KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Arminas Katilius

TINKLALAPIŲ APSAUGA NUO XSS ATAKŲ NAUDOJANT JAVA

Baigiamasis magistro projektas

Vadovas
dr. J. Čeponis

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

TINKLALAPIŲ APSAUGA NUO XSS ATAKŲ NAUDOJANT JAVA

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

Vadovas
dr. J. Čeponis

Recenzentas
doc. G. Vilutis

Projektą atliko
Arminas Katilius

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas
(Fakultetas)
Arminas Katilius
(Studento vardas, pavardė)
621E10003
(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Tinklalapių apsauga nuo XSS atakų naudojant Java“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

___ - _____ ____
Kaunas

Patvirtinu, kad mano, **Arminas Katilius**, baigiamasis projektas tema „Tinklalapių apsauga nuo XSS atakų naudojant Java“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

_____ _____
(vardą ir pavardę įrašyti ranka) *(parašas)*

Arminas, Katilius. Tinklalapių apsauga nuo XSS atakų naudojant Java. Magistro baigiamasis projektas / vadovas dr. Jonas Čeponis; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informacijos ir informacinių technologijų sauga

Reikšminiai žodžiai: XSS, *Java, HTML, Javascript, apsauga, įrankių analizė*

Kaunas, 2016. 58 p.

SANTRAUKA

Darbe analizuojamas vienas iš tinklalapių pažeidžiamumų suteikiantis galimybę piktavaliams vykdyti pavojingą kodą svetimuose tinklalapiuose. Analizuojamos XSS pažeidžiamumų atsiradimo priežastys, XSS pažeidžiamumų tipai ir skirtumai tarp jų.

Kitoje darbo dalyje analizuojami skirtingi literatūros šaltiniai (knygos, moksliniai straipsniai ir tinklaraščių straipsniai) aprašantys apsaugos nuo XSS pažeidžiamumų metodus. Taip pat aptariami įrankiai padedantys apsisaugoti nuo šių pažeidžiamumų, arba suteikiantys galimybę surasti tokius pažeidžiamumus.

Aptarus bendras apsaugos priemones, analizuojama „Java“ programavimo kalba sukurtiems tinklalapiams skirtos apsaugos priemonės. Konkrečiau, tinklalapiams sukurtiems naudojant „Spring“ karkasą. Įvardijus šių įrankių ir priemonių pranašumus, pateikiamas pasiūlymas kaip šiuos įrankius galima patobulinti.

Likusioje darbo dalyje aprašomas XSS apsaugos priemonių patobulinimo įgyvendinimas ir pateikiami panaudojimo pavyzdžiai.

Katilius, Arminas. *Websites Protection from XSS Attacks Using Java*: Master‘s thesis in ..... / supervisor assoc. dr. Jonas Čeponis. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Information and information technology security

Key words: XSS, *Java, HTML, Javascript, security, tools analysis*

Kaunas, 2016. 58 p.

SUMMARY

The paper focuses on one of the websites vulnerabilities that allow person to execute malicious code in vulnerable website. The paper analyzes the causes of XSS vulnerabilities, XSS vulnerability types and the differences between them.

Secondly, paper analyzes the different literature sources (books, scientific articles and blog articles) describing protection against XSS vulnerabilities methods. After that, tools that help protect against these vulnerabilities, or allow them to find such vulnerabilities are analyzed.

Moreover, paper discusses common security measures using Java programming language. More specifically, web pages created using the Spring framework. After identifying advantages of those tools and measures, the proposal on how to improve them is made.

The rest of the work describes proposed improvement implementation details and provides usage examples.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

AJAX – Asynchronous „Javascript“ and XML. Technologija leidžianti kliento pusėje esančiam „Javascript“ kodui bendrauti su serveriu asinchroniškai, t. y. gauti dalį puslapio duomenų iš serverio ir atnaujinti tik pasikeitusią dalį.

Alexa svetainių reitingas – reitingas skaičiuojanti svetainės populiarumą, pagal vartotojų, turinčių įdiegtą Alexa įrankių juostos programą, apsilankymų toje svetainėje skaičių.

ASP - Active Server Pages. Microsoft sukurta tinklalapių kūrimo platforma.

Base64 – grupė užkodavimo algoritmų leidžiančių dvejetainio formato duomenis atvaizduoti tekstu konvertuojant skaičius, kurių pagrindas yra 2 į skaičius, kurių pagrindas yra 64.

ColdFusion – komercinė tinklalapių kūrimo platforma.

CSS – Cascading Style Sheets. Stiliaus aprašymo kalba, dažniausiai naudojama HTML kalba aprašytų puslapių dizainui.

DHTML – Dynamic HTML. Technologijų rinkinys leidžiantis kurti interaktyvius ir animuotus puslapius. Dažniausiai naudojama „Javascript“ programavimo kalba.

DOM – Document Object Mode. Programinė sąsaja leidžianti pasiekti HTML kalba aprašyto puslapio elementus per medžio duomenų struktūrą.

Flash – multimedijos platforma skirta kurti kompiuterinę grafiką, animuotus puslapius ar internetinius žaidimus.

GET – HTTP užklausos tipas.

HTML – Hyper Text Markup Language. Žymų programavimo kalba naudojama aprašyti internetinių puslapių struktūrą.

HTML žyma – HTML kalbos dalis aprašanti puslapio elementą. Žyma gali turėti turinį ir atributus.

HTML žymos atributas – elementas detaliau aprašantis HTML žymą.

HTTP – Hypertext Transfer Protocol. Hiperteksto perdavimo protokolas

Java – kompiliuojama objektiškai orientuota programavimo kalba.

Java anotacijos – Java programavimo kalbos dalis, leidžianti žymėti patį Java programinį kodą.

Javascript – interpretuojama programavimo kalba naudojama interaktyvių puslapių kūrimui, vykdant programinį kodą kliento pusėje. Taip pat galima naudoti ir kaip serverio programavimo kalbą.

JSON – Javascript Object Notation. „Javascript“ kalba paremtas duomenų tipas.

Kompiliatorius – programinė įranga paverčianti programinį kodą parašytą viena programine kalba į kodą, kita programine kalba.

OWASP – Open Web Application Security Project. Internetinė bendruomenė kurianti laisvai prieinamas programas, straipsnius ir technologijas skirtas internetinių programų apsaugai.

PDF – Portable Document Format. Failo formatas naudojamas atvaizduoti dokumentus.

Perl – aukšto lygio dinaminė, interpretuojama programavimo kalba.

PHP – scenarijų kalba dažniausiai naudojama tinklalapių kūrimui.

QuickTime – Kompanijos „Apple“ sukurta multimedijos platforma.

Spring – atvirojo kodo karkasų ir įrankių rinkinys skirtas Java platformai

SQL – Structured Query Language. Programavimo kalba skirta bendrauti su reliacinėmis duomenų bazėmis.

URL – Uniform Resource Locator. Nuoroda į internetinį resursą.

UTF-8 – simbolių užkodavimo sistema.

Wordpress – tinklaraščių ir kitų panašaus turinio svetainių turinio valdymo sistema.

XSS – Cross-site scripting. Tinklalapių pažeidžiamumo tipas leidžiantis vykdyti svetimą programinį kodą pažeidžiamoje svetainėje.

ĮŽANGA

Atliktas darbas priklauso studijų programai „Informacijos ir informacinių technologijų sauga“

Darbe analizuojamos XSS pažeidžiamumų priežastys ir būdai nuo jų apsisaugoti. Nors XSS pažeidžiamumai nėra patys pavojingiausi, tačiau jie yra vieni iš dažniausiai pasitaikančių. Dažnai tinklalapių kūrėjai neskiria pakankamai dėmesio tokioms saugos spragoms, todėl tokie pažeidžiamumai pastebimi tiek populiariose ir gerai žinomose svetainėse, tiek įrankiuose palengvinančiuose svetainių kūrimą (turinio valdymo sistemose ar tinklalapių komponentuose).

Kadangi nėra sprendimo tinkančio visiems atvejams, svarbu gerai suprasti XSS pažeidžiamumų priežastis ir dažniausiai pasitaikančias saugos spragas. Darbo tikslas yra išanalizuoti esamą padėtį ir pateikti patobulintą apsaugos nuo XSS atakų įrankį.

Pagrindinės užduotys:

- Išanalizuoti XSS pažedžiamumų priežastis ir skirtingus jų tipus.
- Išanalizuoti saugos metodus ir įrankius.
- Sudaryti įrankio skirto Java programavimo kalbai patobulinimo pasiūlymą.
- Sukurti įrankio patobulinimą ir aprašyti jo įgyvendinimą bei panaudojimo būdus.

Dokumentas sudarytas iš 6 dalių:

1. XSS atakos ir egzistuojantys apsaugos būdai. Aptariamos pažeidžiamumų priežastys ir būdai nuo jų apsisaugoti.
2. Esamų sprendimų patobulinimo aprašymas. Analizuojami saugos sprendimai naudojant Java programavimo kalbą ir pateikiamas patobulinimo pasiūlymas.
3. Įrankių patobulinimo įgyvendinimas. Aprašomi ir pagrindžiami patobulinimo įgyvendinime atlikti sprendimai.
4. Įrankių patobulinimo įgyvendinimas. Aprašomos konkrečios detalės apie programinį kodą ir jo struktūrą įgyvendinant patobulinimą.
5. Įrankio panaudojimo pavyzdžiai. Pateikiami įrankio panaudojimo pavyzdžiai su kodo ir užklausų pavyzdžiais.
6. Rezultatų apibendrinimas ir išvados. Pateikiama apibendrinta informacija apie šiame darbe atliktų tyrimų rezultatus.
7. Literatūra. Literatūros, kurios pagalba buvo sukurtas šis darbas sąrašas.

1. XSS ATAKOS IR EGZISTUOJANTYS APSAUGOS BŪDAI

Šiame skyriuje apžvelgiamos XSS atakos, jų skirstymas į kategorijas, naudojami bendri apsisaugojimo būdai ir galimybės apsisaugoti nuo atakų, naudojant Java programavimo kalbą.

1.1. XSS atakų ir apsaugos būdų analizės tikslai

Analizės dalyje keliami šie tikslai:

1. Išanalizuoti XSS atakų tipus ir ištirti kokie nauji pažeidžiamumai atsirado kartu su HTML5 ir AJAX technologijomis.
2. Išanalizuoti metodinės ir mokslinės literatūros aktualumą apsisaugojant nuo naujausių XSS pažeidžiamumų.
3. Įvertinti analizės įrankius, skirtus apsisaugoti nuo XSS tipo atakų.
4. Įvertinti apsaugos nuo XSS atakų priemones, suderinamas su Java platforma

Analizės metu siekiama išsiaiškinti naujus XSS pažeidžiamumus ir analizės bei apsaugos įrankių gebėjimą su jais susitvarkyti, juos aptikti ir/ar pataisyti.

1.2. XSS atakos ir jų sukeliami pavojai

Daugelis šiuolaikinių svetainių paremtos interaktyviu bendravimu su vartotoju, t. y. informacijos iš vartotojo paėmimu ir atnaujintos svetainės, pagal tuos parametrus, atvaizdavimu. Pavyzdžiui, vartotojui įvedus savo prisijungimo duomenis jam atveriamas puslapis su veiksmų, kuriuos jis gali daryti, sąrašu. Taip pat dažnai lankytojas gali įvertinti tam tikros svetainės turinį pateikdamas komentarą.

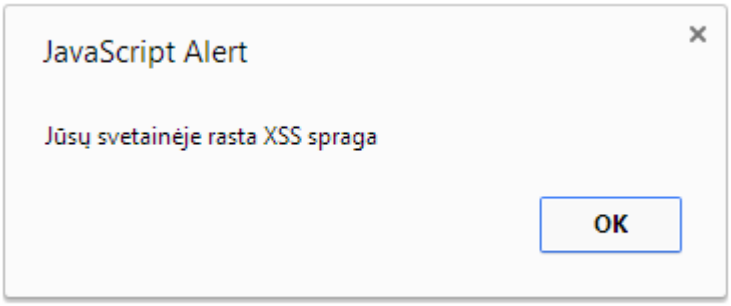
Jei svetainės kūrėjai aklai pasitiki vartotojo įvesta informacija ir ją vaizduoja nepatikrinę, gali atsirasti piktavalių, kurie pasinaudoję šia spraga pateiks tokią užklausą, kuri gali sukelti pavojų kitiems vartotojams ar svetainės administratoriui. Tokios spragos vadinamos XSS pažeidžiamumais.

XSS (angl. *Cross-site scripting*) pažeidžiamumas yra tokia spraga, kuria pasinaudojęs užpuolikas gali įterpti išorinį programinį kodą į ją. Tokį pažeidžiamumą radęs užpuolikas gali tiesiog įspėti jus parodydamas iššokantį pranešimą, kaip pavaizduota 1.1 pav. Tačiau šį pažeidžiamumą galima išnaudoti ir kitaip. Užpuolikas programavimo kalba „Javascript“ gali paslėpti originalų puslapio turinį ir vietoj jo pateikti savo. Taip nieko nenutuokiantys vartotojai galėtų atskleisti savo slaptažodį, pavyzdžiui, užpildydami netikrą prisijungimo formą, kuri siunčia duomenis į užpuoliko nurodytą serverį.

XSS pažeidžiamumo spraga leidžia piktavaliams atlikti tokius veiksmus:

- Pavogti aukos sesijos duomenis.
- Manipuliuoti aukos kompiuteryje arba tinkle prie kurio jis prieina, esančiais failais.

- Įrašyti visus klavišų paspaudimus pažeistoje svetainėje.
- Pasiekti vidinį aukos tinklą (intranetą) ir atlikti atakas jame.



1.1 pav. „Javascript“ funkcija „alert“

Dažniausiai tokiu būti pateikiamas „Javascript“ programinis kodas, kurį įvykdo aukos naršyklė.

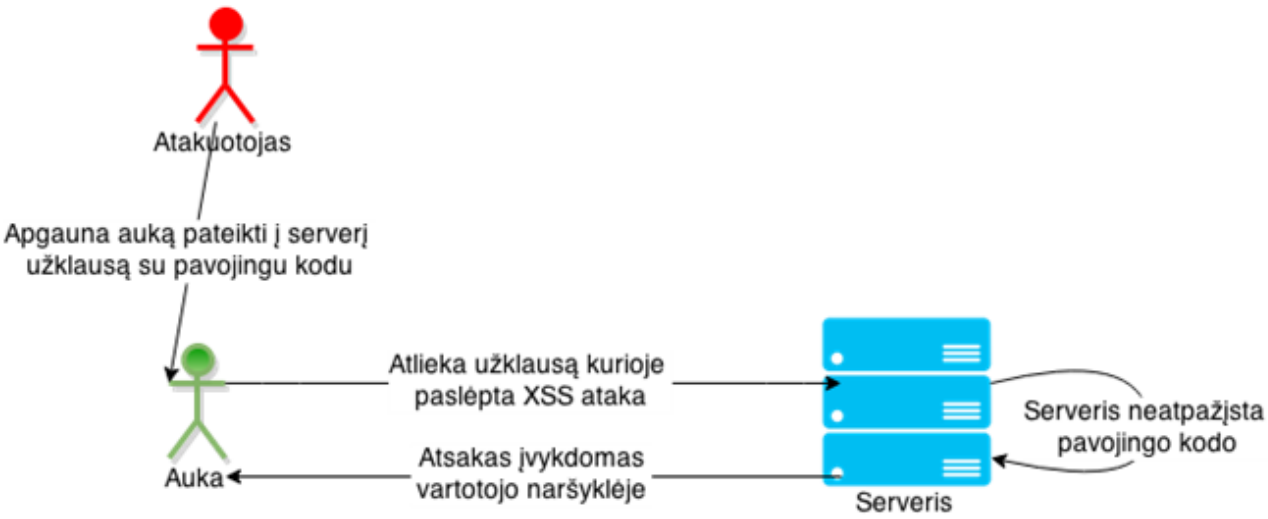
1.3. XSS atakų kategorizacija

XSS spragos gali atsirasti įvairiose sistemos vietose, todėl sistemą galima pažeisti taikant skirtingas technikas. Dėl to skiriasi ir apsisaugojimo būdai. Galima išskirti trijų tipų programinio kodo įterpimus.

1. Vienkartinis (angl. *reflected*)

   Vienkartinis pažeidžiamumas yra vienas iš dažniausiai pasitaikančių (1). Tai toks pažeidžiamumas, kai programinis kodas kartu su užklausa grąžinamas iš karto su atsaku iš serverio ir yra iš karto įvykdomas. Pavyzdžiui, atliekant paiešką tinklalapyje, paieškos frazė gali būti iš karto pateikus užklausą į serverį.

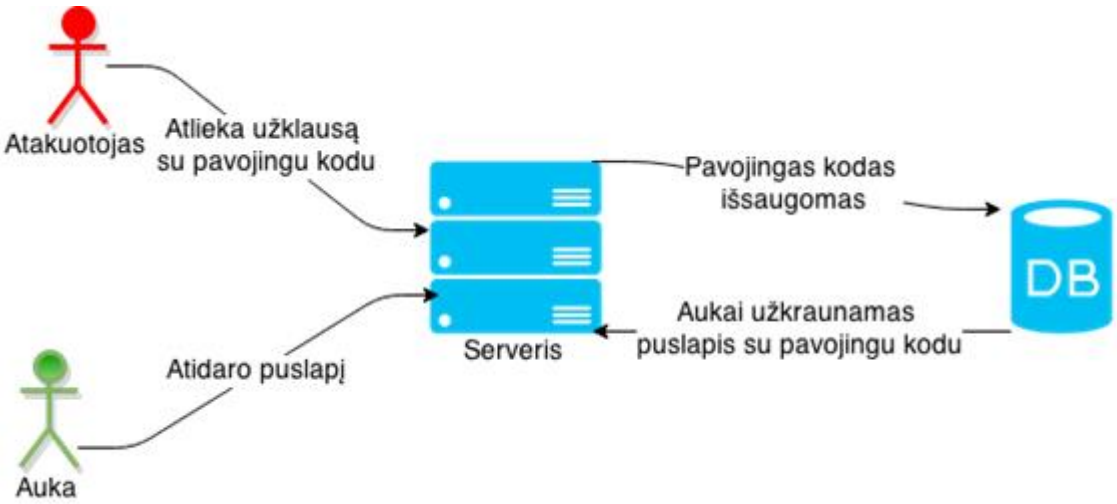
   Kad tokia ataka suveiktų, auka turi atlikti suformuotą pavojingą užklausą, todėl tokios atakos atliekamos pateikiant nuorodas kitose svetainėse ar el. paštu, tikintis, kad auka nuspaus tą nuorodą ir įvykdys pavojingą kodą. Tokios atakos schema pavaizduota 1.2 pav. Vienkartinė XSS ataka.



1.2 pav. Vienkartinė XSS ataka

2. Nuolatinis (angl. *persistent*)

Nuolatiniai XSS pažeidžiamumai yra gana pavojingas pažeidžiamumas, nes nuo jo nukentėti galima tiesiog naršant užkrėstą svetainę, nereikia įvykdyti pavojingos užklausos. Užpuolikui reikia pateikti užklausą su pavojingu kodu, kuri po to yra išsaugojama serveryje ir atvaizduojama. Pavyzdžiui, komentaras su paslėptu „Javascript“ scenarijumi yra išsaugomas svetainėje ir administratorius peržiūrėdamas tą komentarą įvykdo paslėptą scenarijų ir gali atskleisti savo sesijos duomenis užpuolikui. Tokio užpuolimo schema pavaizduota 1.3 pav. Nuolatinė XSS ataka.

