УНИВЕРЗИТЕТ СИНГИДУНУМ
ФАКУЛТЕТ ЗА ИНФОРМАТИКУ И РАЧУНАРСТВО

Фалсификовање унакрсних захтева

- дипломски рад -

Ментор:
др Александар Јевремовић

Кандидат:
Милош Милановић

Београд, 2016.

Садржај

1 Увод....3
2 Основни појмови....4
2.1 Униформни локатор ресурса....4
2.2 Протокол за пренос хипертекста....4
2.2.1 Колачићи....6
2.2.2 Безбедна варијанта протокола за пренос хипертекста....7
2.3 Појам унакрсних захтева....8
2.4 Програмирање на страни сервера....9
2.4.1 Механизам сесије код PHP програмског језика....11
3 Напади на Вебу....12
3.1 Уметање SQL кода....12
3.2 Унакрсно скриптовање....13
3.2.1 Ускладиштени XSS....14
3.2.2 Рефлектовани XSS....14
3.2.3 DOM-базирани XSS....14
3.2.4 Полиса о заштити садржаја....15
3.3 Фалсификовање унакрсних захтева....15
3.3.1 CSRF напад и HTTP методе....17
4 Заштита код фалсификовања унакрсних захтева....19
4.1 Коришћење токена за ауторизацију....19
4.1.1 Анализа заштите код Laravel апликативног оквира....20
4.2 Коришћење CAPTCHA механизма и ре-аутентификација....22
4.3 Заштита на страни сервера....23
4.4 Заштита на страни корисника....24
4.4.1 Заштита унутар Веб-прегледача....24
4.5 Увођење новог атрибута за колачиће....26
5 Предлог сопственог решења за заштиту....27
5.1 Класа за манипулацију токенима....27
6 Закључак....30
7 Литература....31

1 Увод

Од како је 1990. године настао Веб, Интернет је у многим аспектима променио начин живота друштва. Гугл, Фејсбук, Амазон и сличне компаније мењају начин на који комуницирамо, послујемо, купујемо. Људи су сада повезанији него пре захваљујући друштвеним мрежама као што су Фејсбук и Твитер. Гуглов претраживач и Википедија мењају начин на који се друштво информише и учи. Подаци се више не складиште на физичким медијумима, као што су CD, DVD, USB и сл., већ се све чешће људи опредељују за неке од сервиса за складиштење на Вебу, као што су *Dropbox* или *Drive* компаније Гугл. Електронска трговина је такође променила начин пословања многих компанија, можда не колико се очекивало у *dot-com-bubble* ери, али свакако је пословање многих више незамисливо без Веба као инфраструктуре. Са тим интензитетом селидбе вредности из стварног света у виртуални, расту и многи ризици у вези безбедности не само тих вредности, већ и људи који се служе њима.

Кроз само постојање Веба истакли су се напади као што су уметање SQL кода (енгл. *SQL Injection*), напади унакрсним скриптовањем (енгл. *Cross-Site Scripting - XSS*), фалсификовање унакрсних захтева (енгл. *Cross-Site Request Forgery - CSRF*) и напади ускраћивањем услуге (енгл. *Denial of Service - DoS*). Док су за прва три поменута напада решења за заштиту већ одавно позната, разумевање истих је битно не само због саме заштите, већ и због дубљег разумевања осталих напада на Вебу. У овом раду биће представљен трећи по реду - фалсификовање унакрсних захтева. Детаљно ће бити објашњено како напад функционише, биће размотрена популарна решења за заштиту од таквих напада, а на крају ће бити представљено и решење за заштиту развијено од стране аутора овог рада. Размотрићемо предности и мане свих безбедносних решења и извести закључак о тренутном стању заштите код оваквог типа напада.

Код избора ове теме аутору је намера била да стекне дубљи ниво разумевања заштите на самом Вебу, истовремено не анулирајући и остале слојеве комуникације на Интернету. Као што је један од најзначајнијих српских књижевника Борислав Пекић у свом роману *„Беснило“* написао, са безбедношћу све има везе. У случају заштите на Вебу то подразумева не само безбедност на апликативном слоју, већ и на нижим слојевима рачунарске комуникације, безбедност прекоокеанских каблова који повезују чворове на Интернету, па и људски фактор, у смислу обучавања клијената за безбедан рад на Вебу. У складу са тим, у овом раду биће скренута пажња и на неке добре праксе које помажу са заштитом код оваквог типа напада, а тичу се самих корисника.

2 Основни појмови

У уводу рада напоменуто је колики је значај Интернета, како у друштвеном, тако и у пословном аспекту данашњице. Али велику заслугу за толику популарност и устаљеност у друштву Интернет дугује Вебу, сервису који користи Интернет мрежу као инфраструктуру. Иако је током постојања Веб доживљавао значајне измене, у основи то је и даље светска мрежа (енгл. *World Wide Web*) која служи да омогући приступ удаљеним ресурсима који се налазе на рачунарима који се називају сервери. Страна која приступа ресурсу се назива клијент. Дакле, Веб је мрежа која користи клијент-сервер архитектуру за приступ удаљеним ресурсима.

Сами ресурси сами по себи најчешће не служе ничему, већ се користе у склопу много веће целине, која може бити презентација, апликација или сервис. Под појмом презентације се подразумева статичан сајт, код кога се користе само клијентске технологије (нпр. CSS, JavaScript). Под појмом апликације се подразумева динамичан сајт, код кога постоји програмски језик на страни сервера (нпр. PHP). Под појмом сервиса подразумева се испорука података који ће бити коришћени најчешће од стране неке друге апликације (или сервиса). За разлику од апликације и презентације који користе HTML (енгл. *HyperText Markup Language*) стандард за слој приказа, код сервиса се користе стандарди типа JSON (енгл. *JavaScript Object Notation*) или XML (енгл. *Extensible Markup Language*), који су за разлику од HTML стандарда који је прављен за *human-to-machine* комуникацију прављени за употребу у *machine-to-machine* комуникацији.

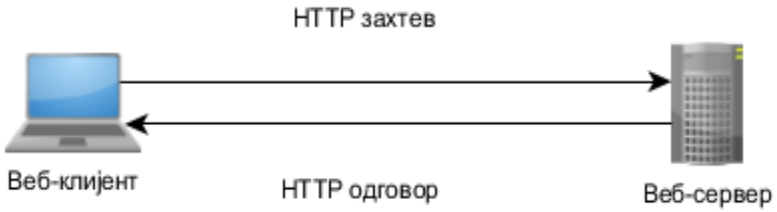
2.1 Униформни локатор ресурса

Сваки ресурс на Вебу одређен је својим униформним локатором (енгл. *Uniform Resource Locator - URL*). Униформни локатор ресурса представља стандард који дефинише локацију ресурса, односно локацију ка којој клијент треба да упути захтев да би затражио приступ том ресурсу. Раније је локатор сваког ресурса био уско повезан са његовом путањом на самом фајл-систему сервера, међутим у модерном MVC приступу са механизмом рутирања захтева ово правило више не важи. Сваки униформни локатор се састоји од више делова, а обавезни делови су шема (FTP, HTTP, HTTPS...) и адреса домаћина - симболичка (домен) или логичка (адреса Интернет протокола). Остали делови униформног локатора ресурса су порт, путања, упит и фрагмент. Стандард такође омогућава и аутентификацију коришћењем корисничког имена и лозинке, међутим овакав приступ се не препоручује из безбедносних разлога.

2.2 Протокол за пренос хипертекста

Протокол по којем се обавља комуникација између клијента и сервера се назива протокол за пренос хипертекста (енгл. *HyperText Transfer Protocol - HTTP*). Важно је напоменути да се сам протокол не користи само за пренос HTML докумената, већ и

за пренос CSS докумената, слика, аудио и видео записа и сл. Протокол користи већ поменуту клијент-сервер архитектуру и ресурсе испоручује сервер у виду одговора на клијентов захтев. Сам Веб-сервер представља сервис на удаљеном рачунару (серверу) који чека на захтеве од клијента, а након што захтев стигне обрађује га и прослеђује клијенту. Под Веб-клијентом се подразумева било који софтвер који може да шаље HTTP захтев. У већини случајева то је GUI (енгл. *Graphical User Interface*) софтвер који се назива Веб-прегледач (енгл. *Web Browser*), али не треба заборавити ни CLI (енгл. *Command Line Interface*) алате као што је cURL.



Слика 1: Протокол за пренос хипертекста

При комуникацији између клијента и сервера, битно је да сви ресурси буду испоручени клијенту. То значи да ће протокол који се користи на нивоу испод, тј. на транспортном слоју, бити протокол за контролу преноса (енгл. *Transmission Control Protocol - TCP*). Сам протокол не обезбеђује никакву заштиту у виду шифровања или аутентификације сервера, зато се за повећање безбедности користи HTTPS (енлг. *HTTP Secure*) варијанта. Подразумевани порт на коме сервер ради је 80 за HTTP, 443 за HTTPS.

Током HTTP комуникације, клијент шаље захтев, на шта сервер шаље одговор. Структура захтева и одговора је слична, и код једног и код другог постоји заглавље и тело, мада се код захтева тело најчешће изоставља. Заглавље се састоји од поља, чије вредности дефинишу параметре комуникације. Нека поља су заједничка за захтев и одговор (нпр. Cache-Control, Connection, Date...), нека поља се односе само на захтев (нпр. Accept, Host, User-Agent...), а нека само на одговор (нпр. Location, Set-Cookie, Server...), док у случају да HTTP порука садржи и тело, постоје специјална поља која се односе на сам садржај (нпр. Allow, Content-Length...).

