, infects file, needs user to spread.
    1) **Polymorphic**: Changes encryption, add dummies.
    2) **Metamorphic**: Changes code, same functionality.
    3) **Stealth**.
    4) **Boot sector infector**.
    5) **Executable infector**: Infects executables.
    6) **Terminate and stay resident**.
    7) **Encrypted**.
    8) **Scripting**.
    9) **Rabbit**.
    10) **Logic bomb**.
    11) **Stuxnet**.
4. **Key loggers**.
5. **Phishing**.
6. **Drive-by downloads**: Browser's vulnerability.
7. **Bot:** Has remote-control facility.
8. **Rootkit**.
9. **Ransomware**.
10. **Phone worm**.
11. **Phone trojans**.

Defenses:

1. **Update, AC**.
2. **Antivirus software**.
    1) **Host-based scanners**: installed in the computer itself.
        - **Simple scanners**: Searches for known string. Weak against polymorphic virus.

        - **Heuristic scanners**: Searches for fragments of code that are associated with viruses.
        - **Activity scanners**: Searches for actions associated to viruses. Like system calls.
        - **Full-featured protection**: Combination.
    2) **Network-based scanners**: Installed in a network (perimeter). Looks for ingress and egress flow content.
    3) **Distributed intelligence gathering approaches**: Combination. Malware analysis machine, administrative machine, leaf machines.
3. **Rootkit detection**.
4. **Checksummers**: Knows the authorised executables in the system.
5. **Type 'data'**: Initially treats your new files as data, if certified, then becomes executable.
6. **Flow distance**.
7. **Proof-carrying code**. Drawback, conclusions are compared to its own security policy.
8. **Isolation and evaluation**.
9. **Education and training**.

Network defenses:

1. **Updates, AC**.
2. **Filtering**: Firewalls, block malwares to enter your network.
3. **Intrusion detection**: Network monitoring program.
4. **Encryption:** TLS and SSH protocols.

Denial of Service:

1. **Denial of Network Service:** Packet quantity. Eg. Ping commands.
2. **Denial of System Service:** Packet type to consume resources or cause bug. Eg. TCP SYN flood. Defense:
    - **Vary timeout period.**
    - **Make client keep state.**
3. **Denial of Application Service:** Requests that are computationally expensive. Eg. HTTP and SIP flood.
4. **Distributed Denial of Service:** Uses botnet.

Attacks:

1. **Reflector**: Attacker spoofs victim's IP. Then send requests.
2. **Amplifier:** Attacker spoofs victim's IP. Then send requests that generate multiple responses.

Defenses:

1. Update.
2. Limit/remove ability to send packets to outside IPs.
3. Limit rate at which packets can be sent.
4. Make user store table.
5. Block broadcasts.
6. Use CAPTCHAs or graphical passwords.

7. Replicate your servers.

Responses:

1. Plan.
2. Automated intrusion detection system.
3. Analyze packets.
4. Change to replicated servers.
5. Update plan.

Intrusion detection:

1. **Software trespass**.
2. **User trespass**:
   1) **Masquerader**: Unauthorized outsider using legitimate user's account.
   2) **Insider**: Legitimate user that access unauthorized data.
   3) **Clandestine**: Either. Focus on stealth.
   4) **Defense**:
      i. **Least privilege**.
      ii. **Log accesses and commands**.
      iii. **Use two factor authentication**: For greater accountability.
      iv. **Safely fire someone**: Turn access off, do it in remote location, make copy of HD.

Intrusion detection system (IDS):

A combination is used nowadays.

Requirements:

1. Fully automated.
2. Restart from crashes.
3. Able to monitor itself.
4. Impose minimal overhead.
5. Configurable.
6. Dynamically reconfigurable.
7. Adaptable.
8. Scalable.
9. Degrade gracefully.
10. Detects many things.
11. Good GUI.
12. Accurate.

