

# Securitate Software

## I. Introducere

# Continut

- 1 Organizare
  - Prezentarea cursului
  - Organizare
- 2 Introducere
- 3 Vulnerabilități
- 4 Necesitatea auditului
- 5 Clasificarea vulnerabilităților
- 6 Amenințări

# De ce?

## Obiective:

- să învățăm să dezvoltăm soft mai sigur
  - ▶ proiectare
  - ▶ implementare
- scrie cod sigur
  - ▶ cunoaște și înțelege vulnerabilități soft obișnuite
  - ▶ evitarea acestor vulnerabilități (design, API securizat)
- code review/audit din punct de vedere al securității
  - ▶ știți ce să căutați în cod
  - ▶ știți cum să cautați vulnerabilități de securitate (instrumente, strategii, etc.)
- tratarea vulnerabilităților găsite
  - ▶ evaluarea probabilității ca vulnerabilitatea găsită să fie exploatată
  - ▶ evaluarea importanței vulnerabilității găsite (ex. minoră, severă, critică)

# Black Hat and White Hat

Black hat: perspectiva atacatorului

- găsește vulnerabilități software
- exploatarea acestor vulnerabilități

White hat: perspectiva apărătorului

- previne apariția în cod a vulnerabilităților
- îngreunează exploatarea vulnerabilităților

## Conținut curs

- 1 Introducere, concepte de bază
- 2 Vulnerabilități legate de coruperea memoriei (ex. buffer overflow)
- 3 Vulnerabilități specifice limbajului C (ex. integer overflow, type conversion)
- 4 Vulnerabilități în utilizarea și manipularea șirurilor de caractere
- 5 Vulnerabilități specifice sistemelor de operare UNIX / Linux
- 6 Vulnerabilități specifice sistemelor de operare Windows

## Conținut curs II

- 7 Vulnerabilități de sincronizare (situații de concurență)
- 8 Vulnerabilități web
- 9 Vulnerabilități de criptografie și specifice aplicațiilor de rețea
- 10 Metode de proiectare, implementare și evaluare a aplicațiilor din punctul de vedere al securității
- 11 Code audit: design review
- 12 Code audit: operational review
- 13 Code audit: application review

# Bibliografie

- M. Down, J. McDonald, J. Schuh, "The Art of Software Security Assessment. Identifying and Preventing Software Vulnerabilities", Addison-Wesley, 2007
- M. Howard, D. LeBlanc, J. Viega, "24 Deadly Sins of Software Security. Programming Flaws and How to Fix Them", McGraw Hill, 2010
- M. Howard, D. LeBlanc, "Writing Secure Code for Windows Vista", Microsoft Press, 2007
- G. McGraw, "Software Security: Building Security In", Addison-Wesley, 2006
- R. Seacord, "CERT C Coding Standard: 98 Rules for Developing Safe, Reliable, and Secure Systems", Addison-Wesley, 2nd edition, 2014
- "Common Weaknesses Enumeration (WCE)",  
<http://cwe.mitre.org/data/index.html>
- The Open Web Application Security Project (OWASP), "Software Vulnerabilities",  
<https://www.owasp.org/index.php/Category:Vulnerability>

# Organizare

- curs: Mihai Suciu ([www.cs.ubbcluj.ro/~mihai-suciu/ss/](http://www.cs.ubbcluj.ro/~mihai-suciu/ss/))
- laborator:
  - ▶ Daniel Ticle (<http://www.cs.ubbcluj.ro/~daniel/ss/>)
  - ▶ Alexandru Mihai LUNGANA-Niculescu

- Pagina web a cursului

[www.cs.ubbcluj.ro/~mihai-suciu/ss/](http://www.cs.ubbcluj.ro/~mihai-suciu/ss/)

<https://moodle.cs.ubbcluj.ro/course/view.php?id=27>

pentru Moodle:

- ▶ vă creați un cont (dacă aveți un cont, numele de utilizator și parola se pot recupera)
- ▶ se recomandă ca numele de utilizator sa fie de forma prenume.nume, ex. mihai.suciu
- ▶ curs: Securitate Software



# Structura

Cursuri: 2 ore / săptămână

Laboratoare: 2 ore / săptămână

Orar:

<https://www.cs.ubbcluj.ro/files/orar/2020-1/disc/MLR8114.html>

# Precondiții

- cursuri: Arhitectura sistemelor de calcul, Sisteme de operare, Structuri de date și algoritmi, Baze de date, Programare web
- competențe: Programare în C, cunoștințe de bază ale arhitecturii Intel x86, elemente de bază în programarea web și SQL.