Сваки HTTP захтев започиње са линијом која дефинише HTTP метод и ресурс коме се захтева приступ, док сваки HTTP одговор започиње са линијом која дефинише кодом статуса одговора. Метод поруке може бити GET, POST, PUT, DELETE, TRACE итд., мада се прва два наведена најчешће користе у пракси. Кодови статуса су троцифрени бројеви који извештавају клијентску страну о статусу одговора. Постоји пет група кодова статуса, а разликују се по почетној цифри, нпр. ако кôд статуса започиње са двојком, тада сервер извештава клијента да је захтев успешно обрађен (нпр. 200 OK), док у случају да кôд статуса започиње са четворком, сервер извештава клијента да је при слању захтева дошло до грешке на клијентској страни (нпр. 404 Not Found).

GET метод се најчешће користи и служи за једноставан захтев преузимања удаљеног ресурса. У случају да се шаљу додатни параметри серверу, они се шаљу кроз униформни локатор ресурса који је наведен у Host пољу. Због тога, GET захтев није погодан за слање осетљивих података као што су лични подаци клијента (нпр. шифре). У том случају користи се POST захтев, који у принципу ради исто што и GET захтев, само што елиминише слање параметара кроз униформни локатор

ресурса. Једна од предност POST захтева је и што је сама дужина униформног локатора ресурса ограничена серверским подешавањима, па при слању веће количине података долази до одсецања дела локатора који прелази одговарајућу дужину.

Такође је дозвољено и коришћење произвољних поља унутар заглавља и метода. Код дефинисања произвољних метода у случају да очекујемо одговор та метода мора бити дефинисана и на страни сервера, а произвољна поља се могу користити при самој обради захтева. Нека од њих су се усталила као de facto стандарди (нпр. X-Requested-With, X-Forwarded-For...), а једно од тих поља се користи баш за заштиту од фалсификовања унакрсних захтева (X-Csrf-Token). Сама чињеница да је могуће додавати сопствена поља и мењати вредности постојећих говори да вредностима поља унутар заглавља не треба веровати, тј. треба користити неки вид ауторизације када се извршавају операције које могу променити стање сервера. Алати као што су већ поменути cURL или проширења за Mozilla Firefox Веб-прегледач као што је Tamper Data нам дају могућност за измену вредности поља унутар заглавља.

2.2.1 Колачићи

Протокол за пренос хипертекста спада у stateless протоколе, тј. сваки наредни захтев је независан у односу на претходни. Да би сачували одређено стање у ком се апликација налази (нпр. пријављивање корисника, избор језика/писма странице...) користимо HTTP колачиће (енгл. *HTTP Cookies*). Колачићи представљају имплементацију меморије на страни клијента (унутар Веб-прегледача) у коју сервер може да ускладишти податке везане за тренутну сесију клијента. Сваки колачић има структуру кључ-вредност. Пример коришћења колачића:

1. Корисник (клијент) се пријављује на Веб-апликацију користећи своје креденцијале (нпр. корисничко име и хеш-вредност лозинке) путем POST захтева
2. Веб-сервер обрађује захтев и врши проверу да ли постоји одговарајуће корисничко име са хешом лозинке у бази података
   2.1. У случају да не постоји, аутентификација је неуспешна
   2.2. У случају да постоји сервер поставља колачић на страни клијента помоћу HTTP заглавља Set-Cookie
3. На страни клијента сада постоји колачић који се шаље при сваком следећем HTTP захтеву помоћу заглавља Cookie
4. Сервер сада сваки пут може да провери да ли је клијент пријављен провером да ли постоји одговарајући колачић унутар HTTP захтева

Напомена: овакав приступ имплементације аутентификације није сигуран јер би нападач могао да погоди лако вредност колачића нпр. коришћењем методе сирове силе (енгл. *brute-force attack*). За отежавање овакве врсте напада користи се механизам сесије који се имплементира преко програмског језика на страни сервера.

Постоје такође и други механизми чувања података на страни клијента (нпр. *Local Storage API*), али је коришћење колачића најпопуларнији начин за решавање

проблема памћења стања код HTTP протокола. Колачиће је могуће поставити на више начина, а неки од њих су:

- Коришћењем PHP језика - setcookie() функција
- Коришћењем JavaScript језика - помоћу document.cookie атрибута
- Коришћењем mod_headers проширења код Apache Веб-сервера - Header Set директива

2.2.2 Безбедна варијанта протокола за пренос хипертекста

Обична варијанта HTTP протокола не обезбеђује никакву додатну безбедност у виду шифровања или аутентификације сервера. Из тог разлога развијена је и безбедна варијанта овог протокола која користи SSL (енгл. *Secure Sockets Layer*) или TLS (енгл. *Transport Layer Security*) протокол. SSL/TLS су криптолошки протоколи који служе за креирање безбедног канала преко незаштићене мреже (Интернета). Оба протокола користе криптографске функције за обезбеђивање сервиса тајности и аутентификације.

Безбедност код SSL/TLS протокола је заснована на сертификационим телима (енгл. *Certificate Authority - CA*). Сертификационо тело представља трећу страну од поверења у комуникацији између клијента и сервера. У HTTPS комуникацији, само сервер треба да се идентификује клијенту, што значи да сервер треба да поседује дигитални сертификат издат од стране неког сертификационог тела. Поступак издавања дигиталног сертификата је следећи:

1. На серверу се генерише пар кључева (јавни и приватни), најчешће коришћењем OpenSSL библиотеке
2. Сервер шаље јавни кључ одговарајућем сертификационом телу, где се повезује одговарајући јавни кључ са одговарајућим идентитетом и издаје се сертификат
3. Сертификат се издаје са одређеним периодом важења, након тога потребно је обновити процес сертификације

При HTTPS комуникацији, клијент и сервер користе више криптолошких функција:

- Размена кључа - клијент и сервер се договарају о међусобном сесијском кључу. Овде се најчешће користи DH (енгл. *Diffie-Hellman*) алгоритам и његова варијација која користи ECC (енгл. *Elliptic Curve Cryptography*).
- Дигитално потписивање - сервер се аутентификује пред клијентом користећи свој приватан кључ за потпис, док на другој страни клијент може верификовати потпис користећи јавни кључ сервера, прослеђен у сертификату. Овде се најчешће користи RSA (енгл. *Rivest-Shamir-Adleman*) алгоритам, док се потпис генерише користећи неку од хеш-функција - MD5 (енгл. *Message Digest*) или SHA-2 и SHA-3 (енгл. *Secure Hash Algorithm*) алгоритми.
- Шифровање - комуникација је шифрована користећи неке од симетричних алгоритама шифровања као што су AES (енгл. *Advanced Encryption Standard*)

или троструки DES (енгл. *Data Encryption Standard*) алгоритми. Увек се користи CBC (енгл. *Cipher Block Chaining*) режим рада, што имплицира и на коришћење иницијализационог вектора при шифровању/дешифровању.

HTTPS протокол игра велику улогу у заштити модерних Веб-апликација. У случају снимања мрежног саобраћаја, сав садржај комуникације ће бити доступан нападачу. Коришћење DH протокола обезбеђује готово сигурно безбедну комуникацију између клијента и сервера, чак и у случају да нападач снима саобраћај између ове две стране. Криптологија игра велику улогу у безбедној комуникацији на Интернету, многи Интернет сервиси као што су SSH (енгл. *Secure Shell*) и виртуалне приватне мреже су само неки који се ослањају на криптолошке механизме. Ипак, и криптолошки механизми су подложни нападима, па се у циљу веће безбедности и заштите од MITM (енгл. *Man-in-the-middle*) напада предлажу се следећи видови заштите:

- На страни сервера - коришћење HSTS (енгл. *HTTP Strict Transport Security*) заглавља за редирекцију са http на https протокол и коришћење HPKP (енгл. *HTTP Public Key Pinning*) механизма за спречавање MITM напада на HTTP протокол
- На страни клијента - коришћење проширења HTTPS Everywhere за Веб-прегледаче

Такође, издвојићемо и Let's Encrypt CA које се појавило 2016. године и нуди бесплатне сертификате у циљу свеприсутности HTTPS протокола. Ова иницијатива даје наду да је време када ће цео саобраћај на Вебу бити шифрован јако близу.

2.3 Појам унакрсних захтева

Под унакрсним захтевом (енгл. *Cross-site request*) подразумевано захтев који прави ресурс када упути захтев ка ресурсу са различитог домена. Данас су унакрсни захтеви све популарнији из разлога што се већина популарних библиотека и оквира диструбира преко посебних сервера – мрежа за испоруку саобраћаја (енгл. *Content Delivery Network – CDN*). Нпр. код популарног CSS оквира Bootstrap уколико укључимо фајлове оквира кроз мрежу за испоруку саобраћаја на наш сајт, након иницијалног захтева корисника ка некој страници нашег сајта, следио би унакрсни захтев ка Bootstrap ресурсима.

Полиса о заједничком пореклу (енгл. *Same-origin policy – SOP*) представља битан концепт у модерним Веб-апликацијама. Подразумевано, стандард дозвољава скриптама са једног сајта да преузму податке са другог само ако имају исту вредност у Origin заглављу (протокол, домен, порт). У случају да желимо да учинимо те податке приступачним другим сајтовима, морамо користити посебна CORS (енгл. *Cross-Origin Resource Sharing – CORS*) заглавља. SOP се не односи на све типове захтева, већ се односи на:

- XHR захтеве (енгл. *XMLHttpRequest*) ка унакрсним доменима (захтев ће свакако бити обрађен)
- Укључивање фонтова са унакрсних домена користећи CSS @font-face директиву (нпр. Гугл фонтови)

- Додавање слика/видео-фрејмова на canvas елемент користећи drawImage() функцију и сл.

Пример укључивања CORS заглавља путем mod_headers проширења кроз .htaccess фајл код Апач Веб-сервера:

```
<IfModule mod_headers.so>
 Header Set Access-Control-Allow-Origin 'http://192.168.0.104'
 Header Set Access-Control-Allow-Methods 'GET, POST'
 Header Set Access-Control-Allow-Credentials 'false'
</IfModule>
```

Могуће је користити и џокер (енгл. *wild card*) оператер '*' да би обезбедили приступ захтевима са било ког домена, међутим ово отвара опасност од CSRF напада. Како PUT и DELETE унакрсни захтеви могу да се пошаљу само користећи XMLHttpRequest, подразумевано SOP и CORS штите од таквих напада. Коришћење џокер оператора отвара могућност и од оваквих напада. Сви модерни Веб-прегледачи имају имплементиран CORS механизам.