Models:

1. **Anomaly detection model**:
   1) Doesn't know about the attacker's behavior.
   2) Looks for anomalous *user* behavior.
   3) Model of a good system.
   4) Has to run to get statistics.
   5) Uses AI.
   6) Eg. Suspicious to use commands far off current statistics

2. **Misuse detection model:**
    1) Knows about the attacker's behavior.
    2) Looks for *attacker*'s behavior.
    3) *Not* a model of a good system.
    4) Has to create rules about the attacker's behavior.
    5) *Doesn't* use AI.
    6) Eg. Antivirus scanner.
3. **Specification model:**
    1) Not associated with user, but with software.
    2) Exact specification of the software is known.
    3) Suspicious if software does anything different that it's supposed to do.
    4) Doesn't use IA.

IDS Architecture:

1. **Agents:** Collect and send statistics to the director. Can be placed in/out the firewall and/or in a subnet.
2. **Director:** Commands agents. Makes decision. Due to performance and being proprietary, runs isolated.
3. **Notifier:** Distribute director's message to employees.

Responses:

1. **Plan.**
2. **Identification of the attack.**
3. **Containment:** Use a honeypot to better understand the attack.
4. **Eradication:** Delete attacker's code.
5. **Recovery**: Return your network to a secure state.
6. **Update plan**.

Firewalls:

- **Additional layer of defense. Separates your organization from the internet.**
- **Goals**:
    1. Block malicious traffic.
    2. Allows legitimate traffic.
- **Provides:**
    1. **Service control**: Controls the types of internet services that can be accessed.
    2. **Direction control**: Controls the direction that specific services can flow.
    3. **User control**: Controls who can access the specific services.
    4. **Behavior control**: Controls how specific services are used.
- **Capabilities:**
    1. Single point administration.
    2. Monitors security related functions.
    3. Platform for other (non-security related) functions.
    4. Platform for IPsec.
        - **Transport mode**: Encrypts body.
        - **Tunnel mode:** Creates another header encrypts the old body and header. Replaces sender by your firewall, and receiver by other's company firewall.

- **Limitations:**
  1. Can't protect against attacks that don't use firewalls.
  2. Can't protect against naïve users that bring viruses from their computers from StarBucks to the company.
- **Types of filters:**
  1. **Positive filter:** Only allows packets with specific types.
  2. **Negative filter:** Only blocks packets with specific types.
  3. **Examine only header.**
  4. **Examine header and payload.**
  5. **Examine patterns of packets.**
- **Types of firewalls:**
  1. **Packet filtering firewall:** Applies a set of rules for every packet that comes in.
  2. **Stateful inspection firewall:** Same as above, plus monitoring of TCP connections.
  3. **Application-level gateway**: Makes every connection pass through the firewall. Looks at the content.
  4. **Circuit-level gateway:** Same as above, but in TCP layer. Looks at the connection.
- **Firewall installation:**
  1. **Personal firewall:** Located in the user machine. Simpler.
  2. **Host-based**: More complex.
- **Firewall location:**
  1. Internet ->
  2. Outer firewall (DNS server, mail server, web server, log server) ->
  3. Demilitarized zone ->
  4. Inner firewall (Development subnet, customer data subnet, internal servers).
- **Intrusion Protection System (IPS):** Combination of Firewall and IDS.
- **Unified threat management:** Unified place to manage and set up the management of threats. However, damages performance.

OS Security:

Top 4 methods to prevent OS attacks:

1. Auto-update OS and applications.
2. Update third-party applications.
3. Least privilege.
4. White-list approved applications.

OS initial setup:

1. Install it in a protected environment.
2. The source of any additional device must be carefully validated.
3. Install only the minimum (hardening).
4. Avoid default configuration.
5. Regularly update the system.
6. Every patch must be validated before deploying the system to production.

Types of virtualization:

1. Native: Physical Hardware -> Hypervisor -> Guest O/S 1 Kernel -> User Apps.

2. Hosted: Physical Hardware -> Host OS Kernel -> (Other User Apps) or (Hypervisor) -> Guest O/S 1 Kernel -> User Apps.

Other types:

1. Application virtualization: Allows applications written for one environment to execute on some other OS.
2. Full virtualization: Multiple full OS instances execute in parallel.
3. Virtual machine monitor (VMM): Hypervisor, acts like an interface between each of the guest OSes and the actual physical hardware resources.