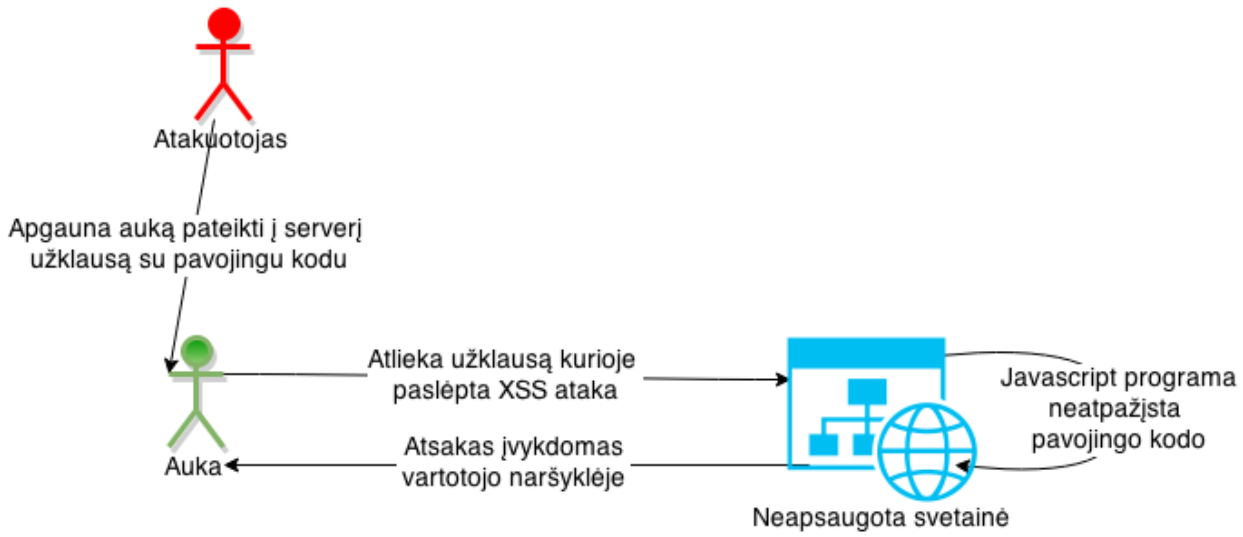


1.3 pav. Nuolatinė XSS ataka

3. Paremtas DOM (dokumentų objektų modeliu)

Dokumentų objektų modeliu paremtos atakos išnaudoja ne užklausų į serverį apsaugos spragas, o „Javascript“ programinio kodo spragas kliento pusėje. Tokį pažeidžiamumą turinčioje svetainėje vykdomas „Javascript“ programinis kodas, kuris paima tam tikrą

informaciją ir atvaizduoja ją nepatikrinęs. Užpuolimo schema primena vienkartinės atakos schema, skiriasi tik tuo, jog vietoj serverio užpuolikas pasinaudoja neapsaugotos svetainės kliento pusės kodu, schema pateikta 1.4 pav. Dokumentų objektų modeliu paremta XSS ataka.



1.4 pav. Dokumentų objektų modeliu paremta XSS ataka

Naršyklės yra įdiegtos saugos priemonės, kurios iš dalies apsaugo nuo vienkartinių XSS atakų. Tokios priemonės neleidžia įgyvendinti paprasčiausių XSS atakų, kai vykdomas „Javascript“ kodas yra tiesiog pateikimas užklausose, tačiau tai neapsaugo nuo sudėtingesnių vienkartinių atakų. Taip pat naršyklės nesugeba apsaugoti vartotojo nuo nuolatinių XSS atakų. Dėl to tinklalapių kūrėjai turėtų neišskirti nei vieno tipo atakų ir stengtis apsisaugoti nuo visų tipų atakų.

1.4. Pažeidžiamų nuo XSS atakų naudotojų analizė

XSS pažeidžiamumai ir šiandien yra nepakankamai įvertintas pavojus svetainei. Nuo jų nukenčia ne tik mažesnės svetainės. Naujų XSS pažeidimų 2014 metais buvo aptikta [1] net ir populiariausiose svetainėse pagal „Alexa“ svetainių reitingą. Keletas tokių svetainių yra:

- ask.com
- microsoft.com
- imdb.com
- bbc.co.uk
- dailymail.co.uk

[1] https://www.xssposed.org.

Svetainės yra registruojamos atvirame XSS pažeidžiamumų archyve „xpossed“ [2] Visi išvardinti pažeidžiamumai yra jau ištaisyti, tačiau tai įrodo, kad naujos XSS saugumo spragos ir nauji metodai vis dar atsiranda.

Remiantis įmonės „White Hat Security“ atliktu tyrimu [3], 2014-aisiais metais XSS pažeidžiamumai atgavo pirmą poziciją populiariausių pažeidžiamumų sąraše, kurią 2013-aisias užleido informacijos nutekėjimui. Be bendrų pažeidžiamumų statistikos, ataskaitoje pateikiamas saugumo palyginimas, pagal programavimo kalbą, su kuria svetainė sukurta. Populiariausiomis internetinių svetainių kalbomis išrinktos Java, .NET ir PHP. Tyrimo išvadose teigiama, kad pažeidžiamumų kiekis iš esmės nuo kalbos nesiskiria, t. y. nėra didelio skirtumo tarp geriausių ir blogiausių rezultatų.

Svetainių pažeidžiamumas, pagal programavimo kalbą, kuria svetainė buvo sukurta procentais pateiktas 1.1 lentelėje. Nors šiame darbe naudojama „Java“ programavimo kalba parašytos svetainės yra vienos iš saugiausių, su „Java“ programavimo kalba parašytų tinklalapių turinčių XSS spragų buvo beveik daugiausiai. Blogiau pasirodė tik „Perl“ programavimo kalba.

1.1 lentelė XSS pažeidžiamumas procentais pagal programavimo kalbą

Programavimo kalba	Svetainių pažeidžiamumas (procentais)
ASP	49
ColdFusion	46
.NET	35
Java	57
Perl	64
PHP	56

Panašūs ir įmonės „Cenzic“ atlikto tyrimo rezultatai (2), jame taip pat teigiama, kad XSS pažeidimai vėl tapo labiausiai paplitusiais pasaulyje, taip pat teigiama, kad iš rastų pažeidžiamumų, net 25 procentai būna būtent XSS.

2014-aisiais metais aptiktas „Wordpress“ XSS pažeidžiamumas (3) pavadintas didžiausia saugumo spraga per 5 metus. Ši saugumo spraga paveikė net 86% procentus svetainių naudojančių “Wordpress“. Pasak Jouko Pynnonen, tyrėjo atradusio šį pažeidžiamumą, šis pažeidžiamumas egzistavo net ketverius metus.

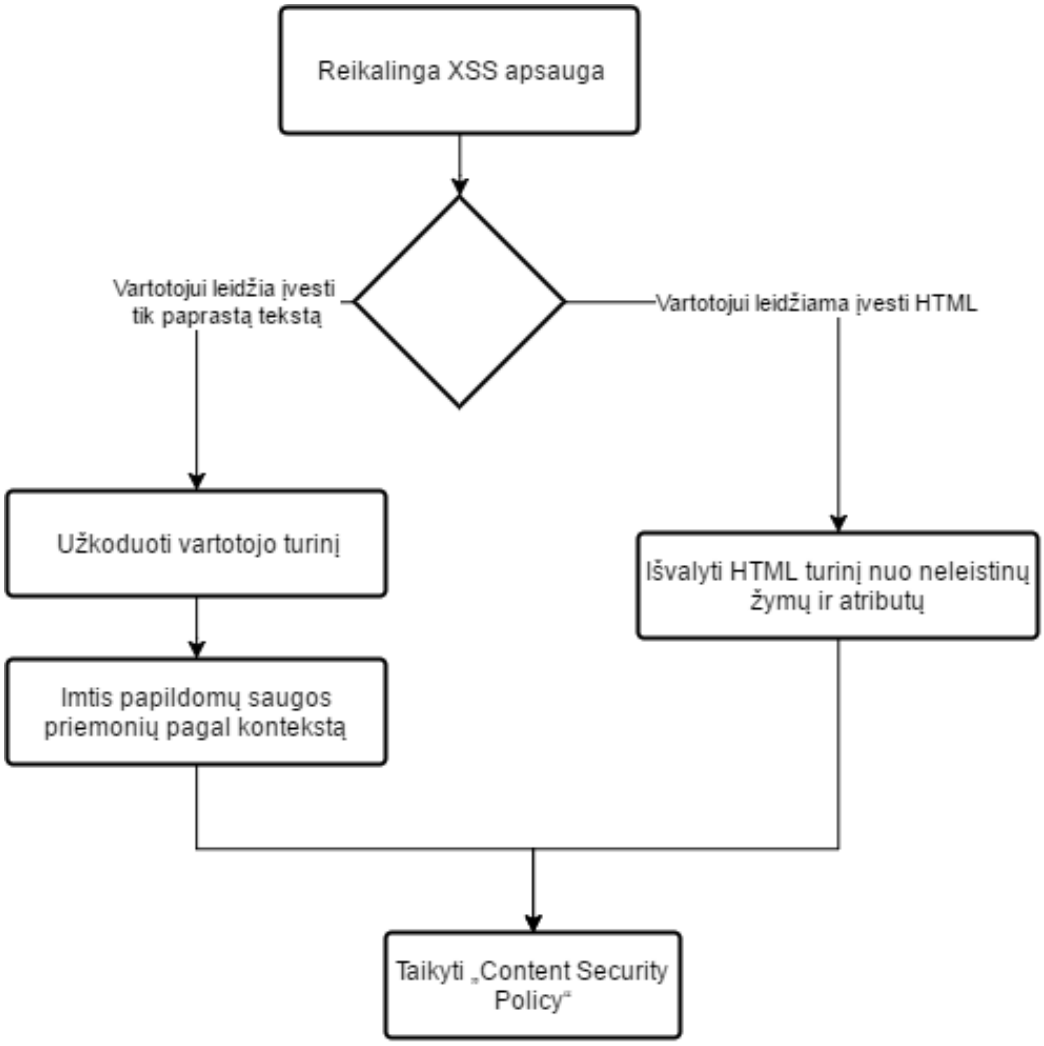
1.5. Esami XSS atakų apsaugos metodai ir priemonės

Apsaugos nuo XSS atakų priemonės stipriai priklauso nuo konteksto (kur vartotojo įvedamas tekstas yra atvaizduojamas) ir nuo įvedamo turinio taisyklių. Pavyzdžiui, jei vartotojui leidžiama įvesti tik paprastą tekstą, turinį apsaugoti yra lengviausia, nes nereikia įvedamo teksto

[2] https://www.xssposed.org.
[3] http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf

interpretuoti kaip HTML kodo. Tačiau, tai apriboja vartotojų galimybes. Pavyzdžiui vartotojai netenka galimybių paryškinti teksto ar kitaip jį suformatuoti.

Reikalingų apsaugos veiksmų seka pateikiamas pav. 1.5. Nors turinio saugumo politika (angl. *„Content Security Policy“*) negarantuoja visiškos apsaugos, rekomenduotina ją visadata taikyti. Taikant šią politiką, vieni to paties šaltinio elementai gali prieiti tik prie kitų to paties šaltino elementų. T. y. iš svetainės „test1.lt“ neturi būti galima pasiekti „test2.lt“ turinio.



1.5 pav. Apsaugos priemonių nuo XSS diagrama

Viena iš keleto grėsmių, kurias gali sukelti XSS spraga yra vartotojo sesijos vagystė. Užpuolikas gavęs prieigą prie aukos slapukų (angl. *Cookies*) gali prisijungti prie svetainės kaip jis. Šiai problemai spręsti buvo sukurtas specialius sausainėlių tipas „HttpOnly“. Toks tipas nustatomas naudojant HTTP atsakymo antraštę. Jei naršyklė palaiko „HttpOnly“ slapukų tipą, tai reiškia, kad slapukų nebus galima nuskaityti iš kliento pusės kodo. Kaip teigiama „Coding Horror“ straipsnyje, „HttpOnly slapukai yra puiki idėja, kuri apsunkina daugybės tipų XSS atakų įgyvendinimą“ (4)

Kita apsaugos priemonė yra naršyklėje įgyvendinta vienkartinių XSS atakų apsauga. Ši apsauga paremta išeinančios užklausos stebėjimu. Naršyklė tikrina vartotojo užklausą ir jei joje randa

„Javascript“ kodą, kuris yra grąžinamas kartu su atsaku iš serverio, šis kodas nėra vykdomas. Visų pirma reikia atkreipti dėmesį, kad ši apsauga nepadės nuo nuolatinių XSS atakų ir DOM elementais paremtų atakų. Taip pat nors ši apsauga neleidžia atlikti pačių paprasčiausių vienkartinių atakų, yra metodų kaip galima apeiti net ir šias naršyklės apsaugas. „Prieš įvykdydama išeinančią užklausą, naršyklė „Internet Explorer“ patikrina, iš kur užklausa buvo atlikta. Taikoma teorija, kad jei užpuolikas nori išnaudoti vienkartinę XSS ataką, užklausa turi ateiti iš tinklalapio, kurį valdo užpuolikas. Vadinasi, jei užklausa vykdoma iš to paties domeno, nėra prasmės tikrinti jos, nes tai būtų resursų švaistymas. Tačiau tai yra neteisinga prielaida, nes XSS atakos gali būti atliktos iš bet kur. Užpuolikas gali išnaudoti šią spragą, apgaudamas sistemą naudodamas nevaliduojamus nukreipimus (angl. *redirects and forwards*)“ (5)

Knygoje „Programų sauga“ (6) pagal „Atvirojo žiniatinklio programų saugumo projektą“ (OWASP) sudarytas šis taisyklių sąrašas:

1. Nepatikimi duomenys privalo būti įkeliami tik į tam tikras vietas;
2. Prieš keliant duomenis į HTML elementą, jie privalo būti patikrinti ir suformatuoti;
3. HTML atributų reikšmės privalo būti patikrintos ir tinkamai suformatuotos;
4. „Javascript“ kintamųjų ar naudojamų atributų reikšmės privalo būti patikrintos ir tinkamai suformatuotos;
5. HTML URL parametrų reikšmės privalo būti patikrintos ir tinkamai suformatuotos;
6. Naudoti HTML saugumo politikos tikrinimo įrankius;
7. Vengti DOM lygio XSS pažeidžiamumų

Pagal sudarytą taisyklių sąrašą galima teigti kad esminė ir svarbiausia apsaugos priemonė nuo XSS atakų yra išeinančio ir įeinančio turinio teisingas apdorojimas, pagal jo naudojimo kontekstą.

Nepaisant taisyklių sąrašų ir įvairiausių saugos metodų, XSS pažeidžiamumų vis atsiranda. „XSS nėra baigta tirti sritis, ji nuolat nustebina pasaulį, nes žmonės sugeba atrasti pačių netikėčiausių metodų užpulti svetainę ir įvykdyti savo kodą. Vis dėlto yra keletas esminių teorijų, kurias reikia suprasti internetinių svetainių programuotojui ar saugumo tyrėjams.“ (7) Knygoje išskiriamos ir aptariamos tokios kategorijos:

- DOM paremtos XSS atakos
- Nukreipimas
- Flash, QuickTime, PDF
- HTTP atsako injekcija
- Programinis kodas ir DHTML realybėje
- XSS ilgio ribojimų įveikimas
- XSS filtrų apėjimas

Knygoje nėra aptariami nauji HTML5 elementai, kuriuos taip pat galima panaudoti XSS atakoms.

2011-tų metų straipsnyje „Countering code injection attacks: a unified approach“ apie svetimo kodo paleidimo atakas siūlomas vieningas metodas tiek SQL, tiek binarinio kodo, tiek XSS atakoms (8). Siūloma sukurti tarpinę funkciją tarp vartotojo ir serverio, kuri veiktų dviem režimais - mokymosi ir produkcijos. Mokymosi rėžime iš vartotojo priimamos užklausos, kuriose pašalinama visa papildoma ir nereikšminga informacija. Užklausa apdorojama taip, kad liktų tik jos signatūra. Produkcijos rėžime atliekami tie patys signatūros generavo žingsniai, tačiau gavus signatūrą ji lyginama su žinomomis, ir jei atitikmuo nerandamas, užklausa ignoruojama kaip pavojinga.

XSS pažeidžiamumų sąrašas pateikiamas OWASP svetainėje. Svetainėje pateikiami keli sąrašai, “XSS Filter Evasion Cheat Sheet“ (9) skirtas jau šiek tiek žinių apie XSS turintiems vartotojams. Čia pateikiamas vadinamas didelis vadinamų vektorių sąrašas su kodo pavyzdžiais ir paaiškinimais. Sąraše „DOM based XSS Prevention Cheat Sheet“ (10) pateikiamas DOM elementams taikomų pažeidžiamumų sąrašas. Jo pagalba galima apsaugoti tinklalapius nuo aktyvių XSS atakų.

Tinklalapyje „Excess XSS“ pateikiamas šis patarimų sąrašas padėsiantis apsisaugoti nuo XSS atakų (11):

- Svarbiausias metodas norint apsaugoti svetainę nuo XSS atakų yra įvestų duomenų tikrinimas.
    - Kuo dažniau reikėtų naudoti įvesties užkodavimą, kad naršyklė interpretuotų duomenis kaip tekstą, o ne kaip programinį kodą.
    - Kartais užkodavimą galima pakeisti papildžius tikrinimo taisykles.
    - Saugus įvesties valdymas turėtų atsižvelgti į puslapio kontekstą.
    - Norint išvengti visų tipų XSS atakų, įvestis reikia apsaugoti tiek kliento, tiek serverio pusėje.
- „Content Security Policy“ prideda papildomą apsaugą, jei užpuolikui pavyktų apeiti įvesties apsaugas.

Jim Manico konferencijoje OWASP-SLC pateikė sąrašą XSS apsaugų, priklausomai nuo duomenų tipo ir konteksto, kuris pateiktas 3 lentelėje (12).

1.2 lentelė XSS apsaugos priemonės pagal duomenų tipą ir kontekstą

Duomenų tipas	**Kontekstas**	**Apsaugos priemonė**
Tekstas (angl. *String*)	HTML žymoje <body>	HTML elementų užodavimas
Tekstas	HTML atributas	Minimalus atributų užkodavimas

Tekstas	GET parametras	Užkuodoti URL (URL Encoding)
Tekstas	Nepatikima URL nuoroda	URL validacija, vengti „javascript:“, atributų užkodavimas, saugių URL verifikavimas
Tekstas	CSS	Griežtas struktūros tikrinimas, CSS „Hex“ reikšmių užkodavimas
HTML	HTML žymoje <body>	HTML validacija (naudojant įrankius „JSoup“, „AntiSamy“, „HTML Sanitizer“)
Bet koks	DOM	Remtis „DOM XSS Cheat Sheet“
Nepatikimas „Javascript“	Bet kur	Taikyti smėlio dėžės režimą
JSON	Kliento pusės JSON nuskaitymas	Naudoti metodus JSON.parse() arba json2.js

„XSS (Cross Site Scripting) Prevention Cheat Sheet“ [4] yra paprasčiausias sąrašas pateikiamas OWASP svetainėje. Jame yra sąrašas geriausių praktikų leidžiančių užtikrinti svetainės apsaugą nuo XSS. Teigiama, kad nors ir yra didžiuliai sąrašai su pažeidiamumais, gali pakakti tiesiog laikytis šių taisyklių ir taip apsaugoti svetainę nuo rimto pavojaus.