2.4 Програмирање на страни сервера

Раније су Веб-сајтови били статични, тј. Веб-сервер је требао само да испоручи документ који се налази на фајл-систему самог сервера и одговара траженом ресурсу. Овакав приступ је безбеднији него приступ код динамичких Веб-сајтова, али ограничава могућности самог сајта. Узмимо за пример портал са вестима где је потребно из базе података учитати последњих пар вести или већ поменути пример са аутентификацијом клијента. Динамички Веб-сајтови су сајтови код којих се пре слања одговора клијенту садржај тела одговора генерише користећи програмски језик на страни сервера. Овакав приступ, за разлику од статичног, смањује перформансе и отвара могућност за више безбедносних проблема, али повећава ниво функционалности Веб-сајта.

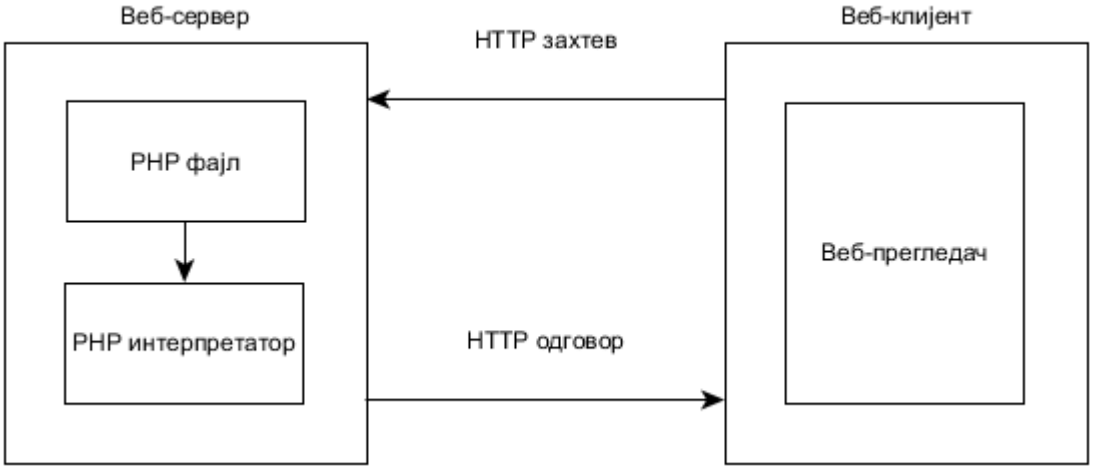
PHP (енгл. *PHP: Hypertext Preprocessor*, рекурзивни акроним, прим. аут.) је програмски језик на страни сервера који се најчешће користи за развој Веб-апликација, иако може бити коришћен и за развој CLI/GUI апликација. Развијен је 1995. године од стране данског програмера Расумса Лердорфа (енгл. *Rasmus Lerdorf*), а данас се развија од стране компаније Zend Technologies. Актуелна верзија је 7.

PHP језик користи Zend Engine као интерпретатор (актуелна верзија 3), који преводи изворни кôд у опкод, што отвара могућност кеширања и подиже перформансе извршавања програма. Алтернатива ZE је Фејсбуков пројекат HipHop Virtual Machine, који користи JIT (енгл. *Just-in-time*) приступ, тј. кôд се компајлира приликом извршавања програма, за разлику од стандардног приступа, где се кôд компајлира пре извршавања. PHP ZE је и даље најзаступљенија имплементација, и углавном примарни избор за многе пројекте.

PHP језик се може користити за израду CLI програма, па чак и за GUI програме коришћењем посебних проширења, међутим, најчешће се користи као језик за

програмирање на страни сервера. У том случају, програмер има три избора:

1. Коришћење PHP језика као модул Apache сервера (mod_php):
   - PHP кôд се извршава од стране Apache сервиса - нема засебних PHP процеса
   - Одличне перформансе на сајтовима који се доста ослањању на PHP (нпр. WordPress сајтови)
2. PHP CGI (енгл. *PHP Common Gateway Interface*):
   - Старији приступ, наведен више из историјских разлога
   - Већа безбедност него код првог приступа, из разлога што се извршавање кода изолује од Веб-сервера
3. PHP FastCGI:
   - Средина између прва два приступа
   - Сваки захтев се прослеђује од стране Веб-сервера ка FastCGI интерфејсу преко сокета
   - Овакав приступ омогућава већу скалабилност - PHP интерпретатор и Веб-сервер могу да се налазе на две различите машине



Слика 2: Програмирање на страни сервера - PHP програмски језик

И поред наведених предности код трећег приступа, и даље је најкоришћенији приступ где се PHP интерпретатор извршава унутар Apache Веб-сервера. Конфигурација PHP језика се дефинише унутар конфигурационог фајла самог Веб-сервера и унутар конфигурационог фајла самог PHP-а (*php.ini* фајл). Сам кôд унутар скрипта се извршава ако се налази између PHP тагова (<?php и ?>) и резултат тог извршавања се исписује у телу одговора (или у заглављу, ако је тако наведено).

Многе популарне CMS (енгл. *Content Management System*) апликације као што су WordPress, Joomla! и phpBB су писане у PHP језику. О стандардизацији језика брине PHP-FIG (енгл. *PHP Framework Interop Group*) група, а Composer пакет-менаџер који функционише на апликативном нивоу омогућава лако додавање библиотека развијених од стране осталих PHP програмера и укључивање истих у сопствени пројекат. Иако се готово сваке године провлачи прича како је популарност

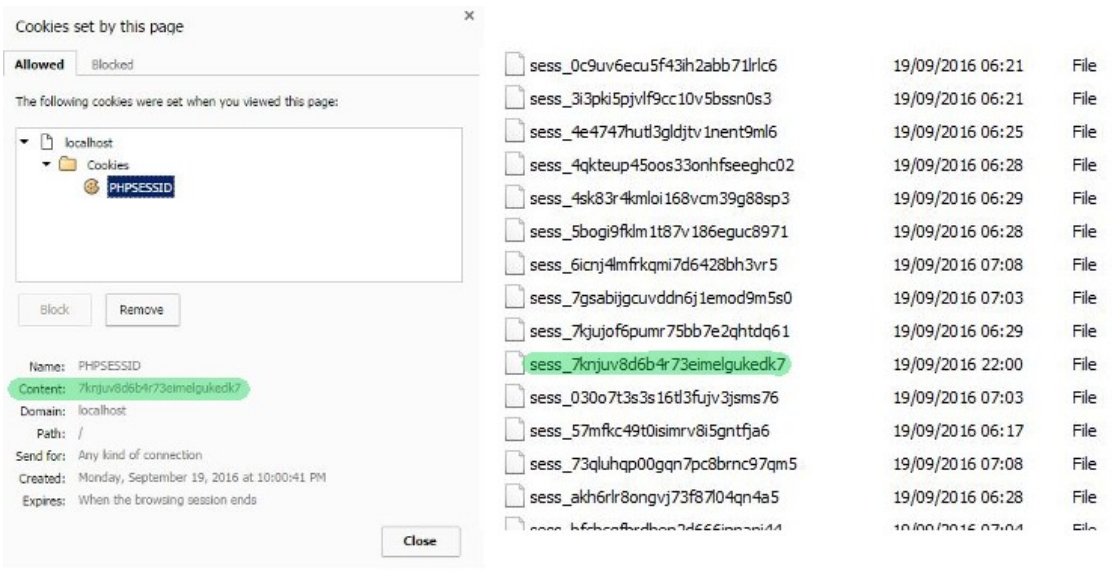
PHP језика у паду, PHP и даље покреће преко 80% сајтова по званичној W3 статистици.

2.4.1 Механизам сесије код PHP програмског језика

Код HTTP колачића подаци се чувају директно на клијенту. Ово није претерано безбедно решење из два разлога:

1. Подаци су доступни клијенту
2. Колачићи су подложни манипулацији од стране клијента

Механизам сесије код PHP језика чува само сесијски идентификатор (псеудо-случајан број) као колачић на страни клијента. Тај идентификатор одговара фајлу на серверу у коме се чувају подаци везани за тренутну сесију клијента. То значи да клијент неће имати приступ подацима који су осетљиви, већ псеудо-случајном броју који му у принципу не значи ништа. Проблем преотимања сесије (енгл. *session-hijacking*) је ипак и даље присутан.



Слика 3: Вредност сесијског колачића код PHP језика одговара запису на фајл-систему у коме се чувају промењиве везане за тренутну сесију

3 Напади на Вебу

Под нападима на Вебу подразумевамо нападе на седмом слоји OSI модела рачунарске комуникације који користе разне пропусте на страни сервера, апликације или у самом протоколу за приступ поверљивим подацима.

Нападе можемо класификовати по сврси напада:

1. Напади на поверљивост (енгл. *Confidentiality*)
2. Напади на веродостојност (енгл. *Integrity*)
3. Напади на доступност (енгл. *Availability*)

Напади на поверљивост су напади који имају за циљ да неауторизованој (неовлашћеној) особи дају приступ неким подацима.

Напади на веродостојност су напади који имају за циљ да неауторизованој особи омогуће да се представи као легитимна страна у комуникацији и при том мења садржај саобраћаја.

Напади на доступност су напади који имају за циљ да онемогуће приступ подацима ауторизованој особи.

