Software Security Testing

Lecturer Paul POCATILU, PhD
Economic Informatics Department, Academy of Economic Studies, Bucharest

In today's world, the security of software applications is one of the most important aspect that has to be considered. In order to be more confident that an application is secure, it has to be tested using specific methods and techniques. Secure software is good quality software. This paper presents the main characteristics of the software security testing.
Keywords: *applications development, software security, software testing, penetration testing.*

Introducere

Asigurarea securității aplicațiilor informatice este una dintre cele mai importate probleme care trebuie rezolvată în contextul societății informaționale. Aplicațiile informatice primesc, prelucrează, stochează şi transmit o multitudine de informații şi date, utilizând o multitudine de tehnologii de transport şi interfață (zone de memorie, socket-uri, variabile de mediu, fişiere, protocoale de comunicație etc.). Identificarea punctelor slabe ale aplicațiilor şi corectarea acestora încă timpul fazelor de dezvoltare se realizează printr-un proces de testare specific, bine planificat şi organizat, conduce la asigurarea încrederii că aplicația are un nivel de securitate ridicat.

După scopul atacurile informatice, în [GONG03] acestea sunt clasificate în atacuri pentru: obținerea şi accesul la informații confidențiale; modificarea integrități datelor; oprirea funcționalității sistemului.

Testarea software este procesul de identificare a erorilor în programe. Testarea securității aplicațiilor informatice are în vedere identificarea breşelor de securitate existente în aplicațiile supuse testării. Testarea securității aplicațiilor informatice, se realizează atât în faza de dezvoltare cât şi în cea de funcționare a aplicațiilor.

Testarea securității aplicațiilor se realizează atât la nivelul codului sursă, în cazul în care este disponibil şi se doreşte acest lucru, precum şi rulând forma executabilă a aplicațiilor.

În cazul în care atacul utilizat într-un test este încheiat cu succes (aplicația permite accesul, deşi nu ar fi trebuit) se consideră că aplicația este nesigură la astfel de atacuri. În caz contrar, aplicația este considerată sigură, ținând cont de condițiile realizării atacului. Succesul unui atac de tip *i* este considerat un eşec al securității sistemului şi invers, un eşec al atacului de tip *i* se consideră un succes din punct de vedere al securității sistemului testat.

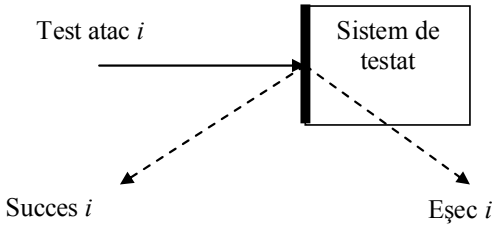

Fig.1. Testarea securității aplicațiilor informatice

Printre erorile introduse de programatori în codul sursă, intenționat sau nu, se remarcă următoarele categorii:

- erori de logică
- erori legate de neverificarea limitelor (zone de memorie, masive, şiruri etc.)
- erori la alocarea memorie
- utilizarea de secvențe de cod pentru accesul la aplicație în faza de testare şi depanare şi neeliminarea acestora în varianta finală
- utilizarea de secvențe de program pentru accesul ulterior la aplicație

Testarea securității aplicațiilor informatice urmăreşte să identifice problemele legate de:

- accesul neautorizat la un sistem sau o aplicație;
- modificarea datelor;
- vizualizarea şi accesul la informații confidențiale;
- obținerea de privilegii;
- împiedicare a altor utilizatori să folosească aplicația.

De asemenea sunt avute în vedere şi posibilităţile aplicaţiei, ca în cazul apariţiei unei tentative de acces neautorizat, de monitorizare şi înregistrare a tuturor accesurilor, în scopul eliminării ulterioare a problemei şi identificării sursei neautorizate.
Testarea securităţii aplicaţiilor informatice se urmând strategii de tipul *white-box* şi *black-box*. Pe lângă aceste strategii, cu metodele lor specifice, sunt recomandate şi analizele şi parcurgerile codului sursă.

