MATEMATICĂ ŞI INFORMATICĂ

CREAREA UNUI GHID PENTRU TESTAREA APLICAŢIILOR. IMPLEMENTAREA ÎN JAVA

Anastasia BELEVȚOVA

CZU: 004.421 avnxce@gmail.com

The increasing number of Web applications leads to the idea that they should be tested meticulously. Efficient testing is possible with test automation. The present work describes testing types, methods, and steps of implementing an automated testing framework using Java programming language, up-to-date open-source libraries, and tools. The web application under test performs encryption and decryption with the ElGamal algorithm.

Testarea constă dintr-o serie de etape: Analiza cerințelor, Planificarea, Dezvoltarea cazurilor de testare, Configurarea mediului de testare, Executarea testelor, Închiderea ciclului de testare. Fiecare din aceste etape include activități specifice. Analiza cerințelor include analiza logicii de afacere, documentarea cerințelor, identificarea tipurilor de teste care trebuie efectuate, se stabilesc prioritățile și direcția de focusare a efortului de testare. Planificarea testării presupune pregătirea planului de testare, selectarea tehnologiilor și instrumentelor de testare, estimarea efortului de testare necesar. Ulterior se definesc cazurile de testare, se setează mediul de testare și se execută testele. La etapa închiderii ciclului de testare are loc raportarea calității produsului și analiza rezultatelor testelor.

Necesitatea efectuării unei testări meticuloase a produselor software este subliniată de multiple cazuri cunoscute a testării insuficiente. În continuare sunt prezentate câteva exemple: a avut loc furtul datelor utilizatorilor poștei electronice Yahoo după care peste 500 de milioane de utilizatori au fost nevoiți să reînnoiască parolele, peste 135 000 de automobile Tesla au fost rechemate pentru remedierea problemelor de memorie a calculatorului, au fost detectate erori în timpul testului navei cosmice CST-100 Starliner, care au fost corectate la timp.

Termenul sistem sub testare (System Under Test, SUT) se referă la un sistem care este testat pentru a valida funcționarea corectă. În această lucrare sistemul sub testare este a aplicație Web pentru criptare și decriptare a valorilor prin algoritmul ElGamal. Astfel corectitudinea decifrării unei valori este un criteriu elocvent al funcționării corecte a aplicației. Descrierea algoritmului este prezentată în continuare.

Algoritmul ElGamal se bazează pe problema logaritmului discret, care este următoarea: Fie p număr prim și $\alpha, \beta \in Zp, \beta \neq 0$. Să se determine $a \in Z_{p-1}$ astfel ca $\alpha^a \equiv \beta \ (mod\ p)$. Acest întreg a – dacă există – este unic și se notează $log_\alpha \beta$. Algoritmul ElGamal are multiple aplicații, dar realizarea în aplicația analizată este precum urmează.

Are loc generarea cheilor:

1. Se alege un număr p – un număr prim.
2. Se alege α - rădăcină primitivă de ordinul $p - 1$ a unității:

$$\forall i\ (0 < i < p - 1),\ \alpha i \not\equiv 1\ (mod\ p).$$

3. Se alege a – cheie privată.
4. Se generează $\beta : \beta \equiv \alpha^a\ (mod\ p)$

Valorile p, α, β sunt publice, constituind cheie publică, dar a este valoare secretă și constituie prima cheie privată. Aceasta este cheia primului utilizator.

Criptarea are loc în modul următor:

1. Se definește valoarea $k \in Z_{p-1}$ aleator și secret. Aceasta este a doua cheie privată, altfel spus cheia celui de-al doilea utilizator.
2. Se criptează mesajul x – un număr întreg, x. Se obține mesaj criptat din două valori y_1 și y_2:

$$y_1 = \alpha^k\ (mod\ p)$$
$$y_2 = x \cdot \beta^k\ (mod\ p)$$

Decriptarea are loc în modul următor:

Se calculează:

$$m = y_2 \cdot (y_1^a)^{-1}\ (mod\ p)$$

Deoarece aplicația testată este o aplicație Web, trebuie de atras atenție la următoarele caracteristici: O aplicație Web poate fi accesată concomitent de un număr mare de utilizatori distribuiți prin toată lumea; Se rulează pe medii de execuție eterogene, compuse din diferite elemente hardware și software; Este capabilă să genereze componente software în timpul rulării.