Напади не би били могући без пропуста, а њих по правилу увек има. Пошто Веб користи клијент-сервер модел комуникације, тј. са једне стране је клијент а са друге сервер, а подаци се преносе преко протокола, пропусте можемо класификовати такође у три групе:

1. Пропусти на страни Веб-сервера
2. Пропусти у Веб-апликацијама
3. Пропусти у самом протоколу

Треба имати у виду да се неки пропусти не могу сврстати у само једну од ових група, као и да напади углавном користе више пропуста. Такође, не треба заборавити ни проблем шпијунирања где нападач може долазити до поверљивих информација због непажње самог корисника и који свакако представља један од већих проблема на Вебу данас, али то није тема овог рада.

3.1 Уметање SQL кода

Овај напад је заузео прво место на OWASP (енгл. *Open Web Application Security Project)* Top 10 листи из 2013. године и свакако је најпознатији пропуст на Вебу. Ради се о пропусту код кога нападач може да унесе малициозан код који ће се интерпретирати на страни сервера. Последице могу бити катастрофалне (уклањање табела из базе података, заобилажење аутентификације...), али заштита је прилично једноставна - довољно је филтрирати податке пре њиховог записивања у базу. У листингу испод је дат пример где је програмер урадио филтрирање за податке прослеђене из уноса, али не и податке које је покупио из заглавља протокола.

```
<?php

$firstName   = filter_input(INPUT_POST, 'f1_fn');
$lastName    = filter_input(INPUT_POST, 'f1_ln');
...
$userAgent   = $_SERVER['HTTP_USER_AGENT'];
...
// Извршавање INSERT упита у табелу са преузетим параметрима
```

У овом примеру довољно би било да нападач користећи cURL или Tamper Data нпр. измени вредност User-Agent поља у HTTP заглављу у ');DROP TABLE `users`;--. Резултат извршавања овог кода би био уклањање одговарајуће табеле из базе података.

3.2 Унакрсно скриптовање

Када се појавио током '90-их година напад је носио име CSS - Cross-Site Scripting. Међутим, пошто се тај термин везује за стилове код HTML докумената и апликација (енгл. *Cascading Style Sheets - CSS*), скраћеница XSS је почела да се везује за овај пропуст. Пропуст се огледа у томе што се кориснички унос исписује клијенту у телу одговора као валидан кôд, без обраде у циљу избегавања неодговарајућих вредности. Разликујемо две врсте XSS пропуста, ускладиштени и рефлектовани XSS.

Један од примера напада који користе овај тип пропуста је FatCookies напад који се ослања на диспропорцију између максималне величине колачића на страни клијента и максималне величине колачића којих ће сервер примити као валидан захтев. Резултат овог напада је напад на доступност сајта код сваке следеће посете жртве.

Такође, раније је био популаран и напад крађе сесијског колачића злоупотребом JavaScript језика. Овај напад је решен увођењем HttpOnly колачића, тј. колачића који су доступни само у HTTP заглављима, не и преко JavaScript језика. Испоставило се да то није било довољно јер су нападачи могли да симулирају TRACE захтев користећи AJAX (енгл. *Asynchronous JavaScript and XML*) захтев након што пронађу XSS пропуст, па је и ово решавано тако што сервер подразумевано не одговара на TRACE захтеве који су упућени преко JavaScript језика. Овај случај иде у прилог тези да очигледна заштита није беспрекорна, јер се може десити да се отвори други пропуст крпљењем првог.

Најпознатији XSS пропуст је свакако чувени Samy црв (енгл. *worm*) који је на друштвеној мрежи MySpace у року мањем од 20 сати имао преко милион жртава, што је за резултат имало привремено обарање сајта. Аутор црва је Семи Камкар (енгл. *Samy Kamkar*), амерички хакер, који је за прекршај осуђен на три године пробације уз могућност коришћења само једног рачунара без Интернет конекције и 20.000 долара новчане казне.

3.2.1 Ускладиштени XSS

Размотримо следећу ситуацију: нападач поставља коментар који ће бити унет у базу података и репродукован свим осталим посетиоцима те странице без ауторизације од стране модератора. Нападач у текстуалном пољу унутар формулара као коментар шаље следећу вредност:

```
<script>alert('!');</script>
```

У случају да је у програмској логици ова вредност исписана без филтрирања на HTML ентитете, XSS би био могућ.

Ово би сваком следећем посетиоцу који сајт посећује са прегледача на коме је подржан JavaScript избацило информативни прозор са поруком нападача. Сама чињеница да нападач може да изврши било какав JavaScript кôд који ће се рендеровати унутар туђег прегледача нарушава поверење клијента у Веб-сајт. Решење је у филтрирању динамичких вредности пре њиховог исписивања у тело одговора, нпр. код PHP језика функција htmlentities().

Многи PHP шаблонизатори (енгл. *template engine*) као што су Smarty, Twig, Blade итд. нуде могућност аутоматског филтрирања садржаја у шаблон. У апликацијама које не користе шаблонизаторе заштита се врши коришћењем одговарајуће функције за уклањање специјалних карактера, као у примеру изнад.

3.2.2 Рефлектовани XSS

Рецимо да имамо Веб-сајт са стандардним формуларом за претрагу. Након претраге, резултати се исписују заједно са вредношћу кључне речи која је унета у формулар. У случају да нападач унесе исто што и у претходном примеру, а да улаз није филтриран на недозвољене карактере, XSS ће опет бити могућ. Међутим, ово не спада у ускладиштени XSS јер је улаз узет из GET или POST променљиве, а не из базе података. Овакви пропусти се називају рефлектовани XSS, и најчешће се користе уз нападе као што су пецање (енгл. phishing). Пример: слање линка жртви са малициозним кодом у виду GET параметра.

3.2.3 DOM-базирани XSS

Постоји и трећа група ових пропуста, а базира се на коришћењу небезбедних JavaScript функција. Примери ових функција су eval() као и атрибут innerHTML. Код оваквих пропуста решење је унутар саме програмске логике, нпр. коришћење textContent атрибута за испис одговора уместо innerHTML. Ово свакако ограничава могућности програмера и DOM-базирани XSS није напад код кога је заштита једноставна.

3.2.4 Полиса о заштити садржаја

Механизам полисе о заштити садржаја (енгл. *Content Security Policy – CSP*) је новији механизам развијен у циљу спречавања пропуста типа XSS. Механизам је имплементиран на свим модерним Веб-прегледачима.

Функционише тако што укључимо CSP заглавље које ограничава изворе са којих укључујемо садржај. Размотримо следећи садржај .htaccess фајла:

```
Header Set Content-Security-Policy "default-src 'self'; script-src
 'self' 'unsafe-inline'; img-src www.*.example.com;"
```

Ово заглавље у HTTP одговору означава следеће ствари:

- default-src 'self' – подразумевано, могуће је учитавати садржај (слике, скриптове и сл.) само са домена сајта (искључујући под-домене)
- script-src 'self' 'unsafe-inline' – допуштено је укључивати тзв. inline скриптове (директно унутар script тага, onload и сл.)
- img-src www.*.example.com – допуштено је учитавање слика са example.com домена (укључујући и под-домене)

Овај механизам не решава проблем XSS пропуста, само смањује ризик од истих.

3.3 Фалсификовање унакрсних захтева

Фалсификовање унакрсних захтева (енгл. *Cross-Site Request Forgery*, такође познат и као *One-Click Attack* или *Sea-Surf*) је један од најопаснијих пропуста на Вебу. За разлику од унакрсног скриптовања, пропуста који експлоатише поверење корисника у одређени Веб-сајт, фалсификовање унакрсних захтева је пропуст који експлоатише поверење које Веб-сајт има у корисников Веб-прегледач. Прецизније речено, ови напади се ослањају на немогућност Веб-сервера да одреди које је захтеве упутио корисник директно.

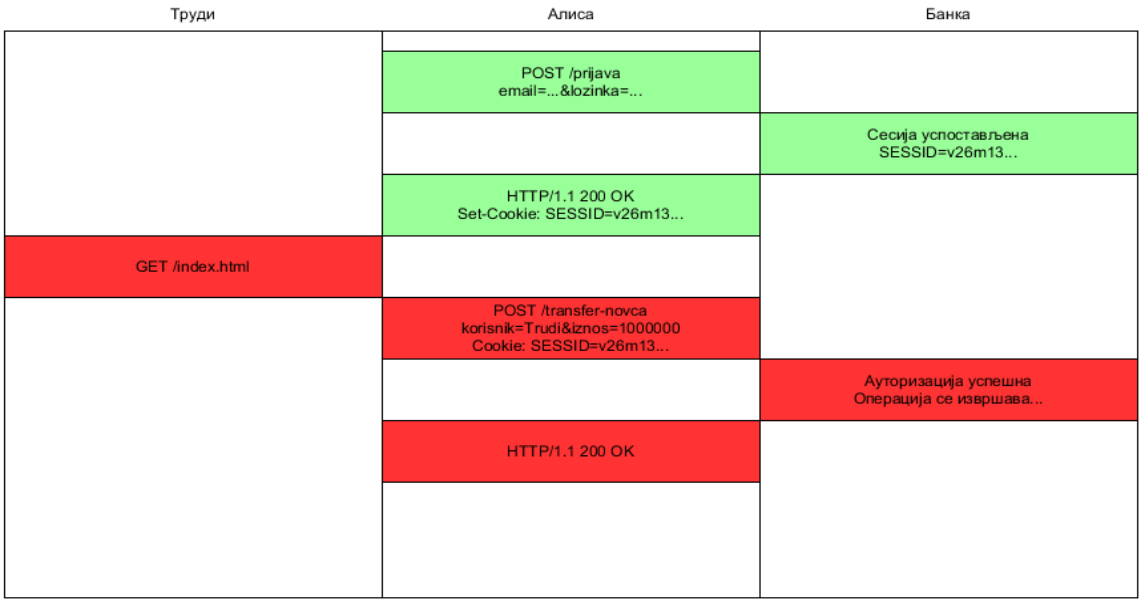
Код напада фалсификовањем унакрсних захтева, нападач експлоатише корисника који је ауторизован у Веб-апликацији - напад нема сврхе ако корисник није пријављен, јер ће фалсификован захтев добити одговор у виду редирекције најчешће. Нападач приморава (директно или индиректно) корисника да упути захтев ка серверу који ће искористити CSRF пропуст и омогућити нападачу да заобиђе контролу приступа јер ће сваки захтев упућен ка серверу долазити са адресе Интернет протокола која одговора ауторизованом кориснику.

