KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Mantas Žukas

Interaktyvios saugos sistemos prototipas apsaugai nuo injekcinių atakų

Magistro darbas

Darbo vadovas

doc. dr. J. Toldinas

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Mantas Žukas

Interaktyvios saugos sistemos prototipas apsaugai nuo injekcinių atakų

Magistro darbas

Recenzentas
doc. dr. G. Činčikas
2012-05-

Vadovas
doc. dr. J. Toldinas
2012-05-

Atliko
IFN-0/3 gr. stud.
Mantas Žukas
2012-05-

Kaunas, 2012

Turinys

INTERACTIVE SECURITY SYSTEM PROTOTYPE TO PROTECT AGAINST INJECTIONS ATTACKS

SUMMARY

The most common security flaw in Web applications are injection attacks. These attacks are used for confidential information disclosure, modification or destruction. Every system which is using database and skips the meta-characters validation is vulnerable to injection attack. In this case, the attacker can modify the database query so that the database will return sensitive information. Confidential information may include: login, personal details or other sensitive information.

To prevent from injection attacks all users' input must be validated before further use of inputted data. Then the user input is validated the system can confirm or reject input to avoid the harmful use of this data. There are two main input validation strategies: blacklists and whitelists. These lists are filled with injection attacks features.

In this research the injection attack prevention method is introduced. Also the interactive security system prototype to protect against injections attacks is proposed. Security system prototype is using blacklist input validation strategy for checking input parameters. Each list item consists of a single type of injection attack description. Descriptions are written in Backus–Naur form. Also this list is easy manageable and should be regularly updated. Each type of attack pattern is transformed into a regular expression array. After transformation input is validated using these regular expressions. In the end security method returns the result, if the input is valid or not.

There are a lot of methods how to protect against injection attacks, but the attack scope is very wide. To select the appropriate security method is very important to reduce the vulnerability of these attacks.

ĮVADAS

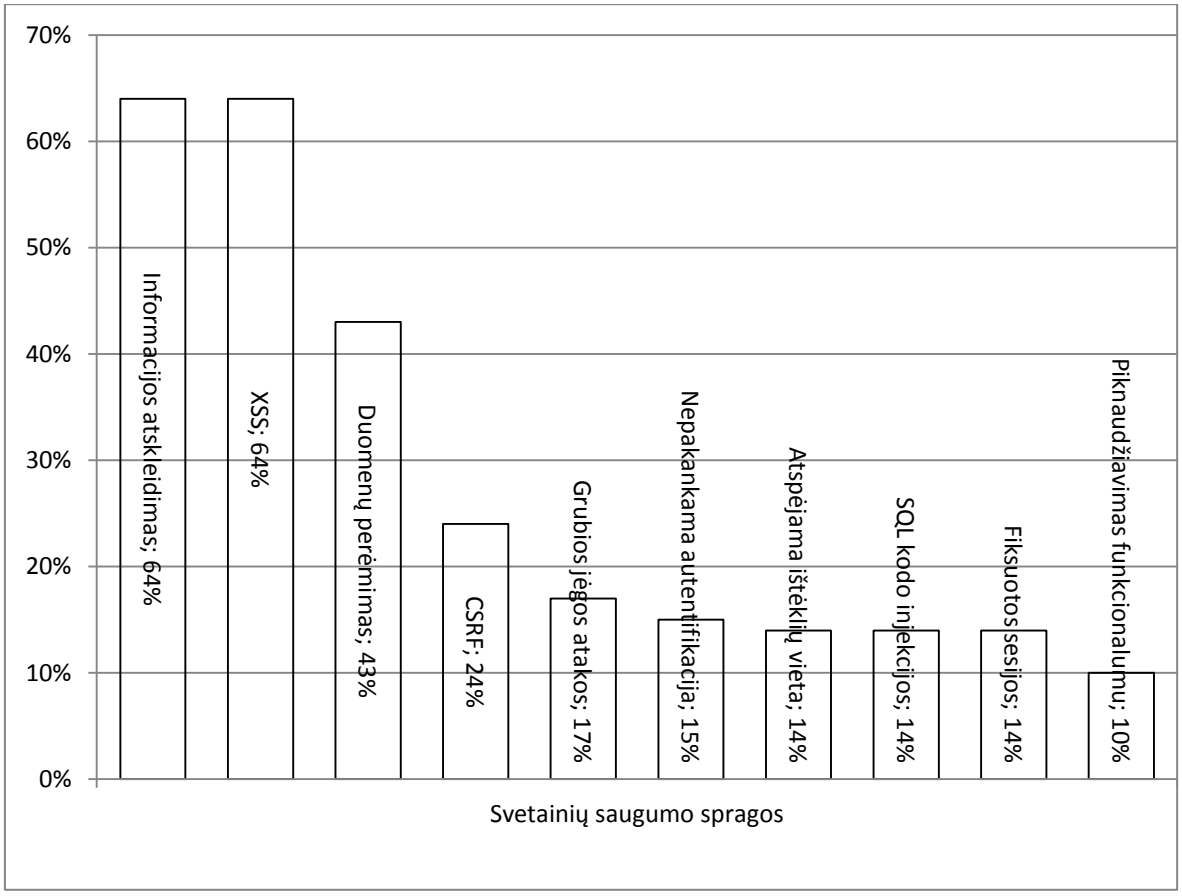
Šiuolaikiniame pasaulyje, kur didžiausia dalis informacijos srautų keliauja internetu, duomenų konfidencialumo, integralumo ir vientisumo sąvokos tampa vis aktualesnės. Tobulėjant programavimo technologijoms plečiasi ir informacijos pažeidžiamumo skalė. Įsilaužimų skaičius taip pat auga: žmogus neturintis didelių programavimo įgūdžių nesudėtingai gali atlikti įsilaužimą, pažeisti duomenis. Informacija tampa brangesnė nei pati įranga, kurioje ji yra patalpinta, todėl vis daugiau dėmesio yra skiriama informacijos saugumui. Nuo 2000 metų iki 2010 metų vartotojų skaičius išaugo 444,8 procentų. 2010 birželio mėnesį buvo registruota 1,9 milijardo interneto naudotojų. Naudotojų skaičius ateityje taip pat didės.

Verslo įmonės sparčiai integravosi į internetą, taip pat atsirado didelės duomenų bazės su daug konfidencialios informacijos, norit apsaugoti informaciją reikia nemažai investicijų. Šiuo metu saugumas nėra taip akcentuojamas kaip funkcionalumas, sistemos kūrėjai visų pirma sukuria sistemą, kuri veiktu, o tik po to ją optimizuoja. Saugumo spragos dažnai būna pastebimos jau išleidus sistemą bei suteikus viešą prieigą prie jos. Šiuo atveju atsiranda rizika, kad galimi duomenų konfidencialumo ar vientisumo pažeidimai. Taip sistemos testavimo bei klaidų taisymo kaštai labai išauga, nes daug pigiau yra taisyti klaidas sistemos kūrimo fazėje. Internetinės sistemos (elektroninė bankininkystė, internetinės parduotuvės ir kita) privalo užtikrinti duomenų saugumą, nes naudotojai nepasitikės sistema bei ja nesinaudos. Šiuo metu galėtumėme aptikti tūkstančius įvairių sistemų, kurios turi tam tikrus pažeidžiamumus. Nėra absoliutaus saugumo yra tik metodai jį pagerinti. Keičiantis technologijomis didėja interneto sparta, atsirado tokia sąvoka kaip programuojamas internetas. Dabar galima sukurti sudėtingą internetinę sistemą, kuriai nebus didelių apribojimų, o reikės tik interneto naršyklės. Vienas iš pavyzdžių gali būti registrų sistemos, kuri yra tiesiog grafinė vartotojo sąsaja su duomenų baze, tam kad būtų galima atlikti visus veiksmus duomenų bazėje. Šiuo metu labai išpopuliarėja socialiniai tinklai. Šiose sistemose informacija yra taip pat pažeidžiama. „Facebook“ sistemoje yra apie 900 mln. registruotų naudotojų. Kiekvieno iš jų asmeninės konfidencialios informacijos saugumas gali būti pažeistas.

Norit daugiau sužinoti apie informacijos saugumą internete, reikia susipažinti su pagrindinėmis atakomis, kurios dažniausiai pasitaiko. Taip pat žinant atakos veikimo principus ir norint nuo jų apsisaugoti. Tikslingos saugumo metodikos pasirinkimas pagerina sistemos saugumą.

1. ŽINIATINKLIO SISTEMŲ PAŽEIDŽIAMUMAI

[21] White Hat Security kompanija, atliekanti svetainių saugumo spragų tyrimus, pateikia visuomenei statistikos ataskaitą. Iš 2011 m. balandžio mėnesio pateiktos ataskaitos matyti, kad viena iš dažniausių tinklapių saugumo spragų yra programinio kodo į vartotojų peržiūrimus tinklapius [21]. Tyrimo duomenys pateikiami išnagrinėjus daugiau kaip 3 tūkst. Svetainių, kurias naudoja įvairaus dydžio komercinės ir valstybinės organizacijos. Duomenys buvo renkami nuo 2006 m. sausio 1 d. iki 2011 m. vasario 16 d.



1 pav. Svetainių saugumo spragos (2011-04-18 duomenys)

Remiantis White Hat Security kompanijos ataskaita, galima teigti, kad daugiausia tinklapių saugumo problemų iškyla dėl jautrios (saugumo prasme) informacijos paviešinimo. Tai susiję su techninės informacijos apie naudojamą programinę ar aparatinę įrangą tinklapio darbui palaikyti ar vartotojų asmeninės informacijos paviešinimu. Tokia informacija gali būti panaudota pažeidžiant tinklapio, jo serverio ar naudotojų saugumą.

[18] Žiniatiklio sistemų rizikos injekcijoms buvo įvardintos kaip pačios pavojingiausios. Rizikos analizė buvo atlikta OWASP nepelno siekiančios organizacijos [18]. Analizė buvo atliekama pasitelkus OWASP rizikų vertinimo metodologiją. Ši organizaciją šio tipo atakas taip pat įvardijo pačiomis pavojingiausiomis ir 2007 metais, todėl injekcijų atakų rizikos yra pačios aktualiausios jau keletą metų.

1 lentelė OWASP pavojingiausios rizikos 2007 – 2010 metais

OWASP pavojingiausios rizikos 2007 metais	OWASP pavojingiausios rizikos 2010 metais
Injekcijnės atakos (Injections)	Injekcinės atakos (Injections)
Programinio kodo įterpimas į vartotojų peržiūrimus puslapius (XSS)	Programinio kodo įterpimas į vartotojų peržiūrimus puslapius (XSS)
Nepakankama autentifikacija (Broken Authentication and Session Management)	Nepakankama autentifikacija (Broken Authentication and Session Management)
Pinknaudžiavimas funkcionalumu (Insecure Direct Object References)	Pinknaudžiavimas funkcionalumu (Insecure Direct Object References)
Užklausų į svetainę klastojimas (CSRF)	Užklausų į svetainę klastojimas (CSRF)
Nesaugūs konfigūraciniai nustatymai	Nesaugūs konfigūraciniai nustatymai
Nepakankama kriptografija	Nepakankama kriptografija
Neapsaugotos nuorodos	Neapsaugotos nuorodos
Nesaugi komunikacija	Nesaugi komunikacija
Kenkėjiškų programų aktyvavimas	Neapsaugoti nuorodų kreipiniai (Unvalidated Redirects and Forwards)
Duomenų perėmimas	-

Lentelėje 1 pateikta pavojingiausių rizikų palyginimas (rizikų kitimas tarp 2007 metų ir 2010 metų). Galima pastebėti, kad pavojingiausios rizikos esančios žiniatiklio sistemose yra injekcijų atakos. Ši lentelė nenusako atakos paplitimą. Lentelėje yra vertinami keli kriterijai. Rizikos vertinimas buvo atliktas naudojant OWASP rizikos vertinimo metodiką.

2 lentelė Injekcijų rizikos vertinimas

Įsilaužėlis	**Atakos vektorius**	**Saugumo spragos**		**Techninis poveikis**	**Žala**
-	**Atakos sudėtingumas: nesudėtingas**	**Paplitimas: dažnai pasitaikantis**	**Aptikimas: vidutinis**	**Poveikis: didelis**	-
Gali būti bet koks asmuo, kuris siunčia kenksmingą užklausą. Tai gali būti išorinis, vidinis naudotojai bei administratorius.	Įsilaužėlis siunčia nesudėtingą tekstinę užklausą išanalizavus atakuojamos sistemos interpretatorių. Visa siunčiama informacija gali būti injekcijos vektorius įskaitant ir išorinius šaltinius.	Injekcijos požymiai atsiranda tuomet, kai nepatikima informacija yra siunčiama į interpretatorių. Injekcijų atakos yra labai paplitusios. Injekcijų pažeidžiamumai dažnai aptinkami: SQL užklausose, LDAP užklausose, Xpath užklausose ir kitur. Pažeidžiamumai yra nesunkiai aptinkami analizuojant sistemos kodą, tačiau sunkiau aptikti testuojant sistemą. Įvairus analizavimo mechanizmai ir įrankiai gali padėti aptikti šio tipo atakas.		Injekcijų atakos gali įtakoti duomenų netekimo, atskleidimo veiksmus. Taip pat galimas prieigos paslaugos sutrikdymas ar net viso serverio užgrobimas.	Visa informacija gali būti pakeista, pavogta ar ištrinta.

1.1. INJEKCINIŲ PAŽEIDŽIAMUMŲ ANALIZĖ

SQL injekcijos

[3] SQL injekcijos kelia rimtas saugumo grėsmes žiniatiklio sistemose. Šio tipo atakos įsilaužėliui suteikia neribotas teises prieiti prie jautrios, konfidencialios informacijos, kuri patalpinta duomenų bazėje. Nors mokslininkai ir atlikti tyrimai pasiūlė daug metodų kaip apsisaugoti nuo šių atakų, tačiau šių tipo atakos yra labiausiai paplitusios tarp žiniatiklio sistemų [3]. Visi apsaugos metodai nuo šio tipo atakų nėra visiškai patikimi ir neapsaugo nuo šių atakų įvairaus panaudojimo. Kenksmingas injekcijų programinis kodas gali būti patalpintas įvairiausiose sistemos vietose , todėl reikia užtikrinti bet kokių parametrų perdavimą į duomenų bazę. Kadangi atakų veikimo sritis yra

labai didelė, todėl sudėtinga pasiūlyti apsaugos metodą, kuris galėtų maksimaliai apsaugoti informaciją esančia duomenų bazėje.

[20] SQL injekcijos gali būti suskirstytos į tris pagrindines klases:

- ✓ Dvikryptės: duomenys yra gaunami tuo pačiu kanalu, kuriame buvo modifikuota SQL užklausa. Tai pati lengviausia atakos forma. Atakos metu duomenys yra atvaizduojami pačioje žiniatiklio sistemoje.
- ✓ Vienakryptės: duomenys yra gaunami naudojant skirtingus kanalus. Tai gali būti rezultatų suformavimas ir išsiuntimas elektroniniu paštu įsilaužėliui.
- ✓ Netiesioginės: nėra realaus duomenų perdavimo, tačiau įsilaužėlis gali stebėti duomenų bazės serverio būsenas, siunčiant tam tikras užklausas į serverį [20].

Nepriklausomai nuo atakos klasės įsilaužėlis privalo modifikuoti užklausos sintaksę. Jei sistema gražina klaidos pranešimą, kuris sugeneruotas perdavus netinkamą užklausą, tuomet įsilaužėlis šią informaciją gali panaudoti tam, kad būtų suformuota sėkminga injekcijos ataka.
Jei klaidos pranešimas yra slepiamas pačios sistemos, tuomet įsilaužėlis turi pakeisti užklausos struktūrą ir logiką, tokia ataka vadinama – nuspėjama SQL injekcija (Blind SQL Injection).

Atakos veikimo principas

Atakos veikimo sritis yra tarp serverio ir naudotojo naršyklės.

2 pav. Žiniatiklio sistemos architektūra

Įsilaužėlis pasinaudojęs viešai prieinama sistema gali manipuliuoti duomenų bazės užklausas. Jei sistema yra pažeidžiama šio tipo atakoms, puslapio administratorius gali net neįtarti, kad buvo įsilaužta į sistemą. Pasinaudojus internetine naršykle galima atvaizduoti duomenų bazės užklausos rezultatus. Duomenų bazė neinterpretuoja gautos užklausos, o tik gražina rezultatą, todėl norint apsaugoti sistemą reikalinga naudotojų įvesties patikra serveryje. Pateikus kenksmingą užklausą duomenų bazės rezultatas gali būti konfidencialios informacijos atskleidimas.

SQL injekcijų aptikimas

Reikia patikrinti ar sistema naudoja duomenų bazę tam, kad būtų galima sėkmingai atlikti ataką. Galimi trys dažniausiai pasitaikantys veiksmai, kada sistema turi jungtis prie duomenų bazės serverio:

- ✓ Prisijungimo prie sistemos langas: kai yra atliekama sistemos naudoto autentifikacija dažniausiai yra patikrinama, ar duomenų bazėje yra įrašas su įvestu naudotojo vardu ir slaptažodžiu (slaptažodis gali būti užkoduotas maišos būdu).
- ✓ Paieškos sistema: eilutės tipo kintamasis gali būti perduotas į duomenų bazę tam, kad būtų atrikti duomenys pagal paieškos raktažodį.
- ✓ E-komercijos sistemos: visos prekės bei jų aprašymai, kainos ir kita tikėtina, kad yra patalpintos reliacinėje duomenų bazėje.