Straipsnyje „Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5“ aptariama keletas naujų pažeidžiamumų, kurie atsirado kartu su naujais HTML5 elementais. „Per pastaruosius metus naršyklės adaptavosi prie HTML5. Deja tai leido ne tik padidinti interaktyvumo galimybes, tačiau kartu atvėrė naujas galimybes XSS atakoms. Šiame dokumente aptariama 14 XSS vektorių, susijusių su HTML5. Remiantis šiais vektoriais buvo sukurta XSS vektorių repozitorija ir dinaminio testavimo įrankis skirtas el. pašto sistemoms. Taikant įrankį kai kurioms populiarioms el. pašto sistemoms, buvo rastos septynios XSS spragos.“ (13). Straipsnyje paminėtos naujos XSS spragos pateiktos 4 lentelėje.

[4] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

1.3 lentelė XSS pažeidžiamumai atsiradę su HTML5

Kategorija	Nr.	Atakos vektorius	Funkcija
Žymos	1.	<EMBED SRC="" type="image/svg+xml" AllowScriptAccess="always"></EMBED>	Pateiktame pavyzdyje žymoje „Embeded“ pateiktas SVG paveikslėlis, užkoduotas Base64 koduote, kuriame yra šis kodas: *<script type="text/ecmascript" > alert("XSS"); </script>*
	2.	<video onerror="alert(1)"><source></source></video>	Jei tinklalapyje galima įdėti savo vaizdo klipus, šią galimybę galima išnaudoti panašiai kaip paveikslėlius. Nepavykus užkrauti klipus bus iškviečiama funkcija „onererror“
	3.	<video src="#" onerror=alert(1)>	
	4.	<video><source onerror="alert(1)">	
	5.	<audio onerror=”alert(1)”><source></source></audio>	Kaip ir vaizdo klipo pavyzdyje taip ir žymai „audio“ galioja „onerrror“ atributas.
	6.	<audio src="#" onerror=”alert(1)”></audio>	
	7.	<audio><source onerror="alert(1)"></audio>	
Atributai	8.	<body oninput=alert(1)><input autofocus>	Įvesties laukų atributai „oninput“, „onfocus“, „onblur“ yra įvykdomi automatiškai, dėl atributo „autofocus“
	9.	<input onfocus=alert(1) autofocus>	
	10.	<input onblur=alert(1) autofocus><input autofocus>	
	11.	<form><button formaction="JavaScript:alert(1)">X</button>	Mygtuko žyma turi atributą „formaction“, kuris leidžia pateikti „Javascript“ kodą, kuris bus įvykdytas paspaudus mygtuką.
SVG	12.	<svg xmlns="http://www.w3.org/2000/svg"><g onload="JavaScript:alert(1)"></g></svg>	Už žymų <g> arba <svg> galima naudoti atributą „onload“
	13.	<svg onload="JavaScript:alert(1)" xmlns="http://www.w3.org/2000/svg"></svg>	
	14.	<svg xmlns="http://www.w3.org/2000/svg"><a xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="JavaScript:alert(1)"><rect width="1000" height="1000" fill="white"/></svg>	Suformuojamas 1000 pikselių aukščio ir pločio paveikslėlis, kurį paspaudus įvykdomas „Javascript“ kodas.

Dalis naujų XSS pažeidžiamumų yra analogiški „img“ žymos pažeidžiamumams, t. y. įterpiant neteisingą nuorodą į vaizdo ar garso įrašą galima priversti naršyklę įvykdyti klaidos atveju vykdomą

kodą. Kiti pažeidžiamumai atsiranda neapsaugant „Base64“ koduote užkuoduoto turinio ar nepatikrinant žymų atributų, kuriuose galima įterpti vykdomą kodą.

1.5.1. Esamų techninių sprendimų analizė

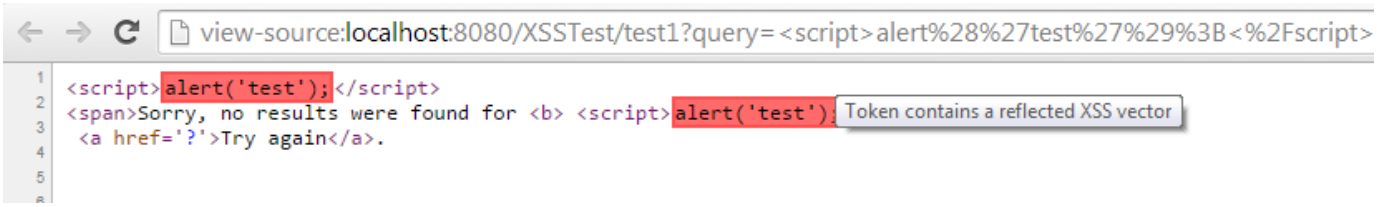
Įrankių analizė paremta Google sukurtu XSS pažeidžiamumų paieškos žaidimu[5]. Originalūs puslapiai parašyti Python programavimo kalba. Šiam testui, pagal tas pačias pažeidžiamumų idėjas parašyta Java internetinė aplikacija. Tiriami tokie pažeidžiamumai:

1. **Neapsaugotas įvesties laukelis**. Paprasčiausias XSS pažeidžiamumas, kai svetainė nuskaito ir atvaizduoja vartotojo pateiktą įvestį nepatikrinusi ar joje nėra pavojingo kodo. Toks pažeidžiamumas priskiriamas vienkartiniam XSS pažeidžiamumui, kadangi ši informacija nėra saugoma ir spragą galima išnaudoti tik pateikiant tam tikrą užklausą. Nors ir šis pažeidžiamumas yra labai pavojingas, populiariausios naršyklės apsaugo vartotoją nuo jo, ignoruodamos „Javascript“ scenarijaus kodą, kuris buvo pateiktas per užklausą (įspėjimas parodytas 1.6 pav.).

   Pažeidžiamumas išnaudojamas į paieškos laukelį įvedus „<script>alert('xss')</script>“

   Arba naršyklėje atlikus užklausą šiuo adresu:

   http://localhost:8080/XSSTest/test1?query=%3Cscript%3Ealert%28%27xss%27%29%3C%2Fscript%3E



1.6 pav. Naršyklės Chrome automatinė apsauga nuo vienkartinės XSS atakos

2. Spraga įterpiant paveikslėlį. Antrajame teste pateikta spraga leidžia užpuolikui įgyvendinti nuolatinę XSS ataką. Pateiktas puslapis apsaugotas nuo žymų „<script>“ panaudojimo, tačiau puslapis neapsaugotas nuo paveikslėlių atributo „onerror“. Šis atributas leidžia nustatyti funkciją, kuri bus įvykdyta, jei nepavyks užkrauti pateikto paveikslėlio.

   Vienas iš pažeidžiamumo išnaudojimo pavyzdžių yra į komentaro lauką įvedamas tekstas „<img src='none' onerror='alert("xss")'>“. Puslapiui užkrovus komentarus bus parodytas iššokantis pranešimas.

3. Neapsaugotas elementų formavimas „Javascript“ programiniame kode. Šiame pavyzdyje iš užklausos nuskaitomas nuotraukos numeris ir suformuojamas nuotraukos elementas. Kadangi paveikslėlio elementas formuojamas naudojant

[5] https://xss-game.appspot.com/

„Javascript“ kodą, užpuolikas gali išanalizuoti tai ir panaudoti prieš auką panašią techniką, kaip ir 2 teste.

4. Nesaugus „Javascript“ funkcijų kvietimas. Kai naršyklė nuskaito žymos atributus, jie automatiškai dešifruojami iš HTML. Vadinasi į užklausą pateikus užšifruotą parametrą, jis automatiškai bus dešifruotas. Taip galima užbaigti parametro kvietimo kodą ir įvykdyti savo kodą.
5. Žymos „a“ href atributo išnaudojimas. Norint atlikti veiksmą kai vartotojas paspaudžia ant nuorodos galima ne tik su atributu „onclick“. „Javascript“ funkcijų kvietimą galima įgyvendinti ir naudojant atributą „href“, parašius jame „javascript:alert('xss')“.

Iš pradžių pateikiama trumpa kiekvieno analizuojamo įrankio apžvalga, o pabaigoje pateikiamos bendros išvados.

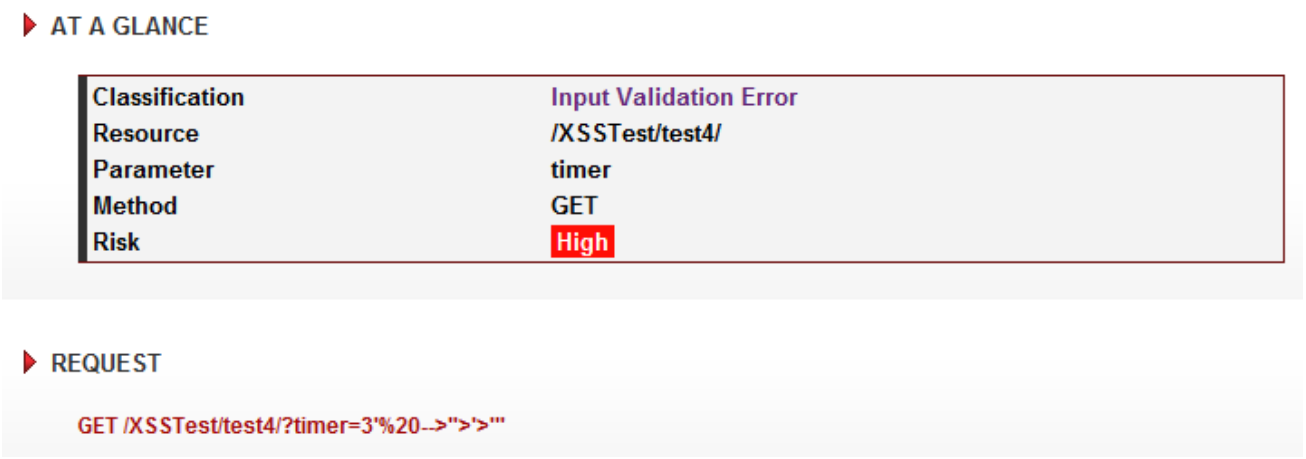
Acunetix[6] siūlo įrankį, kuris leidžia skenuoti tinklalapius ir ieškoti XSS pažeidžiamumų, tačiau šis įrankis yra mokamas. Taip pat yra siūloma realaus laiko paslauga, tačiau ji tinka tik svetainėms parašytomis naudojant PHP ir .NET technologijas. Bandomoji „Acunetix“ versija leidžia pažiūrėti tik pažeidžiamumų kiekį, tačiau ne vietą kurioje jis rastas.

Nemokamas įrankis „Wapiti“[7] atlieka juodosios dėžės skenavimą. T. y. neanalizuoja programinio svetainės kodo serveryje, tačiau analizuoja pačią svetainę ieškodamas scenarijų ir formų į kurias galima įterpti kodą. Įrankis neturi grafinės vartotojo sąsajos, tačiau pateikia pavyzdžius į konkrečias vietas, kuriose rasti pažeidžiamumai (Kaip parodyta 1.7 pav.)

1.7 pav. XSS pažeidžiamumų paieškos įrankis Wapiti

„Java“ programavimo kalba parašytas įrankis „Vega“[8] siūlo panašias skenavimo galimybes kaip ir „Wapiti“, tačiau papildomai turi grafinę vartotojo sąsają, kurioje aprašyti rasti pažeidžiamumai (1.8 pav.).

[6] http://www.acunetix.com/vulnerability-scanner/
[7] http://wapiti.sourceforge.net/
[8] https://subgraph.com/vega/

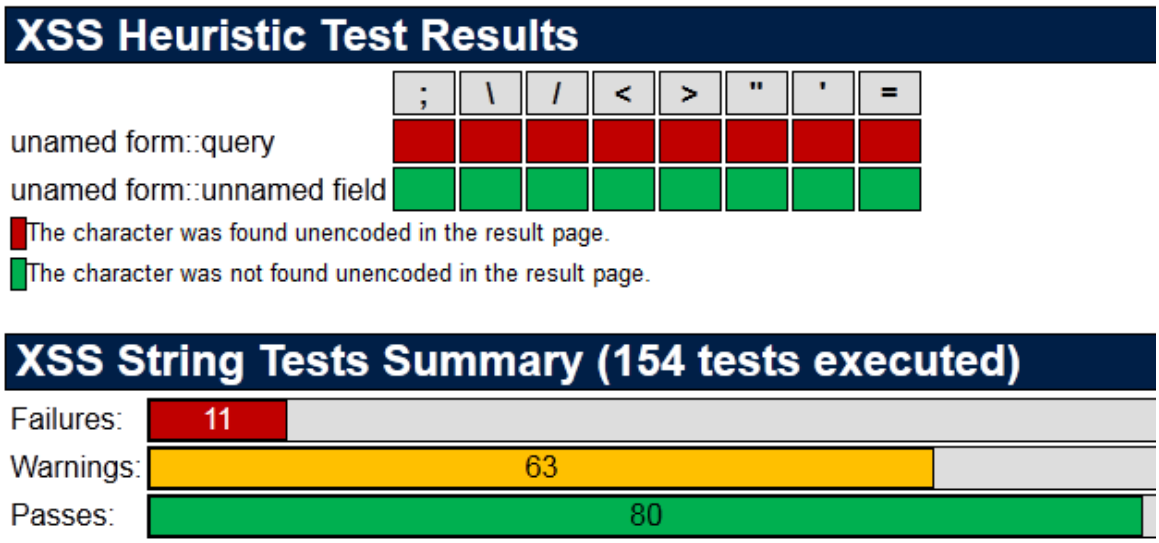


1.8 pav. XSS pažeidžiamumų paieškos įrankis Vega

Komandinės eilutės įrankis „Grabber“[9] pirmiausiai siūlo SQL įterpimo spragų paiešką, o XSS pažeidžiamumų paieška yra tik pradinėje stadijoje. Įrankis nėra patogus, nes neturi funkcijos apeiti visas svetainėje esančias nuorodas. T. y. norint ištestuoti 5 puslapius reikėjo leisti testą 5 kartus vis nurodant skirtingą puslapio adresą.

Naršyklės „Firefox“ įskiepis „XSS Me“[10] skirtas vieninteliam tikslui – tikrinti ar HTML formose nėra XSS pažeidžiamumų. „XSS Me“ atlieka dviejų tipų testus – specialių simbolių ir atakų vektorių sąrašo (pavyzdys pateiktas 1.9 pav. Formų XSS pažeidžiamumų paieškos įrankio XSS Me rezultatų ataskaita). Pirmasis tikrina ar formoje pateiktas specialusis simbolis atfiltruojamas. Antrasis naudoja pavyzdinių atakų sąrašą su „Javascript“ kodu ir tikrina ar tas kodas buvo įvykdytas. Jis neskenuoja srauto ir neranda užklausos parametrų, kuriuos galima išnaudoti, todėl juo pavyko patikrinti tik 3 iš 5 testų. Taip pat įrankis automatiškai netikrina visų nuorodų, kiekvieną formą reikia nurodyti ranka.

[9] http://rgaucher.info/beta/grabber/
[10] https://addons.mozilla.org/en-US/firefox/addon/xss-me/



1.9 pav. Formų XSS pažeidžiamumų paieškos įrankio XSS Me rezultatų ataskaita

Įrankio „Netsparker“[11] kūrėjai teigia, kad šis įrankis visada pateikia teisingus rezultatus (angl. *false-positive-free*). Ši programa turi dvi versijas – nemokamą, bendruomenės versiją (angl. *Community Edition*) ir mokamą. Programa taip pat tikrina SQL įterpimo atakas, palaiko kelias lygiagrečias užklausas ir po rezultatų ataskaitoje pateikia pavyzdžius su užklausomis, kuriose rasti pažeidžiamumai ir net pasiūlymus kaip ištaisyti rastas klaidas.

Nei vienas iš automatizuotų skenavimo įrankių nesugebėjo aptikti visų pavyzdinėje svetainėje pateiktų pažeidžiamumų. Rezultatai pateikti 5 lentelėje. Tai reiškia, kad norint užtikrinti savo svetainės saugumą nuo XSS atakų vien praskenuojant jį tam tikru įrankiu nepakanka.

Vis dėlto galima išskirti įrankį „Netsparker“, buvo naudota tik nemokama įrankio versija tačiau ji aptiko net nuorodos atributo href pažeidžiamumą ir sugebėjo rasti 3 iš 5 XSS spragų.

1.4 lentelė XSS skenavimo įrankių aptikti pažeidžiamumai pavyzdinėje svetainėje

Įrankis	**Aptikta pažeidžiamumų (viso yra 5)**
Acunetix	2
Wapiti	1
Vega	2
Grabber	0
XSS Me	1
Netsparker (Community Edition)	3

1.5.2. Rankinės pažeidžiamumų paieškos įrankiai

Šiame skyriuje apžvelgiami įrankiai kuriems neužtenka nurodyti savo svetainės adreso ir sulaukti analizės rezultatų, tačiau reikia pačiam analizuoti pateiktą srautą.

[11] https://www.netsparker.com/communityedition/

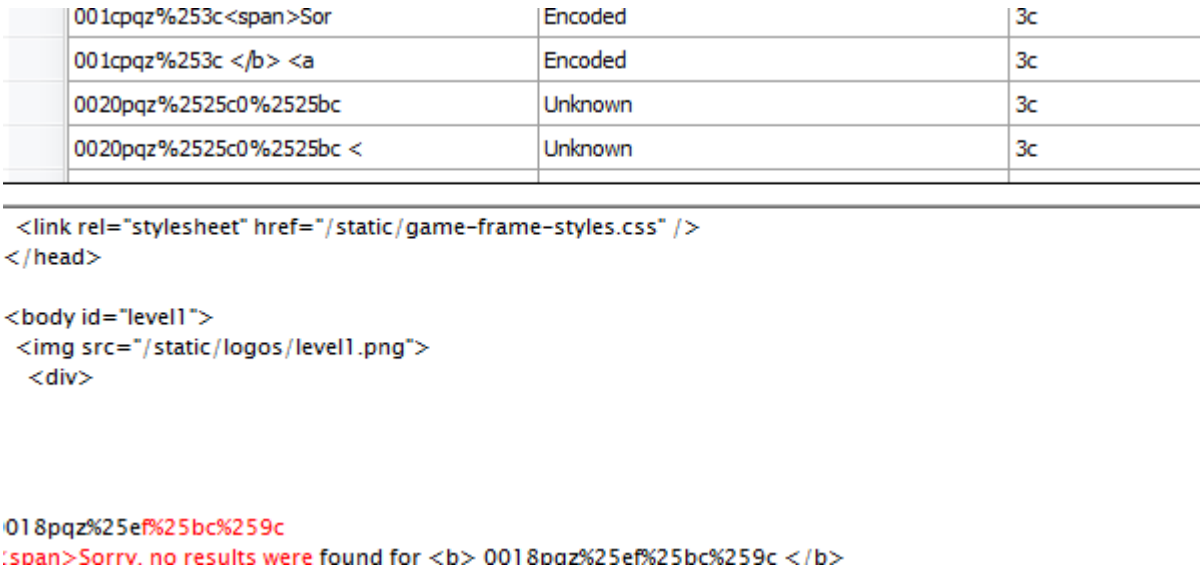
Pirmasis bandytas įrankis yra programos „Fiddler“[12] įskiepis „x5s“[13]. Jis padeda surasti su užkodavimu susijusių XSS spragas. Teigiama, kad gali būti aptiktos tokios spragos:

- Kur saugus užkodavimas nebuvo panaudotas tikrinant vartotojo įvestis
- Kur unikodo simbolių transformacijos gali padėti apeiti apsaugos filtrus
- Kur ne trumpiausi UTF-8 užkodavimai leidžia apeiti apsaugos filtrus (pavyzdžiui, įprastas simbolio „<“ kodas yra 0x3C, tačiau galima naudoti ir 0xC0 arba 0xBC)

Įrankis geriausiai pritaikytas nenuolatinių XSS spragų paieškai. Vartotojas turi pats naršyti pasirinktą svetainę, o šis įskiepis automatiškai ieškos užklausos parametrų ir juos testuos. Bandoma įterpti įvairius specialus simbolius ir tada tikrinama ar serverio pateiktame atsake šie simboliai vaizduojami, t. y. ar serveris sugebėjo atpažinti pavojingus simbolius ir juos išfiltravo. Programa nebando atlikti XSS atakų ir nepateikia pažeidžiamumo sąrašo, tačiau padeda vartotojui surasti vietas į kurias reikia atkreipti dėmesį.