Узмимо за пример да је корисник Алиса пријављен на Веб-апликацију банке. Нападач Труди жели да пребаци 1.000.000 динара са Алисиног рачуна. Пошто је Алиса пријављена на апликацију, то значи да она има привилегије да пребаци поменути износ са свог рачуна на рачун Труди. Труди жели да експлоатише поверење

које Веб-апликација има у Алисин Веб-прегледач. Труди на привременој Веб-локацији поставља документ који ће садржати само малициозан JavaScript кôд следећег садржаја у себи:

```
    <script>
     var xhr = new XMLHttpRequest();
     xhr.open("POST", "https://banka.rs/transfer-novca", true);
     xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
     xhr.withCredentials = true;
     xhr.send("korisnik=Trudi&iznos=1000000");
    </script>
```

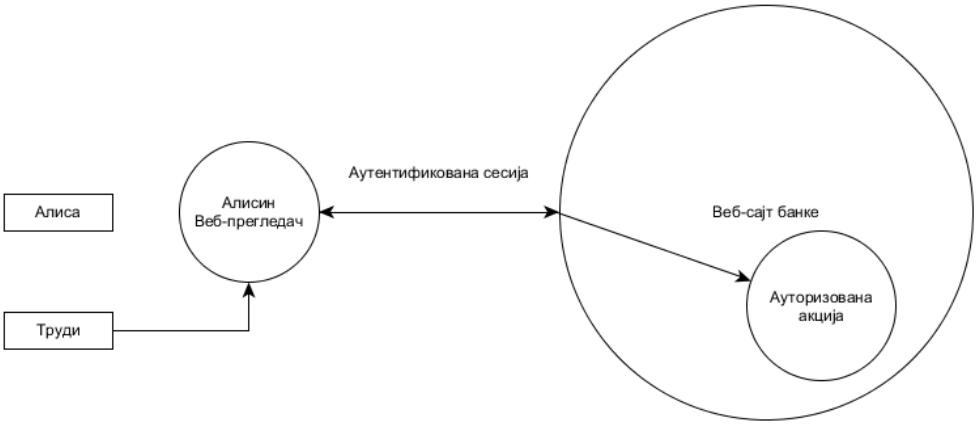
Након тога, све што је потребно да Труди уради је да примора Алису да са свог прегледача у којем је већ пријављена на апликацију банке посети задату локацију и са Алисиног рачунара биће упућен валидан захтев ка банци за трансфером новца. Ако Труди жели да изгледа мање сумњиво, уредиће страницу да изгледа као валидан садржај који жели да подели са Алисом, нпр. неки вирални видео или занимљив текст. Такође, у случају да жели да маскира локатор Труди може користити неку од техника за скраћивање униформног локатора ресурса (нпр. *Google URL Shortener*) у случају да сам локатор изгледа сумњиво. Наиван приступ би био да верујемо да Алисина безбедност зависи само од тога да ли ће отворити линк од Труди или не. У претходном делу описан је XSS пропуст, који може омогућити Труди да малициозан кôд унесе на неком другом сајту, па чак и на сајту банке (овакав тип CSRF пропуста се назива ускладиштени CSRF). На слици испод примећујемо да Труди мора једино да натера Алису да посети Веб-локацију на којој се малициозни кôд налази. Због тога се ови напади још називају и One-click attack. CSRF је могућ увек када се Веб-апликација ослања само на колачиће за ауторизацију.



Слика 4: CSRF напад - временски дијаграм

CSRF спада у тзв. „збуњени изасланик“ (енгл. *confused deputy*) нападе.

Изасланик у претходном примеру је Алисин Веб-прегледач, који верује да је Трудин фалсификовани захтев уствари послала Алиса. Тиме се експлоатише поверење које корисник има у сопствени Веб-прегледач. Сама чињеница да се већина модерних Веб-сервиса данас ослања на унакрсне захтеве, говори да је тешко имплементирати механизам у самом прегледачу који треба да разликује легитиман захтев од фалсификованог. Напади типа „збуњени изасланик“ нису ограничени само на рачунарске мреже, већ се примери могу наћи и у физичком свету – нпр. свакодневни пример је одлазак у супер-маркет по набавку намирница. На каси, касир треба да скенира бар-кодове свих намирница и тако одреди цену. Нападач би могао да замени бар-кодове скупљих намирница са бар-кодовима јефтинијих намирница. У овом случају, „збуњени изасланик“ би био касир, јер користи наизглед валидне бар-кодове да би одредио укупну цену.



Слика 5: CSRF напад - Алисин Веб-прегледач игра улогу „збуњеног изасланика“

Сврха напада не мора бити трансфер новца, резултати успешног CSRF напада могу бити измена корисничког имена, адресе електронске поште корисника, лозинке итд., а у случају да је ауторизовани корисник администратор неког CMS система, успешан CSRF напад може да компромитује целу апликацију.

3.3.1 CSRF напад и HTTP методе

Експолатација CSRF пропуста код GET методе је тривијална, пошто параметри могу да се нађу унутар самог линка који може бити прослеђен жртви, или се може упитити захтев коришћењем <a> или <img> тагова. Међутим, HTTP подразумевано дефинише GET као методу која треба да се користи само за читање података, па и већина данашњих оквира не долази са CSRF заштитом код GET формулара. Веб-апликација свакако треба да користи само POST методе за операције додавања/измене/уклањања података. Такође, већина оквира и библиотека данас долази без заштите за CSRF нападе код GET метода.

Експлоатација CSRF пропуста код POST метода је мало тежа, али много учесталија у већини ових пропуста, баш због чињенице да се POST користи за све операције које мењају стање података. Нпр. уколико нападач нађе XSS пропуст на нашем сајту може га искористити за складиштење малициозног JavaScript кода приказаног у листингу испод. У овом случају (за разлику од претходног примера где

је коришћен XMLHttpRequest захтев) ни CORS механизам не може да заштити потенцијални напад. Такође, коришћење било какве заштите би било недовољно – у случају да XSS пропуст постоји, вредност токена се може ишчитати такође преко JavaScript језика и проследити у фалсификованом формулару.

```
<body onload="document.forms[0].submit()">
 ...
 <form action="https://banka.rs/transfer-novca" method="POST">
       <input type="hidden" name="korisnik" value="Trudi">
       <input type="hidden" name="iznos" value="1000000">
 </form>
 ...
</body>
```

Методе PUT и DELETE су дискутоване у овом контектсу раније.

4 Заштита код фалсификовања унакрсних захтева

За разлику од претходна два наведена напада, овај напад је специфичан јер заштита за њега није толико једноставна. Постоји више механизама заштите али ниједан од њих није елегантан или потпуно поуздан колико је то заштита код претходна два напада. У наредним поглављима размотрићемо неке од метода заштите, као и имплементацију заштите од оваквих напада развијену од самог аутора за потребе апликативног оквира који је развијан на практикуму Интернет и Веб-технологије.

4.1 Коришћење токена за ауторизацију

Код напада фалсификовањем унакрсних захтева постоји више механизама заштите. Најсигурнији метод користи једнократне токене као потврду да је корисник дошао са локације на којој се формулар и налази. Мана оваквог приступа је на првом месту то што ће за сваку посету корисника морати да се генерише нови токен – ово повећава време обраде захтева, као и значајно веће заузеће меморије. Такође, корисник у том случају нема нормалну навигацију у Веб-апликацији (онемогућено коришћење back дугмета). Уз то треба имати у виду да PHP није event-driven језик, тј. сервер треба само да испоручи фајл, не можемо детектовати када је корисник угасио таб са формуларом и уклонити тај токен из меморије пошто је неискоришћен. Дакле, имамо случај у коме генеришемо нови токен по захтеву и случај у коме генеришемо нови токен по сесији. Размотримо предности и мане оба приступа.

Генерисање новог токена по захтеву:

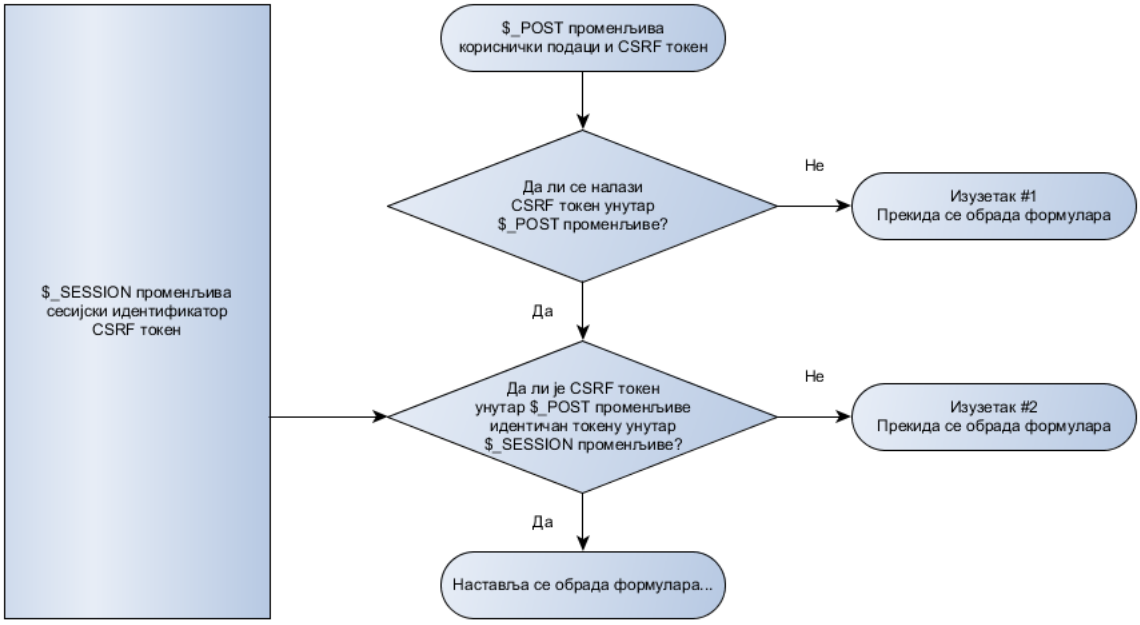
- Минорно већа безбедност
- Веће заузеће меморије – потенцијални DoS/DDoS напад
- PHP генерално није језик који је погодан за овај приступ

