

Webové aplikačné brány

Matúš Makay

STU FIIT

7

LS 2022/23

Abstract—V práci sa budem venovať problematike webových aplikačných brán. Na to, aby som lepšie pochopil na čo slúžia, sa najskôr zameriam na históriu firewallov, zistím aké druhy existujú a akým spôsobom zabezpečujú sieť pred hrozbami. Neskôr sa zameriam na špecifický druh aplikačného firewallu, ktorým je webová aplikačná brána. Zanalyzujem konkrétne riešenia webových aplikačných brán, dostupných na internete, ktorých činnosť následne otestujem pomocou voľne dostupného skenovacieho softvéru.

Kľúčová problematika

- 1) História a druhy firewallov
- 2) Webová aplikačná brána
- 3) Analýza voľne dostupných riešení webových aplikačných brán
- 4) Pred akými hrozbami chránia webové aplikačné brány webové aplikácie
- 5) Testovanie webových aplikačných brán

I. HISTÓRIA A DRUHY FIREWALLOW

Firewall je mechanizmus, ktorý slúži na zabezpečenie sieťovej premávky, nakoľko už v skorých rokoch internetu sieťoví administrátori prišli na to, že ich siete sú zneužívané a napádané externými útočníkmi. ” Túto situáciu vyriešil pojem firewall ktorý Steven Bellovin pracovník z ATT popísal ako proces filtrovania nechcenej sieťovej premávky. ”[10] Princíp bol vložiť filtre medzi odchádzajúce a prichádzajúce pakety a tie, ktoré nesplnia kritéria zahodiť, a tým zabezpečiť premávku, ktorá vstupuje do internej siete.

V priebehu rokov sa firewall vyvinul do rôznych typov firewall-ov. V rôznych zdrojoch som našiel rozličné rozdelenie základných typov firewall-ov, pričom som usudil že toto rozdelenie nie je až také dôležité ako uvedomenie si že každý typ firewallu sa zameriava na inú problematiku. Preto sú pre lepšiu efektivitu a bezpečnosť často využívané viaceré druhy firewallov v kombinácií.

Rozdelenie firewallov do základných skupín podľa filtrovania a analýzy paketov:

- Paketové firewall-y
- Stavové firewall-y
- Aplikačné firewall-y
- Dynamické paketové firewall-y

A. Paketové firewally

Tento typ firewallu operuje na 3(linkovej) vrstve OSI modelu a je najstarší a najjednoduchší typ firewallu ktorý kontroluje tok paketov do siete a von zo siete na základe preddefinovaných nastavení. Tieto nastavenia zahŕňujú používateľské špecifické nastavenia, povolené IP adresy, porty a protokoly. Ak paket nesplní predefinované požiadavky nie je pustený do alebo von zo siete a je zahodený. [6]

Hlavná výhoda tohto typu je jeho jednoduchosť, rýchlosť vyhodnocovania (nespomaluje reaktivitu systému) a efektivita. Na druhú stranu tento typ firewallu nie je schopný hlbšej analýzy paketu a nedokáže rozlíšiť či je daný paket škodlivý alebo nie. Ak príde paket ktorý patrí do množiny povolených IP adries, portov a typov protokolov, bude vpustený do siete aj keď dáta ktoré prenáša môžu byť škodlivé. [7]

Kontrolovanie stavu paketu po príchode do firewall-u, podľa [7]:

- 1) Zdrojová IP adresa
- 2) Cieľová IP adresa
- 3) Typ protokolu
- 4) Porty (Zdrojový a cieľový)
- 5) Nastavené vlajky paketu
- 6) Smer paketu (prichádza/odchádza zo siete)
- 7) Fyzické rozhranie na ktoré paket prišiel

Aké druhy filtrovania paketov poznáme, podľa [7]:

- 1) Dynamické filtrovanie paketov
- 2) Statické filtrovanie paketov
- 3) Bezstavové filtrovanie paketov
- 4) Filtrovanie paketov so stavom

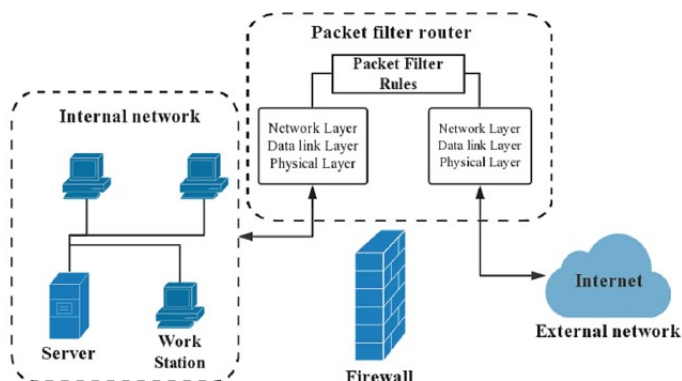


Fig. 1. Princíp filtrovania paketov [8]

B. Stavové paketové firewall-y

Tento typ firewall-u pracuje na piatej (relačnej) vrstve OSI modelu. Je zodpovedný najmä za verifikáciu a zabezpečenie TCP alebo UDP spojenia. To či je komunikácia bezpečná vyhodnocuje najmä na potvrdzovacích paketoch ktoré sa posielajú medzi dvoma uzlami pri sieťovej komunikácii. Principiálne sa dajú porovnať s **A. paketovými filtrami** pretože prichádzajúci paket vyhodnocujú rovnako "povrchne".

