The security imperative in application programming

Prep. Iulian ILIE-NEMEDI
Catedra de Informatică Economică, A.S.E. Bucureşti

Security is vital in writing code. Security concerns detection and reaction as much as protection. It is impossible to have a totally secure system. Hackers often know more about systems than the people who design them. One of the most effective ways to ward off unauthorized access is to run at the least privilege settings and not use sysadmin status. The rules are simple. Use all defense. Be evil, test evil. Do not trust user input. Keep the attackers guessing. Do not give away free information. Use defense in depth. Applications should be secure by design, default and deployment. Reduce the attack surface of your application by moving native C or C++ code to managed C# code.
***Keywords**: Security, Cryptography, Application Design, Hacking, Authentication, Authorization, SQL Injection, Code Red, Cross-site Scripting.*

Securitatea este un aspect vital al programării aplicaţiilor. Atacatorii ştiu adesea mai multe despre sistemele atacate decât cei care le-au proiectat şi implementat. De aceea unul din cele mai eficiente mecanisme de protecţie în faţa accesului neautorizat este execuţia codului cu nivelul minim de privilegii necesar. Automobilele din ziua de azi au închidere automată, sistem de alarmă şi dispozitiv de urmărire astfel încât dacă sunt furate poliţia să-l poată repera pe hoţ în cel mai scurt timp. Ar trebui să aveţi în vedere să vă protejaţi aplicaţiile într-o manieră asemănătoare. Articolul de faţă este o trecere în revistă a aspectelor de care dezvoltătorii trebuie să ţină seama la construirea aplicaţiilor securizate pe platforma Windows. În prefaţa cărţii sale *Secrete şi minciuni*, Bruce Schneier oferă un motiv pentru scrierea ei şi anume corectarea greşelilor precedentei sale cărţi, *Criptografia aplicată*. Acest lucru poate surprinde de vreme ce *Criptografia aplicată* a fost şi este una din lucrările de referinţă din domeniu. El consideră că deşi modelele matematice înglobate în teoria criptografiei sunt deosebite, acest lucru în sine nu este o garanţie a securităţii. Cum este posibil asta? Deoarece aplicaţiile sunt construite în jurul platformelor hardware, software, de comunicaţie în reţea şi nu în ultimul rând sunt centrate pe utilizatori. Breşele de securitate ale acestora fac perfecţiunea matematică a criptografiei irelevantă. Sistemele din ziua de azi sunt atât de complexe încât este imposibil să fie absolut sigure. *Dacă sunteţi de părere că tehnologia vă poate rezolva problemele de securitate, atunci nu înţelegeţi natura problemei şi nu înţelegeţi nici tehnologia*, afirmă Schneier. Ca programator te poţi numai baza pe tehnologie pentru a face aplicaţia mai sigură. Pentru a rezolva problema securizării aplicaţiilor trebuie să fim conştienţi de toate aspectele care afectează siguranţa soft-ului şi a mediilor în care acesta rulează. Apoi putem încerca să le îmbunătăţim. În particular, trebuie să verificăm codul scris pentru eliminarea scăpărilor, atât a celor de implementare cât mai ales a celor de proiectare.

Securizarea comunicaţiei

Securizarea comunicaţiei presupune protejarea confidenţialităţii şi integrităţii datelor transmise. Printre alternativele pentru asigurarea comunicaţiilor sigure se numără tehnologiile IPSec, SSL/TLS precum şi RPC Encryption, utilizată de DCOM. IPSec este o tehnologie de criptare la nivel de pachete, pachetele IP fiind criptate şi apoi semnate. Avantajul acesteia constă în faptul că este localizată la nivelul platformei. Programatorul nu trebuie să facă nici o acţiune, sarcina configurării procesului revenindu-i administratorului de sistem. Tehnologia SSL funcţionează similar tehnologiei IPSec, fiind utilizată în comunicaţia dintre clienţii de tip browser şi server-ele Web precum şi pentru securizarea

apelurilor serviciilor Web. SSL autentifică întotdeauna serverul opțional putând autentifica şi clienții prin intermediul certificatelor digitale. Atât browser-ele cât şi serverul IIS au suport SSL încorporat, deci din nou programatorii nu trebuie să-şi facă griji în privința instalării. Adesea administratorului sistemului îi revine sarcina de a obține câte un certificat pentru site-urile găzduite sau poate apela la servicii de certificare prin care să solicite fiecărui client să se autentifice în Intranet. Tehnologia TSL (Transport Level Security) este în esență SSL 3.1 încorporată în standardul IETF (Internet Engineering Task Force).

