BSY vypracované okruhy na SZZ

- Side channel attacks, Basics of Steganography.
- Discretionary access control (Access control list, Capabilities).
- Detect intruders in operating systems. Hardening operating systems.
- Privilege escalation.
- Virtualization.
- Network protocols, TCP, DNS, TSL, mechanism of certificates, security of protocols.
- Firewalls, network intrusion detection, network intrusion prevention, intrusion deflection.
- Web attacks, SQL injection. • Denial of service attacks, binary exploitation

Side channel attacks

Proces využití postranní informace k napadení. Každý typ postranního kanálu je založen na jedné konkrétní měřitelné informaci. Často mívají tyto informace podobu fyzikální veličiny, kterou potenciální útočník má možnost určitým způsobem změřit.

Obecně:

Elektromagnetický

Zachycení elektromagnetických emisí z procesoru počítače při šifrování. Elektromagnetické záření může odhalit vzory, které útočníkovi umožní odvodit kryptografické klíče.

Akustický

Akustické útoky zahrnují použití vysoce citlivého mikrofonu k zachycení zvuku vydávaného součástmi počítače během kryptografických operací. Zvuk elektrické aktivity může poskytnout vodítka o operacích a klíči.

Napájení

Analýza napájení čipových karet: Útočníci sledují spotřebu energie čipové karty při provádění kryptografických operací, například šifrování RSA. Změny ve spotřebě energie mohou poskytnout informace o klíči použitém pro šifrování.

Meltdown a spectre

Využívají spekulativní vykonávání instrukcí v procesoru k zajištění přístupu k paměti jádra (Meltdown) či k uživatelské paměti (Spectre). Konkrétně např. branch prediction (Spectre).

Časový

Využívá měření času.

Např. zde:

```python
def check_password(password: str,  correct_password:str) -> bool:
    if len(password) != len(correct_password):
        return False
    for a,b in zip(password, correct_password):
        if a != b:
            return False
    return True
```

Délka vykonávání této funkce je úměrná počtu po sobě jdoucích správných znaků v hesle. Toho se dá snadno zneužít za pomoci měření délky vykonávání funkce.

Steganografie

Steganografie je vědní disciplína zabývající se utajením komunikace prostřednictvím ukrytí zprávy. Zpráva je ukryta tak, aby si pozorovatel neuvědomil, že komunikace vůbec probíhá. Síla této komunikace stojí a padá na jejím utajení (jedná se o takzvanou bezpečnost skrze utajení – security through obscurity), a proto zachycení skryté zprávy tak prakticky znamená její prolomení. Aby ani v tom případě nedošlo k prozrazení obsahu zprávy, zpravidla se kombinuje s dalšími metodami šifrování.

Jaké jsou rozdíly oproti šifrování?

- Steganografie skrývá samotnou existenci informací. Šifrování tuto skutečnost neskrývá.
- Steganografie může, ale nemusí být šifrována.
- Šifrování je navrženo tak, aby nebylo prolomitelné. Steganografie je navržena tak, aby nebyla nalezena.

Příklady:

- Textová: Skrývání textových zpráv v jiném textu, například pomocí neviditelných inkoustů nebo jemných změn ve formátování.
- Obrazová: Vkládání informací do digitálních obrázků změnou jednotlivých pixelů, které jsou okem nepostřehnutelné.
- Audio: Skrytí dat v audio souborech změnou některých vzorků zvuku
- Video: Vkládání informací do video souborů úpravou jednotlivých snímků

Access control list

Protokoly ACL nám umožňují použít specifičtější sadu oprávnění k souboru nebo adresáři, aniž bychom museli měnit tradiční písmena vlastnictví a oprávnění. Řeší problémy více skupin, které se soubory provádějí různé věci. Je více granulární.

K ACL lze přistoupit příkazem `getfacl example.txt`. Výstup může vypadat nějak takto:

```
user::rw-
user:john:rw-
user:jane:r--
group::r--
mask::rw-
other::r--
```

Písmena `r`, `w`, `x` symbolizují práva na čtení, zápis a spuštění souboru. Maska označuje maximální práva, která může někdo mít, tzn. zde může někdo maximálně číst a psát do souboru, ale už ne spouštět. Sekce other označuje práva pro ostatní uživatele.

