VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Dažniausiai saityno programose pasitaikančių saugumo pažeidžiamumų prevencija

Prevention of the Most Common Security Vulnerabilities in Web Applications

Baigiamasis bakalauro darbas

Atliko: Karolis Paulauskas

VU el. p.: karolis.paulauskas@mif.stud.vu.lt

Vadovas: prof. dr. Igoris Belovas

Vilnius

2022

TURINYS

SANTRAUKA....5
SUMMARY....6
1. ĮVADAS....7
1.1. Santrumpų žodynas....7
1.2. Temos aktualumas....7
1.3. Tyrimo objektai....8
1.4. Problema....8
1.5. Darbo tikslas....9
1.6. Darbo uždaviniai....9
1.7. Darbo struktūra....9
2. PAŽEIDŽIAMUMŲ ANALIZĖ....10
2.1. Literatūros apžvalga....10
2.2. Prieigos kontrolės pažeidžiamumai....12
2.2.1. IDOR pažeidžiamumas....12
2.2.1.1. IDOR pažeidžiamumo pavyzdžių analizė....13
2.2.1.1.1. IDOR pažeidžiamumas platformoje Nextcloud....13
2.2.1.1.2. IntenseDebate saityno programos IDOR pažeidžiamumas....13
2.2.1.1.3. Semrush Academy saityno programos prieigos kontrolės pažeidžiamumas....14
2.3. Kriptografinės klaidos....14
2.3.1. Sąsajos su vartotoju saugumas....15
2.3.1.1. Sąsajos su vartotoju saugumo pažeidžiamumų pavyzdžių analizė....15
2.3.1.1.1. Instagram brand nesaugios sąsajos su vartotoju pavyzdys....15
2.3.2. Vartotojų slaptažodžių saugojimas....15
2.3.2.1. Neteisingo vartotojų slaptažodžių saugojimo pavyzdžių analizė....16
2.3.2.1.1. Citybee duomenų bazės nutekėjimas....16
2.3.2.1.2. Adobe duomenų bazės nutekėjimas....16
2.4. Injekcijos pažeidžiamumai....17
2.4.1. XSS pažeidžiamumai....17
2.4.1.1. XSS pažeidžiamumų pavyzdžių analizė....17
2.4.1.1.1. Vienkartinis XSS pažeidžiamumas įmonės Equifax svetainėje....18
2.4.1.1.2. Nuolatinis XSS pažeidžiamumas įmonės Shopify svetainėje....18
2.4.1.1.3. Vienkartinis XSS naudojantis HTTP antrašte svetainėje Omise....19
2.4.2. SQL injekcija....19
2.4.2.1. SQL injekcijos pavyzdžių analizė....20
2.4.2.1.1. SQL injekcija Atavist puslapyje....20
2.4.2.1.2. SQL injekcija valstybiniame JAV duomenų katalogo puslapyje....20

2.4.2.1.3. SQL injekcija CONTACT finansinių paslaugų saityno programoje 20
2.5. Pažeidžiamumų analizės apibendrinimas 21
3. PAŽEIDŽIAMUMŲ PREVENCIJOS METODAI 22
3.1. Prieigos kontrolės pažeidžiamumų prevencija 22
3.1.1. IDOR pažeidžiamumo prevencija 22
3.1.1.1. Reikalavimai objektų identifikatoriams 22
3.1.1.2. Prieigos kontrolės užtikrinimas 22
3.2. Kriptografinių pažeidžiamumų prevencija 23
3.2.1. Sprendimas saugiai sąsajai su vartotoju 23
3.2.2. Sprendimas saugiam vartotojų slaptažodžių saugojimui 25
3.3. Injekcijos pažeidžiamumų prevencija 26
3.3.1. Sprendimas XSS spragų prevencijai 26
3.3.2. SQL injekcijos pažeidžiamumų prevencija 27
3.4. Pažeidžiamumų prevencijos metodų apibendrinimas 28
4. PAŽEIDŽIAMUMŲ PREVENCIJOS METODŲ TYRIMAS 29
4.1. Prieigos kontrolės pažeidžiamumų prevencijos metodų tyrimas 29
4.1.1. Objektams siūlomų naudoti identifikatorių vertinimas 29
4.2. Kriptografinių pažeidžiamumų prevencijos metodų tyrimas 31
4.2.1. HTTPS protokolo naudojimo užtikrinimo metodų vertinimas 31
4.2.2. Slaptažodžiams saugoti skirtų maišos algoritmų efektyvumo įvertinimas 32
4.2.2.1. Algoritmų greitis 32
4.2.2.2. Atsparumas vaivorykštinės lentelės atakai 34
4.3. Injekcijos pažeidžiamumų prevencijos metodų tyrimas 34
4.3.1. Priemonių skirtų apsisaugoti nuo XSS pažeidžiamumų įvertinimas 34
4.3.1.1. Įvesties filtravimas 34
4.3.1.2. Išvesties kodavimas 36
4.3.1.3. CSP taikymas 38
4.3.1.4. Slapukų apsaugojimas 40
4.3.2. SQL injekcijos prevencijos metodų įvertinimas 40
4.3.2.1. Įvesties filtravimas 41
4.3.2.2. Paruoštosios formuluotės naudojimas 41
4.4. Pažeidžiamumų prevencijos metodų tyrimo apibendrinimas 42
5. REZULTATAI IR APIBENDRINIMAS 43
LITERATŪRA 45
1 PRIEDAS 49
2 PRIEDAS 50
3 PRIEDAS 51

SANTRAUKA

Pagrindinis baigiamojo darbo tikslas – pateikti rekomendacijų sąrašą, kuriuo naudojantis programuojant saityno programą, būtų padedama išvengti dažniausiai pasitaikančių saityno programų saugumo pažeidžiamumų. Darbo metu buvo atlikti šie uždaviniai: išanalizuotos dažniausiai pasitaikančios saityno programų saugumo spragos, apžvelgti jų pavyzdžiai versle, išanalizuotos ir praktiškai išbandytos bei ištirtos literatūroje siūlomos metodikos, skirtos užkirsti kelią saityno programų pažeidžiamų atsiradimui, galiausiai darbo metu gautos įžvalgos panaudotos rekomendacijų sąrašui sudaryti. Sukurtas rekomendacijų sąrašas padedantis išvengti dažniausių saityno programų saugumo spragų - netinkamos prieigos kontrolės, kriptografijos ir injekcijos pažeidžiamumų.

Raktiniai žodžiai: saityno programos, pažeidžiamumas, prevencija.

SUMMARY

The main goal of the thesis is to create a recommendation list that could be used to help with preventing the most common vulnerabilities in web applications. The goal was achieved by completing the following tasks: analyzing the most common web application vulnerabilities, studying examples of their occurences in real-world applications, researching and examining web application vulnerability prevention methods suggested in literature, and lastly by creating a recommendation list using the insights gained during the research. The final result of the work is a recommendation list that should help developers with preventing the most common web application vulnerabilities – broken access control, cryptographic failures, and injection.

Keywords: web application, vulnerability, prevention.

1. ĮVADAS

1.1. Santrumpų žodynas

XSS – Cross-Site-Scripting. IT sistemų pažeidžiamumas leidžiantis įsilaužėliui įterpti savo kenksmingą kodą į sistemos puslapį.

SQL – Structured Query Language. Struktūrizuota Užklausų Kalba, skirta bendrauti su duomenų baze ir modifikuoti joje esančius duomenis.

SQLi – SQL injection. SQL užklausų injekcijos pažeidžiamumas.

DAST - Dynamic Application Security Testing. Dinaminis programų saugumo testavimas.

SAST – Static Application Security Testing. Statinis programų saugumo testavimas.

CWE - Common Weakness Enumeration. Bendruomenės plėtojamas dažnai pasitaikančių programinės bei techninės įrangos pažeidžiamumų sąrašas.

IDOR - Insecure Direct Object Reference. Prieigos kontrolės pažeidžiamumų tipas, kuris leidžia vartotojui tiesiogiai pasiekti sistemoje naudojamus objektus.

HTTP – HyperText Trasnfer Protocol. Programos lygio duomenų perdavimo protokolas, sukurtas komunikacijai tarp naršyklės ir saityno programos serverio.

HTTPS - HyperText Trasnfer Protocol Secure. HTTP protokolo praplėtimas naudojamas šifruotai komunikacijai tinkle.

3DES - Triple Data Encryption Algorithm. Šifras, kuris buvo išvestas iš originalaus duomenų šifravimo standarto.

HTML - HyperText Markup Language. Žymėjimo kalba, skirta dokumentų atvaizdavimui naršyklėje.

DOM – Document Object Model. Sąsaja, kuri vaizduoja žymėjimo kalbomis parašytus objektus medžio formatu.

CSP – Content Security Policy. Deklaratyvus politikos mechanizmas, leidžiantis saityno programų kūrėjams apibrėžti, kurie kliento pusės ištekliai gali būti įkelti ir vykdomi naršyklės.

UUID – Universal Unique Identifier. 128 bitų reikšmė, skirta unikaliai identifikuoti objektą internete.

HSTS – Strict Transport Security. Paprastas ir plačiai palaikomas standartas, skirtas užtikrinti, jog puslapio lankytojų naršyklės visuomet prie svetainės jungtųsi per HTTPS.

ASIC – Application-Specific Integrated Circuit. Konkrečiai paskirčiai pritaikytas lustas.

OWASP – Open Web Application Security Project. Pelno nesiekianti organizacija, kurios tikslas yra gerinti programinės įrangos saugumą.

1.2. Temos aktualumas

Internetas, prasidėjęs kaip akademinis tyrimų projektas, greitai tapo neatsiejama modernaus žmogaus gyvenimo dalimi. Internetas naudojamas informacijos sklaidai, elektroninės prekybos ir bankininkystės paslaugoms, pramogoms ir tarpusavio bendravimui bei atlieka daug kitų funkcijų. Didžioji dalis šio interneto funkcionalumo įgyvendinama naudojantis saityno programomis.

Dažniausiai saityno programos sudarytos iš dviejų dalių – kodo, kuris vykdomas vartotojo naršyklėje ir kodo vykdomo saityno serverio pusėje [1]. Klientas ir serveris bendrauja perduodami informaciją internetu.

Skaitmeniniame amžiuje verslas privalo prisitaikyti prie vis spartėjančio inovacijų ir pokyčių tempo - dažnai didžiausios sėkmės susilaukia pirmasis įgyvendinęs naują idėją. Tai, be abejo, galioja ir kuriamoms saityno programoms. Spaudimas sparčiai judančioje rinkoje programas kurti greitai, nepakankamas programuotojų žinių kiekis, ribotas biudžetas ar paprasčiausias abejingumas kibernetinio saugumo atžvilgiu yra dažnos priežastys, kodėl saityno programose neretai paliekamos saugumo spragos.

Platus saityno programų atliekamų funkcijų spektras reiškia, jog programoje esančios saugumo spragos gali atverti duris labai rimtoms problemoms. Saityno programų naudojamose duomenų bazėse laikomi milžiniški kiekiai privačių duomenų, įskaitant susirašinėjimus, sveikatos bei finansinius duomenis. Deramai neapsaugoti, privatūs duomenys gali patekti į kibernetinių įsilaužėlių rankas. Taip pat saityno programos naudojamos elektroninėms parduotuvėms ir finansinėms platformoms, saugumo spragos šiose programose leistų užpuolikui nusipirkti prekę nemokant pinigų ar tiesiog įvykdyti nelegalią transakciją. Finansinio nusikaltimo grėsmė ypač aktuali augant įvairioms kriptografinių virtualiųjų valiutų platformoms. Vienoje tokių platformų pavadinimu „Polygon“ rasta saugumo spraga būtų leidusi užpuolikui pavogti per 850 milijonų JAV dolerių. Laimei, ši spraga buvo rasta kibernetinio saugumo tyrėjų, kurie apie spragą pranešė „Polygon“ atstovams. Už spragos radimą buvo išmokėta didžiausia kada nors matyta premija – kartu sudėjus tyrėjai kišenes papildė beveik trimis su puse milijonų JAV dolerių [2]. Saityno programų saugumo spragos gali pakenkti net valstybiniu mastu. Atakos nukreiptos į valstybines programas gali sutrikdyti valstybinių institucijų veiklą.

Kibernetiniams nusikaltėliams prieinama vis daugiau įrankių ir informacijos apie tai, kaip įsilaužti į svetaines. Nepatyręs, bet motyvuotas nusikaltėlis, norėdamas surasti saugumo spragas, gali pasitelkti plačiai prieinamus automatinius spragų paieškos įrankius. Taip pat, nusikalstamajame pasaulyje atsirado ir populiarėja „įsilaužimas kaip paslauga“ (ang. *Hacking as a Service – HaaS*), tai reiškia, jog piktybinis veikėjas gali užsakyti patyrusio kibernetinio įsilaužėlio paslaugą tam, kad įvykdyti savo pasirinktą kibernetinį nusikaltimą [3].

Didelis potencialus atlygis ir palyginus lengvas spragų radimas skatina vis didesnius kibernetinių nusikaltimų mastus. Pagal kibernetinio saugumo įmonės SonicWall 2020 metų kibernetinio saugumo ataskaitą, 2019 metais užfiksuota 52% daugiau saityno programų atakų lyginant su 2018 metais, o pagal CDNetworks interneto saugumo būklės ataskaitą 2020 metais saityno programų atakų užfiksuota 7.4 kartus daugiau nei 2019 metais [4, 5]. Akivaizdu, jog saityno programų kibernetinio saugumo tema laikui bėgant taps tik aktualesnė.

1.3. Tyrimo objektai

- Saugumo spragos, dažniausiai pasitaikančios saityno programose.
- Metodikos, skirtos užkirsti kelią saityno programų saugumo spragų atsiradimui.

1.4. Problema

Kaip užkirsti kelią dažniausioms saityno programų spragoms?

1.5. Darbo tikslas

Pasiūlyti konkrečių rekomendacijų sąrašą dažniausių saugumo spragų saityno programose prevencijai.

1.6. Darbo uždaviniai

1. Apžvelgti ir išanalizuoti dažniausiai pasitaikančias saityno programų spragas, siekiant suprasti jų atakų scenarijus ir atsiradimo priežastis.
2. Surasti ir apžvelgti saityno programų spragas užfiksuotas versle, kad įvertinti, kaip saugumo spragos pasireiškia tikrose sistemose, ir turėti konkrečius spragų pavyzdžius, kuriems galima būtų pritaikyti spragų prevencijos metodikas.
3. Surasti ir išanalizuoti metodikas, skirtas užkirsti kelią spragų atsiradimui, siekiant suprasti jų veikimą.
4. Praktiškai išbandyti ir ištirti siūlomas metodikas, skirtas užkirsti kelią spragų atsiradimui, kad įvertinti jų efektyvumą.
5. Apibendrinus atliktų uždavinių įžvalgas, pasiūlyti konkrečias metodikas galinčias užkirsti kelią analizuotoms saityno programų saugumo spragoms.

1.7. Darbo struktūra

Pasirinkta analizuoti spragas pateikiamas 2021 metų OWASP Top 10 dokumente [6]. Naudojantis dokumentu užtikrinama, jog apžvelgiamos aktualiausios, dažniausiai pasitaikančios ir rizikingiausios spragos. Saityno programose galima rasti daug skirtingų spragų, kurios neapsiriboja OWASP Top 10, tačiau šis sąrašas yra puikus pirmasis žingsnis svetainių saugumo situacijos gerinime. Norint plačiau išnagrinėti spragų prevencijos būdus, nuspręsta apsiriboti trijų pirmųjų OWASP Top 10 dokumento kategorijų apžvalga – „1. Neteisinga prieigos kontrolė“, „2. Kriptografinės klaidos“, „3. Injekcija“. Darbe saugumo spragos analizuojamos apžvelgiant literatūros šaltinius ir pažeidžiamumų pasitaikymo pavyzdžius tikrose sistemose. Tuomet atliekama metodinė dalis, kurioje aprašomi spragų prevencijos būdai. Galiausiai prevencijos metodai ištiriami ir sukonkretinami į rekomendacijas, kurios būtų naudingos norint sukurti saityno programą, atsparią dažniausiai pasitaikantiems saugumo pažeidžiamumams.

Baigiamąjį darbą sudaro septynios dalys: įvadas, pažeidžiamumų analizė, pažeidžiamumų prevencijos metodai, pažeidžiamumų prevencijos metodų tyrimas, rezultatai ir apibendrinimas, literatūros sąrašas, trys priedai.

2. PAŽEIDŽIAMUMŲ ANALIZĖ

Norint praplėsti supratimą apie dažniausius saityno pažeidžiamumus, apžvelgiami literatūros šaltiniai šia tema. Tuomet susipažįstama su kiekviena nagrinėjama spragų kategorija, analizuojami konkretūs pažeidžiamumai kiekvienoje kategorijoje, apžvelgiami jų pavyzdžiai.

2.1. Literatūros apžvalga

Precedento neturintys saityno programų atakų ir įsilaužimų skaičiai reiškia, jog saityno programų saugumo užtikrinimas tampa vis dažniau nagrinėjama tema tiek versle, tiek akademinėje bendruomenėje. Saityno programų saugumo problemos nagrinėjamos daugelyje skirtingų neseniai išleistų šaltinių [7-9]. Autoriai pabrėžia temos aktualumą, detalizuoja saugumo spragų atsiradimo priežastis, daugelis įvardija tokias spragas kaip injekcija (SQLi, XSS) ar netinkamą prieigos kontrolę kaip dažniausiai pasitaikančias ir pavojingiausias spragas.

Dažnai saityno programų spragos nagrinėjamos iš teorinės pusės, neskiriant daug dėmesio konkretiems versle rastų spragų pavyzdžiams. Senesni leidiniai neturėjo tokios galimybės, nes pažeidžiamumų koordinavimo ir klaidų mažinimo platformos kaip Hackerone [10] ar Bugcrowd [11] pasirodė tik 2011-2012 metais, o iki to laiko rasti patikimos informacijos apie tikrus buvusius pažeidžiamumus verslo svetainėse buvo sunku.

Daugelio tyrinėtų autorių darbuose dažnai tyrimo objektas būna net ne spraga, o įvairūs siūlomi automatiniai spragų paieškos įrankiai su tikslu išmatuoti jų efektyvumą [12-15]. Problemai spręsti dažniausiai siūlomi dinaminiai programų saugumo testavimo įrankiai (ang. *dynamic application security testing - DAST*) arba statiniai programų saugumo testavimo įrankiai (ang. *static application security testing- SAST*). Dinaminiai spragų paieškos įrankiai sutelkia dėmesį į testus skirtus įvertinti kaip programos reaguoja į piktybines užklausas jų veikimo metu [16]. Taigi DAST įrankiai bando atrasti spragas žiūrint iš vartotojo perspektyvos, nematant programos kodo ar kitų vidinių sistemų, dar tai galima pavadinti juodosios dėžės (ang. *black-box*) metodu. Šie įrankiai yra labai efektyvūs ieškant tokių pažeidžiamumų kaip XSS, tačiau norint šiuos įrankius naudoti, jau reikia turėti sukurtą programą ar bent jos dalį [17]. SAST įrankiai ieškodami saugumo pažeidžiamumų analizuoja programos kodą. Statinės analizės įrankiai tam tikromis aplinkybėmis gali būti labai efektyvūs ir rasti daugiau nei 70% programose esančių spragų. Tačiau pabrėžiama, jog tikrinant SAST įrankių rezultatus dažnai pasitaiko klaidingų teigiamų rezultatų (ang. *false positives*), tai reiškia, kad reikia nemažai žmogiškųjų išteklių patvirtinant šių įrankių radinius [18].

Apibendrinant, saityno programų saugumo spragų problema sprendžiama bandant spragas surasti, o suradus – palikti atsakomybę programuotojui tą spragą panaikinti. Labai retai kalbama apie priemones, kurių reiktų imtis ankstyvuosiuose programavimo ar sistemos architektūros kūrimo stadijose.

Norint pasiekti geriausią rezultatą privaloma ne tik naudoti statinės ar dinaminės saugumo analizės įrankius, tačiau taip pat programų saugumo klausimą „pastumti į kairę“ (ang. *shift left*). Pastumimas į kairę reiškia, jog apie kuriamos programos saugumą privalu galvoti kuo anksčiau jos kūrimo cikle ir taip iš sistemos kūrimo gyvavimo ciklo (ang. *systems development life cycle - SDLC*) pereiti į saugų sistemos kūrimo gyvavimo ciklą (ang. *secure systems development life cycle - SSDLC*). Sistemos saugumą planuojant iš anksto išvengiama testavimo metu atrastų spragų

taisymo jau sukurtoje sistemoje, kas dažnai užima daug laiko ir resursų. Blogiausiu atveju testavimo metu rastų spragų nusprendžiama netaisyti, norint neviršyti sistemos kūrimo biudžeto ar laiko suvaržymų.

