KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Tomas Kolaitis

CSRF atakų tyrimas

Baigiamasis magistro darbas

Vadovas
Doc. dr. Jonas Čeponis

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

TVIRTINU

Katedros vedėjas

Prof. dr. Algimantas Venčkauskas

CSRF atakų tyrimas

Baigiamasis magistro darbas

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

Vadovas

Doc. dr. Jonas Čeponis

Recenzentas

Doc. dr. Dangis Rimkus

Projektą atliko

Tomas Kolaitis

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos

(Fakultetas)

Tomas Kolaitis

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „CSRF atakų tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 ___ m. _____ ____ d.

Kaunas

Patvirtinu, kad mano **Tomo Kolaičio** baigiamasis projektas tema „CSRF atakų tyrimas“ yra parašytas visiškai savarankiškai, o 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)*

Kolaitis, T. CSRF atakų tyrimas. *Magistro baigiamasis darbas*/ vadovas doc. dr. Jonas Čeponis; Kauno technologijos universitetas informatikos fakultetas, kompiuterių katedra.

Kaunas, 2015. 40 psl.

SANTRAUKA

Laikui bėgant stipriai išaugo interneto vartotojų skaičius. Kiekvienas metais jų atsiranda dar daugiau ir internetas tampa neatsiejama mūsų visų gyvenimo dalimi. Kadangi interneto vartotojų skaičius stipriai auga, į tai suskubo atsižvelgi ir verslas. Kiekvieną dieną kuriamos internetinės sistemos leidžiančios žmonės apsipirkinėti, bendrauti su draugais ar tiesiog leisti laiką internete. Išpopuliarėjus debesų technologijoms, vis daugiau žmonių ir įmonių perkelia savo resursus ir duomenų saugyklas į debesis. Šie duomenys taip pat tampa prieinami per internetines sistemas. Dėl tokio didelio populiarumo atsiranda vis daugiau žmonių siekiančių pasisavinti svetimą informaciją. Vienas iš galimų būdų kaip tai galima pasiekti yra CSRF tipo atakos. Šio tipo atakų tikslas, priversti prisijungusio vartotojo naršyklė atlikti tam tikrus veiksmus to vartotojo vardu, jam to net nenutuokiant. Šio tipo atakos nėra orientuotos į pačių sistemų „laužymą“, o daugiau į paliktų spragų išnaudojimą.

Norint apsisaugoti nuo tokio tipo atakų galima dvejopa apsauga. Apsaugos mechanizmus gali naudoti pats vartotojas, kas nėra labai efektyvu, arba galima bandyti pašalinti visas silpnas vieta sistemoje, kurios leidžia CSRF tipo atakas. Sistemos apsaugojimas yra gerokai efektyvesnis už pačio vartotojo naudojamus apsaugos metodus. Vartotojas neturėtų būti verčiamas pats naudoti papildomus apsaugos metodus.

Sistemos apsaugos metodų galima rasti labai įvairių. Dažniausiai paplitęs metodas yra sistemos formų apsaugojimas pridedant papildomą atsitiktinę žymę. Žymė yra sugeneruojama vartotojui prisijungus ir pridedama prie kiekvienos siunčiamos POST tipo užklausos į serverį kaip nematomas laukas. Šį žymė turi būti atsitiktinė ir neatspėjama, kad atakuotojas negalėtų tokios pačios sugeneruoti pats.

Darbe apžvelgtos galimos apsisaugojimo priemonės nuo CSRF tipo atakų. Apžvelgti jų pliusai ir minusai. Išnagrinėti populiariausi karkasai kuriais galima pasinaudoti apsaugant sistemą nuo anksčiau minėtų atakų. Išnagrinėjus esamus apsisaugojimo būdus buvo pasiūlytas naujas apsaugos metodas kuriuo galima padidinti esamos apsaugos metodus.

Kolaitis, T. Investigation of CSRF attacks. Master's thesis / supervisor Assoc. Prof. Dr. Jonas Čeponis; Department of Computer Science, Faculty of Informatics, Kaunas University of Technology. Kaunas, 2015. 40 p.

SUMMARY

During last few years count of Internet users was increasing rapidly. Every year there are a lot of new users and Internet is becoming vital part of our life. Since amount of Internet users is growing so fast, this was noticed by the business as well. Every day we can see new web based systems created allowing us shop, communicate or just spend time online. With increasing popularity of cloud computing, more and more people and companies moving their resources and data to cloud. All this data is usually accessed through web systems. Since there are so many users on the Internet everybody can notice that numbers of hackers are increasing rapidly as well. The main objective is user's data. One of the way how data could be retrieved is using CSRF attacks. The main purpose of this attack is to force logged in user's browser to make some actions on behalf of that user without even noticing him. The main goal of this attacks it not breaking system itself, but rather exploiting current bugs in the system.

There are two ways to prevent these attacks. Either the user could use protection tools, even though this is not very sufficient, or programmers could try to remove all bugs in the system which allows CSRF attack to take place. Hardening security on the system itself is much more effective than any method user could use himself. User should not be forced to use extra tools to improve security.

There could be found a lot of different ways how programmers try to protect the system. The most popular one is by adding extra random field to form. Random value is generated once user is authenticated and is added to every form which is sent using POST method to server. This value must be truly random, otherwise attacker might be able to guess it.

At this work I have analyzed all possibilities which could be used to protect the system, their advantages and disadvantages. I have also analyzed the most popular frameworks which could be used to protect system from CSRF attacks. After taking into consideration all the methods which could be used to protect the system, I came up with new method which could improve protection from CSRF attacks.

Turinys

PAVEIKSLĖLIŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

1. **SQL (*angl. Structured Query Language*)** – struktūrizuota užklausų kalba. Tai yra viena populiariausių šiuo metu naudojamų kalbų, pritaikyta aprašyti duomenis ir manipuliuoti jais releacinių duomenų bazių valdymo sistemose.
2. **HTML (*angl. Hyper text Markup Language*)** – hiperteksto žymėjimo kalba. Tai yra kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete.
3. **Sesija** – kompiuterių kontekste sesija yra interaktyvūs informacijos mainai tarp dviejų ar daugiau tarpusavyje bendraujančių įrenginių.
4. **Web puslapis** – informacinės sistemos dalis matoma tam tikru metu vartotojui.
5. **HTTP POST** – POST užklausos pagalba duomenys pateikiami sistemai, kad juos apdorotų, dažniausiai naudojama pateikiant formas.
6. **HTTP GET** – GET dažniausia užklausa reikalaujanti tam tikro resurso duotu adresu. Dažniausiai naudojama tik informacijos gavimui.
7. **SSL (*angl.* Secure Sockets Layer)** – kriptografinis protokolas, skirtas informacijos, sklindančios internete apsaugojimui šifruojant.
8. **Karkasas** (*angl.* framework) – rinkinys tarpusavyje bendraujančių klasių, kurios aprašytos taip, kad programuotojui padėtų sukurti sistemos dizainą, paremtą tam tikrais konceptais.
9. **Klasė (*angl.* class)** – į objektus orientuotame programavime klasė yra naudojama, kad būtų galima pernaudoti tuos pačius kintamuosius ir reikšmes ir ji turi visus reikalingus metodus aprašytus. Ji naudojama tam, kad būtų atliekama tam tikra funkcija.
10. **API (*angl. Application programming interface*)** – tai yra aplikacijų programavimo sąsaja. Sąsaja, kurią suteikia kompiuterinė Sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis.
11. **JavaScript** – tai yra scenarijų programavimo kalba, besiremianti prototipų principu. Dažniausiai kalba naudojama internetinių puslapių interaktyvumo realizacijai.
12. **XSS (*angl. Cross-site scripting*)** – Informacinių technologijų sistemų pažeidžiamumas, dažniausiai aptinkamas tinklalapiuose, kurie leidžia įterpti papildomą programinį kodą į vartotojo peržiūrimą tinklalapį. Toks pažeidžiamumas atsiranda dėl nepakankamo įvedamos informacijos filtravimo.
13. **CSRF (*angl.* Cross-Site Request Forgery) –** atakos tipas, kai piktavališkas tinklalapis, tinklaraštis, žinutė arba programa priverčia vartotojo naršyklę atlikti tam tikrus veiksmus (prieš vartotojo valią) į kitą patikimą svetainę, prie kurios tuo metu vartotojas yra prisijungęs
14. **PHP (angl. *Hypertext Preprocessor*)** - plačiai paplitusi dinaminė interpretuojama programavimo kalba, sukurta 1995 m. ir specialiai pritaikyta interneto svetainių kūrimui.
15. **URL (angl. *Uniform resource locator*)** – tai yra universalus informacijos šaltinio adresas, kuris skirtas patogiai nurodyti tikslią informacijos vietą internete.
16. **Serveris** - specialios paskirties kompiuteris ar programa, skirta kitų kompiuterių (klientų) aptarnavimui (paprastai nėra skirta asmeniniam naudojimui).
17. **OWASP (angl. *Open Web Application Security Project*) –** tai yra pelno nesiekianti organizacija, kurios pagrindinis tikslas yra taikomosios programinės įrangos gerinimas. Šiame projekte pateikiami pagrindinės, dažniausiai pasitaikančios, atakų rūšys prieš programinę įranga. Pateikiami galimi apsisaugojimo būdai bei pateikiami metodiniai nurodymai, kaip įvertinti kuriamos sistemos saugumą.
18. **jQuery** – tai yra JavaScript programavimo kalbos biblioteka, kurioje aprašytos dažniausiai naudojamos funkcijos, kurios gali būti panaudotos rašant sistemą. Tokiu būdu siekiama supaprastinti ir pagreitinti sistemų kūrimą.

19. **DOM (angl. *Document Object Model*)** – taip yra nuo programavimo kalbos nepriklausantis susitarimas, aprašantis objektus, ir veiksmus su jais HTML, XHTML ir HML dokumentuose.

20. **Python** – yra bendro naudojimo programavimo kalba sukurta 1991 metais, jos pagrindinis tikslas buvo kuo patogesnis skaitomumas programuotojui (griežtesnė struktūra) ir kad būtų galima gauti norimą rezultatą su kiek įmanoma trumpesniu programiniu kodu.

21. **Apache** – labiausiai paplitęs atviro kodo HTTP serveris. Jį galima naudoti tiek Linux tiek Windows operacinėse sistemose.

22. **Sausainukas (*angl*. cookie) –** informacija atsiųsta iš serverio vartotojui, kuri saugoma pas vartotoją įrenginyje. Dažniausiai naudojama, kad talpinti informaciją apie patį vartotoją arba su jo sesija susijusius duomenis.

ĮVADAS

Sparčiai populiarėjant technologijoms jos tampa neatsiejama ir svarbia kiekvieno žmogaus gyvenimo dalimi. Kuriamos sistemos leidžiančios vis daugiau dalykų pasiekti nepaliekant namų. Spartus informacinių technologijų populiarėjimas sąlygoja ir tai, kad vis daugiau mūsų asmeninės informacija tampa prieinamos internete. Įmonės nenorėdamos atsilikti nuo technologijų ir prarasti konkurencingumo taip pat stengiasi optimizuoti kiek įmanoma daugiau darbų ir vis daugiau svarbių duomenų perkelia į serverius ar debesų kompiuteriją. Matydami tokias tendencijas nusikaltėliai taip pat neleidžia laiko veltui. Atsiranda vis daugiau labai gerai kvalifikuotų žmonių bandančių perimti įvairius duomenis nepriklausančius jiems. Paskutiniu metu stebint didelį įsilaužėlių suaktyvėjimą sistemų kūrėjai stengiasi neatsilikti taikydami vis didesnius saugumo reikalavimus sistemoms. Daugiausia dėmesio skiriama internetinio tipo sistemos, prie kurių priėjimą turi kiekvienas. Prieš tokias sistemas yra nukreipta daug įvairaus tipo atakų kurios yra orientuotos į tam tikras sistemos spragas. Vienos tokių yra CSRF atakos. Šiame darbe plačiau panagrinėsiu šio tipo atakas ir galimus apsaugos metodus nuo jų. Bus bandoma išsiaiškinti kiekvieno apsaugos metodo pliusus ir minusus bei pasiūlyti kaip būtų galima papildyti esamus apsaugos metodus.

Šio tyrimo objektas yra esamų apsaugos metodų nuo CSRF tipo atakų tyrimas ir tobulinimas.

Šio tyrimo tikslas yra išanalizuoti esamus apsaugos metodus nuo CSRF atakų, įvertinti jų privalumus ir trūkumus bei pasiūlyti naują apsaugos metodą kuris galėtų padidinti sistemų saugumą.

Tyrimo uždaviniai:

1. Išanalizuoti CSRF atakų anatomiją, kaip jos gali būti realizuotos..
2. Išanalizuoti apsaugos gerinimą jau esančioje sistemoje.
3. Išanalizuoti apsaugos metodus naujai kuriamoje sistemoje.
4. Išanalizavus visu esamus metodus, įvertinus juos ir pasiūlyti naują būdą kaip juos būtų galima pagerinti.
5. Išbandyti naujai pasiūlytą apsaugos metodą kuris galėtų padidinti sistemų apsaugą.
6. Įvertinti gautus rezultatus ir apibendrinti naujos apsaugos efektyvumą.

Darbas sudarytas iš keturių pagrindinių skyrių: „Informacinių sistemų apsaugos problemos“, „Apsauga nuo CSRF atakų“, „Apsaugos nuo CSRF atakų optimizavimo tyrimas“, „Papildomos apsaugos nuo CSRF atakų realizavimas“. Pirmajame skyrelyje trumpai apžvelgiamos galimos grėsmes internetinio tipo sistemoms. Antrajame skyrelyje analizuojami apsaugos metodai nuo CSRF tipo atakų. Nagrinėjami esami apsaugos metodai, dažniausiai apsaugai naudojami karkasai. Trečiajame skyrelyje modeliuojamas papildomos apsaugos metodas, padėsiantis pagerinti sistemų apsaugą. Aprašomas metodo principas ir kuo jis galėtų būti naudingas. Paskutiniajame aprašomas naujo metodo testavimas. Aprašomi gauti rezultatai ir metodo efektyvumas bei pateikiamos išvados.

1. INFORMACINIŲ SISTEMŲ APSAUGOS PROBLEMOS

Bėgant metams sparčiai išaugo interneto vartotojų skaičius. Tinklalapio „internetlivestats“ [1] pateiktais duomenimis apie 40% viso pasaulio populiacijos (apie 3mlrd. žmonių) turi prieigą prie interneto ir ja reguliariai naudojasi, kai tuo tarpu palyginimui, 1995 metais prieigą prie interneto turėjo vos 1% visų pasaulio žmonių (apie 44 milijonus žmonių). Internetas tampa neatsiejamas nuo mūsų kasdienio gyvenimo dalis. Kiekvieną dieną vis daugiau žmonių ieško informacijos, leidžia laisvalaikį bei bendrauja su draugais. Vis daugiau įmonių pradeda teikti paslaugas internetu. Kad visa tai taptų įmanoma, informacija turi būti kažkur talpinama, todėl buvo sugalvotos informacinės sistemos, kitaip dar vadinamos tinklalapiais. Pirmasis tinklalapis buvo sukurtas daugiau nei prieš 20 metų. Jis buvo labai primityvus, lyginant su dabartinėmis internetinėmis sistemomis. Jame buvo patalpinta informacija apie pasaulinį interneto tinklą (*angl.* World Wide Web). Nuo tada tinklalapiai gerokai pasistūmėjo į priekį, tapo daug sudėtingesni. Informacija nebėra statiškai saugoma sistemos kode, todėl realizuotos galimybės vartotojui paprastai keisti sistemoje pateikiamą informaciją, paprastiems veiksmams nebereikalingas programuotojo įsikišimas, tačiau tai reiškia, kad dažniausiai sistema yra kuriama nebe vieno žmogaus, o komandos arba kelių komandų. Dėl daugelio žmonių įsikišimo į projektavimą ir realizavimą atsiranda sistemos ne vientisumas ir tampa labai sudėtinga užtikrinti sistemos spragų nebuvimą.

Šią spragą bando išnaudoti internetiniai įsilaužėliai (*angl.* hacker). Paprastai tokie žmonės labai gerai išmano savo darbą (puikios programavimo žinios), nors būna ir tokių, kurie naudojasi kitų žmonių parašytomis programomis, tačiau pastarieji dažniausiai neturi konkretaus tikslo, tiesiog nori pabandyti kažką naujo. [2] Sistemas bandoma nulaužti dėl įvairiausių priežasčių [3], tokių kaip:

- Noras pasirodyti geresniu už kitus (siekiama įrodyti kaip gerai atakuotojas išmano savo darbą).
- Noras pasipelnyti (panaudojama kitų žmonių informacija arba vagiami pinigai).
- Noras atkeršyti (dažniausiai kyla dėl asmeninių konfliktų su tam tikra organizacija).
- Šnipinėjimo tikslais (konkuruojančios korporacijos arba valstybinės institucijos siekiančių kompetencinio pranašumo).
- Piktavališki tikslai sutrikdyti sistemą.