Když bychom si přečetli ACL skrz příkaz `ls -alh` můžeme dostat něco jako `drwxr-xr-x 2 user user 4.0K Jun 9 12:00`, kde

- **d** je Typ souboru (d znamená adresář).
- **rwxr-xr-x:** Oprávnění (`rwx` pro vlastníka; `r-x` pro skupinu; `r-x` ostatní)
- **2:** Počet odkazů (včetně podadresářů).
- **user:** Vlastník souboru nebo adresáře.
- **user:** Skupina vlastníka souboru nebo adresáře.
- **4.0K:** Velikost souboru nebo adresáře (v tomto případě 4 kilobajty).
- **Jun 9 12:00:** Datum a čas poslední modifikace.
- **.:** Název souboru nebo adresáře (v tomto případě aktuální adresář).

Chceme-li např. nastavit, aby uživatel `newbie` mohl zapisovat, číst a spouštět soubor s názvem `test`, napíšeme `setfacl -m newbie:rwx test`.

Capabilities

Capabilities jsou sada speciálních oprávnění, která jádro poskytuje procesům vytvořeným ze spustitelných souborů. Capabilities poskytují velmi granulární kontrolu nad superuživatelskými a privilegovanými oprávněními, což umožňuje vyhnout se používání root účtu na všechno.

Tradičně v UNIXových systémech může být proces buď privilegovaný (efektivní uživatelské ID 0), nebo neprivilegovaný (eUID != 0).

- **Privilegované procesy** obcházejí všechny kontroly oprávnění jádra.
- **Neprivilegované procesy** jsou plně kontrolovány na základě jejich oprávnění, která jsou obvykle založena na efektivním UID, efektivním GID a doplňkovém seznamu skupin.

Příklady capabilities:

- **CAP_CHOWN**: Měnit vlastníka souboru.
- **CAP_DAC_OVERRIDE**: Přepsat kontroly přístupu k diskovým souborům.
- **CAP_FOWNER**: Převzít vlastnictví souboru a změnit jeho oprávnění.

Výhody použití capabilities:

1. **Konkrétní oprávnění**: Umožňuje přidělit konkrétní oprávnění bez potřeby přístupu root.
2. **Zvýšení bezpečnosti**: Omezuje rozsah oprávnění, což snižuje potenciální riziko zneužití.
3. **Flexibilita**: Umožňuje specifické oprávnění pro jednotlivé aplikace a procesy, což zlepšuje správu systémových zdrojů.

Detect intruders in operating systems

Chceme-li provést kontrolu na našem počítači můžeme zkontrolovat:

- logy souborů v `/var/log`
- běžící procesy např. pomocí `top`
- otevřené porty např. pomocí `ss -anp`
- procesy spouštěné při přihlášení v `/etc/rc.local`, `/etc/init.d`
- cronjobs např. pomocí `crontab -l`, `cat /etc/crontab`
- využívanou paměť, např. pomocí `cat /proc/meminfo`
- přístupy k souborům ve složkách `/proc` a `/sys`
- fyzikální metriky jako např. teplo vydané procesorem
- kdo je nyní přihlášený pomocí `w`
- poslední přihlášení pomocí `last`
- soubor `/var/log/utmp` zaznamenávající aktuální a dřívější přihlášení
- soubor `/var/log/btmp` zaznamenávající neúspěšné pokusy o přihlášení
- jsou-li hledané záznamy staré, můžeme zkontrolovat, zda je `logrotate` nerotoval či nekomprimoval

Také můžeme použít dedikovaný software jako např. AIDE (Advanced Intrusion Detection Environment), IDS (Intruder Detection System), nebo logcheck.