Apie tai kaip išvengti dažniausių pažeidžiamumų saityno programose ir kokių priemonių saugumo užtikrinimui reikia imtis programų kūrimo ciklo metu nagrinėja S. Gupta ir B. B. Gupta [1]. Leidinyje apžvelgiamas dažniausiai pasitaikančių saityno programų spragų sąrašas remiantis 2013 metų OWASP Top 10 dokumentu. Apžvelgiami tiek spragų atakų scenarijai, tiek kaip pažeidžiamumus rasti bei jų išvengti. Spragos taip pat įvertinamos pagal tai, kaip dažnai jos atsiranda, kaip sunku jas išnaudoti, koks galimas spragos išnaudojimo poveikis bei kaip sunku atrasti spragą testavimo metu. Autoriai pabrėžia kibernetinio saugumo svarbą ankstyvuosiuose programų kūrimo stadijose ir rekomenduoja sekti „OWASP sukčiavimo lapais“ (ang. *OWASP Cheet Sheet Series*) [19] kaip saugių saityno programų kūrimo standartu. Patys autoriai savo darbe rekomendacijas spragų išvengimui pateikia abstrakčiai, nepateikdami skirtingų prevencijos metodų palyginimų ar efektyvumo įvertinimo.

Pelno nesiekianti organizacija OWASP aktyviai prisideda prie programinės įrangos saugumo gerinimo. Organizacija kas keletą metų išleidžia informuotumo dokumentą OWASP Top 10 aprašantį dažniausias ir rizikingiausias saityno programų spragas. Šis spragų sąrašas pateikiamas kaip vienas efektyviausių pirmų žingsnių saugios programinės įrangos kūrimo kultūrai gerinti. Duomenys šiam sąrašui renkami iš kibernetinės saugumo įrangos pardavėjų, saugumo konsultantų, bei pažeidžiamumų koordinavimo ir klaidų mažinimo platformų. Kiekviena pažeidžiamumų kategorija aprašoma pateikiant pažeidžiamumų aprašymus, nurodant kaip jų išvengti bei detalizuojant kokie įmanomi šių pažeidžiamumų atakų scenarijai.

OWASP organizacija aprašydama spragas taip pat pateikia nuorodas į susijusių pažeidžiamumų apibūdinimus „Dažnų Pažeidžiamumų Sąraše“ (ang. *Common Weakness Enumeration - CWE*) [20]. Pagrindinis CWE tikslas yra sustabdyti kelią pažeidžiamumų atsiradimui pradedant nuo jų šaltinio, šviečiant įrangos architektus, dizainerius, programuotojus ir naudotojus apie tai, kaip išvengti dažnai pasitaikančių spragų iki išleidžiant naują produktą. Ypač svarbu šiam darbui, jog CWE pažeidžiamumų aprašuose yra ne tik spragų atakų scenarijai, pasekmės ir jų atradimo mechanizmai, tačiau ir spragos išvengimo priemonių ir programavimo praktikų sąrašas su rekomendacijomis skirtingose programų kūrimo stadijose.

Hackerone - tai populiari atlyginimo už klaidų atradimą programa (ang. *bug bounty*), kurioje kibernetinio saugumo specialistai bei mėgėjai gali ieškoti saugumo pažeidžiamumų programoje dalyvaujančių verslų programinėje įrangoje. Programoje dalyvaujančios įmonės turi atsakingo atskleidimo politiką (ang. *responsible disclosure policy*) - tai taisykles, kuriomis vadovaujantis galima ieškoti ir pateikti saugumo spragas jų produktuose be teisinių veiksmų rizikos. Rastus pažeidžiamumus kibernetinio saugumo tyrėjai dokumentuoja ir pateikia peržiūrai. Verslo atstovams patvirtinus, jog pažeidžiamumas tikras, tyrėjams išmokamas atlygis pagal iš anksto nustatytus rėžius. Taip pat neretai spragas ištaisius pirminė ataskaita redaguojama, paslepiant konfidencialią informaciją, ir paviešinama. Paviešintas ataskaitas galima rasti Hackerone puslapyje, taigi šios ataskaitos leidžia tyrinėti versle pasitaikiusias realias, pačių verslo atstovų patvirtintas saugumo spragas.

2.2. Prieigos kontrolės pažeidžiamumai

OWASP organizacijos duomenimis viena ar kita forma prieigos kontrolės spragų buvo rasta 94 procentuose visų išbandytų saityno programų [6]. Atitinkamai naujame 2021 metų OWASP Top 10 sąraše prieigos kontrolės pažeidžiamumų kategorija užima pirmąją vietą. Prieigos kontrolė užtikrina, jog resursai gali būti prieinami tik autorizuotiems vartotojams, pagal iš anksto numatytą prieigos kontrolės politiką [21]. Prieigos kontrolės pažeidžiamumai leidžia įsilaužėliui apeiti prieigos kontrolės mechanizmą ir pasiekti, modifikuoti ar net ištrinti informaciją, kuri jam turėtų būti neprieinama, taip pat atlikti tam tikrus neautorizuotus verslo logikos veiksmus. Dažni prieigos kontrolės pažeidžiamumai apima, bet neapsiriboja šiais [6]:

- Mažiausios privilegijos principo pažeidimai, kuomet prieiga prie resurso turėtų būti skirta tik tam tikrai vartotojų grupei, bet yra prieinama visiems.
- Prieigos kontrolės patikrinimų apėjimas modifikuojant nuorodą, vidinę programos būseną, HTML puslapį ar užklausas į programą;
- Neautorizuotas kito vartotojo paskyros peržiūrėjimas ar redagavimas naudojant unikalų paskyros identifikatorių;
- Privilegijų eskalacija - apsimetimas vartotoju neprisijungus prie sistemos ar apsimetimas vartotoju su administratoriaus privilegijomis, nors prisijungimas atliktas su paprasto vartotojo paskyra.
- Metaduomenų manipuliavimas, taip pat slapukų (ang. *cookie*) arba paslėpto lauko manipuliavimas siekiant gauti tam tikras privilegijas.
- Neteisinga CORS politikos konfigūracija, leidžianti pasiekti programos duomenis naudojantis nepatikimu šaltiniu.
- Tiesioginė prieiga į neapsaugotus puslapius, kuriuos pasiekti turėtų tik privilegijuoti vartotojai.

2.2.1. IDOR pažeidžiamumas

Pasirinkta nagrinėti vieną dažniausiai pasitaikančių prieigos kontrolės pažeidžiamumų – nesaugią tiesioginę objekto nuorodą (ang. *insecure direct object reference - IDOR*). Šis pažeidžiamumas atsiranda kuomet programuotojas pateikia nuorodas į vidinius objektus – duomenų bazės raktus, katalogus ar failus. Esant neteisingai prieigos kontrolei šios nuorodos gali būti panaudotos tam, kad pasiekti duomenis be reikiamos autorizacijos [22]. Dažniausiai IDOR pasireiškia kaip horizontalus privilegijų eskalavimas, tai yra, jog vienas paprastas vartotojas išnaudodamas tokį pažeidžiamumą gauna prieigą prie kito paprasto, ne administracinio, vartotojo [1].

2013 metų OWASP Top 10 sąraše IDOR pažeidžiamumas buvo atskira kategorija ir užėmė ketvirtąją vietą. 2017 metais šis pažeidžiamumas buvo apjungtas su kita spraga - trūkstama funkcijos lygio prieigos kontrole - į bendrą neteisingos prieigos kontrolės kategoriją, kuri 2017 metų sąraše atsidūrė ketvirtoje vietoje.

Naudoti tiesiogines objektų nuorodas neturi saugumo padarinių savaime - taip tik atskleidžiamas objektų identifikatorių formatas. Reali spraga duomenų saugumui atsiranda tuomet, kai programoje yra ir prieigos kontrolės pažeidžiamumas, leidžiantis naudoti atrastus identifikatorius neteisėtai su jais susietų objektų prieigai.

Galima įsivaizduoti paprasčiausią IDOR prieigos kontrolės pažeidžiamumo pavyzdį - tai svetainė, kurioje vartotojas peržiūri savo privatų profilį naudojantis nuoroda su pabaigoje

nurodomu vartotojo skaitiniu identifikatoriumi - *https://svetaine.lt/vartotojas/1*. Tai pirmoji pažeidžiamumo dalis – vartotojui jo privatus profilis pasiekiamas per skaitinį identifikatorių naudojamą vartotojo įraše duomenų bazėje. Galima saugiai daryti prielaidą, jog, kuriant naują vartotoją, prie prieš tai buvusio vartotojo identifikatoriaus pridedamas vienetas, taigi nuspėti nuorodą į naujai sukurto vartotojo profilį tampa labai lengva - *https://svetaine.lt/vartotojas/2*. Nuspėjamų identifikatorių naudojimas tampa prieigos kontrolės pažeidžiamumu, jei vartotojui „1“ nuorodos pabaigoje esantį identifikatorių iš 1 pakeitus į 2 ar kitą skaičių, gaunamas kito vartotojo profilis. Tai reiškia, jog bet kuris vartotojas gali pasiekti bet kurio kito vartotojo privatų profilį ir tuo pačiu konfidencialią informaciją tiesiog pakeisdamas skaičių esantį profilio nuorodos pabaigoje.

2.2.1.1. IDOR pažeidžiamumo pavyzdžių analizė

IDOR pažeidžiamumai yra dažnas reiškinys, todėl norint suprasti kaip jie pasireiškia praktikoje, pasirinkta paanalizuoti IDOR pažeidžiamumus rastus įvairių įmonių svetainėse. Hackerone platformoje rasti trys pavyzdžiai iliustruojantys problemą.

2.2.1.1.1. IDOR pažeidžiamumas platformoje Nextcloud

Pirmasis nagrinėjamas pavyzdys - IDOR prieigos kontrolės pažeidžiamumas įmonės Nextcloud saityno programoje, kuris leido bet kuriam vartotojui ištrinti bet kurio kito vartotojo įrenginio prieigos raktą. Apie pažeidžiamumą pranešta 2020 metų kovo 15 dieną, pažeidžiamumo grėsmė įvertinta kaip aukšta ir už jo radimą išmokėta 500 JAV dolerių premija. Pranešimo identifikatorius Hackerone puslapyje – 819807.

Nextcloud leidžia peržiūrėti, kokie įrenginiai turi galiojančius prieigos raktus susietus su vartotojo paskyra. Vartotojas gali pašalinti pasirinkto įrenginio prieigą prie paskyros naudojant užklausą *POST /settings/personal/authtokens/wipe/{skaičius}*, užklausos pabaigoje esantis skaičius - tai norimo ištrinti įrenginio identifikatorius.

Pranešėjas parodo, kad vienas vartotojas, žinodamas kito vartotojo įrenginio identifikatorių, gali naudoti užklausą to įrenginio prieigos ištrynimui nurodant jo identifikatorių užklausos pabaigoje. Kadangi identifikatoriai yra skaitiniai, įsilaužėlis galėtų susikurti naują įrenginį ir patikrinti jo identifikatorių, norėdamas sužinoti kiek įrenginių šiuo metu naudojama, tuomet automatinių įrankių pagalba (pvz. BurpSuite Intruder) naudoti pažeidžiamą užklausą einant nuo 1 iki įsilaužėlio sukurto įrenginio identifikatoriaus, taip ištrinant visų sistemos vartotojų įrenginių prieigos raktus.

2.2.1.1.2. IntenseDebate saityno programos IDOR pažeidžiamumas

Antrasis analizuojamas pranešimas apie tai kaip IntenseDebate saityno programoje buvęs IDOR prieigos kontrolės pažeidžiamumas leido redaguoti informaciją apie kito vartotojo svetainę/internetinį dienoraštį jo profilyje, žinant to įrašo identifikatorių. Pranešimas užpildytas 2020 metų rugsėjo 3 dieną, pažeidžiamumo grėsmės įvertinimas – aukštas, o už jo radimą išmokėta 200 JAV dolerių premija. Hackerone puslapyje pranešimą galima rasti pagal identifikacinį kodą 974222.

Analizuojamas atakos scenarijus analogiškas pirmajam pavyzdžiui. Vartotojas savo profilyje gali nurodyti jam priklausančių svetainių/internetinių dienoraščių sąrašą, kiekvienas įrašas šiame sąraše turi savo skaitinį identifikatorių. Norint įrašą redaguoti, jo identifikatorius

nurodomas redagavimo užklausos parametre *hidBlogID[]*. Užklausą vykdant nėra tikrinama ar vartotojas yra to įrašo savininkas. Tai reiškia, jog galimas neautorizuotas kitų vartotojų svetainių/internetinių dienoraščių redagavimas jų profiliuose.

Toks pažeidžiamumas galėjo būti panaudotas konkrečių vartotojų puolimui norint pakenkti jų reputacijai, susiejant vartotoją su nepalankia svetaine. Taip pat galima ir masinė ataka, kurios metu įsilaužėlio norima nuoroda galėjo būti patalpinta visų programos vartotojų profiliuose ir taip apgaule priversti daugybę žmonių apsilankyti nurodytoje svetainėje.

2.2.1.1.3. Semrush Academy saityno programos prieigos kontrolės pažeidžiamumas

Paskutinis nagrinėjamas pavyzdys yra mažesnės grėsmės. Pranešime nurodoma, jog Semrush Academy saityno programoje esantis pažeidžiamumas leido užregistruoti bet kokį vartotoją į nuotolinį kursą be jo sutikimo. Apie spragą pranešimas užpildytas 2020 metų sausio 26 dieną, pažeidžiamumas įvertintas kaip vidutiniškai grėsmingas, o pranešėjui išmokėta 505 JAV dolerių premija. Pranešimo identifikacinis kodas – 783708.

Spraga panaši į prieš tai nagrinėtas - registracijos į kursą užklausoje vienas vartotojas, nurodydamas kito vartotojo identifikatorių, galėjo užregistruoti jį į pasirinktą internetinį kursą. Šiuo atveju nėra tikrinama ar vartotojas, kuris siunčia užklausą, ir vartotojas, kuris nurodomas užklausos parametre, sutampa.

Pranešimo komentaruose aptariamas ribotas spragos panaudojimas. Vartotojo identifikatorius nėra skaitinė reikšmė sudaroma pagal skaičių seką, vietoje to vartotojui identifikuoti naudojamas didelę entropiją turintis unikalus identifikatorius. Šis faktas lemia, jog nors prieigos kontrolės pažeidžiamumas analogiškas pirmiems dviem atvejams, pačios spragos praktinis panaudojimas ribotas dėl papildomos užduoties surasti egzistuojančių vartotojų identifikatorius.

2.3. Kriptografinės klaidos

Šiuolaikinė kriptografija apima saugią komunikaciją, pranešimų autentifikavimą, skaitmeninius parašus, raktų apsikeitimo protokolus, laikomų duomenų įslaptinimą, skaitmeninius pinigus bei daug daugiau [23]. Kadangi kriptografija naudojama duomenims apsaugoti nuo pašalinių, pažeidžiamumai kriptografijoje reiškia, jog duomenys, kurie tikėta buvo saugūs, tampa prieinami įsilaužėliui.

Dažniausiai kai kalbama apie bet kokią kriptografiją yra primygtinai rekomenduojama nebandyti kurti savo kriptografinių mechanizmų, kadangi sukurti kriptografiškai saugų algoritmą yra labai sudėtinga. Istorijoje gausybė kriptografinių algoritmų pavyzdžių, kuriems laikui bėgant atrastos sėkmingos nulaužimo atakos. Pavyzdžiai iš netolimos praeities – nebesaugiais laikomi RC2 šifravimo algoritmas ar MD5 maišos algoritmas [23, 24]. Todėl privaloma naudoti tyrėjų patikrintus ir plačiai pripažintus algoritmus bei juos įgyvendinančias bibliotekas, įsitikinant, jog jų naudojimas vis dar saugus, remiantis naujausiais šaltiniais. Taip pat svarbu laikytis dokumentacijos generuojant raktus ar pasirenkant inicijavimo vektorius. Dauguma saityno programų nenaudoja sudėtingų kriptografinių funkcijų, todėl nuspręsta susitelkti į dvi pagrindines užduotis, būdingas didžiajai daugumai kuriamų saityno programų – sąsajos su vartotoju saugumą bei vartotojų slaptažodžių saugojimą.

2.3.1. Sąsajos su vartotoju saugumas

Norint užtikrinti saugią komunikaciją tarp vartotojo ir saityno programos privaloma šifruoti užklausas perduodamas internetu. Šifravimo metu tekstas paverčiamas į šifruotą tekstą, kurio nepageidaujama trečia šalis negali lengvai keisti ar peržiūrėti [25]. Tai svarbu dėl vadinamosios „žmogaus per vidurį“ atakos (ang *man-in-the-middle – MITM*). Šios atakos principas paprastas – tarp dviejų bendraujančių šalių joms nežinant įsiterpia užpuolikas, kuris slapta skaito ar pakeičia perduodamą informaciją. Įsilaužėlis esantis komunikacijos viduryje galėtų perimti jautrią informaciją, įskaitant slaptažodžius. Taip pat apsimesdamas sistemos serveriu galėtų vartotoją nukreipti į kitą svetainę ar vietoje tikimosi failo iš programos, atsiųsti savo kenkėjiškas programas.

2.3.1.1. Sąsajos su vartotoju saugumo pažeidžiamumų pavyzdžių analizė

2.3.1.1.1. Instagram brand nesaugios sąsajos su vartotoju pavyzdys

Instagram brand, naudodamiesi Mandrill elektroninio pašto produktu siuntė nuorodas skirtas slaptažodžio pakeitimui, kurios naudojo nesaugų HTTP protokolą, vietoje šifruoto HTTPS. Dėl šios priežasties buvo įmanoma „žmogaus per vidurį“ ataka, kurios metu galėjo būti pavogtas slaptažodžio pakeitimo raktas. Hackerone pranešimas užpildytas 2017 metų vasario 15 dieną, pažeidžiamumo grėsmė įvertinta kaip aukšta, dėl spragos suradimo pranešėjui išmokėta 200 JAV dolerių premija. Pranešimo identifikacinis kodas Hackerone sistemoje – 206650.

Spraga elementari - slaptažodžio keitimo nuorodos, kurią vartotojas gauna į savo elektroninio pašto dėžutę, pradžioje nurodomas norimas naudoti protokolas, šiuo atveju, jis nurodytas kaip http://, kas reiškia, jog paspaudus nuorodą užklausa bus siunčiama per nešifruotą HTTP protokolą. Naudojant neapsaugotą HTTP protokolą, įsilaužėlis, galintis matyti siunčiamas užklausas, gali matyti ir per nuorodą pateikiamus parametrus, įskaitant perduodamą raktą naudojamą slaptažodžio keitimui. Nagrinėjamu atveju, praktinė ataka yra sudėtinga, nes raktą galima panaudoti tik vieną kartą, todėl įsilaužėliui jį perskaičius reikėtų raktą panaudoti prieš tai padarant vartotojui, tačiau tai vis dėlto yra įmanoma naudojantis automatinėmis priemonėmis. Pažeidžiamumo panaudojimas ribotas, bet tai puikus pavyzdys iliustruojantis, kodėl būtina visą privačią informaciją perduodi naudojant šifruotą komunikacijos kanalą.

Spraga ištaisyta pašalinant tarpininką Mandrill ir siunčiant HTTPS nuorodas nukreipiant vartotoją tiesiai į paslaugos tiekėjo serverį.

2.3.2. Vartotojų slaptažodžių saugojimas

Didžioji dauguma saityno programų įgyvendina tradicinę prisijungimo sistemą – naudojantis vartotojo vardu ir slaptažodžiu. Tai reiškia, jog yra būtinybė saugoti sistemos vartotojų prisijungimo slaptažodį. Lengviausias būdas – laikyti slaptažodį kaip ir visus kitus duomenis - nedarant jokių pakeitimų, paprasčiausiai kaip lauką vartotojo įraše duomenų bazėje. Jeigu būtų be abejonės žinoma, jog sistema neturi nei vieno pažeidžiamumo leidžiančio įsilaužėliui gauti prieigą prie laikomų slaptažodžių, tuomet sprendimas laikyti slaptažodžius paprastuoju tekstu (ang. *plain text*) nesukeltų problemų. Praktika rodo, jog tobulų sistemų nėra, net tokios didelės įmonės kaip Yahoo, Ebay ar JP Morgan yra patyrusios duomenų bazių nutekinimus [26-27]. Beveik pusės duomenų nutekėjimo atvejų priežastimi yra ne pažeidžiamumas sistemoje, o žmogiškasis faktorius, pavyzdžiui lengvai atspėjami ar pernaudojami darbuotojų slaptažodžiai bei darbuotojų papirkinėjimas siekiant gauti prieigą prie

įmonės vidinių resursų [28]. Tai reiškia, jog slaptažodžius sauganti įmonė privalo būti atsakinga ne tik už slaptažodžių laikymą bet ir jų saugumą tuo atveju, jeigu jie būtų nutekinti.