Генерисање новог токена по сесији:

- Поуздано решење, и ако је безбедност мања него код првог приступа
  - У случају да постоји XSS пропуст било где на сајту, нападач може да украде токен пошто је токен глобалан за све странице, а не идивидуалан
  - XSS није потребан да би CSRF напад успео, али било какав XSS пропуст може да послужи за крађу токена
  - У случају снимања мрежног саобраћаја токен се такође може лакше украсти
  - Решење: коришћење HTTPS протокола
- Мање заузеће меморије
- Токен се мора ресетовати при промени привилегија у апликацији. нпр. када се корисник пријави

Заштита коришћењем случајног токена базира се на чињеници да, да би CSRF напад био успешан, нападач мора да предвиди вредности за сва поља. Пошто је токен псеудо-случајан дугачак низ алфа-нумеричких карактера, нападач мора да украде токен пре тога или има доста среће у погађању. Ово значајно отежава CSRF напад.

Овај приступ је најпопуларнији модел за заштиту и коришћен је од многих разних оквира и библиотека који штите од оваквих напада: Zend Framework, Laravel, OWASP CSRFProtector итд. Већина тих решења се разликује по месту складиштења токена, нпр. Laravel подразумевано користи сопствени механизам сесије док OWASP решење токен чува унутар колачића.



Слика 6: Дијаграм тока код заштите помоћу токена

4.1.1 Анализа заштите код Laravel апликативног оквира

Laravel је најпознатији и најчешће коришћени PHP апликативни оквир развијен од стране америчког програмера Тејлора Отвела (енгл. *Taylor Otwell*) и намењен је за развој MVC (енгл. *Model-View-Controller*) Веб-апликација. Користи модуларан пакет-менаџер који омогућава лако проширивање функционалности оквира додавањем новог посредног софтвера (енгл. *middleware*) у оквир. Актуелна верзија је 5.3.

Laravel користи посредан софтвер за заштиту од CSRF напада. Заштита користи приступ генерисања новог токена по сесији. Вредност токена је потребно укључити у сваки формулар у виду скривеног поља (input type="hidden", прим. аут.), што лако можемо урадити уз помоћне (енгл. *helper*) функције које омогућавају фасадан приступ раду са класом за заштиту. Подразумевано, Laravel не допушта обраду формулара који не укључују CSRF токен. То се може искључити мануално, али није препоручљиво. Пример укључивања поља у формулар преко Blade шаблонизатора је дат у листингу испод.

```
    <form action="{{ URL::route('obrada') }}" method="POST">
      <label for="f1_email">Email:</label>
      <input type="email" id="f1_email" name="f1_email" required>
      <label for="f1_password">Lozinka:</label>
      <input type="password" id="f1_password" name="f1_password"
required>
      {{ csrf_field() }}
      <button type="submit">Slanje</button>
    </form>
```

Након слања формулара, заједно са CSRF токеном и осталим пољима, прво се ради поређење између прослеђене вредности и вредности сачуване у тренутној сесији корисника. За то је задужена класа VerifyCsrfToken. У случају да токен није прослеђен, јавља се TokenMismatchException изузетак. У случају да је токен погрешан, јавља се TokenMismatchException изузетак. Затим се унутар функције tokensMatch() дешава главни део: робусно поређење токена који се налази у сесији и токена који је прослеђен као део формулара. Једино у случају када се те две вредности подударају, обрада формулара се наставља.

CSRF заштита је подразумевано укључена у Laravel оквиру. Додатно се може искључивати појединачно по рутама или глобално за све. Токен се такође чува и у XSRF-Token колачићу, због компатибилности са JavaScript оквиром AngularJS (из тог разлога колачић није HttpOnly). Тада се токен аутоматски додаје у виду мета-тага у заглављу документа. Иако је вредност колачића шифрована, има места дискусији да ли се овако нарушава безбедност овог приступа.

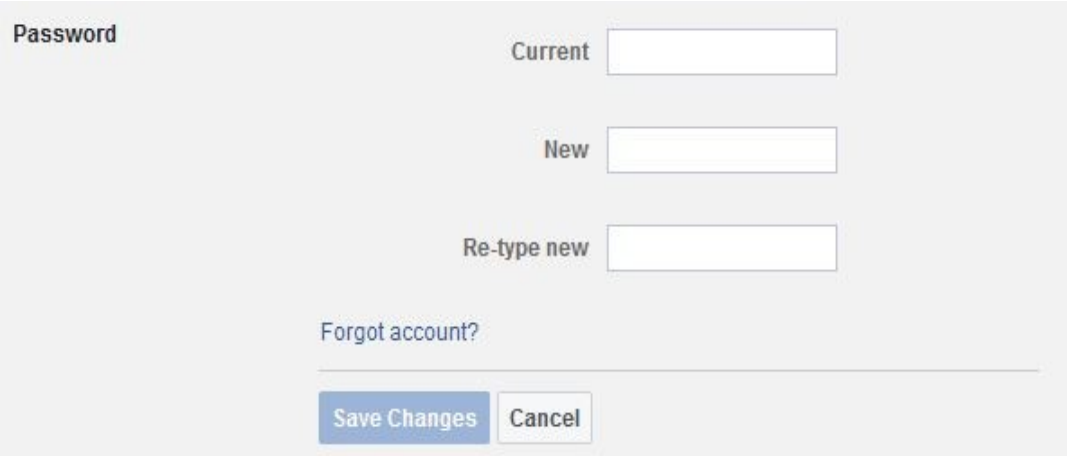
Сам токен је дужине 40 карактера и користи PHP функцију random_bytes() која је намењена за генерисање криптолошки безбедних псеудо-случајних бројева. Пошто је ова функција уведена тек од верзије 7, а многи пројекти користе старије верзије PHP језика (верзија 5 је и даље најзаступљенија), Laravel користи библиотеку за компатибилност уназад (енгл. *backwards compatibility*). Такође, ова функција се и препоручује за генерисање случајних низова који имају област примене у безбедности, и поред тога што PHP поседује више функција које се могу користити за исту сврху. Случајна секвенца је генерисана на следећи начин:

- Код MS Windows оперативних система користећи WinAPI и функцију CryptGenRandom()
- Код Линукс оперативних система системским позивом функцији getrandom()
- Код старијих Линукс дистрибуција биће коришћен /dev/urandom фајл
- У случају да ниједан од наведених извора није доступан, јавиће се PHP изузетак

Све три наведене опције генеришу криптолошки сигурне псеудо-случајне низове/бројеве. Ипак, 2007. године група студената са Универзитета у Јерусалиму је пронашла значајне пропусте у имплементацији алгоритма на старијим Windows системима. Пропуст је исправљен касније, али то није први пут да у Мајкрософту праве овакав тип пропуста (погледати PHP функцију rand()).

4.2 Коришћење CAPTCHA механизма и ре-аутентификација

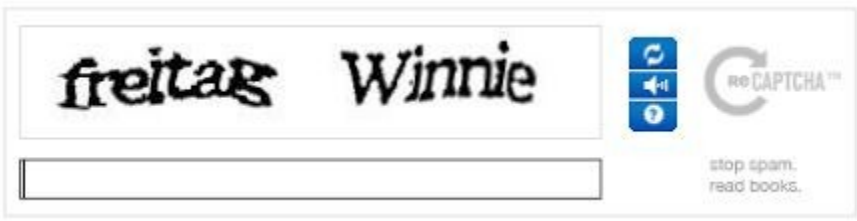
Постоје методе заштите од CSRF напада који укључују интеракцију са корисником током трансакције. Иако се овакав начин заштите имплементиран на правилан начин сматра за јако безбедну одбрану од CSRF напада, није применљив увек из разлога што лоше утиче на корисничко искуство (енгл. *User Experience, UIX*). Најпопуларнији овакви видови заштите су ре-аутентификација и CAPTCHA кодови.



Слика 7: Фејсбук користи метод ре-аутентификације за заштиту од CSRF напада при измени лозинке

CAPTCHA (енгл. *Completely Automated Public Turing test to tell Computers and Humans Apart*) спада у групу изазов-одговор (енгл. *challange-response*) тестова који се користе у рачунарству примарно у циљу препознавања да ли акцију изводи човек или бот. Додатно, могу се користити у заштити од CSRF напада, и у том случају је имплементација слична као и код приступа генерисања случајног токена, уз разлику што се захтева и одговарајућа интеракција од корисника за разлику од претходног случаја где је токен био прослеђен унутар скривеног поља.

Постоје различити типови CAPTCHA тестова, најпопуларнији тренутно је Гуглов reCAPTCHA систем који се може лако имплементирати у сопствене пројекте. Овај софтвер се ослања на чињеницу да је OCR (енгл. *Optical Character Recognition*) софтверу тешко да „прочита“ алфа-нумерички садржај са слике у разумном временском периоду, док људи поседују већ развијене функције за то и ту акцију могу да обаве у веома кратком временском периоду. Овај приступ се може и користити за заштиту од CSRF напада, чак и безбеднији од генерисања токена по сесији јер ће за сваки захтев бити дат другачији садржај. Актуелне су две верзије reCAPTCHA софтвера, а међусобно се највише разликују у корисничком интерфејсу.



Слика 8: Прва верзија reCAPTCHA софтвера

Постоје различити типови CAPTCHA система: текстуални, звучни, логички...