Įsilaužėlis privalo identifikuoti visus įvedimo laukus, taip pat ir paslėptus. Pasinaudojus kiekvienu lauku atskirai yra reikalinga modifikuoti užklausą taip, kad būtų gautas klaidos pranešimas suformuotas duomenų bazėje. Patys pirmi veiksmai, kuriuos galima atlikti norint gauti klaidos pranešimą, gali būti tiesiog kabučių (’) ar kabliataškio (;) įvedimas į lauką. Kabutės duomenų bazėje aprašo eilutės tipo kintamąjį, todėl jas įvedus duomenų atrinkimas gali nesuveikti ir taip bus gaunama klaida. Kabliataškis aprašo užklausos pabaigą. Tikėtina, kad duomenų atrinkimas taip pat gali nesuveikti, todėl ir yra tikimybė, kad bus gautas klaidos pranešimas. Netinkamai panaudojus užklausą galimos klaidos pranešimo pavyzdys gali būti toks (Microsoft SQL serverio atveju):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before
the character string ''. /target/target.asp, line 2254
```

Taip pat ir komentaro simboliai (--) ar kiti raktažodžiai tokie kaip „AND“ arba „OR“ gali būti panaudoti siekiant modifikuoti užklausą. Dažnai pasitaikanti sistemos klaida gali būti sugeneruota tuomet kai į lauką yra įvedama eilutės tipo kintamas, nors sistema tikisi, kad bus įvestas skaitmuo. Atlikus tokius veiksmus galimas klaidos pranešimas:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int. /target/target.asp, line 6542
```

Tokie klaidos pranešimai pateikia pakankamai informacijos, kuria įsilaužėlis gali pasinaudoti ir atlikti sėkmingą injekcijos ataką. Tačiau dažnai sistemos pateikia mažiau informacijos apie klaidą, tai gali būti paprastas „500 Server Error“ klaidos pranešimas. Galimas ir

specialaus klaidos puslapio rodymas. Tokiu atveju yra naudojamos nuspėjamų SQL injekcijų (Bling SQL injection) atakos principai. Bet kokiu atveju reikia patikrinti, kuris laukas yra pažeidžiamas.

SQLIA pavyzdys

Turime suformuotą SQL užklausą:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Tokio tipo užklausos yra dažnai formuojamos kai yra autentifikuojamas sistemos naudotojas. Jei įvykdžius užklausos komandą yra gražinamas rezultatas, tuomet leidžiama prisijungti prie sistemos, sukuriama sesija ir atliekami kiti veiksmai. Kitu atveju prieiga prie sistemos yra draudžiama. Visos užklausos parametrų reikšmės yra gaunamos iš prisijungimo puslapio formos laukų. Įvesta informacija į laukus:

```
$username = 1' or '1' = '1'
$password = 1' or '1' = '1'
```

Tokiu atveju kai įvesta tokia informacija užklausa bus tokia:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' =
'1'
```

Jeigu reikšmės į serverį bus perduodamos metodu „GET“ ir pažeidžiamos sistemos adresas bus www.pavyzdys123.lt užklausa bus suformuota taip:

```
http://www.pavyzdys123.lt/default.aspx?username=1'%20or%20'1'%20=%20'1&password=
1'%20or%20'1'%20=%20'1
```

Įvykdžius tokią užklausą galima pastebėti, kad visada gražinamas teigimas rezultatas. Šiuo atveju sistema autentifikuoja sistemos naudotoją, nors naudotojo vardas ir slaptažodis yra nežinomi.

Kitas užklausos pavyzdys galimas toks:

```
SELECT * FROM Users WHERE ((Username='$username') AND ( Password = MD5
('$password')))
```

Šiuo atveju yra dvi problemos, tai kad naudojamas naudotojo vardo palyginamas su lenktiniais skliaustais ir slaptažodžio MD5 kodavimas. Tam, kad suformuoti teisingą užklausą reikia užklausos gale pridėti tiek skliaustelių, kol užklausa bus teisinga. Norint išspręsti antrą problemą įvedimo lauko gale yra pridedami simboliai (/*), kurie reiškia komentaro pradžią. Visa užklausos dalis, kuri yra už šių simbolių yra interpretuojama kaip komentaras. Kiekviena duomenų

bazės valdymo sistema turi savo komentaro simbolius, tačiau dažniausiai yra (/*). Oracle duomenų bazėje yra dviejų tipų komentarų simboliai, tai (/*) ir (--).

```
$username = 1' or '1' = '1'))/*
$password = foo
```

Tokiu atveju gaunama užklausa:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND
(Password=MD5('$password')))
```

Užklausų perdavimas sistemai bus naudojant nuorodą:

```
http://www.pavyzdys123.lt/default.aspx?username=1'%20or%20'1'%20=%20'1'))/*&pass
word=foo
```

Ši užklausa gražina rezultatų kiekio skaičių. Autentifikacijos metu galima nurodyti, kad gražinamas rezultatas būtų lygus 1. Šiame metode reikia modifikuoti SQL užklausą ir įrašyti operatorių: „LIMIT <num>“ . Naudojant šį operatorių yra nurodoma, kiek rezultatų tikimės gauti. Įvedimo laukų reikšmes reikia pakeisti į:

```
$username = 1' or '1' = '1')) LIMIT 1/*
$password = foo
```

Tokiu atveju yra suformuojama užklausa:

```
http://www.pavyzdys123.lt/default.aspx?username=1'%20or%20'1'%20=%20'1'))%20LIMI
T%201/*&password=fo
```

Užklausų sąjungos

Šio tipo atakose naudojamas „UNION“ operatorius, kuris prijungia lentelės duomenis prie pirminių rezultatų. Šiuo metodu suklastota užklausa yra prijungiama prie pirminės užklausos. Tarkime turime užklausą, kuri yra vykdoma serveryje:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Taip pat nustatome identifikacinio kodo reikšmę:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

Tuomet turėsime sekančią užklausą:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCarTable
```

Ši užklausa prijungs rezultatus apie kreditinių kortelių turėtojus prie pirminių rezultatų. Ši informacija bus sukompromituota. Reikalingas operatorius „ALL“ tam, kad būtų gražinama visi rezultatai, taip pat ir pasikartojantys. Užklausoje naudojamos papildomos dvi reikšmės. Šios dvi reikšmės reikalingos tam, kad užklausos turėtų tokį patį parametrų skaičių. Tokiu atveju bus išvengta sintaksės klaidų.

Nuspėjamos SQL injekcijos (Blind SQL injections)

Šio tipo atakos yra kita kategorija SQL injekcijų atakų. Tokios atakos naudojamos kuomet nėra žinoma duomenų bazės struktūra, lentelės pavadinimas ar kita. Pavyzdžiui, kai įsilaužėlis neatskleidžia informacijos apie duomenų bazę esant klaidos pranešimui. Atvaizduojamas specialus klaidos apdorojimas.

Naudojant palyginimo metodiką siunčiant užklausas, galima atvaizduoti konfidencialios informacijos laukus, ir taip atlikti sėkmingą ataką. Šio tipo atakos yra pagrįstos siunčiant logines komandas į serverį ir analizuojant gautus rezultatus. Naudojant nuorodą į užklausą, kuri turi pažeidžiamų parametrų galima įvykdyti šio tipo atakas:

```
http://www.pavyzdys123.lt/default.aspx?id=1'
```

Įvykdžius užklausą gali būti suformuojamas specialus klaidos pranešimas, kad įvyko sintaksės klaida, tačiau įsilaužėlis gali modeliuoti situaciją, kad serveryje buvo vykdoma užklausa:

```
SELECT laukas1, laukas2, laukas3 FROM Naudotojai WHERE Id='$Id'
```

Ši užklausa yra pažeidžiama ir prieš tai aprašytoms atakoms. Įsilaužėlio tikslas yra atskleisti naudotojo vardo lauko reikšmę. Pasinaudojus tam tikromis funkcijomis galimas simbolių atvaizdavimas, kol bus gautas visas vardas. Šiam tikslui galimas kelių pseudo funkcijų panaudojimas:

SUBSTRING (tekstas, pradžia, ilgis): ši funkcija gražina simbolius, kurie prasideda pradžios reikšme ir baigiasi ilgio reikšme. Jei pradžios reikšmė didesnė už ilgio reikšmę, tuomet gražinama null reikšmė.

ASCII (simbolis): gražinama įvesties simbolio ASCII reikšmė. ‚Null“ reikšmė gražinama, jei simbolis lygus 0.

LENGTH (tekstas): gražinama teksto ilgio reikšmė.

Naudojant šias funkcijas aptinkamas pirmas simbolis. Gavus simbolio reikšmę yra imamas sekantis tol, kol gaunama visa sudėtinė reikšmė. „SUBSTRING“ funkcija bus panaudota tam, kad gauti vieną simbolinę reikšmę (parametro ilgis turi būti lygus 1). ASCII funkcija yra naudojama gauti ASCII lentelės reikšmę. Naudojant šią funkciją yra galimas skaitmenų lyginimas. Lyginimo funkcija yra naudojama, kol randami simboliai, tenkinantys sąlygą:

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

Naudojant funkcijas bus suformuojama užklausa:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND
ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

Ši užklausa gražins teigiamą rezultatą tik tuo atveju, jei pirmas vardo simbolis atitiks ASCII lentelės reikšmę 97. Jei sąlyga nėra tenkinama, toliau didinama ASCII reikšmė, kol bus gautas teigiamas rezultatas. Jei gaunamas teigiamas rezultatas simbolis yra žinomas ir galima analizuoti sekančią parametro reikšmę. Tam, kad būtų atskirta, kada yra gaunama teigiam, kada neigiam reikšmė, sąlygoje reikia panaudoti loginę sąlygą, kuri visada yra neigiama.

```
$Id=1' AND '1' = '2
```

Tuomet galima suformuoti užklausą taip:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

Sužinojus visas parametro reikšmes yra įvykdoma užklausa su neigiama sąlyga, tam užtenka nustatyti ar atspėta reikšmė yra teisinga.

Tačiau naudojant šią metodiką nebuvo įvertinta, kada reikia baigti simbolių analizavimą. Tokiu atveju reikia papildomos „LENGTH“ procedūros užklausoje. Jei lyginama reikšmė yra lygi ASCII simboliui 0 (reikšmė null) ir gražinamas sąlygos teigiamas rezultatas, tuomet yra baigiamas simbolių analizavimas arba analizuojama reikšmė yra lygi null.

Naujai suformuota užklausa naudojant ilgio funkciją:

```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

N yra iki tol analizuoto simbolio numeris (null reišmė neskaičiuojama). Tuomet užklausa bus:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND
'1' = '1'
```

Jei užklausa gražina teigiamą rezultatą, tuomet baigiama simbolių analizė ir žinome parametro reikšmę. Jei gražinamas neigiamas rezultatas, tuomet analizuojamas simbolis yra null reikšmė. Tokiu atveju reikia toliau nagrinėti sekančius simbolius, kol bus gauta sekanti null reikšmė.

Norint panaudoti šio tipo atakas reikia modeliuoti sudėtingas SQL užklausas.

Injekcijos DB procedūrose (Stored Procedures)

Neteisingai suformuotos saugios procedūros gali būti taip pat pažeidžiamos injekcijų atakoms, kaip ir dinaminės SQL užklausos. Naudojant dinamines užklausas ir saugias procedūras reikia atlikti naudotojo įvesties patikrą tam, kad išvengti injekcijų atakų. Jei užklausos nebus tinkamai patikrintos įsilaužėlis galės įterpti kenksmingą kodą į užklausą.

Netinkamai suformuota saugi procedūra:

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = ‘
Select 1 from users
Where username = ‘ + @username + ‘ and passwd = ‘ + @passwd
exec(@sqlstring)
Go
```

Suformavus tokią procedūrą ir į įvedimo laukus įvedus netinkamus duomenis, galimas procedūros sukompromitavimas:

```
anyusername or 1=1'
anypassword
```

Ši procedūra nėra apsaugota ir netikrina perduodamų parametrų, todėl yra pažeidžiama injekcijų atakoms.

Tačiau galima ir saugesnis procedūrų formavimas:

```
Create procedure get_report @columnamelist varchar(7900) As
Declare @sqlstring varchar(8000)
Set @sqlstring = ‘
Select ‘ + @columnamelist + ‘ from ReportTable‘
exec(@sqlstring)
Go
```

Šios procedūros yra saugesnes, nes galimas parametrų tikrinimas pačioje duomenų bazėje, prieš vykdant užklausą.

Xpath injekcijos

Daugelis žiniatiklio sistemų naudoja duomenų bazes atlikti tam tikras operacijas bei manipuliuoti duomenimis. Žiniatiklio sistemose populiariausios yra reliacinės duomenų bazės, tačiau pastaraisiais metais vis didėja duomenų bazių populiarumas, kuriuos operuojama duomenimis xml kalbos pritaikymu. Kaip ir reliacinės duomenų bazės jos yra pasiekiamos naudojant SQL užklausas, tačiau rezultatai pateikiami xml formatu naudojant xpath technologiją. Xpath technologija yra labai panaši kaip ir SQL kalbos, todėl xpath injekcijų atakų principai taip pat yra panašūs. Kai kuriais atvejais xpath technologija yra galingesnė nei SQL kalba. Todėl xpath injekcijos ataka gali būti panaudota ir SQL injekcijos atakos metu. Tokio tipo atakos vektorius yra didesnis ir taikymo sritis apima kelias technologijas. Taip pat šio tipo atakos yra pranašesnės už SQL injekcijų atakas dėl ACL (Access control list) sąrašų nebuvimo. Tai reiškia, kad naudojant xpath technologiją galima pasiekti kiekvieną xml dokumento elementą.

Xpath injekcijų atakos pavyzdys

Šio tipo atakos panašios i SQL injekcijos atakas, todėl standartinis pavyzdys pateikiamas prisijungimo langas. Naudojant prisijungimo formą ir tam tikrą xml duomenų struktūrą reikalinga sumodeliuoti injekcijos ataką.

Xml duomenų struktūra:

```
<?xml version="1.0" encoding="utf-8"?>
<Naudotojai>
   <Naudotojas ID="1">
      <Vardas>Jonas</Vardas>
      <Pavarde>Jonaitis</Pavarde>
      <NaudotojoVardas>aNaud</NaudotojoVardas>
      <Slaptazodis>SlaptaFraze</Slaptazodis>
      <Tipas>Administratorius</Type>
   </Naudotojas>
   <Naudotojas ID="2">
      <Vardas>Petras</Vardas>
      <Pavarde>Petraitis</Pavarde>
      <NaudotojoVardas>aNaudotojas</NaudotojoVardas>
      <Slaptazodis>Neatspesi</Slaptazodis>
      <Tipas>Registruotas</Tipas>
   </Naudotojas>
</Naudotojai>
```

Naudotojui suvedus prisijungimo duomenis sistema, naudojant xpath technologiją, turi autentifikuoti naudotoją, rasti prisijungimo duomenis xml faile.

```
String RastiNaudotojaXPath;
RastiNaudotojaXPath = "//Naudtojas[NaudotojoVardas/text()='" +
Request("NaudotojoVardas") + "' +
Slaptazodis/text()='" + Request("Slaptazodis") + "']";
```

Tokiu atveju naudotojo išrinkimas suveiks, tačiau įsilaužėlis gali atrinkti naudotoją ir nežinodamas prisijungimo vardo bei slaptažodžio ir taip gauti reikalingą xml dokumento mazgą.

```
Naudoto vardas: testNaudotojas' or 1=1 or 'a'='a
Slaptažodis: neatspesi
RastiNaudotojaXPath bus //Naudotojas[NaudotojoVardas/text()='testNaudotojas' or
1=1 or 'a'='a' + Slaptazodis/text()='neatspesi']
Loginė sąlyga bus lygi:
        //Naudotojas[(VartotojoVardas/text()='testNaudotojas' or 1=1) or
        ('a'='a' Ir Slaptažodis/text()='neatspesi')]
```

Šiuo atveju tik pirmoji xpath elemento dalis turi būti teigiama. Slaptažodžio lyginimo dalis yra nesvarbi, o naudotojo vardo išrinkimo dalis visada atrinks naudotoją dėl loginės sąlygos panaudojimo 1 = 1.

Galimas ir apsisaugojimas nuo tokio tipo atakos. Taip pat kaip ir nuo SQL injekcijos galima apsisaugoti naudojant kintamųjų konvertavimą į eilutę. Kabutės turi būti interpretuojamos kaip eilutė tam, kad jos nesudarytų loginės sąlygos ir neturėtų įtakos rezultatų gražinimui.

```
String RastiNaudotojaXPath;
RastiNaudotojaXPath = "//Naudotojas[NaudotojoVardas/text()='" +
Request("NaudotojoVardas").Replace("'", "'") + "' +
Slaptazodis/text()='" + Request("Slaptazodis").Replace("'", "'") + "']";
```

[22] Pagal pateiktą sprendimą [22] galimas ir patikimesnis apsisaugojimas nuo šio tipo atakų. Sukompiliuotas xpath yra suformuotas prieš vykdymą. Kitu atveju xpath formuojamas iškart įvedus informaciją į laukus. Naudojant pasiūlytą metodą nereikia vertinti pavojingų simbolių panaudojimo.

1.2. ŽINIATINKLIO SISTEMŲ APSAUGOS METODAI

Dažniausiai pasitaikanti žiniatiklio sistemų saugumo spraga yra netinkama įvesties patikra. Parametrai į sistemą gali būti perduoti iš sistemos naudotojo ar kitos sistemos. Beveik visi sistemų pažeidžiamumai atsiranda dėl netinkamos įvesties patikros. Tokie pažeidžiamumai kaip interpretatoriaus ar kodo injekcijos, failų sistemos atakos bei buferio perpildymo atakos, atsiranda dėl netinkamos įvesties. Niekada neturi būti pasitikima sistemos įvestimi, kurios pagalba yra atliekama bet kokia manipuliacija su duomenimis.

Daugeliu atveju šifravimas gali sumažinti duomenų įvesties pažeidžiamumą. Pavyzdžiui, jei yra naudojamas HTML šifravimas pieš tai, kai įvesti duomenys yra perduodami į naršyklę galima išvengti daugumą XSS atakų.