Vulnerabilitatea programelor
Probleme de securitate ale programelor apar datorită codului rău intenţionat sau datorită erorilor introduse neintenţionat de dezvoltatorii aplicaţiei. Există următoarele tipuri de cod scris rău intenţionat [PFLE02]:

- *Virus* – programe care se propagă ataşându-se la fişiere executabile;
- *Cal troian* – Programe care conţin funcţii adiţionale scopului declarat al acestora;
- *Bombă logică* – Cod care acţionează la îndeplinirea unei condiţii logice;
- *Bombă cu ceas* – Cod care acţionează la un moment dat în timp;
- *Trapdoor* – Cod care permite accesul neautorizat la program;
- *Vierme* – Cod care se propagă prin copierea în reţea;
- *Iepure* – Cod care se propagă în scopul epuizării resurselor sistemelor de calcul care au fost infectate;

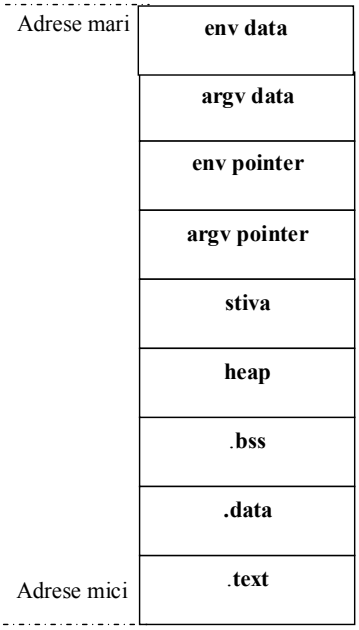

Fig. 2. Structura memoriei programelor

Una dintre cele mai importante probleme o reprezintă cea legată de depăşirea anumitor zone speciale de memorie, acest lucru ducând la posibilitatea de a executa alt cod decât cel al programului. Această categorie de erori este denumită generic *buffer overflow*. Exploatarea acestor breşe de securitate, generate de neefectuarea de verificări, se realizează cunoscând structura memoriei programelor în momentul execuţiei acestora.
În figura 2 este prezentată structura memoriei programelor care se execută. Semnificaţia zonelor de memorie este prezentată în tabelul 1.

Tabelul 1. Segmentele de memorie ale programelor

Segment	Semnificaţie
.text	Codul executabil al procesului
.data	Data iniţializate
.bss	Date neiniţializate
heap	Memorie iniţializată dinamic
env	Variabilele de mediul şi adresa acestei zone
argv	Argumentele din linia de comandă şi adresa şirului corespunzător acestora

Prin atacurile de acest tip, secvenţele de cod rău intenţionat scrise direct în limbaj maşină, sunt inserate şi executate de către programul care nu prezintă protecţie la astfel de atacuri. Pentru determinarea zonei de memorie care este alocată stivei ce conţine adresa următoarei instrucţiuni executabile (valoarea salvată a registrului EIP), persoanele care exploatează aceste programe în diferite scopuri, efectuează o serie de încercări.
În această de categorie de atacuri intră şi cele de depăşire a memoriei dinamice (heap), precum şi a limitelor masivelor.

Parcurgerea şi analiza codului sursă

Această strategie de testare necesită accesul la codul sursă şi la structura programului şi pune accentul pe acoperirea prin teste a căilor, ramificaţiilor şi fluxurilor programului. Principalele metode de testare structurală au în vedere gradul în care cazurile de test acoperă sau execută codul sursă al programului.

Strategiile de testare bazate pe căi utilizează fluxul de control al programului. Acestea reprezintă o familie de tehnici de testare bazate pe selectarea cu atenţie a unei mulţimi de căi din program. Dacă mulţimea căilor este aleasă corespunzător, atunci se va atinge o anumită măsură a profunzimii testului. Pentru utilizarea acestor tehnici este necesară cunoaşterea completă a structurii programului şi accesul la codul sursă.

Graful asociat programului este o reprezentare grafică a structurii de control al programului, care utilizează elemente ca proces, decizie şi joncţiune.

Un *bloc de bază* este o secvenţă de instrucţiuni ale programului, neîntrerupte de decizii sau de joncţiuni.

Un *proces* are o singură intrare şi o singură ieşire şi constă dintr-o singură declaraţie/instrucţiune, dintr-o secvenţă de declaraţii/instrucţiuni, un singur apel de subrutină sau o secvenţă din acestea.

O *decizie* este un punct al programului în care fluxul de control continuă pe una din alternativele oferite. Construcţiile *if-else* şi *switch-case* sunt exemple de decizii cu două ramuri, respectiv cu n ramuri, n≥2.

Joncţiunile sunt punctele din program în care fluxul de control se uneşte, care pot fi etichetele ţintă ale unui salt, punctele în care se unesc ramurile unei instrucţiuni *if-else* etc.