Hardening operating systems

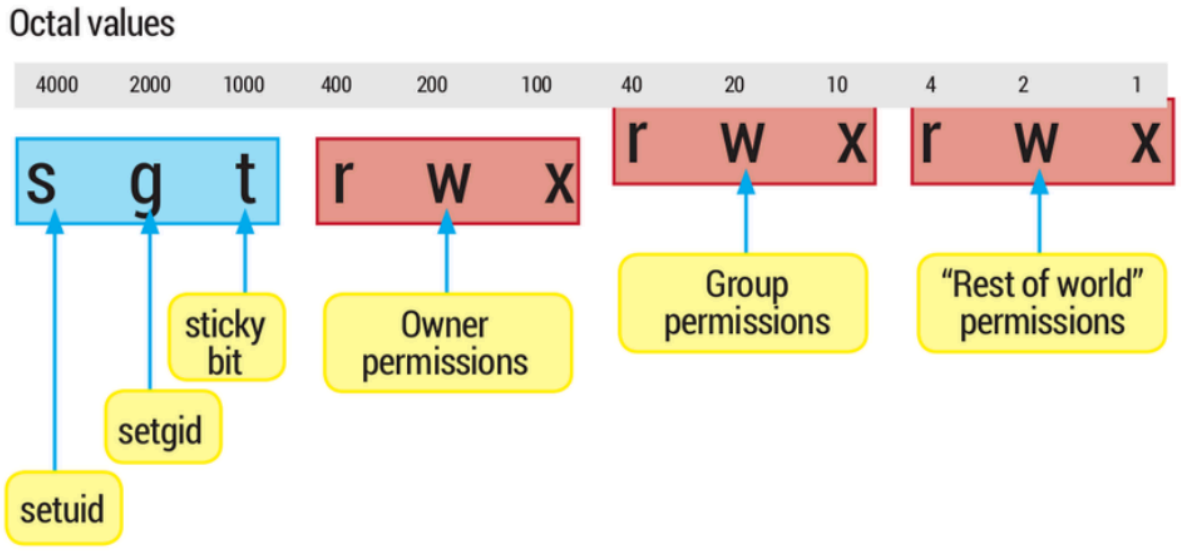
- Základní technikou je ochrana skrze **SSH tunneling**. Pokud by se chtěl např. někdo napojit na náš FTP server, použil by SSH tunnel, díky čemuž by veškerá komunikace probíhala šifrovaně a vyžadovala by SSH login.
- Můžeme také výrazně zvýšit zabezpečení SSH připojení skrze **vyžadování šifrovacích klíčů místo běžných hesel**. Login pak probíhá automaticky skrze ověření digitálního podpisu.
- V `/etc/ssh/sshd_config` můžeme nastavit, **kdo se vůbec může připojit**.
- **Zakázat login za uživatele root**. Tzn. přihlašovat se za jiný účet a všechny aplikace spouštět na uživatelské úrovni a až v případě potřeby jim dát sudo.
- V `/etc/pam.d/common-password` se dají nastavit **minimální požadavky na sílu hesla**. Jsou tam parametry na nastavení min. délky, min. počtu čísel apod.
- Nastavit **umask**, neboli základní oprávnění, která mají nově vzniklé soubory.
- U souborů používat **file attributes**, tj. metadata, která specifikují, co lze se souborem dělat. Např. `m` - nekomprimovat, `a` - append only.
- U procesů monitorovat jejich **capabilities** (viz předchozí kapitola).
- Udržovat **aktualizovaný operační systém**.

Privilege escalation

Spouštět soubory a procesy lze s či bez root práv. Root práva jsou potřeba např. na

- globální instalaci programů
- modifikaci kernelu
- používání omezených portů
- přidávání uživatelů

Příkaz s root právy spustíme pomocí předpony `sudo`. Snažíme se ho ale používat jen když je to nutné.
Některé binárky mají kromě klasických `rwx` oprávnění nastavený ještě tzv. SUID bit:



Ten říká, že když danou binárku spustíme, tak bude spuštěna s právy jejího vlastníka. Klasickým příkladem je změna hesla. Běžný uživatel nemá do `/etc/shadow` přístup, tudíž nemá ani způsob, jak se do souboru dostat, aby si změnil heslo. Z toho důvodu má binárka `passwd` nastavený SUID bit. Když jej běžný uživatel spustí a následuje jeho výzvy, může si skrz něj změnit heslo, což je možné díky tomu, že `passwd` běží díky SUID bitu jakožto root, a tak může změnit obsah `/etc/shadow`.
SUID bit zneužívá známý exploit, který určitým způsobem přinutí některé staré verze binárek s nastaveným SUID bitem provést nějakou root činnost, kterou by neměly.