Taikymo vektorius

Vientisumo patikra turi būti taikoma ten, kur yra bent kokia abejonė dėl įvesties patikimumo, pavyzdžiui, perduodamų paslėptų laukų reikšmės į naršyklę arba internetinis apmokėjimas per trečias šalis, perduodant transakcijos identifikacijos numerį. Galimi keli integralumo patikros metodai:

- Kontrolinė suma (cheksum)
- HMAC- šifruotos žinutės autentifikacijos kodas
- Šifravimas
- Elektroninis parašas

Įvesties patikra turi būti atlikta kiekviename sistemos lygmenyje. Patikra turi būti atliekama serveryje. Pavyzdžiui, sistemos prezentacijos sluoksnyje turi būti tikrinama su tuo sluoksniu susijusios saugumo spragos, duomenų sluoksnyje turi būti tikrinama informacija susijusi su duomenis: SQL ar HQL injekcijos, failų nuorodos turi būti tikrinamos nuo LDAP injekcijų atakų ir kita.

Įvesties patikros strategija

Yra taikomas keturios pagrindinės įvesties patikros strategijos, jos turi būti ir naudojamos.

Priimti žinomus gerus parametrus. Ši metodika dar žinoma kaip baltasis sąrašas (whitelist) arba teigiama patikra (positive validation). Šio metodo pagrindinė idėja yra įvesties tikrinimas, ar reikšmė atitinką sudarytą, žinomą galimų rezultatų sąrašą. Jei įvestis neatitinka sąraše esančių reikšmių, tuomet įvedimas turi būti atmestas. Duomenys turėtų būti tokie:

- Visada nustatytas tipas
- Duomenų ilgis patikrintas ir minimizuotas
- Patikrintos ribos, jei skaičius
- Sintaksė ar gramatika turi būti patikrinta prieš pirmą panaudojimą

Jei yra tikimasi pašto indekso, atliekam įvesties patikra (tipo, ilgio ir sintaksės):

```
public String PastoIndeksas(String indeksas) { return (indeksas != null &&
Pattern.matches("^(((2|8|9)\d{2})|((02|08|09)\d{2})|([1-9]\d{3}))$",
indeksas)) ? indeksas : ""; }
```

Atmesti žinomus blogus parametrus. Ši metodika dar žinoma kaip (blacklist) arba negatyvi patikra (negatyve validation). Ši metodika yra gan pavojinga, nes žinomų blogų eilučių

sąrašas gali būti begalinis. Naudojant šią metodiką jūsų „blogų“ parametrų sąrašas turi būti pildomas nuolatos ir jis niekada nebus baigtas.

```
public String removeJavascript(String input) {
Pattern p = Pattern.compile("javascript", CASE_INSENSITIVE);
p.matcher(input);
return (!p.matches()) ? input : '';
}
```

Patikrinti kiekvieną blogą elementą gali reikti panaudoti daugiau nei 90 reguliarių išraiškų masyvų ir tai turėtų būti ant kiekvieno įvedimo lauko. Greitaveikos atžvilgiui tai yra lėta. Ši metodika yra panaši į antivirusinių sistemų atnaujinimų metodiką.

Įvesties pakeitimas į tinkamus vykdyti (whitelist). Kiekvienas įvesties elementas nesantis sąraše (whitelist) turi būti pašalintas, užšifruotas ar pakeistas.

Jei yra tikimasi telefono numerio, tuomet galimas raidinių ženklų išmetimas iš įvesties. „(666)113-1234", "555.113.1234“ ir „666\";DROP TABLE USER;--123.1234“ eilutės būtų paverstos į „6661131234“. Reikia patikrinti ir gautus skaičius. Šis metodas yra geras, nes visuomet bus priimama įvestis nepriklausomai, kas buvo įvesta, tačiau tam tikrais atvejais ne visada galima numatyti sistemos veiksmus su duomenimis, todėl tai gali sukelti nenumatytą įvesties panaudojimą ar net pateikti klaidingą informaciją naudotojui.

Jei yra naudojami vien tik simboliai, tuomet sunku įverti, ar simbolių masyvas sudarytas iš leistinų simbolių. Tuomet reikia ne raidinius simbolius pakeisti šifruotais. „Man patinka jūsų puslapis“ eilutė būtų pakeista į „Man+patinka+jūsų+puslapis%21“. Šis pavyzdys naudoja URL šifravimą.

Jei puslapyje yra talpinami tam tikri failai taip pat sunku įvertinti, ar failo turinys yra leistinas ar ne. Vienintelis rūpestis yra apsaugoti pačią sistemą nuo netinkamos įvesties, todėl nereikia atlikti papildomų veiksmų su failu, užtenka tik užšifruoti failą, pavyzdžiui, base64 koduote.

Įvesties pakeitimas į tinkamus vykdyti (blacklist). Išmesti ar pakeisti (HTML elementus ar kabutes) tam, kad įvestis būtų patikima. Ši metodika yra taip pat niekad neužbaigiama, neturinti baigties taško, sąrašas pildomas nuolat.

```
public String quoteApostrophe(String input) {
    if (input != null)
        return input.replaceAll("[\']", "&rsquo;");
    else
        return null;
}
```

Įvesties apsauga nuo klastojimo

Yra nemažai įvesties šaltinių, tai gali būti:

- HTTP antraštės (REMOTE_ADDR ar PROXY_VIA)
- Sistemos kintamieji, (getenv() ar serverio parametrai)
- Visi GET, POST metodai, slapukų informacija
- Į šį sąrašą įeina ir kiekvienas statinis HTML elementas. Jis gali būti perrašytas tam, kad atitiktų įsilaužėlio poreikius.
- Išorinės sistemos (XML įvestis, RMI, žiniatiklio servisai).
- Nustatymų informacija. Serverio nustatymai ir kita.

Visa šie šaltiniai yra nepatikimi, todėl bet kokios informacijos gavimas iš tokių šaltinių turi būti tinkamai patikrintas prieš pirmą panaudojimą.

Specialūs simboliai

Yra daugybė simbolių, kurie turi specialią prasmę įvairiose sistemose. Dažnai naudojami ir kenksmingi simboliai:

- NULL (zero) %00
- LF - ANSI chr(10) "\r"
- CR - ANSI chr(13) "\n"
- CRLF - "\n\r"
- CR - EBCDIC 0x0f
- Kabutės " '
- Komandos, pasvirieji brūkšniai, tarpai, tabuliacijos simboliai - naudojami CSV ir kituose specialiuose duomenų formatuose
- <> - XML ir HTML elementų žymekliai, nukreipimo simboliai
- @ - naudojama elektroninių paštų adresuose
- 0xff
- Ir kiti

1.3. FORMALIOS KALBOS KALBŲ SINTAKSEI APIBRĖŽTI

1.3.1. META PROGRAMAVIMAS

[26] Programinį kodą generuojančios programos yra vadinamos meta programomis, o tokių programų rašymas, vadinamas meta programavimu [26].

Įvairūs meta programavimo panaudojimai

Tarkim jei rašoma didelės apimties programa ir daugybę funkcijų sudaro standartinis kodas, tuomet galima sudaryti mini kalbą, kuri sugeneruotų standartinį programinį kodą. Taikant tokią metodiką labiau susikoncentruojama yra ties svarbesnėmis funkcijomis. Dažnai naudojamą kodą

reikia taip pat aprašyti kaip funkciją, kurią galima iškviesti. Prieš kompiliuojant programą galimas tokių funkcijų iškvietimas, kurios bus paverčiamos į dažnai naudojamą programinį kodą.

Naudojant meta programavimą galima aprašyti sudėtingas funkcijas paprastais metodais. Naudojant meta funkcijas yra sutaupoma nemažai programinio kodo rašymo laiko, tuo pačiu ir sumažėja klaidų padarymo tikimybė. Programos parašytos taikant tokią metodiką yra suprantamesnės, galimas panaudojimas tarp skirtingų programavimo kalbų, taip pat tokio tipo kodą lengviau redaguoti.

C parengiamojo doroklė

Meta programavimas sudarytas iš tekstinių makrokomandų. Makrokomandos tiesiogiai įtakoja programinį kodą, šios komandos neinterpretuoja pačios komandos reikšmės. Dvi dažniausiai naudojamos makrokomandų sistemos yra c parengiamoji doroklė (preprocessor) ir M4 makrokomandų parengiamoji doroklė.

C programavimo kalboje naudojama define makrokomanda

```
#define SWAP(a, b, type) { type __tmp_c; c = b; b = a; a = c; }
```

Ši makrokomanda leidžia keisti tam tikro tipo dvi reikšmes. Šias funkcijas geriau aprašyti makrokomandomis nes:

- Funkcijos iškvietimas būtų sudėtingesnis paprastai operacijai
- Galimas parametrų adresų perdavimas, o ne pačių reikšmių
- Nereikia rašyti skirtingų funkcijų keičiant skirtingo tipo reikšmes

Šios makrokomandos galimas panaudojimas:

```
#define SWAP(a, b, type) { type __tmp_c; c = b; b = a; a = c; }
int main()
{
    int a = 3;
    int b = 5;
    printf("a is %d and b is %d\n", a, b);
    SWAP(a, b, int);
    printf("a is now %d and b is now %d\n", a, b)
    return 0;
}
```

Kai yra paleidžiamas C preprocesorius, tuomet pakeičiamas tekstas iš SWAP(a, b, int) į { int __tmp_c; __tmp_c = b; b = a; a = __tmp_c; }.

Kai reikia aprašyti makrokomandas ir sąlygų reiškinius, pačios komandos tampa ganėtinai sudėtingos. Tarkime turime makrokomandą:

```
#define MIN(x, y) ((x) > (y) ? (y) : (x))
```

Sąlyga tampa sudėtinga, dėl dažno skliaustelių naudojimo. Pavyzdžiui, jei procedūra bus iškviesta su parametrais MIN(27, b=32) ir nebus naudojami skliausteliai, tuomet sąlyga bus interpretuojama: 27 > b = 32 ? b = 32 : 27. Toks sąlygos panaudojimas iššauks kompiliavimo klaidą, nes kompiliatorius 27 > b vertins pirmą sąlygą, kuri yra neapdorojama. Sudėjus skliaustelius problema išsisprendžia, tačiau sąlygų aprašymas naudojant makrokomandas tampa sudėtingas.

Egzistuoja ir kitų problemų sąlygų panaudojimo atveju. Jei perduodami parametrai bus kviečiami naudojant procedūras, tuomet procedūros bus kviečiamos daugiau nei vieną kartą, nors tai yra visiškai perteklinės funkcijos. Preprocesorius neinterpretuoja pačios programavimo kalbos, jis tik pakeičia programinį kodą kitu kodu. Jei iškviestume MIN(do_long_calc(), do_long_calc2()), tuomet ji išsiplėstų į ((do_long_calc()) > (do_long_calc2()) ? (do_long_calc2()) : (do_long_calc())) procedūrą esančią makrokomandoje. Toks funkcijos panaudojimas užtruks daugiau laiko, nes skaičiavimai bus atliekami mažiausiai du kartus.

Galimas ir blogesnis scenarijus, kai yra naudojami šalutiniai procesai (spausdinimas, globalių kintamųjų modifikavimas ir kita), nes šios šalutinės funkcijos bus vykdomos kelis kart. Galimas net blogos reikšmės gražinimas, kai yra gražinamas skirtingas rezultatas du kartus.

1.3.2. BEKAUS IR NAURO FORMA

Kalba apibrėžiama taisyklių aibe. Kiekvienos taisyklės kairėje pusėje rašomas kalbos sąvokos pavadinimas (meta kintamasis), o dešinėje – jos formali apibrėžtis, išreikšta kitų sąvokų pavadinimais arba kalbos elementais, kurie čia vadinami terminaliniais simboliais. [24] BNF yra laisvo konteksto gramatikos taisyklių rinkinys [24]. Taisyklės pateikiamos šiek tiek kitokiu pavidalu, negu priimta gramatikose:

- ✓ Vietoj rodyklės (→) rašomas simbolis (::=)
- ✓ Neterminaliniai simboliai rašomi tarp kampinių skliaustų (<>)

Antrojo tipo gramatikos taisyklė (S→aSb|ab) būtų užrašoma taip:

<S> ::= a<S>b|ab

Kampiniai skliaustai apibrėžia pavadinimo pradžią ir pabaigą. Jais apskliaustas pavadinimas, nežiūrint iš kiek raidžių arba žodžių būtų sudarytas, laikomas vienu neterminaliniu simboliu, pavyzdžiui, <kintamojo vardas>.

Skaičius su ženklu

1. <skaičius> ::= <skaičius be ženklo>
   |<ženklas><skaičius be ženklo>
2. <skaičius be ženklo> ::= <skaitmuo> |<skaitmuo><skaičius be ženklo>
3. <skaitmuo> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
4. <ženklas> ::= + | -

Vardas. Sudaromas iš raidžių ir skaitmenų, prasideda raide.

1. <vardas>::= <raidė>
   | <vardas> <raidė>
   |<vardas> <skaitmuo>
2. <raidė>::= a | ą | b | c | d | e | ę | ė | f | g | h | i | į | y| j | k | l| m | n | o | p | q | r | s | š | t | u | ų | ū | v | w | x | z | ž

Loginės sąlygos

1. <sąlyga>: := <operandas> <lyginimo operatorius> <operandas>
2. <reiškinys> : := <termas>
   | <reiškinys> + <termas>
   | <reiškinys> - <termas>
3. <termas> : := <operandas>
   | <termas> * <operandas>
4. <operandas>: := <skaičius>
   | <kintamojo vardas>
   | (<reiškinys>)
5. <lyginimo operacijos ženklas>: := < | <= | = | <> | > | >=

Kampiniai skliaustais nusako ar simbolis yra terminalinis, pavyzdžiui, <kintamasis> yra terminalinis simbolis, o tas pats užrašas be skliaustų reiškia žodį.

Pirminį simbolį taip pat galima nustatyti iš taisyklių susitarus, kad jis nebus panaudotas nė vienos taisyklės dešinėje pusėje. Kalbos gramatikai apibrėžti pakanka Nekaus ir Nauro formų taisyklių (nereikia nurodyti pradinio neterminalinio simbolio, neterminalinių simbolių abėcėlės ir terminalinių simbolių abėcėlės).

Pačią BNF (Bekaus ir Nauro forma) taip pat galima vadinti kalba. Kadangi ji vartojama kitai kalbai aprašyti, tai ją reikėtų vadinti metakalba.

Išplėstinė Bekaus ir Nauro forma

BNF metakalba užrašytų taisyklių struktūra yra tokia pati, kaip ir matematiniai gramatikos taisyklių užrašai.

Vėliau atsirado BNF išplėstinių modifikacijų, kuriose pakeista taisyklių struktūra. Naujomis išraiškos papildyta BNF dažnai vadinama išplėstąja BNF. Joje yra numatytas taisyklės dešinės pusės simbolių grupavimas panaudojant skliaustus.

3 lentelė Simbolių panaudojimas EBNF

Notacija	Reikšmė
Žodis be kampinių skliaustų	Neterminalinis simbolis
„ ... “	Terminalinis simbolis
, ... ‘	Terminalinis simbolis
(...)	Simbolių grupavimas
[...]	Neprivalomi simboliai
{ ... }	Simboliai yra kartojami nulį ar daugiau kartų
{ ... }-	Simboliai yra kartojami vieną ar daugiau kartų
=	Priskyrimo simbolis
;	Taisyklių skyriklis
|	Simbolis alternatyvai (loginė arba operacija)
,	Sujungimo simbolis
-	Simbolio neigiamumas (išskyrus)
*	Pasikartojimo simbolis
(* … *)	Komentaras
? … ?	Speciali seka

Simbolių grupavimas. Simbolių grupavimas atliekas naudojant paprastus skliaustelius (). Skaičius, kuris yra būtinai su ženklu galima aprašyti taip:

<skaičius su ženklu>::=(+|-)<skaičius>

Simbolių grupių kartojimas. Simbolių grupavimo pasikartojimas atliekamas naudojant laužtinius skliaustus. Galimas dviejų tipų pasikartojimas:

- Nulį ar daugiau kartų
- Vieną ar daugiau kartų

Nulį ar daugiau kartų:

<vardas>::= <raidė> {<raidė> | <skaitmuo>}

<vardų sąrašas>::= <vardas> {, <vardas>}

Vieną ar daugiau kartų:

<vardas>::= <raidė> [{<raidė> | <skaitmuo>}]

<vardų sąrašas>::= <vardas> [{, <vardas>}]

Skaičius su ženklu. Ženklas gali būti ir neprivalomas.

1. <skaičius> ::=[(+|-)] <skaitmuo> {<skaitmuo>}
2. <skaitmuo>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Vardas. Vardą sudaro raidiniai ir skaitiniai ženklai (neprivalomai).