O *cale* într-un program este o secvenţă de instrucţiuni care începe cu o intrare, joncţiune sau decizie şi se termină la altă (sau aceeaşi) joncţiune, decizie sau ieşire. O cale poate să treacă prin câteva joncţiuni, procese sau decizii o dată sau de mai multe ori. Numele căii este dat de numele nodurilor de-a lungul căii. Nodurile sunt numerotate corespunzător fluxului secvenţei de program. Un *segment* constă dintr-o succesiune de legături consecutive care aparţin aceleiaşi căi.

Lungimea unei căi este dată de numărul de legături şi nu de numărul de declaraţii sau instrucţiuni executate de-a lungul căii. O metodă alternativă de măsurare a lungimii unei căi este numărul de noduri traversate. Dacă programul are un singur nod de intrare şi un singur nod de ieşire, numărul de legături traversate este mai mic cu unu decât numărul nodurilor traversate.

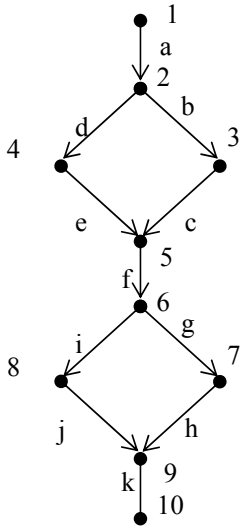

Fig. 3. Nodurile şi legăturile grafului asociat unui secvenţe de program

O cale de la punctul de intrare până la ieşire se notează de exemplu cu *1-2-3-5-6-7-9-10*. În mod asemănător, dacă se notează legăturile, numele căii este dat de succesiunea numelor legăturilor de-a lungul căii. Pentru aceeaşi cale, numele ei este *abcfghk*. În practică, o cale de test este o cale care începe în punctul de intrare în program şi se termină la ieşirea din program (figura 3).

Tehnicile de testare bazate pe căile programului au la bază o serie de criterii de acoperire a codului care se testează precum: acoperirea instrucţiunilor, acoperirea ramificaţiilor, acoperirea condiţiilor, acoperirea ramificaţiilor şi a condiţiilor, acoperirea condiţiilor multiple şi acoperirea tuturor căilor programului. Pe baza acestor criterii de acoperire a codului se determină seturile de date de test care se utilizează în testările structurale corespunzătoare: testarea instrucţiunilor, testarea ramificaţiilor, testarea căilor etc.

Testarea fluxului datelor utilizează graful fluxului de control pentru a studia anomaliile fluxului de date. Acest tip de testare formea-

ză o familie de strategii de test bazate pe selecţia unei căi din fluxul de control al programului în scopul cercetării secvenţelor de evenimente referitoare la starea datelor.
Testarea fluxului de date se bazează pe faptul că cel puţin jumătate din codul sursă constă în declaraţii de date, declaraţii care definesc structuri de date, obiecte individuale, valori iniţiale (sau implicite) şi atribuite [BEIZ90], [PETE00].
Graful fluxului datelor este alcătuit din noduri şi legături direcţionate. Obiectivul său este acela de a prezenta deviaţiile de la fluxul de date implementat faţă de ceea ce s-a dorit la proiectare.
Acţiunile posibil de efectuat asupra datelor sunt: definire (**d**); este explicită, când variabila apare într-o declaraţie de date şi implicită, când variabila este într-o instrucţiune de atribuire; nedefinire (**k**); atunci când data nu mai este disponibilă sau când conţinutul ei nu este cu certitudine cunoscut; utilizare (**u**) - în calcule (**c**); variabila apare în partea dreaptă a unei atribuiri, într-o expresie de calcul sau într-un predicat (**p**); când apare direct, de exemplu `if (a<b)`, sau în condiţia de ciclare a unei bucle.
Anomaliile sunt reprezentate prin secvenţe de două acţiuni:
- dd – variabilă definită, dar nereferită;
- ku – variabilă nedefinită, dar referită;
- dk – variabilă definită, dar nereferită.

Pentru utilizarea acestor tehnici este necesar accesul la codul sursă şi de asemenea timpul alocat procesului de testare să fie corespunzător, astfel încât să permită efectuarea tuturor parcurgerilor de cod necesare.

Testarea de penetrare

Testarea securităţii aplicaţiilor informatice constă dintr-o serie de metode şi tehnici de testare. Printre acestea sunt incluse:
- evaluarea configuraţiei;
- testarea funcţională;
- scanarea vulnerabilităţii;
- testarea de penetrare;
- accesul neautorizat etic.