## Metoda de evaluare și cerințe

- **20%** Activitate la curs (teste la curs); nu se impune nota minimă.
- **40%** Activitate la laborator (4 teste grilă pe Moodle, susținute pe parcursul semestrului); nota la fiecare test trebuie să fie **cel puțin 5**.
- **40%** Colocviu (test grilă pe Moodle); nota trebuie să fie **cel puțin 5**.
- **1** punct bonus pentru activitate deosebită la laborator.
- Restanțe:
  - ▶ Pentru restanțe formula rămâne identică. Nota de la colocviu se înlocuiește cu nota obținută la un nou test susținut în sesiunea de restanțe.
  - ▶ În sesiunea de restanțe **nu se poate recupera punctajul pentru activitate la curs, respectiv activitate la laborator!** (acestea fiind activități ce se desfășoară pe parcursul semestrului)
- Nota minimă la colocviu trebuie să fie 5. Nota la fiecare test de la laborator trebuie să fie minim 5.
- Este necesar un număr de **minim 10 prezențe** la laborator. Este necesară participarea studenților la ambele ore de laborator pentru a fi luată în considerare prezența.

# Vulnerabilități software

# Obiective

- importanța și complexitatea în scrierea de cod sigur
- principalele aspecte legate de cod sigur (secure coding)

## De ce? Motivație

importanța și omniprezența software

importanța proprietăților de securitate pentru software

# Ce anume? Conținut

- ❶ vulnerabilități software
  - ▶ înțelegerea unor vulnerabilități elementare și implicațiile acestora
- ❷ verificarea codului (code auditing)
  - ▶ instrumente pentru a înțelege și a evalua cât de sigur este codul
- ❸ scrie cod sigur
  - ▶ tehnici și API pentru a evita vulnerabilități în cod

# Cum? Objective

- ① cunoștințe de bază pentru a efectua o evaluare cuprinzătoare a securității unei aplicații:
  - ▶ diseca o aplicație
  - ▶ descoperi vulnerabilități
  - ▶ evalua consecințele / riscurile vulnerabilităților găsite
  - ▶ eficientizarea procesului de audit: concentrare pe aspectele relevante legat de securitatea codului
  - ▶ identificarea vulnerabilităților critice
- ② capacitatea de a scrie cod corect din punct de vedere al securității
  - ▶ proiecta și scrie cod sigur
  - ▶ înlocuirea codului greșit cu cel corect



# Definiții

- defecte / deficiențe (defects / weaknesses): comportament greșit al codului
  - ▶ greșeli (flaws) în design
  - ▶ defecte (bugs) în implementare
- defecte relevante din punct de vedere al securității ce ar permite unui atacator să le exploateze
- nu toate vulnerabilitățile se datorează defectelor software, ex. aplicație ce permite accesul la fișiere critice sistemului de operare

# Corectitudine vs. Securitate

- corectitudine / fiabilitate

- ▶ aplicația se comportă așa cum a fost proiectată (chiar și în cazul unor intrări și condiții excepționale)
- ▶ siguranța: sistemul nu este afectat de execuția aplicației, sistemul este protejat de aplicație

- securitate

- ▶ prevenirea comportamentului nedorit în cazul în care un atacator rău intenționat abuzează de aplicație
- ▶ aplicația nu este afectată în mod neintenționat de sistem, aplicația este protejată de mediul înconjurător

# Corectitudine vs. Securitate II

- perspectiva corectitudinii

- ▶ incorect, dar comportamentul se manifesta rar
- ▶ imposibil de a livra un soft fara defecte
- ▶ reparate prioritar defectele ce afecteaza utilizatorii obișnuiți

- perspectiva securității

- ▶ atacatorul nu este un utilizator obișnuit
- ▶ încearcă să găsească defecte pentru moduri neobișnuite de utilizare a aplicației (non-regular usage cases)
- ▶ gasește un mod de a folosi aceste defecte în avantajul lui
- ▶ evita blocarea / oprirea aplicației exploatare
- ▶ schimbă comportamentul aplicației

## Obiective din p.d.v. al securității

Pentru a asigura securitatea unei aplicații trebuie:

- eliminate greșelile și defectele din cod (ce ar permite exploatarea aplicației)
- aplicații mai greu de exploatat

# Securitate Software [2]

- proiectarea și implementarea aplicației având în vedere și aspectele de securitate
- codul este prioritar
- diferențe la nivel de securitate între aplicație și sistemul de operare
  - ▶ nu se pot impune politici de securitate specifice aplicației
  - ▶ nu se poate restricționa fluxul de informație
- diferențe la nivel de securitate între aplicație și "perimetru" (ex firewall, IDS)
  - ▶ nu se opresc atacuri pe date/canale nefiltrate
  - ▶ bazată pe analiza sintactică vs. analiza semantică
  - ▶ reguli de securitate de granularitate mare — > penalizarea performanței, trebuie găsit un compromis
- securitate software — > cod cu greșeli