Access Control:

Access control list (ACL): Stores based on the resources. Resource 1: { (read: [user 1, user 2]), (write: [user 2, user 3]) }.

1. Not suited to change status of a user.
2. Not suited for large populations (specially when constantly changing).
3. Not suited for delegation.
4. Suited to change status of an object.
5. The originator can change it.

Capability list (CL): The opposite of ACL.

Authorization table:

- Can be sorted to act as an ACL or CL.
- It has to be stored centrally (may become a bottleneck).
- Less performance than locally stored in each computer.

Implementation considerations: It's done after you choose which AC to use.

- **ACL implementation considerations:**
    1. Which subjects can modify the ACL?
    2. The ACL applies to privileged users?
    3. Does it support groups?
    4. Does it support wild cards?
    5. How to deal with delegations?
- **CL implementation considerations:**
    1. How to protect it from the user?
        - Hardware solution (architecture).
        - Software solution (protected pages in memory).
        - Cryptographic solution (CL is signed by the OS's private key).
    2. Can use indirection to a name in order to turn off user's permission.

Cryptography:

Asymmetric keys: Guaranties confidentiality, authenticity and integrity. Key distribution disadvantage.

Symmetric keys: Guaranties only confidentiality. Speed advantage.

Digital signatures: Guaranties authenticity and integrity. Uses symmetric keys.

MAC: Guaranties integrity and authentication. Uses a private key.

Hash:

1. Guaranties only integrity.
2. Arbitrary input size, fixed output size.
3. One-way function.
4. Collision resistance.

Cryptographic algorithms:

- Don't "roll your own".
- Use standardized open algorithms, list NIST.
- Crypto agility: Quickly change your crypto algorithm in case it has been broken.

KDC (Key distribution center) data vs CA (certification authority) data:

1. KDC *holds* a symmetric key for every user in the system.
2. CA *holds* only its own private key.
3. KDC *creates* a symmetric key for every pair of people that want to communicate.
4. CA *creates* a certificate for every user.

KDC implications if attacker breaks in:

1. Can cease the data that is held and generated.
2. Can read all traffic (previous present and future).
3. Can impersonate every user.

CA implications if attacker breaks in:

1. Can generate trusted certificates.
2. Can't read previous data.
3. Can generate a new pair for Bob and impersonate Bob.
4. Bob would easily detect it.

Trust anchor:

1. Is the CA's public key.
2. Allows users to get to any other public key in the system.
3. Can get it from physical stores or from the browser (risky).

KDC: Part of a cryptosystem to reduce the risks of exchanging keys.

- Used in symmetric key environment.
- Shares a key with each of all the other parties.
- Produces a ticked based on a server key.
- Kerberos includes KDCs, there KDC is partioned in:
  - Authentication Server.
  - Ticket Granting Service.

CA: Trusted entity that manages and issues certificates and public keys.

- Used in asymmetric key environment.

- Is part of PKI.

PKI: Manages and issues CAs and manages public-key encryption.

- Facilitates secure electronic transfer of information, such as e-mail.
- Required when password is an inadequate authentication method.
- The PKI role that assured valid and correct registration is called Registration Authority.

Logging and Auditing:

It's used after the breach.

Logging: Logs the sequence of events leading to your system being in an insecure state.

1. We need to collect info that will facilitate analysis.
2. We need to understand what is needed to violate our security policy.
3. Logging helps to understand user behaviour and detect anomalies.
4. Logging helps auditing.
5. Logging scare attackers.

How to log:

1. We may log only part of the information.
2. We may include the context afterwards.
3. We may use a formal grammar-based approach, to make it human-readable.

Auditing: Analysis of the logging that gives you information about how to proceed.

- Must have good GUI.
- Must be configurable.