Din aspectele enumărate rezultă cazuri adiționale necesare de a fi testate. Trebuie să fie identificate soluții eficiente pentru executarea testelor de performanță și disponibilitate pentru a verifica comportamentul aplicației web atunci când este accesată concomitent de un număr mare de utilizatori. Aplicațiile Web trebuie testate pentru a ne asigura că comportamentul aplicației este cel așteptat folosind diferite browsere web și sisteme de operare. O altă caracteristică a unei aplicații Web care trebuie testată este securitatea și capacitatea de a fi protejată de accesul neautorizat.

Partea practică a lucrării cuprinde descrierea implementării unui proiect de testare a aplicației Web. Testele automatizate simulează acțiunile utilizatorului și validează corectitudinea funcționării aplicației. Pentru a ușura compunerea testelor funcționalitatea aplicației poate fi prezentată ca diagrama cazurilor de utilizare compusă pe baza algoritmului descris anterior, unde fiecare etapă e o acțiune a utilizatorului aplicației. Adițional utilizatorul poate genera toate rădăcinile primitive existente la etapa de generare a cheilor.

Necesitatea automatizării trebuie să fie analizată în dependență de specificul sistemului sub testare. Efortul inițial este mai semnificativ la automatizarea testelor: se necesită mai mult timp pentru automatizare de cât pentru verificarea manuală a unui caz testat. Însă la fiecare rulare repetată a testelor automatizate efortul va fi mai mic decât

cel al testării manuale. Astfel se întoarce efortul și resursele investite în automatizare. Deci, automatizarea testării este convenabilă în cazul că va avea loc rularea periodică a testelor automatizate. Din alt punct de vedere, testarea manuală este de neînlocuit în testarea exploratorie, a utilizabilității, ad-hoc. Testarea de performanță nu este posibilă manual, deoarece de regulă trebuie de simulat accesarea aplicației de un număr prea mare de utilizatori. În cazul testării de securitate pot exista multe particularități specifice diverselor aplicații care motivează necesitatea testării manuale. Însă sunt in dezvoltare instrumente de testare a securității care fac posibilă automatizarea testelor. Totodată, testarea statică a codului pentru identificarea vulnerabilităților este o practică comună și oferă posibilitatea securizării dependențelor și codului.

Există multiple tehnologii accesibile gratuit scrise în limbajul Java care pot fi integrate în proiectul de testare. Într-un proiect de testare de tip Maven pot fi adăugate dependențe precum, de exemplu: Cucumber (cucumber-java, cucumber-junit-platform-engine) pentru compunerea testelor in limba de comunicare, de exempu engleza sau româna, Selenium (selenium-java) pentru setarea driver-ului prin Selenium Manager și JavaScriptExecutorului, Junit (junit-jupiter, junit) și Hamcrest (hamcrest-all) pentru validarea corectitudinii rezultatelor. De asemenea se poate adăuga funcționalitate adițională în teste: efectuarea capturilor de ecran cu commons-io, testarea interfeței program a aplicației prin httpclient5, logarea pașilor executați cu log4j-core. Pentru validarea algoritmului criptografic pe baza bibliotecii cu sursă liberă a algoritmilor criptografici e posibil de inclus BouncyCastle (bcprov-jdk18on). De asemenea, completarea suitei de teste cu verificări de securitate și performanță poate fi realizată adăugând dependențe precum OWASP ZAP și Apache JMeter. Testarea statică a codului e posibilă cu Veracode sau Sonar.

Realizarea proiectului dat de testare subliniază faptul că pentru testarea eficientă trebuie de analizat particularitățile aplicației testate și de îmbinat aplicarea diverselor tipuri și tehnici de testare în corespundere cu contextul existent. De asemenea, este o practică bună de urmat etapele unui ciclu de testare, care se vor repeta la fiecare iterație de dezvoltare continuă. Pentru rularea periodică a testelor automatizate este posibilă setarea unei sarcini Jenkins și urmărirea rezultatelor execuției pe baza rapoartelor generate.

Recomandat
***Olga CERBU**, dr., conf. univ.*