Gestiunea identității

Autentificarea şi autorizarea sunt adesea confundate una cu cealaltă. Autentificarea este procesul de verificare a asocierii dintre utilizator şi identitatea pretinsă de acesta prin intermediul unui împuterniciri. Împuternicirea poate consta într-un obiect, de exemplu o cartelă magnetică, o informație cunoscută de utilizator, precum un cont şi o parolă sau un identificator fizic, de exemplu amprenta. Metodele de autentificare software includ autentificarea de bază, paşaport Microsoft sau Kerberos. Odată stabilită identitatea utilizatorului prin autentificare, autorizarea este utilizată pentru permiterea accesului la anumite date sau pentru executarea anumitor metode, de exemplu un ACL (Access Control List) poate stabili dacă un utilizator are drept de scriere asupra unui fişier. La nivel de cod sursă, verificarea accesului la resursele programului se poate realiza în două moduri: imperativ sau declarativ.

Criptarea datelor

Chiar dacă algoritmii de criptare pot fi extrem de puternici, ei se dovedesc eficienți securizării aplicației în funcție de maniera în care sunt utilizați. De aceea un algoritm de criptare care şi-a dovedit eficacitatea este de preferat unuia încă insuficient testat şi cu atât mai mult este preferabil unui algoritm propriu. Pe platforma Windows asta se traduce prin utilizarea interfețelor CryptoAPI, CAPICOM sau a pachetului System.Security. Cryptography din .NET. Cheile de criptarea sunt adesea obținute din parole, astfel încât este necesară utilizarea parolelor puternice şi a vectorilor de inițializare pentru a da robustețe cheilor de criptare. Deşi vectorii de inițializare nu sunt secreți, ei putând fi ataşați mesajului transmis, menirea lor este să introducă o componentă aleatoare în textul criptat, în aşa fel încât chiar dacă textul ce urmează a fi criptat este identic pentru două mesaje diferite, rezultatul criptării acestora să fie diferit, împiedicându-l pe atacator să extragă conținutul mesajului din secvențele repetitive ale mesajelor criptate. Nu în ultimul rând țineți seamă de faptul că un atacator nici măcar nu trebuie să spargă algoritmul de criptare dacă parolele utilizate sunt stocate într-un mod nesigur. De aceea se recomandă stocarea informațiilor cu caracter secret în sisteme separate, inaccesibile pe cât posibil utilizatorilor aplicației, utilizând ceva similar interfeței DAPI, protejarea cheilor de criptare fiind vitală pentru succesul criptării datelor.

Erorile de tip *buffer overruns* şi validarea intrărilor

Erorile de tip buffer overruns constituie vulnerabilitatea cea mai comună şi mai serioasă în securitatea aplicațiilor. Originea lor rezidă într-o problemă a codului C şi C++ care se produce atunci când datele introduse depăşesc dimensiunea aşteptată, rescriind alte valori din program, fiind posibilă în anumite condiții chiar rescrierea secvențelor executabile ale aplicației gazdă. Acest tip de eroare este cunoscută sub numele de *Cod Red*. Prin exploatarea sa un hacker poate cauza o violare a drepturilor de acces a aplicației la resursele sistemului, în special de memorie, poate crea instabilitate în execuția aplicației sau în cel mai rău caz, poate determina execuția unor secvențe arbitrare de cod cu toate privilegiile aferente aplicației atacate, în cazul *Cod Red* contul fiind SYSTEM. Iată de ce este vitală execuția codului cu minimul de privilegii solicitate, IIS 6.0 executându-se ca NetworkService, care are asociat un nivel scăzut de privilegii. Exemplul amintit anterior se referă la *stack overrun*, dar mai trebuiesc amintite şi alte forme de rescriere cum

sunt *heap overrun*, rescrierea tabelei virtuale de funcții (*v-table overwites*), rescrierea pointer-ilor la funcții sau a funcțiilor de tratarea a excepțiilor (*exception handlers overwites*). O altă modalitate de corupere şi rescriere a codului are la bază şirul de format din C:

```
prinf("%s",sirUtilizator);
```