Главни проблеми код оваквих напада су квалитетни OCR алгоритми, као и тимови људи из земаља са ниским стандардом – на већини фриленс (енгл. *freelance*) сајтова могу се наћи огласи за посао који траже да се у одређено време на рачунару решавају разни CAPTCHA тестови за мале паре. Особе које раде овакве послове на Интернету се зову микро-радници (енгл. *microworkers*).

Наравно, није обавезно користити Гуглов сервис за увођење CAPTCHA тестова на сопственом сајту, PHP има јако добре библиотеке за рад са графичким садржајем (GD библиотека подразумевано инсталирана у већини дистрибуција), па је лако могуће генерисати слику мањег формата која ће бити укључена у формулар као CAPTCHA тест. Вредност за CAPTCHA изазов може бити сачувана унутар сесијске променљиве, и може бити генерисана по сесији или по захтеву.

Напомена: приступ генерисања новог токена/CAPTCHA кода по захтеву доноси већу безбедност, али отвара могућност за DoS нападе. Сам PHP нема ограничење када је у питању број фајлова који ће се наћи у session.save_path директоријуму, али већина фајл-система (NTFS, ext4...) пружа лошије перформансе када поседује огроман број записа. Решење је у опционом аргументу који се додељује као вредност session.save_path променљивој. У питању је целобројна вредност која означава дубину директоријума где ће сесијски фајл бити сачуван. Нпр. ако је вредност session.save_path променљиве '5;/tmp', то значи да сесијски фајл може бити сачуван на локацији /tmp/4/b/1/e/3/sess_4b1e384ad74619bd212e236e52a5a174If. Стабло директоријума морамо пре тога да креирамо све под-директоријуме (постоје скрипте које аутоматизују ово). Такође, овај број не треба бити већи од 2 (64^2 под-директоријума за вредност '2;/tmp').

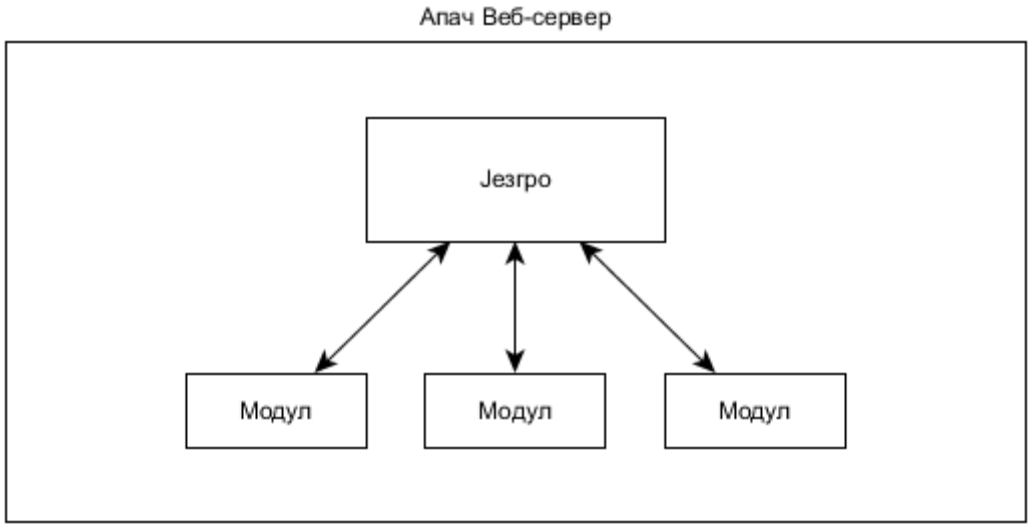
Генерално, и CAPTCHA и ре-аутентификација пружају добру безбедност од CSRF напада, под условом да су правилно имплементирани. Нападач мора да зна лозинку (у случају да је зна, тешко да ће му вектор напада бити CSRF) или да погоди случајно генерисану CAPTCHA вредност генерисану за одређеног корисника. Ипак, неозбиљно је користити овакав вид заштите код свих формулара, тако да ово не представља директну замену претходном приступу, већ само потенцијалну опцију.

4.3 Заштита на страни сервера

Још једно решење које користи токен за додатну ауторизацију је OWASP CSRFProtector пројекат. Оно што је занимљиво је што они не нуде ово решење само као PHP библиотеку, већ и као додатни модул за Апач Веб-сервер.

Основна функционалност Апач Веб-сервера је процесирање HTTP захтева и одговора, као и подршка за вишенитно и вишепроцесно обрађивање. Модуларна архитектура Апач Веб-сервера омогућава лако проширивање основних функционалности сервера, додавањем нових проширења, тзв. модова (енгл. *module, mod*). Тако нпр. користимо mod_rewrite када желимо да имамо семантичке локаторе ресурса на Веб-сајту, mod_status је популаран у понуди код хостинг-провајдера јер пружа информације о активности и перформансама сервера, mod_headers и mod_php су већ поменути итд. Исто тако могуће је писати своје модуле, компајлирати их и укључивати у Апач по потреби (C програмски језик). Тако је развијен и OWASP CSRFProtector мод који је намењен за заштиту од CSRF напада на самом серверу,

што скида одговорност са самог програмера. Мод се назива mod_csrfprocetor и доступан је на GitHub репозиторијуму.



Слика 9: Модуларна архитектура Апач Веб-сервера омогућава лако проширивање основних функционалности сервера

Након укључивања модула кроз Апач конфигурациони фајл, потребно је конфигурисати опције мануално. Омогућена је и ауторизација за GET захтеве (иако је напоменуто да се они свакако не би требали користити за било шта осим за операције читања), избор дужине токена и сл.

4.4 Заштита на страни корисника

У оквиру овог рада наведене су и неке ствари које сами корисници могу да ураде да би се заштитили од оваквих напада. Ствари које су наведене у овом делу нису гаранција за апсолутну заштиту од оваквих напада, али у многоме отежавају нападачу посао.

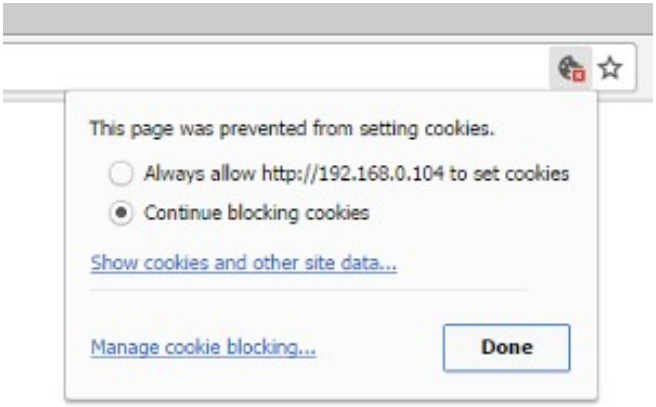
4.4.1 Заштита унутар Веб-прегледача

Можемо закључити да је CSRF напад могућ кад год Веб-сајт може да натера корисника да изврши захтев ка другом Веб-сајту на којем је пријављен као легитиман корисник. Напад међутим не би био могућ када би се сваки захтев који стиже са другог сајта водио као засебан захтев, тј. када не би могао да у оквиру заглавља захтева пошаље легитимну вредност сесијског колачића. Ово је имплементирано већ на нивоу самих прегледача на страни клијента, а ради се и на имплементацији у виду нове опције при постављању колачића на страни сервера. Ово је добар вид заштите јер ће сајту нападача (осим уколико није коришћен XSS пропуст) изгледати као да ви нисте пријављени на Веб-апликацију у којој жели да заобиђе процес ауторизације кроз CSRF пропуст.

Код Google Chrome прегледача опција коју треба означити је:

```
    Settings > Show advanced settings... > Privacy / Content Settings >
Block third-party cookies and site data
```

Након тога појавиће се обавештење о томе да Веб-сајт покушава да користи колачиће неког другог сајта, уз подразумевану опцију да му то не допусти. У случају да игноришемо то обавештење, бићемо сигурни од неускладиштених CSRF напада.



Слика 10: Google Chrome прегледач нуди опцију онемогућавања колачића "треће стране"

Такође постоји и опција Manage exceptions... која нам омогућава да додамо листу сајтова на којој допуштамо колачиће треће стране. Занимљиво, сајтови који долазе са Гугловог домена су подразумевано допуштени. Треба имати у виду да је је и сам Гугл имао проблема у прошлости са CSRF нападима (погледати YouTube CSRF пропуст), тако да је безбедније уклонити и тај изузетак.

Иста опција постоји и код другог најпопуларнијег Веб-прегледача тренутно, Mozilla Firefox. Потребно је изабрати опцију Never на путањи:

```
    Options > Privacy > History / Accept third-party cookies
```

Овде такође постоји листа са изузецима, али подразумевано се ту не налази ни један домен.

Овај приступ неће штити од ускладиштених CSRF напада (XSS пропуст на самом сајту), као ни од напада где сајт на којем смо аутентификовани захтева нешто од малициозне Веб-локације. Исто тако треба водити рачуна о томе да неки случајеви тренутно захтевају омогућавање колачића треће стране (нпр. приступ приватним пројектима код GitHub Веб-репозиторијума).

Ипак, треба водити рачуна о томе да неће сви корисници издвојити време да прочитају о томе како да се заштите на Вебу. Већина корисника користи подразумеване опције и ретко када мења нешто. Један научни рад са Универзитета Беркли у Калифорнији, симболично назван „Зашто Џони не може да шифрује?“

(енгл. *Why Johnny Can't Encrypt?*) се бави управо овом темом, тј. колики је утицај самих корисника на нарушавање безбедносних механизама апликација. Корисна функционалност Веб-прегледача је тзв. инкогнито мод, тј. могућност да сурфујемо Вебом без чувања историје претраживања, кеширања садржаја и чувања колачића. Ову функционалност би требало користити сваки пут када се пријављујемо на неку Веб-апликацију са рачунара који није наш лични рачунар.