Bandant įrankį su pavyzdine svetaine jis aptiko 3 puslapius į kuriuos reikėtų atkreipti dėmesį, t. y. kurie teisingai neišfiltruoja pavojingų simbolių (įspėjimo pavyzdys pateiktas 1.10 pav.). Kadangi įrankis analizuoja užklausas į serverį, jis nerado 2 iš 5 pažeidžiamų puslapių, nes dviejuose puslapiuose yra pateiktos „Javascript“ XSS spragos, kurioms užklausos į serverį daryti nereikia.

Įrankis tinkamas paprastose svetainėse, kuriose nėra laukų priimančių simbolius su specialiais simboliais. Jei svetainė pakankamai sudėtinga ir joje yra įvesties laukų, kuriuose galima įvesti specialius simbolius, o XSS apsauga įgyvendinta kitais metodais, šis įrankis pateiks daug neteisingų įspėjimų (angl. *false positive*).



1.10 pav. Srauto skenavimo įrankio x5s rastas įspėjimas

[12] http://www.telerik.com/fiddler
[13] http://xss.codeplex.com/

Rankiniam testavimui yra tinkamas įrankis „Burp Suite“, jis veikia kaip tarpininkas (angl. *proxy*) tarp vartotojo ir serverio. Programos pagalba galima stebėt į serverį siunčiamą ir iš serverio gaunamą informaciją. Knygoje „XSS Attacks Cross Site Scripting Exploits and Defense“, teigiama, kad „Burp yra vienas iš naudingiausių internetinių svetainių saugumo įrankių rankiniam testavimui. Jis ne tik leidžia atrasti akivaizdžias spragas, tačiau leidžia rašyti savo taisykles, jei jūs žinote ko ieškote. Pavyzdžiui, jei norėtumėte surasti tik XML failus AJAX naudojančių svetainių tikrinimui. Burp programoje galima sukurti taisyklę leidžiančią aptikti tik šią informaciją“ (7)

1.5.3. Apsaugos priemonės

Vienas iš paprastesnių apsaugos nuo XSS atakų metodų pateiktas tinklalapyje javacodegeeks.com. Tam sukuriamas papildomas „Java“ „Servlet“ filtras, kuris turėtų būti vykdomas pirmas filtrų grandinėje. Šis filtras apgaubia HTTP užklausą ir pašalina joje esančius keistus parametrus. (14) Programos kodas gana paprastas, tačiau tam nėra sukurto komponento, tai reiškia, kad reikėtų kopijuoti kodą, užuot panaudojus komponentą. Taip pat reikėtų atsižvelgti į tai, kad naujausią šio sprendimo versija[14] buvo pateikta 2012 metais. Atlikus analizę pastebėta, kad šį filtrą apeiti nėra sudėtinga, apačioje pateikiama keletas atakų pavyzdžių, kurių neatpažintų filtras:

1. <img src="a" onerror="alert('XSS spraga')">
2. <scr<script>ipt>alert('XSS spraga')</scr<script>ipt>
3. XSS

Taip pat pateiktas filtras neatmeta pavojingos užklausos, o bando išfiltruoti pavojingą turinį. Filtro saugumą būtų galima padidinti tiesiog atmetant užklausas, kuriose yra XSS atakos kodas.

Sudėtingesnis sprendimas yra biblioteka „xssprotect“[15]. Ši biblioteka sukuria HTML žymų medį, kuris yra šiek tiek optimizuojamas geresnei greitaveikai. Tada kviečiamas filtras, kuris atsakingas už netinkamų HTML atributų ar XSS atakų filtravimą. Filtras paremtas OWASP pažeidžiamumų sąrašu. Biblioteką taip pat galima naudoti HTML žymų medžiui kurti ir redaguoti. Tačiau projektas paskutinį kartą atnaujintas 2008 metais ir nebeapsaugo nuo naujausių XSS spragų.

Įrankis „Owasp Java Html Sanitizer“ padeda sudaryti leidžiamų naudoti HTML elementų sąrašą. Jei kuriamoje svetainėje reikia leisti vartotojui pateikti HTML kodą, šis įrankis leidžia užtikrinti, kad vartotojas nepateiks nesaugių elementų. Analogiškas taip pat „Java“ programavimo kalbai skirtas įrankis yra „Jsoup“. Pagrindinė šio įrankio funkcija yra nuskaityti HTML turinį ir suteikti struktūrizuota prieigą prie jo.

Struktūrizuotas turinio nuskaitymas yra tinkamas būdas apsisaugoti nuo XSS atakų, kadangi galima tiksliai žinoti kokie elementai sudaro turinį ir kokius atributus jie turi. Tokie įrankiai įrankiai kaip „Jsoup“ pasirūpina, kad HTML turinys būtų teisingos struktūros ir tuo nebūtų galima

[14] http://ricardozuasti.com/2012/stronger-anti-cross-site-scripting-xss-filter-for-java-web-apps/
[15] https://code.google.com/p/xssprotect/

pasinaudoti piktavaliams. Pavyzdžiui HTML tekstas „<<*p*></*p*><*p*>“ yra neteisingas dėl dviejų priežasčių: pradžioje teksto yra specialusis simbolis „<“ neturintis elemento pavadinimo ir uždarymo simbolio „>“, taip pat antrasis elementas „ “ neturi uždarymo elemento „ “. Tiek „Owasp Html Sanitizer“, tiek „Jsoup“ interpretuoja šį tekstą kaip HTML turinį su dviem pastraipomis ir papildomą specialųjų simbolį „<“ paverčia į paprastą tekstą. Visada turint teisingą HTML struktūrą programuotojui belieka sudaryti teisingą taisyklių sąrašą su leidžiamais elementais ir leidžiamais atributais.

Galima vadovautis saityno konsorciumo (angl. *World Wide Web Consortium*) HTML validatoriuje naudojamų saugių HTML atributų sąrašu ir leisti naudoti tik šiuos HTML atributus: abbr, accept, accept-charset, accesskey, action, align, alt, axis, border, cellpadding, cellspacing, char, charoff, charset, checked, cite, class, clear, cols, colspan, color, compact, coords, datetime, dir, disabled, enctype, for, frame, headers, height, href, hreflang, hspace, id, ismap, label, lang, longdesc, maxlength, media, method, multiple, name, nohref, noshade, nowrap, prompt, readonly, rel, rev, rows, rowspan, rules, scope, selected, shape, size, span, src, start, summary, tabindex, target, title, type, usemap, valign, value, vspace ir width. (15)

1.5.4. Kitos priemonės

Remiantis tinklalapiu stackoverflow.com, „Spring“ yra populiariausias „Java“ karkasas tinkamas kurti internetinius tinklalapius (nėra kitos žymos susijusios su tinklalapių kūrimu naudojant „Java“ nei „spring“[16]). Dėl to šiame darbe bus nagrinėjami metodai leidžiantys palengvinti apsaugos nuo XSS užtikrinimą naudojant „Spring MVC“ karkasą.

„Spring“ karkasas savaime turi vienintelę apsaugą nuo XSS atakų – turinio užkodavimą. T.y. vartotojui įvedus „ “ šis tekstas bus paverstas į „<div></div>“. Atlikus tokį pakeitimą specialieji simboliai yra interpetuojami kaip parastas tekstas ir tiesiog atvaizduojami vartotojui.

1.6. Išvados

1. Atlikus XSS pažeidžiamumų analizę nustatyta, kad nors XSS nėra pats pavojingiausias svetainės pažeidžiamumas, nuo jo apsaugoti svetainę būtina. Jis yra vienas iš dažniausiai pasitaikančių pažeidžiamumų svetainėse.
2. Buvo išanalizuoti nauji HTML5 elementai ir surasti nauji XSS pažeidžiamumai. Nauji XSS pažeidžiamumai atsirado ne tik dėl naujų elementų, tačiau ir dėl naujų atributų ir senų elementų naujų savybių.

[16] http://stackoverflow.com/tags?page=2&tab=popular

3. Išanalizavus mokslinę ir metodinę literatūrą pavyko išsiaiškinti, jog nauji XSS pažeidžiamumai atsiranda pakankamai dažnai, todėl apsisaugojimo metodai geriausiai aprašyti metodiniuose straipsniuose ir žiniaraščio įrašuose, o ne knygose.
4. Buvo ištirti 6 analizės įrankiai, tačiau nei vienas iš jų nesugebėjo surasti visų pažeidžiamumų pateiktų pavyzdinėje svetainėje. Taip pat nei vienas analizės įrankis nesugebėjo rasti dokumentų objektų modeliu paremtų pažeidžiamumų.
5. „Java“ platformoje yra įrankių galinčių tiek išvalyti įvestą turinį, tiek apsaugoti grąžinamą turinį vartotojui, tačiau jei spraga yra kliento pusėje, šie įrankiai nepadės.
6. Yra įvairių įrankių skirtų „Java“ programavimo kalbai, tačiau jie ne visada yra patogūs naudoti, todėl reikalingi jų patobulinimai.

2. ESAMŲ SPRENDIMŲ PATOBULINIMO APRAŠYMAS

Probleminės srities analizė parodė, kad naujos XSS spragos atsiranda nuolat. Taip pat tai, kad XSS pažeidžiamumai stipriai priklauso nuo konteksto, t.y. vienas metodas gali pilnai apsaugoti tam tikrą puslapio dalį, tačiau kitoje dalyje to paties metodo gali neužtekti. Tai reiškia, kad sunku sukurti įrankį, kuris užtikrintų visišką apsaugą nuo XSS atakų. Dėl to svarbu turėti ne tik greitai ir stabiliai veikiančias apsaugos priemones, tačiau turėti lengvai palaikomą programinį kodą. Tai leistų didelėje sistemoje greitai pakeisti saugumo taisykles ir patikimai ištaisyti naujai atsiradusias saugumo spragas.

Kaip buvo minėta analizės dalyje, 1.5.4 skyriuje, internetinis tinklalapis bus kuriamas naudojant „Java“ programavimo kalbą ir „Spring“ karkasą.

2.1. Esami sprendimai

Skyriuje apžvelgiami jau įgyvendinti sprendimai naudojant „Java“ programavimo kalbą ir „Spring“ karkasą leidžiantys apsisaugoti nuo XSS atakų. Tam tikri saugos sprendimai gali būti įgyvendinami vien naudojant teisingus duomenų tipus, kiti sprendimai yra įgyvendinti kaip numatytieji įrankių nustatymai, o kitus sprendimus reikia konfigūruoti pačiam.

2.1.1. „Spring“ automatinė apsauga nuo XSS

Yra keletas atvejų, kai dėl „Java“ kalbos ir „Spring“ karkaso savybių, papildomų priemonių dėl apsaugos nuo XSS imtis nereikia. Šiame skyriuje minimi šie atvejai.

1. Stipriai tipizuoti laukai. „Java“ programavimo kalboje yra griežtai užtikrinami kintamųjų tipai. Tai reiškia, kad turint kintamąjį, kuris saugo sveikąjį skaičių, jam niekad nebus įmanoma priskirti tekstinės reikšmės. Pavyzdžiui turint metodą, kuris iš vartotojo priima tam tikrą sveikąjį skaičių ir atvaizduoja jį puslapyje, vartotojas niekada negalės pateikti paslėpto „Javascript“ kodo, tikėdamasis, kad jis bus įvykdytas atvaizduotame puslapyje. Kodo pavyzdžiai pateikti 2.1 pav., 2.2 pav. ir 2.3 pav.

```
@RequestMapping("/owners/{ownerId}")
public class TestController {

 @RequestMapping("/pets/{petId}")
 public void findPet(@PathVariable Long ownerId,
                     @PathVariable Long petId,
                     Model model) {
        // implementation omitted
    }
}
```

2.1 pav. „Spring MVC“ metodas priimantis tik skaitinius adreso parametrus

```
@RequestMapping("/pets")
public class TestController {
```

```
    @RequestMapping("/")
    public void list(@QueryParam Integer page,
                     @QueryParam Integer count,
                     Model model) {
        // implementation omitted
    }
}
```

2.2 pav. „Spring MVC“ metodas priimantis tik skaitinius užklausos parametrus

```
@RequestMapping("/petTypes")
public class TestController {

    @RequestMapping(value="/save", method=RequestMethod.POST)
    public void list((@ModelAttribute PetType pet,
                     Model model) {
        // implementation omitted
    }
}
```

2.3 pav. „Spring MVC“ metodas priimantis „PetType“ tipo formos parametrą

2. HTML žymų užkodavimas. „Spring“ karkasą galima sukonfigūruoti taip, kad visi užklausų parametrai būtų užkoduoti, taip, kad juose nebūtų HTML žymų ir naršyklės turinį interpretuotų kaip paprastą tekstą. Taip ne visada užtikrina pilną svetainės apsaugą nuo XSS ir kartais gali reikėti atvaizduoti vartotojo pateiktą tekstą kaip HTML, tačiau kai kuriais atvejais tokios apsaugos gali pakakti. Norint įjungti kodavimą visiems parametrams visoje sistemoje reikia naudoti 2.4 pav. parodytą konfigūraciją. Taip pat galima nustatyti kodavimą vienam puslapiui arba vienai formai.

```
<context-param>
    <param-name>defaultHtmlEscape</param-name>
    <param-value>true</param-value>
</context-param>
```

2.4 pav. Visų HTML simbolių užkodavimo konfigūracija

2.1.2. Anotacijos „Java“ programavimo kalboje

„Java“ anotacijos yra kodo žymėjimo būdas leidžiantis pažymėti klases, metodus, metodų parametrus. Šis žymėjimas tiesiogiai nekeičia programinio kodo vykdymo, tačiau programos vykdymo metu anotacijos yra išsaugomos ir pasiekiamos. Tai reiškia, kad galima parašyti programos veikimą išplečiantį kodą, kuris pasiekdamas anotacijų informaciją galėtų įtakoti programos vykdymą.

Anotacijos gali būti naudojamos šiose situacijose:

- Nurodyti informaciją kompiliatoriui – anotacijos leidžia aptikti kompiliavimo klaidas arba paslėpti kompiliavimo įspėjimus. Pavyzdžiui, anotacija „@Override“

nurodo, jog metodas perrašo paveldėtos klasės metodą. Jeigu tokio metodo paveldėtoje klasėje nėra, kompiliatorius parodo klaidą.

- Nurodyti informaciją kodo apdorojimui kompiliavimo ir diegimo metu – anotacijas galima naudoti programinio kodo generavimui, XML kodo generavimui ir pan.
- Nurodyti informaciją programos veikimo metu – anotacijos kurios bus naudojamos šio darbo metu. Šios anotacijos yra tikrinamos programos veikimo būdu ir gali, pavyzdžiui, tikrinti formos įvesties laukus.

Anotacijos žymimos simboliu „@“ ir anotacijos pavadinimu. Anotacija gali turėti elementus, pavyzdžiui:

```
@Author(
         name = "Benjamin Franklin",
         date = "3/27/2003"
)
```

2.5 pav. Anotacijos pavyzdys

Jei anotacija turi tik vieną elementą, jo vardo žymėti nebūtina, pavyzdžiui „@SuppressWarnings("unchecked")“. Anotacijas galima taikyti klasėms, kintamiesiems, metodams ir įvairiems tipų panaudojimams, pavyzdžiui tipų konvertavime.

„Spring“ karkase anotacijos naudojamos tiek HTTP parametrų žymėjimui, tiek saugos aprašymui, tiek konfigūracijai. 4-tojo „Spring“ karkaso versijoje galima visiškai atsisakyti XML konfigūracijos ir naudoti vien klases su anotacijomis.

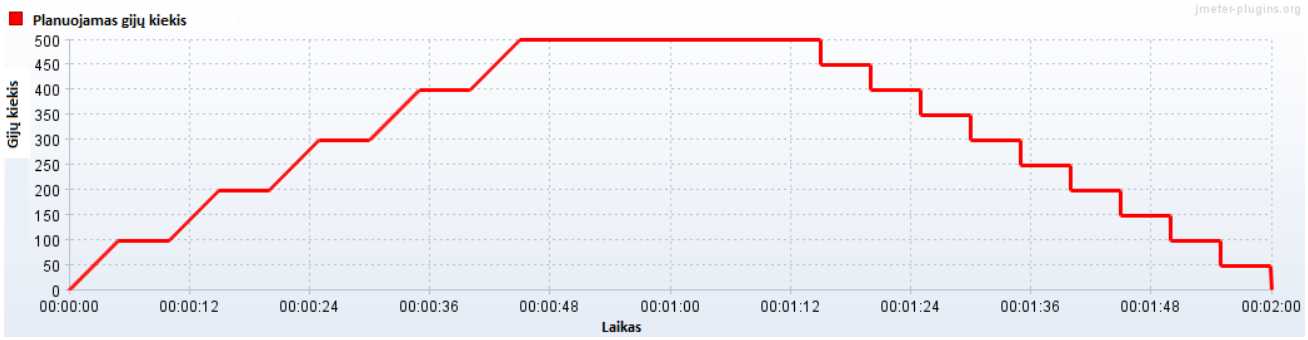
Vis dėlto „Spring“ neturi pakankamai anotacijų leidžiančių valdyti leidžiamą klientams pateikti HTML kodą per HTTP užklausas.

2.1.3. HTML turinio išvalymo įrankių palyginimas

Yra keletas įrankių skirtų HTML turinio išvalymui. Pagrindinis jų tikslas yra atpažinti pateikto teksto struktūrą, sudaryti elementų medį ir gebėti aptikti pavojingus medžio elementus ar medžio elementų atributus. Tyrime išbandyti 2 įrankiai : „Jsoup“ ir „Owasp HTML Sanitizer“.

Tyrimas atliktas naudojant „Apache Jmeter“ įrankį. Pasirinktas išorinis testavimas, o ne mikro testavimas (angl. *Micro benchmarking)*, kadangi taip ištestuojama sistema panašesnė į realią sistemą ir dėl įvairių Java programavimo kalbos optimizacijų atlikti tikslų mikro testavimą pakankamai sunku.

Tyrimo metu testuojama užklausų sparta apkrovus sistemą didelių vartotojų kiekiu. Atliekamas bandymas su 500 lygiagrečių vartotojų (t. y. vienu metu sistema peleidžia 500 gijų kurios kreipiasi į sistemą). Pradedama nuo 0 gijų, ir tada kas 5 sekundes pridedama po 100 gijų iki kol pasiekiama 500 gijų, tada tokia apkrova palaikoma 30 sekundžių. Vartotojų kiekis pavaizduotas 2.6 pav.