1. <vardas> ::= <raidė> {<raidė> | [<skaitmuo>]}
2. <raidė> ::= a | ą | b | c | d | e | ę | ÷ | f | g | h | i | į | y| j | k | l| m | n | o | p | q | r | s | š | t | u | ų | ū | v | w | x | z | ž

Loginės sąlygos

1. <sąlyga> ::= <operandas> <lyginimo operatorius> <operandas>
2. <reiškinys> ::= <termas> {(+ | -) <termas>}
3. <termas> ::= <operandas> {* <operandas>}
4. <operandas> ::= <skaičius>
   |<kintamojo vardas>
   |(<reiškinys>)
5. <lyginimo operacijos ženklas>::= < | <= | = | <> | > | >=

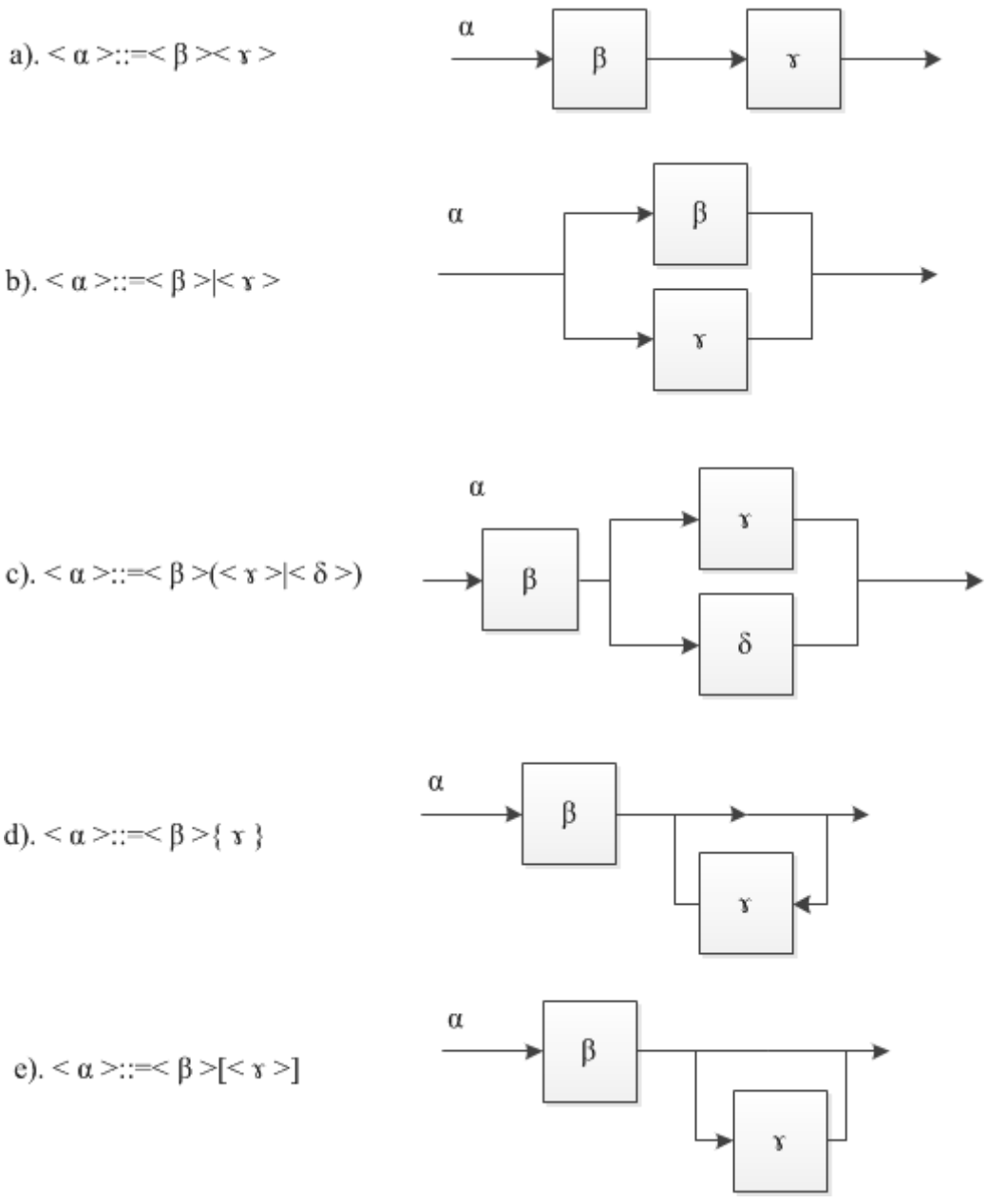
Yra ir kitokių BNF išplėstinės formos modifikacijų. Pavyzdžiui, neterminaliniai simboliai į kampinius skliaustus neskliaudžiami, o jeigu jų pavadinimai sudaryti iš kelių žodžių, tai tie žodžiai sujungiami brūkšneliais, pavyzdžiui, vardų-sąrašas.

Tokiu atveju reikia kaip nors atskirti terminalinius simbolius, išreiškiamus žodžiais (bazinius žodžius) nuo neterminalinių. Dėl to baziniai žodžiai rašomi pusiau juodu šriftu arba visi terminaliniai simboliai rašomi tarp kabučių. Antrasis būdas, nors ir netoks vaizdus, yra universalesnis, nes tinka ir tada, kai nėra galimybės vartoti skirtingus šriftus. Pastarasis būdas dažniausiai vartojamas oficialiuose dokumentuose, pavyzdžiui standartuose, kur skirtingos simbolių interpretacijos gali turėti labai neigiamas pasekmes.

Bekaus ir Nauro formos schemos

Vaizdesnis ir suprantamesnis būdas kitų programavimo kalbų sintaksei aprašyti – sintaksės diagramos.

Kiekviena sintaksės taisyklė vaizduojama diagrama. Diagramos viršuje rašomas jos pavadinimas – taisyklės kairioji pusė. Dešinės pusės simboliai rašomi į geometrines figūras: neterminaliniai į stačiakampius, o terminaliniai į apskritimus arba ovalus. Figūros sujungiamos rodyklėmis taip, kad figūrų apėjimas rodyklių kryptimi duotų visas galimas simbolių sekas, kurias galima gauti iš apibrėžiamo simbolio.



3 pav. BNF pagrindinių ženklų schemos

Paveikslėlyje 3 pavaizduotos Bekaus ir Nauro formos pagrindinių operacijų diagramos. Šios diagramos vaizdžiai atspindi logines operacijas, jų ciklišKumą bei priklausomybes funkcijose. Ženklas | reiškia alternatyvą, todėl diagramoje pakeičiamas išsišakojimu, ženklai { } – kartojimą, todėl pakeičiami atgal grįžtančiomis rodyklėmis (sudarančiomis ciklą), ženklai [ ] – nebūtiną simbolių seką, todėl pakeičiami rodykle, aplenkiančia konstrukciją.

1.4. IŠVADOS

- ✓ Šiuolaikinės žiniatiklio sistemos sudėtingos, atlieka aibę įvairių funkcijų ir dėl minėtų priežasčių tampa labiau pažeidžiamos.
- ✓ Remiantis White Hat Security kompanijos ataskaita, galima teigti, kad daugiausia tinklapių saugumo problemų iškyla dėl jautrios (saugumo prasme) informacijos paviešinimo. Tokia informacija gali būti panaudota pažeidžiant tinklapio, jo serverio ar naudotojų saugumą.
- ✓ Žiniatiklio sistemų injekcijų rizikos buvo įvardintos kaip pavojingiausios 2010 ir 2007 metais. Šio tipo atakos yra aktualiausios jau keletą metų. SQL injekcijos turi aukščiausią pavojingumo įvertinimą CWE duomenų bazėje.
- ✓ Įvesties patikrai naudojami „baltieji“ ir „juodieji“ sąrašai bei parametrų modifikavimas naudojant šiuos sąrašus.
- ✓ Sudarinėjant „baltuosius“ ar „juoduosius“ sąrašus tinka Bekaus ir Nauro forma, nes ši forma naudojama kitos kalbos sintaksėj aprašyti taip pat yra paprasta ir aiški.

2. INTERAKTYVIOS SAUGOS SISTEMOS PROTOTIPAS APSAUGAI NUO INJEKCINIŲ ATAKŲ

Darbo tikslas – sukurti sistemą, kuri analizuotų gautas eilutes. Kuriamos programos pagrindinis tikslas – atpažinti potencialią ataką prieš sistemą. Ataka bus aptinkama naudojant (blacklist) sąrašus.

Prevencinės programinės įrangos tikslas nėra vien nustatyti kenksmingą ataką. Svarbus tikslas yra ją sustabdyti bei informuoti apie tai naudotoją, sistemos administratorių.

2.1. INJEKCINIŲ ATAKŲ APRAŠYMAS BEKAUS IR NAURO FORMA

Pagrindinė BNF forma modeliui

Visos sąlygos, pagal kurias bus tikrinamos eilutės yra sudarytos meta kalbos pagrindu. Meta sąlygos aprašo galimas simbolių sekas eilutėje. Naudojant Bekaus ir Nauro išplėstinę formą galimas pagrindinių injekcinių atakų šablonų sudarymas.

Pagrindiniai terminaliniai elementai:

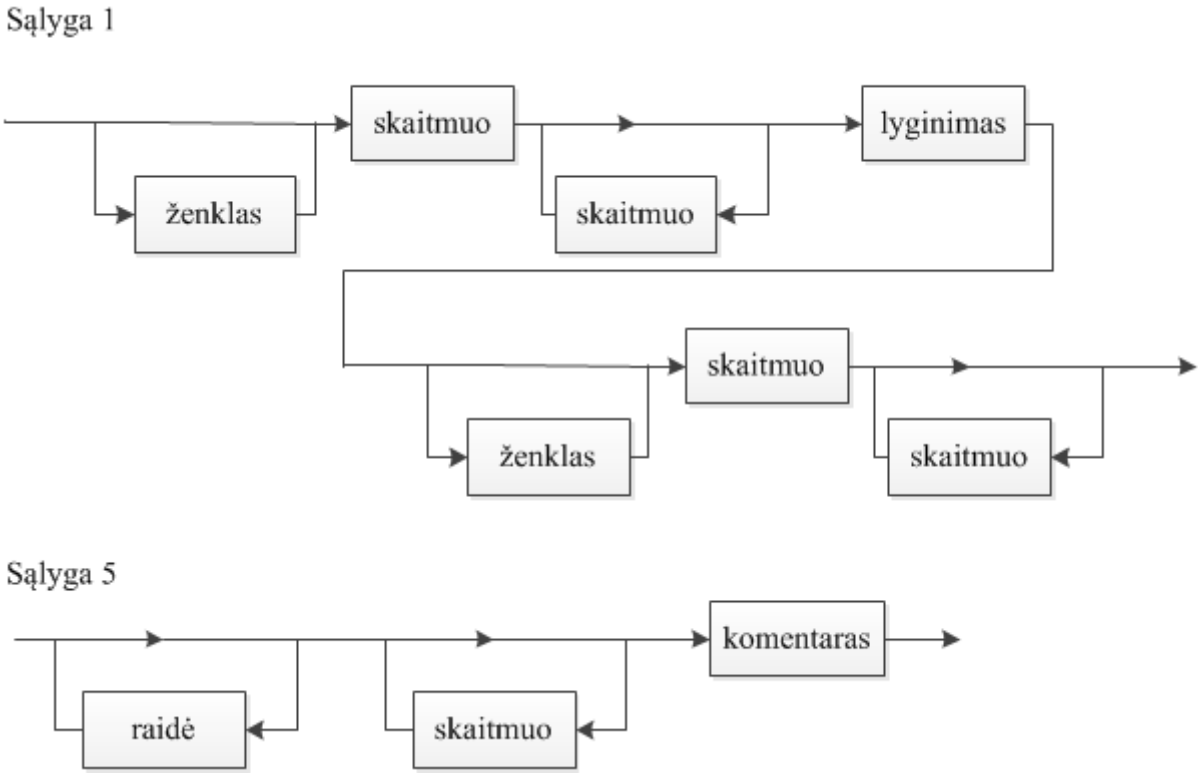
- <skaitmuo> :: = 0| 1| 2| 3| 4| 5| 6| 7| 8| 9
- <ženklas> :: = + | -
- <raidė> :: = a | ą | b | c | d | e | ę | ė | f | g | h | i | į | y| j | k | l| m | n | o | p | q | r | s | š | t | u | ų | ū | v | w | x | z | ž
- <lyginimas> :: = < | <= | = | > | >=
- <komentaras> :: = -- | /*
- <arba> :: = OR | ||
- <neleistini simboliai> :: = ‘ | “ | % | ;
- <neleistini žodžiai> :: = drop | delete | exec | union | like | select | commit

Terminaliniai elementai yra sudaryti iš galimų reikšmių, kurie yra atskirti alternatyvos simboliu. Suformavus pagrindinius terminalinius elementus galimas meta sąlygų formavimas, visos meta sąlygos sudarytos iš terminalinių elementų. Meta sąlygai negalima priskirti prieš tai aprašytas sąlygos.

4 lentelė Suformuotos pagrindinės BNF meta sąlygos naudojamos modelyje

Nr.	Sąlyga
1.	[<ženklas>] <skaitmuo> {<skaitmuo>} <lyginimas> [<ženklas>] <skaitmuo> {<skaitmuo>}
2.	<skaitmuo> {<skaitmuo>} <lyginimas> <skaitmuo> {<skaitmuo>}
3.	{<raidė>} <komentaras>
4.	{<skaitmuo>} {<raidė>} <komentaras>
5.	{<raidė>} {<skaitmuo>} <komentaras>
6.	<neleistini simboliai>
7.	<neleistini žodžiai>
8.	<arba> [<ženklas>] <skaitmuo> {<skaitmuo>} <lyginimas> [<ženklas>] <skaitmuo> {<skaitmuo>}
9.	<arba> <raidė> {<raidė>} <lyginimas> <raidė> {<raidė>}
10.	<raidė> {<raidė>} <lyginimas> <raidė> {<raidė>}

Lentelėje 4 pateiktos pagrindinės meta sąlygos, kurias naudojant bus identifikuojama potenciali ataka. Lentelės dešinėje pusėje surašyti terminaliniai simboliai, kurie ir sudaro meta sąlygą.



4 pav. Meta sąlygų schemos

5 lentelė Injekcijų atakų pavyzdžiai suformuotoms meta sąlygoms

Sąlygos nr.	Injekcinės atakos pavyzdys
1.	select t.* from lentele t where +123 = 123
2.	select t.* from lentele t where 1 = 1
3.	drop tabale lentele--
4.	drop table 123lentele--
5.	drop table lentele123--
6.	select t.* from lentele t where ('ab' = 'abc') or '1'='1--
7.	select t.* from lentele where t.vardas = ‚jonas‘ union select n.* from naudotojai n
8.	select t.* from lentele where t.reiksme = 123 or 12=12
9.	select t.* from lentele where t.reiksme = 123 or ‘abc‘=‘abc‘
10.	select t.* from lentele where ‘abc‘=‘abc‘

Lentelėje 5 pateiktos injekcinės atakos pavyzdžiai. Sąlygos numeris nusako prieš tai Lentelėje 4 suformuotos meta sąlygos numerį. Pateiktas tik vienas iš daugelio galimų atakos pavyzdžių. Meta sąlygos aprašo tam tikto tipo atakų požymius.

Panaudojus (blacklist) įvesties patikros metodiką yra sudaromas taisyklių rinkinys „juodasis“ sąrašas. Šis sąrašas turi būti nuolat pildomas, o tikrinimas atliekamas su kiekviena suformuota meta sąlyga.

Dažniausias injekcijų atakų panaudojimo principas yra loginių sąlygų naudojimas. Norint, kad duomenų bazė gražintų rezultatą reikia suformuoti tokią loginę sąlygą, kuri visada yra teigiama. Duomenų bazė neinterpretuoja perduodamų parametrų, o tiesiog gražina rezultatą. Tik DB procedūrose (stored procedures) galimas parametrų tikrinimas prieš vykdant užklausą. Galimas ir kitų funkcijų panaudojimas, pavyzdžiui, oracle „bind“ funkcija, kuri priskiria parametrus kintamiesiems, todėl naudojant šią funkciją negalimas parametrų tipų klastojimas.

2.2. BEKAUS IR NAURO FORMOS TRANSFORMAVIMAS INJEKCINIŲ ATAKŲ APTIKIMUI

Formalios kalbos aprašymai bus panaudoti prevenciniame modulyje. Suformuotos sąlygos bus talpinamos duomenų bazėje ir atvaizduojamos lentele. Lentelės sąrašas turi būti nuolat tobulinamas, pildant naujomis sąlygomis ar redaguojant jau sugalvotas.

5 pav. EBNF panaudojimas

Paveikslėlis 5 vaizduoja sąlygų formavimą, kuriomis bus tikrinamos eilutės. Jei eilutė atitiks nors vieną suformuotą sąlygą, tai tokia eilutė bus blokuojama. Pradinėje formavimo stadijoje yra formuojamas meta kalbos aprašymas, pasinaudojus aprašymais suformuojamos meta sąlygos. Iš gautų meta sąlygų yra formuojami reguliarių išraiškų masyvai.

Toks sąlygų sudarymo principas suteikia sistemai plėtojimo galimybių. Norint įvesti naują sąlygą nereikia keisti pačios programos struktūros, nes užteks papildyti sąlygų sąrašą, kurios bus saugomos duomenų bazėje. Šie veiksmai bus atliekami valdymo modulyje.