Nesvarbu kokie yra tikslai, tačiau pavykusios atakos gali turėti rimtų pasekmių sistemų savininkams.

1.1 Dažniausiai naudojamos atakos

Atakos naudojamos prieš informacines sistemas gali būti įvairių rūšių:

- Siekis sustabdyti sistemos veikimą.
- Informacijos vagystė.
- Sistemos veikimo keitimas kitokiu nei buvo numatyta.
- Informacinės sistemos vaizdo pateikimo iškraipymas.

Žemiau trumpai pateiksiu tokių atakų pavyzdžių.

Puikus sistemos veikimo sustabdymo pavyzdys būtų DDoS tipo ataka prieš Spamhaus organizaciją. Spamhaus yra pelno nesiekianti organizacija, kuri kaupia juodąjį brukalų (*angl*. Spam) siuntėjų sąrašą. Prieš šia organizaciją buvo pradėta didžiausia kada nors buvusi DDoS tipo ataka pasaulyje. Tinklo apkrautumas padidėjo nuo 10G iki 30G, kas sąlygojo sistemų nepasiekiamumą. Įdomu tai, kad daugiau nei dešimt milijonų interneto vartotojo buvo paveikti šios atakos netiesiogiai. Kadangi tai buvo labai didelio mąsto DDoS ataka, naršymo greitis gerokai sulėtėjo net naršant kituose, nieko bendro neturinčiuose su šia ataka, tinklalapiuose [4].

Informacijos vagystės taip pat nėra retai sutinkamos tarp įvairių įsilaužimo atvejų. Didelėms kompanijos (ypač bankinėms institucijoms) bet kokios informacijos saugojimas turėtų būti pagrindinis dalykas, bet kaip praktika rodo dažnai būna priešingai. Taip buvo ir „TJX companies“ korporacijos atžvilgiu. 2006 metais kompanija surado, kad apie 94 milijonus kredito kortelių duomenų buvo pavogti. Įsilaužėliai pasinaudojo prasta organizacijos vidinio tinklo apsauga ir prastu duomenų šifravimu. Kadangi

organizacija nebuvo stipriai orientuota į duomenų apsaugą, ji įsilaužimą sugebėjo pastebėti tik po daugiau nei pusės metų [5].

Vienas labiausiai išgarsėjusių sistemos pakeitimo atvejų, buvo 2005 metais išnaudotas „MySpace“ pažeidžiamumas. Atakos autorius yra Samy Kamhar, o vėliau jo vardu pavadintas ir pats “kirminas“ Samy. Jis parašė pirmąjį save sugebantį platinti kompiuterinį kirminą išnaudodamas nepakankamą apsauga nuo XSS atakų „MySpace“ tinklalapyje. Kiekvienam apsilankusiam Samy profilyje, kirminas automatiškai aktyvuodavosi įrašydamas „Samy is my hero“ (angl. Samy yra mano herojus) aukos profilyje ir pridėdavo autorių prie draugų sąrašo. Negana to, kenksmingas kodas buvo perkeliamas ir į aukos profilį, tokiu būdu, jei aukos profilį aplankydavo kitas vartotojas, jo profilyje taip atsirasdavo ta pati žinutė ir jis automatiškai pridėdavo Samy prie draugų sąrašo ir gaudavo ta patį žalingą kodą. Tai sukėlė grandininę reakciją ir tuo metu tai tapo sparčiausiai plintančiu kirminu istorijoje. Per 20 valandų vartotojas Samy jau turėjo daugiau nei milijoną draugų MySpace svetainėje. Vėliau, kai MySpace tinklalapis išsitaisė pažeidžiamumus, Samy pateikė visą informaciją, kaip jam pavyko įvykdyti šią ataką. [6]

Pagrindinis saugojimo objektas informacinių sistemų atžvilgiu yra vertinga informacija. Pašaliniam asmeniui priėjus prie šių vertybių kyla didelis pavojus, kad su šia informacija bus atlikti nereglamentuoti veiksmai. Todėl pirmaeiliu uždaviniu visada išliks informacijos sistemų apsauga nuo nesankcionuoto jų panaudojimo, o žemesnį prioritetą turės pačios informacijos saugumo užtikrinimas šiose sistemose. Galimi pavojai turi būti vertinami pagal tai, kaip jie gali paveikti tris pagrindinius saugumo elementus - sistemos resursų vientisumą, konfidencialumą ir prieinamumą.

Dažniausiai pasitaikančios atakos nukreiptos prieš informacines sistemas yra šios:

- CSRF
- XSS
- Injekcija
- Direktorijų klastojimas
- DoS ataka
- DDoS ataka

Žemiau pateikiamas trumpas kiekvienos atakos aprašymas.

CSRF – atakų metu kaip jaukas yra naudojamas nekaltai atrodantis puslapis, verčiantis autentifikuotos aukos naršyklę keistis informacija su pažeidžiama Web aplikacija. Pasisekusi CSRF ataka gali pažeisti vartotojo duomenis, vykdyti neautorizuotas operacijas, administratoriaus privilegijos atveju, net sudarkyti visą Web sistemą. Prieš nagrinėjant CSRF ataką reikia suprasti, kaip atrodo pažeidžiama Web užklausa.

XSS – pažeidžiamumas leidžia įsilaužėliui įterpti naršyklėje vykdomą kodą į kitų vartotojų peržiūrimą Web puslapį. Šiomis dienomis tai ypač dažnai išnaudojamas pažeidžiamumas. XSS išnaudojimai paprastai remiasi tos pačios kilmės politikos koncepcija (*angl.* Same Origin Policy), kuri iš esmės leidžia tik to paties puslapio resursams sąveikauti tarpusavyje be jokių specialių apribojimų. XSS atakos metu savavališkai įterptas kodas yra vykdomas nieko nenutuokiančio vartotojo naršyklėje, šis kodas paprastai naudojasi papildomomis privilegijomis (pagal tos pačios kilmės politiką). XSS yra išnaudojamas tokiems tikslams, kaip vartotojo sesijų užgrobimas (*angl.* session hijacking), Web puslapių sudarkymas, kenkėjiško turinio įterpimas, vartotojų nukreipimas į kitus puslapius/turinį ir t.t.

Injekcija - pažeidžiamumo išnaudojimas interpretatoriui vykdant pateiktuose duomenyse įterptą kodą. Viena dažniausiai pasitaikančių injekcijos pažeidžiamumo rūšių – SQL injekcija, kurios metu įvestyje yra įterpiama SQL išraiškos dalis ar forma, siekiant įvykdyti naujai suformuotą išraišką (pvz. iš duomenų bazės gauti informaciją apie vartotojus, įterpti kenkėjiškus įrašus ir pan.). Paprastai tokio tipo pažeidžiamumai lengviausiai aptinkami analizuojant programos kodą (ieškoma vietų, kuriose yra interpretuojama galimai nepatikima įvestis).

Direktorijų klastojimas - pažeidžiamumas leidžia įsilaužėliui prieiti prie kitų failų esančių už informacinės sistemos katalogo. Jeigu vartotojo keliami failų vardai nėra tikrinami, jis gali gauti prieigą prie kitų failų išeities kodo arba prie kitų svarbių sistemos failų, pvz. Slaptažodžių failas.

DoS ataka – (*angl.* Denial of Service) metodas kai užpuolikas bando sudaryti tokį dideli srautą tam tikrai mašinai ar tinklo resursui, kad ji taptų neprieinama nei vienam vartotojui. Dažniausias taikinys yra serveris, kuriame patalpinta informacinė sistema.

DDoS ataka – (*angl.* Distributed Denial of service) toks pats metodas kaip ir atliekant DoS ataką, tik užklausos siunčiamos ne iš vieno šaltinio, o iš daugelio užkrėstų kompiuterių, jų savininkams apie tai nieko nežinant.

Paskutinių dviejų paminėtų atakų (DoS ir DDos) pagrindinis tikslas yra sustabdyti sistemos veikimą. Jos yra daugiau protestuojamojo pobūdžio (panašus principas kaip mitingai gatvėse). Nors atakų pagrindinis tikslas dažniausiai būna sustabdyti sistemos veikimą, tačiau kartais galima sistemą privesti iki tokios ribos, kai sistema, prieš kurią nukreipta ataka, nesugeba adekvačiai skirtyti resursų visoms ateinančioms užklausoms ir pereina i kritinę veikimo būseną, kurios nenumatė programuotojai. Tai gali reikšti, kad programa gali leisti nenumatytą prieigą atakuotojui prie sistemos resursų arba duomenų, tokiu būdu atskleidžiant jautrią informaciją.

Kiekvienais metais augant naujų sistemų skaičiui, tuo pačiu atsiranda vis daugiau pažeidžiamų sistemų. Pamažu skiriamas vis didesnis dėmesys ir sistemų apsaugojimui. Kaip bebūtų gaila, šiai dienai skiriamas dėmesys yra vis tiek per mažas. Atsirandant poreikiui, atsiranda ir į saugumą labiausiai orientuotų įmonių. 2014 metais buvo išleista daugiau nei 70 milijardų dolerių įvairių sistemų saugumo didinimui ir saugumo įrankių kūrimui, tačiau net ši didžiulė pinigų suma nepadeda pristabdyti įsilaužėlių kiekio didėjimo. Per tuos pačius metus, finansinės institucijos patyrė daugiau nei 400 milijardų dolerių žalą, dėl netinkamos saugumo politikos. [7] Didžiausią dalį sudarė pačių institucijų darbuotojai, kurie dažniausiai nepakankamai atsižvelgia į galimai kylančias grėsmes ir naršo nepatikimose svetainėse bei atidarinėja neaiškias nuorodas atsiunčiamas elektroniniu paštu. Dabartiniai virusai kuriami labai sudėtingi, dažnas jų orientuotas į greitą vystymąsi. Vienam darbuotojui paspaudus ant netinkamos nuorodos, greitai visi įmonės kompiuteriai ar net serveriai gali tapti užkrėsti. 2015 metų saugumo kompanijos publikuotame pranešime pateikiama praėjusių metų statistika. Pasitelkę automatizuotą sistemų tikrinimą, jie patikrino pusę milijono didžiausių svetainių sąrašą pagal „alexa.com“. Šio sąrašo viršuje yra tokios svetainės kaip „google“, „facebook“ ar „amazon“. Gauti rezultatais yra stebinantys:

- Viena iš trijų svetainių yra pažeidžiama kokio nors tipo atakoms
- Viena iš penkių svetainių naudoją programinę įranga su žinomais pažeidžiamumais
- Viena iš dvidešimties buvo užfiksuota kaip siuntinėjanti brukalus arba esanti tam tikro „botneto“ dalis

Bendru atveju apie 30% visų jų nagrinėtų sistemų yra potencialiai pavojingos, t.y. nėra pilnai apsaugotos [7].

Facebook yra didžiausias socialinis tinklas pasaulyje. Kiekvienais metais jie organizuoja renginius, kurių metu kviečia geriausius saugumo specialistus, kad jie ieškotų pažeidžiamumų jų tinklalapyje ir siūlo premijas jei jiems pasiseka. Šiais metais buvo rastas dar vienas CSRF ir sesijos pažeidžiamumas sistemoje. Bet kuris vartotojas galėjo ištrinti kito vartotojo parašytus komentarus. Trinant komentarą buvo tikrinami du laukai nusiųsti į serverį. Šiuos abu laukus buvo galima sužinoti paspaudus „like“ mygtuką svetainėje, tuo pat metu stebint siunčiamus paketus savo kompiuteryje. Gavus šias dvi reikšmes, užtekdavo sugeneruoti užklausą iš savo kompiuterio, serveriui pateikiant būtent šias dvi reikšmes kartu su kitais atributais. [8]

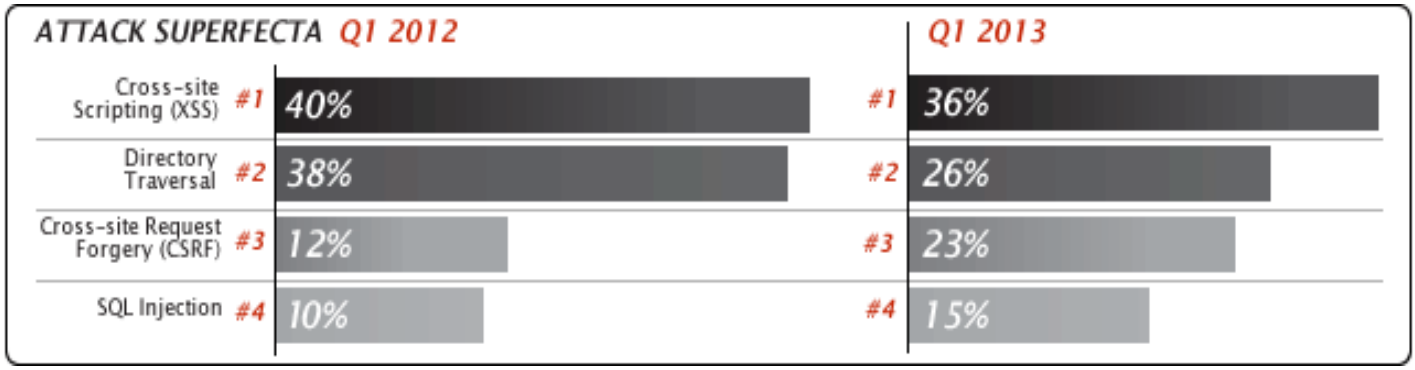
Toliau dokumente nagrinėsime plačiau CSRF tipo atakas. Būtent šio tipo atakos yra vienos dažniausiai pasitaikančių atakų prieš internetinio tipo sistemas. Šio tipo atakos prieš kelerius metus skaitytos kaip visai nepavojingos atakos. Tik vėliau į jas pradėta žiūrėti rimtai, kai ėmė pasirodyti daugiau pranešimų apie sėkmingai įvykdytas CSRF tipo atakas. Šiai dienai programuotojai dažniausiai nuvertina šio tipo atakų svarbą, nes reikalingas tarpinė sistema į kurią turėtų užeiti prisijungę vartotojai, tokiu būdu

būtų galima sugeneruoti ataką. Nuvertindami šio tipo atakas programuotojai palieka dideles saugumo skyles, kurios dažniausiai gali paveikti daugelio vartotojų duomenis sistemoje, o tai sąlygotų dideli reputacijos ir galimai piniginį nuostolį. Pasirinkau nagrinėti šį atakų tipą dėl to, nes jos yra vienos dažniausiai sutinkamų pažeidžiamumų dabartinėse sistemos. Programuotojai dažniausiai skiria per mažai dėmesio sistemos apsaugai nuo tokio tipo atakų, neįvertindami galimos žalos, kurią patirtų vartotojai tinkamai neapsaugojus sistemos. Plačiau šio tipo atakas aptarsiu kitame skyriuje.

1.2 CSRF atakos

Pirmą kartą CSRF atakos buvo pastebėtos 2001 metais. Kiekvienais metais atakų skaičius vis auga. Tai sąlygoja tokios tipo atakų kaip XSS ir CSRF paprastumas. Jų realizavimas (programinės pusės atžvilgiu) nėra labai sudėtingas. Galima rasti daugybę pavyzdžių internete, taip pat jos yra automatizuotos, todėl įsilaužėliui užtenka kartą suprojektuoti ir suprogramuoti įsilaužimą realizuojančią užklausą. SQL injekcijos atakos yra šiek tiek sudėtingesnės, reikalauja geresnio sistemos išmanymo, bei geresnių įsilaužėlio įgūdžių.

„Firehost“ yra vienas didžiausių debesų kompiuteris paslaugų teikėjų. Kompanija orientuojasi į padidinto saugumo debesų paslaugų teikimą. Kiekvienais metais jie pateikia duomenis apie per paskutinius metus aktualiausius pažeidžiamus informacinėms sistemoms. Paskutiniais pateiktais duomenimis, nuo 2012 metų pirmo ketvirčio ir 2013 metų pirmo ketvirčio išaiškintų CSRF atakų skaičius išaugo daugiau nei dvigubai, ir tai sudarė 23% visų jų atremtų atakų (žr. 1 pav.).



Pav. 1 Firehost atremtų atakų kiekis procentais

Iš paveikslėlio esančio aukščiau matome, kad nemažai informacinių sistemų projektuotojų ir programuotojų, pamiršta apie apsaugą nuo CSRF atakų, o įsilaužėliai nesnaudžia automatizuodami atakas ir ieškoti naujų klaidų sistemose.