2.6 pav. Apkrovos testavimo vartotojų kiekio diagrama

Testavimo metu panaudoti kelios skirtingo ilgio užklausos ir matuojami 90% procentilio, 99% procentiliai bei įvykdytų užklausų kiekis per sekundę. Tam tikro procentilio reikšmė nurodo užklausos trukmę, už kurią visos likusios užklausos buvo trumpesnės, t. y. 50 milisekundžių 99% procentilis nurodo, kad 99 procentai užklausų buvo trumpesnės nei 50 milisekundžių.

Buvo paruoštos 2 užklausos testavimui. Viena 70 simbolių, o kita 3000 simbolių. Abiejose užklausose pateiktas HTML kodas su neleistinomis žymomis ar elementais. Testavimo rezultatai pateikti žemiau esančiose lentelėse.

2.1 lentelė 70 simbolių ilgio užklausos spartos rezultatai

Testavimo metodas	90% procentilis	99% procentilis	Užklausų per sekundę
Be apsaugos	72	145	3268.456
Jsoup	139	206	3061.975
Owasp HTML Sanitizer	127	200	3257.886

2.2 lentelė 3000 simbolių ilgio užklausos spartos rezultatai

Testavimo metodas	90% procentilis	99% procentilis	Užklausų per sekundę
Be apsaugos	195	331	2129.72
Jsoup	153	323	1942.33
Owasp HTML Sanitizer	163	315	1923.75

Iš testavimo rezultatų matyti apylygiai rezultatai. Nei vienas įrankis nebuvo žymiai greitesnis už kitą, taip pat verta paminėti, kad rezultatai nenaudojant jokios apsaugos, t. y. tiesiog nuskaitant turinį ir gražinant jį vartotojui, taip pat nebuvo žymiai greitesni nei apsaugoti variantai. Tai reiškia, kad gerai apsaugotas nuo XSS atakų tinklalapis nebūtinai bus žymiai lėtesnis nei visai neapsaugotas.

Kadangi nebuvo žymaus lyderio lyginant spartą, palyginimui buvo pasirinktas įrankis „Owasp HTML Sanitizer“, kadangi šis įrankis yra tiesiogiai skirtas turinio valymui ir turi daugiau galimybių nei „Jsoup“, kuris visų pirma yra HTML turinio analizatorius tik papildomai teikiantis galimybes išvalyti turinį nuo pavojingo kodo.

2.1.4. Html Sanitizer konfigūravimo aprašymas

Patobulinimui pasirinktas OWASP įrankis „Java“ HTML Sanitizer. Kaip buvo minėta analizės dalyje, jis leidžia nurodyti leidžiamų HTML žymų ir atributų sąrašą. Galima kurti bendrus

tokių elementų sąrašus ir pernaudoti juos keliose vietose. Pavyzdžiui, kartu su įrankiu yra pateikiamas ebay.com svetainės saugumo politiką emuliuojantis taisyklių rinkinys. „Ebay“ svetainėje vartotojai gali pateikti turinį su įvairiomis HTML žymomis, todėl tai geras pavyzdys kaip galima užtikrinti saugumą leidžiant įvesti ne tik paprastą tekstą. 2.7 pav. pateiktas pavyzdys, kaip šiuo metu reikėtų inicijuoti HTML Sanitizer įrankį. Pateiktas metodas tiesiog atspausdina komentaro pavadinimą ir komentarą, prieš tai patikrindamas ar pavadinimą sudaro tik žodžiai ir jokių specialių simbolių, o komentare yra tik tekstas su žymomis „div“, „img“ ar „a“.

```
private static void currentMethod(String title,String commentWithLinks){
        PolicyFactory policyFactory = new HtmlPolicyBuilder().allowElements(
        "img","div","a").toFactory();
        title= title.replaceAll("[^\\p{L}\\p{Nd}]+","");
        commentWithLinks = policyFactory.sanitize(commentWithLinks);
        System.out.println(title);
        System.out.println(commentWithLinks);
        }
```

2.7 pav. OWASP „Java“ HTML Sanitizer inicijavimas

Kodas pateiktame pavyzdyje gana paprastas, tačiau sudėtingesnėje programoje su daugiau parametrų vien įrankio inicijavimas užimtų nemažai vietos ir paslėptų tikrąjį metodo funkcionalumą.

2.1.5. „Spring Boot“ automatinė konfigūracija

„Spring Boot“ yra „Spring“ karkaso patobulinimas leidžiantis patogiau integruoti įvairius komponentus. Jis palaiko automatinio konfigūravimo mechanizmą. Komponentų kūrėjai gali numatyti tam tikrus parametrus, kurių komponento vartotojui nereikia nurodyti iš karto.

Tai leidžia naudoti komponentus iš karto ir konfigūruoti tik tada, jei numatytasis konfigūravimas netinka. Šiame darbe „Spring Boot“ automatinė konfigūracija bus naudojama kaip sukurto įrankio plėtinys, tai leis iš karto naudoti sukurta įrankį, o ateityje atnaujinus konfigūracijos struktūrą, įrankio naudotojams nereikės daryti jokių papildomų pakeitimų, tik atnaujinti naudojamą versiją.

2.2. Patobulinimo aprašymas

Šiame darbe bus apjungiami 2.1 skyriuje aprašyti metodai, t. y. „Html Sanitizer“ įrankis bus integruotas į „Spring“ karkasą naudojant anotacijas. Atlikus spartos tyrimus nustatyta, kad įrankis programos darbą darbą sulėtina nežymiai. O XSS pažeidžiamumų analizės metu nustatyta, kad apsaugant tinklalapį nuo XSS atakų svarbu kontekstas, dėl to atliekamas patobulinimas leis patogiau konfigūruoti ir pritaikyti „Html Sanitize“ įrankį kuriant tinklalapius.

2.3. Html Sanitizer konfigūracijos patobulinimas

Bus sukurta OWASP „Java“ HTML įrankio integraciją su „Spring MVC“ karkasu, taip, kad HTTP užklausų parametrus būtų galima tiesiog pažymėti anotacijomis nurodančiomis saugumo

priemones. Tokio kodo pavyzdys pateikiamas 2.7 pav., pavyzdyje aiškiau matosi, kad programa tiesiog atspausdina komentaro pavadinimą ir tekstą, o apsaugos priemonės nurodomos anotacijomis „WordsOnly“ ir „SanitizeHtml“.

```
private static void improvedMethod(
        String @WordsOnly title,
        String @SanitizeHtml({"img", "div", "h1"}) commentWithLinks) {
        System.out.println(title);
        System.out.println(commentWithLinks);
        }
```

2.8 pav. Patobulintas „Java“ HTML Sanitizer įrankio naudojimas

Šie du metodai skiriasi vienu esminiu dalyku, viename nurodoma kaip atlikti tam tikrus veiksmus, o kitame nurodoma tik kokio rezultato tikimasi. Anotacijų naudojimą „Java“ programavimo kalboje galima vadinti deklaratyviuoju programavimu, kuris nuo imperatyvaus ir skiriasi tuo, kad nurodomas tik norimas rezultatas, o ne žingsniai kaip tą rezultatą pasiekti. (16)

2.3.1. Įvesties laukų tipai

Visus įvesties laukus galima skirstyti į tris kategorijas:

1. Skaitinius. Tai laukai, kurie naudojami įvesti tiesiog skaičiams, arba reikšmėms, kurias galima paversti į skaičius, pavyzdžiui, datą paverstą į laiką milisekundėmis nuo epochos pradžios. Kaip buvo minėta 2.1.1 skyriuje, tokie laukai yra automatiškai apsaugomi teisingai naudojant „Java“ ir „Spring“ karkasą.
2. Turinčius aiškiai apibrėžtą elementų aibę. Šio tipo įvesčiai galima priskirti įvairius sąrašus iš kurių vartotojui reikia pasirinkti vieną ar kelias reikšmes. Tam naudojami įvairūs HTML elementai. Apibrėžtos elementų aibės laukai taip pat gali būti lengvai apsaugoti. Tereikia sutikrinti ar pateikiami duomenys tikrai priklauso įrašų aibei.
3. Laisva forma įvedamo teksto laukai. Tai gali būti įvairiausio tipo turinys, pradedant paieškos frazes ir baigiant tekstu su aprašytu stiliumi ir formatavimo informacija. Tokius įvesties laukus apsaugoti sunkiausiai, todėl jie ir bus aptariami šiame skyriuje.

Galima įvairiai riboti tekstinių laukų leidžiamų simbolių sąrašą. Žemiau aptariami įvairūs ribojimo lygiai pradedant nuo didžiausio.

1. Tik raidės ir skaičiai. Labiausiai apribotas įvedamų duomenų lygis. Naudingas paprastuose paieškos laukeliuose, vardo ar pavardės įvedimo laukeliuose ir pan.
2. Tekstas. Toks apribojimas leidžia įvesti tam tikrus specialius simbolius, pvz. kablelius, taškus, dvitaškius ir pan. Tinkamas naudoti paprastų komentarų sistemose, atsiliepimuose ir pan.

3. Paprastas HTML. Kartais paprasto teksto neužtenka ir norima suteikti vartotojams tam tikras formatavimo galimybes. Toks ribojimas leistų paprastai suformatuoti tekstą nesuteikiant per daug teisių įterpti pavojingą kodą.
4. Sudėtingas HTML. Toliau skirstyti ribojimą į lygius pakankamai sunku, nes kiekvienu atveju gali reikėti skirtingų HTML žymų ar kitų galimybių, todėl šio lygio elementu teisingiausia leisti nurodyti pačiam programuotojui.

Pagal skirtingus ribojimo lygius buvo suformuotos įvesties ribojimo anotacijos. Lentelėje 2.3 lentelė pateikiamas planuojamų įgyvendinti anotacijų sąrašas.

2.3 lentelė Planuojamų įgyvendinti „Java“ anotacijų sąrašas

Anotacija	**Parametrai**	**Funcionalumas**
@WordsOnly	**whiteList** – parametras leidžia įterpti papildomus skyrybos ženklus ar kitus simbolius.	Išvalomi visi specialieji simboliai, paliekamos tik raidės ir skaičiai.
@TextOnly	**whiteList** – parametras leidžia įterpti papildomus skyrybos ženklus ar kitus simbolius.	Išvalomi visi simboliai išskyrus raides, skaičius ir tekste naudojamus skyrybos ženklus.
@SimpleHtml		Taisyklių rinkinys leidžiantis įtepti paprasčiausias HTML žymas, tokias kaip „div“, „p“ ir pan.
@HtmlSanitizePolicy	**policy** – taisyklių rinkinys	Leidžiama nurodyti taisyklių rinkinį, pagal kurį bus išvalomas HTML kodas
@CustomHtmlSanitize	**allowElements** – leidžiamų elementų sąrašas **allowStandardUrlProtocols** – leisti naudoti nuorodas su standartiniais protokolas (http, https, mailto) **allowStyling** – leisti naudoti tam tikrus saugius CSS elementus **requireRelNofollowOnLinks** – prie nurodų pridedamas „rel=nofollow“ atributas	

Viena iš anotacijų leidžia pateikti taisyklių rinkinį. Tai yra sudėtingesnis leidžiamų elementų aprašymas. „Owasp Html Sanitizer“ programiniame kode pateikiami keli tokių taisyklių rinkinių pavyzdžiai vienas iš jų pateikiamas pav.

```
public static final Function<HtmlStreamEventReceiver, HtmlSanitizer.Policy>
        POLICY_DEFINITION = new HtmlPolicyBuilder()
        .allowStandardUrlProtocols()
        // Allow title="..." on any element.
        .allowAttributes("title").globally()
        // Allow href="..." on <a> elements.
        .allowAttributes("href").onElements("a")
        // Defeat link spammers.
        .requireRelNofollowOnLinks()
        // Allow lang= with an alphabetic value on any element.
        .allowAttributes("lang").matching(Pattern.compile("[a-zA-Z]{2,20}"))
        .globally()
        // The align attribute on <p> elements can have any value below.
        .allowAttributes("align")
        .matching(true,"center","left","right","justify","char")
        .onElements("p")
        // These elements are allowed.
        .allowElements(
        "a","p","div","i","b","em","blockquote","tt","strong",
        "br","ul","ol","li")
        // Custom slashdot tags.
        // These could be rewritten in the sanitizer using an ElementPolicy.
        .allowElements("quote","ecode")
        .toFactory();
```

2.9 pav. Html Sanitizer taisyklių rinkinio pavyzdys

2.4. Saugaus HTML5 kodo taisyklių rinkinys

Kaip buvo minėta pirmame skyriuje, įrankis „OWASP Java HTML Sanitizer“ leidžia sudaryti taisyklių rinkinį išvalantį nesaugų HTML kodą palikdamas tik saugias žymas ir atributus. Realizacijos metu, pagal analizės dalyje sudaryta HTML5 XSS pažeidžiamumų sąrašą bus sudarytas taisyklių rinkinys leidžiantis vartotojams įterpti svetainėje HTML5 žymas, tačiau apsaugantis sistemą nuo šių žymų išnaudojimo XSS atakoms.

2.5. Vertinimo metodika

Įgyvendinto metodo testavimui ketinama sukurti panašų puslapį į analizės dalyje sukurtą tinklalapį, tačiau šį kartą koncentruojamasi į nuolatines ir vienkartines XSS atakas, kurias sukelia nesaugus serverio kodas, o ne nesaugus „Javascript“ kodas. Po 1.5.1 skyriuje atlikto tyrimo paaiškėjo, jog geriausias įrankis pažeidžiamumų analizei yra „Netsparker“, todėl šis įrankis ir bus naudojamas svetainės pažeidžiamumų tikrinimui.

Ketinamos naudoti technologijos:

- Windows 7 64-bit operacinė sistema
- Java 8 programavimo kalba
- „Spring 4“ karkasas
- H2 duomenų bazė
- Apache Tomcat 8.0

Testavimui skirta svetainė bus sudaryta iš įvairių formų su didėjančiu leidžiamų panaudoti HTML žymų kiekiu:

1. Puslapis, kuriame leidžiama įvesti tik tekstą
2. Puslapis, kuriame leidžiama paprastai formatuoti tekstą (išskirti skyrius, paryškinti tekstą ir pan.)
3. Puslapis kuriame leidžiama įterpti nuotraukas ir vaizdo įrašus
4. Puslapis, kuriame leidžiama pritaikyti stilių tam tikriems HTML elementams.

Kadangi anotacijų naudojimas gali veikti lėčiau nei įprasto imperatyvaus kodo rašymas, sukurta svetainė bus apsauga dviem būdais: naudojant įprasta HTML Sanitizer programą ir naudojant patobulintą šio įrankio versiją. Bus lyginama šie sistemų požymiai:

- Greitaveika (Taikomi apkrovos testai). Apkrovos testams bus naudojamas įrankis Apache JMeter.
- Bendras programinio kodo eilučių kiekis

2.6. Išvados

1. „Spring“ karkasas yra plačiausiai naudojamas „Java“ karkasas internetinių sistemų kūrimui. Vis dėlto, jis neturi pakankamai galimybių patogiai valdyti įvedamą turinį, kai svetainėse kuriose vartotojui leidžiama įvesti turinį turintį saugių HTML žymų.
2. Html Sanitizer įrankis leidžia konfigūruoti įvedamą HTML turinį, tačiau neturi integracijos su „Spring“ karkasu ir visas konfigūracijas reikia atlikti kode prie verslo logikos.
3. Kaip įrankio plėtinys bus sukurtas saugių žymų sąrašas, suteikiantis laisvę vartotojams įvesti HTML, tačiau apsaugantis sistemą nuo XSS atakų.
4. Atlikti patobulinimai bus vertinami greitaveikos atžvilgiu, taip pat tikrinama ar išlaikomas tas pats saugos lygis, kaip ir naudojant nepatobulintas įrankių versijas.

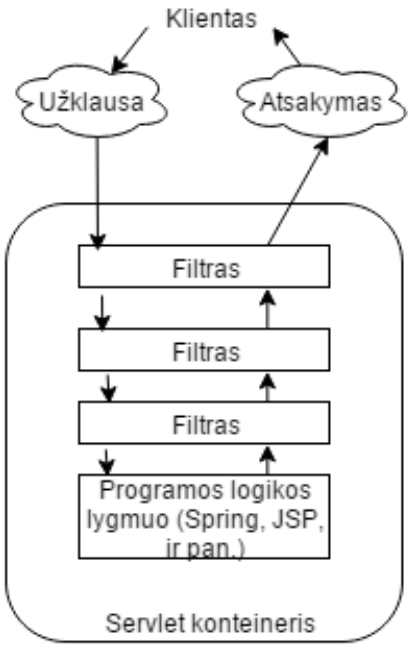
3. ĮRANKIŲ PATOBULINIMO ĮGYVENDINIMAS

Atlikus jau esamų sprendimų analizę nuspręsta įgyvendinti „Spring“ karkaso patobulinimą. Skyriuje aprašomos konkrečios detalės apie patobulinimo įgyvendinimą.

3.1. Įrankio funkcionalumo įgyvendinimas

Metodo įgyvendinimui naudojamas populiariausias [17] „Java“ karkasas tinklalapių ir internetinių servisų kūrimui „Spring“ ir su juo susijusios technologijos. 3.1 pav. pateikiama įprasta „Java“ Servlet filtro veikimo diagrama. Programuotojas gali parašyti savo logiką įgyvendinantį filtrą ir jį užregistruoti. Šis sprendimas yra paprastas ir tiktų visoms sistemoms parašytomis pagal „Java Servlet“ specifikaciją. Tačiau naudojant šį sprendimą būtų sunku užregistruoti skirtingą logiką skirtingiems užklausų parametrams ir reikėtų saugoti šią logiką atskirose vietose nuo programos logikos.

Dėl to buvo pasirinktas kiek mažesnį pritaikomumą turintis sprendimas, tačiau leidžiantis saugoti filtravimo logiką prie kiekvieno užklausos parametro.



3.1 pav. Servlet filtro veikimo diagrama

Nuo filtro išvalančio visą įeinantį ir ateinantį turinį, sprendimas skiriasi tuo, kad yra pritaikytas tiksliai tam tikriems įrankiams naudojamiems kuriant tinklalapius „Java“ programavimo kalba. Pavyzdžiui kartu su „Spring MVC“ karkasu naudojamas JSON vertimo į „Java“ objektus ir atgal įrankis „Jackson“. Naudojant standartinius „Spring“ įrankius patobulinimas veiks, tačiau pasirinkus kitą biblioteką JSON operacijos, šis įrankis nebetiks.