Architecture of an auditing system: Logger -> Analyzer -> Notifier.

System Assurance and Evaluation:

System Assurance: Convincing yourself that you did the right thing.

System assurance may be based on:

1. Process used to build the system, like Garry Mcgraw's three pillars.
2. Quality of the development team.
3. Experience of the development team.
4. Usage of formal methods.
5. Testing the system by deliberately introducing bugs.
6. Simplicity of the system, "complexity is the enemy of security".

System assurance may assess:

1. Functionality.
2. Strength of the mechanisms, like the cryptographic algorithm.
3. Implementation, like the possibility of having a buffer overflow problem.
4. Usability (varies from person to person), does the system cope with unexpected inputs? how easily can the developers test it.

How to prove to yourself that the system is secure:

1. **Security technique**s: Like white and black box testing.
    I. Look for flaws.
    II. Show that every flaw objective was met by at least one protection mechanism.
    III. Use standardized testing.
2. **Formal techniques**: BAN logic, proposed to analyzing and reasoning about beliefs.
    I. "A |≡ B" means "A believes B".
    II. May help to root out (find and remove) assumptions.
    III. You can build a logical path that demonstrates that Bob trusts a CA.
3. **Analyze bug discovery:**
    I. Deliberately introduce bugs (fault injection) to test your development team.
    II. Looks for bug founds and estimate how many bugs there are in the system. Like 10 bugs in 1,000 lines, but software has 100,000 lines.
    III. Make developers responsible for fixing their own bugs, no matter when.
    IV. Hire and train your developers.
    V. Write documentation about your project and make people read it.
    VI. Do prolonged testing (that's the difference between security and normal testing).
        - Alice (angel) can test more bugs, but Bob (bad guy) needs only one bug to exploit the system.
        - The type of bug must be considered, if you find that buffer overflow is a common bug, changing the compiler may significantly reduce the amount of bugs.

System Evaluation: Convincing others that you did the right thing.

- Assembly evidence.
- **Problems:**
    1) Evaluation being ignored by the others. Like buying a product poorly evaluated.
    2) Evaluation may be too narrow and not include, e.g. laws and usability.
- **Formal evaluation:**
    1) **Done by relying party (the party that will use your system).**
        i. Trusted Computer Systems Evaluation Criteria, US, evaluation for OS.
        ii. Information Technology Security Evaluation Criteria, Europe, evaluation for IT product.
        iii. Canadian Trusted Computer Product Evaluation Criteria, Canada.
        iv. Process:
            1. Government requests the product to be evaluated.
            2. People allocate personal to test it.
            3. Evaluation takes 2-3 years.
            4. Problems:
                a. Took to long.
                b. Geographically specific.
    2) **Done by third party.**
        i. Common Criteria was created to overcome the previous problems.
        ii. Not geographically specific anymore.
        iii. Expensive.
        iv. Takes long time.
        v. Vendor pays.
        vi. Seven levels.

vii. Need to be done to sell for the government. Useful to other non-governmental companies as well.
viii. The documents that will be tested can be checked and you can test it yourself.

3) Done by relying party and third party.

i. Called like that because it used to be required by the government. But now is also required by normal companies.
ii. Looks at the cryptographic model.

- **Informal evaluation:**
  1) Open-source.
     i. Advantage: People can look at the code.
     ii. Disadvantage: If too complex, people stop looking at it.
  2) Instead of publicly posting your code's bug, people can directly communicate the company and get recognized by doing so.
  3) Bug bounty means that the company is paying people to find bugs.

Passwords Biometrics and Identity

Biometrics:

- Alternative way to identify a user to a system.
- It's easier than a password for a legitimate user.
- It's harder than a password for a bad guy.

Biometrics examples:

1. Fingerprints.
2. Voice recognition.
3. Iris scan.
4. Retinal scan.
5. Face recognition.
6. Hand geometry.
7. Key strokes (the way you type).
8. Gait (the way you walk).
9. Earshape.

Identity: The computer representation of an entity.