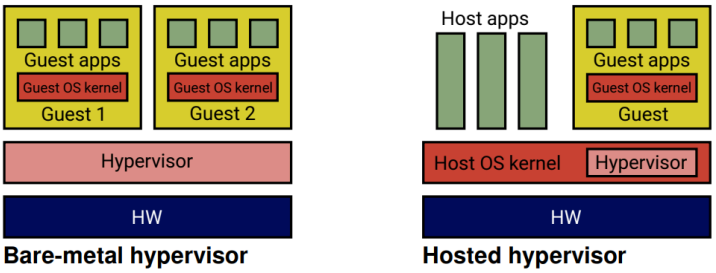
Spousta exploitů míří na to, jak se stát rootem. Proto je best practice používat root přístup co nejméně a jen když je to nutné.

Virtualizace

Virtualizace je jednou konkrétní metodou sandboxingu. Sandboxing je obecně bezpečnostní mechanismus, který umožňuje izolované spouštění softwaru, přičemž zajišťuje plnou kontrolu nad prostředím a prostředky, které jsou softwaru přístupné. Sandboxové techniky poskytované linuxem, které stojí za zmínku, jsou:

- Systémové volání **chroot**, které danému procesu a všem jeho potomkům změní lokaci kořenového adresáře na námi stanovenou
- **Cgroups** omezují a izolují využití zdrojů (CPU, paměť, diskové I/O, atd.) skupiny procesů, což zajišťuje kontrolované a efektivní řízení zdrojů.
- **Namespaces** poskytují izolaci globálních systémových prostředků (jako jsou ID procesů, síťová rozhraní, připojené soubory, atd.), vytvářejí oddělená prostředí pro běh procesů bez vzájemného ovlivňování, podobně jako kontejnery.

Klasická virtualizace zahrnuje kompletní simulaci výpočetního prostředí se všemi náležitostmi. Software, který toto provádí se nazývá **hypervizor**. Existují hypervizory dvou typů – typ 1 hardwarový (např. VMware) a typ 2 hostovaný (např. VirtualBox).



Hardwarový hypervizor běží přímo na hardwaru, a tak umožňuje dosažení nižších latencí redukcí nutné režie. Hypervizor typu 2 běží uvnitř hostitelského operačního systému. Hypervizory typu 2 typicky využívají techniku ***trap-and-emulate***, při které jsou privilegované operace a operace nad hardwarem tzv. *trapped* hypervizorem, a jejich efekt je emulovaný, což dává guest systému iluzi, že má přímý přístup k hardwaru.

Je důležité **neplést si virtualizaci s dokerizací**. Při virtualizaci dochází ke **kompletní simulaci operačního systému**, tzn. máme hostovací systém a virtuální stroj má svůj vlastní simulovaný operační systém, který si myslí, že má svůj vlastní hardware a prochází všemi standardními procesy. Oproti tomu dockerovský kontejner **využívá stejný kernel jako hostovací OS** a je oddělený za pomocí **namespaces**. Protože všechny distribuce linuxu používají stejný kernel, lze na kontejnerech rozbíhat libovolnou z nich. Docker kontejnery za cenu menší izolace spotřebovávají méně prostředků.

Dalším příkladem sandboxingu může být např. **emulace**, kdy se jeden program imituje jiný (typický příklad je emulace videoherních konzolí na počítači nebo proces spouštění Windowsu na Macu).

Síťové protokoly TCP, DNS, TLS

TCP (Transmission Control Protocol)

- **Spojově orientovaný protokol**: Vytváří spolehlivé spojení mezi dvěma koncovými body.
- **Zaručené doručení**: Data jsou doručena v pořadí a bez chyb.
- **Kontrola toku**: Řídí rychlost přenosu dat mezi odesílatelem a příjemcem.
- **Detekce a oprava chyb**: Používá sekvenční čísla a potvrzení (acknowledgements).
- **Použití**: Například při přenosu webových stránek (HTTP), e-mailů (SMTP), souborů (FTP).