Esant meta sąlygai:

[<ženklas>] <skaitmuo> {<skaitmuo>} <lyginimas> [<ženklas>] <skaitmuo> {<skaitmuo>}

Ir įvykdžius reguliarių išraiškų transformaciją yra gaunamas reguliarių išraiškų masyvas:

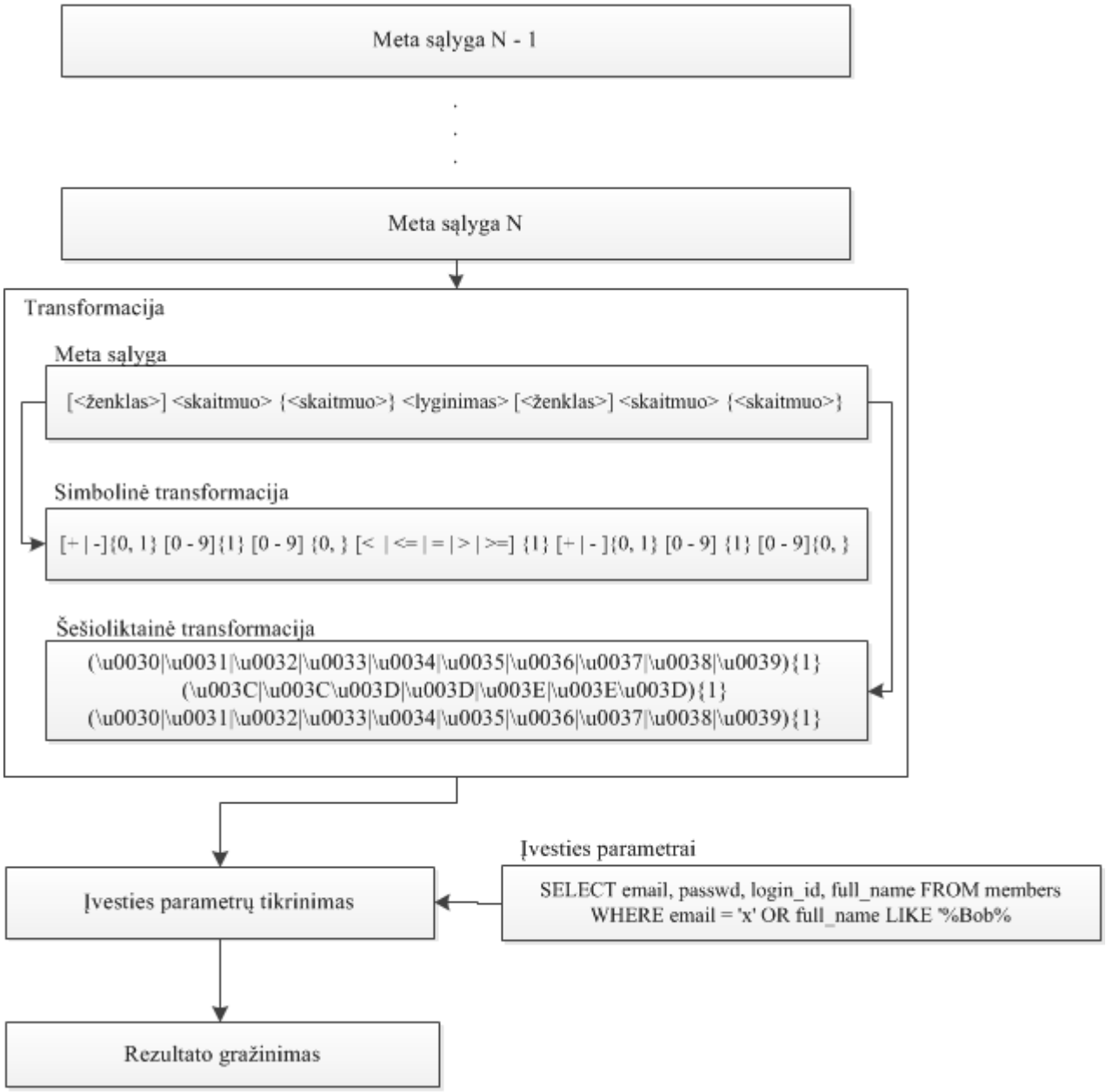
[+ | -] {0, 1} [0 - 9] {1} [0 - 9] {0, } [< | <= | = | > | >=] {1} [+ | -] {0, 1} [0 - 9] {1} [0 - 9] {0, }

Gaunamas simbolinis masyvas, galimos transformacijos ir į kitas skaičiavimo sistemas. Turint tokio tipo masyvą galima patikrinti bet kokią eilutę, ar ji atitinka suformuotas reguliarias išraiškas. Jei eilutėje bus parametrai „1=1“, „1=+1“ ar „-123<+123“, tuomet tokios eilutės bus atpažintos kaip kenksmingos.

Sistema gavusi naudotojo įvesties parametrus gauna sąlygas iš duomenų bazės, jas transformuoja ir lygina su įvestimi.

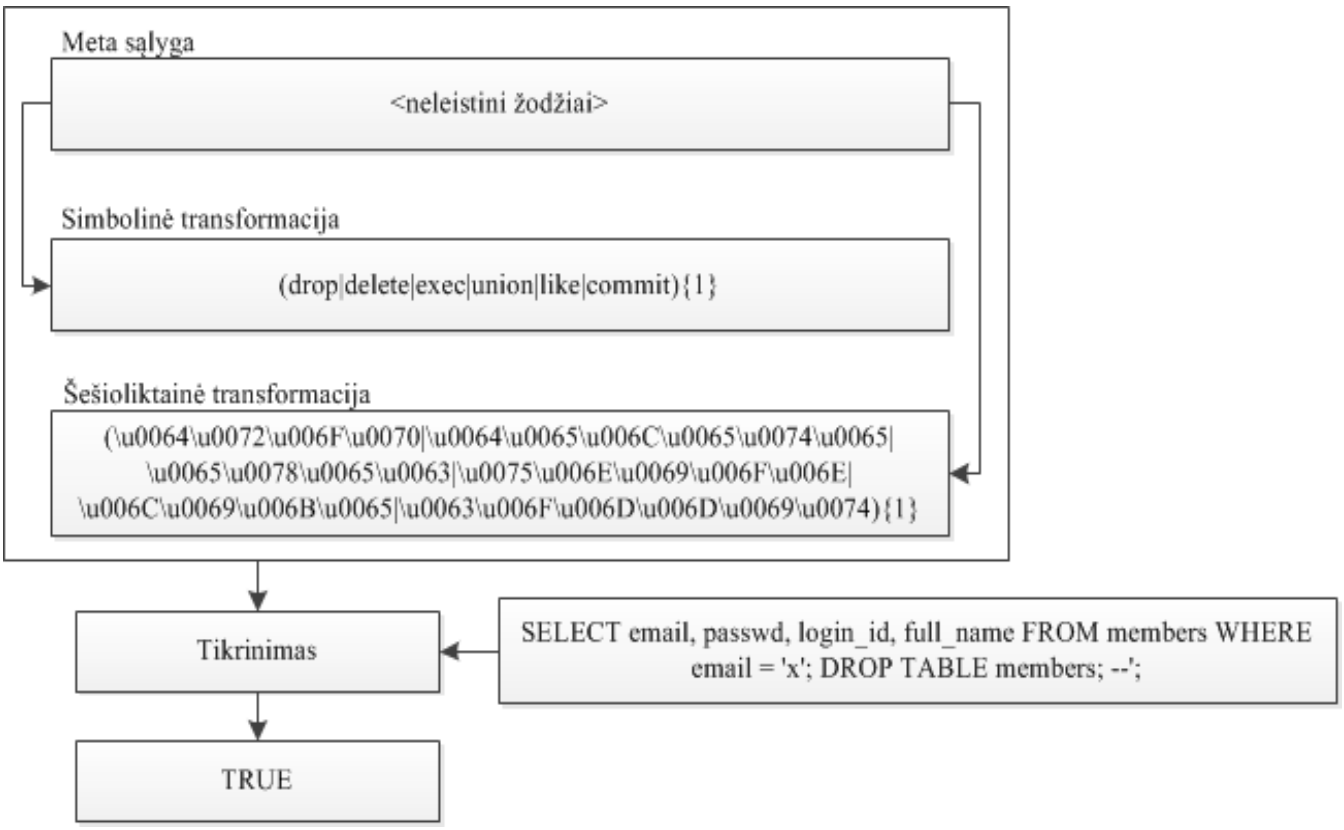
Perduodami įvesties parametrai į funkciją. Toliau vykdoma transformacija ir tikrinimas. Visos meta sąlygos yra transformuojamos į reguliarių išraiškų masyvus.

Meta sąlygų transformacijos:



6 pav. Sąlygų transformacijos bendra schema

Tikrinimas yra nutraukiamas, kai randama nors viena atitiktis. Jei nei vien sąlyga neatitinka įvesties parametrų, tuomet yra gražinamas neigiamas rezultatas. Transformacijos yra vykdomos, kol visos sąlygos yra paverčiamos į reguliarias išraiškas.



7 pav. Sąlygų transformacijos pavyzdys

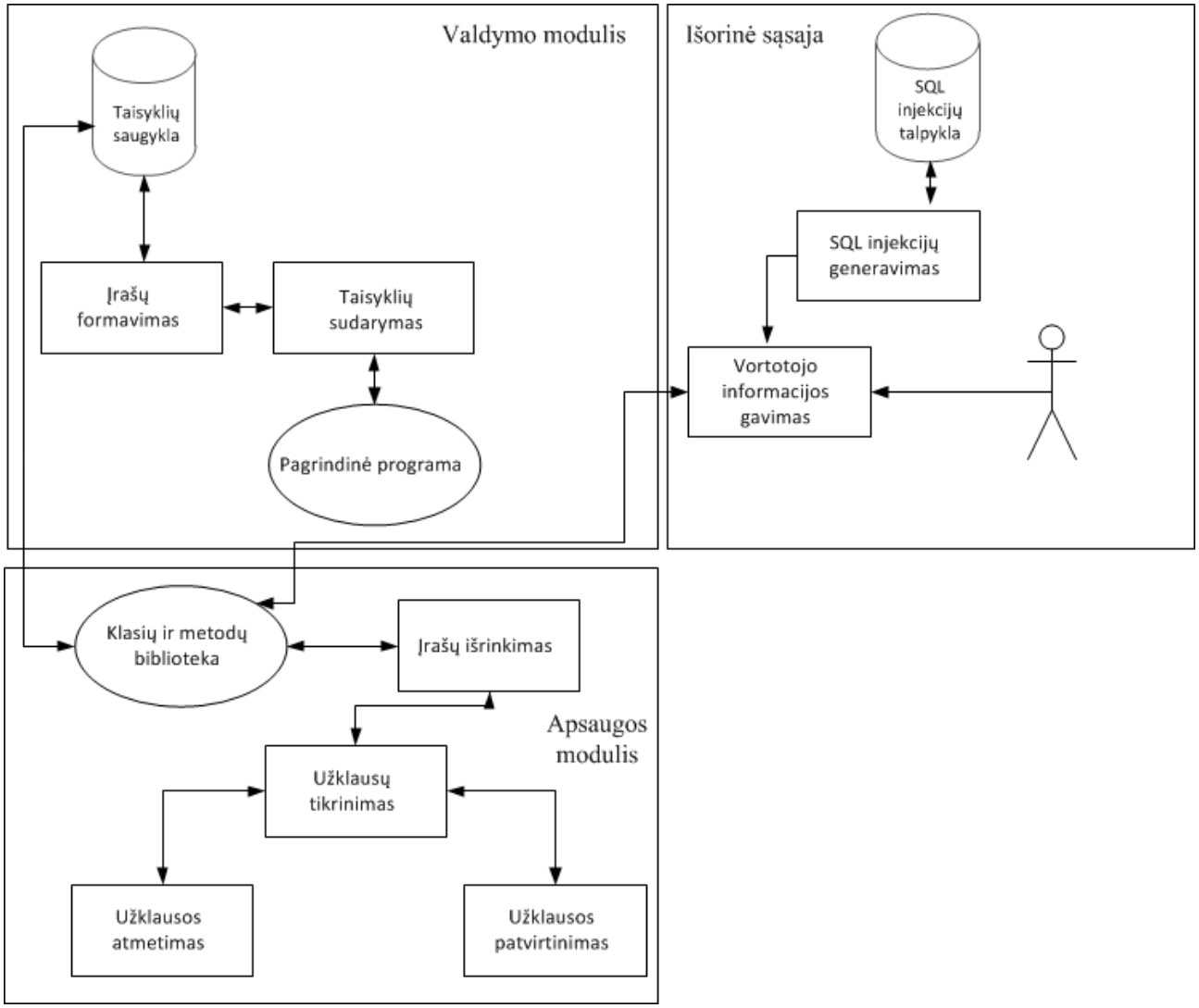
Paveikslėlyje 7 pavaizduota sąlygos transformacija į reguliarių išraiškų masyvą. Pavertus sąlygas į reguliarias išraiškas yra vykdoma tikrinimo sąlygą. Tikrinimas vykdomas su parametrais: „SELECT email, passwd, login_id, full_name FROM members WHERE email = 'x'; DROP TABLE members; --';“. Ši užklausa yra kenksminga, nes užklausa yra modifikuota. Šie įvesties parametrai atitiko vieną aprašą iš aprašytų injekcinių atakų šablono, todėl metodas gražina teigiamą reikšmę, tai nusako, kad užklausa yra potencialiai kenksminga.

Suformuotos sąlygos gali nebūtinai aptikinėti SQL injekcijų atakas, tačiau galimas ir bet koks xml, html ar kito dokumento įvertinimas, ar dokumentas atitinka suformuotas taisykles, ar neturi atakos požymių. Įvertinami gali būti tie parametrai, kurie paverčiami simbolių masyvais.

2.3. INTERAKTYVIOS SAUGOS SISTEMOS PROTOTIPO ARCHITEKTŪRA

Šiame skyriuje pateikiamos realizuojamos programos aprašymas. Pateikiamos naudojamos funkcijos, modulis paruošiamas tyrimo atlikimui, nustatomi duomenų kokybės reikalavimai, siekiant geriausių eksperimento rezultatų. Eksperimento programos realizacija atlikta pagal projekto aprašymą. Išpildyti visi funkciniai ir nefunkciniai reikalavimai sistemai. Sistema atlieka du pagrindinius veiksmus: taisyklių sudarymo ir parametrų analizavimo. Realizuotas algoritmas, kuris pateikia atsakymą ar perduoti parametrai turi požymių pagal suformuotas taisykles.

Paveikslėlyje 8 pavaizduota bendra dviejų modulių schema, taip pat ir sistemos sąsaja su išoriniais duomenimis.



8 pav. Interaktyvios saugos sistemos prototipo architektūra

Bendrą struktūrą sudaro trys pagrindiniai procesai: duomenų gavimas, taisyklių sudarymas ir parametrų tikrinimas. Yra naudojama bendra taisyklių duomenų bazė. Duomenų bazė yra pildoma valdymo modulio pagalba. Galima sudaryti bet kokį skaičių taisyklių ir įrašyti jas į saugyklą.

Kiekvienas taisyklių elementas yra griežtai apibrėžtas ir taip pat saugomas duomenų bazėje. Apsaugos modulis tikrina gautus duomenis iš išorinės sąsajos. Duomenys gali būti gauti keliais būdais. Apsaugos modulis taip pat naudoja taisyklių duomenų bazę. Iš gautų duomenų yra suformuojamos reguliarių išraiškų transformacijos. Transformacijos yra dviejų tipų: simbolinės ir šešioliktainės. Kiekviena gauta eilutė tikrinama ar yra tenkinamos transformuotų reguliarių išraiškų taisyklės. Aptikus eilutę, kuri tenkina sąlygas yra grąžinamas teigiamas rezultatas.

Funkciniai reikalavimai

Šioje dalyje suformuluoti reikalavimai eksperimento programai. Įvardinta, kokias funkcijas (galimybes) ji turi vykdyti, kad sistema atpažintų kenksmingus įvesties parametrus.

Funkciniai reikalavimai, kurie būtini kuriamai sistemai:

- ✓ Įeinančių duomenų tikrinimas nuo gerai žinomų injekcijų atakų;
- ✓ Meta sąlygų transformacija į kelių tipų reguliarių išraiškų masyvus;
- ✓ Sąlygų, pagal kurias identifikuojama ataka, praplėtimas, papildymas, naikinimas;
- ✓ Naujų terminalinių elementų sudarymas;
- ✓ Tam tikra reakcija į potencialią ataką;
- ✓ Informacija sistemos naudotojui (apie užfiksuotas atakas, reakcijos laiką ir kt.);
- ✓ Meta kalbos panaudojimas sudarinėjant sąlygas;
- ✓ Rezultatų, sistemos veiklos įrašymas į failą;

Nefunkciniai reikalavimai

Šioje dalyje išdėstyti sistemos reikalavimai taikomajai eksperimento programai, kad sistema veiktų ir būtų gauti eksperimento rezultatai.

Reikalavimai programinei ir techninei įrangai

Norint atlikti eksperimentą reikalingas kompiuteris su „Windows“ operacine sistema. Sistemoje turėtų būti įdiegtas „MS Visual Studio“ (rekomenduojama 2010 metų versija) programinis paketas su .NET Framework 4 versija. Esant spartesniam kompiuteriui eksperimento rezultatai gaunami greičiau.

Reikalavimai techniniai įrangai:

- Kompiuteris su 1.6GHz taktinio dažnio procesoriumi (gali būti ir spartesnis)
- 1 GB (32 Bit) arba 2 GB (64 Bit) virtualios atminties (RAM)

Nefunkciniai reikalavimai eksperimento programai

- ✓ Didelė įeinančių parametrų analizavimo sparta;
- ✓ Saugi sistema;
- ✓ Nesudėtinga programos integracija į projektą;
- ✓ Lengvai suprantami eksperimento rezultatai;
- ✓ Nesudėtingas terminalinių elementų kūrimas ar ištrynimas;
- ✓ Nesudėtingas meta sąlygų formavimas;
- ✓ Vartotojo grafinė sąsaja turėtų būti aiški ir suprantama sistemos naudotojui;
- ✓ Pati sistema turėtų būti saugi ir apsaugota nuo nenumatytų situacijų. Atsiradus sistemos klaidoms apie tai turi būti informuojamas naudotojas.

Sistemos sudedamosios dalys

Eksperimento programa sudaryta iš kelių modulių. Yra du pagrindiniai moduliai apsaugos modulis ir valdymo modulis. Pagrindinė užduotis yra aptikti ir informuoti apie kenksmingą ar netinkamą įvesties parametrą.

Apsaugos modulis. Tai modulis, kuris analizuoja gautas eilutes. Šis modulis tikrina, ar simbolių masyvas tenkina suformuotas sąlygas, bei rezultatus įrašo į failą. Modulis gražina loginį kintamąjį „true“ arba „false“.

Galimos modulio reakcijos:

- Įvesties parametras atitinka suformuotas meta sąlygas (atitinka atakos požymius);
- Įvesties parametras neatitinka suformuotų meta sąlygų (įvesties parametras saugus);

Valdymo modulis. Šiame modulyje yra pagrindinė vartotojo sąsaja darbui su duomenų baze. Visos sąlygos yra formuojamos valdymo modulyje, naudojant įvedimo laukus ir terminalinius elementus. Sąlygos yra sudarinėjamos naudojant terminalinius elementus, iš kurių yra formuojami reguliarių išraiškų masyvai.