2.3.2.1. Neteisingo vartotojų slaptažodžių saugojimo pavyzdžių analizė

Šiame poskyryje analizuojami pavyzdžiai parodantys neteisingas vartotojų slaptažodžių saugojimo praktikas. Analizės metu gautos įžvalgos naudojamos metodinėje dalyje svarstant sprendimą tinkamam slaptažodžių saugojimui.

2.3.2.1.1. Citybee duomenų bazės nutekėjimas

Vienas žymiausių duomenų bazių nutekėjimų Lietuvoje įvyko 2021 metais, kuomet buvo paviešinta 110 tūkstančių bendrovės Citybee klientų asmens duomenų rinkinys CSV formatu. Po valstybinės duomenų apsaugos inspekcijos atlikto tyrimo nustatyta, jog Citybee neužtikrino tinkamo asmens duomenų saugumo, viena sprendimo priežastis – netinkamai laikomi vartotojų slaptažodžiai [29]. Slaptažodžiams saugoti pritaikytas SHA-1 maišos algoritmas, kurio naudojimas jautriai informacijai laikyti nerekomenduojamas jau nuo 2011 metų dėl esminių saugumo trūkumų įrodytų analizėse ir teorinėse atakose, o 2017 metais taip pat buvo parodytas pirmasis praktinis kolizijos generavimo pavyzdys [30]. Citybee vartotojų slaptažodžių duomenų nutekinimas ir nepakankamas slaptažodžių apsaugos mechanizmas, reiškia, jog sistemos vartotojų slaptažodžiai motyvuotam kibernetiniam nusikaltėliui yra lengvai prieinami ir gali būti panaudoti norint įsilaužti į kitas sistemas, kuriose asmuo naudojo tuos pačius prisijungimo duomenis kaip ir Citybee sistemoje. Apart finansinės žalos, įmonė Citybee patyrė nuostolį reputacijos atžvilgiu - įvykis pritraukė žiniasklaidos dėmesį, straipsniuose buvo pabrėžiama, kad Citybee neužtikrino vartotojų duomenų saugumo.

2.3.2.1.2. Adobe duomenų bazės nutekėjimas

Kitas atvejis, parodantis, jog net tarptautiniai kompiuterių programinės įrangos kūrimo milžinai yra nesaugūs nuo duomenų nutekėjimų ir šiurkščių slaptažodžių laikymo klaidų tai 2013 metais įvykęs Adobe duomenų bazės nutekėjimas. Šis incidentas paveikė bent 38 milijonų aktyvių vartotojų duomenis. Apart vartotojų vardų, kontaktų, kreditinių kortelių duomenų ar kitos informacijos, taip pat buvo nutekinti ir užšifruoti vartotojų slaptažodžiai [31]. Adobe slaptažodžiams šifruoti naudojo Trigubą Duomenų Šifravimo Algoritmą (ang. *Triple Data Encryption Algorithm – 3DES*). 3DES algoritmas yra simetrinis – tas pats raktas naudojamas ir duomenų šifravimui ir iššifravimui [32]. Tai yra prastas pasirinkimas slaptažodžių saugojimui, kadangi duomenų bazės nutekinimo atveju, užtenka nulaužti vieną šifravimo algoritmą ir surasti vieną raktą, kuris leistų greitai iššifruoti visus vartotojų slaptažodžius. Priklausomai nuo kibernetinės atakos masto, kartu su vartotojų duomenų baze gali būti ir pažeista vieta, kurioje saugojami šifravimo raktai - tokiu atveju slaptažodžių šifravimas tampa bereikšme apsaugos priemone. Slaptažodžius šifruojant taip pat atsiranda vidinio užpuoliko rizika – darbuotojas, turintis prieigą prie šifravimo raktų ir duomenų bazės, gali lengvai slaptažodžiais pasinaudoti pats ar juos nutekinti. Konkrečiu 3DES naudojimo atveju, 3 metai po Adobe duomenų nutekinimo, algoritme rasta nauja saugumo spraga, dėl kurios nacionalinis standartų ir technologijų institutas (*ang. National Institute of Standards and Technology – NIST*) 3DES paskelbė nerekomenduojamu naudoti naujoms sistemoms nuo 2017 metų ir nebenaudotiną visoms sistemoms nuo 2023 metų [33].

2.4. Injekcijos pažeidžiamumai

Injekcijos pažeidžiamumai yra vieni dažniausių ir neretai turi labai rimtų pasekmių. OWASP Top 10 sąraše injekcijos pažeidžiamumai užėmė pirmąją vietą nuo pat 2007 metų ir tik 2021 metais nukrito į trečiąją sąrašo vietą. Dabar tai daugumai puikiai žinomi pažeidžiamumai, turintys efektyvias prevencijos priemones, tačiau kartais injekcijos spragų vis dar galima atrasti net didžiausių įmonių saityno programose. Vieni pavojingiausių ir dažniausių injekcijos pažeidžiamumų tai XSS ir SQLi.

2.4.1. XSS pažeidžiamumai

XSS pažeidžiamumas leidžia užpuolikui įterpti savo kodą į kliento pusės saityno programos dalį, tuomet šis kodas yra vykdomas aukos naršyklėje [34]. XSS pažeidžiamumai paprastai skirstomi į tris pagrindinius tipus:

1. Išsaugotas arba nuolatinis XSS (ang. *stored or persistent XSS*), kuomet kenksmingas kodas yra laikomas saityno programos serveryje [35]. Tai viena pavojingiausių XSS apraiškų, nes pažeidžiamais tampa visi vartotojai, kuriems pateikiamas resursas užkrėstas kenksmingu kodu. Spragos atsiradimo priežastis – pažeidžiamas programos kodas, kuris tinkamai nepatikrina vartotojo įvesties ir priima kenksmingo kodo dalį, tuomet ją išsaugo ir pateikia kartu su užkrėstu resursu klientui. Kliento pusės kodas neatlieka veiksmų užtikrinti, jog atvaizduojant užkrėstą resursą vartotojo įvestis nėra traktuojama kaip kodas, todėl kenksmingas kodas yra vykdomas kaip teisėta programos dalis. Vienas paprasčiausių atakos pavyzdžių – XSS pažeidžiamumas komentarų funkcionalume. Užpuolikas palieka komentarą su kenksmingu kodu, tuomet šis komentaras išsaugomas saityno programos serveryje. Kuomet bet koks kitas vartotojas atsidaro puslapį, kuriame turėtų būti atvaizduojami komentarai, jam pateikiamas komentaras su kenksmingu kodu ir šis kodas yra naršyklėje įvykdomas išsiunčiant vartotojo slapukuose esantį prieigos raktą užpuolikui.
2. Atsispindintis arba vienkartinis XSS (ang. *reflected or non-persistent XSS*) - šis pažeidžiamumas naudoja svetainės elementus, kurie atvaizduoja kliento pateiktus duomenis [35]. Saityno programos kodas neužtikrina, jog kliento pateikti duomenys nebus interpretuoti kaip kodo dalis. Klientui pateikus užklausą su kenksmingu kodu, gaunamas atsakas iš serverio, kur pateiktas kodas yra vykdomas kaip teisėta programos dalis. Norint priversti auką išsiųsti užklausą su užkrėstu kodu, jis pateikiamas kartu su nuoroda į pažeidžiamą svetainę. Vienkartinį XSS panaudoti yra sunkiau, kadangi tai reikalauja pasinaudoti socialine inžinerija, norint priversi auką paspausti ant kenksmingos nuorodos. Tačiau vykusi ataka taip pat leis pavogti informaciją iš aukos slapukų, todėl jos nuvertinti negalima.
3. XSS paremtas dokumentų objektų modeliu (ang. *Document Object Model – DOM*), kuomet pateikta informacija atvaizduojama kliento pusės kode, be sąveikos su serveriu [35]. DOM paremtas XSS išnaudoja tik kliento kodo dalies patikrinimo trūkumą, tam, kad teisėtas programos kodas veiktų netikėtu būdu [19]. Kaip ir vienkartinio XSS atveju, pažeidžiamumui išnaudoti auka privalo atsidaryti nuorodą su patalpinta kenksminga parametro reikšme.

2.4.1.1. XSS pažeidžiamumų pavyzdžių analizė

Siekiant daugiau suprasti apie tai kaip XSS pažeidžiamumai gali atrodyti tikroje sistemoje, pasirinkta paanalizuoti jų pavyzdžius aprašytus Hackerone platformos pranešimuose.

2.4.1.1.1. Vienkartinis XSS pažeidžiamumas įmonės Equifax svetainėje

Aprašomas vienkartinis XSS pažeidžiamumas įmonės Equifax svetainės slaptažodžio nustatymo iš naujo puslapyje, leidęs sukurti nuorodą, kurią atsidarius būtų pavogtas vartotojo prieigos raktas, pakeistas svetainės turinys ar įvykdytos kitos XSS atakos. Pranešimas apie pažeidžiamumą buvo patalpintas Hackerone platformoje 2022 metų sausio 29 dieną, pažeidžiamumas įvertintas kaip vidutinės grėsmės. Hackerone puslapyje pranešimą galima rasti pagal identifikacinį kodą 1463638.

Slaptažodžio nustatymo iš naujo formoje vartotojo vardo reikšmė neturėjo tinkamos patikrinimo, ko pasėkoje įvedus JavaScript kodo dalį į vartotojo vardo kintamąjį nuorodoje, kodas buvo įvykdomas kaip teisėta programos dalis. Kadangi vartotojo vardas atvaizduojamas teksto laukelio HTML elemente, visų pirma buvo reikalinga išeiti iš HTML elemento ribų - tai padaryta į kintamąjį nuorodoje pateikus simbolius *"<*. Tuomet sukurtas papildomas paveikslėlio tipo HTML elementas, kuriame pridėtas parametras *onerror*. Parametras *onerror* nurodo JavaScript kodą, kuris turi būti paleistas tuo atveju, jeigu paveikslėlio užkrauti nepavyksta. Kadangi paveikslėlio parametras *src*, nurodantis paveikslėlio buvimo vietą, pateiktas kaip neteisingas, bandant užkrauti paveikslėlį veiksmas nepavyks ir bus iškviestas kodas esantis parametre *onerror*. Pavyzdys kaip atrodytų saityno programos kliento pusės kodas įterpus kenksmingą kodo dalį pateikiamas apačioje (2.1 pav.). Kodo dalis, kuri būtų įterpta užpuoliko pažymėta raudonai.

```
<input class="block" autocomplete="off" type="text" name="username" value=""><img src=x onerror=alert(1)>"
```

2.1 pav. HTML kodo dalis iliustruojanti vienkartinio XSS rezultatą

Taigi, užpuolikas galėjo sukurti nuorodą, kuri vykdys jo nurodytą JavaScript kodą.

2.4.1.1.2. Nuolatinis XSS pažeidžiamumas įmonės Shopify svetainėje

Shopify priklausančioje svetainėje www.linkpop.com rastas nuolatinis XSS pažeidžiamumas nuorodos parametre, kurį panaudojus būtų įterptas užpuoliko pasirinktas kodas į naujai sukurtą nuorodos parametrą užpuoliko profilyje. Hackerone platformoje apie pažeidžiamumą pranešta 2022 metų sausio 5 dieną, pažeidžiamumo grėsmė įvertinta kaip aukšta, už jo radimą pranešėjui išmokėta 1600 JAV dolerių premija. Pranešimo identifikacijos kodas Hackerone puslapyje – 1441988.

Svetainė leidžia vartotojui susikurti savo profilį ir jame pridėti nuorodas į vartotojo pasirinktas svetaines. Vartotojo profilį vėliau gali peržiūrėti kiti vartotojai, jiems bus atvaizduojamos pridėtos nuorodos HTML elemento pagalba, kurio *href* atributas bus lygus nuorodos reikšmei. Pridedant naują svetainę kliento pusės kodas neleido įvesti reikšmės, kuri neatitiktų galiojančios nuorodos šablono. Tačiau jeigu pasitelkiama programa, leidžianti redaguoti siunčiamas užklausas, kaip pavyzdžiui BurpSuite Proxy, tuomet kliento pusės draudimas įvesti netinkamą nuorodos reikšmę apeinamas. Apėjus kliento pusės apribojimus vartotojas galėjo visiškai kontroliuoti HTML elemento *href* atributų reikšmes. Atributas *href* ypatingas, nes jame galima naudoti JavaScript pseudoprotokolą, tai yra vykdyti JavaScript kodą atributo viduje. Taigi, vartotojas savo profilyje galėjo pridėti savo JavaScript kodą sukurdamas naują nuorodą, kurios pradžioje simbolių eilutė *javascript:* o toliau sektų norimas vykdyti JavaScript kodas (2.2 pav.).

```
'<a class="image" href="javascript:alert(document.domain)" target="_blank"
```

2.2 pav. HTML kodo dalis iliustruojanti nuolatinio XSS pažeidžiamumo rezultatą

Tokiu būdu užpuolikas galėjo savo profilyje prisidėti nuorodą ir jos reikšmę prilyginti JavaScript kodui, skirtam pavogti prieigos raktus iš slapukų. Tuomet užpuolikas galėtų laukti, kol kiti sistemos vartotojai atsitiktinai atsidarys jo profilį arba bandyti išsiųsti nuorodas į savo profilį norimiems taikiniams ir taip paprastai pasisavinti bet kurių jo profilio lankytojų prieigos raktus.

2.4.1.1.3. Vienkartinis XSS naudojantis HTTP antrašte svetainėje Omise

Paskutinis nagrinėjamas pavyzdys - tai vienkartinis XSS pažeidžiamumas svetainėje Omise, tačiau šį kartą kenksmingas kodas yra pateikiamas per X-Forwarded-Host antraštę. Pranešimas apie pažeidžiamumą Hackerone platformoje užpildytas 2021 metų lapkričio 6 dieną, pažeidžiamumas įvertintas kaip vidutinės grėsmės, už jo radimą išmokėta 200 JAV dolerių premija. Pranešimą Hackerone platformoje galima rasti naudojantis identifikaciniu kodu 1392935.

Kreipiantis į www.omise.co buvo galima pridėti savo nurodytą antraštę, pranešėjas tai iliustruoja su antrašte X-Forwarded-Host. Pridė toji antraštė vėliau atvaizduojama puslapyje nepritaikius jokio filtravimo. Tai reiškia, jog užpuolikas antraštės reikšmę galėjo įvesti analogiškai, kaip aptarta pavyzdyje 2.4.1.1.1 ir taip paleisti savo norimą JavaScript kodą. Šiuo atveju, praktinės atakos galimybės ribotos, kadangi pažeidžiamumo negalima išnaudoti paprasčiausiai atsiunčiant nuorodą, reikėtų kito būdo, kuriuo aukai būtų nustatyta užpuoliko pasirinkta antraštė.

2.4.2. SQL injekcija

Struktūrizuota Užklausų Kalba (ang. *Structured Query Language - SQL*) skirta bendrauti su duomenų baze ir modifikuoti joje esančius duomenis [36]. SQL injekcijos metu užpuolikai gali vykdyti kenkėjiškas užklausas su tikslu neteisėtai modifikuoti ar pavogti duomenų bazėje saugojamus duomenis. Pažeidžiamumo priežastis – saityno programos kūrėjai pasitiki vartotojo įvestimi ir jos tinkamai neapdoroja, to pasėkoje kenkėjiška informacija yra įdedama į programoje vykdomą SQL užklausą.

Vienas paprasčiausių SQL injekcijos pavyzdžių, tai SQL injekcijos panaudojimas norint apeiti slaptažodžio reikalavimą prisijungimo metu. Prisijungimo metu vartotojo įvestas vardas ir slaptažodis patikrinami programos serveryje naudojantis užklausa:

```
SELECT * FROM Users WHERE username = 'vartotojo_įvestas_vardas' AND password =
'vartotojo_įvestas_slaptažodis'
```

Pavyzdyje mėlynai pažymėti SQL raktažodžiai, o raudonai – vartotojo įvesta informacija. Užklausa suranda duomenų bazėje vartotoją, kurio vardas ir slaptažodis atitinka vartotojo įvestą informaciją. SQL injekcijos metu įsilaužėlis į vartotojo vardo laukelį įveda aukos prisijungimo vardą, o slaptažodžio lauke simbolių eilutę „'1' OR '1' = '1'“. Tuomet užklausa atrodo taip:

```
SELECT * FROM Users WHERE username = 'vartotojo_įvestas_vardas' AND password = '1' OR
'1' = '1'
```

SQL užklausoje OR yra loginis operatorius, reiškiantis „arba“, taigi užklausa dabar grąžins įvesto vartotojo duomenis jeigu įvestas slaptažodis sutampa arba jeigu 1 yra lygu 1. Kadangi 1 visuomet lygu 1 ir naudojamas operatorius „arba“, užklausos slaptažodžio tikrinimo dalis bus praleista. Tai labai paprastas pavyzdys iliustruojantis patį pažeidžiamumo veikimo principą.

SQL injekcijos pažeidžiamumai yra vieni pavojingiausių, kadangi jie duoda užpuolikui prieigą prie programos duomenų bazės, kurioje dažnai laikoma visa programoje naudojama informacija, įskaitant vartotojų duomenis. Užpuolikas gali duomenis pavogti, ištrinti, neteisėtai

modifikuoti, už jų grąžinimą prašyti išpirkos iš paslaugų tiekėjo. Pavogti duomenys taip pat gali būti panaudoti kitų sistemų nulaužimui. Todėl SQL injekcijos pažeidžiamumo prevencija yra kritinės svarbos.

2.4.2.1. SQL injekcijos pavyzdžių analizė

Norint suprasti SQL injekcijos pažeidžiamumų specifiką ir plačiau pažinti jų atsiradimo priežastis, pasirinkta paanalizuoti SQL injekcijos pažeidžiamumų pavyzdžius versle naudojamose saityno programose.

2.4.2.1.1. SQL injekcija Atavist puslapyje

Apžvelgiama SQL injekcijos spraga Atavist saityno programoje, kuri sukurta naudojantis Newspack atvirojo kodo platforma. Pažeidžiamumas leido užpuolikui keisti dalį SQL užklausos ir taip duomenų bazėje atlikti neautorizuotus veiksmus. Apie pažeidžiamumą pranešta 2020 metų lapkričio 20 dieną, pažeidžiamumo grėsmės įvertinimas aukštas, už pranešimą išmokėta 200 JAV dolerių premija. Pranešimo identifikacinis kodas platformoje Hackerone – 1039315.

Pranešėjas nurodo, jog spraga buvo atrasta naudojantis automatiniu įrankiu „sqlmap“, kuris skirtas SQL injekcijos spragoms aptikti. Įrankis nurodo, jog Atavist puslapio „search“ parametre rastas SQL injekcijos pažeidžiamumas. Spragos buvimas įrodomas parametre įterpiant SQL funkciją „sleep“, kuri pavėlina SQL komandos vykdymą nurodytu laiko tarpu. Šiuo atveju duomenų bazei nurodoma laukti 5 sekundes ir patikrinama, jog atsakymas iš serverio gaunamas praėjus šiek tiek daugiau nei 5 sekundėms. Pranešime nenurodoma, kaip atrodė pažeidžiama SQL užklausa, tačiau galima spėti, jog ji turėjo būti sudaryta maždaug taip:

```
SELECT * FROM organizations WHERE organization_name LIKE vartotojo_įvestis
```

Kadangi vartotojo įvestis nebuvo tinkamai filtruojama ir dedama tiesiai į SQL užklausą, užpuolikas galėjo valdyti SQL užklausos pabaigą ir atlikti savo pasirinktas SQL operacijas.

2.4.2.1.2. SQL injekcija valstybiniame JAV duomenų katalogo puslapyje

Nagrinėjama SQL injekcijos spraga JAV valstybiniame viešųjų duomenų rinkinių katalogo puslapyje, kuri buvo atrasta per HTTP antraštę. Apie pažeidžiamumą Hackerone platformoje pranešta 2017 metų gruodžio 13 dieną. Pažeidžiamumo grėsmė įvertinta kaip kritinė, pranešėjui išmokėta 2000 JAV dolerių premija. Pranešimą Hackerone puslapyje galima rasti identifikaciniu kodu 297478.