Şirul de format permite atacatorului să introducă specificatori de format, inclusiv %n care scrie pe stivă. Ceea ce uimeşte însă în privința programării aplicațiilor în C este insecuritatea moştenită de la funcții foarte des folosite, preferându-se utilizarea funcțiilor precum *strcpy*, *strcat*, *memcpy*, *sprintf*, în locul versiunilor sigure ale acestora, reunite în *strsafe.h*.

Comenzile cross-site (XSS)

Mecanismul cross-site scripting permit atacatorului să execute comenzi proprii în browser-ul clientului. Dacă într-o pagină Web tot ceea ce este introdus de utilizator este întors ca ieşire, atunci site-ul este vulnerabil, indiferent ce tehnologie utilizează. Atacatorul poate încadra tag-uri de script sau form-uri în hiperlink-uri legitime. Atributul *action* al formularelor poate fi astfel setat la un script aflat pe site-ul atacatorului către care se duc informații private introduse de utilizator. Un exemplu trivial, dar cu mare incidență practică, este legat de clienții Web de e-mail. Dacă un client Web de e-mail permite cross-site scripting, un hacker poate trimite un e-mail care atunci când este deschis face redirectare în pagina curentă la o pagină publicată pe site-ul său similară ca desing paginii de autentificare a clientului de e-mail. Utilizatorul autentificat deja pe site-ul original poate să se reautentifice din neatenție utilizând această pagină, care redirectează cererea la autentificatorul clientului de e-mail, nu înainte de a salva datele de autentificare într-o bază de date a hacker-ului. În acest mod eminamente psihologic, atacatorul obține informații confidențiale prin neglijența utilizatorului, fără să spargă sistemul de securitate al aplicației de e-mail, exploatând doar mecanismul cross-site scripting.

Injecția de cod SQL şi securizarea accesului la date

Injecția de cod SQL se produce atunci când programatorul oferă acces utilizatorilor la criteriile cererilor SQL ceea ce permite atacatorilor să introducă valori care modifică intenția inițială a instrucțiunilor SQL. Încă o dată se impune să nu aveți încredere în datele introduse de utilizatori. Iată un exemplu în PHP:

```
$query = "select * from users where
cont='" . $_GET["cont"] . "' and pa-
  rola='" . $_GET["parola"] . "'";
```

Ce se întâmplă dacă utilizatorul introduce următoarele informații în câmpul *cont*:

```
1. administrator' --
2. administrator' or 1=1; drop table
   utilizatori --
3. administrator' or 1=1; execute
   master.xp_cmdshell --
```

Răspunsuri:

1. Parola este evitată, -- este comentariu în SQL.
2. Sunt întoarse toate înregistrările din tabelă, 1=1 este întotdeauna adevărat. Tabela utilizatori este ştearsă.
3. Pe SQL Server este lansată o fereastră de comenzi. Dacă serverul de baze de date rulează ca SYSTEM, atunci atacatorul devine proprietar al consolei maşinii server.