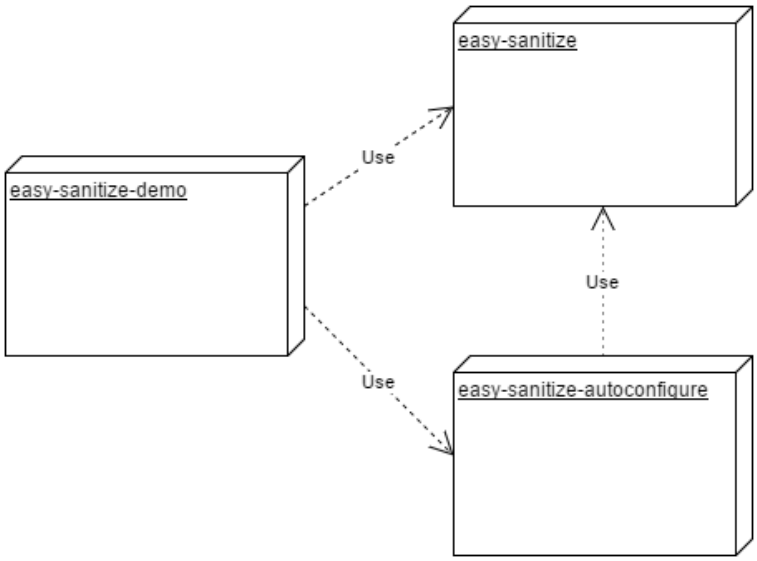
[17] http://stackoverflow.com/tags?page=2&tab=popular

3.2. Projekto struktūra

Projektas sudarytas iš 3 dalių, jo struktūra pateikta paveikslėlyje 3.2 . Pagrindinė dalis yra „easy-sanitize“. Šioje dalyje yra visa tikrinimo ir turinio išvalymo logika. Ši dalis gali būti naudojama bet kokioje „Spring 4“ karkasą naudojančioje sistemoje, nepriklausomai nuo „easy-sanitize-autoconfigure“. Konfigūracijos dalis gali būti naudojama sistemose su „Spring Boot“ karkasu bei su automatinės konfigūracijos parametru.

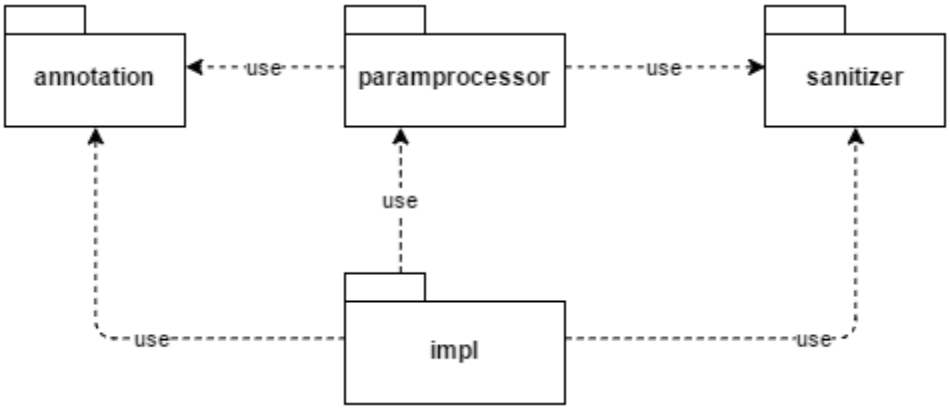
Nors diagramoje pavaizduota, kad „easy-sanitize-autoconfigure“ naudoja „easy-sanitize“, tai nereiškia, kad klientams kaip priklausomybę užtenka nurodyti „easy-sanitize-autoconfigure“. „Spring Boot“ komponentų kūrimo gairėse nurodoma, kad automatinės konfigūracijos komponentai neturėtų pridėti visų savo priklausomybių į kliento sistemą. Klientas turėtų pats nuspręsti ką naudoti, ir tik tada automatinė konfigūracija suveiktų.

Paskutinė projekto dalis, „easy-sanitize-demo“ nėra būtina įrankio veikimui, tai tėra pavyzdinis projektas naudojantis abu komponentus ir vaizduojantis įrankio veikimą.



3.2 pav. Projekto struktūra

Detalesnė modulio „easy-sanitize“ paketų diagrama pateikiama 3.3 pav. Modulį sudaro 4 pagrindiniai paketai su skirtinga logika ir skirtinga paketų struktūra jų viduje.



3.3 pav. Modulio „easy-sanitize“ paketų diagrama

Modulį sudarantys paketai:

- „Annotation„ – pakete laikomi anotacijų struktūros aprašymai.
- „Sanitizer“ – paketas skirtas vieno ar kelių HTML turinio išvalymo bibliotekų abstrakcijai. Pakete aprašomos skirtingo tipo turinio išvalymo klasės.
- „Param Processor“ – paketas susiejantis turinio išvalymo klasių parametrus su anotacijomis. Skirtingi turinio išvalymo komponentai turi skirtingus parametrus, pvz vienas komponentas leidžia nurodyti leistinų HTML elementų sąrašą. Šiame pakete laikomi komponentai paverčiantys anotacijos parametrus į komponento parametrus. Tai leidžia visiškai atskirti anotacijų logiką nuo turinio išvalymo logikos ir ateityje vetoj anotacijų naudoti visiškai kitą metodą, nekeičiant turinio išvalymo klasių ir jų testų.
- „Impl“ – pakete laikomomos implementacijos skirtingoms 3 šalių bibliotekoms. Pvz. naudojant „Spring MVC“ karkasą reikia rašyti vienus papildinius, o „Jackson“ JSON failų vertimą į Java objektus ir atgal (angl. *deserialization and serialization*) reikia rašyti kitokius papildinius.

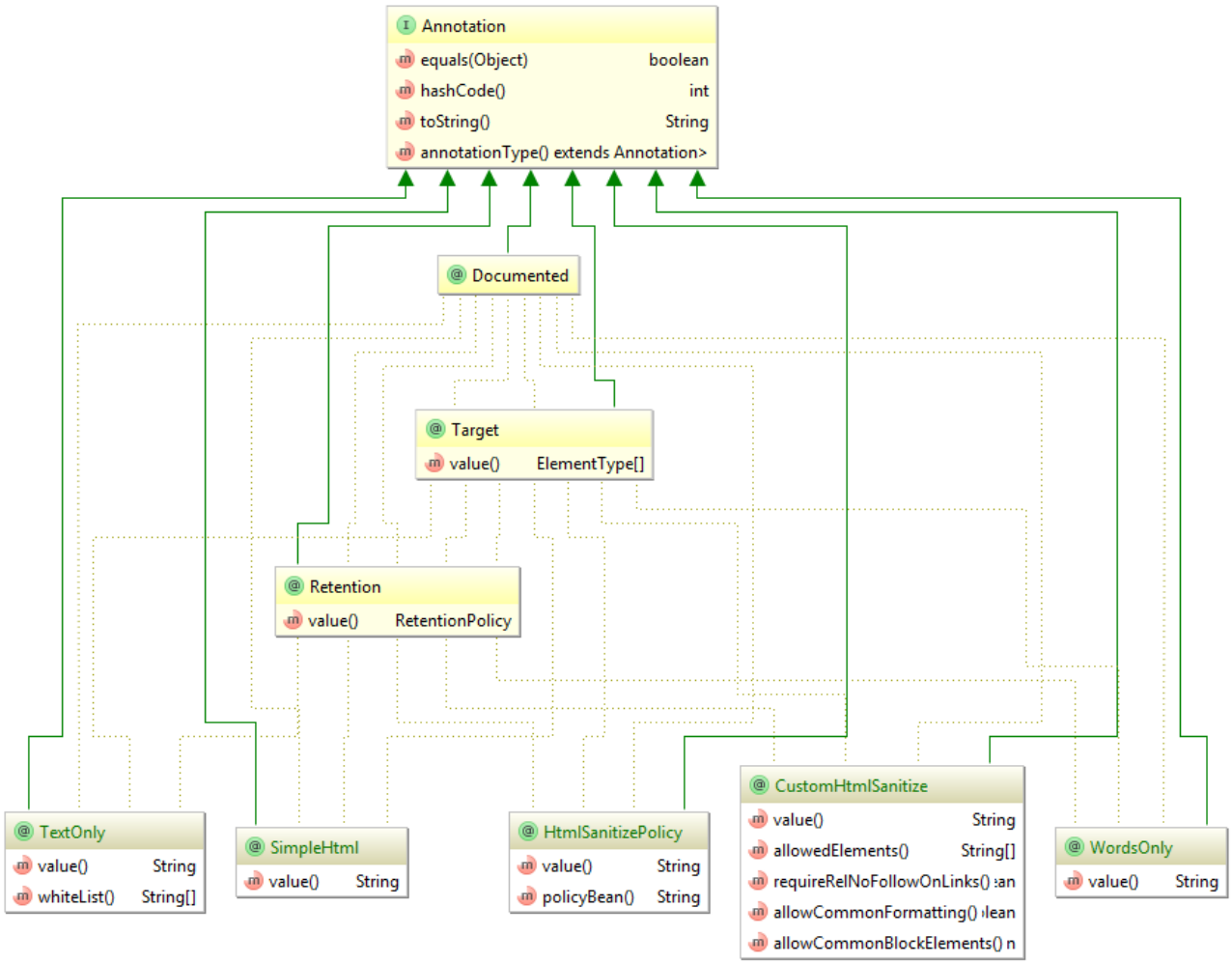
Modulis suprojektuotas stengiantis kuo labiau sumažinti tarpusavio priklausomybes (angl. *coupling*). Modulį sudaro 4 pagrindiniai paketai, kurie plečiantis projektui galėtų tapti atskirais nepriklausomais moduliais.

3.2.1. „Java“ anotacijų struktūra projekte

Parametrų pažymėjimui buvo pasirinktas anotacijos, šis metodas yra aprašytas skyriuje 2.2. Metodo įgyvendinimui reikia aprašyti specialias „Java“ anotacijas. Jų diagrama pavaizduota 3.4 pav.

Kiekviena anotacija pažymėta 3 papildomomis anotacijomis leidžiančiomis sumažinti klaidų ir problemų kiekį:

- @Target – anotacijas „Java“ programavimo kalboje galima taikyti įvairiems elementams, šia anotacija galima pažymėti kam bus taikoma aprašyta anotacija. Šiuo atveju pažymėta, kad visos anotacijos taikomas tik parametrams.
- @Retention – nurodo kaip ilgai saugoma informacija apie tipą anotuotą pasirinkta anotacija
- @Documented – pažymi, kad nurodyta anotacija bus dokumentuota naudojant „Java“ Doc ar kitą dokumentacijos kalbą.



3.4 pav. Įgyvendinamo metodo anotacijų klasių diagrama

3.2.2. Turinio išvalymo komponentų struktūra projekte

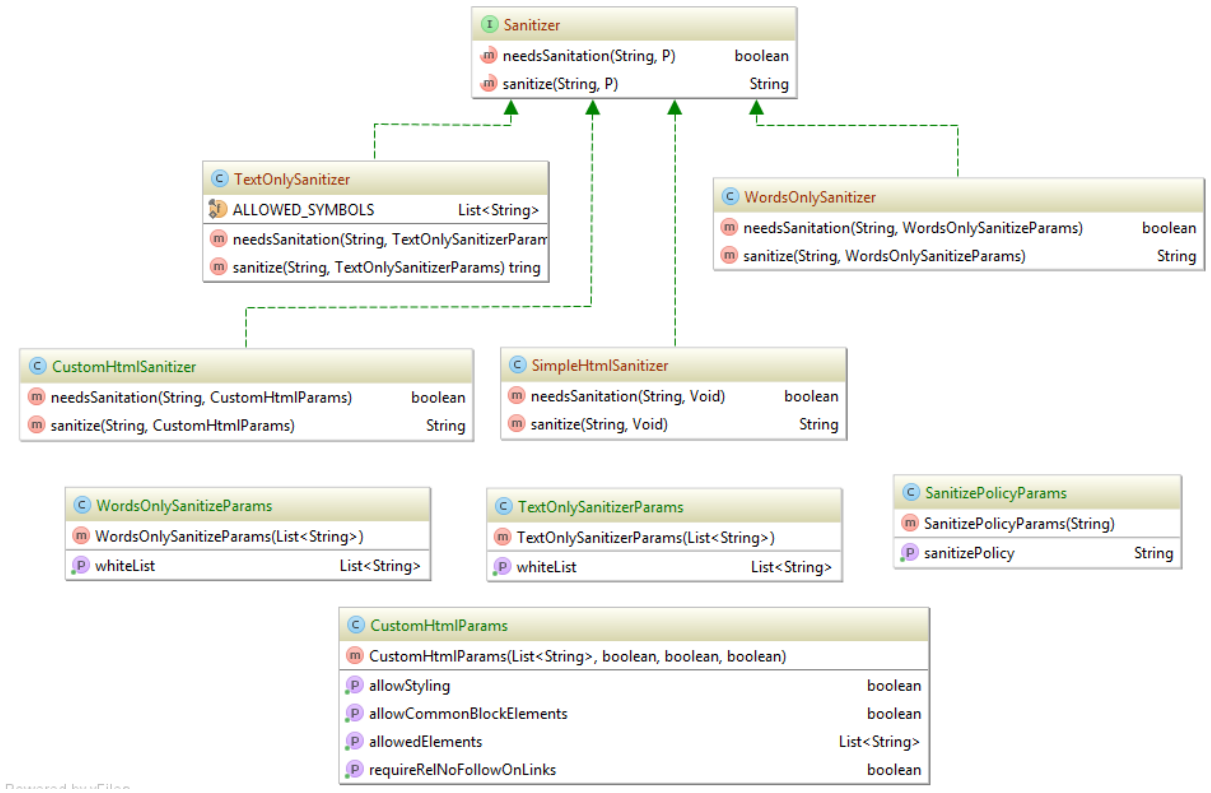
Tam, kad būtų atskirta logika susijusį su „Spring“ ir logika iš tiesų išvalanti pateikiamą turinį, buvo sukurta „Sanitizer“ sąsaja turinti du metodus:

- needsSanitation – ar pateiktą tekstą reikia išvalyti nuo nepageidaujamų simbolių.
- sanitize – išvalyti pateiktą tekstą nuo nepageidaujamų simbolių.

Šios sąsajos panaudojamas leidžia suvienodinti su „Spring“ susijusią logiką ir patogiau ištestuoti teksto išvalymą, taip pat labiau laikomasi S.O.L.I.D. objektinio programavimo principų. (17) S.O.L.I.D. principas reiškia:

- S – Single-responsiblity. Programinio kodo dalis (pvz. klasė ar metodas) yra atsakinga už vieną dalyką.
- O – Open-closed. Klasės ar metodai turėtų būti atviri praplėtimui, bet uždari modifikavimui.
- L – Liskov substitution. Liskovo pakeitimo principas, teigiama, kad klasę galėtų pilnai pakeisti jos subklasė.
- I – Interface segregation. Sąsajos turėtų būti kuo paprastesnės, t.y. klientas neturėtų privalėti įgyvendinti nereikalingų metodų.
- D – Dependency Inversion. Kodas turėtų būti paremtas sąsajų naudojimu, o ne konkrečių sąsajų implementacijų naudojimu.

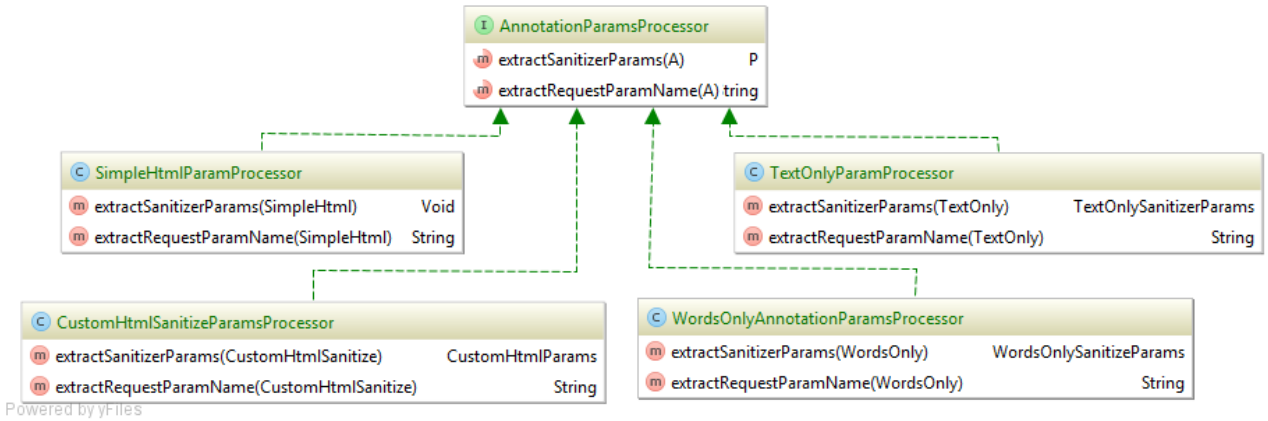
Klasių diagrama pateikiama 3.5 pav.



3.5 pav. Sanitizer sąsajos klasių diagrama

Kiekvienas turinio išvalymo komponentas gali turėti skirtingus parametrus, kurių struktūra aprašoma atskira klase. Pavyzdžiui „CustomHtmlSanitizer“ suteikia vartotojui galimybę detaliai konfigūruoti leidžiamus parametrus, todėl klasėje „CustomHtmlParams“ aprašomas leidžiamų parametrų sąrašas ir kiti parametrai.

Anotacijų parametrai sutampa su turinio išvalymo komponentų parametrais, tačiau jie nėra tarpusavyje susiję. Vienas turinio išvalymo komponentas gali būti konfigūruojamas su keliomis skirtingomis anotacijomis ir pan. Toks logikos atsiejimas pasiektas ankstesniame skyriuje minėtame pakete „paramprocessor“ esančiomis klasėmis. Šių klasių diagrama pateikia 3.6 pav.



3.6 pav. Anotacijų parametrų nuskaitymo klasių struktūra

Visos klasės įgyvendina „AnnotationParamProcessor“ sąsają turinčią du metodus:

- „extractSanitizerParams“ – metodas priimantis anotacijos objektą, kaip parametrą ir sugeneruojantis turinio išvalymo komponento parametrų klasę.
- „extractRequestParamName“ – metodas priimantis anotacijos objektą, kaip parametrą ir gražinantis užklausos parametro vardą, jei toks egzistuoja. Šis metodas naudojamas, kai anotacijomis yra pažymimi HTTP užklausų parametrai.

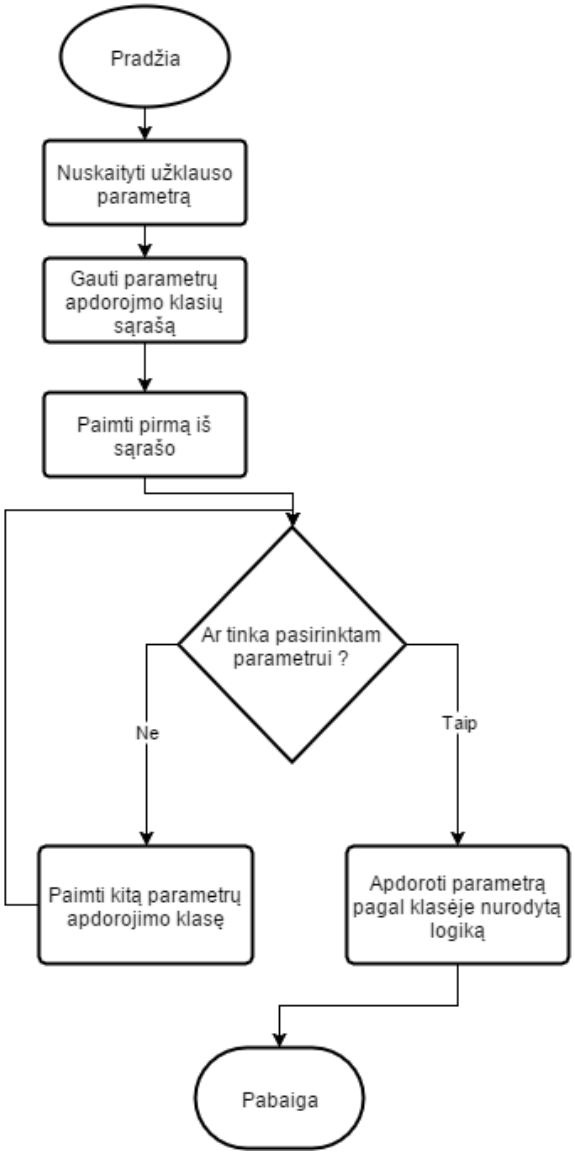
Parametrų generavimo klasės yra detalizuojamos dviem parametrais – anotacijos tipu ir parametro tipu, tai reiškia, kad teoriškai bet kokiai anotacijai galima aprašyti parametrų generavimo klasę leidžiančią susieti bet kurią anotaciją su bet kuriuo komponentų išvalymo komponentu.