Pranešime teigiama, jog HTTP antraštėje *User-Agent* nustatyta SQL injekcijos spraga. Pažeidžiamumo egzistavimas patvirtintas į saityno programos serverį nusiuntus užklausą, kurios *User-Agent* antraštėje nurodyta matematinė operacija SQL funkcijos „sleep“ viduje. Funkcijoje „sleep“ nurodoma reikšmė „5*5“, po kurios atsakymas iš serverio gaunamas po 25 sekundžių. Tai parodo, jog aritmetinė operacija buvo įvykdyta ir tuomet vykdoma SQL funkcija „sleep(25)“.

Apžvelgiamas pavyzdys parodo, jog SQL injekcija gali būti įvykdyta ne vien naudojantis parametrais, tačiau ir per HTTP antraštes - tai situacija apie kurią dažnai nėra pagalvojama.

2.4.2.1.3. SQL injekcija CONTACT finansinių paslaugų saityno programoje

Apžvelgiamas SQL injekcijos pažeidžiamumas vienoje tarptautinių mokėjimų sistemos CONTACT saityno programoje. Pranešimas, aprašantis pažeidžiamumą, Hackerone puslapyje patalpintas 2020 metų birželio 19 dieną, pažeidžiamumas pripažintas kritinės grėsmės, už jo

pranešimą išmokėta 5500 JAV dolerių premija. Pranešimo identifikacinis kodas Hackerone platformoje – 816254.

Pranešėjas nurodo, jog pažeidžiamumas kilo dėl nepakankamos parametro *SCEN_ID* validacijos, ko pasėkoje įsilaužėlis galėjo įterpti savo SQL sakinius į naudojamos SQL užklausos WHERE sąlygą. Pranešime pateikiama numanoma pažeidžiamos SQL užklausos struktūra:

```
SELECT * FROM tblName WHERE id='vartotojo_įvestis' order by STEP
```

Pranešimas unikalus tuo, jog pažeidžiamumas panaudojamas priverčiant programos serverį vykdyti užpuoliko nurodytą kodą. Tai įmanoma naudojantis SQL komanda EXEC, kuri gali vykdyti saugomas procedūras (ang. *stored procedure*). Saugomos procedūros - tai SQL kodo fragmentai, atliekantys iš anksto numatytas funkcijas, kuriuos vėliau galima vykdyti EXEC komandai nurodant saugomos procedūros pavadinimą.

Naudojantis EXEC komanda, įsilaužėlis gali priversti SQL serverį išsiųsti užklausą į įsilaužėlio nurodytą resursą, pavyzdžiui, jo valdomą serverį. Tai vadinamoji SQL injekcijos „už ribų“ (*ang. out-of-bound*) ataka, kurios metu programa negrąžina SQL užklausos rezultato vartotojui tiesiogiai, o vietoje to atsakymui grąžinti pasitelkiamas kitas, užpuoliko nurodytas komunikacijos kanalas.

Taip iliustruojama, jog net jeigu iš pirmos pažiūros SQL injekcijos pažeidžiamumas negrąžina vartotojui rezultato, vis dėlto yra būdų įvykdyti sėkmingą ataką.

2.5. Pažeidžiamumų analizės apibendrinimas

Apžvelgtos saityno programose dažniausiai pasitaikančios saugumo spragos – IDOR, nesaugus slaptažodžių laikymas, nesaugi vartotojo sąsaja su serveriu, XSS ir SQL injekcija. Išanalizuoti šių pažeidžiamumų pavyzdžiai tikrose sistemose. Nustatytos pagrindinės spragų atsiradimo priežastys:

- Norint pasiekti programoje naudojamus objektus naudojami jų tiesioginiai identifikatoriai.
- Vartotojui bandant pasiekti resursą, nėra įsitikinama, jog jis turi tinkamą autorizaciją.
- Slaptažodžiai programų duomenų bazėse laikomi jiems pritaikant pažeidžiamą maišos algoritmą arba juos šifruojant.
- Saityno programoje komunikacijai tarp vartotojo ir serverio naudojamas nešifruotas HTTP protokolas.
- Vartotojo įvestis nėra tinkamai filtruojama.
- Neapdorota vartotojo įvestis atvaizduojama saityno programos vartotojo sąsajoje, kur ji gali būti interpretuojama kaip teisėta programos kodo dalis.
- Neapdorota vartotojo įvestis naudojama sudarant SQL užklausas, kur vartotojo įvesti SQL raktažodžiai interpretuojami ir vykdomi kaip SQL užklausos dalis.

Efektyviai pažeidžiamumų prevencijai saityno programose turi būti pritaikyti metodai sprendžiantys minėtąsias spragų atsiradimo priežastis.

3. PAŽEIDŽIAMUMŲ PREVENCIJOS METODAI

Norint užtikrinti, kad spragų atsiradimo būtų išvengta, privalu atsirinkti efektyvius prevencijos metodus. Šiame skyriuje apžvelgiami literatūroje siūlomi metodai. Pasitelkiant praktines žinias šie metodai išanalizuojami, atsižvelgiama į metodų pritaikymą spragų analizės metu aptartuose pavyzdžiuose.

3.1. Prieigos kontrolės pažeidžiamumų prevencija

3.1.1. IDOR pažeidžiamumo prevencija

IDOR prieigos kontrolės problema turi būti sprendžiama dviem žingsniais:

- Nenaudojant tiesioginio identifikatoriaus, jį pakeičiant unikalia sudėtingai atspėjama reikšme.
- Taikant tinkamą prieigos kontrolės politiką, įsitikinant, jog vartotojas yra autorizuotas prieiti prie prašomo resurso [1].

3.1.1.1. Reikalavimai objektų identifikatoriams

Norint apsunkinti potencialios prieigos kontrolės spragos eksploatavimą, rekomenduojama tokias tiesiogines objektų nuorodas kaip duomenų bazių raktus pakeisti į sistemoje sugeneruojamas kuo sunkiau atspėjamas, unikalias reikšmes, kurios garantuotų anonimiškumą ir neatsekamumą [37].

Generuojamo identifikatoriaus unikalumo būtinybė akivaizdi – jeigu būtų sugeneruojami du vienodi identifikatoriai, tuomet programa vienu identifikatoriumi kreiptųsi į du objektus, jie taptų neatskiriami.

Identifikatorius privalo būti aukštos entropijos tam, kad potencialūs įsilaužėliai, supratę identifikatorių kūrimo mechanizmą, negalėtų paprasčiausiai nuspėti koks objektas būtų susietas su kokiu identifikatoriumi ir taip maksimizuoti potencialaus prieigos kontrolės pažeidžiamumo išnaudojimą.

Anonimiškumas reiškia, jog identifikatorius negali būti panaudotas siekiant atsekti, kokiam objektui jis sukurtas. Neatsekamumas reikalingas norint užtikrinti, kad piktybinis veikėjas negalėtų nustatyti ar du identifikatoriai nebuvo sukurti tos pačios sistemos [37].

Netiesioginiai identifikatoriai, atitinkantys minėtuosius reikalavimus, naudojami nurodant objektą vartotojų užklausose. Tuomet saityno programos serverio kode sudaromi žemėlapiai, kuriuose netiesioginiai identifikatoriai susiejami su tų objektų tiesioginiais identifikatoriais. Taip užtikrinama, jog viešai objektai pasiekiami tik žinant jų netiesioginius identifikatorius, o tiesioginiai identifikatoriai naudojami tik programos viduje.

3.1.1.2. Prieigos kontrolės užtikrinimas

Siekiant užtikrinti tinkamą prieigos kontrolę, siūloma naudoti rolėmis grindžiamą autorizacijos modelį, kuriame kiekvienam vartotojui priskiriama atitinkama autorizacijos rolė – pavyzdžiui, paprastas vartotojas arba administratoriaus privilegijas turintis vartotojas [38]. Tuomet programoje vykdyti prieigos kontrolės politiką, kuri nustato, jog prieiga prie resurso atmetama pagal nutylėjimą. Tai yra, jeigu resursui nenurodoma, kokios rolės vartotojai jį galį

pasiekti, jo negalės pasiekti jokie vartotojai. Ši politika užtikrina, jog rolės nustatymas resursui nebus pamirštas.

Pirmajame aptartame IDOR prieigos kontrolės pažeidžiamumo pavyzdyje, Nextcloud verslo atstovas pateikia tiek originalų kodą su spraga, tiek siūlomą problemos sprendimą. Pateiktame kodo fragmente matoma esminė prieigos kontrolės spraga – kviečiant funkciją įrenginio prieigos rakto trynimui (markTokenForWipe(int $id)) nėra tikrinama, ar vartotojas, kuris kviečia funkciją, ir vartotojas, kuriam priklauso įrenginys, sutampa. Čia pat pasiūlomas ir sprendimas - prieš atliekant trynimo veiksmą yra gaunamas vartotojo, kuriam priklauso įrenginio raktas, identifikatorius ir jis palyginamas su vartotojo, kuris kviečia funkciją, identifikatoriumi. Kodas pateikiamas apačioje (3.1 pav.), raudonai pažymėtos kodo eilutės su pažeidžiamumu, kurios bus pašalintos, o tuo tarpu žaliai pažymėtos eilutės tai siūlomi pakeitimai.

```
30  -   public function markTokenForWipe(int $id): bool {
31  +   public function markTokenForWipe(int $id, string $userId): bool {
32          $token = $this->tokenProvider->getTokenById($id);
33
34  -       if (!($token instanceof IWipeableToken)) {
35  +       if (!$token instanceof IWipeableToken || $token->getUID() !== $userId) {
36              return false;
37          }
```

3.1 pav. Nextcloud pažeidžiamumo kodo fragmentas

Svarbu pabrėžti, kad vartotojas negali pats pateikti savo identifikatorius ar rolės kaip parametro kartu su užklausa, nes tuomet jį būtų galima lengvai pakeisti norint apsimesti kitu vartotoju. Identifikatorius turi būti randamas jau sistemos viduje, pagal prieigos raktą, kuris patvirtina vartotojo tapatybę. Esant saugiam autentifikacijos procesui vartotojas negali gauti rakto priklausančio kitam vartotojui.

Norint užkirsti kelią pirmuose dvejuose pavyzdžiuose nagrinėtoms IDOR problemoms (poskyriai 2.2.1.1.1, 2.2.1.1.2), programuojant privalu įsitikint, jog atliekant veiksmus su objektais, kurie priklauso tam tikram vartotojui, objekto savininkas bei vartotojas, norintis atlikti veiksmą, sutampa. Tam galima naudoti unikalų vartotojo identifikatorių ir saugoti jį objekto įraše duomenų bazėje.

Trečiame pavyzdyje (poskyris 2.2.1.1.3) esančios prieigos kontrolės problemos galima išvengti pasitelkiant principą, jog vartotojas turi būti identifikuojamas pagal jo prieigos raktą, o ne pagal lengvai modifikuojamą parametrą užklausoje.

3.2. Kriptografinių pažeidžiamumų prevencija

3.2.1. Sprendimas saugiai sąsajai su vartotoju

Norint užtikrinti, jog sąsaja su vartotoju saugi, privalu užtikrinti, jog visi duomenys kelyje tarp vartotojo ir paslaugos tiekėjo serverių yra šifruoti. Saugiai sąsajai sukurti naudojamas puikiai žinomas HTTPS protokolas. HTTPS patvirtina paslaugos tiekėjo autentiškumą naudojantis skaitmeniniais sertifikatais bei įgyvendina transporto lygio saugumo protokolą, kuris atlieka perduodamų duomenų šifravimo funkciją [39]. HTTPS būtinybė perduodant privačią informaciją

nenuginčijama, tačiau, kaip pastebima nagrinėjant pavyzdį aprašytą skyriuje 2.3.1.1.1, vis dar pasitaiko atvejų kai HTTPS nėra naudojamas.

Norint užtikrinti informacijos saugumą, rekomenduojama naudoti HTTPS visoje saityno programos apimtyje, net tada, kai nėra perduodama privati informacija. Tokiu būdu pašalinama rizika, jog per žmogiškąją klaidą bus tinkamai neapsaugos užklausos, kurioms šifravimas būtinas.

Siūloma nepalaikyti paprastojo HTTP protokolo ir užtikrinti HTTPS naudojimą pasitelkiant saugumo antraštes. HTTP griežto transporto saugumo (ang. *HTTPS Strict Transport Security – HSTS*) politika užtikrina, jog visa komunikacija tarp vartotojo ir paslaugos tiekėjo serverių bus atliekama per HTTPS ir bus užkirstas kelias protokolo pažeminimo atakoms, kurių metu vartotojas priverčiamas naudoti HTTP vietoje HTTPS [40]. HTTPS griežto transporto saugumo politika įgyvendinama per Strict-Trasport-Security antraštę. Antraštėje nurodomos direktyvos, aprašančios politikos veikimą:

- **Galiojimo laiko direktyva *max-age*.** Vartotojo naršyklė, gavusi Strict-Trasport-Security antraštę, patikrina ir išsisaugo direktyvos *max-age* reikšmę. Tuomet naršyklė neleidžia vartotojui siųsti užklausų į tą pačia saityno programą per nesaugų HTTP protokolą, kol nepasibaigia nurodytas politikos galiojimo laikas. Galiojimo laikas nurodomas sekundėmis, dažniausiai rekomenduojama galiojimo laiką nustatyti lygų bent metams (31536000 sekundėms) [19].
- **Direktyva *includeSubDomain*.** Ši direktyva nurodo, jog HSTS politiką privalu taikyti ir visiems svetainės subdomenams. Direktyvos naudojimas naudingas siekiant apsaugoti visą svetainės funkcionalumą.
- **Direktyva *preload*.** Ji reiškia, jog saityno programa užregistruota Google platformoje ir HSTS politikos naudojimo faktas yra iš anksto įkeltas į Google Chrome naršyklę ir taip pat visas kitas naršykles, kurios naudoja Google *preload* sąrašą. Buvimas Google *preload* sąraše užtikrina, kad naršyklės, naudojančios šį sąrašą, iš anksto žinos apie HSTS direktyvos taikymą svetainei ir nebandys siųsti net pirmosios nesaugios užklausos, po kurios būtų įrašyta HSTS direktyva. Tam, kad patekti į Google *preload* sąrašą, neužtenka tik pridėti antraštės - privaloma užpildyti formą ir laukti, kol ji bus patvirtina Google. Tik po patvirtinimo saityno programos Strict-Trasport-Security antraštėje gali būti pridėta *preload* direktyva. Buvimas Google *preload* sąraše yra labai pageidautinas, dėl minėtojo fakto, kad dauguma vartotojų naršyklių iš anksto žinos apie HSTS politikos buvimą ir negalės siųsti nei vienos užklausos per nesaugų HTTP protokolą.

Spraga aprašyta poskyryje 2.3.1.1.1 parodo, kaip svarbu, kad būtų kuo mažiau trečiųjų šalių, kurioms būtų siunčiama privati informacija. Praktikoje pastebimi atvejai, kuomet raktai yra nutekinami į saityno programose naudojamus analitikos įrankius, kaip kad Google Analytics. Raktų nutekinimo trečiosioms šalims reiktų vengti, nes tai dar viena vieta, kurioje, atsiradus pažeidžiamumui, įsilaužėlis galėtų gauti paslaugos tiekėjo privačią informaciją. Dažnai trečiųjų šalių programinės įrangos saugumo negalima objektyviai įvertinti ar pagerinti, todėl sunku pamatuoti riziką perduodant informaciją į jų rankas. Konkrečiu Mandrill atveju, sistema leido naudoti tiek HTTPS, tiek HTTP, kas yra nepageidaujama perduodant privačią informaciją.

3.2.2. Sprendimas saugiam vartotojų slaptažodžių saugojimui

Slaptažodžiai turi būti saugomi jiems pritaikius maišos algoritmus, šie algoritmai informaciją paverčia fiksuoto ilgio simbolių eilute ir, priešingai nei įprasti šifravimo algoritmai, neturi atvirkštinės funkcijos, kuri leistų vėl matyti atvirojo teksto informaciją. Tai svarbu todėl, kad užtikrinti, jog net pats paslaugos tiekėjas neturi lengvo būdo atšifruoti vartotojų slaptažodžių ar nutekinti raktų, kurie leistų tai padaryti įsilaužėliams, kaip iliustruota pavyzdyje su nutekinta Adobe duomenų baze. Vienintelis būdas sužinoti įvestą slaptažodį turint jo maišos santrauką yra spėti, koks slaptažodis galėjo būti, jam pritaikyti tą patį maišos algoritmą ir tuomet palyginti gautą rezultatą su turima maišos santrauka, jei jie sutampa - įvestas teisingas slaptažodis. Svarbu paminėti, jog privaloma naudoti maišos algoritmus, kurie neturi žinomų pažeidžiamumų ar rastų kolizijų. Taip pat, niekada nesiūloma bandyti kurti savo maišos algoritmų, o vietoje to rinktis srities ekspertų sukurtus ir patikrintus sprendimus.

Renkantis maišos algoritmą, kuris bus naudojamas slaptažodžiams saugoti, privaloma atsižvelgti į jų atsparumą pagrindinėms slaptažodžių nulaužimo atakoms.

- **Grubios jėgos ataka.** Dauguma plačiai naudojamų maišos algoritmų yra greitai apskaičiuojami - tai reiškia, kad turint pakankamus skaičiavimo resursus, trumpame laiko tarpe įmanoma padaryti daug spėjimų. Jeigu slaptažodis trumpas, tam gali būti panaudota grubios jėgos ataka (ang. *brute-force attack*). Šios atakos metu yra spėjami visi įmanomi simbolių deriniai, pavyzdžiui pradedant nuo eilutės *aaaaaaaa* ir tuomet keičiant po vieną simbolį - *baaaaaaa, caaaaaaa, daaaaaaa* ir t.t.
- **Žodyno ataka.** Taip pat gali būti pasitelktos sudėtingesnės atakos, viena jų tai žodyno ataka (ang. *dictionary attack*), kuomet bandomi dažnai naudojami žodžių, skaičių ar raidžių deriniai. Tikslinėje atakoje prieš konkretų vartotoją generuojant potencialius slaptažodžius gali būti pasitelkta viešai prieinama informacija apie jį, kaip pavyzdžiui vartotojo gimimo data ar gyvenamasis miestas. Dėl žodyno ir grubios jėgos atakų svarbu, kad maišos algoritmas nebūtų spartus ir to pasėkoje kiekvienas spėjimas užtruktų pakankamai ilgą laiko tarpą, taip sumažinant atakos efektyvumą [41].
- **Vaivorykštinės lentelės ataka**. Įsilaužėliai gali pasitelkti ir jau žinomais slaptažodžiais iš kitų nutekintų duomenų bazių. Saityno saugumo konsultanto Troy Hunt valdomoje svetainėje haveibeenpwned.com siūlomą slaptažodžių sąrašą sudaro per 847 milijonus slaptažodžių [42], o svetainės Cybernews slaptažodžių rinkinyje įrašų skaičius siekia daugiau nei 7.8 milijardo [43]. Dažnai naudojamiems slaptažodžiams gali būti iš anksto pritaikyti dažnai naudojami maišos algoritmai taip panaikinant algoritmo veikimo laiko elementą ir suprastinant ataką iki paprasčiausio palyginimo su jau turimu maišos santraukų sąrašu. Šis metodas vadinamas vaivorykštinės lentelės ataka (ang. *rainbow-table attack*) [41]. Efektyviausia priemonė užkirsti kelią vaivorykštinės lentelės atakoms yra prieš pritaikant maišos algoritmą prie slaptažodžio pridėti atsitiktinai sugeneruotą simbolių eilutę, vadinamą druska (ang. *salt*), kuri būtų unikali kiekvienam saityno programos vartotojui. Druskos reikšmės naudojimas reiškia, jog to pačio slaptažodžio maišos santrauka skirsis net jeigu du vartotojai naudos tą patį slaptažodį [41, 44].

Saityno programoje slaptažodžiams taikomas maišos algoritmas turi būti apsaugotas nuo kiekvienos iš minėtųjų atakos tipų.

3.3. Injekcijos pažeidžiamumų prevencija

3.3.1. Sprendimas XSS spragų prevencijai

Norint užkirsti kelią XSS, privalu užtikrinti, jog jokia vartotojo įvestis nebus atvaizduota kaip teisėta programos kodo dalis. Tai galima padaryti atliekant dvi pagrindines užduotis - filtruojant vartotojo įvestį ir, prieš atvaizduojant vartotojo įvestus duomenis, pasirūpinti jų kodavimu.