Testarea de penetrare conduce la identificarea unor vulnerabilităţi ale sistemelor precum: servicii vulnerabile, politici de securitate ineficiente, probleme de configurare. Se utilizează pentru diferite aspecte ale reţelelor precum diferite topologii de reţele (dial-up, LAN, WAN), firwewall-uri, sisteme de operare şi aplicaţii.
Testarea de penetrare din exterior vizează identificarea punctelor slabe ale reţelelor din afara acestora. Se realizează având sau nu informaţii despre modul în care este configurată reţeaua.
Testarea de penetrare din interior are în vedere identificarea punctelor slabe ale reţelelor prin analiza internă a configuraţiei acestora şi prin efectuarea de teste diferite.
Testarea de penetrare se realizează manual sau automatizat.
Printre instrumente utilizate în testarea de penetrare sunt enumerate următoarele categorii de programe:
- de interogare DNS (*nslookup*);
- de scanare a adreselor de reţea şi a porturilor deschise;
- de spargere a parolelor;
- de filtrare a pachetelor în reţea;
- navigatoare si programe depanare la distanţă;
- pentru identificarea sistemului de operare;
- utilitare (*whois*, *tracert*, *ping*);

Pentru rezultate cât mai bune se recomandă utilizarea atât a testării manuale cât şi a celei automate, fiecare depistând tipuri specifice de probleme. De asemenea, se va utiliza atât testarea de penetrare internă cât şi cea externă.

Concluzii

Prin proiectarea şi implementarea unui proces de testare a securităţii aplicaţiilor informatice adecvat, creşte încrederea în utilizarea acestor aplicaţii. Testarea securităţii aplicaţiilor informatice garantează în totalitate securitatea acestora, efectuarea periodică a testelor şi precum şi actualizarea acestora, având în vederea evoluţia continuă a tehnologiilor informatice.
Pentru aplicaţii informatice sigure din punct de vedere al securităţii trebuie pornit de la procesul de dezvoltare software, când anumite standarde şi proceduri trebuie utilizate în scrierea de aplicaţii. La garanţia securităţii

aplicațiilor informatice concură factori umani, procese şi tehnologie, toți aceşti factori trebuind testați pentru atingerea tuturor obiectivelor propuse.

Bibliografie

[ALBE02] Christopher Alberts, Audrey Dorofee – *Managing Information Security Risks: The OCTAVE^{SM} Approach*, Addison Wesley, 2002

[BEIZ90] Beizer, Boris – *Software Testing Techniques – Second Edition*, Van Nostrad Reinhold, New York, 1990

[BHAS03] Shweta Bhasin – *Web Security Basics*, Premier Press, 2003

[CRAI02] Rick D. Craig and Stefan P. Jaskiel, *Systematic Software Testing*, Artech House 2002, ISBN 1580535089

[GRAF03] Mark G. Graff, Kenneth R. van Wyk, *Secure Coding: Principles & Practices*, O'Reilly, 2003, ISBN 0-596-00242-4

[HOWA02] Michael Howard, David LeBlanc, *Writing Secure Code*, Microsoft Press, 2002, ISBN 0-7356-1588-8

[HUTC03] Marnie L. Hutcheson, *Software Testing Fundamentals: Methods and Metrics*, John Wiley & Sons, 2003, 047143020X

[KLEV02] T. J. Klevinsky, Scott Laliberte, Ajay Gupta – *Hack I.T.: Security through Penetration Testing*, Addison Wesley, 2002

[GONG03] Li Gong, Gary Ellison, Mary-Dageforde – *Inside Java™ 2 Platform Security: Architecture, API Design, and Implementation, Second Edition,* Addison Wesley, 2003

[LOCK04] Andrew Lockhart **–** *Network Security Hacks***,** O'Reilly, 2004

[PETE00] Peters, James F., Pedrycz, Witold – *Software Engineering – An Engineering Approach*, John Wiley & Sons, Inc, 2000

[PFLE02] Charles P. Pfleeger, Shari Lawrence Pfleeger **–** *Security in Computing, Third Edition,* Prentice Hall PTR, 2002

[PIPK02] Donald L. Pipkin, *Halting the Hacker: A Practical Guide to Computer Security*, Second Edition, Prentice Hall PTR, 2002, ISBN 0-13-046416-3

[POCA04] Paul Pocatilu – Costurile Testării Software, Editura ASE, Bucureşti, 2004

[SCHI03] Mike D. Schiffman – Building Open Source Network Security Tools: Components and Techniques, John Wiley & Sons, 2003, ISBN 0471205443

[TULL03] Mitch Tulloch, Microsoft Encyclopedia of Security, Microsoft Press, 2003