Există mai multe metode prin care se poate evita exploatarea criteriilor SQL. Cu toate acestea prima problemă este evitarea dezvăluirii oricărei informații despre baza de date. În mod implicit paginile ASP şi PHP întorc informații despre erorile OLE-DB şi ODBC către utilizator. Prin modificarea parametrilor de intrare, atacatorul poate utiliza aceste informații pentru verificarea bazei de date în scopul descoperirii numelor de tabele, coloane şi tipurile de date corespunzătoare, precum şi despre valorile înregistrărilor conținute în acestea. Revenind la problema injecției de cod SQL, există două moduri prin care aceasta poate fi evitată: prin citarea datelor introduse şi prin utilizarea procedurilor stocate. Prima metodă dublează fiecare apostrof din datele introduse de utilizator înainte de a executa cererea rezultat. În acest mod dacă atacatorul încearcă să introducă un apostrof este generată o cerere SQL invalidă care nu va putea fi executată. Această tehnică nu este

sigură din două motive. Mai întâi dacă există variabile de intrare care acceptă tipuri SQL altele decât string, atunci atacatorul nu trebuie să delimiteze prin apostrof datele introduse. În al doilea rând ar putea folosi valoarea hexa 0x27 pentru apostrof, ceea ce complică dublarea acestuia. Procedurile stocate ajută la eliminarea vulnerabilității față de clauzele de reuniune de tip *or*, dar nu şi față de adăugarea de noi instrucțiuni în clauzele SQL precum *insert* sau *update*. Ceea ce este cu adevărat eficient este pregătirea în prealabil a cererilor pentru execuție prin adăugarea parametrilor criteriilor cu ajutorul construcțiilor specializate de tip *PreparedStatement*. Nu în ultimul rând cea mai bună protecție este execuția cererilor SQL cu nivelul de acces cel mai scăzut solicitat. În condițiile date, securitatea integrată a aplicației este de preferat securizării SQL, utilizându-se un cont care are exact privilegiile cerute de rolul utilizatorului şi nimic mai mult. Suprafața de atac SQL poate fi redusă prin utilizarea cererilor parametrizate cu control direct asupra tipului SQL al parametrilor. Exemplu PHP de mai sus s-ar rescrie securizat în Java astfel:

```
String query = "select * from utili-
zatori where cont=? and parola=?";
Java.sql.PreparedStatement statement
=
connection.prepareStatement(query);
statement.setString(1,requert.getPar
ameter("cont"));
statement.setString(2,request.getPar
ameter("parola"));
```

Concluzii

Aplicațiile trebuie securizate prin proiectare, implementare şi publicare. Sper că v-am oferit câteva exemple vizând modul în care aplicațiile dumneavoastră pot fi vulnerabile precum şi modalitățile prin care puteți preveni atacurile. Cu toate acestea, construirea unui sistem sigur nu-l face totalmente invulnerabil. Acest lucru este pur şi simplu imposibil pentru că securitatea deplină ar necesita închiderea sistemului în fața oricărei încercări de pătrundere din exterior, inclusiv a utilizatorilor de drept ai acestuia. Trebuie să avem mereu în vedere accesul utilizatorilor şi proceselor la aplicația noastră. De aceea securitatea vizează mereu detecția şi reacția adecvată a aplicației la atacurile externe pentru a contracara viteza atacurilor prin viteza de închidere sigură a aplicației în fața acestora.

Recomandări privind scrierea de cod sigur:

1.Securitatea este un imperativ al proiectării: gândiți-vă şi țineți seama de mecanismele de securitate încă din prima zi a construirii aplicației.

2.Executați modulele aplicației cu minimul de privilegii necesare.

3.Reduceți suprafața de atac a aplicației dumneavoastră.

4.Nu aveți încredere în datele introduse de utilizatorul sistemului.

5.Lăsați-l mereu pe atacator să ghicească: nu expuneți gratuit informații despre structura aplicației.

6.Folosiți mecanisme de apărare în adâncime, până la baza sistemului.

7.Testați fiecare breşă de securitate. Gândiți destructiv, acționați destructiv, testați destructiv sau apelați la un expert în materie.

8.Înlocuiți, pe cât posibil, codul cu execuție nativă pe platformă, de exemplu cod nesigur C sau C++, cu cod executat pe maşina virtuală, de exemplu C#.

Bibliografie

1. Bruce Schneier, *Applied Cryptography*, Second Edition, Wiley Computer Publishing, John Wiley & Sons, Inc., January 1996

2. Watling Nigel, *Security*, Code Zone Magazine, Microsoft Press, April 2004