Authentication: The way that the computer knows that you belong to an identity.

What do we do when a user must be known to several systems?

1. We need global unique identifiers and identity mapping.
2. We create a hierarchy of names.
3. The name of an entity is defined by the full path of names starting from the root.

Database Security

Database access control:

- Determines:
  - What a user can access.
  - What access right the user has (create, insert, update, read, write).

- Supports:
    - Centralized administration: a few people can grant or revoke rights.
    - Ownership-based administration: the creator of the table can grant or revoke rights
    - Decentralized administration: same as ownership-based but the granted users can grant and revoke rights.

Role-Based Access Control (RBAC): Eases administrative burden and improves security.

1. **Needs to provide the following capabilities:**
    - Create and delete roles.
    - Define permissions for a role.
    - Assign and cancel assignment of users to roles.

Inference: Authorized queries that allows inference about unauthorized information.

Countermeasures:

1. Change the access control.
2. Alter the database structure.
3. Monitor and alter queries.
4. Reject queries.
5. Add noise to statistics generated from original data.

Database encryption: Can be applied at the entire database at the record level, the attribute level, or level of the individual field.

Cloud security:

- **Multi-instance:**
    - Provides a unique DBMS running on a VM instance for each cloud subscriber.
    - Gives the subscriber complete control over administrative tasks related to security.
- **Multi-tenant model:**
    - Provides a predefined environment for the cloud subscriber that is shared with other tenants.
    - Gives the appearance of exclusive use of the instance but relies on the cloud provider to stablish and maintain a secure database environment.

OWASP and software security

Software security: Writing better code.

- Attacker trying to manipulate your system.
- **Handling inputs:**
    1. **Input sources:** Network connection, mouse, keyboard, sensors, cameras…
    2. **Input content:** Interpretation of program input.
        - Injection attack: Unexpected input intended to cause damage.
        - Command injection: Against OS.
        - SQL injection Against databases.
        - Code injection: Executes on its own.
        - Cross-site scripting: Enables attackers to inject client-side scripts into web pages viewed by other users.

3. **Input testing:**
    - Input fuzzing: Trying random inputs to check if you program fails/fails gracefully.
- **Writing safer code:**
    1. **Ensure correct algorithm implementation**: Proper random number generator, removal of the debug code…
    2. **Ensure machine language corresponds to source code:** Used in highly sensitive classified security environments.
    3. **Ensure correct interpretation and use of data values.** In C is easy to change an integer by a pointer.
    4. **Ensure correct use of dynamic memory:** Garbage collector, otherwise easy target for DoS.
    5. **Ensure that race conditions are preserved.**
    6. **Ensure that interactions are protected:** Like between your program and OS.

McGraw's pillars:

1) **Applied Risk Management:**
    - **Risk Management Framework:** A decision support tool for the business leaders. It identifies, ranks and prioritizes the risks in your software based on business goals and requirements.
    - **Steps:**
        1) Understand the business context. As market share and ROI.
        2) Identify the business and technical risks.
            - Business risks: Damage to brand, financial loss…
            - Technical risks: Crash of a service, privacy leak.
        3) Prioritize risks: What should we do first, where to allocate resource?
        4) Define risk mitigation strategy: Likelihood of success and completeness.
        5) Carry out fixed and validate: Did the mitigation solved the problem? Needs to be measurable. It needs to be convincing.
2) **Software Security "Touch Points":**
    - **Focus on software artifacts.**
    - **Rank the most effective touch points:**
        1) Code review: "code" artifact.
        2) Architecture risk analysis: "requirements and use cases", "architecture and design", "test results" artifacts.
        3) Penetration testing: "test results", "feedback from the field".
        4) Security testing: "test plans documents" artifact.
        5) Abuse cases: "requirement and use cases" artifact.
        6) Examining security requirements: "requirements and use cases" artifact.
        7) Security operations: "feedback from the field" artifact.
3) **Knowledge:**
    - Company principles.
    - Rules of the company.
    - Guidelines of coding.
    - Vulnerabilities, exploits, attack patterns and historical risks.
    - Coding errors: Tools to help with errors.