Iš pirmos pažiūros, gali pasirodyti, jog XSS prevencijai atlikti užtenka dirbti tik su kliento pusės kodu, kadangi čia yra kontroliuojama tiek vartotojo įvestis, tiek išvestis, tačiau tai dažnas nesusipratimas. Kalbant apie vartotojo įvesties filtravimą, daugelis problemų atsiranda iš klaidingos nuomonės, jog vartotojas su saityno programos serveriu gali komunikuoti tik naudodamasis klientinės programos dalies vartotojo sąsaja. Tokiu atveju nepatyrę programuotojai užtikrina, jog vartotojo įvesti duomenys atitinka tai, ko tikimąsi, naudodamiesi apsaugas vartotojo sąsajoje, tačiau nevykdo jokios vartotojo įvesties patikros duomenims pasiekus programos serverio kodą.

Vartotojo įvesties patikrinimo nebuvimas saityno programos serverio pusėje matomas pavyzdyje 2.4.1.1.2. Patikrinimas, ar vartotojas įvedė tinkamą nuorodą buvo atliktas tik vartotojo sąsajoje. Problema, jog užpuolikas gali lengvai modifikuoti užklausas naudojantis tokiais įrankiais kaip BurpSuite ir apeiti bet kokį filtravimą atliekamą vartotojo sąsajos lygyje. Taip pat, svarbu nepamiršti, jog vartotojo įvestis, tai nėra vien užklausos parametrai. Įvestimi gali būti ir HTTP užklausų antraštės ar kita informacija, kurią gali modifikuoti vartotojas, kaip iliustruojama pavyzdyje aptartame poskyryje 2.4.1.1.3. Taigi, pirmasis žingsnis norint apsisaugoti nuo XSS pažeidžiamumų, tai suprasti, jog niekada negalima pasitikėti vartotojo įvestimi, ji turi būti patikrinama ir filtruojama tiek programos kliento, tiek serverio lygiu.

Saityno programose dažnai pageidaujamas funkcionalumas išsaugoti ir atvaizduoti vartotojo įvestas simbolių eilutes, pavyzdžiui komentarus, kuriuose gali būti specialių simbolių naudojamų HTML, JavaScript ar CSS kode. Norint saugiai atvaizduoti specialiuosius simbolius naudojamas išvesties kodavimas. Kodavimas yra panaši priemonė į filtravimą, kurią galima taikyti tiek kliento tiek serverio pusės kode, pakeičiant tam tikrus simbolius į nepavojingus jų ekvivalentus [19]. Pavyzdžiuose poskyriuose 2.4.1.1.1 ir 2.4.1.1.3 parodyta, kaip kabučių simbolis leido vartotojo įvesčiai užeiti už simbolių eilutės kintamojo ribų, o simbolis > panaudotas, norint išeiti iš HTML elemento, kuriame atvaizduojama vartotojo įvestis, konteksto. Tai atlikus, vartotojo įvestis buvo interpretuojama kaip HTML kodo dalis. Taikant išvesties kodavimą, šie simboliai būtų pakeisti į simbolių eilutes, kurios vartotojo sąsajoje atrodytų taip pat, kaip pradiniai įvesti simboliai, tačiau negalėtų būti interpretuojami kaip HTML kodo dalis. Analogiškai HTML kodavimui, taip pat taikomas HTML atributų, JavaScript, CSS, bei kintamųjų esančių nuorodose kodavimas - simboliai galintys būti interpretuoti kaip kodo dalis kiekviename kontekste, pakeičiami nepavojingu ekvivalentu [19].

Papildoma apsauga, kuri tampa vis dažniau rekomenduojama, norint apsisaugoti nuo XSS pažeidžiamumų, tai turinio saugumo politika (ang. *Content Security Policy – CSP*). CSP naudojimas apibrėžiamas Content-Security-Policy antraštėje arba HTML <*meta*> elemente [45].

CSP apsaugo nuo kodo iš nepatikimų šaltinių įterpimo, kadangi apibrėžia visus šaltinius, iš kurių naršyklė gali vykdyti kodą, ar įterpti kitą informaciją. CSP *default-src* ir *script-src* direktyvos gali užkirsti kelią JavaScript kodo veikimui, kuris įterptas į HTML dokumentą, taip pat blokuoti nesaugios *eval()* funkcijos naudojimą [46]. Pritaikius CSP rekomenduotina įjungti pažeidimų fiksavimą (ang. *reporting*). Taip programos kūrėjai gali gauti informaciją apie politikos pažeidimus ir kodą ištaisyti taip, kad jis atitiktų norimą taikyti CSP.

Dažniausiai XSS pažeidžiamumai panaudojami jautriai informacijai iš slapukų pavogti. Todėl yra pageidaujama slapukus apsaugoti nuo galimybės juos pasiekti JavaScript kode. Slapukai gali būti apsaugoti nurodant *HttpOnly* atributą, užtikrinant, jog jie galėtų būti perduodami tik su HTTP užklausomis.

Taigi, yra keletas skirtingų priemonių, kurios siūlomos norint apsisaugoti nuo XSS pažeidžiamumų. Idealiausiu atveju, siūloma taikyti visas priemones, norint užtikrinti maksimalią apsaugą.

3.3.2. SQL injekcijos pažeidžiamumų prevencija

Siekiant užkirsti kelią SQL injekcijos pažeidžiamumams, reikia užtikrinti, kad vartotojo įvestis SQL užklausoje bus interpretuojama tik kaip tekstas ir joje esantys SQL raktažodžiai nebus vykdomi. Šiam tikslui taikomos priemonės panašios į XSS pažeidžiamumų prevencijos metodus – vartotojo įvestis filtruojama, ji nėra tiesiogiai dedama į SQL užklausos vidų.

Vartotojo įvesties filtravimas aptartas darbo poskyryje 3.3.1. Kalbant apie SQL injekciją, privalu prisiminti jau minėtuosius principus:

- Negalima pasitikėti vartotojo įvestimi, jai privaloma taikyti filtravimą, įsitikinant, jog įvesta informacija atitinka, tai, ko iš vartotojo tikimąsi.
- Vartotojas gali modifikuoti duomenis ne tik naudojantis vartotojo sąsaja, todėl filtravimą privalu taikyti informacijai atkeliavus į saityno programos serverį.
- Vartotojo įvestis nepasiriboja tik užklausų parametrais, svarbu atsižvelgti ir į kitus dalykus, kaip kad HTTP antraštes.

Vien filtravimas nėra pakankama apsauga nuo SQL injekcijos pažeidžiamumų, kad jų išvengti taip pat siūloma naudoti paruoštąsias formuluotes (*ang. prepared statement*) [34]. Paprastai SQL užklausa galėtų būti suformuojama kaip simbolių eilutė, sudedant iš anksto numatytą užklausos dalį su vartotojo įvestimi, pavyzdžiui:

```
užklausa = "SELECT * FROM vartotojai WHERE vardas =" + vartotojo_įvestis;
```

Tuomet ši sudaryta simbolių eilutė vykdoma kaip SQL užklausa. Tai reiškia, jog vartotojo įvestis yra lygiateisė iš anksto numatytai užklausos daliai ir vartotojo įvesti SQL raktažodžiai bus interpretuojami kaip teisėta užklausos dalis. Paruoštosios formuluotės naudojimo atveju sukuriamas SQL užklausos šablonas, kuriame vartotojo įvestis nenurodoma. Vietose, kuriose tikimąsi vartotojo įvesties nurodomi parametrai, dažniausiai žymimi simboliu „?“:

```
užklausa = paruoštosiosFormuolėsFunkcija("SELECT * FROM vartotojai WHERE vardas = ?");
```

Tuomet SQL užklausos šablonas sukompiliuojamas ir išsaugomas. Prieš vykdant SQL užklausą, programa užklausos šablono parametrus susieja su vartotojo įvesties reikšmėmis. Užklausos vykdymo metu, užklausos šablonas yra iš anksto numatytas, todėl vartotojo įvestyje esantys SQL raktažodžiai nėra laikomi SQL užklausos struktūros dalimi, jie interpretuojami tik kaip tekstas.

SQL injekcijos spragų prevencijai siūlomos dvi efektyviausios priemonės – įvesties filtravimas ir paruoštųjų formuluočių naudojimas. Tinkamas jų taikymas žada užkirsti kelią SQL injekcijos pažeidžiamumų atsiradimui.

3.4. Pažeidžiamumų prevencijos metodų apibendrinimas

Apžvelgti pažeidžiamumų prevencijos metodai sprendžiantys spragų atsiradimo priežastis. Pagrindiniai siūlomi metodai:

- Tiesioginių objektų nuorodų pakeitimas identifikatoriais, kurie atitinka anonimiškumo bei neatsekamumo reikalavimus ir yra sunkiai atspėjami.
- Kiekvienam resursui, kuriam reikalinga autorizacija, privalomas patikrinimas, ar vartotojas gali jį pasiekti. Vartotojo autorizacija turi būti nustatoma pagal jo prieigos raktą.
- Visai vartotojo komunikacijai su serveriu vietoje HTTP, turi būti naudojamas HTTPS protokolas. Protokolo naudojimas užtikrinamas HSTS antraštės pagalba.
- Vartotojų slaptažodžiams turi būti taikomi maišos algoritmai, kurie yra atsparūs grubios jėgos, žodyno ir vaivorykštinės lentelės atakoms.
- Visai vartotojo įvesčiai, neapsiribojant užklausų parametrais, turi būti taikomas filtravimas saityno programos serverio pusės kode.
- Prieš atvaizduojant vartotojo įvestį, jai turi būti pritaikytas kodavimas.
- CSP gali būti naudojama kaip papildoma apsauga nuo XSS pažeidžiamumų.
- SQL užklausoms sudaryti turi būti naudojamos paruoštosios formuluotės.

Siūlomi metodai yra bendriniai, nepatyręs programuotojas sistemos kūrimo metu gali juos netinkamai pritaikyti, todėl metodus pageidaujama sukonkretinti.

4. PAŽEIDŽIAMUMŲ PREVENCIJOS METODŲ TYRIMAS

Skyriuje siekiama sukonkretinti siūlomus pažeidžiamumų prevencijos metodus, įsitikinti jų efektyvumu. Vertinant metodus atsižvelgiama į jų panaudojimą užkertant kelią pažeidžiamumams analizės metu apžvelgtuose pavyzdžiuose.

4.1. Prieigos kontrolės pažeidžiamumų prevencijos metodų tyrimas

4.1.1. Objektams siūlomų naudoti identifikatorių vertinimas

Šiame poskyryje palyginami du populiariausi universaliai unikalūs identifikatoriai (ang. *universally unique identifier - UUID*) – pirmosios ir ketvirtosios versijų UUID.

UUID yra 128 bitų ilgio simbolių eilutė, kuri arba garantuojama, kad bus unikali, arba jos unikalumas yra labai tikėtinas [47].

Pirmosios versijos identifikatorius generuojamas fiksuojant laiko žymą generavimo metu. Laiko žyma yra 60 bitų ilgio, kurios detalumas 100 nanosekundžių, taip garantuojamas generuojamų identifikatorių unikalumas 1600 metų į priekį. Identifikatorių taip pat sudaro 48 bitų ilgio unikalus tinklo įrenginio identifikatorius (ang. *media access control address – MAC address*) [47]. Tinklo įrenginio identifikatorių galima laikyti kaip konstantą saityno programos kontekste, darant prielaidą, jog UUID sudaromas centralizuotai serveryje, kuriame programa talpinama. Priešingu atveju, jeigu UUID būtų sudaromi pačio vartotojo, nebūtų galima pasitikėti jo unikalumu, vartotojas kurdamas objektą galėtų pateikti jau egzistuojantį UUID, tikėdamasis taip gauti prieigą prie kitam vartotojui priklausančio objekto. Taip pat būtų pažeisti anonimiškumo ir neatsekamumo principai, nes identifikatorius atskleistų jį sukūrusio vartotojo tinklo įrenginio adresą.

Į pirmosios versijos UUID įeina versijos (4 bitai) ir varianto (2 arba 3 bitai) reikšmės, dažniausiai viso 6-7 bitai. Taip pat naudojama 14 bitų ilgio laikrodžio eilės reikšmė, kuri atsitiktinai sugeneruojama programos paleidimo pradžioje. Vėliau laikrodžio eilė gali būti didinama arba naujai sugeneruojama programos veikimo metu esant laikrodžio sinchronizacijos poreikiui [47]. Naudojantis internetiniu UUID generavimo įrankiu buvo sugeneruoti du pirmosios versijos identifikatoriai, apytiksliai 10 sekundžių laiko tarpe:

a5e3b416-b34b-11ec-b909-0242ac120002

ace286ca-b34b-11ec-b909-0242ac120002

Kadangi identifikatoriai sudaryti to pačio įrenginio, simboliai, nurodantys MAC adresą, laikrodžio eilę, versijos ir varianto reikšmes, sutampa. Tuo tarpu UUID sudaranti laiko žyma aktuali tik tam tikruose rėžiuose, pavyzdžiui, pirmieji 12 simbolių (nuo 00000000-0000 iki ffffffff-ffff) nurodo laiko tarpą lygų apytikriai 326 dienoms. Kartais objektų sukūrimo laikas yra viešas, pavyzdžiui komentarų laiko žyma ar paskyrų profiliuose skelbiama informacija apie vartotojo paskyros sukūrimą. Žinant objekto sukūrimo laiką sekundžių tikslumu, jo UUID garantuojamas atspėti iš 10 milijonų bandymų. Taigi, pirmosios versijos UUID identifikatoriai yra unikalūs, taip pat jei generuojami centralizuotai jie anonimiški ir neatsekami kūrėjo atžvilgiu, tačiau atskleidžia objekto sukūrimo laiką. Atspėti identifikatorių yra ganėtinai sudėtinga, tačiau tai vis tiek yra įmanoma žinant apytikslį laiką, kuriuo sukurtas objektas. Beveik pusė

identifikatoriaus saityno programos kontekste dažniausiai nebus aktuali ir išliks tokia pat visiems programoje naudojamiems identifikatoriams, taigi jo panaudojimas galėtų būti efektyvesnis.

Ketvirtosios versijos identifikatorius generavimui pasirenkama visiškai kitokia strategija – pseudoatsitiktinių arba tikrai atsitiktinių skaičių generavimas. Kaip ir kituose UUID, 4 bitai nurodo naudojamą versiją ir tuomet 2 arba 3 bitai naudojami atvaizduoti variantui. Tačiau visi likę identifikatorių sudarantys bitai yra atsitiktiniai [47]. Ketvirtosios versijos UUID identifikatorių aukšta entropija matoma palyginant du sugeneruotus identifikatorius:

9f9862de-6f22-4563-9dc5-4536f68bb38c

df4ffdd7-f6a2-465e-a6e8-49f35ad1beec

Akivaizdu, jog didžioji dalis simbolių tarp sugeneruotų identifikatorių skiriasi.

Kadangi didžioji dalis ketvirtosios versijos identifikatoriaus sudaroma atsitiktinai, prarandamas unikalumo garantas. Tačiau tai veikiau teorinis nuostolis, praktikoje pirmasis ketvirtosios versijos UUID variantas turės 122 atsitiktinius bitus, taigi viso su juo galima sudaryti 2^{122} skirtingus identifikatorius - tai milžiniškas skaičius.

Siekiant iliustruoti kokia maža kolizijos tikimybė, galima pritaikyti gimtadienio dienų paradoksą (*ang. birthday paradox*). Paradoksas sprendžiamas atsakant į klausimą – kiek žmonių reikia, kad bent dviejų jų gimtadienių sutapimo tikimybė būtų 50%. Į šį klausimą apytiksliai galima atsakyti naudojantis lygybe, kur *n* tai žmonių skaičius [48]:

$$n \approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot \ln(2) \cdot 365}$$

Gimtadienio dienų paradokso sprendimas gali būti pritaikomas ir UUID kolizijoms – kiek identifikatorių reikia sugeneruoti, kad bent dviejų jų sutapimo tikimybė būtų 50%. Formulėje dienų skaičius pakeičiamas visų unikalių ketvirtosios versijos UUID skaičiumi - 2^{122}.

$$n \approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot \ln(2) \cdot 2^{122}} \approx 2.715 \cdot 10^{18}$$

Jeigu kiekvienam žmogui Žemėje (apvalinama iki 8mlrd.) sugeneruotume po vieną UUID kiekvieną minutę, 50% kolizijos tikimybė būtų pasiekta po maždaug 645 metų (nepaisant keliamųjų metų):

$$\frac{2.715 \cdot 10^{18}}{8 \cdot 10^{9} \cdot 60 \cdot 24 \cdot 365} \approx 645.69$$

Šis milžiniškas galimų identifikatorių skaičius taip pat reiškia, jog, esant teisingam ketvirtosios versijos UUID įgyvendinimui, atspėti objekto identifikatorių tampa praktiškai neįmanoma. Būtina saugaus įgyvendinimo dalis – kriptografiškai saugaus atsitiktinių skaičių generatoriaus naudojimas, tai reiškia, kad trečioji šalis negali nuspėti, koks skaičius bus sugeneruotas. Rekomenduojama rinktis plačiai naudojamas ir patikrintas atsitiktinių skaičių generavimo bibliotekas.

Palyginus dvi UUID versijas galima teigti, jog saityno programų kontekste pageidautinas ketvirtosios versijos UUID naudojimas, dėl atitikimo visiems keturiems iš anksto numatytiems reikalavimams – unikalumo, aukštos entropijos, bei anonimiškumo ir neatsekamumo.

Nagrinėtuose IDOR pažeidžiamumų pavyzdžiuose ketvirtosios versijos UUID naudojimas objektams identifikuoti reikštų, kad prieigos kontrolės pažeidžiamumo rizika būtų stipriai sumažinta – net esant prieigos kontrolės pažeidžiamumui įsilaužėlis negalėtų gauti prieigos prie bet kokio objekto, nes papildomai reikėtų sužinoti to objekto identifikatorių.

4.2. Kriptografinių pažeidžiamumų prevencijos metodų tyrimas

4.2.1. HTTPS protokolo naudojimo užtikrinimo metodų vertinimas

Norint praktiškai išbandyti, kaip atrodo komunikacija taip vartotojo ir saityno programos serverio, imituojama „žmogaus per vidurį“ ataka. Šiam tikslui naudojamas įrankis Wireshark, galintis stebėti užklausas, kurios siunčiamos iš naršyklės į saityno programos serverį [49].

Pavyzdžiams naudojama prisijungimo formos užklausa svetainėje http://hack-yourself-first.com/ [50]. Svetainė sukurta saugumo saityno saugumo konsultanto Troy Hunt ir yra skirta padėti programuotojams gerinti kibernetinio saugumo gynybinius įgūdžius. Svetainė palaiko tiek nesaugų HTTP protokolą, tiek šifruotą HTTPS.

Pirmiausiai, naršyklėje išsiunčiama užklausa http://hack-yourself-first.com/Account/Login naudojantis HTTP protokolu. Tuomet užklausa ir atsakymas iš saityno programos peržiūrimi Wireshark įrankyje (4.1 pav.).

```
GET /Account/Login HTTP/1.1
Host: hack-yourself-first.com
Connection: keep-alive
Cache-Control: max-age=0
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/101.0.4951.67 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en,lt;q=0.9,en-US;q=0.8,ru;q=0.7,pl;q=0.6
Cookie: ASP.NET_SessionId=1x4dsghbrog2jvbr53kemcvk; VisitStart=5/19/2022 10:35:05
AM

HTTP/1.1 200 OK
Date: Thu, 19 May 2022 10:50:03 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Cache-Control: private
Vary: Accept-Encoding
X-XSS-Protection: 0
X-AspNetMvc-Version: 5.1
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
CF-Cache-Status: DYNAMIC
```

4.1 pav. HTTP užklausa ir atsakymas atvaizduoti įrankyje Wireshark

Paveikslėlyje matoma užklausos informacija paprastuoju tekstu. Užklausoje matomi net vartotojo siunčiami slapukai. Tokią pat informaciją galėtų matyti užpuolikas, gebantis perimti vartotojo tinklo srautą.

Tuomet į programą išsiunčiama ta pati užklausa, šįkart naudojantis HTTPS protokolu - https://hack-yourself-first.com/Account/Login. Užklausa atvaizduojama įrankyje Wireshark (4.2 pav.).

4.2 pav. HTTPS užklausa ir atsakymas atvaizduoti įrankyje Wireshark

Šiuo atveju perduodami duomenys yra šifruoti. Užpuolikas galėtų matyti tik svetainės domeną, tačiau negalėtų patikrinti nei pilnos nuorodos, kuri buvo siunčiama, nei kokie duomenys ar slapukai buvo išsiųsti kartu su užklausa. HTTPS efektyvumas duomenų apsaugai akivaizdus.