DNS (Domain Name System)

- **Překlad jmen na IP adresy**: Překládá doménová jména na IP adresy, které jsou potřebné pro směrování dat.
- **Hierarchický systém**: Struktura DNS je hierarchická s kořenovými servery, top-level doménami a nižšími úrovněmi.
- **Distribuovaný systém**: DNS je rozdělený do mnoha serverů po celém světě, což zvyšuje robustnost a dostupnost.
- **Caching**: Ukládá dočasně překladové záznamy pro rychlejší přístup v budoucnu.
- **Použití**: Při vyhledávání webových stránek a dalších internetových služeb.

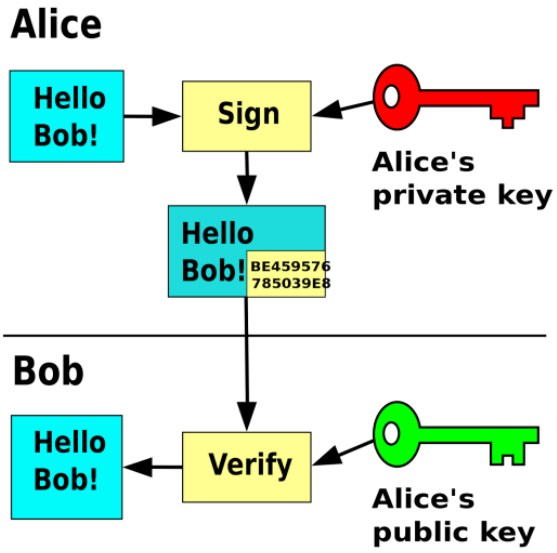
TSL (Transport Layer Security)

- **Šifrování komunikace**: Zajišťuje šifrování dat mezi klientem a serverem, čímž chrání před odposlechem.
- **Autentizace**: Ověřuje identitu serveru (a případně i klienta) pomocí certifikátů.
- **Integrita dat**: Zajišťuje, že data nebyla změněna během přenosu pomocí kryptografických hashovacích funkcí.
- **Použití**: Například při zabezpečení webových stránek (HTTPS), e-mailů (SMTPS), nebo VPN.

Mechanismus za certifikáty

Při symetrickém šifrování lze stejným klíčem zašifrovat i dešifrovat zprávu. U asymetrického šifrování jsou dva různé klíče (šifrovací a dešifrovací).
Digitální podpis funguje následovně:

1. Alice zahashuje zprávu; ten hash zašifruje soukromým klíčem a odešle výsledný podpis společně se zprávou.
2. Bob z přijaté zprávy odejme podpis a veřejným klíčem Alice jej dešifruje.
3. Bob porovná hash zprávy s dešifrovaným podpisem a zkontroluje, zda jsou stejné.



Příklad fungování certifikátu:

1. 1. Vlastník webové stránky požádá certifikační autoritu (CA) o vydání certifikátu pro svou doménu. Po úspěšném ověření vydá CA digitální certifikát, který obsahuje veřejný klíč žadatele, informace o doméně a údaje o certifikační autoritě.
2. Vlastník webu nainstaluje certifikát na svůj webový server. Server je nakonfigurován tak, aby používal HTTPS místo HTTP, což znamená, že veškerá komunikace mezi klientem a serverem bude šifrována pomocí SSL/TLS.
3. Uživatel vkročí na tuto stránku, během čehož se provede tzv. SSL/TLS handshake – server posílá svůj digitální certifikát webovému prohlížeči.
4. Webový prohlížeč ověří certifikát serveru. Zkontroluje, zda certifikát byl vydán důvěryhodnou certifikační autoritou, platnost certifikátu a zda certifikát nebyl odvolán.
5. Pokud je certifikát validní, prohlížeč vygeneruje symetrický klíč pro šifrování relace a zašifruje ho veřejným klíčem serveru, který získal z certifikátu.
6. Server dešifruje symetrický klíč pomocí svého soukromého klíče. Nyní mohou server a klient komunikovat bezpečně pomocí symetrického šifrování.