Apsisaugojimo nuo CSRF atakų svarbą, parodė 2008 metais Bill Zeller ir Ed Felten atliktas tyrimas **[9]**. Jie atrado, kad keturios dideles, tarpusavyje nesusijusios, sistemos turi rimtų trūkumų:

- Ingdirect.com – internetinis bankas, jie surado, kad galima kurti naujus banko vartotojus, kitų vartotojų vardu, taip buvo atrasta spraga kuri leido pervesti vartotojų pinigus kitiems vartotojams. Tai buvo vienas iš labai retų atvejų, kai finansinė institucija, turėjo tokią spragą.
- Youtube.com - visiems gerai žinoma vaizdinės medžiagos dalinimosi sistema. Buvo atrasta CSRF spraga, leidusi įsilaužėliui atlikti beveik visus veiksmus kuriuos vartotojas gali atlikti sistemoje. Buvo galima pridėti vartotoją į draugų sąrašą, siųsti pranešimus jo vardu, priversti “pasidalinti“ filmukais, su vartotojo kontaktuose esančiais asmenimis, pridėti filmukų į vartotojo „peržiūrėsiu vėliau“ sąrašą ir pan.
- Metafilter.com – pažeidžiamumą sistemoje leido įsilaužėliui pilnai ‘‘užvaldyti‘‘ kito vartotojo profilį. Pradžioj nusiųsdavo užklausą, kad pakeisti vartotojo elektroninį adresą į įsilaužėlio, po to nusiųsdavo dar vieną užklausą ‘‘pamiršau slaptažodį‘‘, tam, kad atakuotojas gautų naują aukos slaptažodį į savo elektroninį paštą.

• Nytimes.com – pažeidžiamumas sistemoje leido įsilaužėliui gauti prisijungusio vartotojo elektroninį paštą. Tokiu būdu atakuotojas galėjo susirinkti daugybės žmonių elektroninius paštus, pvz. Reklamini laiškų (*angl.* spam) siuntinėjimui. „Nytimes“ laikydavo vartotojus prijungtus daugiau nei metus, nebent patys vartotojai atsijungdavo, todėl tikimybė surasti vartotojų prisijungusiu prie sistemos buvo labai didelė.

Visi aukščiau paminėti sistemų pažeidžiamumai buvo išspręsti beveik iškart. Sistemų kūrėjai buvo suinteresuoti spręsti problemą iškart. Laimei tyrimą atliko ne piktavaliai įsilaužėliai, o akademiniai darbuotojai, kurie domėjosi CSRF pažeidžiamumais, todėl jie iškart pranešė apie savo atradimus sistemų kūrėjams, kad šie galėtų atitinkamai reaguoti. Padariniai galėjo būti katastrofiški (ypač „ingdirect“ atveju), jei šiuos pažeidžiamumus pirmiau būtų atradę tikrieji įsilaužėliai.

1.3 CSRF atakų požymiai

Kaip jau minėta ankstesniame skyrelyje, CSRF atakos tikslas yra panaudoti aukos aktyvią sesiją kitoje svetainėje (pvz. banke) ir atlikti veiksmus prieš jo valią, jam to net nesuvokiant.

Panagrinėkime paprastą pavyzdį, kuriame naudojama GET tipo užklausa. Sakykime auka yra prisijungęs prie banko sistemos, ir tuo metu jis užeina į mėgstamą forumą paskaityti naujienų. Ten įsilaužėlis patalpinęs paveikslėlį, kurio šaltinis yra ne tikra nuoroda į paveikslėlį, o GET užklausa į banką, kad iš aukos sąskaitos būtų pervedami pinigai įsilaužėliui:

```
<img src="http://bank.example.com/withdraw?account=auka&amount=1000000&for=isilauzelis">
```

Auka nieko nenutuokdamas paspaudžia ant paveikslėlio, jo naršyklė nusiunčia GET užklausą į banką, o kadangi jis ten vis dar yra prisijungęs (yra aktyvi sesija), bankas autentifikuoja vartotoją pagal sesiją ir mano, kad jis yra tikrai nusiteikęs pervesti pinigus įsilaužėliui, todėl įvykdo užklausą ir pinigai yra pervedami įsilaužėliui į banko sąskaitą.

Anksčiau buvo manyta, kad užteks sistemose nenaudoti GET užklausų, kadangi jas labai paprasta imituoti, nes formos reikšmės tiesiog pridedamos prie adreso laukelio, tačiau kaip vėliau paaiškėjo, POST užklausas ne ką sunkiau suklastoti.

Panagrinėkime pavyzdį kuriam CSRF ataka būtų realizuota POST metodo pagalba. Kaip ir ankstesniame pavyzdyje, auka turi būti prisijungęs prie pažeidžiamos sistemos. Šiuo atveju tai bus sistemos dalis, kurioje vartotojas gali matyti ir keisti savo asmeninius duomenis. Žemiau pateiktas pavyzdys kaip turi atrodyti užpildyta forma:

```
<input type="text" value="Vardenis Pavardenis" name="urname" style="width:200px">
<input type="text" value="1254-5498-5233-5569" name="ucc" style="width:200px">
<input type="text" value="v.pavardenis@example.com" name="uemail" style="width:200px">
<input type="text" value9="+370 612 1235" name="uphone" style="width:200px">
<textarea wrap="soft" name="uaddress" rows="5" style="width:200px">Vilnius, Lietuva</textarea>
```

Kai vartotojas paspaudžia mygtuką siųsti, HTTP POST užklausa nusiunčiama į serverį. Kadangi pažeidžiama sistema neturi jokios apsaugos nuo CSRF atakų, įsilaužėlis pasinaudodamas paslėptu iframe gali pažeidžiamos sistemos formą įsidėti savo tinklalapyje:

```
<iframe src="http://www.vulnweb.com/updateif.php" style="display:none"></iframe>
```

Ši eilutė užkrauna pažeidžiamą puslapį (ta vietą kur įvedami duomenys), kas kartą kai koks nors vartotojas prisijungia. Įsilaužėlio puslapis turi užpildytą formą, kuri kam nors prisijungus iškarto bando

nusiųsti HTTP POST užklausą į pažeidžiamą sistemą, su jam (įsilaužėliui) naudingais duomenims. Formos pavyzdys:

```
<body onload="document.getElementById('f').submit()">
<form id="f" action="http://testphp.vulnweb.com/userinfo.php" method="post" name="form1">
<input name="urname" value="attacker'svalue">
<input name="ucc" value=" attacker'svalue">
<input name="uemail" value=" attacker'svalue">
<input name="uphone" value=" attacker'svalue">
<textarea name="uaddress" wrap="soft"><attacker'svalue></textarea>
<input name="update" value="update">
</form>
</body>
```

Į formą duomenys gali būti užkrauti iš failo, automatiškai išsiuntus užklausą, likęs įsilaužėlio puslapis užkraunamas. Šios operacijos atliekamos paslėptame frame objekte, todėl auk nieko negali nutuokti.

Sistemai kuri bando apsiginti nuo CSRF labai sunku atkirti kurios GET ir POST užklausos yra padirbtos, o kurios yra tikrai vartotojo *siųst*os, nes sistemos atžvilgiu jos visos ateina iš prisijungusio vartotojo naršyklės.

Norint sėkmingai įvykdyti CSRF tipo ataką, neužtenka vien tik programavimo ir sistemos išmanymo žinių. Norint, kad viskas pavyktų, reikia aukas priversti paspausti ant nuorodos arba apsilankyti tam tikroje svetainėje, kurioje paslėptas jūsų piktavališkas kodas. Dažniausiai naudojami būdai:

- Elektroniniai laiškai
- Socialiniai tinklai
- Nuorodos įdedamos į el. dokumentus

Taigi, kaip matėme iš aukščiau pateiktų pavyzdžių, prieš sistemą, kuri nesisaugo nuo CSRF atakų, labai nesunku įvykdyti tokio tipo atakas. Darant sistemas programuotojai ir projektuotojai dažnai neapgalvoja, kad visų įmanomų įsibrovimo galimybių, ir vertėtų turėti galvoje tai, kad net minimaliausi duomenų keitimai ar atskleidimai gali turėti labai didelių pasekmių sistemos vartotojams.

1.4 CSRF ir XSS panašumai ir skirtumai

Paskutiniu metu daug dėmesio skiriama XSS atakoms. Šio tipo atakų metu įsilaužėlis patalpina piktavališką kodą jo pasirinktoje, neapsaugotoje informacinėje sistemoje. Dažniau kodas būna parašytas Java script programavimo kalba. Pavyzdžiui, sistema gali leisti vartotojams patalpinti komentarus forume ir netikrinti tų duomenų. Tokių būdu įsilaužėlis patalpins piktavališką kodą ir kas kartą kai koks nors žmogus atidarys svetainę (forumą), JavaScript kodas bus vykdomas. Tokio tipo atakos nukreiptos prieš sistemos, kurioje patalpintas žalingas kodas, naudotojus. Jei įsilaužėliui pavyksta patalpinti kodą sistemoje, tada kas kartą tą kodą vykdant, jis vykdomas prisijungusio žmogaus teisėmis, taip pat įsilaužėlis galėtų *siųst*i ir gauti užklausas iš bet kurio puslapio sistemoje ir galėtų perskaityti vartotojų sausainukus (*angl.* cookies). Norint apsisaugoti nuo tokio tipo atakų, sistema privalo filtruoti visą įvedamą informaciją taip, kad įsilaužėlis nesugebėtų įterpti jokio kodo, tai reiškia, kad jo įterpiami duomenys butų skaitomi kaip paprastas tekstas arba jam neleisti tam tikru simbolių kombinacijų.

XSS ir CSRF atakos skiriasi tuo, kad XSS atakoms reikalingas JavaScript, kai tuo tarpu CSRF atakoms jis nėra privalomas. XSS atakoms reikalingas, kad sistema priimtų žalingą kodą, o CSRF atakų atveju, žalingas kodas talpinamas trečių šalių sistemose. Filtravimas įvedamų duomenų padėtų apsisaugant nuo žalingo kodo įterpiamo į sistemą, tačiau neapsaugos nuo kodo, kuris randasi trečių šalių sistemose, todėl apsauga nuo XSS atakų, neapsaugo sistemos nuo CSRF atakų.

1.5 Informacinių sistemų saugos problemų išvados

Kiekvienais metais augant naujų sistemų skaičiui, tuo pačiu atsiranda vis daugiau pažeidžiamų sistemų. Pamažu skiriamas vis didesnis dėmesys ir sistemų apsaugojimui. Kaip bebūtų gaila, šiai dienai skiriamas dėmesys yra vis tiek per mažas. Atsirandant poreikiui, atsiranda ir į saugumą labiausiai orientuotų įmonių. 2014 metais buvo išleista daugiau nei 70 milijardų dolerių įvairių sistemų saugumo didinimui ir saugumo įrankių kūrimui, tačiau net ši didžiulė pinigų suma nepadeda pristabdyti įsilaužėlių kiekio didėjimo. Per tuos pačius metus, finansinės institucijos patyrė daugiau nei 400 milijardų dolerių žalą, dėl netinkamos saugumo politikos. [7] Didžiausią dalį sudarė pačių institucijų darbuotojai, kurie dažniausiai nepakankamai atsižvelgia į galimai kylančias grėsmes ir naršo nepatikimose svetainėse bei atidarinėja neaiškias nuorodas atsiunčiamas elektroniniu paštu. CSRF tipo atakos yra vienos dažniausiai pasitaikančių atakų prieš internetinio tipo sistemas. Šio tipo atakos prieš kelerius metus skaitytos kaip visai nepavojingos atakos. Tik vėliau į jas pradėta žiūrėti rimtai, kai ėmė pasirodyti daugiau pranešimų apie sėkmingai įvykdytas CSRF tipo atakas. Pirmą kartą CSRF atakos buvo pastebėtos 2001 metais. Kiekvienais metais atakų skaičius vis auga. CSRF atakos tikslas yra panaudoti aukos aktyvią sesiją kitoje svetainėje (pvz. banke) ir atlikti veiksmus prieš jo valią, jam to net nesuvokiant. Sistemai kuri bando apsiginti nuo CSRF labai sunku atkirti kurios GET ir POST užklausos yra padirbtos, o kurios yra tikrai vartotojo *siųst*os, nes sistemos atžvilgiu jos visos ateina iš prisijungusio vartotojo naršyklės.

2. APSAUGA NUO CSRF ATAKŲ

Kaip buvo paminėta pirmame skyrelyje, apsauga nuo CSRF atakų yra būtina. Nuo šių atakų galima dvejopa apsauga – tiek iš sistemos pusės tiek iš vartotojo. Efektyvesnė apsauga yra saugojimas pačios sistemos, o ne kiekvienam vartotojui saugotis atskirai. Dažniausiai vartotojai yra nepakankamai kvalifikuoti, kad galėtų identifikuoti potencialias grėsmes, taip pat apsaugos metodai naudojami vartotojo pusėje nėra tokie efektyvus, kaip efektyviai apsaugota pati sistema. Kuriant bet kokią informacinę sistemą, svarbiausias prioritetas turėtų būti sistemos saugumas. Vartotojas neturi rūpintis papildomais savęs apsaugojimo būdais. Sekančiuose skyreliuose labiau detalizuosiu apsisaugojimo būdus skirtus tiek pačio vartotojo apsaugai, tiek apsaugai iš serverio ar pačios sistemos.

2.1 Apsauga nuo CSRF atakų egzistuojančioje sistemoje

Aktualiausia apsauga dabartiniu metu yra jau sukurtoms sistemos. Jau esančių sistemų apsaugojimas dabar tampa dideliu galvos skausmu tiek įmonėms tiek privatiems asmenims. Anksčiau kuriamoms sistemoms buvo taikomi visai kiti reikalavimai nei dabar. Per eilę metų technologijos gerokai pažengė į priekį, o tuo pačiu neatsiliko ir įsilaužėliai. Kiekviena įmonė ar asmuo su šia problema bando kovoti jiems priimtinais būdais. Labiausiai paplitęs sprendimo būdas yra naudoti karkasus (*angl.* framework). Iš esmės tai yra iš anksto jau paruošti programinio kodo blokai. Juos galima įterpti į bet kurią jau egzistuojančią sistemą. Pridėjus juos prie sistemos, atitinkamai galima modifikuoti ir pačios sistemos kodą, kad apsaugai būtų naudojami metodai aprašyti karkasuose. Karkasų naudojimas žymiai supaprastina programuotojo darbą, nes nereikia sukti galvos, kokius būdus naudoti efektyviai sistemos apsaugai. Žemiau pateikiami populiariausi karkasų pavyzdžiai:

- Code Igniter (PHP)
- Ruby on Rails (Ruby)
- django (Python)
- Catalyst (Perl)
- Struts (Java)
- ESAPI
- Yiiframework
- Csrf-magic

Apsaugą perkeliant į karkasų lygį, sumažinama rizika susidurti su klaidomis atsirandančiomis dėl per mažos kompetencijos arba neapdairumo. Bendru atveju norint apsaugoti sistema nuo CSRF atakų reikia imtis tam tikrų atsargumo priemonių:

- Leisti GET užklausas tik tam, kad nuskaityti informaciją iš serverio, o ne tam, kad ją modifikuoti. Nors tai nėra saugus būdas užtikrinti apsaugą nuo CSRF atakų, tačiau tai kartu derinant su kitomis saugumo priemonėmis, galima visiškai užkirsti kelią tokio tipo atakoms.
- Visoms POST užklausoms būtina naudoti atsitiktinę reikšmę. Kai vartotojas apsilanko sistemoje, ji turi sugeneruoti (kriptografiškai stiprų) atsitiktinį skaičių, ir jį išsaugoti sesijos parametruose. Su kiekviena POST užklausa turi būti pridedamas šis atsitiktinis skaičius. Užklausa skaitoma teisinga tik tada, jei sausainuko ir formos atsitiktiniai skaičiai sutampa. Kai įsilaužėlis bando padirbti POST užklausą, jis negali žinoti atsitiktinio skaičiaus, bei dėl tos pačios kilmės taisyklės (*angl.* Same origin policy) jis negali perskaityti, pakeisti sausainuke esančios informacijos. Kadangi su forma perduodama reikšmė turi būti vienoda su sausainuko reikšme, įsilaužėlis negali sėkmingai įvykdyti POST užklausos, nebent jis kokiu nors būdu atspėja atsitiktinę reikšmę.
- Taip pat galima kelių lygių autentifikavimą prieš svarbius veiksmus. Sistema ir vartotojas iš anksto turi turėti paruoštą grupę atsitiktinių skaičių, kurių galėtų paklausti vartotojo prieš patvirtinant

veiksmą, arba galimas, mažiau saugus variantas, kai vartotojo paprašoma pakartotinai įvesti savo slaptažodį.