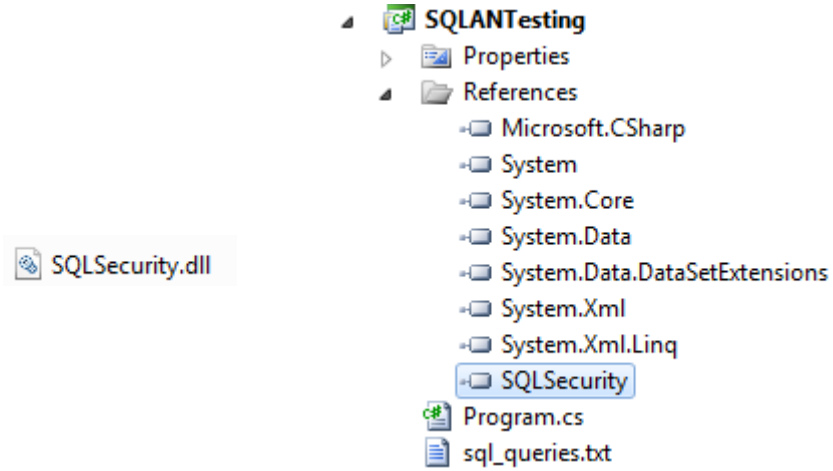
2.3.1. INTERAKTYVIOS SAUGOS SISTEMOS PROTOTIPO PREVENCINIS MODULIS

Reikalavimai duomenims

Duomenys priimami eilutės tipo, tai gali būti bet koks simbolių masyvas su galutine „null“ reikšme. Didžiausias simbolių skaičius gali būti ne didesnis 1000. Duomenys priimami nešifruoti.

Apsaugos modulio integracija į sistemas

Modulis turi būti integruotas į sistemos projektą, kaip saugumą užtikrinanti funkcija. Modulio išeities tekstai nėra laisvai prieinami, todėl jų redaguoti negalima. Galima tik papildyti saugos modulyje naudojamas taisykles. Atnaujinus taisykles parametrų analizės algoritmas nesikeičia. Modulio veikimo principas yra panašus į antivirusinių programų veikimą. Atnaujinama yra tik duomenų bazė, o ne algoritmas. Sukurta analizatoriaus biblioteka yra laisvai naudojama pagal poreikius ir sistemos architektūrą. Biblioteka gali būti panaudota tik .NET architektūros sistemose.



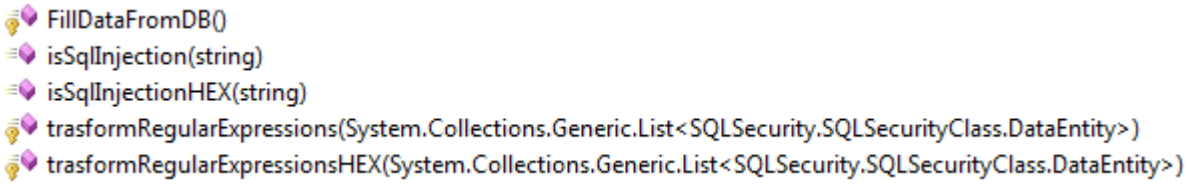
9 pav. Modulio integracija į sistemas

Sistemos veikimas nenaudojant prevencinio modulio

Sistemos naudotojai įvedinėja įvairiausią informaciją, kurią apdoroja sistema. Netikrinant įvesties galimi kenksmingi veiksmai sistemai. Nenaudojant tokio tipo prevencinio modulio įvestis yra nepakankamai analizuojama, todėl padidėja rizika sistemos pažeidžiamumui injekcinėms atakoms. Sistema dažniausiai priima įvestus parametrus ir iš jų suformuoja užklausas, kurios yra siunčiamos į duomenų bazę (galimi ir kiti sistemos įvesties panaudojimo atvejai). Prevencinio modulio pagrindinis uždavinys būtų padidinti įvestų parametrų patikimumą.

Apsaugos modulio bibliotekos metodai

Įtraukus biblioteką į projektą galima pasinaudoti visais bibliotekos atributais, metodais ir klasėmis.



10 pav. Apsaugos modulio bibliotekos metodai

Galime pastebėti, kad objektas turi kelis metodus. Žyma su raktu nusako tai, kad metodas yra apsaugotas ir neprieinamas iš išorės, tai vidinis objekto metodas. Metodas be rakto žymos yra prieinamas už objekto ribų.

1. Duomenų struktūros užpildymas naudojant duomenų bazę.
2. Metodas, kuriame realizuotas analizės algoritmas.
3. Taip pat užklausų analizės metodas, kuris naudoja šešioliktainės reguliarias išraiškas.
4. Reguliarių išraiškų transformacijos metodas. Transformuojama į simbolines reguliarias išraiškas.
5. Reguliarių išraiškų transformacijos metodas. Transformuojama į šešioliktainės reguliarias išraiškas

Pagrindiniai objekto metodai yra loginiai kintamieji, kurie gražina dvi reikšmes taip arba ne. Naudojant šiuos metodus reikia perduoti eilutės tipo kintamąjį, tai gali būti ne tik duomenų bazės užklausos, tačiau ir bet koks parametras įvestas naudotojo. Galimas xml struktūros analizavimas ar nėra kenksmingų parametrų. Šiuo metodu galima analizuoti bet kokio tipo eilutes parametrus ir nustatyti ar eilute atitinka suformuotas taisykles.

Parametrų tikrinimo algoritmas

Paveikslėlyje 11 pavaizduotas gautų parametrų tikrinimas. Gražinamas teigiamas arba neigiamas atsakymas.

11 pav. Įvesties parametrų tikrinimo algoritmas

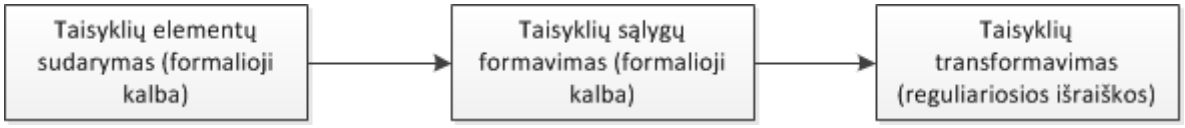
Algoritmas pradeda veikti, kai gaunami įvesties parametrai (eilutės tipo kintamasis). Meta sąlygos turi tam tikrą duomenų struktūra ir talpinamos operatyviojoje atmintyje. Užpildžius duomenų struktūra duomenimis iš duomenų bazės vykdoma transformacija į reguliarias išraiškas. Galutinai suformavus reguliarių išraiškų masyvus yra tikrinama įvestis. Rezultatas yra gražinamas bei įrašomas į failą.

2.3.2. BEKAUS IR NAURO FORMOS TAISYKLIŲ FORMAVIMO MODULIS

Šiame skyriuje aprašytas taisyklių sudarymo modulio veikimas. Aprašytas sudarymo algoritmas.

Realizuoto modulio atliekamos funkcijos

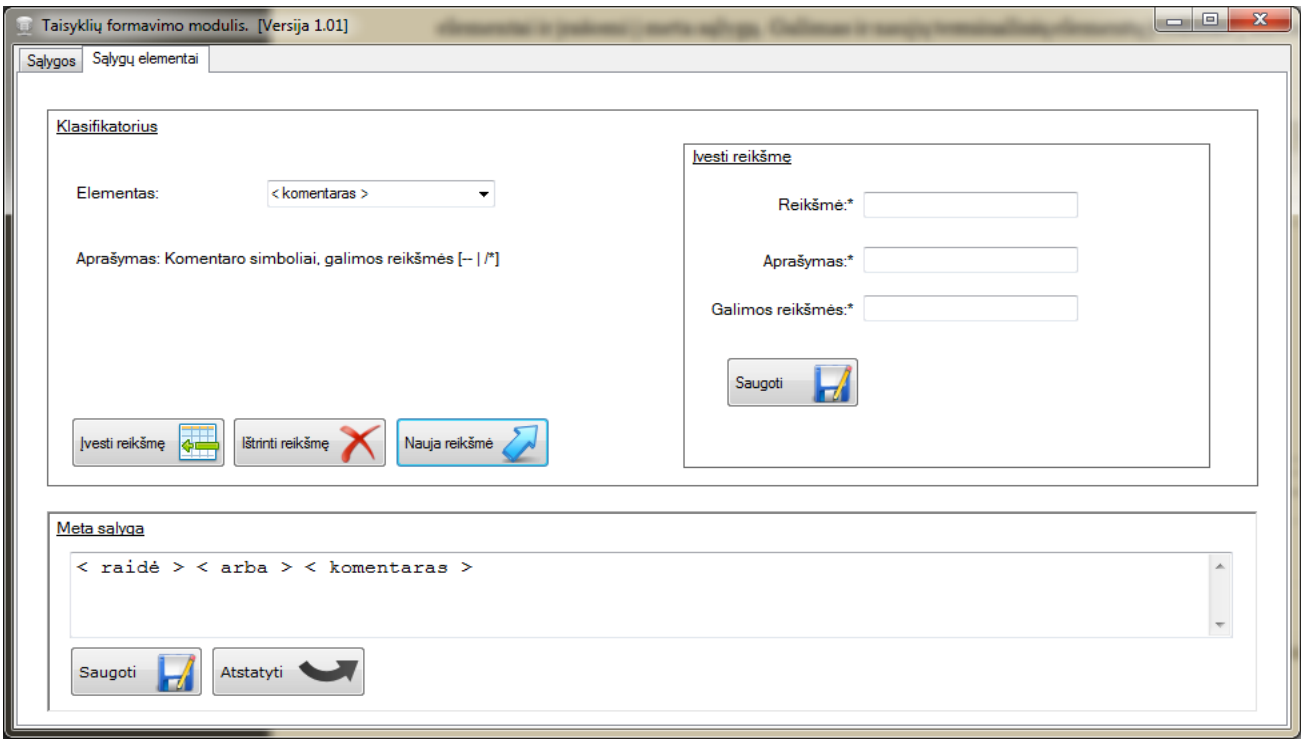
Pagrindė šio modulio funkcija yra sudaryti tam tikras taisykles, kurias naudos apsaugos modulis. Pasinaudojant moduliu yra sudaromi taisyklių elementai, kurie apibrėžia vieną taisyklės elementą, tai gali būti skaičius ar raidė, ar kiti elementai. Sudaryta taisyklė iš suformuotų formalios kalbos elementų yra išsaugoma duomenų bazėje. Apsaugos modulis naudoja suformuotas taisykles, jas transformuoja į reguliarias išraiškas. Naudojant reguliarių išraiškų masyvus yra tikrinami visi gauti eilutes parametrai. Paveikslėlis 12 vaizduoja taisyklių panaudojimo ciklą.



12 pav. Injekcinių atakų aprašo taisyklių naudojimo ciklas

Realizuoto modulio taisyklių formavimo grafinė sąsaja

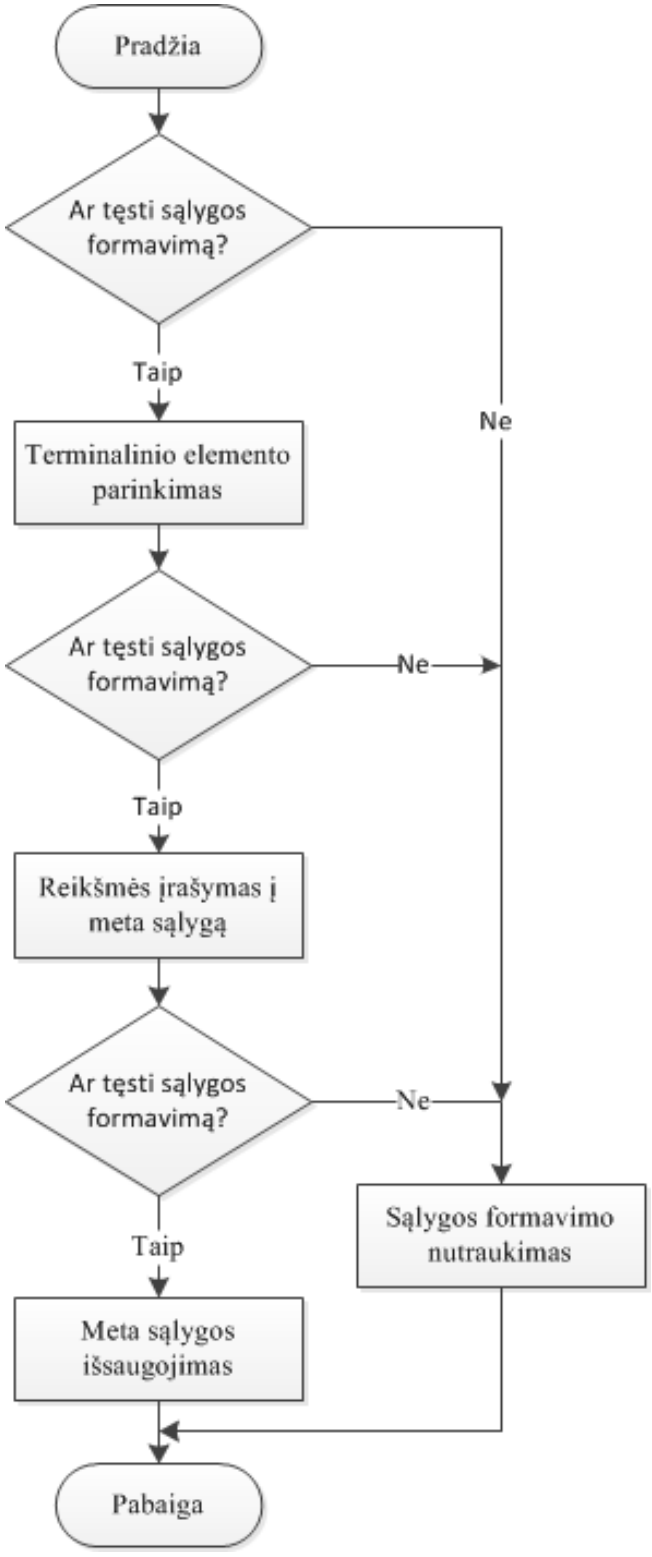
Paveikslėlis 13 vaizduoja taisyklių sudarymo langą. Iš sąrašo pasirenkami terminaliniai elementai ir įrašomi į meta sąlygą. Galimas ir naujų terminalinių elementų įvedimas į sistemą. Suformavus norimą meta sąlygą ji yra išsaugoma duomenų bazėje.



13 pav. Terminalinių elementų sudarymo langas

Taisyklių sudarymo algoritmas

Paveikslėlyje 14 pavaizduotas taisyklių sudarymo algoritmas. Naudojant šį algoritmą yra sudaromos taisyklės, kurios saugomos duomenų bazėje. Suformuotas taisyklės naudoja prevencinis modulis.

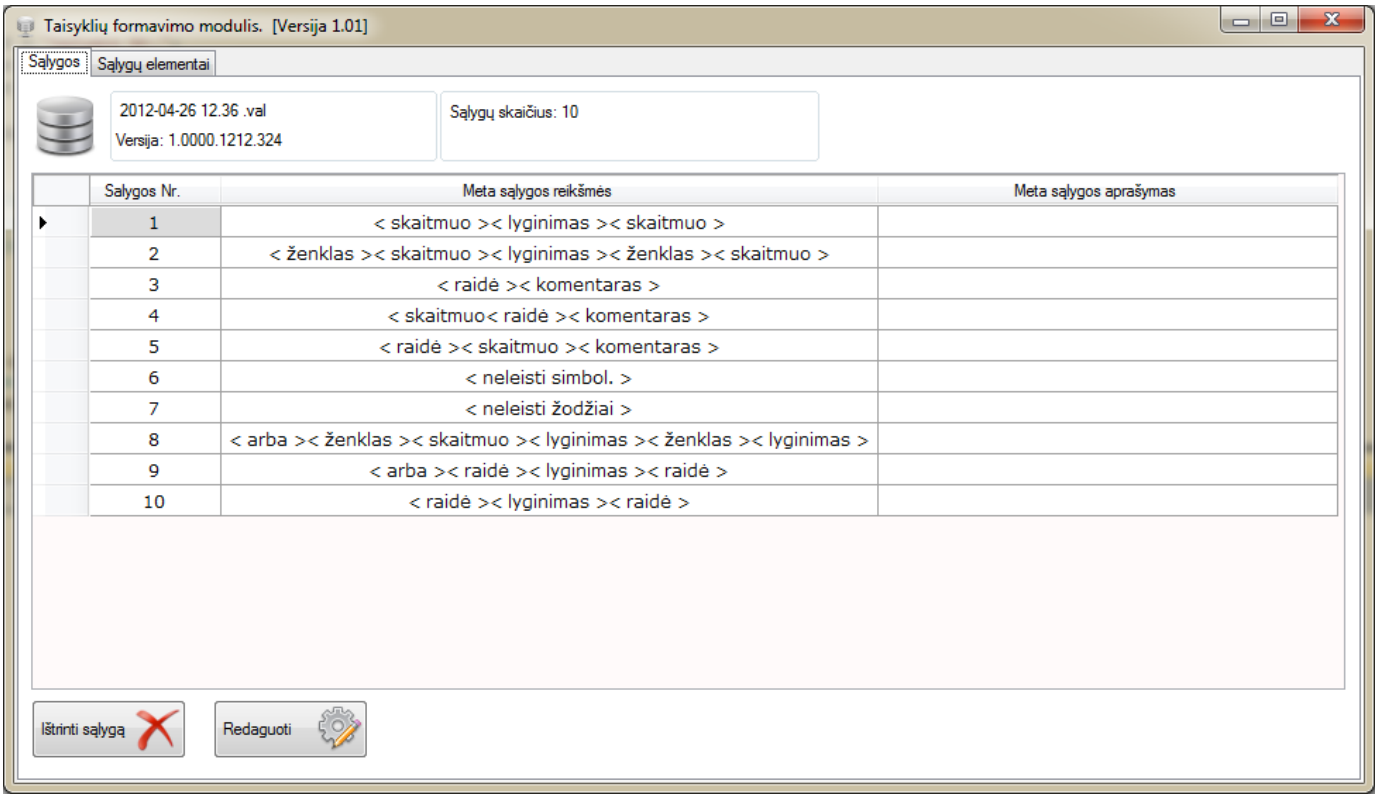


14 pav. Meta sąlygų sudarymo algoritmas

-

Meta sąlygų sąrašo langas

Paveikslėlyje 15 vaizduojamas sudarytų taisyklių sąrašas, kuriame matome sąlygos numerį, formalios kalbos taisyklę ir aprašymą. Šį sąrašą galima vadinti (blacklist), kuris yra nuolat pildomas ir atnaujinamas. Galimas kiekvienos sąlygos panaikinimas ar redagavimas.