3.2.3. „Spring MVC“ parametrų išvalymo įgyvendinimo veikimas ir struktūra

Pasirinktas sprendimas naudoja „Spring“ klasę skirtą HTTP užklausos parametrų pavertimui į „Java“ tipą. Jo veikimas pavaizduotas 3.7 pav. Tam reikia užregistruoti klasę parašytą pagal sąsają „HandlerMethodArgumentResolver“ turinčią du metodus:

- supportsParameter – ar parametrų apdorojimo klasė palaiko pasirinktą parametrą
- resolveArgument – atlikti parametro apdorojimą.

Kiekvienai anotacijai sukuriamas atskiras „HandlerMethodArgumentResolver“ objektas tikrinantis ar prie parametro yra pažymėta anotacija, ir jei nėra – tiesiog gražinantis vartotojo pateiktus duomenis, jų nepakeitus.

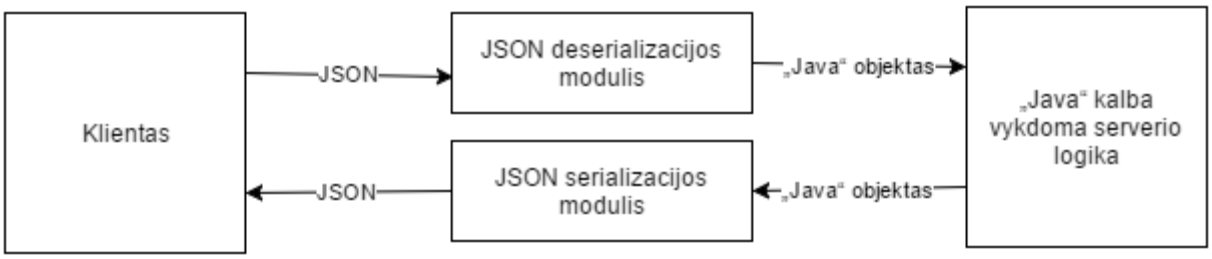


3.7 pav. „Spring“ parametrų apdorojimo registro veikimas

3.2.4. „Jackson“ objektų nuskaitymo ir rašymo į JSON plėtinio aprašymas

Kita projekto dalis skirta tinklalapiams, kurie naudoja „Java“ programavimo kalba parašytą sistemą kaip žiniatinklio tarnybą. Tarnyba yra kviečiama iš kliento pusėje vykdomo „Javascript“ kodo, kuris ir suformuoja tinklalapio dizainą bei suteikia interaktyvumo.

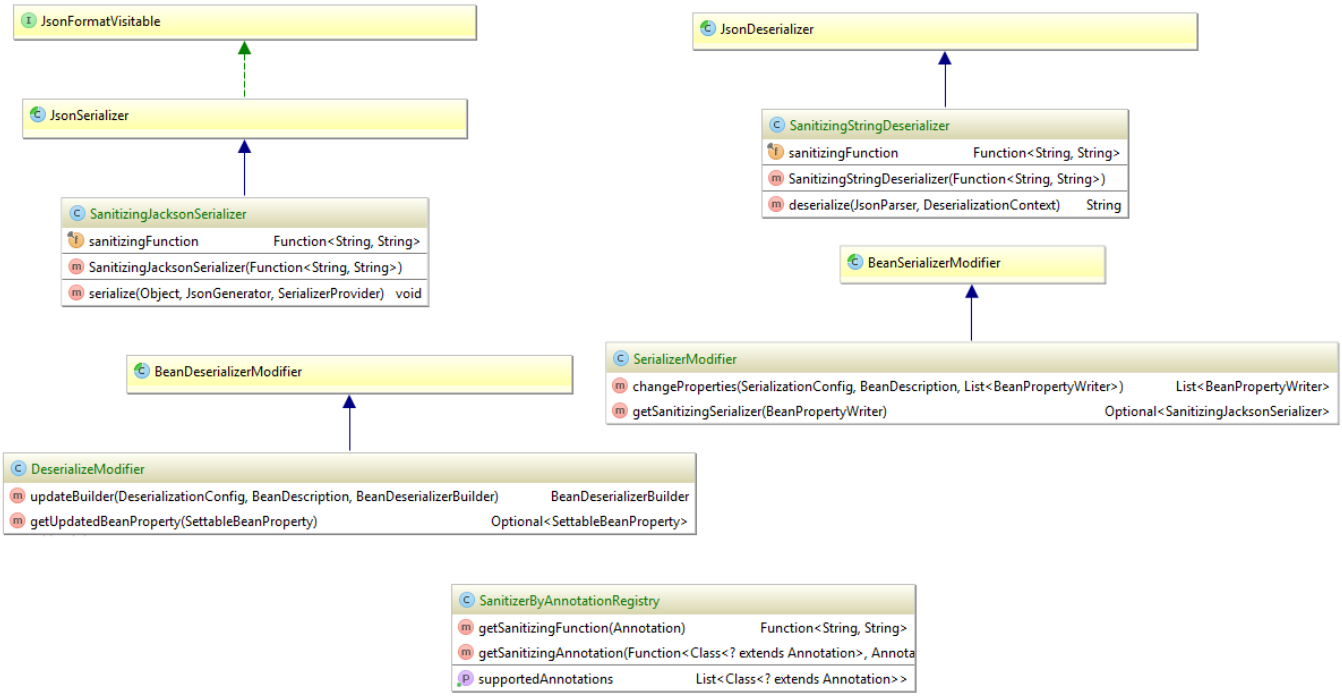
Dėl savo lankstumo ir suderinamumo su „Javascript“ kalba, duomenų apsikeitimui dažnai naudojama JSON duomenų struktūra. Kadangi „Java“ kalba nėra dinaminė ir tiesiogiai JSON struktūros, reikalingi papildomi komponentai galintys paversti JSON struktūrą į Java kalbos objektus ir atvirkščiai. Veikimo modelis parodytas 3.8 pav.



3.8 pav. JSON vertimo į Java objektus ir atgal schema

„Spring MVC“ karkase naudojama „Jackson“ biblioteka, kurios plėtiniai yra įgyvendinti šiame projekte. Kadangi „Jackson“ biblioteka turi platų plėtinių palaikymą, įgyvendintas patobulinimas išvalo tiek iš vartotojo užklausos ateinantį JSON turinį, tiek iš serverio vartotojui gražinamą turinį.

„Jackson“ plėtinys paremtas dviejų klasių paveldėjimu ir išplėtimu: „BeanDeserializerModifier“ ir „BeanSerializerModifier“. Šiose klasėse prieš verčiant JSON struktūrą į „Java“ objektą ir atvirkščiai, patikrinama ar nėra su turinio išvalymu susijusių anotacijų. Klasių diagrama pateikta 3.9 pav.



3.9 pav. „Jackson“ plėtinio klasių diagrama

Pateiktas plėtinio įgyvendinimas suteikia dvigubą saugumą, nes ne tik neleidžia į duomenų bazę įrašyti pavojingo kodo, tačiau užtikrina, jog pavojingas turinys esantis duomenų bazėje būtų išvalomas prieš pateikiant vartotojui.

3.3. Įgyvendinto įrankio panaudojimo pavyzdys

Šis įrankis yra naudojamas kaip programavimo procesą palengvinanti priemonė, todėl reikia užtikrinti, kad ją būtų patogu naudoti. Tam įrankis pritaikytas „Apache Maven“[18] priklausomybių valdymo įrankiui. Tai leidžia patogiai įterpti šį įrankį į jau egzistuojančią sistemą. Tam reikia prie priklausomybių sąrašo pridėti elementą pateiktą 3.10 pav.

```
<dependency>
    <groupId>net.katilius</groupId>
    <artifactId>easy-sanitize</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

3.10 pav. priklausomybės pridėjimo pavyzdys

Įrankis pateikiamas su jau paruošta konfigūracija užregistruojančia visas anotacijas. Vartotojas turi importuoti šią konfigūraciją prie savo programos konfigūracijų. Pavyzdys pateikiamas 3.11 pav.

```
@SpringBootApplication
@Import(ResolverConfiguration.class)
public class Application {

    public static void main(String[] args) throws IOException {
        SpringApplication.run(Application.class, args);
    }
}
```

3.11 pav. Konfigūracijos importavimo pavyzdys

Atlikus priklausomybės užregistravimą ir konfigūracijos importavimą, įrankį galima pradėti naudoti. 3.12 pav. pateikiamas anotacijos panaudojimo pavyzdys. Vartotojas pateikdamas formą gali užpildyti užklausos parametrą „Text“. Panaudojus anotaciją „@WordsOnly“ nebus leidžiami jokie HTML simboliai.

```
@RequestMapping(value = "/WordsOnly", method = RequestMethod.GET)
public String wordsOnly(@WordsOnly String text, Model model) {
    model.addAttribute("text", text);
    model.addAttribute("type", "WordsOnly");
    model.addAttribute("types", TYPES);
    return "demo";
}
```

3.12 pav. Įrankio anotacijos panaudojimo pavyzdys

[18] https://maven.apache.org/

3.4. Išvados

1. „Spring“ karkasas leidžia įgyvendinti įvairius pakeitimus, kurie leidžia pažymėjimus kodo elementus paprasta žyma visiškai pakeisti jų veikimą.
2. Labiau atskirti komponentai padeda išlaikyti programos kodą paprastesnį ir lengviau testuojamą.
3. Įgyvendintas sprendimas sukuria sąsają leidžiančią ateityje bet kada pakeisti ar papildyti naudojamų įrankių sąrašą ir vietoj „Owasp Html Sanitizer“ naudoti visai kitą įrankį.

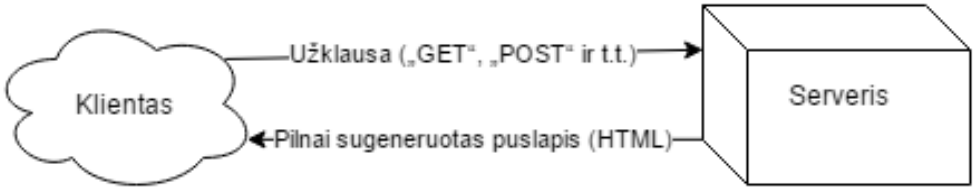
4. ĮRANKIO PANAUDOJIMO PAVYZDŽIAI

Šiame skyriuje pateikiami įrankio panaudojimo pavyzdžiai. Stengiamasi pateikti skirtingas su „Java“ kalba parašytas sistemas, kurių architektūra galėtų atitikti realios sistemos architektūrą. Sukūrus pavyzdines sistemas pateikiami sistemos kodo pavyzdžiai prieš ir po apsaugos pritaikymo. Reikėtų atsižvelgti į tai, kad pavyzdinės sistemos yra itin paprastos, todėl sumažėjęs kodo kiekis matomas ne taip akivaizdžiai, kaip pavyzdžiui sistemoje su daugybe klasių.

Be pritaikymo pavyzdžių, lyginama ir sistemos sparta. Kadangi pavyzdinės sistemos yra paprastos ir viduje nėra vykdoma jokia verslo logika, net ir didelis spartos sumažėjimas realioje sistemoje gali būti kur kas mažesnis. Taip pat reikia atsižvelgti į tai, kad „Java“ parašytų programų sparta stipriai padidėja, po kurio laiko, kai sistema veikia. Ne „Java“ pasaulyje testavimas yra paprastas: parašoma programa ir išmatuojama jos vykdymo sparta. „Java“ pasaulyje yra viena svarbi detalė: „kompiliavimas pačiu laiku“ (angl. *just-in-time compilation*), iš esmės tai reiškia, kad kodas pilnai optimizuojamas ir veikia geriausia sparta po keleto minučių. Dėl šios priežasties (ir keleto kitų), „Java“ spartos analizėse daug dėmesio skiriama apšilimo periodams: kodo sparta dažniausiai matuojama po to, kai programa buvo vykdoma pakankamai ilgą laiką, tam, kad kodas būtų sukompiliuotas ir optimizuotas (18)

4.1. Komentarų komponentas su galimybe formatuoti tekstą

Pirmasis panaudojimo pavyzdys aprašo įrankio panaudojimą svetainėje, kurios visas turinys yra generuojamas serveryje ir grąžinamas HTML formatu. Svetainės diagrama pateikta 4.1 pav.



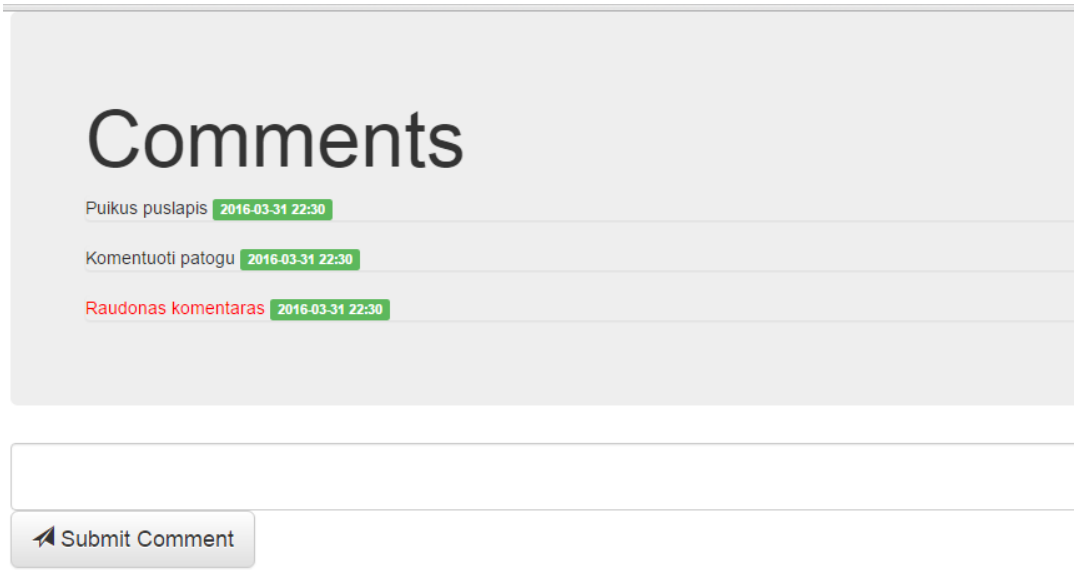
4.1 pav. Komentarų sistemos veikimo diagrama

Tokio modelio svetainėje, pridėjus naują komentarą ar atsinaujinus turiniui, norint pamatyti pakeitimus reikia atlikti užklausą į serverį, serveris gražina pilnai sugeneruotą naują puslapį, kurį naršyklė turi iš naujo apdoroti. Puslapis nėra toks interaktyvus, tačiau supaprastėja jo kūrimo procesas.

4.1.1. Komponento aprašymas prieš apsaugos pritaikymą

Sukurtas paprastas komponentas leidžiantis palikti komentarus puslapyje. Lankytojai turi turėti galimybę įterpti suformatuotą tekstą, pvz. paryškinti tam tikrus žodžius ar pakeisti jų spalvą. Tinklalapio vaizdas pateiktas 4.2 pav. Norint įgyvendinti tokį funkcionalumą reikia išjungti specialių

simbolių užkodavimą, t. y. tekstą „“ naršyklė turėtų interpretuoti ne kaip paprastą tekstą, tačiau kaip HTML žymą.



4.2 pav. Pavyzdinis komentarų puslapis

Puslapis sukurtas naudojant „Spring MVC“ karkasą ir „Thymeleaf“ šablonų karkasą. Puslapio dizaino kodas pateiktas 4.3 pav. Svarbiausia vieta kode yra atributas „th:utext“ šis atributas nurodo, kad pateikto teksto nereikia apsaugoti jį užkoduojant. 4.4 pav. Pateikiamas komentarų išsaugojimo kodas. Vartotojo paliktas komentaras tiesiog išsaugomas duomenų bazėje, o išsaugojus atvaizduojami visi egzistuojantys komentarai.