Vieni populiariausių karkasų apsaugai nuo CSRF atakų yra šie:

- CSRF-magic
- Yiiframework
- Django

CSRF-magic yra HTML Purifier projekto dalis. HTML Purifier yra standartais pagrįsta biblioteka kurios pagrindinis tikslas buvo padėti programuotojams apsaugoti kuriamas sistemas nuo XSS (kodo įterpimo) atakų. Tai yra plačiai vystomas projektas, prie kurio prisidėjo daug gerų programuotojų, sukurdami gerą karkasą apsaugai nuo XSS atakų. Projektui išsiplėtus apsauga buvo išplėsta ir nuo CSRF atakų, kuri yra neįmanoma be tinkamos XSS apsaugos. CSRF-magic įdiegimas į kuriamą (ar jau sukurtą) sistemą yra pakankamai paprastas, prie sistemos išeities kodo pridedama nuoroda į pačią biblioteką. Biblioteka prideda prie formų, esančių sistemoje, papildomą žymą kuri siunčiama kartu su formomis. Žymos reikšmė sugeneruojama unikali, kad įsibrovėlis negalėtų atspėti. [10]Yiiframework taip pat neapsiriboja vien tik apsauga nuo CSRF atakų. Taip pat ji dar saugo ir nuo XSS tipo atakų, bei nuo sausainukų informacijos vagysčių. Nuo CSRF atakų karkasas siūlo saugotis naudojant ta patį principą kaip ir HTML Purifier karkasas. Nenaudojant GET tipo užklausų duomenų pakeitimams, o prie visų POST tipo užklausų pridėti po papildomą žymę kurioje būtų saugomas tikrai atsitiktinis tekstas. [11]

Labiausiai paplitęs ir plačiausiai naudojamas karkasas apsaugai nuo CSRF tipo atakų yra django. Jis yra paprašytas Python programavimo kalbos pagalba, todėl gali būti ne vien tik sistemoms veikiančioms HTML/PHP pagrindu. Django suprojektuotas taip, kad saugotų tik POST užklausas, nes priimta, kad GET užklausos negali būti naudojamos informacijai perduoti į sistemą. Karkasas naudoja anksčiau minėtą apsaugą naudojant papildomą žymę. Taip pat jis leidžia išplėsti apsaugą pasinaudojant papildoma antrašte (*angl.* Referer header), kurioje saugoma informacija iš kur buvo gauta užklausa. Kadangi visos užklausos siunčiamos iš vartotojo naršyklės, tai tikrinama kokiame tinklalapyje vartotojas buvo prieš ateidamas į šį, t.y. kuris puslapis sugeneravo POST tipo užklausą. Šiam metodui būtinas HTTPS palaikymas, nes HTTP protokolas yra per daug nesaugus, galima nesunkiai padirbti šią antraštę. [12]

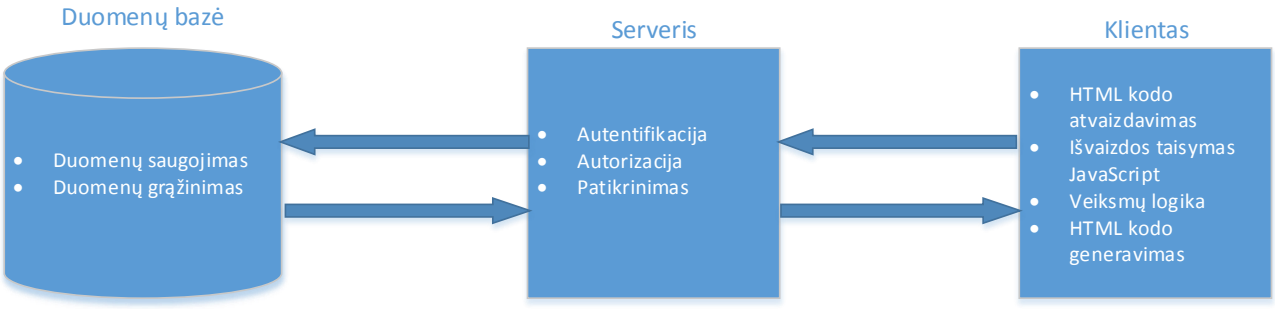
2.2 Apsauga nuo CSRF atakų naujai kuriamoje sistemoje

Kuriant visiškai naujas sistemas, o ne taisant egzistuojančias, atsiranda daugybė kitų galimybių kaip apsisaugoti nuo CSRF tipo atakų. Sparčiai populiarėjant interaktyvių tinklalapių paklausai, didelė dalis programuotojų pasitelkia į pagalbą JavaScript programavimo kalbą. Šiuo metu tai yra 6 populiariausia programavimo kalba remiantis TIOBE indeksu [13]. Tinklalapis „mashable.com“ JavaScript įvardina kaip antrą populiariausią kalba 2015 metais [14].

Sistemos kuriamos naudojant tik JavaScript, skiriasi nuo įprastinių sistemų tuo, kad jose pagrindinis sistemos vaizdas negeneruojamas php, python ar kita serverio pusėje veikiančia programavimo kalba. Visas puslapio vaizdas kuriamas HTML, CSS ir JavaScript pagalba. Tokiu metodu pasiekiama graži, sklandi ir realiu laiku atsinaujinanti sistemos sąveika su vartotoju. Tokios sistemos dažniausiai vadinamos vieno puslapio sistema (*angl.* Single Page Aplication). Kadangi tai yra visiškai naujas dalykas, tokių sistemų dar nėra daug, tačiau jos sparčiai populiarėja. Įprastinės sistemos struktūra pateikta 2 paveikslėlyje.

Pav. 2 Tradicinės sistemos modelis

Kaip matome paveikslėlyje už pagrindinę sistemos dali atsakingas serveris. Serveryje ir duomenų bazėje atliekami visi veiksmai, o tada jie tiesiog atvaizduojami vartotojui. Šiuo atveju dažniausiai serverio pusėje naudojama PHP programavimo kalba, o kliento pusėje JavaScript, kad būtų galima sistemai suteikti dinamiškumo ir kad būtų pasiekiamas kiek didesnis interaktyvumas. Žemiau esančiame paveikslėlyje pavaizduojama, kaip atrodo sistema, kurioje naudojama vien JavaScript programavimo kalba.



Pav. 3 Sistemos kuriamos naudojant vien JavaScript modelis

Kaip matome aukščiau esančiame paveikslėlyje, sistemos veikimo principas pasikeičia iš esmės. Visas sistemos veikimas ir logika, perkeliama į vartotojo pusę. Serveris reikalingas tik pačios sistemos failų saugojimui bei kreipimųsi į sistemą autentifikavimui, autorizavimui ir patikrinimui. Jokie veiksmai nebeatliekami serverio pusėje. Tokiu būdų serveriui sumažinama apkrova, nes kiekvienas vartotojas visus sistemai reikalingus veiksmus atlieka savo naršyklėje. Tokiu sistemų kūrimas aprašomas knygoje „Single Page Web application“ [15].

Kalbant apie apsaugą nuo CSRF atakų, sistemos parašytos vien JavaScript programavimo kalba yra gerokai paprasčiau apsaugomos, nei tos, kurios aprašytos kelių įvairių kalbų pagalba (dažniausiai pasitaikanti kombinacija yra PHP ir JavaScript). Bet kuriuo atveju kuriant internetinio tipo sistemas, be HTML žymėjimo kalbos ir stilių aprašo CSS negalėsim apsieiti. Programavimo kalbos PHP atveju bet kokios užklausos metu, naršyklė automatiškai prideda sesijos informaciją iš prisijungusio vartotojo sausainuko, todėl visos užklausos serveriui vienodos. JavaScript atveju, atsitiktinės žymės generavimas išlieka toks pats. Žymė sugeneruojama ir įrašoma į prisijungusio vartotojo sausainuką kaip papildomas atributas. Dėl tos pačios kilmės taisyklės, kitos svetainės negalės perskaityti informacijos esančios sausainuke, todėl ji išliks nežinoma. Prieš generuodama bet kokią užklausą, sistema nuskaito atsitiktinai sugeneruotą reikšmę išsaugota sausainuke, tada ją prideda prie generuojamos užklausos kaip papildomą antraštės lauką. Serveris gavęs bet kokia užklausą, pirmiausia patikrina ar yra pridėta papildoma antraštės reikšmė. Jeigu serveris nesugeba nuskaityti papildomos reikšmės, užklausa atmetama kaip netinkama, tikėtina, kad ji buvo siųsta ne iš pačios

sistemos. Jeigu prie užklausos randama pridėta papildoma reikšmė, tada ji patikrinama, kad būtų galima nustatyti ar jos nebuvo bandoma padirbti. Galiausiai patikrinus reikšmę ir nustačius, kad ji nebuvo padirbta, serveris galiausiai patvirtina užklausą ir ja įvykdo.

2.2.1 Populiariausi JavaScript karkasai

Šiame skyrelyje plačiau aptarsiu populiariausius JavaScript karkasus. Kuriant bet kokią sistemą būtina numatyti sistemos plėtimąsi. Bėgant laikui, sistemai nuolatos prireikia atnaujinimų. Dažniausiai atnaujinamos sistemos dalys kurios veikia nekorektiškai arba turi paliktų saugumo spragų. Taip pat neretai siekiama ir patobulinti jau esančias sistemas išplečiant jų funkcionalumą. Nenaudojant karkasų išeities tampa labai komplikuotas. Dažniausiai sistemos pakeitimus atlieka kiti asmenys, o ne patys kūrėjai. Pradedama nesilaikyti standartų kurie buvo naudojami kuriant sistemą, taip sistemos kodas tampa sunkiai analizuojamas ir atsiranda dar daugiau saugumo spragų. Žemiau pateiksiu sąrašą su keletu populiariausių atviro kodo JavaScript karkasų:

- AngularJS
- Backbone.js
- Ember.js

Visi aukščiau paminėti JavaScript karkasai yra padaryti jQuerry pagrindu.

AngularJS yra seniausias karkasas iš visų paminėtų. Jos palaikymu ir plėtimu rūpinasi kompanija Google, atskiros programuotojų grupės ir kai kurios kitos kompanijos, kurioms reikalingas galingas ir funkcionalus įrankis vieno puslapio aplikacijoms kurti. Biblioteka skaito HTML kodą, prie kurio pridėti papildomi atributai. Šie atributai reikalingi tam, kad karkasas žinotų kam vėliau priskirti atitinkamą informaciją arba iš kurio lauko ją paimti. Karkasas vietoj to, kad pridėtų papildomo HTML kodo, keičia tiesiog elementus, kuriems priskirti šiam karkasui būdingi atributai. Tokių būdu sistema veikia žymiai greičiau. Šis karkasas leidžia programuotojui atlikti tuos pačius dalykus įvairiais būdais, tačiau tai ne visada gali būti geras dalykas. Gerai įvaldžius karkaso teikiamas galimybes, programuotojas gali padaryti daug įspūdingų dalykų, tačiau didelis galimybių turėjimas pradedančiajam programuotojui būna didelis iššūkis, nes nėra tiksliai aišku kada kokį metodą naudoti. Lyginant su kitais paminėtais karkasais, AngularJS yra lėtokas. Geriausiai šis karkasas tinka mažiems ar vidutinio dydžio projektams. Dėl savo unikalaus dizaino karkasas leidžia programuotojui žymiai sumažinti programinio kodo eilučių kiekį, kas stipriai palengvina sistemos testavimą ir skaitomumą. Nepaisant visų paminėtų trūkumų, šiuo metu tai yra vienas populiariausių (geriausių) JavaScript karkasų.

Backbone yra “lengviausias“ karkasas iš visų. Suspaustas ir minimizuotas karkasas užima 6.4 kilobaitus, kai tuo tarpu AngularJS užima 36 kilobaitus. Šio karkaso mažumas yra didžiausias pliusas lyginant su kitais. Ankstesnės versijos buvo stipriai surištos su JQuery biblioteka ir be jos negalėjo veikti visai, tačiau paskutiniai leidimai vis labiau tolsta nuo šios bibliotekos, palikdami programuotojui pasirinkimo laisvę kurią biblioteką pasirinkti (galima kartu ir toliau naudoti JQuery). Pagrindinis šio karkaso trūkumas ir privalumas yra karkaso dydis. Kadangi jis yra mažas, sistema veikia greičiau, tačiau programuotojui reikia žymiai daugiau pasikartojančio kodo parašyti pačiam, kad sistema pilnai veiktų.

Ember yra naujausia biblioteka iš visų trijų paminėtų, tačiau ji sparčiai plečiasi. Kolkas prie jos ir su ja dirbančių žmonių ratas yra mažiausias, tačiau skaičiai sparčiai didėja. Ember yra daugiausiai užimantis karkasas iš visų minėtų, jo suspausto dydis siekia 64 kilobaitus. Kadangi pats karkasas nėra mažas, programuotojui reikia gerokai mažiau rašyti pasikartojančio programinio kodo, norint pasiekti pilną sistemos funkcionalumą. Sintaksė specialiai buvo kuriama panaši į JQuery bibliotekos naudojamą sintaksę, kad perėjimas būtų kur kas paprastesnis tiek patyrusiam tiek vidutinio lygio programuotojui. Kuriant šį karkasą buvo bandoma surinkti geriausias idėjas iš jau esamų sprendimų. Tai leido sukurti karkasą kuris leidžia kurti tinklalapius, kurie yra panašiausi į aplikacijas telefonuose, ar programas naudojamas asmeniniuose kompiuteriuose.

Visi paminėti karkasai turi savų pliusų ir minusų. Dažniausiai karkaso pasirinkimas priklauso daugiausiai nuo pačio programuotojo įpročių ir kuriamos sistemos reikalavimų. Kadangi karkasai nėra visiškai identiški, kiekvienas jų geriausiai tinka specifiniams sprendimams ir projektams.

2.3 Apsaugos nuo CSRF atakų galimybės vartotojui

Vartotojo lygio apsaugai būtina naudoti tam tikrus įrankius, pavyzdžiui:

- „RequestRodeo“, tačiau jis yra neveiksmingas jeigu naudojamas SSL (*angl.* Secure Socket Layer) arba JavaScript naudojamas sugeneruojant dalį puslapio, nes „RequestRodeo“ analizuoja duomenis einančius per proxy, prieš tai kai jie parodomi naršyklėje.
- Dar galima naudoti „Firefox“ naršyklės įskiepį, kuris buvo suprogramuotas William Zeller ir Edward W. Felten ir pasiūlytas 2008 metais, „Cross-Site Request Forgeries: Exploitation and Prevention” darbe. Jų įrankis tikrina kiekvieną HTTP užklausą ir sprendžia ar ji gali būti leidžiama. Atmetus tam tikras užklausas, vartotojui pasirodo pranešimas, kad užklausa buvo blokuota ir klausimas, gal jis norėtų atblokuoti ir pakartoti ją iš naujo.

2.4 OWASP svarbiausi sistemų pažeidžiamumai

Owasp – (*angl.* Open Web Application Security Project), tai yra atvira interneto bendruomenė, kurios pagrindinis tikslas pristatyti organizacijoms ir paprastiems žmonėms, sistemas kuriomis būtų saugu naudotis. Šiame projekte nagrinėjami saugumo pavojai, bei siūlomi būdai kaip būtų galima įvertinti saugumo rizikas ir apsaugoti savo aplikacijas nuo kylančių pavojų. OWASP pateikia dešimt pavojingiausių sistemai atakų sąrašą. Apsaugai serverio pusėje OWASP projekto kūrėjai siūlo naudoti „ESAPI“. Tai yra nemokama, atviro kodo internetinių aplikacijų saugumo kontrolės biblioteka, kuri palengvina programuotojams saugesnės programos rašymą. Šią biblioteką savo internetinėse aplikacijose naudoja ne viena žinoma kompanija: American Express, Apache Foundation, Booz Allen Hamilton, Aspect Security, Foundstone(McAfee) ir t.t. Ši biblioteka yra gerai dokumentuota ir yra išleista visoms populiariausioms platformos Java, PHP, ASP ir pan.

2.5 Tiriamosios dalies išvados

Bėgant metams sparčiai išaugo interneto vartotojų skaičius. Vis daugiau įmonių pradeda teikti paslaugas internetu. Kad visa tai taptų įmanoma, informacija turi būti kažkur talpinama, todėl buvo sugalvotos informacinės sistemos, kitaip dar vadinamos tinklalapiais. Sistemos laikui bėgant tampa vis sudėtingesnės, o tai reiškia, kad vis daugiau žmonių dalyvauja jų kūrime. Dėl daugelio žmonių įsikišimo į projektavimą ir realizavimą atsiranda sistemos ne vientisumas ir tampa labai sudėtinga užtikrinti sistemos spragų nebuvimą. Šią spragą bando išnaudoti internetiniai įsilaužėliai (*angl.* hacker).