**Citiți documentația!!!!**

# Politici de securitate

- securitatea unui sistem este dată de politicile de securitate
- o listă de reguli (ce este permis și ce este interzis)
- specificare formală: costisitoare, nu este practică în majoritatea cazurilor
- formal, document scris cu mai multe clauze
- informal, colecție ambiguă de așteptări

# Așteptări / premise

## ① confidențialitate

- ▶ datele private trebuie pastrate secret
- ▶ ascunderea datelor și dovada existenței datelor
- ▶ aspecte legate de intimitate
- ▶ mecanisme: compartimentare, autentificare și autorizare, criptare

## ② integritate

- ▶ încredere și corectitudinea datelor
- ▶ împiedicarea modificării datelor
- ▶ prevenirea și detectarea alterării neautorizate a datelor și a sursei datelor
- ▶ mecanisme: autentificare, autorizare, criptografie

## ③ disponibilitate

- ▶ capacitatea de a folosi datele, resursele, serviciile
- ▶ atacuri DoS (denial-of-service) <http://www.digitalattackmap.com/#anim=1&color=0&country=ALL&list=0&time=17443&view=map>
- ▶ mecanism: autentificare, autorizare, limitarea resurselor



# Necesitatea auditului

- majoritatea dezvoltatorilor de aplicații nu oferă garanții legate de integritatea aplicației
- se pune accentul pe funcționalitate, disponibilitate și stabilitate
- tendința: testare și pe partea de securitate
- analiza automată a codului, testare din punct de vedere al securității, audit manual al codului
- *code audit* = analiza codului aplicației (sursa sau executabil) pentru a descoperi eventualele vulnerabilități exploatabile
- situații ce implică audit de cod:
  - ▶ in-house pre-release software audit
  - ▶ in-house post-release software audit
  - ▶ third-party product range comparison
  - ▶ third-party evaluation
  - ▶ third-party preliminary evaluation
  - ▶ independent research

# Audit vs Black Box testing

Black box testing:

- evaluarea unui sistem software doar prin manipularea interfețelor expuse
- fuzz-testing
- avantaj: instrumente automate, rezultate rapide
- dezavantaje: proces limitat, nu se analizează multe posibile căi de execuție

# Auditul codului și ciclul dezvoltării

Auditul de cod ar putea fi efectuat în orice etapă a ciclului de viață al sistemului (SDLC - System Development Life Cycle)

- 1 studiul de fezabilitate
- 2 definirea cerințelor
- 3 proiectarea aplicației
- 4 implementare
- 5 integrare și testare
- 6 funcționare și întreținere

# Vulnerabilități în proiectare (Design Vulnerabilities)

- vulnerabilități la nivel înalt, deficiențe de arhitectură, probleme cu cerințe sau constrângeri de software (SDLC 1, 2, 3)
- probleme care apar din cauza unei greșeli fundamentale în proiectarea software-ului
- chiar dacă este implementat corect, software-ul nu este încă sigur
- presupuneri greșite privind mediul
- proiectarea aplicației este motivată de cerințe și specificații
- exemplu: TELNET - lipsa de criptare

# Vulnerabilități în implementare

- defecte tehnice
- în general aplicația face ceea ce ar trebui dar problema este modul în care se desfășoară operațiunea
- mediul în care se execută aplicația
- SDLC 4,5
- exemple: buffer overflow, SQL injection

# Vulnerabilități în funcționare

- apar prin procedurile operaționale și utilizarea generală a unui soft într-un anumit mediu
- nu este prezent în codul sursa, ci în modul în care soft-ul interacționează cu mediul său
- probleme cu:
  - ▶ configurarea soft-ului în mediul său
  - ▶ configurarea soft-ului și a calculatoarelor
  - ▶ procese automate și manuale ce se execută
  - ▶ anumite tipuri de atacuri asupra utilizatorilor sistemului (*social engineering* și furt)
- exemplu: utilizarea TELNET într-un mediu care îl expune la atacuri (datorită defectului său de proiectare cunoscut)

## "zona gri"

- dificil de a face distincție între etapele de proiectare și implemetare
- nu există o distincție clară între vulnerabilitățile de funcționare și vulnerabilitățile de implementare și de proiectare

# Clasificare [5]

## ① Input Validation and Representation

- ▶ buffer overflow, command injection, XSS, format strings, illegal pointer, integer overflow, SQL injection, XML validation etc.

## ② API Abuse

- ▶ dangerous functions, directory restrictions, exception handling, unchecked return values etc.