OWASP Top 10:

1) **Injection:** Receiving user input that goes into the back-end databased command.
2) **Broken Authentication:** Sessions should be unique to each individual.
3) **Sensitive Data Exposure:** Applications should ensure that access be authenticated and data be encrypted.
4) **XML External Entities (XXE):** Occurs when XML input containing a reference to an external entity is processed by a weakly configured XML.
5) **Broken Access Control:** Ensure that the application is performing solid authorization and doing proper authentication, in order to distinguish privileged used from random internet users.

Buffer Overflow

- Currently one of the most common threat.
- It's easier to write beyond the bounds of an array in older languages.
- The attack is done with the same privileges as the program that is has affected.

Easiest targets:

1) Poorly written code.
2) Legacy code.

Best defenses:

- Write better code.
- Use modern programming languages.

Defenses:

1. **Compile-time:**
    1) **Choose modern program languages**.
    2) **Use safe coding techniques**.
    3) **Harden your programming environment**.
        i. OpenBSD project: OS which only 2 bugs were found over 13 years.
        ii. Windows: Used to have many bugs, doesn't have so much anymore.
        iii. Augmented compilers: Compilers that automatically checks for problems.
        iv. Use standard libraries.
    4) **Stack protection mechanisms**: Change the way that the stack operates.
        i. **Stack guard (canary)**: Warning sign that means not to trust the return address. Recompile needed.
        ii. **Stack shield:** Return address defender. Recompile not needed.
2. **Run-time:**
    1) **Executable address space protection:** Not allowing executable code in stack. Drawback: legitimate programs may actually need to use stack.
    2) **Address space randomization:** Randomize the address where stack and heap are located. Limits the spread of worm, because makes it more specific.
    3) **Guard pages:** Doesn't allow to go beyond the stack.
    4) **Deep pocket inspection:** Look for basic attempts to cause buffer overflow.

Information Flow

Information flow is the transfer of information from a variable to another variable. Not all flows are desirable; for example, a system should not leak any secret (partially or not) to public observers.

Defense against information flow:

1. **Through programs:**
    1) **Compiler-based mechanisms.**
        i. Compiler-based mechanisms check information flow throughout a program are authorized.
        ii. Explicit: y := x; (Information flow from y to x)
        iii. Implicit: If x == 1 then y := 0; Else y:= 1; If you entered in the second line, we would know about x.
        iv. Information class: The underscore below the variable, could be a integrity label, security label…
        v. Confidentiality: Information can flow from x to y if $\underline{x}$ is less than or equal $\underline{y}$.
        vi. Integrity: Information can flow from x to y if $\underline{x}$ is more than or equal $\underline{y}$.
        vii. A set of program statements is certified with respect to an information flow policy if the info flows in these statements do not violate the policy.
    2) **Execution-based mechanisms**.
        i. Fenton Data Mark Machine: Turns implicit flow (harder to detect) into explicit flow (easier to detect), so that they can be handled by usual mechanisms.
2. **Through channels:**
    1) **System mechanisms.**
        i. Putting something between a communication.
        ii. Example: Security Pipeline Interface. Checks of integrity.
        iii. Example: Secure Network Server Mail Guard. Checks for integrity and confidentiality.
    2) **Protocols.**
        i. Example: Protocol that doesn't rely on reusable passwords.
        ii. Example: One-time password. Instead of asking for a password, it asks for a challenge. There is a limit of challenges.
        iii. Example: Public key challenge. There isn't a limit of challenges.

Covert channels: Path of communication that you weren't aware about.

- Example: Using sounds not heard by humans to transfer data.
- Protection:
    - Identify covert channels.
    - Give every process the same amount of resources.
    - Inject randomness on the covert channel.

Containment/isolation:

- **Total isolation**: Not so useful, but there is no information flow in this case.
- **Partial isolation**: More useful**.** May use VMs, dockers, or sandboxes.