Dažniausiai pasitaikančios atakos prieš tinklalapius yra CSRF, XSS, injekcija, direktorijų klastojimas, Dos ir DDos atakos. CSRF – (*angl.* Cross-Site Request Forgery) atakų metu kaip jaukas yra naudojamas nekaltai atrodantis puslapis, verčiantis autentifikuotos aukos naršyklę keistis informacija su pažeidžiama Web aplikacija. CSRF atakos tikslas yra panaudoti aukos aktyvią sesiją kitoje svetainėje (pvz. banke) ir atlikti veiksmus prieš jo valią, jam to net nesuvokiant. Labiausiai paplitęs apsisaugojimo būdas jau sukurtose sistemos yra naudoti karkasus. Galima rasti daugybę jau paruoštų naudoti karkasų sistemos saugumui užtikrinti.

Norint apsaugoti sistema nuo CSRF atakų reikia laikytis tokių taisyklių: Leisti GET užklausas tik tam, kad nuskaityti informaciją iš serverio, o ne tam, kad ją modifikuoti, visoms POST užklausoms būtina naudoti atsitiktinę reikšmę kuri būtų pridedama prie kiekvienos užklausos siunčiamos į serverį informacijai atnaujinti. Sparčiai populiarėjant interaktyvių tinklalapių paklausai, didelė dalis programuotojų pasitelkia į pagalbą JavaScript programavimo kalbą. JavaScript

programavimo kalba yra gerokai paprasčiau apsaugomos, nei tos, kurios aprašytos kelių įvairių kalbų pagalba (dažniausiai pasitaikanti kombinacija yra PHP ir JavaScript). Programavimo kalbos PHP atveju atveju bet kokios užklausos metu, naršyklė automatiškai prideda sesijos informaciją iš prisijungusio vartotojo sausainuko, todėl visos užklausos serveriui vienodos. JavaScript atveju, atsitiktinės žymės generavimas išlieka toks pats. Žymė sugeneruojama ir įrašoma į prisijungusio vartotojo sausainuką kaip papildomas atributas. Dėl tos pačios kilmės taisyklės, kitos svetainės negalės perskaityti informacijos esančios sausainuke, todėl ji išliks nežinoma. Prieš generuodama bet kokią užklausą, sistema nuskaito atsitiktinai sugeneruotą reikšmę išsaugota sausainuke, tada ją prideda prie generuojamos užklausos kaip papildomą antraštės lauką. Serveris gavęs bet kokia užklausą, pirmiausia patikrina ar yra pridėta papildoma antraštės reikšmė. Jeigu serveris nesugeba nuskaityti papildomos reikšmės, užklausa atmetama kaip netinkama, tikėtina, kad ji buvo siųsta ne iš pačios sistemos. Visi svarbiausiai internetinių sistemų pažeidžiamumai ir būdai kaip su jais galima kovoti aprašomi OWASP projekte, tai yra atvira interneto bendruomenė, kurios pagrindinis tikslas pristatyti organizacijoms ir paprastiems žmonėms, sistemas kuriomis būtų saugu naudotis.

3. APSAUGOS NUO CSRF ATAKŲ OPTIMIZAVIMO TYRIMAS

CSRF tipo atakos yra labai dažnas pažeidžiamumas aptinkamas tinklapiuose, kuris leidžia įsibrovėliui pavogti arba pakeisti paprastam vartotojui dažnai svarbią informaciją. Šis pažeidžiamumas atsiranda dėl to, kad programuotojai dažnai nepakankamai išnagrinėja savo kuriamą sistemą ir nenumato kritinių vietų, kuriose turi būti naudojama maksimali apsauga. Anksčiau laikyta, kad pakanka nenaudoti GET tipo užklausų svarbioms operacijoms ir to pakaks apsaugoti tinklapį nuo CSRF tipo atakų, tačiau laikui bėgant paaiškėjo, kad POST užklausos taip pat nėra visiškai saugios. Nors jos negali būti taip akivaizdžiai ištyrinėtos ir nulaužtos kaip GET tipo užklausos (nes nesimato naršyklės adreso laukelyje), tačiau jas taip pat nėra sunku atsekti. Įsilaužėlius turi pakankamai didelį pasirinkimą kaip išnagrinėti POST užklausų siuntimą. Vienas paprasčiausių būdų yra panaudoti tinklo stebėjimo programinę įrangą, kad būtų įmanoma stebėti naršykle siunčiamus paketus. Iš jų galima aiškiai atskirti POST tipo užklausas ir jose siunčiamą informaciją. Jeigu su užklausa nėra siunčiamas joks papildomas atsitiktinai sugeneruotas atributas, tada įsilaužėlis gali patalpinti užpildytą formą savo tinklalapyje ir ją išsiųsti HTTP_POST metodu, kai auka prisijungia prie jo tinklalapio. Jeigu tuo metu vartotojas yra prisijungęs ir prie sistemos kuri yra pažeidžiama CSRF atakoms, tai įsilaužėlis galės sėkmingai įvykdyti užklausą to prisijungusio vartotojo vardu. Kaip bebūtų keista, retais atvejais vis dar pasitaiko sistemų kuriose atnaujinti duomenims naudojamas GET metodas, šiuo atveju atakuotojui tampa dar paprasčiau aptikti ir išnaudoti pažeidžiamumą, nes visos GET užklausos matomos naršyklės adreso laukelyje.

Apsisaugojimui nuo tokio tipo atakų yra sukurta nemažai priemonių tiek vartotojo pusėje tiek pačio tinklalapio apsaugojimui. Vartotojas turėtų būti pats suinteresuotas apsisaugoti nuo įmanomų atakų, tačiau dauguma interneto vartotojų nėra informacinių technologijų specialistai, todėl dažniausiai net nenutuokia apie galimus pavojus. Net informavus vartotojus apie visus galimus pavojus internete, kurie neapsiriboja vien tik CSRF atakomis, tai neduotų teigimų rezultatų, nes kas dieną atsiranda vis naujos grėsmės, o kad jas visas žinoti reikia aktyviai domėtis informacinių technologijų saugumu.

Kadangi vartotojo apsauga nėra labai efektyvi, reikia programuotojams apsaugoti pačią sistemą. Sistemos apsaugai šiuo metu galima rasti nemažai karkasų, kuriais programuotojai gali pasinaudoti, apsaugodami sistemas nuo CSRF atakų. Dauguma šių karkasų siūlo kaip pagrindinę apsaugą naudoti papildomą žymę, nematomą lauką, pridedamą prie kiekvienos siunčiamos POST užklausos. Siunčiama reikšmė turi būti unikali ir sunkiai nuspėjama, nes atakuotojui išsiaiškinus pagal kokį algoritmą sudaroma ta reikšmė, jis ją galėtų nuspėti. Pridedant tokia žymą prie POST užklausos, atakuotojas negali iš anksto jos žinoti, todėl negali paruošti iš anksto užpildytos formos ir patalpinti jos savo sistemoje, laukdamas kol potenciali auka aplankys ją. Sistemose sukurtose naudojant vien JavaScript programavimo kalbą, apsauga nuo CSRF tipo atakų praktiškai nekinta. Vis tiek reikia sugeneruoti atsitiktinę reikšmę, kurią būtų galima perduoti iš vartotojo naršyklės į serverį. Šiuo atveju ta žymė pridedama ne prie kiekvienos formos, bet prie pačios užklausos kaip papildomą antraštės atributą. Serveris gavęs bet kokią POST užklausą, pirmiausiai patikrina ar yra pridėta papildoma antraštė ir ar ji tinkama ir tik po to įvykdo užklausą. Sekančiuose skyreliuose detaliau aprašysiu papildomą sistemos apsaugą, kurią galima panaudoti jau sukurtose sistemose. Buvo pasirinkta daugiau tobulinti jau sukurtų sistemų apsaugą, nes didžioji dalis anksčiau kurtų sistemų nėra pakankamai apsaugotos, tiek dėl žinių trūkumo, tiek dėl programuotojų neatsižvelgimo į tobulėjančias technologijos ir didėjantį įsilaužimų skaičių. Šis apsaugos metodas yra labiau orientuotas į jau esamas sistemas, tačiau jį taip pat paprastai galima pritaikyti ir į naujai kuriamas.

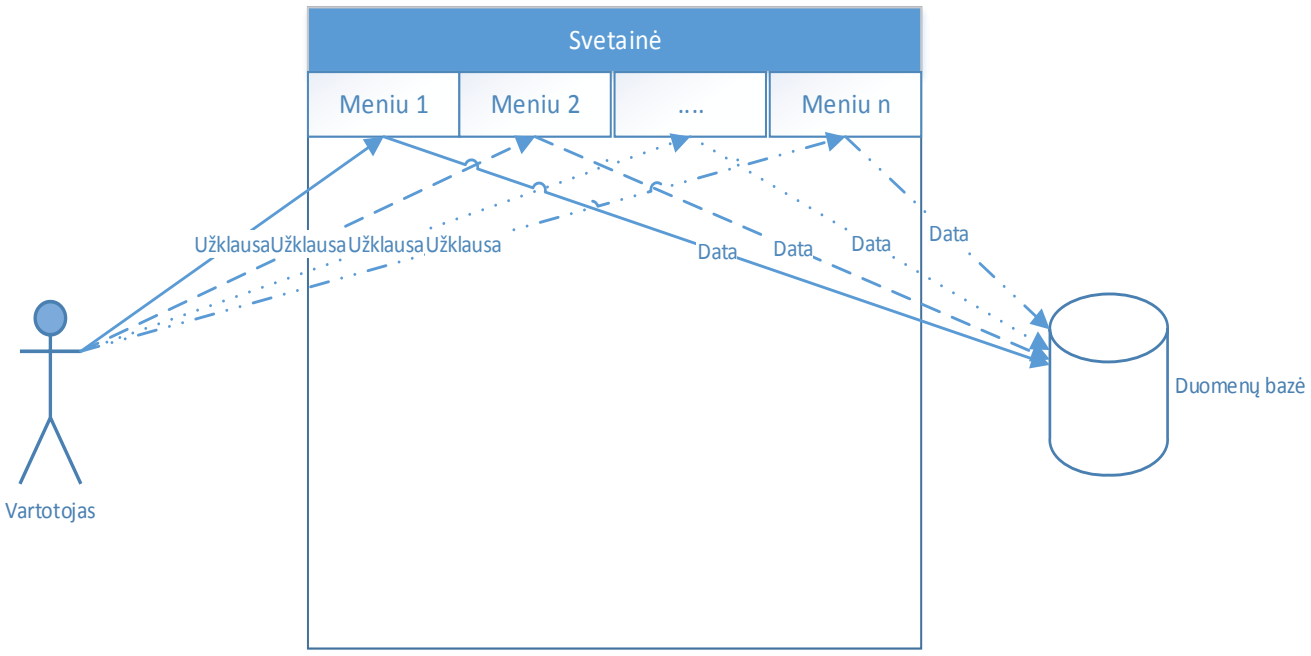
3.1 Papildoma apsauga panaudojant laiko žymę

Vartotojui bandant nusiųsti POST užklausą sistemoje, kuri apsaugota nuo CSRF atakų, prieš išsiunčiant formą, prie jos pridedamas nematomas laukas, dar kitaip vadinamas CSRF žyme, kuriame įrašomas iš anksto sugeneruotas atsitiktinis tekstas. Dažniausiai laikoma, kad šios apsaugos pakankamai užtenka užtikrinti sesijos identifikavimui, tačiau ją galima išplėsti.

Kadangi vartotojai nesiunčia užklausų iš komandinės eilutės, o jungiasi prie sistemos per naršyklės grafines sąsajas, tokį sprendimą yra pakankamai paprasta įgyvendinti ir pritaikyti skirtingų tipų įrenginiams. Bet kuris vartotojas, prieš galėdamas nusiųsti bet kokią POST užklausą į serverį, dažniausiai turi įvykdyti bent keletą iš žemiau esančių veiksmų:

1. Prisijungti prie sistemos.
2. Pereiti per keletą vidinių meniu, kol pasiekia norimą formą.
3. Užpildyti formos laukus.
4. Nusiųsti užklausą.

Žinant, kad vartotojas privalomai turės atlikti didesnę dalį anksčiau minėtų veiksmų, galima sumodeliuoti papildomą apsaugą pridedant laiko žymę prie vartotojo informacijos duomenų bazėje - registruojant vartotojo veiksmus sistemoje atliekant laikinus kiekvienos užklausos įrašus veiksmų registre. Šis metodas leistų užfiksuoti užklausos laiko žymę, užklausos pavadinimą, sesijos pradžią ir pabaigą, todėl mėginimas sugeneruoti kenksmingą veiksmą vartotojo vidinėje sistemoje būtų nutrauktas sistemos serveryje dėl informacijos neatitikimų. Sėkmingai autentifikavus vartotoją, jo prisijungimo metu būtų įrašoma dabartinė data į duomenų bazę. Vėliau vartotojui naršant svetainėje prisijungus su kiekviena jo naujai atidaryta svetainės dalimi išsaugoma esama, tiksli, data. Supaprastintas sistemos veikimas matomas paveikslėlyje žemiau (4 pav.).



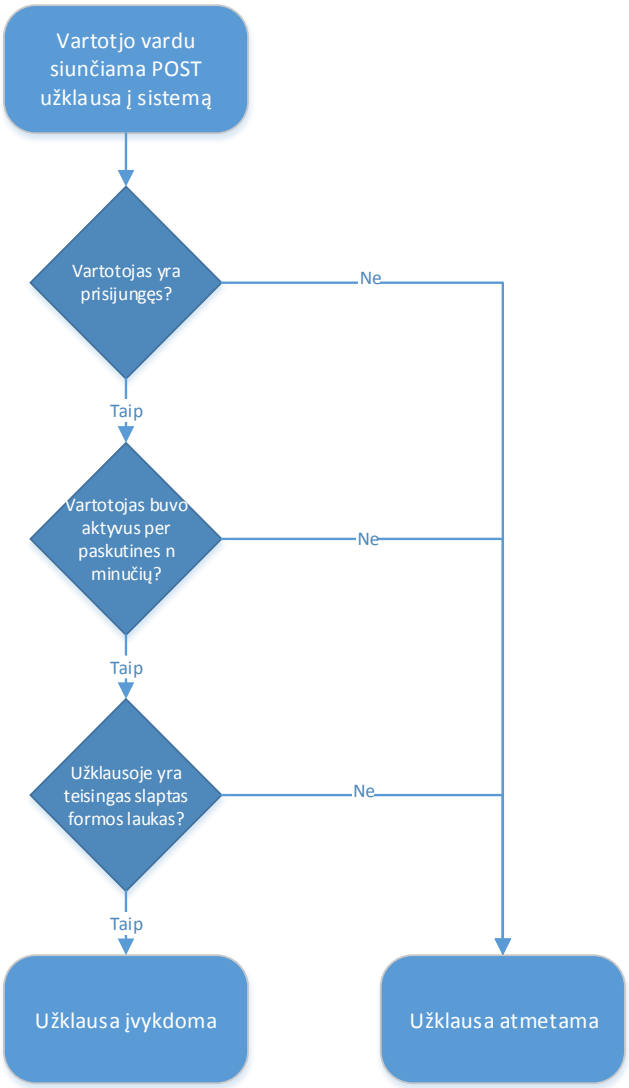
Pav. 4 Siūlomo metodo veikimas sistemoje

Šis metodas yra ne tik veiksmingas, tačiau ir vienas iš lengviausiai pritaikomų. Jį galima nesunkiai įdiegti jau esamose sistemose (nes nereikia papildomai modifikuoti esamos informacijos ieškant kodavimo sprendimų, o užtenka papildomai įtraukti naują įrašų elementą lentelėse), taip pat ir kuriant naują sistemą. Vartotojų duomenų lentelėje reikia pridėti papildomą lauką „Paskutinės_užklausos_laikas“. (žr. 5 pav.).

Pav. 5 Duomenų bazės lentelės pavyzdys

Paruošus duomenų bazę įrašų datai saugoti, reikia kiekviename svetainės puslapyje, bei kiekvienoje vartotojo generuojamoje užklausoje įterpti papildomą programinį kodą, kuris vartotojui atliekant , bet kokius veiksmus išsaugotų datą. Programinio kodo modifikacijas atlikti nėra sudėtinga, kadangi daugelis programavimo kalbų jau turi aprašytas datos ir laiko klases bei metodus. Naudojant labai populiarią internetinių puslapių kūrimo metodiką JavaScript galima netgi fiksuoti ar skirtukas yra aktyvus ar vartotojas tiesiog pamiršo jį uždaryti. Tai leidžia realiu laiku matyti ar vartotojas toliau aktyviai atlieka užklausas ir veiksmus ar tai gali būti papildomos rizikos registruojamos užklausos, kurias sistema atmeta.