Tačiau išlieka problema, jog vartotojas vis tiek turi galimybę naudoti nesaugią HTTP puslapio versiją. Šiai problemai spręsti išbandomas jau aptartas HSTS antraštės naudojimas. Norint išbandyti HSTS funkcionalumą, domenas hack-yourself-first.com pridedamas į naršyklės HSTS sąrašą rankiniu būdu. Chrome naršyklėje HSTS sąrašą galima redaguoti į naršyklės paieškos lauką įvedus chrome://net-internals/#hsts. Domeną pridėjus į HSTS sąrašą, naršyklėje vėl bandoma atidaryti nesaugi HTTP nuoroda http://hack-yourself-first.com/Account/Login. Įrankyje Wireshark pastebima, jog nešifruota užklausa nėra perduodama. O naršyklės adreso lauke svetainės nuoroda automatiškai pakeista iš „http“ į „https“ (4.3 pav.).

4.3 pav. Naršyklės adreso lauke pakeista svetainės nuoroda

Svetainės buvimas naršyklės HSTS sąraše užkirto kelią nesaugios HTTP užklausos išsiuntimui. Todėl atakos, kuriose bandoma vartotoją apgaule priversti išsiųsti nesaugią HTTP užklausą, su tikslu perimti užklausoje siunčiamus duomenis, nėra galimos.

4.2.2. Slaptažodžiams saugoti skirtų maišos algoritmų efektyvumo įvertinimas

Pasirinkta palyginti keletą maišos algoritmų, įvertinant jų atsparumą poskyryje 3.2.2 minėtosioms slaptažodžių nulaužimo atakoms. Nagrinėjami algoritmai – paprastasis SHA256, PBKDF2 ir Scrypt. Pastaba – pasirinkimo procese iš karto nuspręsta nenagrinėti maišos algoritmų su žinomais pažeidžiamumais, nes jų naudojimas neturi būti svarstomas.

4.2.2.1. Algoritmų greitis

Algoritmų greičiui išmatuoti pasirinkta sukurti paprastą programą, kuri vartotojo pateiktai simbolių eilutei pritaiko kiekvieną iš nagrinėjamų maišos algoritmų ir išmatuoja kiekvieno jų veikimo laiką. Programa sukurta naudojantis Python programavimo kalba, maišos algoritmų funkcijos įgyvendintos naudojantis hashlib biblioteka. Programos kodas patalpintas 1 priede.

Bandymų metu, kiekvienas algoritmas 10000 kartų pritaikomas tai pačiai simbolių eilutei, išbandytos trijų skirtingų ilgių atsitiktinai sugeneruotos simbolių eilutės – „7op27KAj“, „IQ%U9jdHX3l724” ir „bUtO4ffdgDO66Ih^t1TL“.

PBKDF2 algoritmui parinkti parametrai pagal minimalias IETF rekomendacijas pateiktas dokumente RFC2898, tai yra 1024 maišos algoritmo taikymo iteracijų, atsitiktinė 64 bitų dydžio druskos reikšmė ir naudotas SHA256 maišos algoritmas.

Scrypt algoritmo parametrai taip pat parinkti pagal IETF rekomendacijas, jos pateikiamos dokumente RFC7914. Iteracijų skaičius lygus 1024, lygiagretinimo (ang. *parallelization*) parametro reikšmė – 1, o bloko dydžio reikšmė nustatyta kaip 8. Naudota ta pati druskos reikšmė kaip ir PBKDF2 algoritmui.

Apačioje pateikiami tyrimo rezultatai (4.1 lentelė).

4.1 lentelė. Maišos algoritmų spartos bandymo rezultatai

	Naudotas algoritmas		
Simbolių eilutė	**Paprastasis SHA256**	**PBKDF2**	**Scrypt**
7op27KAj	0.009s	6.804s	29.15s
IQ%U9jdHX3l724	0.01s	6.888s	28.939s
bUtO4ffdgDO66Ih^t1TL	0.01s	6.975s	28.967s

Svarbu paminėti, jog rezultatus interpretuoti galima tik lyginant algoritmus tarpusavyje, skirtingi algoritmų įgyvendinimai skirtingomis programavimo kalboms turės gerokai kitokius algoritmų veikimo laikus.

Tyrimo rezultatai parodo, koks didelis algoritmų veikimo spartos skirtumas yra tarp paprastojo SHA256 maišos algoritmo ir PBKDF2 bei Scrypt, kurie yra tikslingai sukurti, kad būtų lėtesnio veikimo bei užkirstų kelią slaptažodžių nulaužimui juos spėliojant. Vienas spėjimas PBKDF2 slaptažodžiui atspėti vidutiniškai užtruktų 750 kartų ilgiau nei vienas spėjimas paprastajam SHA256. Scrypt, lyginant su PBKDF2, esant tam pačiam 1024 iteracijų skaičiui, užtrunka dar virš 4 kartų ilgiau. Tačiau Scrypt ir PBKDF2 greičio skirtumas nėra svarbus, kadangi kitas šių algoritmų privalumas – tai konfigūracijos galimybės, vienos simbolių eilutės maišos rezultato skaičiavimo laikas gali būti padidintas ar pamažintas keičiant iteracijų skaičių. Algoritmo veikimo sparta gali būti pritaikyta atsižvengiant į sistemos resursų kiekį. Taip pat, konfigūracijos galimybės reiškia, jog laikui bėgant ir kartu augant kompiuterių skaičiavimo galiai, algoritmai išliks aktualūs.

Scrypt turi vieną esminį privalumą, kurio svarbą lengva nuvertinti - tai yra kad Scrypt sudėtingumas susidaro ir iš reikalaujamų atminties resursų, kas apsunkina Scrypt naudojimą įrenginiuose, kurie optimizuoti maišos algoritmų skaičiavimams [51]. Kriptografinių valiutų populiarumas sukūrė paklausą įrenginiams, galintiems kuo sparčiau ir pigiau skaičiuoti maišos algoritmų rezultatus, norint turėti pranašumą kriptografinių valiutų kasimo procese. Dėl šios ir kitų priežasčių technologija kurti įrenginius, naudojančius konkrečių programų integruotas schemas (ang. *Application Specific Integrated Circuit – ASIC*), skirtus maišos algoritmų vykdymui, sparčiai tobulėjo ir dabar ASIC įrenginiai maišos algoritmus geba skaičiuoti šimtus tūkstančių kartų greičiau nei tradiciniai procesoriai. Kadangi slaptažodžiai taip pat naudoja maišos

algoritmus, tokie įrenginiai gali būti panaudoti ir slaptažodžių laužymui, sukeldami nuogąstavimus dėl slaptažodžiams taikomų maišos algoritmų saugumo.

ASIC įrenginiai specializuodamiesi vienai paskirčiai, optimizuoja reikalingą procesoriaus plotą tam pačiam skaičiavimui atlikti lyginant su universalios paskirties procesoriais, taip sumažindami įrenginio kainą ir energijos sąnaudas [52]. Šiam pranašumui pašalinti pasiūlyta naudoti maišos algoritmus, kurie reikalautų tam tikro kiekio operatyviosios atminties resursų veikimo metu. Kadangi nėra būdo operatyviosios atminties resursų optimizuoti vienai paskirčiai, tai stipriai sumažina ASIC įrenginių pranašumą lyginant su tradicinėmis sistemomis. Dėl atsparumo ASIC įrenginių panaudojimui iššifruojant slaptažodžius, Scrypt turi svarų privalumą prieš PBKDF2 algoritmą.

Pagal naujausius duomenis, atsižvelgiant į dabartinių įrenginių skaičiavimo galimybes, siūlomas Scrypt minimalus iteracijų skaičius 65536 [19]. Laikui bėgant technologijų tobulėjimas rekomenduojamų iteracijų skaičių didins, todėl kuriant sistemą rekomenduojama atsižvelgti į OWASP rekomendacijas tuo metu.

4.2.2.2. Atsparumas vaivorykštinės lentelės atakai

Kaip aptarta skyriuje 3.2.2, itin svarbu taikant maišos algoritmus naudoti druskos reikšmes, norint apsisaugoti nuo vaivorykštinės lentelės atakų. PBKDF2 ir Scrypt algoritmus įgyvendinančios bibliotekos sudaro galimybes naudoti druskos reikšmę ją pateikiant į algoritmo funkciją, tuo tarpu naudojant paprastąjį SHA256 maišos algoritmą sistemos kūrėjai privalėtų druskos reikšmę prie slaptažodžio pridėti įgyvendindami savo funkcionalumą. Taip pat PBKDF ir Scrypt bandytos bibliotekos druskos reikšmę laiko privalomu parametru, taigi jas naudodamas programuotojas nepamirš naudoti druskos reikšmės. Druskos reikšmė turi būti sugeneruota naudojantis kriptografiškai saugiu atsitiktinių skaičių generatoriumi, jos ilgis - ne mažesnis nei 32 baitai. Kiekvienam slaptažodžiui turi būti sugeneruota nauja druskos reikšmė.

4.3. Injekcijos pažeidžiamumų prevencijos metodų tyrimas

4.3.1. Priemonių skirtų apsisaugoti nuo XSS pažeidžiamumų įvertinimas

Norint įvertinti aptartas XSS pažeidžiamumų prevencijos priemones kiekviena jų išbandyta praktikoje pavyzdiniuose Python ir HTML/JavaScript kodo fragmentuose.

4.3.1.1. Įvesties filtravimas

Įvesties filtravimas dažniausiai įgyvendinamas aprašant taisykles, apie tai, kaip turėtų atrodyti įvestis. Tuomet įvesties taisyklės taikomos tikrinant, ar vartotojo siunčiama informacija tas taisykles atitinka, kitu atveju informacija turi būti nepriimama arba modifikuojama. Tačiau dažna problema yra tai, jog kuriant taisykles yra pervertinamas sukurtų taisyklių efektyvumas ir neapsvarstomi visi būdai, kaip tas taisykles galima apeiti.

Sprendžiant kaip ištaisyti spragą aprašytą poskyryje 2.4.1.1.2, viena strategija užkirsti kelią JavaScript pseudoprotokolo naudojimui - paprasčiausiai iš įvesties pašalinti simbolių eilutę „javascript:“. Norint iliustruoti šios strategijos trūkumus, parašyta paprasta programa iš vartotojo įvesties pašalinanti simbolių eilutę „javascript:“. Pašalinimas įgyvendintas naudojant funkciją replace(), kuri pakeičia vieną simbolių eilutę kita, šiuo atveju „javascript:“ pakeičiama į tuščią reikšmę. Programai pateikus reikšmę „javascript:alert(document.domain)“, kuri buvo naudota

iliustruoti XSS pažeidžiamumą, gaunamas atsakymas: „alert(document.domain)“. Taigi, atrodytų, jog XSS pažeidžiamumas ištaisytas, kadangi nėra galimybės naudoti JavaScript pseudoprotokolo, tačiau tai būtų klaidinga prielaida. Užpuolikas tokį filtravimo funkcionalumą galėtų apeiti į simbolių eilutės „javascript:“ vidurį įterpdamas vėl tą pačią reikšmę „javascript:“, pavyzdžiui „java*javascript*:script:alert(document.domain)“. Išbandžius šią simbolių eilutę su filtravimo programa, gaunamas rezultatas – „javascript:alert(document.domain)“. Programos trūkumas – nepatikrinama ar po filtravimo pritaikymo gautame rezultate vėl neatsiras nepageidaujamų reikšmių.

Paprastas praktinis pavyzdys parodo, jog pavojingų reikšmių pašalinimas iš įvesties yra netinkamas sprendimas. Konkrečiu atveju, filtravimo būdą galima modifikuoti panaudojus ciklą, kurio metu filtravimas taikomas tol, kol iš įvesties pašalintos visos nepageidaujamos reikšmės. Taip pat, problemą galima spręsti nepriimant vartotojo įvesties su pavojingomis simbolių eilutėmis ir vartotojui grąžinant klaidos kodą. Tačiau tai yra netinkamas požiūris į XSS problemos sprendimą. Siekiant išfiltruoti pavojingas reikšmes privalu sudaryti visą jų sąrašą, atsiranda didelė tikimybė nenumatyti potencialiai pavojingų simbolių eilučių ir jų į sąrašą neįtraukti. Net numačius visas potencialiai pavojingas reikšmes, nėra garantuojama, jog kartu su programinės kalbos atnaujinimais, jų neatsiras daugiau. Taip pat yra rizika, jog motyvuotas įsilaužėlis sugebės atrasti būdą kaip filtravimą apeiti.

Saugesnis filtravimo įgyvendinimo metodas – apibrėžti leidžiamų reikšmių ar jų struktūros sąrašą (ang. *whitelist*). Remiantis pavyzdžiu poskyryje 2.4.1.1.2, vartotojas gali įvesti reikšmę, kuri atitinka nuorodos formatą ir joje gali būti naudojami tik tie simboliai, kurie naudojami nuorodose. Kadangi tikimasi nuorodos į puslapį internete, nuoroda turėtų prasidėti „http“ arba „https“, tuomet sekti simboliai „://“, po jų svetainės domenas su galimais subdomenais ir viršutinio lygio domenas („.com“, „.net“ ir t.t.). Galiausiai, nuorodoje gali būti nurodytas kelias. Norint patikrinti, ar vartotojo įvestis atitinka apibrėžtą formą ir leidžiamus simbolius, galimas reguliarios išraiškos taikymas (ang. *regular expression*). Įvesčiai neatitikus reguliarios išraiškos, ji turėtų būti atmesta, o vartotojui grąžintas klaidos kodas.

Remiantis IETF RFC3986 standartu, sudaryta pavyzdinė reguliarioji išraiška, kurią galima taikyti norint patikrinti, ar įvesta tinkama nuoroda - ^((http|https):\/\/)[a-zA-Z0-9-._~]+\.[a-z]+[?:a-zA-Z0-9-._~/]*. Siūloma išraiška neapibrėžia visų įmanomų nuorodų ir neleidžia nuorodos pabaigoje nurodyti parametrų, todėl, esant poreikiui, galėtų būti modifikuojama. Pavyzdinė reguliarioji išraiška išbandyta programiniame kode (2 priedas). Bandymo rezultatai užfiksuoti lentelėje (4.2 lentelė).

4.2 lentelė. Sukurtos reguliariosios išraiškos bandymo rezultatai

Nuoroda	**Formato atitikimas**
https://www.google.com	Taip
https://is.vu.lt/vuis/	Taip
https://www.vu.lt/paieska?gsquery=aaa&searchin=in-website	Ne
javascript:alert(document.domain)	Ne
javascript:alert(document.domain),https://www.google.com	Ne

Kuriant reguliarias išraiškas, rekomenduojama pradėti nuo labai griežto norimos įvesties formato apibrėžimo, vėliau jį pildant, jeigu randamos pageidaujamos reikšmes, kurios neatitinka apibrėžtų taisyklių. Taip pat svarbu atlikti bandymus, norint įsitikinti, jog parašytoji reguliarioji išraiška atitinka užsibrėžtas taisykles.

Užtikrinti tinkamai įvesčiai, taip pat galima naudoti bibliotekas, kurios atlieka reikšmių formato patikrinimo funkcijas. Išbandyta Python „validators“ biblioteka siūlo funkciją, tikrinančią, ar reikšmė yra tinkamas svetainės adresas. Programos kodas pateikiamas 2 priede. Bandymo rezultatai pateikiami lentelėje (4.3 lentelė).

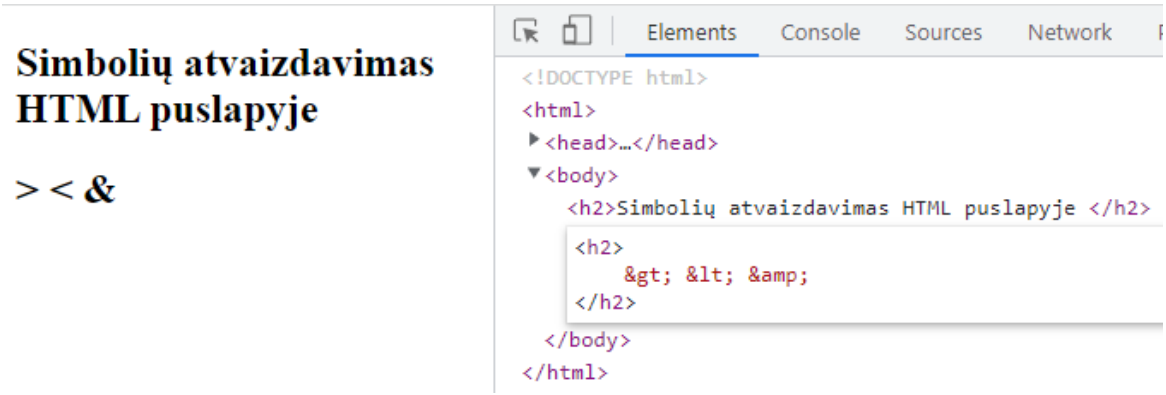
4.3 lentelė. Python „validators“ bibliotekos bandymo rezultatai

Nuoroda	**Formato atitikimas**
https://www.google.com	Taip
https://is.vu.lt/vuis/	Taip
https://www.vu.lt/paieska?gsquery=aaa&searchin=in-website	Taip
javascript:alert(document.domain)	Ne
javascript:alert(document.domain),https://www.google.com	Ne

Bibliotekos naudojimas yra greitesnė ir dažnai saugesnė filtravimo priemonė, kadangi jos plačiai naudojamos ir laiko patikrintos. Tačiau reikšmių patikrinimo bibliotekas galima naudoti tik tais atvejais, kuomet reikalingas patikrinimas pagal plačiai naudojamus formatus, kaip kad svetainių nuorodos, elektroninio pašto adresai ar UUID identifikatoriai. Jeigu reikalinga reikšmė atitinkanti specifinius programos reikalavimus, tuomet siūlomas reguliarios išraiškos taikymas.

4.3.1.2. Išvesties kodavimas

Išvesties kodavimo tikslas – vartotojo įvestį pakeisti taip, jog naršyklė ją galėtų interpretuoti tik kaip tekstą, bet ne kaip dalį programos kodo. Išvesties kodavimo pavyzdys, tai simbolio >, kuris yra specialusis HTML simbolis, pakeitimas į simbolių eilutę *>*. Simbolių eilutė *>* vartotojo sąsajoje bus atvaizduojama kaip simbolis >, tačiau nebus interpretuojama kaip HTML kodo dalis (4.4 pav.).



4.4 pav. HTML puslapis parodantis specialiųjų simbolių atvaizdavimą

Kodavimas turėtų būti pritaikomas visai vartotojo įvestai informacijai. Automatinio kodavimo funkcijos yra integruotos į daugumą saityno programoms kurti naudojamų karkasų

(ang. *framework*). Norint kuo labiau sumažinti XSS pažeidžiamumų atsiradimo riziką, siūloma išvesties kodavimą taikyti tiek kliento, tiek serverio pusės kode.

Dažniausiai klientinės pusės kodas bus parašytas JavaScript programavimo kalba, kuri turi keletą metodų automatiniam duomenų kodavimui (lentelė 4.4) [38].

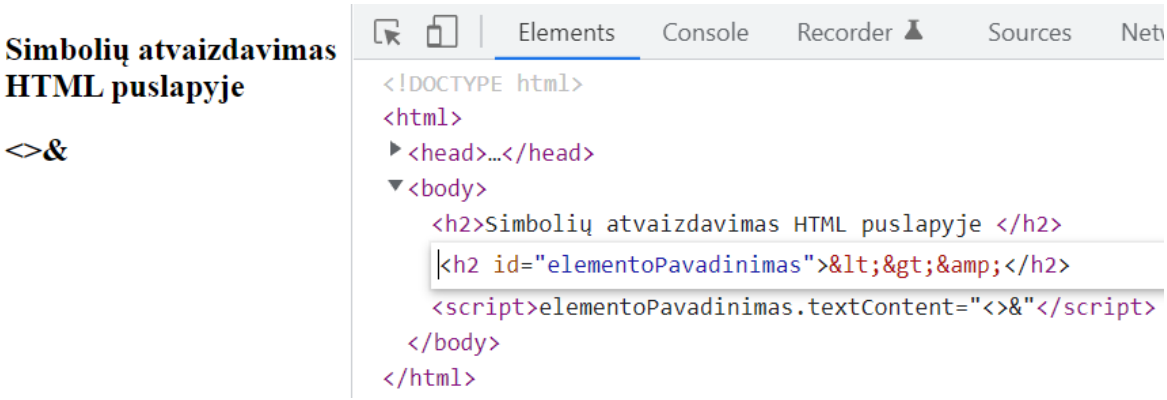
4.4 lentelė. JavaScript kodavimo funkcijos

Kontekstas	**Metodas**
HTML elementui	*elementas*.textContent = *vartotojoĮvestis*
HTML atributo reikšmei	*elementas*.setAttribute(*atributoPavadinimas*, *vartotojoĮvestis*)
Užklausos reikšmei	window.encodeURIComponent(*vartotojoĮvestis*)
CSS reikšmei	elementas.style.*savybė* = *vartotojoĮvestis*

Kodavimas turi būti pritaikytas atitinkamai kontekstui, kuriame bus atvaizduota vartotojo įvestis. Todėl kuriant saityno programą svarbu nustatyti visus kontekstus, kuriuose galėtų būti atvaizduojama konkreti vartotojo įvestis ir atitinkamai pritaikyti koduotę.