4.5 Увођење новог атрибута за колачиће

Тренутно IETF (енгл. *Internet Engineering Task Force*) ради на стандардизацији новог атрибута за колачиће. У питању је SameSite атрибут, који може да има две вредности:

- Lax (ср. лабаво)
- Strict (ср. строго, стриктно)

Као и што називи вредности кажу, код првог је коришћење колачића треће стране допуштено у неким случајевима (нпр. обичан GET захтев), док је код стриктног мода колачић недоступан другим сајтовима. Коришћење лабавог мода је корисно у случајевима као од мало пре, када Гугл користи нпр. колачић за локализацију па жели да га има доступног и на свим под-доменима. Стандард се тренутно води као нацрт (енгл. *draft*), мада је имплементиран у Chrome прегледачу од верзије 51 (актуелна верзија 53), као и у новијим верзијама Opera прегледача.

На табели испод дат је пример када ће се колачићи од треће стране (енгл. *third-party cookies*) шаљу у случају да су у питању обични колачићи, а када у случају да су дефинисани са SameSite=lax атрибутом.

Тип захтева	**Пример кода**	**Колачићи који се шаљу**
линк	<a href="...">	обични, лабави
пре-рендеровање	<link type="prerender" href="...">	обични, лабави
GET формулар	<form method="GET" action="...">	обични, лабави
POST формулар	<form method="POST" action="...">	обични
фрејм	<iframe src="...">	обични
AJAX	$.get('...')	обични
слика	<img src="..." alt="...">	обични

5 Предлог сопственог решења за заштиту

За потребе предмета „Практикум – Интернет и Веб-технологије“ развијен је едукативни MVC оквир у PHP језику, који се, уз мало дораде, може користити и за личне пројекте. Главне функционалности које су додате у оквир од стране аутора овог рада су:

- Прилагођавање оквира тако да подржава PHP-FIG PSR (енгл. *PHP Standards Recommendations*), са посебним освртом на PSR-4 стандард који омогућава лакше укључивање других библиотека користећи Composer пакет-менаџер
- Имплементација заштите од CSRF напада: класа за манипулацију токенима

Што се раније имплементираних функционалности тиче, оквир не заостаје ништа за модерним микро-оквирима као што је нпр. Slim Framework. Наравно, циљ развоја оквира није био да се пореди са full-stack решењима као што су Laravel или Zend Framework.

5.1 Класа за манипулацију токенима

У оквиру прошлог поглавља понуђена су два приступа код заштите путем токена – генерисање новог токена по сесији и генерисање новог токена по захтеву. Пошто је безбедносна бенефит код другог занемарљива у односу на перформансе и потенцијалне проблеме које носи са собом, аутор овог рада се определио за прву варијанту. Не треба заборавити да се ово решење да би било потпуно безбедно ослања на две ствари: да се користи шифрована верзија HTTP протокола и да не постоји XSS пропуст у апликацији.

Пошто треба штитити само формуларе који захтевају да је корисник аутентификован, то значи да ће бити коришћен механизам сесије. Из тог разлога, одлучено је да се унутар сесијске променљиве чува и вредност токена. Такође, пошто се токен генерише само једном (на почетку сесије), користимо особине Singleton софтверског дизајн-патерна, који ограничава Веб-апликацију на само једну инстанцу те класе. То се постиже манипулацијом видљивости метода саме класе – метод __construct() који представља метод конструктор код PHP језика треба да буде приватан, како би се спречили позиви ван саме класе. Уместо њега, уводимо јавне статичне методе који се користе за извршавање операција саме класе. У нашем случају, не желимо да имамо ниједну инстанцу класе, већ само статичне методе кза рад са вредношћу сачуваном у сесијској променљиви. Осим конструктора, да би Singleton био потпуно правилан унутар PHP језика, пожељно је и учинити приватним следеће методе:

- __clone() - спречава појаву дупликата класе коришћењем оператора clone
- __wakeup() - спречава појаву дупликата класе коришћењем глобалне функције unserialize()

Што се тиче генерисања самог токена, већ је поменута PHP функција уведена од верзије 7 која је препоручена за генерисање псеудо-случајних низова/бројева.

Функција random_bytes() као улазни параметар има обавезан параметар типа int (целобројна вредност) који означава дужину секвенце која ће бити генерисана у бајтовима. Излаз из функције је стринг те дужине. Како је секвенца у бинарном формату, можемо користити још једну функцију, bin2hex() која ће улазни бинарни стринг конвертовати у хексадецимални. Пошто се ради конверзија из бинарног у хексадецимални ASCII (енгл. *American Standard Code for Information Interchange*) запис, дужина стринга који садржи хексадецимални запис ће бити дупло већа од дужине првог који је садржао бинарни запис. Друга опција за ово је коришћење base64_encode() функције, где би евентуално након генерисања Base64 записа избацили знакове '+', '-' и '='. Трећа опција била је коришћење неке хеш-функције. И у другом и у трећем случају добили би токен који користи више различитих карактера. Међутим, одлучено је да је и оваква заштита довољна ако се генерише довољно дугачак токен. За саму дужину токена тврдо-кодована је вредност 32, што даје стринг дужине 64 карактера.

У листингу испод дата је прва функција, која служи за генерисање токена:

```
<?php

class Token
{
 ...
 public static function generate()
 {
       if (!Session::isStarted())
       {
             // Логује се порука о грешци
       }
       Session::set(Config::TOKEN_NAME, hex2bin(random_bytes(32)));
 }
 ...
}
```

Прво је позвана функција isStarted() из класе Session, која служи да обезбеди фасадни приступ сесијској променљиви. У случају да сесија није започета, додаје се лог-запис на нивоу грешке – обуставља се рад скрипте и кориснику се избацује порука о грешци. Унутар лог-фајла чувају се детаљи грешке: датум, време, порука. Такође примећујемо да назив токена није тврдо-кодован, већ се чува у конфигурационом фајлу под константом TOKEN_NAME. Ова функција се мора позвати при аутентификацији корисника, након генерисања сесијског колачића.

Следеће две функције које су дефинисане унутар класе Token су функција value() која служи да врати вредност токена директно или false у случају да сесија није покренута и токен не постоји, и функција generateInputField() која служи да генерише сакривено улазно поље са вредношћу токена тако да се избегне понављање кода унутар дефинисања формулара у слоју приказа. Ове функције су такође урађене у робусном стилу, где се логује грешка у случају да сесија није покренута или генерише токен у случају да функција generate() није позвана на првом месту.

Последња функција која је дефинисана и коју ћемо приказати је функција за валидацију токена и она је дата у листингу испод. Аргумент функције се односи на

метод који се користи за обраду формулара, а подразумевана вредност је POST метод.

```
    <?php

    class Token
    {
     ...
     public static function validate($method = INPUT_POST) {
           $userToken = filter_input($method, Config::TOKEN_NAME);
           if (!$userToken)
           {
                 return FALSE;
           }
           elseif (strlen($userToken) !== 64 or $userToken !==
Session::get(Config::TOKEN_NAME))
           {
                 return FALSE;
           }
           return TRUE;
           }
     ...
    }
```

Овде је такође тврдо-кодована вредност дужине токена, што јесте лоша пракса, али пошто се вредност налази на само два места, већи трошак би био ишчитавање вредности из конфигурационог фајла него ручна измена на та два места.

6 Закључак

Поставља се питање колико SameSite атрибут за колачиће решава проблем CSRF напада и да ли је и даље потребно да се користи заштита путем токенизације. Прво, треба имати у виду да је ово стандард који неће ускоро бити имплементиран на свим прегледачима. Друго, и даље постоји опасност од ускладиштених CSRF напада – у случају да у нашој апликацији постоји XSS пропуст ни један ни други метод заштите нам не помаже. И треће, токенизација пружа већи ниво контроле. У случају да сваки корисник има различит CSRF токен који ће бити прослеђен у формулару и упоређен на страни сервера као провера да ли је корисник стварно послао захтев, бићемо сигурни да је захтев легитиман, а не фалсификован. Код коришћења SameSite колачића, немамо овакав вид контроле. Закључак: користити и један и други вид заштите.

Такође, скренута је и пажња на одговорност програмера код отварања оваквих пропуста. Коришћење џокер оператора код CORS механизма или искључивање CSRF посредног софтвера у Laravel оквиру могу деловати као универзално решење за проблеме који се јављају при развоју апликације/сервиса, али такође могу отворити и гомилу безбедносних проблема. Треба имати у виду да безбедност на Вебу зависи од много фактора, а многи од њих се могу решити у процесу самог развоја софтвера, тако што се познају и примењују безбедносне праксе.

И за крај су дате смернице и за саме кориснике – иако је CSRF превасходно пропуст у Веб-апликацији, сами корисници могу значајно повећати сопствену безбедност на Вебу уколико се едукују мало више о томе шта су претње на Интернету генерално.

7 Литература

[1] Александар Јевремовић, Младен Веиновић, Марко Шарац, Горан Шимић - „Заштита у рачунарским мрежама“, Универзитет Сингидунум, Београд, 2014.

[2] Младен Веиновић, Александар Јевремовић - „Интернет технологије“, Универзитет Сингидунум, Београд, 2013.

[3] Младен Веиновић, Саша Адамовић - „Криптологија 1“, Универзитет Сингидунум, Београд, 2013.

[4] Александар Јевремовић, Младен Веиновић, Марко Шарац, Ненад Ристић, Душан Стаменковић - „FatCookies: HTTP cookies based attack on Web sites availability“, IcETRAN, 2015.

[5] William Zeller, Edward W. Felten - „Cross-Site Request Forgeries: Exploitation and Prevention“, Princeton University, 2008.

[6] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[7] Alma Whitten, J. D. Tygar - „Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0“, USENIX Security Symposium, 1999.

[8] Leo Dorrendorf - „Cryptanalysis of the Random Number Generator of the Windows Operating System“, The Hebrew University of Jerusalem, 2007.