Kiekviena svarbi forma, kuria perduodama vartotojo privati informacija arba konfidencialūs veiksmai, turi papildomą apsaugą tikrinant paskutinio kreipimosi į sistemą laiką. Šis automatinis tikrinimas nėra pastebimas vartotojui, todėl nesukuria papildomų neigiamų sistemos vartojimo potyrių, taip pat persiunčiamas minimalus duomenų kiekis neapkrauna sistemos ir leidžia nuosekliai vykdyti sekančias užklausas. Išsaugotos informacijos tikrinimas ir lyginamas su pirminiais įrašais yra paprastas ir efektyvus.



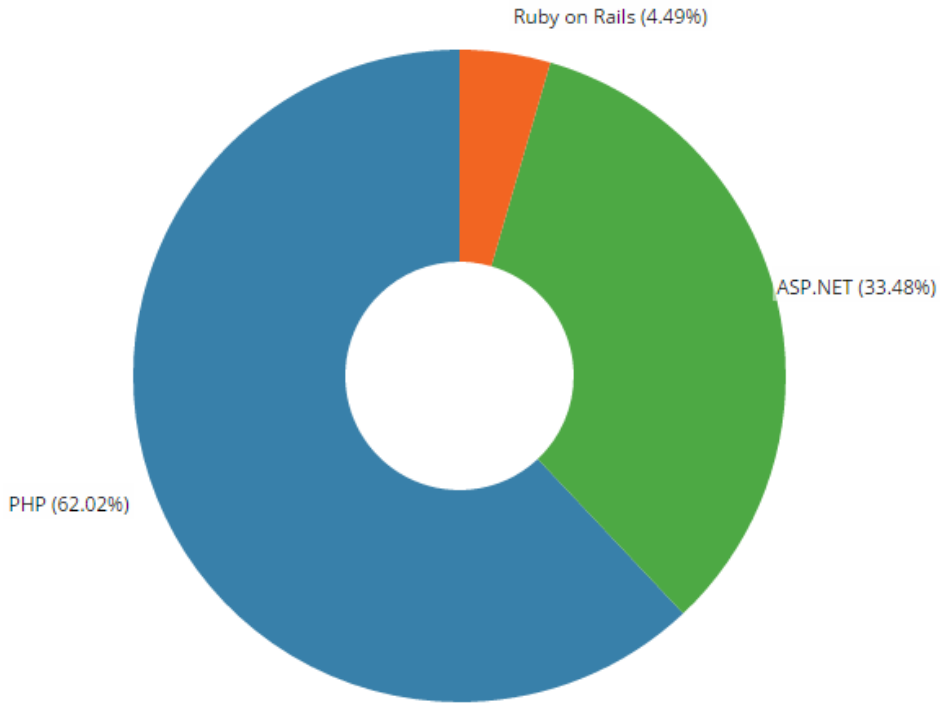
Pav. 6 Užklausų tikrinimas

Aukščiau pateikiamas nuoseklaus tikrinimo proceso pavyzdys (žr. 6 pav.). Atliekami veiksmai vykdomi nustatyto eiliškumo tvarka ir jei sistema fiksuoja nesutapimus, programai gražinamas klaidos kodas ir tolimesni veiksmai nebevykdomi:

1. Sistema gavusi bet kokią užklausą pradžioje patikrina ar vartotojas yra prisijungęs esamu metu, t.y. ar turi aktyvią sesiją. Jei vartotojas yra neprisijungęs arba sesijos laikas jau pasibaigęs, užklausa negali būti vykdoma ir procesas yra nutraukiamas, jei vartotojas yra prisijungęs ir aktyvios sesijos laikas dar nepasibaigęs, pereinama prie sekančio žingsnio.
2. Sekančiame žingsnyje patikrinama prieš kiek laiko vartotojas buvo paskutinį kartą aktyvus svetainėje (tikrinamas paskutinės užklausos laikas su naujos užklausos atlikimo momentu fiksuota data). Jei vartotojas yra neaktyvus didesnį laiko tarpą negu pasirinktas sistemos administratoriaus laikas (pavyzdžiui 10 minučių), užklausa neįvykdoma arba vykdymas atidedamas paprašant papildomos autorizacijos (dažniausiai bankinių sistemų praktikoje taikoma metodika), jei laiko tarpas mažesnis, pereinama prie paskutinio tikrinimo.
3. Paskutiniame žingsnyje patikrinama ar paslėptame formos lauke perduodama teisinga atsitiktinė reikšmė, jei reikšmė teisinga, užklausą įvykdoma, jei neteisinga, užklausa atmetama.

3.2 Testavimo priemonės

Norint atlikti testavimus pirmiausia reikės sukurti keturias sistemas. Pirmojoje sistemoje prisijungimas ir kelios formos skirtos nusiųsti duomenims į serverį. Antroji sistema bus pirmosios papildinys. Apsauga nuo CSRF tipo atakų bus realizuota panaudojant HTML/PHP programavimo kalbas. Trečioji sistema taip pat bus pirmosios sistemos papildinys. Papildoma apsauga nuo CSRF tipo atakų bus realizuota JavaScript programavimo kalbos pagalba. Paskutiniojoje sistema bus kenkėjiškas tinklalapis bandantis įvykdyti CSRF tipo ataką prieš pirmas tris sistemas. Svetainių kūrimui pasirinktos labiausiai paplitę technologijos: HTML, PHP, JavaScript. Šios, tinklapio kūrimo, technologijos buvo pasirinktos dėl didelio paplitimo tarp kūrėjų. Tinklalapio „builtwith“ duomenimis, naudojant programavimo kalbą PHP parašyti tinklalapiai sudaro 62% visų egzistuojančių sistemų [16] (žr. 7 pav.).



Pav. 7 Internetinio programavimo kalbų populiarumas

Tokio tipo apsauga tiks ir sistemoms parašytoms kitoms technologijomis. Principai išlieka tokie patys, skirsis tik realizacija.

Svetainių testavimui reikalingas serveris, kadangi testuojama bus nešiojamame kompiuteryje, kuriame įdiegta windows 7 operacinė sistema, buvo panaudota „xampp“ programinė įranga, skirta emuliuoti serverį. Xampp versija 1.8.3, ja sudaro:

1. Apache web serveris (versija 2.4.10)
2. MySQL duomenų bazė (versija 5.6.20)
3. PHP (versija 5.5.15)
4. phpMyAdmin (versija 4.2.7.1)

Toliau skyrelyje aprašysiu testavimo scenarijus kurie bus naudojami tolimesniame darbe.

3.3 Testavimo scenarijus

Kaip minėjau ankstesniame skyrelyje bus naudojamos keturios sistemos. Kadangi sistemos sukurtos grynai testavimo tikslais, jos nėra labai išplėtotos. Pirmojoje sistemoje realizuotas prisijungimas ir kelios formos skirtos nusiųsti duomenims į serverį. Antroji sistema bus pirmosios papildinys. Apsauga nuo CSRF tipo atakų bus realizuota panaudojant HTML/PHP programavimo kalbas. Kiekviename sistemos puslapyje bus pridėtas papildomas kodas, kuris atidarinėjant tą sistemos dalį užfiksuos kreipimosi laiką ir išsaugos duomenų bazėje su tuo vartotojo vardu. Bus naudojami metodai aprašyti PHP programavimo kalba. Trečioji sistema taip pat bus pirmosios sistemos papildinys. Papildoma apsauga nuo CSRF tipo atakų bus realizuota JavaScript programavimo kalbos pagalba. Kiekviename sistemos puslapyje bus įterptas papildomas kodas, kuris bus aktyvus tol, kol sistemos langas bus atidarytas. Tokį sistemos funkcionalumą galima pasiekti panaudojant JQuery biblioteką, o tiksliau vieną iš jos funkcijų „setInterval“. Šios funkcijos paskirtis yra periodiškai paleidinėti tam tikrą programinį kodą, su iš anksto nustatytais intervalais. Ši funkcija veikia tik tol, kol skirtukas yra naršyklės pagrindiniame lange. Atsidarius naują skirtuką, JavaScript kodas tampa neaktyvus, o vėl grįžus prie atidaryto skirtukas jis ima veikti toliau. Panaudojant šia funkciją duomenys bus periodiškai ir realiu laiku siunčiami į duomenų bazę su esamu aktyvumu. Verta paminėti, kad šis metodas turi ir silpnąją vietą, atsidarius naują naršyklės langą (ne skirtuką) ir sistemos skirtuką palikus senajame lange kaip pagrindinį atidarytą, JavaScript kodas ir toliau bus aktyvus. Tai reiškia, kad vartotojas nesinaudos tuo metu sistema, tačiau duomenys apie jo aktyvu bus ir toliau atnaujinami duomenų bazėje. Paskutiniojoje sistemoje bus tiesiog atvaizduojamas vienas paveikslėlis, o vartotojui apsilankius šioje sistemoje, bus automatiškai bandoma nusiųsti užklausą į likusias sistemas, tam, kad pakeisti prisijungusio vartotojo duomenis. Įsitikinti metodo efektyvumu, testuotojas turės atlikti keletą scenarijų:

1. Testuotojas pradžioje turės prisijungti prie bet kurios iš pirmų trijų paminėtų sistemų, tada naujame skirtuke atsidaryti antrąją (atakuotojo) svetainę, kuri nusiūs POST tipo užklausą į pirmąją svetainę. Kadangi apsaugai nuo CSRF tipo atakų bus naudojama tik laiko žymė, ataka turėtų įvykdyti užklausą.
2. Sekančio bandymo metu testuotojas turės prisijungti prie sistemos, tada palaukti bent penkias minutes, tada nueiti į atakuotojo svetainę. Atakuotojo svetainė bandys siųsti POST tipo užklausą į apsaugotą svetainę. Šiuo atveju, sistema atmes užklausą, nes vartotojas prabuvo neaktyvus tam tikrą laiko tarpą.

Įvykdžius abu anksčiau paminėtus scenarijus, bus galima nustatyti naujo siūlomo metodo efektyvumą. Pirmojo testu metu įsilaužimas turėtų įvyktu sėkmingai į visus aprašytus puslapius, nes juose nenaudojama jokio apsauga arba naudojamas metodas saugant tik papildoma laiko žymė. Atliekant antrą testavimo scenarijų, ataka turėtų nepavykti prieš dvi iš trijų sistemų. Prieš pirmą sistemą ataka turėtų įvykti sėkmingai, nes joje nenaudojama jokia apsauga. Prieš likusias dvi sistemas ataka neturėtų sėkmingai įvykti nes užklausa bus atmesta, kaip netinkama, nes patikrinus laiko žymę bus praėję daugiau nei penkios minutes po paskutinio prisijungimo prie sistemos.

3.4 Tyrimo išvados

CSRF tipo atakos yra labai dažnas pažeidžiamumas aptinkamas tinklapiuose, kuris leidžia įsibrovėliui pavogti arba pakeisti paprastam vartotojui dažnai svarbią informaciją. Šis pažeidžiamumas atsiranda dėl to, kad programuotojai dažnai nepakankamai išnagrinėja savo kuriamą sistemą ir nenumato kritinių vietų, kuriose turi būti naudojama maksimali apsauga. Atakuotojas suradęs sistemos pažeidžiamumą gali patalpinti užpildytą formą savo tinklalapyje ir ją išsiųsti HTTP_POST metodu kai auka prisijungia prie jo tinklalapio. Jeigu tuo metu vartotojas yra prisijungęs ir prie sistemos kuri yra pažeidžiama CSRF atakoms, tai piktavalis galės sėkmingai įvykdyti užklausą to prisijungusio vartotojo vardu.

Dauguma šių karkasų siūlo kaip pagrindinę apsaugą naudoti papildomą žymę, nematomą lauką, pridedamą prie kiekvienos siunčiamos POST užklausos. Siunčiama reikšmė turi būti unikali ir sunkiai nuspėjama, nes atakuotojui išsiaiškinus pagal kokį algoritmą sudaroma ta reikšmė, jis ją galėtų nuspėti. Sistemose sukurtose naudojant vien JavaScript programavimo kalbą, apsauga nuo CSRF tipo atakų praktiškai nekinta. Vis tiek reikia sugeneruoti atsitiktinę reikšmę, kurią būtų galima perduoti iš vartotojo naršyklės į serverį. Šiuo atveju ta žymė pridedama ne prie kiekvienos formos, bet prie pačios užklausos kaip papildomą antraštės atributą.

Papildoma apsauga nuo CSRF tipo atakų gali būti realizuota pridedant papildomą laiko žymę kurioje būtų saugoma paskutinio kreipimosi į sistemą data. Kadangi vartotojai nesiunčia užklausų iš komandinės eilutės, o jungiasi prie sistemos per naršyklės grafines sąsajas, tokį sprendimą yra pakankamai paprasta įgyvendinti ir pritaikyti skirtingų tipų įrenginiams. Sėkmingai autentifikavus vartotoją, jo prisijungimo metu būtų įrašoma dabartinė data į duomenų bazę. Vėliau vartotojui naršant svetainėje prisijungus su kiekviena jo naujai atidaryta svetainės dalimi išsaugoma esama, tiksli, data Naudojant labai populiarią internetinių puslapių kūrimo metodiką JavaScript galima netgi fiksuoti ar skirtukas yra aktyvus ar vartotojas tiesiog pamiršo jį uždaryti. Tai leidžia realiu laiku matyti ar vartotojas toliau aktyviai atlieka užklausas ir veiksmus ar tai gali būti papildomos rizikos registruojamos užklausos, kurias sistema atmeta. Kiekviena svarbi forma, kuria perduodama vartotojo privati informacija arba konfidencialūs veiksmai, turi papildomą apsaugą tikrinant paskutinio kreipimosi į sistemą laiką. Norint atlikti testavimus pirmiausia reikės sukurti keturias sistemas. Pirmojoje sistemoje realizuotas prisijungimas ir kelios formos skirtos nusiųsti duomenims į serverį. Antroji sistema bus pirmosios papildinys. Apsauga nuo CSRF tipo atakų bus realizuota panaudojant HTML/PHP programavimo kalbas. Trečioji sistema taip pat bus pirmosios sistemos papildinys. Papildoma apsauga nuo CSRF tipo atakų bus realizuota JavaScript programavimo kalbos pagalba. Paskutiniojoje sistema bus kenkėjiškas tinklalapis bandantis įvykdyti CSRF tipo ataką prieš pirmas tris sistemas.

4. PAPILDOMOS APSAUGOS NUO CSRF ATAKŲ REALIZAVIMAS

Pagal aprašytas tyrimo metodikas buvo atliekami programavimo darbai, tam, kad išsiaiškintume pasiūlytos idėjos efektyvumą. Testavimui buvo sukurtos keturios sistemos. Pirmojoje sistemoje buvo nenaudojamos jokios apsaugos priemonės, tam kad išsiaiškintume visus įmanomus ir aukščiau aprašytus pažeidžiamumus. Antrojoje sistemoje buvo daugiausiai naudojama PHP programavimo kalba bei įdiegti saugumo sprendimai. Trečioji sistema realizavo JavaScript programavimo kalbos apsaugos metodus. Ketvirtojoje sistemoje sukurtas CSRF atakos pavyzdys. Atlikus veiksmus buvo tikrinami rezultatai ir apsaugos patikimumas.

Sistemose realizuotas apsisaugojimas nuo CSRF atakų, panaudojus laiko žymę. Ketvirtoji sistema buvo sukurta kaip pavyzdinė atakuotojo sistema. Vartotojui atsidarius tinklalapį, jam parodomas paprastas paveikslėlis, tačiau fone bandoma siųsti POST tipo užklausą į anksčiau minėtus tinklalapius.

4.1 Tinklalapis nenaudojantis apsaugos nuo CSRF atakų

Testavimo tikslais buvo sukurtas tinklalapis nenaudojantis jokios apsaugos nuo CSRF atakų. Kaip buvo paminėta ankstesniame skyrelyje tinklalapis yra labai paprastas. Jame nėra realizuotų sudėtingų funkcijų. Vartotojas turi minimalias naršymo bei pakeitimų darymo galimybes.