   Důležitá poznámka – asymetrické šifrování je velmi pomalé. Používá se výhradně pro bezpečné předání klíče pro symetrickou šifru, kterou se šifruje následná komunikace.

Firewall

Firewall je bezpečnostní zařízení nebo software, které monitoruje a kontroluje příchozí a odchozí síťový provoz na základě předem definovaných bezpečnostních pravidel. Jeho hlavní funkcí je zabránit neoprávněnému přístupu do nebo z privátní sítě a chránit ji před různými typy kybernetických útoků.

Pravidla firewallu, známá také jako **firewallové politiky** nebo **pravidla ACL (Access Control List)**, určují, jaký síťový provoz je povolen a jaký je blokován. Tato pravidla jsou definována administrátorem a typicky zahrnují následující komponenty:

1. **Zdrojová adresa**: IP adresa nebo rozsah IP adres odkud paket přichází.
2. **Cílová adresa**: IP adresa nebo rozsah IP adres kam paket směřuje.
3. **Protokol**:Typ síťového protokolu (např. TCP, UDP, ICMP).
4. **Zdrojový port**: Port číslo na zdrojovém zařízení.
5. **Cílový port**: Port číslo na cílovém zařízení.
6. **Akce**: Co má firewall udělat s paketem (např. povolit nebo zablokovat).

Firewally mohou být čistě softwarové (na aplikační vrstvě) a bránit tak jednotlivé uzly v síti. Ty nastavujeme u sebe v systému. Oproti tomu síťové firewally fungují pro celou síť a jsou obvykle součástí routeru. Jsou zpravidla kombinací softwaru a hardwaru a nastavují se v administračním rozhraní routeru.

Další dělení:

- **Paketové filtry**: Sledují a filtrují síťové pakety na základě IP adres, portů a protokolů.
- **Aplikační brány (Proxy firewally)**: Analyzují data na aplikační vrstvě, mohou rozpoznat a blokovat specifické aplikace nebo protokoly.
- **Stavové paketové filtry (Stateful firewally)**: Udržují stav spojení a rozhodují na základě kontextu spojení (např. zda je paket součástí existující relace).
- **Stavové paketové filtry s kontrolou známých protokolů**: Kombinují stavovou kontrolu s detekcí specifických protokolů a mohou zahrnovat prvky detekce průniků (IDS).

Network intrusion detection

- **Intrusion Detection System (IDS)**: Monitoruje síťový provoz a systémy na podezřelé aktivity nebo porušení politik. Existují dva hlavní typy IDS: Network-based IDS (NIDS) a Host-based IDS (HIDS) (příklad: Snort).
- **Intrusion Prevention System (IPS)**: Kromě detekce podezřelých aktivit také aktivně zasahuje a blokuje útoky. IPS je často implementován jako rozšíření IDS (příklad: Cisco Firepower).
- **Firewallové logy**: Analýza logů z firewallu může odhalit neobvyklé nebo podezřelé aktivity. Mnoho firewallů má integrované nástroje pro monitorování a analýzu provozu.
- **Security Information and Event Management (SIEM)**: Kombinuje a analyzuje data z různých zdrojů (např. IDS, IPS, firewallů, serverových logů) pro detekci anomálií a potenciálních bezpečnostních incidentů (příklad: Splunk, IBM QRadar).
- **Behaviorální analýza**: Sleduje a analyzuje běžné chování uživatelů a systémů. Jakmile se objeví neobvyklé chování, je označeno jako potenciální hrozba (příklad: Vectra AI).
- **Penetrační testování**: Aktivní metoda, kde bezpečnostní odborníci simulují útoky na síť, aby identifikovali slabá místa a možné vstupní body pro vetřelce (příklad: Metasploit).