15 pav. Meta sąlygų sąrašo langas

2.4. IŠVADOS

- ✓ Pasiūlytas Bekaus ir Nauro formos taikymas injekcijų atakų sintaksei aprašyti.
- ✓ Pasiūlytas ir praktiškais realizuotas Bekaus ir Nauro formos transformavimo į reguliarias išraiškas algoritmas.
- ✓ Suprojektuotas ir realizuotas interaktyvus Bekaus ir Nauro formos transformacijų į reguliarias išraiškas programinis modulis.
- ✓ Suprojektuotas ir realizuotas interaktyvios saugos sistemos prototipas apsaugai nuo injekcinių atakų.
- ✓ Sukurtos sistemos prototipas suteikia naudotojui galimybę, atsiradus naujai injekciniai atakai, savarankiškai ir interaktyviai sukurti injekcinės atakos aprašus naudojant Bekaus ir Nauro formą. Naujos injekcinės atakos aprašo įterpimas į apsaugos sistemą nereikalauja programavimo darbų.

3. INTERAKTYVIOS SAUGOS SISTEMOS APSAUGAI NUO INJEKCINIŲ ATAKŲ EKSPERIMENTINIS TYRIMAS

Šiame skyriuje išnagrinėsime programos rezultatus, kurie buvo gauti atlikus tyrimą. Programos prototipas priima eilutes, jų masyvą. Eksperimento tikslas išnagrinėti tikrinimo algoritmo greitaveiką, nes pagrindinis šio prototipo panaudojimo kriterijus yra tai, ar tikrinimo sąlygos atliekamos pakankamai greitai. Reikia išnagrinėti kaip tai gali įtakoti visos sistemos darbą, koks bus vėlinimo laikas.

Eksperimentas atliekamas naudojant DELL Latitude E6500 nešiojamą kompiuterį. Kompiuterio sudedamųjų dalių charakteristikos: procesorius Intel Core 2 Duo P9600 2.66 GHz , darbinė atmintis 4 GB RAM, operacinė sistema Windows 7 Enterprise N 64 bit.

3.1. EKSPERIMENTO METODIKA

Tyrimo duomenys – tekstiniai failai, kuriuose saugomi tikrinimo parametrai. Failai skirstomi į tam tikrą eilučių skaičių turinčius failus. Eilutės perduodamos nešifruotos atvirus tekstu.

Eksperimentas atliekamas naudojant dvi tikrinimo funkcijas. Viena funkcija atlieka simbolinę tikrinimo sąlygų transformaciją, kita funkcija transformuoja sąlygas į šešioliktaines išraiškas.

Tam, kad atlikti eksperimentą buvo sukurta testinė sistema, kuri įvertina kreipinių į funkcijas laikus ir gražinamą rezultatą. Į testinę sistemą buvo įtraukta sukurta dll biblioteka su joje esančiais metodais. Eksperimento metu testinės sistemos atliekami veiksmai:

- ✓ Turint tikrinamų parametrų masyvą yra vykdomas ciklas.
- ✓ Tikrinami visi tekstiniai failai esantys projekto kataloge
- ✓ Tikrinami visi parametrai masyve.
- ✓ Laikas pradedamas fiksuoti pirmo ciklo iteracijos metu, duomenų įrašymo į masyvo laikas nėra vertinamas.
- ✓ Laikas stabdomas paskutinio ciklo iteracijos metu.
- ✓ Laikas matuojamas milisekundėmis.
- ✓ Pateikti rezultatai yra tikslūs, nes vertinama tik bibliotekos metodų skaičiavimų trukmė.
- ✓ Atlikti skaičiavimai įrašomi į failą.

Eksperimento metu svarbu pasiekti kritinės skaičiavimo ribas, didžiausią vėlinimą. Taip pat yra svarbu nustatyti, koks laikas užtrunkamas apdoroti 1 KB informacijos, naudojant skirtingų dydžių failus.

Eksperimento metu bus naudojami tekstiniai failai. Informacija apie juos pateikta lentelėje 6. Atliekant eksperimentą bus naudojama 12 failų, kuriuose bus skirtingas tikrinamų parametrų skaičius. Mažiausias parametrų skaičius 5000, failo dydis 253 KB, o didžiausią parametrų skaičių turintis failas bus su 100000 parametrų ir 5073 KB dydžio. Failą naudojamą eksperimente apibūdina: jo dydis kilobaitais, eilučių (tikrinamų parametrų) skaičius ir simbolių skaičius.

6 lentelė Tyrimo duomenų failai

Pavadinimas	Failo dydis, KB	Eilučių skaičius	Simbolių skaičius
sql_queries_5000.txt	253	5000	249708
sql_queries_10000.txt	506	10000	497624
sql_queries_15000.txt	761	15000	749124
sql_queries_20000.txt	1011	20000	995248
sql_queries_30000.txt	1520	30000	1496419
sql_queries_40000.txt	2022	40000	1990479
sql_queries_50000.txt	2542	50000	2502372
sql_queries_60000.txt	3044	60000	2996469
sql_queries_70000.txt	3551	70000	3495912
sql_queries_80000.txt	4058	80000	3995328
sql_queries_90000.txt	4566	90000	4494744
sql_queries_100000.txt	5073	100000	4994160

Visi failai bus patalpinti viename projekto kataloge. Testinė programa pradėjusi darbą skaitys visus kataloge patalpintus tekstinius failus. Kai kurios suformuotos eilutės turės injekcinių atakų požymių. Programa (apsaugos modulis) aptikusi kenksmingą ataką informaciją įrašys į įvykių failą ir nutrauks tikrinimo algoritmą.

Tam, kad skaičiavimai būtų tikslesni, skaičiavimai bus vykdomi tris kartus, o į duomenų lentelę bus įrašoma gautų rezultatų vidurkiai. Atlikus vieno tipo visų parametrų tikrinimus yra fiksuojamos laiko reikšmės testinėje programoje. Šios laiko reikšmės įrašomos į duomenų failą.

Atlikus eksperimentą yra gaunamas tekstinis failas „Ataskaita.txt“, kuriame patalpinta visų skaičiavimų įvertinimo rezultatai (pavyzdyje pateiktas tik failo fragmentas).

```
Failas 'C:\Users\mzu\Documents\Visual Studio
2010\Projects\SQLAN\SQLANTesting\bin\Release\queries\sql_queries_10000.txt'.
Užklausų skaičius 10000
Simbolinės: kenksmingos 6593 (65,93%)
HEX: kenksmingos 6647 (66,47%)
Laikas:
1) Simboline: 13503 ms.
1) HEX: 11898 ms.
2) Simboline: 13699 ms.
2) HEX: 11536 ms.
3) Simboline: 14149 ms.
3) HEX: 11142 ms.
Failas 'C:\Users\mzu\Documents\Visual Studio
2010\Projects\SQLAN\SQLANTesting\bin\Release\queries\sql_queries_100000.txt'.
Užklausų skaičius 100000
Simbolinės: kenksmingos 55713 (55,71%)
HEX: kenksmingos 56980 (56,98%)
Laikas:
1) Simboline: 317447 ms.
1) HEX: 119639 ms.
2) Simboline: 352398 ms.
2) HEX: 110506 ms.
3) Simboline: 352110 ms.
3) HEX: 112442 ms.
...
```

Pirmoje eilutėje yra nurodytas failas, kuriame buvo tikrinimo parametrai. Toliau yra pateiktas užklausų (tikrinamų parametrų) skaičius ir kiek iš tikrintų parametrų buvo įvertinta, kaip potencialiai kenksmingos užklausos, skaičius. Atlikus vieną skaičiavimo ciklą yra išvedami skaičiavimo laikai milisekundėmis. Surinkti duomenys bus patalpinti į duomenų lenteles.

Apsaugos modulio įvykio failo informacija:

```
--------2012-04-26 16.47.43--------
Statement: SELECT * from KL_PARAMETRAI
Regular expression: None
Result: False
--------2012-05-04 11.32.21--------
Statement: ') or ('1'='1--
Regular expression: (\u2018|\u201C|\u0025|\u003B|\u0027){1}
Result: True
--------2012-05-04 11.32.21--------
Statement: SELECT t.* FROM USERS t WHERE +1=+1 /**/;
Regular expression:
(\u002B|\u002D){1}(\u0030|\u0031|\u0032|\u0033|\u0034|\u0035|\u0036|\u0037|\u003
8|\u0039){1}(\u003C|\u003C\u003D|\u003D|\u003E|\u003E\u003D){1}(\u002B|\u002D){1
}(\u0030|\u0031|\u0032|\u0033|\u0034|\u0035|\u0036|\u0037|\u0038|\u0039){1}
Result: True
--------2012-05-12 15.51.14--------
Statement: SELECT * from KL_PARAMETRAISELECT ProductName, Product Description
FROM Products WHERE ProductNumber = 123 OR 1=1;
Regular expression:
(0|1|2|3|4|5|6|7|8|9){1}(<|<=|=|>|>=){1}(0|1|2|3|4|5|6|7|8|9){1}
Result: True
--------2012-05-12 15.51.14--------
Statement: admin' #
Regular expression: (`|"|%|;|'){1}
Result: True
```

Į įvykio failą yra įrašoma įvykio data, kada buvo kviečiama tikinimo funkcija. „Statement“ dalyje matome tikrinamą eilutę. „Regular expression“ dalyje yra išvedama „None“ jei nei vien sąlyga nebuvo tenkinama. Jei buvo tenkinama sąlyga, tai išvedama reguliarios išraiškos reikšmė. „Result“ eilutėje išvedamas gražinamas rezultatas „True“ arba „False“ (priklausomai ar buvo tenkinama tikrinimo sąlyga).

3.2. EKSPERIMENTO REZULTATAI

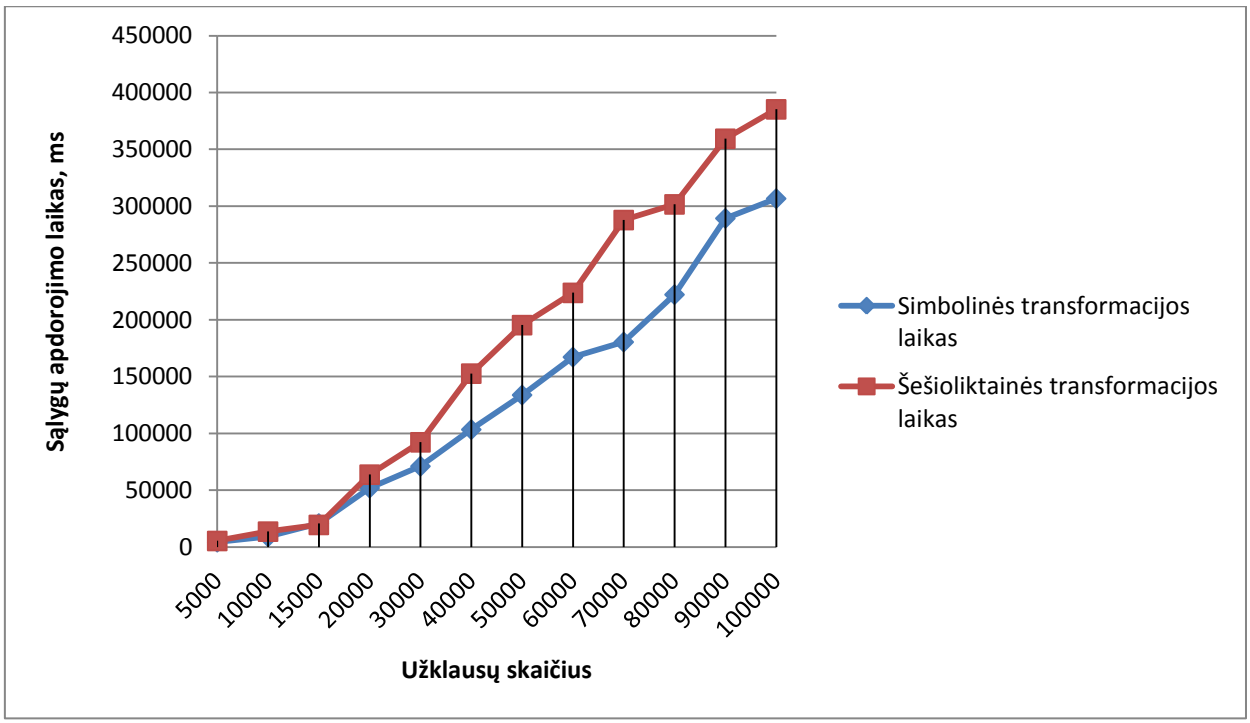
Eksperimento metu gauti skaitiniai tyrimo duomenys. Buvo suskaičiuota visų failų, naudojančių reguliarių išraiškų transformacijų laikai. Transformacijos buvo vykdomos simbolinės ir šešioliktainės. Tyrimo rezultatų duomenys pateikti lentelėje 7.

7 lentelė Atliktų transformacijų laikai

Užklausų skaičius	**Simbolinės transformacijos laikas, ms**	**Šešioliktainės transformacijos laikas, ms**
5000	4320	5397
10000	7104	13614
15000	17780	19531
20000	52013	63765
30000	71177	92255
40000	103451	152700
50000	133861	195374
60000	167265	223778
70000	180504	287883
80000	222246	301664
90000	289347	359412
100000	306749	385315

Atlikus tyrimą, galima pastebėti, kad šešioliktainės transformacijos yra vykdomos ilgiau nei simbolinės. Vykdymo laikas išlieka panašus esant iki 20000 ciklo iteracijų. Kai užklausų skaičius yra 5000 simbolinės transformacijos laikas trunka 4,32 sekundės, o šešioliktainės transformacijos trunka 5,397 sekundės. Analizuojant didesnius duomenų failus laikai sparčiai didėja. Esant 100 tūkstančių užklausų visas transformacijų laikas simbolinių užklausų yra 5,112 minutės, o šešioliktainių transformacijų 6,755 minutės.

Lentelėje pateikti laikai yra apskaičiuoti sumuojant reikalingą transformacijos laiką norint įvertinti tam tikrą užklausų ir duomenų kiekio parametrų skaičių. Šie laikai buvo apskaičiuoti apsaugos modulyje.



16 pav. Reguliarių išraiškų transformacijos palyginimas

Jei reikalingas spartesnis funkcijų reakcijos laikas rekomenduojama naudoti simbolinės reguliarių išraiškų transformacijos. Transformuojant į šešioliktainės išraiškas gaunamas sudėtingesnių ASCII lentelės simbolių tikslesnis įvertinimas. Įvairūs simboliai („\“, „*“) yra reguliarių išraiškų kalbos sudedamosios dalys, todėl yra tikimybė, kad reguliari išraiška bus suprantama ir įvertinama klaidingai. Simbolinio tipo išraiškos injekcinių požymių įvertinimo metu ne visi parametrai bus įvertinti tinkamai.

Iš grafiko galime pastebėti, kad iki 20 tūkstančių užklausų transformacijų laikas yra panašus ir jos atliekamos sugaištant apie 50 sekundžių. Toliau didėjant užklausų skaičiui atskirtis tarp transformacijų laiko didėja. Ties 30 tūkstančių užklausų laiko skirtumas yra 21 sekundė. Didėjant užklausų skaičiui laiko skirtumas nėra proporcingas, galime pastebėti taškų, kuriuose nors ir užklausų skaičius padidėjo, tačiau laiko skirtumas sumažėjo. Tolygus laiko skirtumo didėjimas yra iki 50 tūkstančių užklausų. Ties 50 tūkstančių užklausų laiko skirtumas yra 62 sekundės, o esant 60 tūkstančių užklausų laiko skirtumas yra 57 sekundės. Didžiausias laiko skirtumas pastebimas ties 70 tūkstančių užklausų, jis yra apie 1,7 minutės.