Atsidaręs tinklalapį vartotojas yra nukreipiamas į prisijungimo/registracijos puslapį, kuriame pateikęs teisingus prisijungimo duomenis galės pasiekti savo paskyros duomenis ir prieiti prie informacijos skelbiamos tinklalapyje. Vartotojas taip pat gali matyti bei keisti savo asmeninę informaciją saugomą duomenų bazėje. Norint atnaujinti vartotojo duomenis, reikia užpildyti jau paruoštą formą tinklalapyje. Užpildžius visus laukus, ir paspaudus atnaujinti, sistema patikrinta įvestus laukus ir nusiunčia POST tipo užklausą į serverį, kad būtų atnaujinta duomenų bazėje esanti informacija naujais duomenimis. Kadangi kiekvienas vartotojas gali matyti tą pačią formą ir kokie duomenys gali būti atnaujinti, šis tinklalapis gali tapti pažeidžiamu. Žinant kokie duomenys bus sugeneruoti ir išsiųsti POST metodu į serverį, galima iš anksto paruošti POST tipo užklausą, ir ją patalpinti savo tinklalapyje. Šiai dienai tokių, nepilnai apsaugotų, tinklalapių yra stebėtinai daug. Saugumo bendrovės „Sakurity“, tiriančios pažeidimus internetinėse aplikacijose, duomenimis yra nuo 50% iki 70% tinklalapių kurie yra pažeidžiami įvairiausio tipo atakom [17]. Dauguma jų yra pažeidžiami CSRF tipo atakom. Dažniausiai bet kokiai informacijai atnaujinti naudojamos HTML formos ir POST metodas, todėl daugumą jų programuotojai pamiršta apsaugoti, o piktavališkam panaudojimui galimybių suteikiama labai daug (“Sakurity” specialisto pateikiamuose testuose matyti, kad dažniausiai tokiems veiksmams užtenka tik vienos programinio kodo eilutės). Paprastam vartotojui praktiškai neįmanoma nustatyti ar jo lankomose svetainėse taikoma pakankama apsauga nuo CSRF tipo atakų ar ne, kadangi nėra standartizuoto tikrinimo metodo, kuris leistų sertifikuoti puslapius. Norint nustatyti ar tinklalapis turi saugos spragų reikia turėti daug programavimo žinių ir žinoti bendrus principus kaip gali būti atliekamos atakos prieš tinklalapius. Turint pakankamai žinių galima dažniausiai nesunkiai įvertinti ar duomenys siunčiami apsaugoti nuo CSRF atakų. Kadangi dažniausia apsauga yra pridedant papildomą atsitiktinę reikšmę prie kiekvienos formos, tai galima pamatyti iš tinklalapio išeities kodo kuris matomas naršyklėse. Visos šiuolaikinės naršyklės turi galimybę parodytu jau sugeneruoto tinklalapio kodą. Peržvelgus formos duomenis siunčiamus į serverį galima rasti ar yra pridėtas papildomas sugeneruotas atsitiktinis kintamasis, kuris ir leidžia identifikuoti vartotojo sugeneruotą užklausą ir atšaukti tas užklausas, kurios yra kenkėjiškos.

Šiame puslapyje nenaudojama jokia apsauga, todėl formos išvesties kodas yra lengvai iššifruojamas, o duomenų pateikimas yra tiesioginis. Beveik visas formos reikšmes galima registruoti iš anksto, kaip tai parodoma pavyzdyje žemiau:

```
<form action="duomenys.php" method=post>
<input name="name" value="Vardas">
<input name="lastname" value="Pavardė">
```

```
<input name="age" value="15">
</form>
```

Nusiuntus tokią iš anksto sugeneruotą formą į serverį, jei nėra tikrinamas sesijos autentiškumas ar laiko žymė, kai vartotojas jau turi kompiuteryje sugeneruotą slapuką ir (netiesiogine prasme) prieigą prie serverio, duomenys būtų pakeičiami vartotojui net neįtariant. Tai tik pats paprasčiausias CSRF atakos pavyzdys.

4.2 Tinklalapis apsaugotas nuo CSRF atakų papildoma laiko žyme

Norint išbandyti laiko žymės panaudojimo efektyvumą buvo sukurta antroji ir trečioji sistemos apsaugotas nuo CSRF atakų. Kaip minėjau ankstesniuose skyreliuose, apsaugojimas nuo CSRF tipo atakų panaudojant tik laiko žymę – nerekomenduotinas. Išlieka galimybė, kad naudojant tokį apsaugos tipą, vis tiek bus galima įvykdyti CSRF tipo ataką, nes ji bus įvykdyta anksčiau, nei bus pradėtos blokuoti atitinkamo vartotojo užklausos. Realizuojant tinklalapį, buvo panaudotas anksčiau sukurta sistema, kuri buvo neapsaugota nuo CSRF tipo atakų. Kuriant apsaugą su laiko žyme stengtasi sukurti tokį modulį, kurio pagalba sistemos apsauga galima būtų padidinti kiek įmanoma mažiau keičiant originalų tinklalapio išeities kodą.

Jungiantis prie sistemos vartotojas nukreipiamas į prisijungimo puslapį, kaip ir pirmoje svetainėje, tačiau dabar ne tik patvirtinami vartotojo duomenys (t.y. nustatoma ar vartotojas egzistuoja ir ar slaptažodis yra teisingas), bet ir esant sėkmingam autentifikavimuisi į duomenų bazę įrašomas dabartinis laikas, taip pat sugeneruojamas ir atsitiktinė CSRF žymė, kuri vėliau galės būti panaudota papildomai apsaugai ir įrašinėjant laiką į duomenų bazę, kad patikrinti POST užklausos į serverį autentiškumą. Ši laiko žymė bus tikrinama prieš įvykdant POST užklausas. Prisijungus prie sistemos vartotojui naršant po tinklalapį, kas kartą fiksuojamas laikas, kada buvo bandyta prieiti prie bet kurio tinklalapio puslapio. Tokiu būdu gaunama informacija ar vartotojas ir toliau aktyviai naudojasi sistema ir ar tai nėra tiesiog paliktas atidarytas skirtukas ar atskiras langas. Vartotojui neatlikus jokių veiksmų bent 10 minučių, POST užklausos jam bus uždraustos iki tol, kol vėl aktyviai pradės naudotis sistema.

Sukurtas modulis įrašantis vartotojo yra ganėtinai paprastas. Jis sukurtas PHP programavimo kalbos pagalba. Kadangi ši programavimo kalba veikia serverio pusėje, o ne kliento, taip sumažinamas apkrovimas vartotojui, todėl bus mažiau nusiskundimų iš vartotojų. Šiuo atveju augant vartotojų kiekiui, reikės galingesnės aparatinės įrangos norint palaikyti tinklalapio greitaveiką. Modulis pilnai realizuotas atskirame faile, todėl gali būti paprastai importuojamas į bet kurį tinklalapio puslapį, užtenka iškviesti vieną funkciją, kuria kraunant tinklalapio puslapį įvykdo serveris. Žemiau pateiktas pavyzdys kaip reikalingas modulis įtraukiamas į sistemą.

```
<?php
   .....
   require_once('CSRF_module.php');
   ......
   store_date();
   ....
?>
```

Kaip matome iš aukščiau pateikto pavyzdžio, kiekvienam puslapiui užtenka įterpti dvi papildomas eilutes, kad modulis būtų aktyvuotas. Pirmoji eilutė nurodo kuriame faile patalpintas modulis ir įterpia jį į dabartinį puslapį. Kadangi modulis jau buvo įtrauktas į puslapį, toliau jau galima naudoti funkcijas aprašytas modulio, kas ir daroma antroje pavyzdžio eilutėje. Iškviečiama funkcija „store_date“, kurios pagrindinė paskirtis yra įrašyti esamą data duomenų bazėje prie vartotojo duomenų. Funkcija neturi jokių parametrų, nes visą reikiamą informaciją apie vartotojo duomenis ji surenka pati.

Pridėjus šį modulį prie sistemos, netampa visiškai apsaugota nuo CSRF tipo atakų, tačiau tai stipriai padidina saugumą jeigu naudojama su anksčiau minėta ir plačiai paplitusia apsauga

panaudojant atsitiktinę reikšmę pridedama prie formos duomenų ir kartu juos siunčiant į serverį. Panaudojus šiuos modulius kartu, būtų pasiekiamas gerokai didesnis saugumas. Jeigu atakuotojui kaip kaip nors pavyktų nustatyti atsitiktinai generuojamą reikšmę, jis turėtų spėti per 10 minučių nuo paskutinio vartotojo naršymo sistemoje pabandyti nusiųsti POST užklausą į serverį.

Galimas ir kitas realizavimo būdas panaudojant JavaScript programavimo kalbą kuri ir buvo panaudota trečioje sistemoje. Kiekviename sistemos puslapyje įterpiamas JavaScript kuris kas penkias minutes bando siųsti atnaujinimą į PHP failą serveryje, kuris atnaujintų datą duomenų bazėje prisijungusiam vartotojui. Programinio kodo pavyzdys pateiktas žemiau.

```
setInterval("update()", 300000);

function update() {

  $.post("update_time.php", {csrf_token: '<%= Session["csrf_token"] %>'});

}
```

Kaip matome iš aukščiau pateikto programinio kodo ištraukos, komanda kartojama kas 300000 milisekundžių (5 minutes). Komanda siunčia POST tipo užklausą į serveryje esantį failą update_time.php kartu siųsdamas ir prisijungimo metu sugeneruota atsitiktinę žymę sesijos kūrimo metu.

PHP failas gavęs POST užklausą pirmiausia patikrina ar atsitiktinė žymė sutampa su saugoma sesijos duomenyse ir jei ji atitinka, atnaujina duomenų bazę su esama data ir laikas. Panaudojama MySQL NOW() funkcija. Komandai nurodžius norimą datos ir laiko formatą, į duomenų bazę įrašoma esama data ir laikas. Kadangi naudojama MySQL komanda, datos visą laiką išliks vienodos, t.y. joms neturės įtakos kurioje laiko zonoje yra vartotojas. Užklausos atnaujinti laiką pavyzdys pateikiamas žemiau.

```
UPDATE users

SET paskutines_užklausos_laikas = NOW()

WHERE ID = $ID
```

Kaip matome iš pavyzdžio užklausau užtenka pateikti prisijungusio vartotojo identifikacinį numerį ir ji atnaujins tam vartotojui „paskutines_uzklausos_laikas“ lauką duomenų bazėje.

4.3 Tinklalapis bandantis pakeisti duomenis pakeisti duomenis apsaugotame ir neapsaugotame tinklalapyje

Ketvirtoji sistema sukurta kaip tariamo atakuotojo tinklalapis. Iš pirmo žvilgsnio tai tėra paprasta nuotraukų galerija. Vartotojas prisijungęs prie sistemos gali be registracijos naršyti puslapius, t.y. peržiūrinėti nuotraukas nieko nenutuokdamas apie esančią grėsmę.

Atidarius ketvirtąją sistemą vartotojui pateikiama eilė paveikslėlių, tačiau be jų dar užkraunamas ir nematomas „iframe“ elementas. Šio elemento pagalba atakuotojas paleidžia piktybinį kodą kuris bando pakeisti duomenis anksčiau aprašytuose tinklalapiuose.

```
<iframe src="http://localhost/site4/update.php" style="display:none"></iframe>
```

Kaip matote iš aukščiau pateikto pavyzdžio atakuotojas pridėjo nematomą iframe elementą prie tinklalapio, kuriame nurodė, kur yra patalpinta iš anksto sugeneruota forma su duomenimis kuriuos reikia atnaujinti. Kiekvieną kartą atidarius tinklalapį, užkraunant paveiksliukus taip pat yra įvykdomas ir programinis kodas esantis update.php faile. Žemiau yra pateiktas pavyzdinis failo update.php variantas naudojamos siųsti duomenis į pažeidžiamas sistemas.

```
<body onload="document.getElementById('f').submit()">

<form id="f" action="http://localhost/site2/update_userinfo.php" method="post" name="form1">

<input name="uname" value="atakuotojo nustatytas vardas">
```

<input name="uphone" value=" atakuotojo nustatytas telefono numeris ">

<input name="uemail" value=" atakuotojo nustatytas el. paštas ">

<input name="update" value="update">

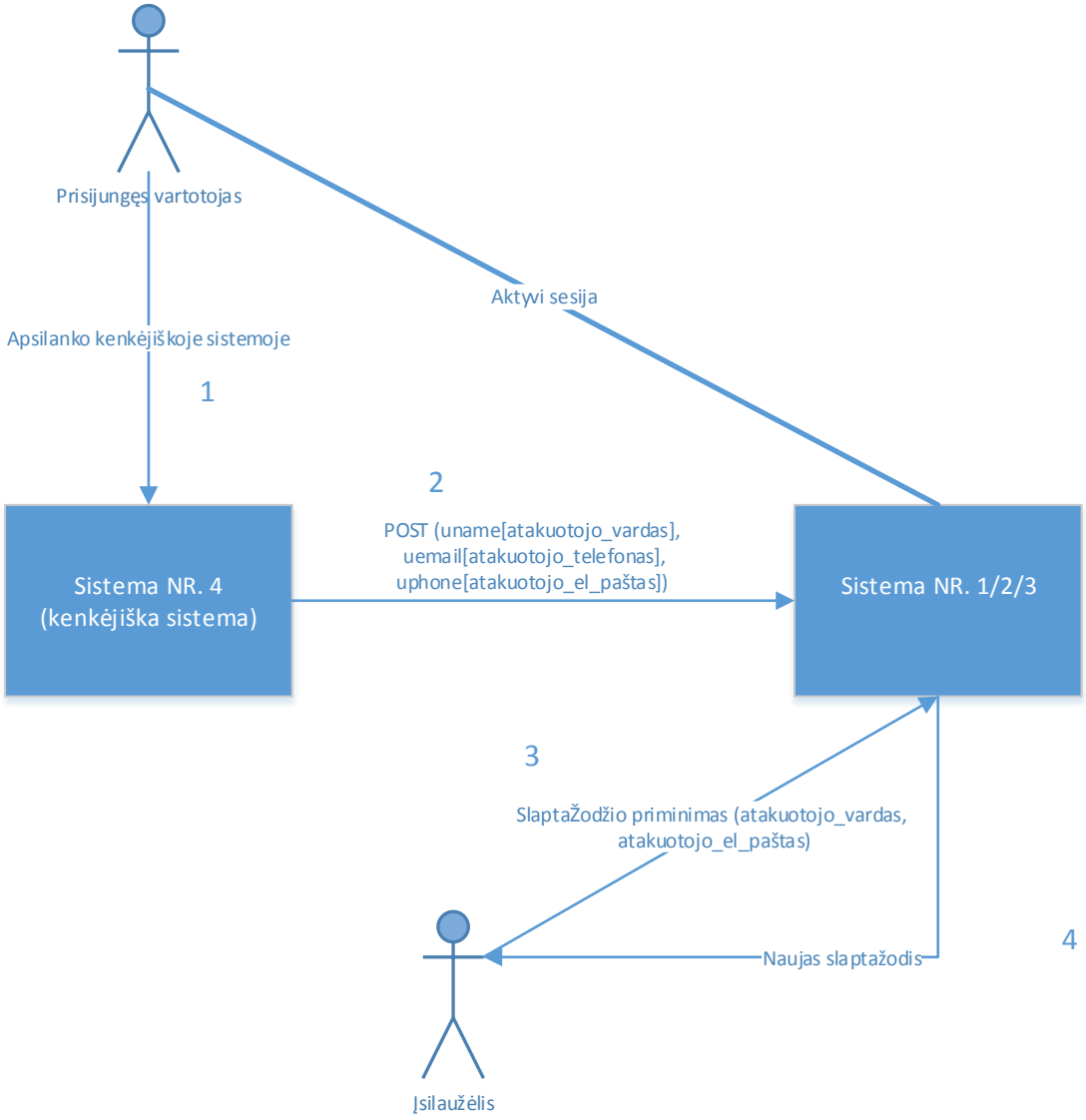
</form>

</body>

Pateiktas pavyzdys yra labai paprastas, bet jis skirtas tik patikrinti patį principą kaip atliekamos tokio tipo ir nustatyti ar laiko žymės naudojimas gali būti naudingas apsaugant sistemą nuo CSRF tipo atakų. Kaip matome formoje yra iškarto apibrėžtos kelios prisijungusio vartotojo asmeninių duomenų reikšmės, kurias atakuotojas kurdamas formą nurodo iš anksto. Atakuotojas bando pakeisti tris vartotojo parametrus:

- Pilną vartotojo vardą,
- Vartotojo telefono numerį,
- Vartotojo el. pašto adresą.

Žemiau pateikiamas tokio atakos pavyzdys. Paveikslėlyje parodomas atakos scenarijus, kurios metu bandoma perimti visos paskyros kontrolę.