Network intrusion prevention

- **Firewallové nastavení**: Implementace a správná konfigurace firewallů je klíčová pro blokování neoprávněného přístupu. Používání různých typů firewallů, jako jsou paketové filtry a stavové paketové filtry, poskytuje více vrstev ochrany.
- **Aktualizace a záplaty**: Pravidelné aktualizace veškerého softwaru a operačních systémů jsou nezbytné pro ochranu proti známým zranitelnostem. Bezpečnostní záplaty by měly být aplikovány co nejdříve po jejich vydání.
- **Silná autentizace**: Zavedení silných hesel a vícefaktorové autentizace (MFA) zvyšuje bezpečnost přístupu do sítě a kritických systémů. MFA kombinuje něco, co uživatel ví (heslo), s něčím, co má (mobilní zařízení), nebo něčím, co je (biometrie).
- **Intrusion Detection/Prevention Systems (IDS/IPS)**: Nasazení IDS a IPS systémů umožňuje detekci a prevenci podezřelých aktivit v reálném čase. IDS monitoruje síťový provoz na známé vzory útoků, zatímco IPS aktivně zasahuje a blokuje identifikované útoky.

Web attacks, SQL injection

- **SQL Injection (SQLi)**: Útok, při kterém útočník vloží škodlivý SQL kód do dotazu, který aplikace provádí nad databází. Například ve formuláři pro přihlášení může útočník zadat jako uživatelské jméno `admin' OR '1'='1` a jakékoliv heslo, což vytvoří dotaz `SELECT * FROM users WHERE username = 'admin' OR '1'='1' AND password = 'password';`.
  Tento dotaz vrátí všechny záznamy v tabulce `users`, což umožní útočníkovi přístup.
- **Cross-Site Request Forgery (CSRF)**: Útok, při kterém útočník přiměje uživatele k provedení nechtěných akcí na webové aplikaci, kde je uživatel přihlášen. Například útočník může vytvořit skrytý formulář na své stránce
  ```
  <form action="http://example.com/transfer" method="POST">
      <input type="hidden" name="amount" value="1000">
      <input type="hidden" name="to" value="attacker_account">
      <input type="submit">
  </form>,
  ```
  který se automaticky odešle pomocí JavaScriptu. Pokud uživatel navštíví tuto stránku, formulář se odešle a peníze se převedou na útočníkův účet.
- **Cross-Site Scripting (XSS)**: Útok, při kterém útočník vloží škodlivý skript do obsahu webové stránky, která je následně zobrazena ostatním uživatelům. Například útočník může vložit škodlivý skript do profilového obrázku uživatele: `<img src="profile.jpg" onload="alert('XSS Attack');">`. Tento skript se spustí, když si jiný uživatel načte profil s tímto obrázkem. Může být zneužit k ukradení cookies, session ID nebo k provedení jiných škodlivých akcí v kontextu uživatele.
- **Insecure Deserialization**: Útoky, které využívají zranitelnosti při deserializaci dat, což může vést k vzdálenému spuštění kódu nebo eskalaci oprávnění. Například útočník může poslat manipulovaný serializovaný objekt, který při deserializaci spustí škodlivý kód: `'O:8:"UserData":2:{s:4:"name";s:5:"admin";s:8:"isAdmin";b:1;}'`. Tento objekt může při deserializaci nastavit `isAdmin` na true a útočník získá administrátorská práva.
- **Server-Side Request Forgery (SSRF)**: Útok, při kterém útočník přiměje server, aby provedl síťové požadavky na neoprávněné nebo škodlivé cíle. Například útočník může zadat škodlivou URL do formuláře, který server použije k načtení dat: http://example.com/fetch?url=http://attacker.com/malicious. Server pak provede požadavek na http://attacker.com/malicious, což může vést k odhalení interních systémů nebo jiným škodlivým akcím.

Denial of service attacks

Útok typu DoS (Denial-of-Service) je útok, jehož cílem je odstavit počítač nebo síť a znemožnit tak jejich dostupnost. Útoky DoS se provádějí tak, že cíl je zahlcen provozem nebo jsou mu zaslány informace, které způsobí jeho zhroucení. V obou případech útok DoS připraví legitimní uživatele (tj. zaměstnance, členy nebo majitele účtů) o službu nebo prostředek, který očekávali.