```
<div class="jumbotron">
    <h1>Comments</h1>
    <div class="list-group" th:each="comment : ${comments}">
        <span class="list-group-item-text"
th:utext="${{comment.text}}">Text</span>
        <span class="label label-success" ${{comment.date}}">>2011-11-11
10:30</span>

    </div>
</div>

<form action="#" th:action="@{/post-comment}" th:object="${comment}"
method="post">
    <textarea class="form-control" th:field="*{text}"></textarea>
    <button type="submit" class="btn btn-default btn-lg">
        <span class="glyphicon glyphicon-send" aria-hidden="true"></span> Submit
Comment
    </button>
</form>
```

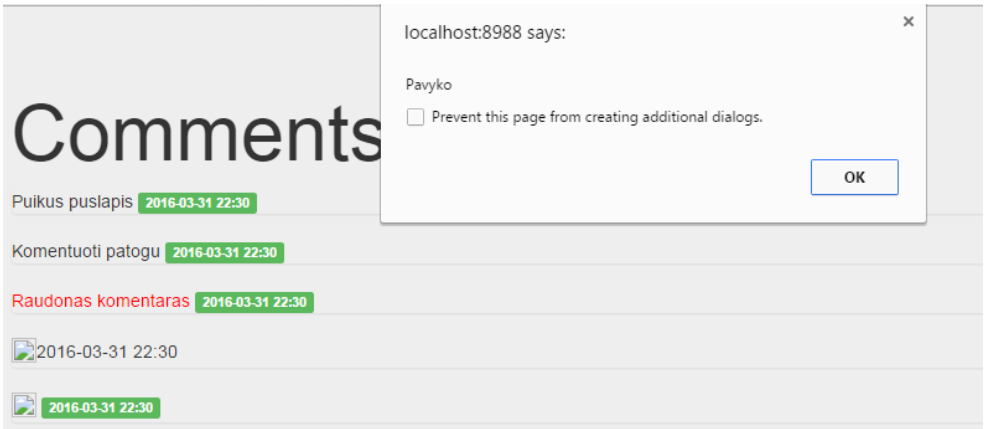
4.3 pav. Pavyzdinio komentarų puslapio dizaino kodas

```
@RequestMapping(value = "/post-comment", method = RequestMethod.POST)
public String postComment(Model model, @ModelAttribute CommentModel comment) {
    repository.save(comment);
    model.addAttribute("comments", repository.findAll());
    return "comments";
```

```
}
```

4.4 pav. Pavyzdinio komentarų puslapio komentarų išsaugojimas

Neapsaugotų žymų įvedimas leidžia lengvai vykdyti norimą „Javascript“ kodą. Užtenka į komentarų laukelį įvesti tekstą „<img src="/none" onError="alert('Pavyko')></img>“ išsaugoti. Kodas nėra įvykdomas iš karto, kadangi šiuolaikinės naršyklės sugeba atpažinti „Javascript“ kodą pateiktą užklausoje. Tačiau, kadangi duomenys yra saugomi duomenų bazėje ir atvaizduojami vėliau, perkrovus puslapį matomas vaizdas pateiktas 4.5 pav.



4.5 pav. Pavyzdinio komentarų puslapio pažeidžiamumo pavyzdys

4.1.2. Apsaugos pritaikymas

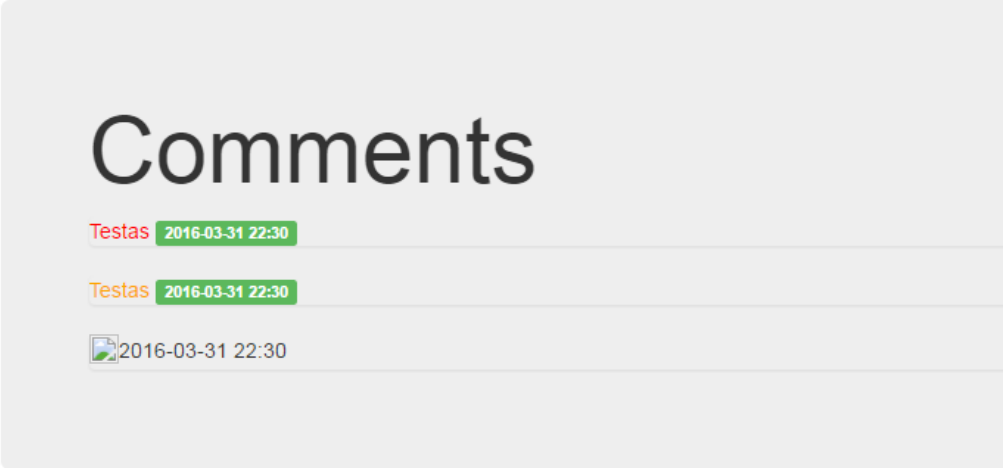
Komponentas pridedamas pom.xml faile atnaujinant priklausomybių sąrašą su nauju elementu:

```
<dependency>
    <groupId>net.katilius</groupId>
    <artifactId>easy-sanitize</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Komentaro išsaugojimo kodas atnaujinamas anotacija nurodančia, leidžiamus HTML elementus ir leidžiamą paprastą elementų stiliaus taikymą:

```
@RequestMapping(value = "/post-comment", method = RequestMethod.POST)
public String postComment(Model model,
                          @CustomHtmlSanitize(allowCommonFormatting = true,
allowedElements = {"b"})
                          CommentModel comment) {
    repository.save(comment);
    model.addAttribute("comments", repository.findAll());
    return "comments";
}
```

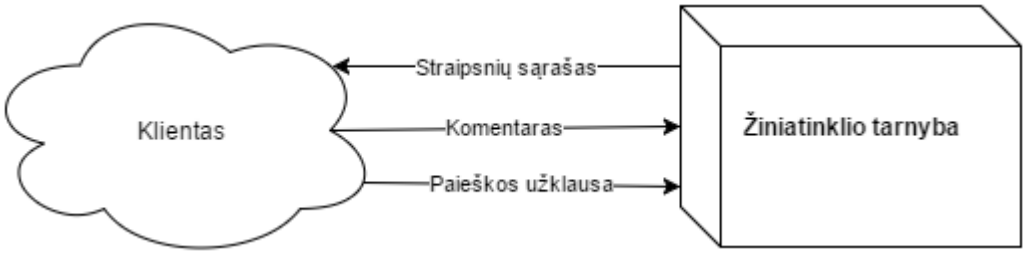
Atnaujinus tinklalapį pateiktu kodu, vis dar galima pritaikyti stilių komentarams. Pavyzdžiui, įrašius komentarą „Testas“. Tačiau komentaro dalys turinčios pavojingus atributus, leidžiančius vykdyti „Javascript“ kodą yra pašalinamos. Todėl įvykdžius ankstesniame skyriuje pateiktą kodą gaunamas 4.6 pav. pateiktas rezultatas. Komentaras išsaugomas, tačiau pavojingas kodas nėra įvykdomas.



4.6 pav. Pavyzdinio komentarų puslapio pavyzdys pritaikius apsaugą

4.2. Žiniatinklio tarnybos apsaugos pavyzdys

Ne visi tinklalapiai yra kuriami šablonų principu minimu ankstesniame skyriuje. Kitas būdas yra interaktyvesnės „vieno puslapio“ (angl. *Single-page app*) svetainės. Tokia sistema dažniausiai sudaryta iš dviejų dalių – žiniatinklio tarnybos pateikiančios turinį ar išsaugančios vartotojo užklausas ir kliento naršyklėje vykdomos interaktyvios dalies, kuri bendrauja su žiniatinklio tarnyba.



4.7 pav. Vieno puslapio sistemos pavyzdys

Naudojant karkasus, XSS pažeidžiamumų pavojus sumažėja, kadangi karkasai taiko įvairias apsaugos priemones neleidžiančias lengvai vykdyti „Javascript“ kodo, jei jis nurodytoje puslapio vietoje neturėtų būti vykdomas. Tačiau atsiranda ta pati problema, kaip ir kuriant serveryje generuojamus puslapius – kartais vartotojui reikia leisti pateikti saugų HTML ir vėliau jį atvaizduoti.

Šiame skyriuje pateikiamas turinio išvalymo sprendimas naudojant žiniatinklio tarnybą parašytą „Java“ programavimo kalba ir bendraujančią su klientu „JSON“ formato pranešimais.

4.2.1. Struktūros aprašymas

Pavyzdiniai žiniatinklio tarnybai naudojama paprasta struktūra iš laukų skirtų straipsnių pateikimui: straipsnio autorius, straipsnio pavadinimas ir straipsnio tekstas.

Struktūros pavyzdys pateikiamas 4.8 pav. Pavyzdinė žiniatinklio tarnyba išsaugo ir atvaizduoja straipsnį pateiktą per „POST“ užklausą. Kadangi straipsnio autorius turi galimybę formatuoti tekstą, tekstas turi būti interpretuojamas kaip HTML turinys.

```
{
    "text": "Straipsnio <b>tekstas</b>",
    "title": "Straipsnio pavadinimas",
    "author": "Vardenis",
    "tags": [
        "xss"
    ],
    "_links": {
        "self": {
            "href": "http://localhost:8080/articles/1"
        },
        "article": {
            "href": "http://localhost:8080/articles/1"
        }
    }
}
```

4.8 pav. Straipsnių pateikimo žiniatinklio tarnybos struktūros pavyzdys

Norint išsaugoti duomenis ir atlikti kitus veiksmus su jais, JSON struktūra turi būti parvesta į Java objektą. Jo struktūra aprašoma klasę pateikta 4.9 pav. Įprastoje sistemoje duomenų ir atvaizdavimo struktūrų aprašymai yra atskiriami. Tuo atveju, jei vartotojui reikėtų pateikti skirtingos struktūros duomenis, nereikėtų keisti duomenų bazės struktūros ir pakeitus duomenų bazės struktūrą nereikėtų keisti atvaizdavimo struktūros. Dėl paprastumo, šiame pavyzdyje duomenų bazės struktūra aprašoma ta pačia klase kaip ir atvaizdavimo struktūra.

```
@Entity
public class Article {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String text;
  private String title;
  private String author;
  @ElementCollection
  private List<String> tags;

  /*
     Setters and Getters

  */
}
```

4.9 pav. Straipsnio klasės struktūra prieš apsaugojimą

Norint pritaikyti saugą šioje sistemoje yra kelios vietos, kuriose galima įterpti turinio išvalymo funkciją:

- Nustatant ir nuskaitant objekto reikšmes (angl. *setters and getters*). Patogu, nes galima nustatyti vienintelę vietą kurioje nustatomi duomenys. Tačiau sunku išvengti kodo kartojimo taikant panašias taisykles kelioms skirtingoms klasėms.
- Išsaugojant į duomenų bazę.
- Nuskaitant vartotojo duomenis ir gražinant juos atgal.

4.2.2. Apsaugos pritaikymas

Naudojant patobulinimą nebereikia rūpintis įrankių objektų inicijavimu ar iškvietimu. Pasirinktiems metodams tereikia deklaratyviai nurodyti leidžiamas taisykles, ir sukompiliuoti naujai aprašytą kodą. Panaudojimo pavyzdys pateiktas 4.10 pav.

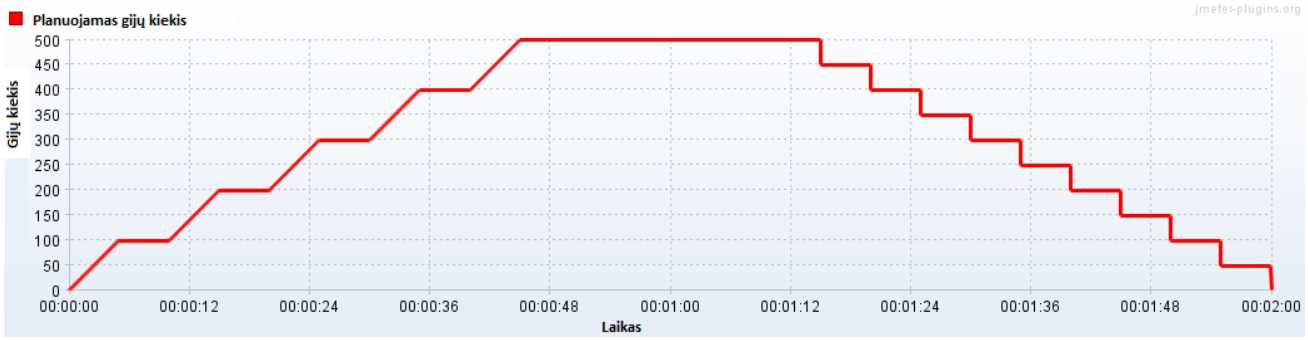
```
@Entity
public class Article {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
    @CustomHtmlSanitize(
            allowedElements = {"b", "p", "i", "h1", "h2", "h3"},
            allowCommonFormatting = true)
  private String text;
  private String title;
  private String author;
  @ElementCollection
  private List<String> tags;

  /*
     Setters and Getters

  */
}
```

4.10 pav. Straipsnio klasės struktūra pritaikius apsaugą

Pateiktame pavyzdyje naudojama anotacija „CustomHtmlSanitize“, suteikianti galimybę nurodyti leidžiamas žymas ir kitus parametrus. Naudojant kitas anotacijas, su jau nurodytais parametrais, kodo kiekis sumažėja dar labiau.

4.3. Spartos tyrimas

Spartos tyrimui naudojamas analogiškas apkrova kaip ir tiriant nepatobulintus įrankius. T. y. 500 lygiagrečių vartotojų. Vartotojų kiekis didinamas po 100 kas 5 sekundes. Apkrovos diagrama pavaizduota 4.11 pav.



4.11 pav. Apkrovos testavimo vartotojų kiekio diagrama

Kadangi pavyzdinėje sistemoje naudojama reliacinė duomenų bazė, užklausų greitis, palyginus su tyrimu be duomenų bazės turėtų žymiai sumažėti. Tačiau tai tuo pačiu suteikia šiek tiek realesnius rezultatus, kadangi įprastoje sistemoje užklausos laikas labiausiai įtakojamas integracijų su duomenų baze ar kita žiniatinklio tarnyba, o ne turinio validavimo ar išvalymo. Apačioje esančiose lentelėse pateikiami spartos tyrimo rezultatai.

4.1 lentelė „JSON“ struktūros išvalymo spartos rezultatai

Testavimo metodas	**90% procentilis**	**99% procentilis**	**Užklausų per sekundę**
Be apsaugos	1111	3051	640
Pritaikius patobulintą apsaugą	1075	2598	587
Owasp HTML Sanitizer be patobulinimo	1084	2376	605

Iš rezultatų galima matyti, kad sumažėjo užklausų per sekundę kiekis, t. y. sistemai vienu metu pavyko įvykdyti mažiau užklausų nei įprastai. Tačiau sumažėjo ir procentiliai, tai reiškia, kad netaikant apsaugos serverio procesorius buvo mažiau apkrautas. Mažiau apkrovus procesorių, daugiau užklausų buvo apdorojamos ir siunčiamos į duomenų bazę, kuri nuo didelio užklausų kiekio pradėjo veikti lėčiau. Taikant apsauga užklausos įvykdomos šiek tiek lėčiau, tačiau panašesniu vidutiniu grečiu.

Atlikus palyginimą su patobulinimu ir be, matyti, kad patobulinimas šiek tiek sulėtinimo sistemą. Kuriant patobulinimas pagrindinis dėmesys buvo teikiamas konfigūracijos patogumui, todėl kai kuriose vietose nebuvo atsižvelgiama į spartą.

4.2 lentelė Testavimo rezultatai

Testavimo metodas	**90% procentilis**	**99% procentilis**	**Užklausų per sekundę**
Be apsaugos	504	644	1357
Pritaikius patobulintą apsaugą	723	1060	1015
Owasp HTML Sanitizer be patobulinimo	1084	2376	605

4.4. Išvados

1. Pateiktą patobulinimą pavyko panaudoti tiek serveryje užkraunamai svetainei, tiek „vieno puslapio“ svetainei. Skirtingo tipo sistemos buvo valdomos su tokiomis pat anotacijomis
2. Bibliotekos „Jackson“ papildymas suteikė didesnį saugumą, kadangi jis užtikrino saugumą tiek įeinančiam, tiek išeinančiam turiniui. Anotacijos leido tą pačią konfigūraciją aprašyti vieną kartą ir taikyti abiejų tipų turiniui.
3. Spartą ištirti „Java“ programavimo kalba parašytoje sistemoje buvo sunku, dėl kalboje nuolat vykdomos kodo optimizacijos. Dėl to reikėjo atlikti apšilimo operacijas, prieš pradedant tikrąjį tyrimą.
4. Pritaikius patobulinimą sistemos sparta nežymiai sumažėjo. Vis dėlto sumažėjimas buvo testuojamas itin paprastoje sistemoje, todėl didesnėje sistemoje šis sumažėjimas būtų dar mažesnis. Tačiau, prieš plečiant komponentą reikėtų atlikti detalesnius spartos tyrimus ir optimizuoti jos veikimą.

5. IŠVADOS

1. Nors XSS nėra pats pavojingiausias svetainės pažeidžiamumas, nuo jo apsaugoti svetainę būtina. Jis yra vienas iš dažniausiai pasitaikančių pažeidžiamumų svetainėse. Nauji XSS pažeidžiamumai atsiranda pakankamai dažnai, todėl apsisaugojimo metodai geriausiai aprašyti metodiniuose straipsniuose ir žiniaraščio įrašuose, o ne knygose.
2. Nėra visiems atvejams tinkamo sprendimo apsaugai nuo XSS. Apsauga nuo XSS turėtų būti taikoma priklausomai nuo konteksto kuriame turinys yra pateikiamas ir nuo turinio taisyklių. T. y. jei įterpiamame turinyje visai neleidžiama įterpti HTML elementų, reikalingos skirtingos apsaugos priemonės nei tada, kai HTML žymos yra leidžiamos
3. Populiariausias karkasas „Java“ programavimo kalboje nesuteikia pakankamai galimybių lanksčiai valdyti įeinantį ir išeinantį turinį, jei vartotojui turi būti leidžiama naudoti HTML žymas. Šią problemą išsprendžia „Owasp Html Sanitizer“ įrankis, tačiau jis nėra pakankamai patogus naudoti.
4. Sukurtas sprendimas integruoja „Owasp Html Sanitizer“ įrankį į „Spring“ infrastruktūrą suteikdamas deklaratyvaus konfigūravimo ir išvalymo galimybes. Patobulinimas sumažina tikimybę neapsaugoti to paties turinio įeinančio per skirtingus šaltinius, arba pamiršti apsaugoti išeinančio to paties tipo turinio. Atliekant tolimesnius įrankio tobulinimus, pirmiausiai reikėtų išanalizuoti jo veikimą ir optimizuoti greitaveiką.

6. LITERATŪRA

1. **Hope, Paco ir Walther, Ben.** *Web Security Testing Cookbook.* Sebastopol : O'Reilly Media, Inc, 2008. p. 128. ISBN 978-0-596-51483-9.
2. **Cenzic Vulnerability Report 2014. [Tinkle] [Cituota: 2014 m. 12 15 d.] http://www.cenzic.com/downloads/Cenzic_Vulnerability_Report_2014.pdf.**
3. Worst Wordpress Hole for five years affects 86 of sites. *The Register.* [Tinkle] 2014 m. 11 24 d. [Cituota: 2014 m. 12 15 d.] http://www.theregister.co.uk/2014/11/24/worst_wordpress_hole_for_five_years_affects_86_of_sites/.
4. Protecting Your Cookies: HttpOnly. *Coding Horrow.* [Tinkle] 2008 m. 108 28 d. [Cituota: 2015 m. 01 15 d.] http://blog.codinghorror.com/protecting-your-cookies-httponly/.
5. *Bypassing Internet Explorer's XSS Filter.* Brooks, Michael. 2011.
6. Kazanavičius, Egidijus, Toldinas, Jevgenijus ir Čeponis, Jonas. *Programų sauga.* s.l. : TEV, 2011. eISBN 978-609-433-064-3.
7. Jeremiah , Grossman, et al. *XSS Attacks Cross Site Scripting Exploits and Defense.* Burlington, : Syngress Publishing, Inc, 2007. p. 18. 1-59749-154-3.
8. Mitropoulos, Dimitris , et al. Countering code injection attacks: a unified approach. *Information Management & Computer Security.* 2011 m., T. 19, 3.
9. XSS Filter Evasion Cheat Sheet. *OWASP.* [Tinkle] 2016 m. 03 17 d. [Cituota: 2016 m. 03 23 d.] https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.
10. DOM based XSS Prevention Cheat Sheet. *OWASP.* [Tinkle] 2016 m. 03 03 d. [Cituota: 2016 m. 03 20 d.] https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet.
11. Kallin, Jakob ir Valbuena, Irene Lobo. A comprehensive tutorial on cross-site scripting. *Excess XSS.* [Tinkle] [Cituota: 2015 m. 01 27 d.] http://excess-xss.com/.
12. *Proactive Web App Defenses.* Manico, Jim. s.l. : OWASP, 2013.
13. *Detecting Cross Site Scripting Vulnerabilities.* Dong, Guowei, et al. s.l. : IEEE, 2014. 978-1-4799-5821-4.
14. Zuasti, Ricardo. Anti cross-site scripting (XSS) filter for Java web apps. *Java Code Geeks.* [Tinkle] 2012 m. 07 2 d. [Cituota: 2016 m. 03 20 d.] https://www.javacodegeeks.com/2012/07/anti-cross-site-scripting-xss-filter.html.
15. FEED Validator. *W3C Html Validator Documentation.* [Tinkle] World Wide Web Consortium (W3C). [Cituota: 2014 m. 01 27 d.] http://validator.w3.org/feed/docs/warning/SecurityRiskAttr.html.
16. Declarative Programming in Java. *O'Reilly Media.* [Tinkle] 2004 m. 04 21 d. [Cituota: 2015 m. 06 09 d.] http://www.onjava.com/pub/a/onjava/2004/04/21/declarative.html.
17. Oloruntoba, Samuel. S.O.L.I.D: The First 5 Principles of Object Oriented Design. *Scotch.io.* [Tinkle] 2015 m. 03 18 d. [Cituota: 2016 m. 04 04 d.] https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.
18. Oaks, Scott. *Java Performance: The Definitive Guide.* Sebastopol : O'Reilly Media, Inc., 2014. 978-1-449-35845-7.