Analogiškai, kodavimas serverio pusės kode taip pat turėtų būti atliekamas atsižvelgiant į kontekstą, kuriame bus atvaizduota informacija. Dažnu atveju viena vartotojo įvestis bus atvaizduota keliuose skirtinguose kontekstuose, tuomet įvesčiai atitinkama koduotė turi būti pritaikyta keletą kartų.

JavaScript funkcijos, kuri saugiai nustato HTML elemento reikšmę, veikimo pavyzdys pateikiamas apačioje (4.5 pav.).



4.5 pav. HTML puslapis parodantis JavaScript kodavimo funkcionalumą

Tarp HTML *script* žymų pridedamas JavaScript kodas elementui saugiai priskiriantis reikšmę lygią simboliams <>&. Patikrinant nustatytą HTML elemento reikšmę, pastebima, jog vietoje simbolių matoma jų koduotė (*<>&*).

Vien kodavimas nėra sprendimas visoms XSS spragoms išvengti. Tai iliustruojama nagrinėjant XSS pažeidžiamumą atsirandantį dėl JavaScript pseudoprotokolo naudojimo *href* HTML atribute (4.6 pav.). Pavyzdyje elemento *href* atributui reikšmė prilyginama naudojantis, atrodytų, tinkama tokiam kontekstui funkcija. Tačiau paspaudus ant nuorodos elemento, JavaScript pseudoprotokole nurodyta funkcija suveikia ir vartotojui parodomas iššokantis langas.

4.6 pav. HTML puslapis parodantis *href* atributo kodavimo pritaikymą

Kodavimas šiuo atveju nepašalino XSS pažeidžiamumo, kadangi problema nėra susijusi su pavojingų simbolių naudojimu, o su JavaScript pseudoprotokolu. Užklausos kodavimas šiuo atveju taip pat nėra tinkamas sprendimo būdas, jis užkirs kelią XSS pažeidžiamumui, tačiau tuomet net ir tinkamos nuorodos bus tik atvaizduojamos, bet ant jų paspaudus vartotojas nebus nuvestas į norimą puslapį.

Dėl šios priežasties visai norimai atvaizduoti vartotojo įvesčiai svarbu naudoti tiek tinkamą įvesties filtravimą, tiek išvesties kodavimą, atsižvelgiant į kontekstą, kuriame ji bus atvaizduota.

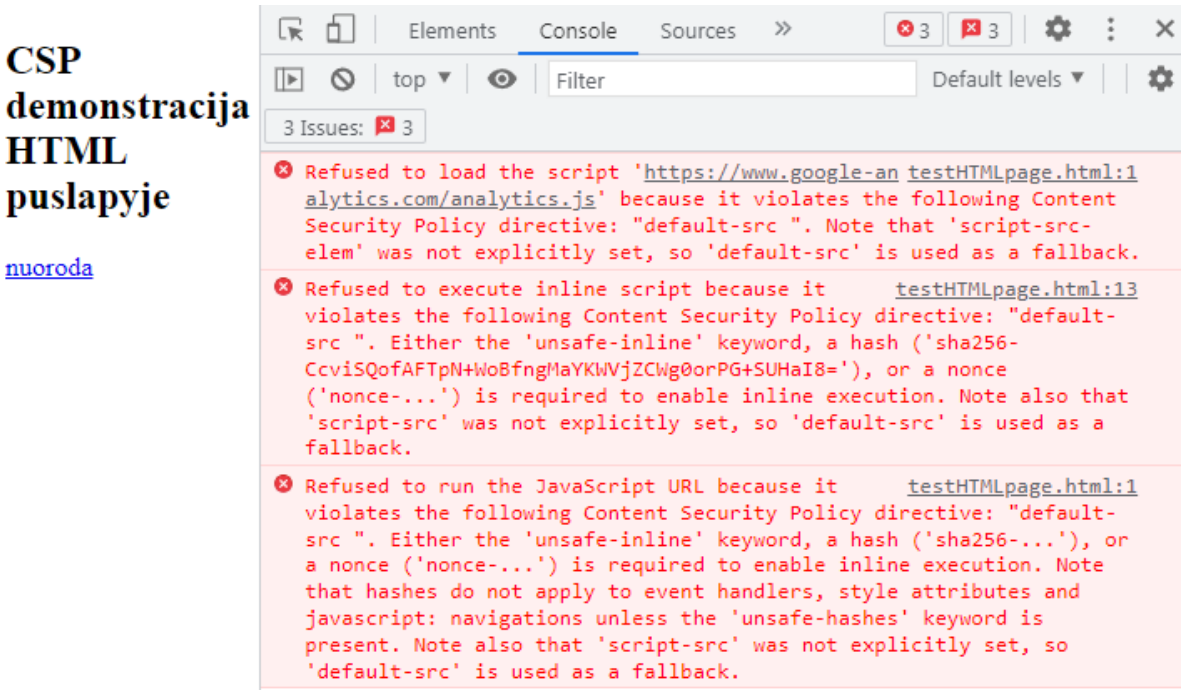
4.3.1.3. CSP taikymas

Taikant CSP, galima valdyti resursus, kurie leidžiami užkrauti puslapyje. Tai gali būti JavaScript kodo, vaizdo, CSS stiliaus ar kito tipo failai. Leidžiami resursai nurodomi CSP antraštėje. Pritaikius CSP direktyvą *default-src* bus užblokuotas viso JavaScript kodo, esančio HTML dokumente veikimas, taip pat bus neleista užkrauti nepatikimo kodo iš trečiųjų šalių. Pateikiamas pavyzdys, kuriame parodomas CSP veikimas praktikoje. HTML puslapio *meta* elemente nurodoma CSP politikos *default-src* direktyva (4.7 pav.). Taip pat puslapyje bandomas paleisti kodas:

- Užkraunantis JavaScript kodo failą analytics.js iš Google Analytics šaltinio.
- Parodantis iššokantį langą vartotojui tik atsidarius puslapį.
- Parodantis iššokantį langą vartotojui paspaudus ant elemento „nuoroda“.

4.7 pav. HTML puslapis parodantis CSP veikimą

Puslapis atidaromas naršyklėje, tuomet paspaudžiama ant elemento „nuoroda“ ir užfiksuojamos klaidos žinutės naršyklės konsolėje (4.8 pav.).



4.8 pav. Naršyklės konsolė su klaidos žinutėmis dėl CSP pažeidimų

Visi trys pavyzdiniame puslapyje esančio kodo fragmentai buvo užblokuoti dėl CSP pažeidimų. Norint naudoti JavaScript ar kitus resursus, jie turi būti apibrėžti CSP direktyvose, tuomet jų naudojimas nebus blokuojamas. Šiuo atveju, jeigu Google Analytics failo analytics.js naudojimas yra pageidaujamas, tai turėtų būti nurodoma CSP direktyva *script-src* - „script-src https://www.google-analytics.com/analytics.js“.

CSP konfigūracija turėtų būti atliekama atsižvelgus į saityno programos veikimo reikalavimus. Tinkama konfigūracija yra platus klausimas, neįeinantis į šio darbo apimtį. Taikant CSP siūloma atsižvelgti į OWASP organizacijos siūlomus resursus, pradedant nuo „Content Security Policy Cheat Sheet“ dokumento.

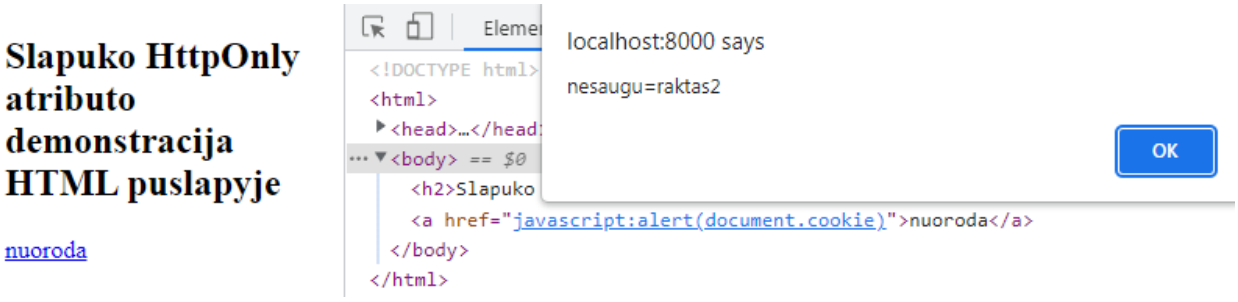
CSP apsauga nuo XSS pažeidžiamumų efektyvi tik tuo atveju, jeigu yra teisingai sukonfigūruojama ir programos vartotojai naudoja naršykles, palaikančias CSP. Taip pat, reikia žinoti, kad CSP savaime neištaiso XSS spragų, o tik užkerta kelią jų išnaudojimui. Dėl šios priežasties rekomenduojama pirmiausiai pritaikyti įvesties filtravimo bei išvesties kodavimo apsaugos priemones, siekiant neleisti įsilaužėliui įterpti savo kodo į puslapį, o tuomet įgyvendinti CSP naudojimą, kaip antrą saugumo liniją.

4.3.1.4. Slapukų apsaugojimas

Norint sumažint XSS pažeidžiamumo poveikį, tam atvejui, jeigu apsaugos nuo XSS nebuvo tinkamai pritaikytos, siūloma vartotojų prieigos raktų slapukams taikyti *HttpOnly* atributą. Šis slapukų atributas neleidžia slapukams būti pasiektiems puslapio JavaScript kodui. Taigi, įsilaužėliui įvykdžius XSS ataką, jo įterptas JavaScript kodas negalės pasiekti vartotojų slapukų naudojantis funkcija *document.cookie* (4.10 pav.).

Name	Value	D▲	P..	E..	S..	HttpOnly
nesaugu	raktas2	l...	/...	S...	14	
saugu	raktas1	l...	/...	S...	12	✓

4.9 pav. Slapukai su ir be *HttpOnly* atributo



4.10 pav. HTML puslapis parodantis slapukų *HttpOnly* atributo veikimą

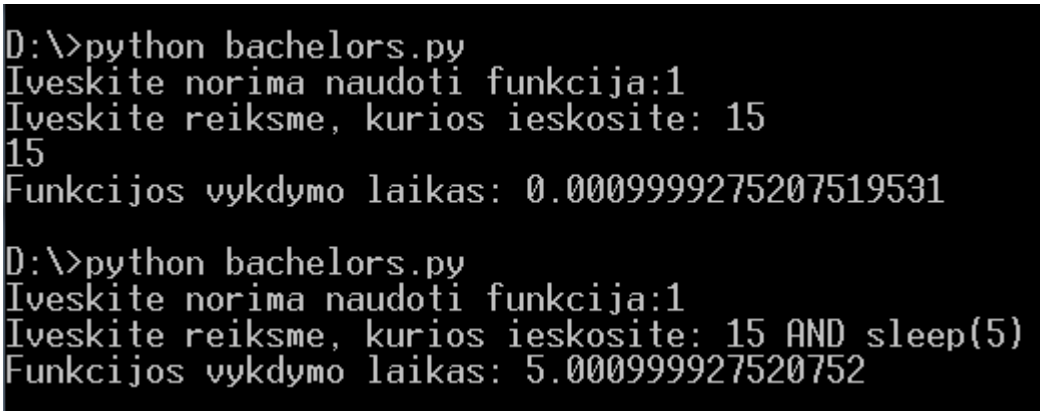
Pavyzdyje išsaugomi du slapukai, vienas pavadinimu „nesaugu“ be *HttpOnly* atributo ir kitas - „saugu“ su *HttpOnly* atributu (4.9 pav.). Vykdant pavyzdinę XSS ataką matoma, jog iššokančiame lange pateikiamas tik slapukas neturintis *HttpOnly* atributo.

Svarbu pabrėžti, jog *HttpOnly* atributo naudojimas nepašalina XSS pažeidžiamumų, tai veikiau priemonė sumažinanti XSS pažeidžiamumų panaudojimo galimybės. Užpuolikas negalės pavogti vartotojų prieigos raktų, tačiau vis tiek turės galimybę veikti vartotojų vardu.

4.3.2. SQL injekcijos prevencijos metodų įvertinimas

Norint įvertinti įvesties filtravimo ir paruoštųjų formuluočių naudojimą SQL injekcijos prevencijai, Python programavimo kalba parašyta programa su funkcijomis skirtomis siųsti užklausas į lokalią MySQL duomenų bazę. Programos kodas pateikiamas 3 priede. Visos funkcijos atlieka SQL užklausą SELECT, kurioje vartotojo įvestis įterpiama WHERE sąlygoje. Tam, kad patikrinti ar SQL injekcija galima, vartotojo įvestyje bandoma įterpti SQL funkcija *sleep* ir tuomet matuojamas užklausos įvykdymo laikas.

Pirmoji programos funkcija sudaro ir vykdo nesaugią SQL užklausą. Ji išbandoma su dvejomis skirtingomis vartotojo įvestimis – „15“ ir „15 AND sleep(5)“ (4.11 pav.).

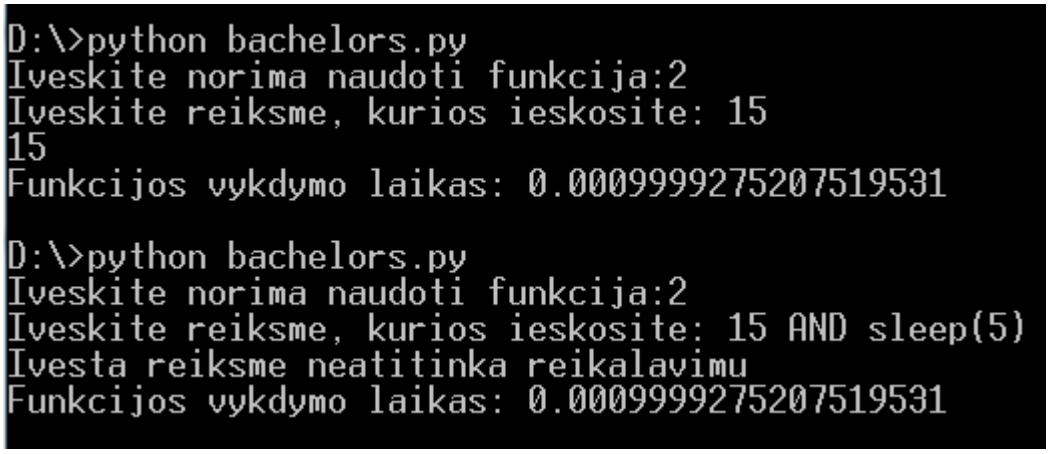


4.11 pav. Pažeidžiamos SQL užklausos sudarymo funkcijos veikimo rezultatai

Funkcijos vykdymo laikas su paprasta skaitine reikšme - apytiksliai 0.0001 sekundės. Naudojant įvestį su įterpta funkcija *sleep(5),* užklausos vykdymo laikas lygiai 5 sekundėmis ilgesnis. Akivaizdu, jog pavyzdinė funkcija turi SQL injekcijos pažeidžiamumą.

4.3.2.1. Įvesties filtravimas

Norint parodyti įvesties filtravimo galimybes apsisaugant nuo SQL injekcijos, pavyzdinėje programoje sukurta funkcija, sudaranti nesaugią SQL užklausą, tačiau šįkart vartotojo įvesčiai pirmiausiai pritaikomas filtravimas. Kadangi WHERE sąlygoje naudojama reikšmė „id“, kuri nurodo elemento skaitinį identifikatorių, filtravimo metu turėtų būti įsitikinama, kad vartotojas įvedė būtent skaitinę reikšmę. Programos veikimas išbandomas (4.12 pav.).



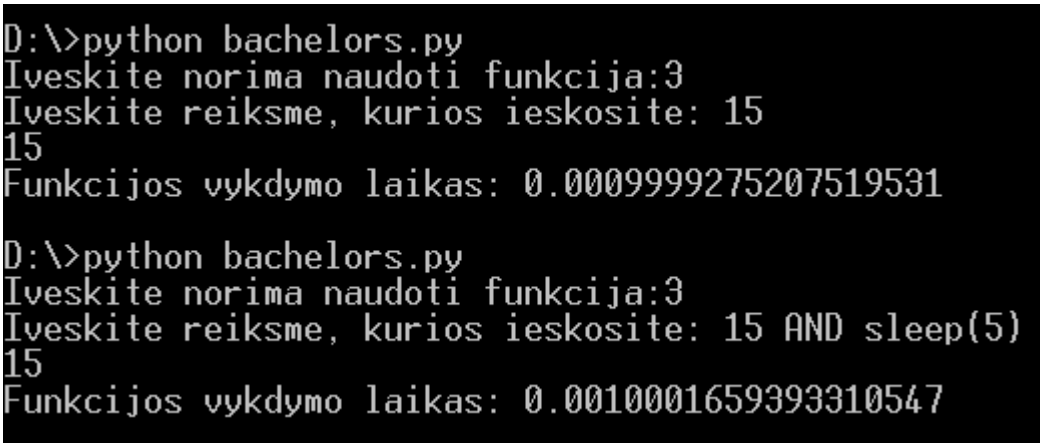
4.12 pav. SQL užklausos sudarymo funkcijos su pritaikytu įvesties filtravimu veikimo rezultatai

Kaip ir tikėtasi, vartotojo įvestis su įterpta SQL *sleep(5)* funkcija nepriimama. Filtravimas šiuo atveju labai efektyvi priemonė, bet taip yra todėl, jog vartotojo įvestis yra griežtai apibrėžta. Kaip jau nagrinėta poskyryje 4.3.1.1, dažnai nenorima riboti, kokie simboliai yra galimi vartotojo įvestyje. Taip pat, patirtis rodo, jog kartais apsaugos priemonės paprasčiausiai pamirštamos pritaikyti, todėl visuomet yra galimybė, jog filtravimas taip pat bus pamirštas.

4.3.2.2. Paruoštosios formuluotės naudojimas

Pavyzdinėje programoje sukurta funkcija, sudaranti SQL užklausą naudojantis paruoštąja formuluote. Sukuriamas užklausos šablonas, kuriame vietoje vartotojo įvesties kintamojo nurodoma reikšmė „%s”. Tuomet užklausa vykdoma užklausos funkcijoje nurodant užklausos

šabloną ir vartotojo įvesties kintamąjį. Funkcija išbandoma su reikšmėmis naudotomis prieš tai buvusiuose bandymuose (4.13 pav.).



4.13 pav. Paruoštosios formuluotės SQL užklausos sudarymo funkcijos veikimo rezultatai

Pastebima, jog duomenų bazės atsakymo laikai praktiškai vienodi, taigi SQL injekcijos ataka buvo nesėkminga.

Paruoštoji formuluotė sėkmingai užkirto kelią SQL injekcijos atakai. Paruoštosios formuluotės panaudojimas kodo prasme elementarus, o jos funkcionalumas prieinamas daugumoje plačiai naudojamų saityno programų karkasų. Dėl paruoštosios formuluotės sprendimų prieinamumo ir efektyvumo užkertant kelią SQL injekcijos pažeidžiamumams, paruoštosios formuluotės naudojimas turėtų būti privalomas visoms programoje naudojamoms SQL užklausoms sudarinėti.

4.4. Pažeidžiamumų prevencijos metodų tyrimo apibendrinimas

Tyrimo metu pažeidžiamumų prevencijos metodai išbandyti praktiškai, paanalizuota jų įgyvendinimo specifika. Pažeidžiamumų prevencijos metodai sukonkretinti, įvertintas jų efektyvumas. Tyrimo metu gautos žinios pritaikytos sudarant rekomendacijų sąrašą pateikiamą darbo rezultatų skyriuje.