Oběti útoků DoS se často zaměřují na webové servery významných organizací, jako jsou bankovní, obchodní a mediální společnosti nebo vládní a obchodní organizace. Ačkoli útoky DoS obvykle nevedou ke krádeži nebo ztrátě významných informací nebo jiných aktiv, jejich řešení může oběť stát mnoho času a peněz v podobě nákladů.

Existují dvě obecné metody útoků DoS: zahlcení služeb nebo zhroucení služeb. K útokům typu flood dochází tehdy, když systém přijímá příliš velký provoz, než aby jej server mohl vyrovnávací pamětí vyrovnat, což způsobí jejich zpomalení a nakonec zastavení. Mezi oblíbené útoky typu flood patří např:

- **Buffer overflow** - nejčastější útok DoS. Jeho podstatou je odeslání většího provozu na síťovou adresu, než na jaký byl systém programátory postaven. Zahrnuje útoky uvedené níže, kromě dalších, které jsou navrženy tak, aby využívaly chyby specifické pro určité aplikace nebo sítě
- **ICMP flood**: využívá špatně nakonfigurovaných síťových zařízení odesíláním podvržených paketů, které pingují každý počítač v cílové síti, nikoli pouze jeden konkrétní stroj.
- **SYN flood** - odesílá požadavek na připojení k serveru, ale nikdy nedokončí handshake. Pokračuje tak dlouho, dokud nejsou všechny otevřené porty přesyceny požadavky a žádný není k dispozici pro připojení legitimních uživatelů.

Dalším typem útoku DoS je útok DDoS (Distributed Denial of Service). K útoku DDoS dochází, když více systémů organizuje synchronizovaný útok DoS na jeden cíl.
Jak předejít DoS a DDos útoku:

- Použití firewallů: Firewally mohou blokovat podezřelý provoz na základě pravidel.
- Load balancery: Load balancery rozkládají provoz mezi více servery, což snižuje riziko přetížení jednoho serveru.
- Rate limiting: Omezení počtu požadavků od uživatelů nebo IP adres za určitý časový interval omezuje efektivitu DoS a DDoS útoků.
- Anti-DDoS služby: Specializované služby jako Cloudflare, Akamai nebo AWS Shield jsou navrženy pro ochranu před DDoS útoky.

Binary exploitation

Jde o disciplínu, která se snaží přesměrovat vykonávání kódu jinak než bylo zamýšleno vývojářem. Typickým příkladem je buffer overflow. Viz následující kód:

```
#include <stdlib.h>
#include <stdio.h>

int main() {
    volatile int modified;
    char buffer[64];

    modified = 0;

    gets(buffer);

    if (modified != 0 ) {
       printf("Access granted\n");
    } else {
       printf("Access denied\n");
    }

    return 0;
}
```

Proměnná `modified` je na zásobníku hned za proměnnou `buffer`. Funkce `gets` nemá ošetřenou situaci, kdy přijde na vstup delší řetězec než je očekávaný. Teoreticky, když na vstup vložíme 64 bytů náhodného textu a následně 8 nenulových bytů tak se vytiskne `Access granted`. V praxi se to ale může trochu lišit v závislosti na architektuře. Některé architektury např. zarovnávají na 16 bytů apod. Takže bychom např. museli místo 8 dalších bytů přidat třeba 26.
V tomto případě lze útoku předejít použitím bezpečnějšího `fgets`.

Další důležitý koncept je, co by se stalo, kdybychom nechali buffer přetéct ještě dál? Mohli bychom dokonce přepsat i návratovou hodnotu z funkce tak, aby se spustila jiná funkce. Bývá to ale složitější – je nutné přepsat správné registry atd. Tím, jak tohoto dosáhnout, se zabývá disciplína *Return Oriented Programming*.

Jak se ubránit?

- Pomocí tzv. *stack canaries*, což jsou hodnoty, které kompilátor vloží před návratovou hodnotu funkce, díky čemuž v případě jejich změněné hodnoty pozná, že došlo k útoku.
- ASLR – feature operačního systému, která randomizuje adresní prostor tak, aby bylo pro útočníka těžší v ní spolehlivě přeskakovat
- NX (non executable) stack – stack část procesu je flagnutá jako read only, což zabraňuje zápisu na něj skrz buffer overflow útok