## ③ Security Features

- ▶ insecure randomness, least privilege violation, password management, privacy violation

## ④ Time and State

- ▶ deadlock, TOCTOU, insecure temporary files, signal handling etc.

## ⑤ Errors

- ▶ catch “null-pointer exception”, empty catch block, overly-broad catch block etc.



## Clasificare II [5]

### ⑥ Code Quality

- ▶ double free, memory leaks, null dereference, undefined behavior, uninitialized variables, use after free etc.

### ⑦ Encapsulation

- ▶ comparing classes by name, data leaking between users, leftover debug code, private array-typed field returned from a public method, public data assigned to private array-typed field, system information leak

### ⑧ Environment (extra)

- ▶ everything that is outside of the source code, still critical to the application's security

# Top vulnerabilități

## “OWASP Top Ten Project” [3]

- 1 Injection
- 2 Broken Authentication and Session Management
- 3 Cross-Site Scripting (XSS)
- 4 Insecure Direct Object References
- 5 Security Misconfiguration
- 6 Sensitive Data Exposure
- 7 Missing Function Level Access Control
- 8 Cross-Site Request Forgery (CSRF)
- 9 Using Components with Known Vulnerabilities
- 10 Unvalidated Redirects and Forwards

# Date de intrare si fluxul datelor

- majoritatea vulnerabilităților software rezultă din comportamentul neașteptat declanșat de răspunsul aplicației la date de intrare neadecvate
- datele de intrare dăunătoare sunt injectate / furnizate de un atacator
- în etapa de recenzie a codului trebuie să se țină cont de datele controlate de utilizator
- atacatorul poate trimite datele în mai multe moduri
- în etapa de recenzie a codului trebuie să se țină cont de fluxul datelor (de unde vin datele)
- greu de analizat

# Relația de încredere

- între diferite componente ale unui sistem software
- există un grad diferit de încredere
- relațiile de încredere sunt cruciale în fluxul de date
- determină cantitatea de date schimbate între componente pentru validare (overhead)
- ia în considerare natura tranzitivă a încrederii
- încredere gresită ar putea duce la vulnerabilități

# Presupuneri greșite

- proiectanții și programatori fac presupuneri neîntemeiate asupra
  - ▶ validitatea și formatul datelor primite
  - ▶ securitatea programelor de sprijin
  - ▶ potențiala ostilitate a mediului
  - ▶ capacitatea atacatorilor și a utilizatorilor
  - ▶ comportamentul și nuanțele anumitor apeluri API sau funcții de limbă
- similar cu încrederea pierdută
- un atacator caută ipoteze pe care un dezvoltator le-a făcut, încercând să eludeze aceste presupuneri furnizând aplicației date greșite

# Presupuneri asupra datelor de intrare

- date de intrare de lungime finită
- structuri de intrare bine formatate

# Presupuneri asupra interfețelor

- interfețele sunt mecanisme prin care componentele software comunică
- proprietățile de securitate ale unor astfel de interfețe sunt de multe ori neînțelese
- dezvoltatorii presupun în mod fals că numai utilizatorii de încredere pot accesa și utiliza interfețele
- dezvoltatorul supraestimează dificultatea unui atacator de a accesa o interfață
- ex. un protocol personalizat de comunicare și criptare

# Presupuneri asupra mediului

- unele vulnerabilități apar atunci când un atacator manipulează mediul de bază al aplicației
- astfel de vulnerabilități sunt cauzate de ipoteze făcute în legătură cu mediul
- dezvoltatorul ar trebui să înțeleagă pe deplin potențialele problemele de securitate ale fiecărei tehnologii de sprijin
- exemplu: problema `"/tmp race"`



## Lecturi recomandate

- ① “The Art of Software Security Assessments”, chapter 1, “Software Vulnerability Fundamentals”, pp. 1 – 23
- ② “Seven Pecernious Kingdoms: A Taxonomy of Software Security Errors”, <https://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf>

# Bibliografie

- M. Dowd, J. McDonald, and J. Schuh. The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Addison-Wesley Professional.
- M. Hicks. Software Security Course (Univ. of Maryland). <https://class.coursera.org/softwaresec-006>, Fall 2015. Accessed: 2015-09-17
- OWASP. OWASP Top 10. The Ten Most Critical Web Application Security Risks. [https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project#tab=OWASP\\_Top\\_10\\_for\\_2013](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013), 2013. Accessed: 2016-10-03
- Steve Christey (MITRE). CWE/SANS Top 25 Most Dangerous Software Errors. <http://cwe.mitre.org/top25/index.html>, September 2011. Accessed: 2016-10-03.
- K. Tsipenyuk, B. Chess, and G. McGraw. Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. Security and Privacy, IEEE, 3(6):81–84, Nov. 2005