Pav. 8 Atakos scenarijus

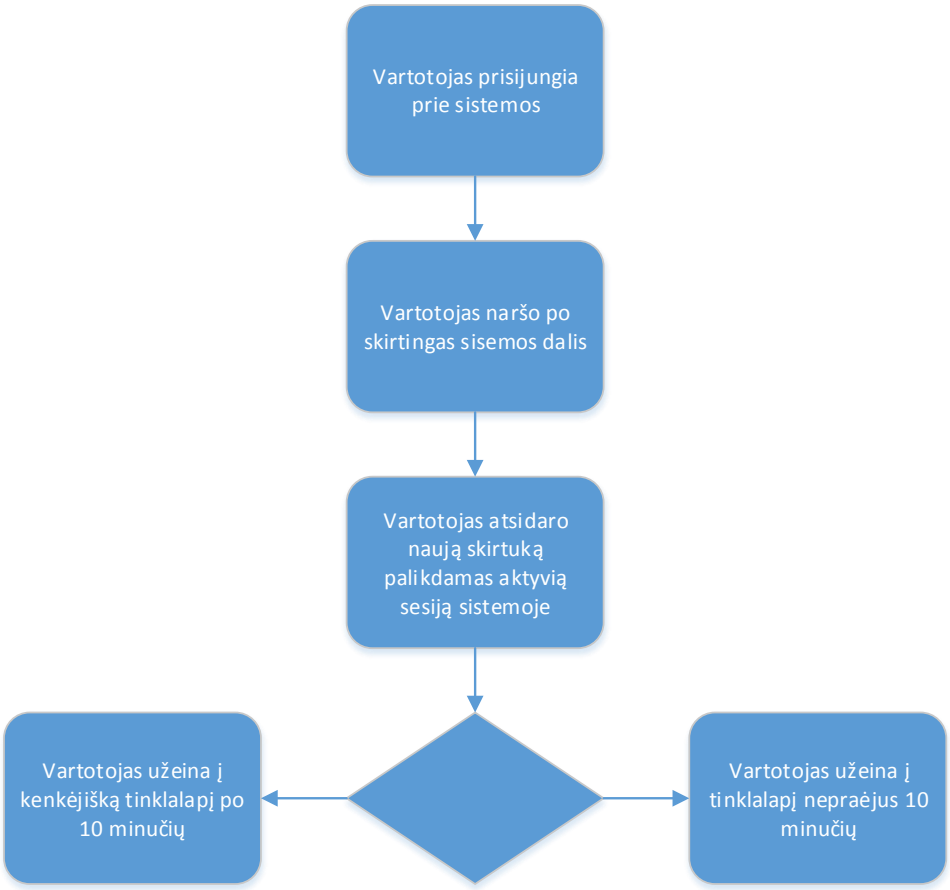
Kaip matome iš aukščiau pateikto paveikslėlio, sėkmingai įvykdžius ataką prisijungusio vartotojo duomenys būtų pakeisti. Tokiu būdų atakuotojas gali perimti pilną paskyros kontrolę pasinaudodamas slaptažodžio priminimo funkcija. Toks paskyros perėmimas gali būti pavojingas ir pačiai sistemai. Tuo metu prisijungęs ir į atakuotojo svetainę užklydęs vartotojas gali būti administratorius. Atakuotojas gavęs pilna administratoriaus paskyros kontrolę, gali pridaryti daug problemų pačiai sistemai, kurias būtų galima ištaisyti nebent atkuriant pačią sistemą iš atsarginių kopijų. Tuo tarpu pats vartotojas niekaip negali atskirti, kad tapo įsilaužimo auka iki tol, kol pabando prisijungti prie perimtos paskyros ankstesniame tinklalapyje. Būtent dėl šios priežasties tokio tipo atakos yra ypač populiarios tarp įsilaužėlių. Dažniausiai duomenys gali būti keičiami netiesiogiai, t.y. bandoma atnaujinti tokius vartotojo duomenis kurie nepraneštų apie įsilaužėlį. Pavyzdžiui tokia spraga buvo aptikta tinklalapyje „vimeo“ , kuris skirtas talpinti vaizdo įrašams. Atakuotojas galėjo pridėti bet kokį filmuką prie peržiūrėjimo sąrašo bei priversti vartotoją paspausti “ patinka“ [17]. Surasti žmones kurie inicijuoja CSRF tipo atakas labai sudėtinga, kadangi visos užklausos atliekamos paties vartotojo naršyklės.

4.3 Sistemų testavimas

Baigus programavimo darbus, buvo atliekami sistemų testavimo ir darbai, kurių metu buvo bandoma išsiaiškinti realizuoto metodo efektyvumą. Bandymu metu buvo bandoma imituoti įprasto vartotojo naršymą. Vartotojas pirmiausiai prisijungia prie pirmos, antros arba trečios sistemos. Šiek tiek panaršęs pačioje sistemoje jis neatsijungęs nuo sistemos atsidaro naują skirtuką naršyklėje (tuo metu jo sesija lieka vis dar aktyvi) jis toliau tęsia naršymą internete. Po kurio laiko, užeinama į kenkėjišką sistemą, kuri bando išnaudoti prisijungusio vartotojo sesiją ir nusiųsti POST tipo užklausą į kitas sistemas. Toliau tęsiant testavimą galimi du atvejai:

- Vartotojas užėjo į kenkėjišką sistemą nepraėjus 10 minučių.
- Vartotojas užėjo į kenkėjišką sistemą praėjus daugiau nei 10 minučių.

Žemiau pateikiama sekų diagrama kurioje pavaizduojama koki žingsniai buvo atliekami testavimų metu. Pateiktas testo scenarijus buvo kartojamus tris kartus, tam, kad išbandyti kiekvienos sistemos saugumą atskirai.



Pav. 9 Testavimo scenarijus

Pirmoji sistema yra visiškai neapsaugota nuo CSRF tipo atakų, todėl nesvarbu kada būtų generuojama POST užklausa, ji vis tiek priims ją kaip teisingą ir įvykdys ją, todėl ketvirtoji sistema sėkmingai įvykdė CSRF ataką visais kartais. Likusioms dvejoms sistemoms, kurios yra apsaugotos tik naudojant papildomą laiko žymę, užklausos laikas yra svarbus. Pirmuoju atveju, kai vartotojas užėjo į kenkėjišką sistemą nepraėjus 10 minučių, ataka nebuvo atremta nes nebuvo naudojamos jokios papildomos apsaugo sistemos, tam, kad supaprastinti pačio metodo testavimą. Antruoju atveju, kai vartotojo naršymas internete ir patekimas į kenkėjišką sistemą trukdavo ilgiau nei dešimt minučių, atakos buvo atremtos sėkmingai tiek antroje tiek trečioje sistemose. Apžvelgus testų metu gautus rezultatus buvo prieita prie išvados, kad norint efektyviai apsaugoti sistemą yra būtina šį apsaugos metodą derinti kartu su kitais, anksčiau jau minėtais,

apsaugos metodais. Vienas pats šis metodas nėra pakankamai efektyvus, kad galėtų atremti visas CSRF tipo atakas. Lyginant abi apsaugotas sistemas, efektyvesnis sprendimas yra naudoti laiko atnaujinimą pasitelkiant JavaScript programavimo kalbą. Kadangi metodas yra visada aktyvus, galima sumažinti laiko tarpą, po kurio užklausos būtų atmetamos.

4.4 Eksperimento išvados

Pagal aprašytas tyrimo metodikas buvo atliekami programavimo darbai, tam, kad išsiaiškintume pasiūlytos idėjos efektyvumą. Testavimui buvo sukurtos keturios sistemos. Pirmojoje sistemoje buvo nenaudojamos jokios apsaugos priemonės, tam kad išsiaiškintume visus įmanomus ir aukščiau aprašytus pažeidžiamumus. Antrojoje sistemoje buvo daugiausiai naudojama PHP programavimo kalba bei įdiegti saugumo sprendimai. Trečioji sistema realizavo JavaScript programavimo kalbos apsaugos metodus. Ketvirtojoje sistemoje sukurtas CSRF atakos pavyzdys. Atlikus veiksmus buvo tikrinami rezultatai ir apsaugos patikimumas. Sistemose realizuotas apsisaugojimas nuo CSRF atakų, panaudojus laiko žymę. Ketvirtoji sistema buvo sukurta kaip pavyzdinė atakuotojo sistema. Vartotojui atsidarius tinklalapį, jam parodomas paprastas paveikslėlis, tačiau fone bandoma siųsti POST tipo užklausą į anksčiau minėtus tinklalapius.

Papildomos apsaugos priemonės nebuvo naudojamos norint supaprastinti sistemos testavimą, kitu atvejų būtų reikėję papildomų veiksmų norint apeiti kitas apsaugos priemones. Papildoma laiko žymė negali būti naudojama kaip vienintelė apsaugos priemonė, tačiau ji stipriai padidina sistemos saugumą, jeigu yra realizuotos anksčiau minėtos apsaugos priemonės.

Ketvirtoji sistema (atakuotojo tinklalapio pavyzdys) . Sugebėjo sėkmingai atlikti CSRF tipo ataką prieš pirmąją (neapsaugotą) sistemą. Kadangi pirmojoje sistemoje nebuvo realizuota jokio apsauga, įsilaužimą įvykdyti buvo galima be didelių pastangų. Vartotojo duomenys buvo sėkmingai atnaujinti iš anksto nustatytomis reikšmėmis. Norint nustatyti aprašyto metodo efektyvumą, buvo bandoma imituoti paprasto vartotojo naršymą Internete. Vartotojas pradžioje trumpai panaršė sistemoje, tam, kad būtų užregistruota paskutinio veiksmo data, vėliau jis perėjo prie kitų tinklalapių. Vienas iš jo vėliau aplankytų tinklalapių buvo kenkėjiškas. Kadangi paprastai nuoroda į kenkėjišką svetainę būna patalpinta kitoje, nekenkėjiškoje, svetainėje, tai pirmu bandymu neišėjo sėkmingai įvykdyti CSRF atakos prieš antrąją sistemą Ataka sėkmingai buvo atremta, nes naršymas truko ilgiau nei 10 minučių. Antruoju bandymu vartotojas į kenkėjišką tinklalapį užėjo greičiau nei po dešimt minučių, todėl CSRF ataka buvo sėkminga. Duomenys buvo pakeisti sėkmingai.

Apibendrinant eksperimentą reikia pabrėžti, kad negalima remtis vien tik laiko žyme norint apsaugoti sistemą nuo CSRF tipo atakų. Norint efektyviai apsaugoti sistemą yra būtina šį apsaugos metodą derinti kartu su kitais, anksčiau jau minėtais, apsaugos metodais. Vienas pats šis metodas nėra pakankamai efektyvus, kad galėtų atremti visas CSRF tipo atakas. Lyginant abi apsaugotas sistemas, efektyvesnis sprendimas yra naudoti laiko atnaujinimą pasitelkiant JavaScript programavimo kalbą. Kadangi metodas yra visada aktyvus, galima sumažinti laiko tarpą, po kurio užklausos būtų atmetamos. Pavyzdžiui nuo dešimt minučių iki penkių (ar net dviejų) kas stipriai padidintų sistemos saugumą.

5. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

- Išanalizavus saugumo problemas internetinėse sistemose, buvo pastebėta, kad CSRF tipo pažeidžiamumai yra pakankamai dažnai randami įvairiose sistemose. Tik prieš kelerius metus į šio tipo atakos pradėta žiūrėti rimtai, tačiau dauguma programuotojų jas vis dar nuvertina kaip nesvarbias. Panagrinėjus plačiau šio tipo atakas buvo išsiaiškinta, kad jos yra nuvertinamos nepelnytai ir neužtikrinus pilnos apsaugos nuo jų galima turėti skaudžių pasekmių.
- Išnagrinėti dažniausiai naudojami ir siūlomi metodai apsaugai nuo CSRF atakų, įvertinti jų privalumai ir trūkumai. Nustatyti patogiausi ir efektyviausi apsaugos metodai naujai kuriamose sistemose bei jau esamose. Išnagrinėti populiariausi karkasai naudojami sistemų apsaugai.
- Išanalizavus esamus apsaugos metodus, buvo sumodeliuotas papildomas apsaugos metodas sistemoms apsaugoti. Metodo pagrindinis principas yra periodiškai saugoti vartotojo kreipimosi datą sistemoje. Vėliau šią datą galima panaudoti patvirtinant arba atmetant vartotojo užklausas į serverį. Orientuotasi tik į POST tipo užklausas, nes GET tipo užklausos saugu naudoti tik informacijos pateikimui, o ne modifikavimui.
- Pasiūlytas apsaugos metodas buvo sumodeliuotas realiai sistemai, kad vėliau būtų galima išbandyti jo efektyvumą testavimams skirtose sistemose.
- Buvo sukurtos keturios testavimui skirtos sistemos. Pirmoji buvo neapsaugota jokiu metodu, antroji apsaugota naujai pasiūlytu apsaugos metodu panaudojant daugiausiai PHP programavimo kalbą. Trečioji sistema apsaugota naujai pasiūlytu metodu tačiau naudojant daugiau JavaScript programavimo kalbą. Ketvirtoji sistema buvo pavyzdinė atakuotojo sistema kurios pagalba buvo išbandomas pasiūlyto metodo efektyvumas.
- Apibendrinant gautus rezultatus galima įvertinti, kad pasiūlytas metodas gali stipriai padidinti sistemos saugumą, tačiau šio metodo negalima naudoti kaip vienintelės apsaugos nuo CSRF tipo atakų. Naudojant šį metodą išlieka tikimybė, kad ataka bus vykdoma anksčiau negu 10 minučių po paskutinio užregistruoto vartotojo veiksmo sistemoje. Bandymų atveju buvo naudojamas 10 minučių laiko tarpas, po kurio buvo atmetamos užklausos į serverį.

6. LITERATŪRA

[1] "internetlivestats," [Online]. Available: http://www.internetlivestats.com/internet-users/. [Accessed 2 Balandis 2015].

[2] T. J. a. P. Taylor, "A sociology of hackers," 2010.

[3] V. Sabadash, "crime-research.org," [Online]. Available: http://www.crime-research.org/news/10.05.2004/246/. [Accessed 20 Rugsėjo 2014].

[4] nsfocus.com, "Analysis of DDoS Attacks on Spamhaus and," http://www.nsfocus.com/SecurityView/Analysis%20of%20DDoS%20Attacks%20on%20Spamhaus%20and%20recommended%20solution-EN-20130510.pdf, 2013.

[5] R. McMillan, "csoonline.com," [Online]. Available: http://www.csoonline.com/article/2122737/identity-theft-prevention/tjx-staffer-sacked-after-talking-about-security-problems.html. [Accessed 7 Balandžio 2015].

[6] S. Kamkar, "Samy's personal web page," [Online]. Available: http://namb.la/popular/. [Accessed 01 05 2015].

[7] Menlo_Security_Vulnerability_Report_Mar_2015, "menlosecurity.com," 2015.

[8] J. Balhis, "securityfocus.com," VULNERABILITY LABORATORY, [Online]. Available: http://www.securityfocus.com/archive/1/534682. [Accessed 1 Gegužė 2015].

[9] W. Z. a. E. W. Felten, "Cross-Site Request Forgeries: Exploitation and Prevention," 2008.

[10] htmlpurifier, "HTML Purifier," [Online]. Available: http://htmlpurifier.org/live/configdoc/plain.html. [Accessed 1 Gegužės 2015].

[11] yiiframework, "yiiframework," [Online]. Available: http://www.yiiframework.com/features/. [Accessed 15 Vasario 2015].

[12] django, "django," [Online]. Available: https://www.djangoproject.com/start/overview/. [Accessed 1 Gegužė 2015].

[13] "tiobe.com," [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Accessed 02 Balandis 2015].

[14] "mashable.com," [Online]. Available: http://mashable.com/2015/01/18/programming-languages-2015/. [Accessed 2 Balandis 2015].

[15] J. C. P. G. D. B. Michael S. Mikowski, Single Page Web Applications, 2014.

[16] builtwith, "builtwith," [Online]. Available: http://trends.builtwith.com/framework/programming-language. [Accessed 1 Gegužė 2015].

[17] E. Homakov, "www.sakurity.com," [Online]. [Accessed 1 Sausio 2015].

[18] E. W. William Zeller, "Cross-Site Request Forgeries: Exploitation and Prevention," 2008.

[19] Litnet_komanda, "cert.litnet.lt," [Online]. Available: https://cert.litnet.lt/lt/dokumentai/pagrindiniai-web-sistemu-pazeidziamumai-ir-saugos-budai. [Accessed 13 Sausio 2015].

[20] B. Zeller, "freedom-to-tinker.com," [Online]. Available: https://freedom-to-tinker.com/blog/wzeller/popular-websites-vulnerable-cross-site-request-forgery-attacks/. [Accessed 03 Gegužė 2015].

[21] J. W. M. Johns, "Protecting the Intranet Against "JavaScript Malware" and Related Attacks.," in Detection of Intrusions and Malware, and Vulnerability Assessment, 2007.

[22] J. Grossman, "CSRF, the sleeping giant," [Online]. Available: http://jeremiahgrossman.blogspot.com/2006/09/csrf-sleeping-giant.html. [Accessed 14 Vasario 2015].

[23] C. Shiflett, "Security corner," [Online]. Available: http://shiflett.org/articles/cross-site-request-forgeries. [Accessed 15 Kovo 2014].

[24] P. Z. R. R. a. D. L. Boyan Chen, "A Study of the Effectiveness of CSRF Guard," 2011.

[25] P. Z. R. R. D. L. Xiaoli Lin, "Threat Modeling for CSRF Attacks," 2009.

[26] M. S. Siddiqui, "Cross Site Request Forgery: A common web application weakness," 2011.