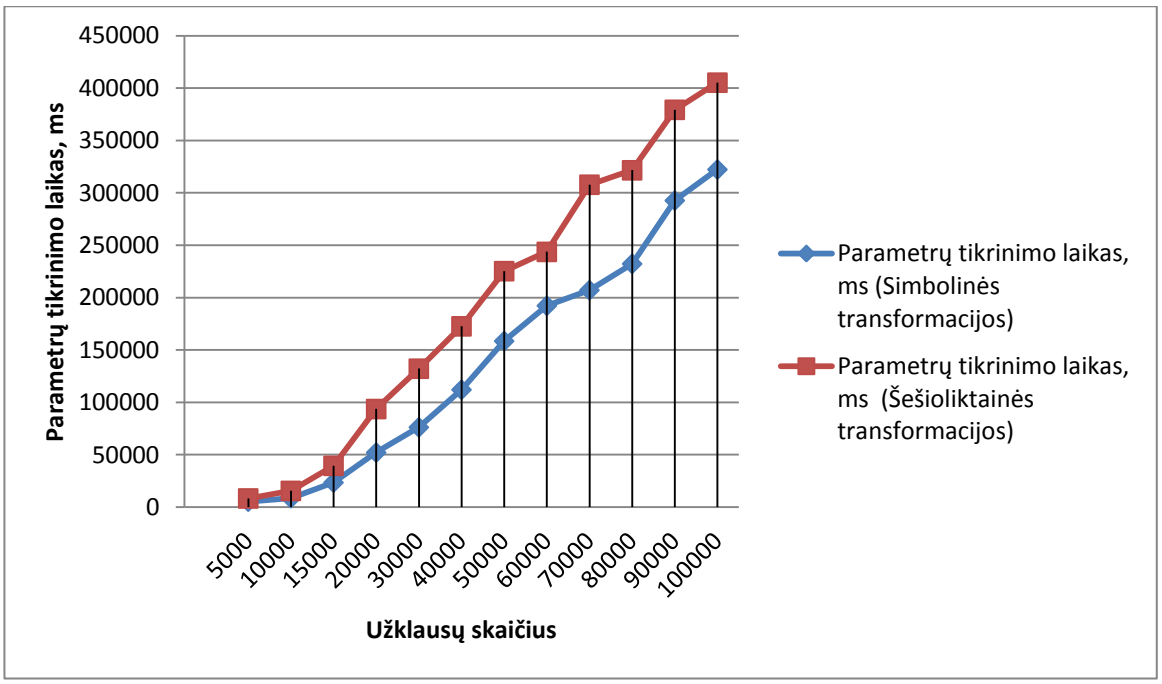
Lentelėje 8 pateikti parametrų tikrinimo laikai, kurie buvo gauti naudojant eksperimento biblioteką ir testinę programą. Skaičiavimai buvo vykdomi tris kartus. Tikrinimo algoritmas atliko reguliarių išraiškų transformacijas (simbolines ir šešioliktaines). Vykdytas užklausų diapazonas nuo 5 tūkstančių iki 100 tūkstančių eilučių.

8 lentelė Tyrimo duomenys

Užklausų skaičius	**Parametrų tikrinimo laikas, ms (Simbolinės transformacijos)**	**Parametrų tikrinimo laikas, ms (Šešioliktainės transformacijos)**
5000	5333	8397
10000	8699	15614
15000	23594	39531
20000	52221	93765
30000	76313	132255
40000	112181	172700
50000	158615	225374
60000	192337	243778
70000	207230	307883
80000	232350	321664
90000	292779	379412
100000	322398	405315

Iš gautų duomenų galima pastebėti, kad laikai skiriasi nuo prieš tai atlikto eksperimento. Tikrinimo algoritmas įvertino visą eilutės tikrinimo laiką. Į laiką įskaičiuota transformacijų laikai, įvykių įrašymai į failą, tikrinimo algoritmas.

Šešioliktainis tikrinimas ir transformacijos užtrunka ilgiau, nes algoritme atliekamos papildomos funkcijos, kurios simbolį verčia į baitus, o baitai verčiami į šešioliktainę išraišką. Pati reguliari išraiška yra 4 kartus ilgesnė nei simbolinė reguliari išraiška. Todėl ir pats tikrinimo algoritmas užtrunka ilgiau, nes reikia apdoroti didesnius simbolių masyvus. Nors ilgiai skiriasi 4 kartus, tačiau laiko santykis yra mažesnis.



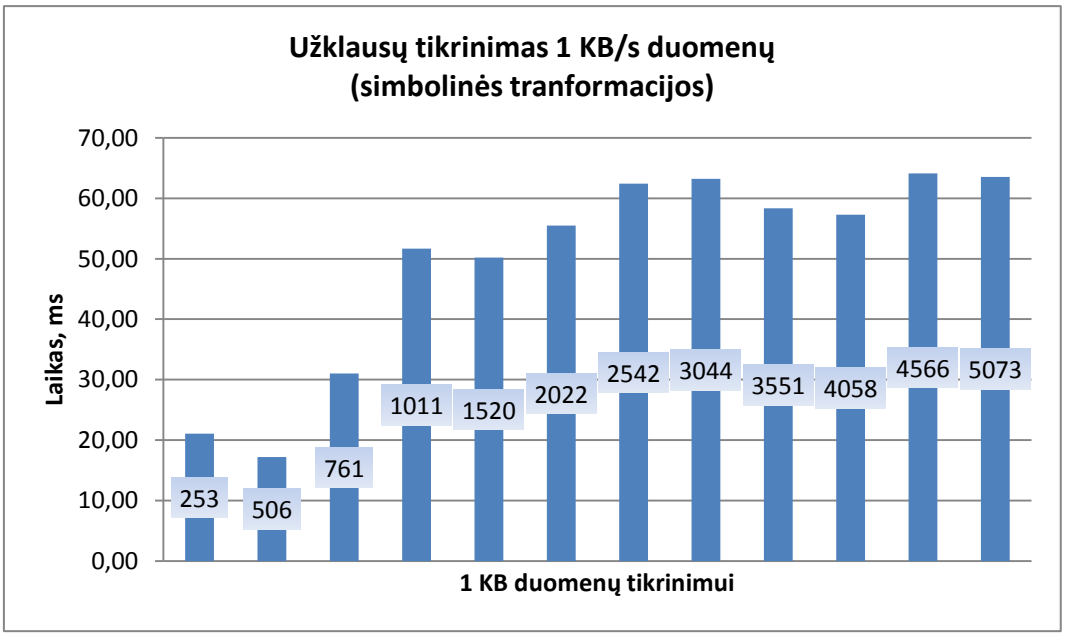
17 pav. Įvesties parametrų tikrinimo laikas

Iš grafiko galima pastebėti, kad visas tikrinimo algoritmo laikas yra ganėtinai panašus į vykdomų transformacijų laiką. Laiko sąnaudos yra panašios iki 15 tūkstančių užklausų, toliau didėjant užklausų skaičiui laiko skirtumas taip pat didėja. Didžiausias laiko skirtumas yra ties 70 tūkstančių užklausų, jis yra apie 1,6 minutės, o mažaisiais laiko skirtumas yra apdorojant 253 KB dydžio failą, 5 tūkstančius užklausų. Laiko skirtumas yra apie 3 sekundes.

Kaip ir galima buvo tikėtis tikrinimas, kuriuose vykdomos šešioliktainės reguliarių išraiškų transformacijos užtrunka ilgiau. Svarbiausias šių metodų panaudojimo realiame gyvenime kriterijus yra laikas, o po to tik požymių aptikimas, nors jis yra nežymus.

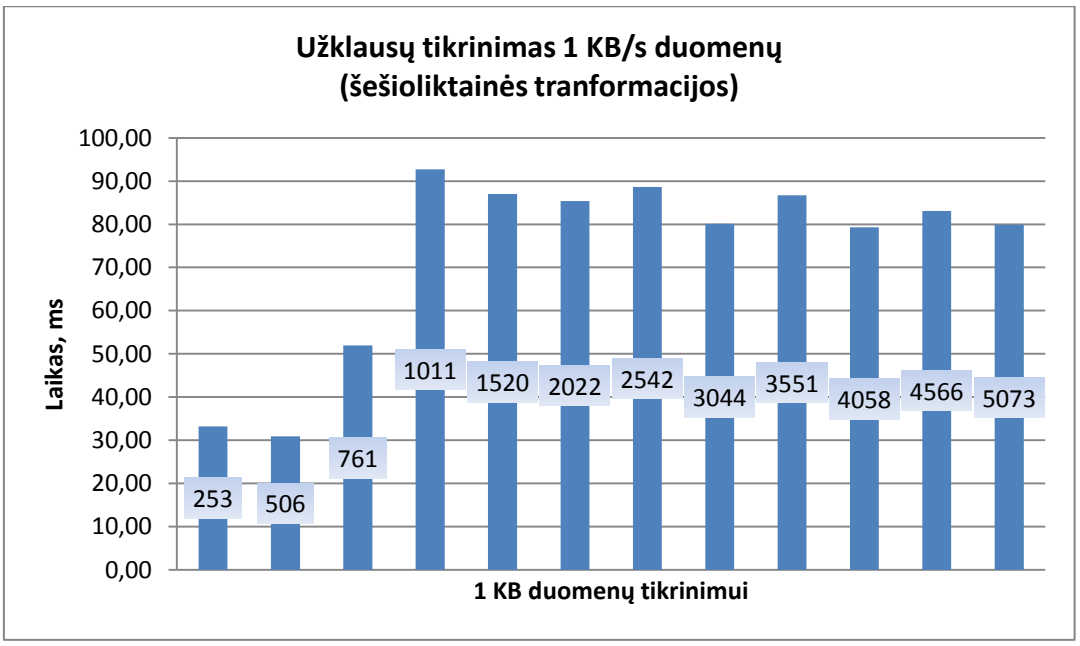
Vidutiniškai tikrinimo ir simbolinių transformacijų laikai skiriasi apie 10%, o šešioliktainių transformacijų apie 18% viso tikrinimo laiko. Ši laiko dalis yra sugaištama atliekant kitus veiksmus, o ne transformacijas. Galime padaryti išvadą, kad didžioji laiko dalis yra eikvojama būtent transformacijų funkcijoms įvykdyti. Taip pat žinant laiko santykį galima pastebėti, kad tikrinimo algoritmas šešioliktainių reguliariųjų išraiškų užtrunka ilgiau nei simbolinių reguliariųjų išraiškų.

Įvertinus bibliotekos funkcijų greitaveiką galima pateikti suminę diagramą, kuri vaizduoja, kiek laiko užtrunka 1 KB informacijos apdorojimui. 1 KB informacijos apdorojimo laikas skiriasi nuo tikrinamo failo dydžio. Šie skirtumai yra dėl paties tikrinimo algoritmo veikimo greičio, neproporcingų laiko sąnaudų.



18 pav. Laiko sąnaudos 1 KB duomenų tikrinimui naudojant simbolines išraiškas

Grafikas vaizduoja kiek laiko reikia norint apdoroti 1 KB informacijos, apdorojant skirtingų dydžių failus bei naudojant simbolines reguliarių išraiškų transfomacijas. Grafiko stulpelio viduryje, kvadtrate pateiktas failo dydis kilobaitais. Apdorojant 0,5 MB dydžio failą buvo sugaišta mažiausiai laiko. Laiko sąnaudos 1 kilobaitui yra apie 18 milisekundžių. Didžiausias vėlinimas apdorojant 1 KB informacijos yra apdorojant 2,5 MB, 3 MB, 4,5 MB ir 5 MB dydžio failus. Esant tokiam duomenų kiekiui, laikas per kurį apdorojama 1 KB informacijos yra apie 62 milisekundės.



19 pav. pav. Laiko sąnaudos 1 KB duomenų tikrinimui naudojant šešioliktaines išraiškas

Paveikslėlyje 19 pavaizduotas analoginis laiko sąnaudų grafikas, kaip ir 16 paveikslėlyje, tačiau čia vaizduojamos laiko sąnaudos, naudojant šešioliktaines transformacijas.

Naudojant būtent šias transformacijas duomenų apdorojimo laiko sąnaudos yra proporcingesnės. Didesniems failams nei 1 MB, sugaištama apie 80 milisekundžių. Apdorojant 0,25 – 0,5 MB dydžio failus laiko sąnaudos yra apie 30 milisekundžių.

Bendru atveju, galime teigti, kad esant mažesniems duomenų kiekiams laiko sąnaudos, apdorojant 1 KB informacijos yra žymiai mažesnės. Naudojant šias funkcijas vėlinimas daugiausia priklausys nuo sistemos apkrovimo. Jei duomenų apdorojimo kiekis padidės, tuomet išaugs ir vėlinimas, naudojant šias funkcijas.

3.3. IŠVADOS

Atlikto interaktyvios saugos sistemos prototipo apsaugai nuo injekcinių atakų eksperimento tyrimo metu nustatyta:

- ✓ Šešioliktainė reguliari išraiška yra 4 kartus ilgesnė nei simbolinė reguliari išraiška. Vidutiniškai tikrinimo ir simbolinių transformacijų laikai skiriasi apie 10%, o šešioliktainių transformacijų apie 18% viso tikrinimo laiko.
- ✓ Esant mažesniems įvesties parametrų skaičiams laiko sąnaudos, apdorojant 1 KB informacijos yra apie 2,5 karto mažesnės.
- ✓ Naudojant simbolines transformacijas vidutiniškai 1 kilobaito informacijos apdorojimui sugaištama apie 50 milisekundžių. Naudojant šešioliktaines transformacijas vidutiniškai 1 kilobaito informacijos apdorojimui sugaištama apie 73 milisekundes.

4. IŠVADOS

- ✓ Žiniatiklio sistemų injekcijų rizikos buvo įvardintos kaip pavojingiausios 2010 ir 2007 metais. Šio tipo atakos yra aktualiausios jau keletą metų. SQL injekcijos turi aukščiausią pavojingumo įvertinimą CWE duomenų bazėje.
- ✓ Pasiūlytas ir praktiškais realizuotas Bekaus ir Nauro formos transformavimo į reguliarias išraiškas algoritmas.
- ✓ Suprojektuotas ir realizuotas interaktyvus Bekaus ir Nauro formos transformacijų į reguliarias išraiškas programinis modulis.
- ✓ Šešioliktainė reguliari išraiška yra 4 kartus ilgesnė nei simbolinė reguliari išraiška. Vidutiniškai tikrinimo ir simbolinių transformacijų laikai skiriasi apie 10%, o šešioliktainių transformacijų apie 18% viso tikrinimo laiko.
- ✓ Naudojant simbolines transformacijas vidutiniškai 1 kilobaito informacijos apdorojimui sugaištama apie 50 milisekundžių. Naudojant šešioliktaines transformacijas vidutiniškai 1 kilobaito informacijos apdorojimui sugaištama apie 73 milisekundes.

LITERATŪRA

1. **Nuno Antunes, Nuno Laranjeiro, Marco Vieira, Henrique Madeira.** // Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services // 2009 IEEE International Conference on Services Computing //Department of Informatics Engineering University of Coimbra – Portugal.
2. **Wei, K.; Muthuprasanna, M.; Suraj Kothari.** // Preventing SQL injection attacks in stored procedures. In Software Engineering Conference, 2006. Australian.
3. **S. San-Tsai, W. Ting Han, L. Stephen, L. Sheung.** // Classification of SQL injection attacks. Electrical and Computer Engineering, University of British Columbia.
4. **F. Valeur, D. Mutz, and G. Vigna.** // A Learning- Based Approach to the Detection of SQL Attacks. In Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Vienna Austria, July 2005.
5. **Prithvi Bisht, P. Mathusadan and V. N. Venkatakrishnan.** // CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACMTransactions on Information and System Security,Vol. 13, No. 2, Article 14, Publication date: February 2010.
6. **Amit Klein, former Director of Security and Research, Sanctum.** // Blind xpath injection, A whitepaper from Watchfire, 2005.
7. **Emmi, M., Majumdar, R. and Sen.** // K. Dynamic test input generation for database applications. In International Symposium on Software Testing and Analysis (ISSTA'07) (2007), ACM.
8. **Meiko Jensen, Lijun Liao and Jörg Schwenk.** // The Curse of Namespaces in the Domain of XML Signature. SWS'09, November 13, 2009, Chicago, Illinois, USA.
9. **M. A. Rahaman and A. Schaad.** // SOAP-based secure conversation and collaboration," in IEEE International Conference on Web Services (ICWS 2007). IEEE CS, 2007.
10. **Z. Su and G. Wassermann.** // The Essence of Command Injection Attacks in Web Applications, Proc. of Annual Symposium on Principles of Programming Languages (POPL), 2006.
11. **Benjamin Livshits and U´ lfar Erlingsson.** // Using Web Application Construction Frameworks to Protect Against Code Injection Attacks. PLAS'07 June 14, 2007, San Diego, California, USA.
12. **Donald Ray Jay Ligatti.** // Defining Code-injection Attacks. POPL'12, January 25–27, 2012, Philadelphia, PA, USA.

13. **X. Zhang and Z. Wang.** // A static analysis tool for detecting web application injection vulnerabilities for ASP program. In International Conference on e-Business and Information System Security (EBISS), May 2010.
14. **Martin Johns and Christian Beyerlein.** // SMask: Preventing Injection Attacks in Web Applications by Approximating Automatic Data/Code Separation. SAC'07 March 1115, 2007, Seoul, Korea.
15. **Zhendong Su and Gary Wassermann.** // The Essence of Command Injection Attacks inWeb Applications. POPL '06 January 11–13, 2006, Charleston, South Carolina, USA.
16. **Konstantinos Kemalis and Theodoros Tzouramanis.** // SQL-IDS: A Specification-based Approach for SQL-Injection Detection. SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.
17. **Bala Neerumalla.** // Quick Security Reference: SQL Injection, November 20, 2012. Microsoft Corporation.
18. **Dave Wichers** // OWASP Top 10 – 2010. The Top 10 Most Critical Web Application Security Risks.
19. **OWASP Risk Rating Methodology.** [Žiūrėta 2012-03-13] . Prieiga per internetą: < https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology >
20. **Matteo Meucci** // OWASP Testing Guide, Version 3.0, December 16, 2008.
21. **Egidijus Kazanavičius, Jevgenijus Toldinas ir Jonas Čeponis**. // Programų sauga. Mokomoji knyga. // KTU Informatikos fakultetas, 2011 m..
22. **Mitigating XPath Injection Attacks in .NET. //** [Žiūrėta 2012-04-04] . Prieiga per internetą: < http://www.tkachenko.com/blog/archives/000385.html >
23. **Data Validation.** // [Žiūrėta 2012-04-10]. Prieiga per internetą: **<** https://www.owasp.org/index.php/Data_Validation **>**
24. **Valentina Dagienė ir Gintautas Grigas.** // Programavimo kalbų teoriniai pagrindai. // Vilniaus universitetas, Vilnius 2007 m..
25. **R. S. Scowen.** // Extended BNF Ageneric base Standard. // 9 Birchwood Grove, Hampton, Middlesex, Great Britain TW12 3DU. September 17, 1998.
26. **Jonathan Bartlett. //** The art of metaprogramming, Part 1: Introduction to metaprogramming. // [Žiūrėta 2012-05-01] . Prieiga per internetą: < http://www.ibm.com/developerworks/linux/library/l-metaprog1/index.html >