5. REZULTATAI IR APIBENDRINIMAS

Dažniausiai pasitaikantys saityno programų pažeidžiamumai kelia didelę grėsmę jų vartotojų duomenų saugumui. Kibernetinių atakų nukreiptų į saityno programas kiekvienais metais vis daugėja, todėl sistemų apsaugojimas tampa vis aktualesnė tema. Sistemų kūrėjai privalo užtikrinti, jog jų sukurtos saityno programos bus atsparios įsilaužėliams, o tokie duomenys kaip vartotojų slaptažodžiai bus saugūs net po sėkmingos atakos.

Išanalizuotos spragų atsiradimo priežastys, jų pavyzdžiai. Padarytos išvados, jog pažeidžiamumai atsiranda dėl klaidingo autorizacijos mechanizmo bei naudojamų tiesioginių objektų identifikatorių, netinkamo vartotojo įvesties filtravimo, atvaizdavimo bei naudojimo sudarinėjant užklausas į duomenų bazes, nesaugaus slaptažodžių laikymo ir neužtikrintos saugios komunikacijos tarp vartotojo ir serverio.

Problemai spręsti apžvelgti spragų prevencijos metodai. Nustatyta, jog IDOR spragų prevencijai objektams siūloma naudoti sunkiai atspėjamus identifikatorius atitinkančius anonimiškumo ir neatsekamumo reikalavimus, taikyti rolėmis grįsta autorizacijos modelį bei vartotoją identifikuoti pagal jo prieigos raktą. Saugiai komunikacijai tarp vartotojo ir serverio rekomenduojama naudoti HTTPS protokolą, kurio naudojimas užtikrinamas HSTS antraštės pagalba. Vartotojų slaptažodžius galima apsaugoti jiems pritaikant maišos algoritmus, kurie būtų atsparūs grubios jėgos, žodyno ir vaivorykštinės lentelės atakoms. Injekcijos pažeidžiamumams išvengti vartotojo įvestis turi būti filtruojama, ją atvaizduojant taikomas kodavimas, o įvestį naudojant SQL užklausose, joms sudaryti turi būt naudojamos paruoštosios formuluotės.

Išbandyti ir ištirti konkretūs prevencijos metodai, įvertintas jų efektyvumas, atsižvelgta į analizuotus versle buvusių spragų pavyzdžius. Pagal analizės bei tyrimų rezultatus, siūlomas dažniausių saugumo spragų saityno programose prevencijos priemonių sąrašas:

- IDOR pažeidžiamumų prevencijai:
    - Kaip nuorodas objektams naudoti netiesioginius identifikatorius. Pagal šio darbo metu atliktą tyrimą rekomenduojama naudoti ketvirtosios versijos UUID identifikatorius paremtus kriptografiškai saugiais atsitiktinių skaičių generatoriais.
    - Naudoti rolėmis grįstą autorizacijos modelį ir taikyti prieigos uždraudimo pagal nutylėjimą politiką.
    - Kiekvienos užklausos į objektą metu įsitikinti, jog vartotojas vykdantis užklausą turi teisę prieiti prie prašomo objekto.
    - Identifikuoti vartotoją privalu naudojantis prieigos raktu, užuot pasitikėjus vartotojo pačio pateiktu tiesioginiu identifikatoriumi ar rolės kintamuoju.
- Kriptografinių pažeidžiamumų prevencijai:
    - Siekiant užtikrinti, jog visai komunikacijai tarp vartotojo ir serverio naudojamas HTTPS protokolas, svetainė turi įgyvendinti HSTS politiką, nustatydama Strict-Trasport-Security antraštę su direktyvomis *max-age=31536000* ir *includeSubDomains*.
    - Pageidautina užpildyti paraišką Google, kad būtų galimas Strict-Trasport-Security antraštės *preload* direktyvos naudojimas.
    - Slaptažodžiams laikyti rekomenduojamas Scrypt maišos algoritmas su 65536 iteracijomis bei lygiagretinimo parametro reikšmė 1 ir bloko dydžio reikšme lygia 8.

    - Kiekvienam slaptažodžiui naudoti druskos reikšmę, kuri yra bent 32 baitų ilgio ir yra atsitiktinai sugeneruojama kiekvienam vartotojo slaptažodžiui.
- Injekcijos pažeidžiamumų prevencijai:
    - Vartotojo įvestį saityno programos serveryje filtruoti naudojantis karkasų siūlomomis funkcijomis arba reguliariųjų išraiškų pagalba, įsitikinant, jog vartotojo įvestis atitinka pageidaujamą formatą.
    - Visai atvaizduojamai vartotojo įvesčiai pritaikyti išvesties kodavimą, atsižvelgiant į kontekstą, kuriame įvestis bus atvaizduota. Šiam tikslui taikyti saityno programų karkasų siūlomas funkcijas serverio pusėje ir JavaScript funkcijas programos kliento pusės kode.
    - Programoje taikyti CSP politiką.
    - Programoje naudojamiems slapukams taikyti *HttpOnly* atributą.
    - SQL užklausoms sudaryti naudoti paruoštąsias formuluotes.

Pateiktų rekomendacijų taikymas negarantuoja, jog kuriama saityno programa bus atspari visoms kibernetinėms atakoms, tačiau tai geras atspirties taškas, padedantis apsisaugoti nuo dažniausiai pasitaikančių saityno programų pažeidžiamumų.

Darbo uždaviniai sėkmingai atlikti, pasiektas išsikeltas tikslas. Ateityje rekomendacijų sąrašas gali būti praplečiamas, norint aprėpti daugiau saityno programų pažeidžiamumų arba redaguojamas atsižvelgiant į pokyčius nagrinėtose spragose ar prevencijos metoduose.

LITERATŪRA

1. S. Gupta and B. B. Gupta, „Detection, Avoidance, and Attack Pattern Mechanisms in Modern Web Application Vulnerabilities: Present and Future Challenges*"*. *International Journal of Cloud Applications and Computing (IJCAC)*, pp. 1–43, Jul. 2017, doi: 10.4018/IJCAC.2017070101.
2. Immunefi. „Polygon Lack Of Balance Check Bugfix Review — $2.2m Bounty". Medium.com. https://medium.com/immunefi/polygon-lack-of-balance-check-bugfix-postmortem-2-2m-bounty-64ec66c24c7d (Accesed Feb. 17, 2022).
3. B. W. Vincent, „Understanding Hacking-as-a-Service Markets". M.S. thesis, Dep. Comp. Sci., Arizona State University, AZ, USA, 2018.
4. „2020 SonicWall Cyber Threat Report", SonicWall, Milpitas, California, USA, 2020. [Online]. Accessed: Feb. 18, 2022. Available: https://www.sonicwall.com/resources/white-papers/2020-sonicwall-cyber-threat-report
5. „State of the Web Security 2020", CDNetworks, Diamond Bar, California, USA, 2020. [Online]. Accessed: Feb. 18, 2022. Available: https://www.cdnetworks.com/news/state-of-the-web-security-2020
6. A. van der Stock, B. Glas, N. Smithline and T. Gigler. OWASP Top 10 – 2021 (2021). Accessed: Feb 21, 2022. Available: https://owasp.org/Top10/
7. F. Kong (2016). Research on Security Technology based on WEB Application. Linyi University, Shandong China.
8. D. Bhatt, „Cyber Security Risks for Modern Web Applications: Case Study Paper for Developers And Security Testers". *International Journal of Scientific and Technology Research 7.5*, pp. 232-235, May 2018.
9. S. Nagpure and S. Kurkure, „Vulnerability Assessment and Penetration Testing of Web Application". *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pp. 1-6, Aug. 2017, doi: 10.1109/ICCUBEA.2017.8463920.
10. Hackerone vulnerability coordination and bug bounty platform. Hackerone.com. https://www.hackerone.com/ (Accessed Feb. 22, 2022).
11. Bugcrowd crowdsourced security platform. Bugcrowd.com. https://www.bugcrowd.com/ (Accessed Feb. 22, 2022).
12. H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu and J. Xu, „An Automatic Vulnerability Scanner for Web Applications". *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1519-1524, 2020, doi: 10.1109/TrustCom50675.2020.00207.
13. Q. Wang, J. Sun, C. Wang, S. Zhang, S. Xuanyuan and B. Zheng, „Access Control Vulnerabilities Detection for Web Application Components". *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC)* and *IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 24-28, May 2020, doi: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00016.
14. J. Indranil, and A. Oprea. „AppMine: Behavioral Analytics for Web Application Vulnerability Detection". Proceedings of the *2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp. 69-80, Nov. 2019, doi: 10.1145/3338466.3358923.
15. J. Thomé, L. K. Shar, D. Bianculli and L. Briand. „An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications Through Security Slicing and

Hybrid Constraint Solving“. *IEEE Transactions on Software Engineering, vol. 46, no. 2*, pp. 163-195, Feb. 2020, doi: 10.1109/TSE.2018.2844343.
16. T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen. „Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines“. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 145-154, Oct. 2020. doi: 10.1109/EDOC49727.2020.00026.
17. J. Im, J. Yoon, and M. Jin. „Interaction Platform for Improving Detection Capability of Dynamic Application Security Testing“. *SECRYPT*, pp. 474-479, Jul. 2017.
18. J. R. B. Higuera, , J. B. Higuera, J. A. S. Montalvo, J. C. Villalba, and J. J. N. Pérez. „Benchmarking approach to compare web applications static analysis tools detecting OWASP top ten security vulnerabilities“. *Computers, Materials & Continua vol.64*, no.3, pp. 1555-1577, Jun. 2020. doi: 10.32604/cmc.2020.010885.
19. J. Manico, J. Maćkowski. OWASP Cheat Sheet Series (2018). Accessed: Feb 21, 2022. Available: https://cheatsheetseries.owasp.org/index.html
20. Community-Developed List of Software and Hardware Weakness Types. cwe.mitre.org. https://cwe.mitre.org/data/index.html (Accessed Feb. 27, 2022).
21. J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su and B. Fang, „A Survey on Access Control in the Age of Internet of Things“. *IEEE Internet of Things Journal, vol. 7, no. 6*, pp. 4682-4696, Jun. 2020, doi: 10.1109/JIOT.2020.2969326.
22. O. B. Fredj, O. Cheikhrouhou, M. Krichen, H. Hamam, A. Derhab. „An OWASP top ten driven survey on web application protection methods“. *International Conference on Risks and Security of Internet and Systems*, pp. 235-252, Nov. 2020, doi: 10.1007/978-3-030-68887-5_14.
23. J. Katz , and Y. Lindell, *Introduction to Modern Cryptography. 2nd ed. Chapman & Hall/CRC Cryptography and Network Security Series*. Boca Raton, FL, USA: CRC Press, 2014.
24. M. A. Al-Shabi. „A survey on symmetric and asymmetric cryptography algorithms in information security“. *International Journal of Scientific and Research Publications (IJSRP)*, pp. 576-589, Mar. 2019.
25. O.G. Abood and S. K. Guirguis. „A Survey on Cryptography Algorithms“. *International Journal of Scientific and Research Publications*, pp. 495-516, Jul. 2018, doi: 10.29322/IJSRP.8.7.2018.p7978.
26. Yahoo data breach: NCSC response. ncsc.gov.uk. https://www.ncsc.gov.uk/news/yahoo-data-breach-ncsc-response (Accessed Mar. 11, 2022).
27. The Need for Greater Focus on the Cybersecurity Challenges Facing Small and Midsize Businesses. sec.gov. https://www.sec.gov/news/statement/cybersecurity-challenges-for-small-midsize-businesses.html (Accessed Mar. 11, 2022).
28. „ENISA Threat Landscape 2020 – Data Breach“, The European Union Agency for CyberSecurity, Athens, Greece, 2020. [Online]. Accessed: Mar. 16, 2022. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-data-breach/at_download/fullReport
29. Automobilių nuomos bendrovei skirta bauda dėl duomenų saugumo pažeidimo pagal Bendrąjį duomenų apsaugos reglamentą. vdai.lrv.lt. https://vdai.lrv.lt/lt/naujienos/automobiliu-nuomos-bendrovei-skirta-bauda-del-duomenu-saugumo-pazeidimo-pagal-bendraji-duomenu-apsaugos-reglamenta (Accessed Mar. 21, 2022).

30. M. Stevens, E. Bursztein, P. Karpman, A. Albertini and Y. Markov. „The first collision for full SHA-1“. *Annual international cryptology conference*, pp. 570-569, Jul. 2017, doi: 10.1007/978-3-319-63688-7_19.
31. Customer security alert. helpx.adobe.com. https://helpx.adobe.com/x-productkb/policy-pricing/customer-alert.html (Accessed Mar. 27, 2022).
32. R. P. Adhie, Y. Hutama, A. S. Ahmar and M. I. Setiawan. „Implementation cryptography data encryption standard (DES) and triple data encryption standard (3DES) method in communication system based near field communication (NFC)“. *Journal of Physics: Conference Series*, Feb. 2018, doi: 10.1088/1742-6596/954/1/012009.
33. Update to Current Use and Deprecation of TDEA. csrc.nist.gov. https://csrc.nist.gov/News/2017/Update-to-Current-Use-and-Deprecation-of-TDEA (Accessed Apr. 2, 2022).
34. S. Kumar, R. Mahajan, N. Kumar and S. K. Khatri. "A study on web application security and detecting security vulnerabilities". *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 451-455, Sep. 2017, doi: 10.1109/ICRITO.2017.8342469.
35. A. W. Marashdih and Z. F. Zaaba. "Detection and Removing Cross Site Scripting Vulnerability in PHP Web Application". *2017 International Conference on Promising Electronic Technologies (ICPET)*, pp. 26-31, Oct. 2017, doi: 10.1109/ICPET.2017.11.
36. M. A. M. Yunus, M. Z. Brohan, N. M. Nawi, E. S. M. Surin, N. A. M. Najib and C. W. Liang. „Review of SQL injection: Problems and prevention“. *JOIV: International Journal on Informatics Visualization*, pp. 215-219, Jun. 2018, doi: 10.30630/joiv.2.3-2.144.
37. M. Schaffer, P. Schartner and S. Rass. „Universally Unique Identifiers: How To Ensure Uniqueness While Protecting The Issuer's Privacy“. *The 2007 International Conference on Security and Management, SAM'07*, pp. 198-204, Jun.. 2007.
38. J. Kallin, I. L. Valbuena. *A comprehensive tutorial on cross-site scripting*. Accessed: Apr. 29, 2022. Available: https://excess-xss.com/
39. A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel and P. Tabriz. „Measuring HTTPS Adoption on the Web“. *26th USENIX security symposium (USENIX security 17)*, pp. 1323-1338, Aug. 2017.
40. M. Busato, „The State of Strict Transport Security: Current Deployment and Correct Configuration“. M.S. thesis, Dep. Comp. Sci., Università Ca' Foscari Venezia, Italy, 2021.
41. C. Ntantogian, S. Malliaros, C. Xenakis. „Evaluation of Password Hashing Schemes in Open Source Web Platforms“. *Computers & Security*, pp. 206-224, Mar. 2019, doi: 10.1016/j.cose.2019.03.011.
42. Have I Been Pwned leaked account lookup service. haveibeenpwned.com. https://haveibeenpwned.com/ (Accessed May 2, 2022).
43. Cybernews password leak checking tool. cybernews.com. https://cybernews.com/password-leak-check/ (Accessed May 3, 2022).
44. G. Hatzivasilis. „Password-Hashing Status“. *Cryptography*, Jun. 2017, doi: 10.3390/cryptography1020010.
45. L. Weichselbaum, M. Spagnuolo, S. Lekies and A. Janc. „Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy“. In Proceedings of the *2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1376-1387, Oct. 2016, doi: 10.1145/2976749.2978363.

46. MDN Web Docs - Content Security Policy (CSP). developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP (Accessed May 11, 2022).
47. *Information technology — Procedures for the operation of object identifier registration authorities — Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers*, ISO/IEC Standard No. 9834-8:2014, International Organization for Standardization, 2014.
48. F. H. Mathis. „A Generalized Birthday Problem“. *SIAM Review*, pp. 265-270, Jun. 1991, doi: 10.1137/1033051.
49. Wireshark tool homepage. www.wireshark.org. https://www.wireshark.org/ (Accessed: May 14, 2022).
50. Hack Yourself First vulnerable website to help web developers identify security risks. hack-yourself-first.com. https://hack-yourself-first.com/ (Accessed Apr. 6, 2022).
51. J. Alwen, B. Chen, K. Pietrzak, L. Reyzin and S. Tessaro. „Scrypt Is Maximally Memory-Hard“. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 36-62, Apr. 2017, doi: 10.1007/978-3-319-56617-7_2.
52. L. Ren and S. Devadas. „Bandwidth hard functions for ASIC resistance“. *Theory of Cryptography Conference*, pp. 466-492, Nov. 2017, doi: 10.1007/978-3-319-70500-2_16.

1 PRIEDAS

Programa, kuri buvo naudota maišos algoritmų greičio testavimui.

```
import argparse
import hashlib
import time

#Paruosiamas komandines eilutes argumentu interpretatorius
parser = argparse.ArgumentParser(description='Tikrinamas maisos algoritmu greitis')
parser.add_argument('toHash', type=str)

args = parser.parse_args()
toHash = args.toHash.encode('utf-8')
salt = bytes.fromhex('df1f2d3f155d78a4')

#Tikrinamas 10000 SHA256 operaciju veikimo laikas

t_0 = time.time()

for a in range(10000):
    sha256_hash = hashlib.sha256(toHash).hexdigest()

time_sha256 = time.time() - t_0
print(time_sha256)

#Tikrinamas 10000 PBKDF2 operaciju veikimo laikas

t_0 = time.time()

for a in range(10000):
    pbkdf2_hash = hashlib.pbkdf2_hmac('sha256', toHash, salt, 1024).hex()

time_pbkdf2 = time.time() - t_0
print(time_pbkdf2)

#Tikrinamas 10000 Scrypt operaciju veikimo laikas

t_0 = time.time()

for a in range(10000):
    scrypt_hash = hashlib.scrypt(toHash, salt=salt, n=1024, r=8, p=1).hex()

time_scrypt = time.time() - t_0
print(time_scrypt)
```

2 PRIEDAS

Funkcijos, naudotos įvesties filtravimo bandymams.

```
def url_filtravimas(url_reiksme):
    filtruota_reiksme = url_reiksme.replace("javascript:", "")

    return filtruota_reiksme

def ar_tinkamas_url_biblioteka(url_reiksme):
    check = validators.url(url_reiksme)

    if isinstance(check, ValidationFailure):
        return False

    return check

def ar_tinkamas_url_regex(url_reiksme):
    regex = ("((http|https):\/\/)[a-zA-Z0-9-._~]+\.[a-z]+[?:a-zA-Z0-9-._~/]*")

    regex_url = re.compile(regex)

    if(re.search(regex_url, url_reiksme)):
        return True,
    else:
        return False
```

3 PRIEDAS

Programa, kuri buvo naudota SQL prevencijos priemonėms testuoti.

```
import mysql.connector
import time

passdb = mysql.connector.connect(
    host="127.0.0.1",
    user="redaguota",
    password="redaguota",
    database="redaguota"
)

pasirinkimas = input("Iveskite norima naudoti funkcija:")
vartotojo_ivestis = input("Iveskite reiksme, kurios ieskosite: ")

def SQL_uzklausa_nesaugi():
    dbcursor = passdb.cursor()

    sql_uzklausa = "SELECT * FROM vartotojai WHERE id =" + vartotojo_ivestis

    dbcursor.execute(sql_uzklausa)
    res = dbcursor.fetchall()

    for it in res:
        print(it[0])

def SQL_uzklausa_filtravimas():
    dbcursor = passdb.cursor()

    if(vartotojo_ivestis.isnumeric()):
        sql_uzklausa = "SELECT * FROM vartotojai WHERE id =" + vartotojo_ivestis

        dbcursor.execute(sql_uzklausa )
        res = dbcursor.fetchall()

        for it in res:
            print(it[0])
    else:
        print("Ivesta reiksme neatitinka reikalavimu")

def SQL_uzklausa_paruostoji_formuluote():
    dbcursor = passdb.cursor(prepared=True)

    sql_uzklausa = "SELECT * FROM vartotojai WHERE id = %s"
    parametrizuota_ivestis = (vartotojo_ivestis,)

    dbcursor.execute(sql_uzklausa, parametrizuota_ivestis)
    res = dbcursor.fetchall()

    for it in res:
        print(it[0])

t_0 = time.time()
if pasirinkimas == "1":
    SQL_uzklausa_nesaugi()
elif pasirinkimas == "2":
    SQL_uzklausa_filtravimas()
elif pasirinkimas == "3":
    SQL_uzklausa_paruostoji_formuluote()
else:
    print("Neteisingas pasirinkimas\n")

print("Funkcijos vykdymo laikas: " + str(time.time() - t_0))
```