Postup spracovania komunikácie:

- 1) Prijatie inicializačného paketu a odsledovanie správneho nadviazania komunikácie (Three Way Handshake)
- 2) Vytvorenie virtuálneho spojenia medzi komunikujúcimi uzlami (princíp proxy servera)
- 3) Vytvorenie príslušnej tabuľky na udržiavanie stavu komunikácie, na základe tejto tabuľky rozhoduje ktorá komunikácia prejde dovnútra siete alebo bude zahodená
- 4) Po nadviazaní komunikácie už sa nezaobera kontextom paketu ale sleduje len sieťové informácie, pričom ak sa sieťová informácia paketu zhoduje s uloženými tabuľkami prepustí paket dnu.
- 5) Po ukončení komunikácie vymaže príslušnú tabuľku

Výhody tohto typu firewall-u:

- Maskuje vnútorný uzol komunikácie pred vonkajším, správa sa ako proxy server
- Jednoduchý na implementáciu
- Nefiltruje pakety ale namiesto toho sleduje stav a správnosť komunikácie
- Menej spracovania ako aplikačné firewall-y z čoho vyplýva menšie zaťaženie siete

Nevýhody tohto typu firewallu

- Ako pri **A. paketových filtroch** neobsahujú potrebnú hĺbkovú analýzu paketov z čoho vyplýva nebezpečenstvo prepustenia škodlivých dát.
- Časté updatovanie systému je nevyhnutné

C. Aplikačné firewall-y

Aplikačný firewall tiež známy ako proxy firewall je implementovaný ako proxy server na siedmej(aplikačnej) vrstve OSI modelu. Podobne ako **B. Stavové paketové firewall-y** vytvára spojenie medzi vnútorným a vonkajším, pričom nezáleží či sa dopytujeme uzlom čím maskuje a zamedzuje priamy prístup z vonkajšej siete do vnútornej, pričom nezáleží či sa vnútorný uzol dopytuje von zo siete alebo sa vonkajší uzol dopytuje na uzol vo vnútornej sieti.

Narozdiel od ostatných firewallov aplikačné firewally neanalyzujú len sieťové hlavičky ale prevádzajú hĺbkovú analýzu paketu, spracovávajú hlavičku a aj prenášané dáta paketu. Ako všetky firewally majú prednastavený zoznam pravidiel na základe ktorých vyhodnocujú prichádzajúcu premávku. Paket ktorý prichádza zanalyzujú, vytvoria výsledok analýzy a na základe výsledku rozhodnú či paket prepustia ďalej alebo zahodia.

Výhody

- Tým že sprostredkovávajú komunikáciu medzi uzlami zabezpečujú anonymitu komunikujúceho uzla
- Hĺbkovou analýzou paketu zastavujú aj pakety ktoré prenášajú škodlivé dáta čím oproti ostatným firewallom pridávajú ďalšiu vrstvu ochrany

Nevýhody

- Hĺbková analýza paketov si žiada svoju daň v podobe zníženej sieťovej latencie systému
- Správne ich nakonfigurovať pre optimálnu ochranu môže byť náročné
- Nepracujú so všetkými sieťovými protokolmi

D. Dynamické firewall-y

Operujú na tretej (linkovej) a štvrtej (transportnej) vrstve OSI modelu podobne ako A. **Paketové filtre**. Narozdiel od nich si však tento typ firewall-u udržuje *stavovú tabuľku* spojení. Takže nekontroluje premávku len na základe preddefinovaných pravidiel ale kontroluje ich aj so stavom komunikácie v tabuľke ktorý sa môže dynamicky meniť podľa stavu komunikácie napríklad pomocou sledovanie stavových atribútov TCP komunikácie (SYN, FIN, ACK, RST).

Čo si udržuje v stavovej tabuľke:

- Zdrojovú, cieľovú IP adresu
- Zdrojový, cieľový port
- Stav spojenia
- Stavové atribúty spojenia
- Poradové a potvrdzovacie čísla

Ako vyhodnocuje prichádzajúci paket:

- 1) Zanalyzuje hlavičku paketu a extrahuje z nej IP adresy, porty a typ protokolu
- 2) Skontroluje či je daný paket súčasťou niektorej z komunikácií v stavovej tabuľke
- 3) Ak je súčasťou niektorej z komunikácie potom skontroluje či stav paketu vyhovuje zabezpečovacím pravidlám komunikácie. Ak vyhovuje je prepustený a nie je zahodený.
- 4) Ak nie je súčasťou komunikácie, preskúma či sa daný paket pokúša otvoriť nové spojenie. Ak hej vytvorí nový záznam v tabuľke a paket prepustí.

E. Moderné firewally

Nakoľko siete čelili stále komplexnejším a zložitejším útokom ich obrana sa musela vyvíjať. Moderné firewally preto kombinujú funkcionality ktoré som uvádzal vyššie aby sieť ešte lepšie zabezpečili. V moderných firewalloch sa ďalej môžu vyskytnúť systémy ako IPS (Intrusion Prevention System) systém. Ktorý neustále spracováva sieťovú premávku a ak nastane nejaká

hrozba mal by ju zdetegovať a vykonať opatrenia aby zabránil väčším škodám. Ďalej môžeme spomenúť IDS (Intrusion detection system) ktorý funguje podobne ako IPS len s tým rozdielom že nemôže zakročiť a zablokovať ju ale iba upovední administrátorov a nebezpečenstve. [3]

II. WEBOVÉ APLIKAČNÉ BRÁNY

V posledných dekádach sa popularita webových aplikácií značne zvýšila. Technológie ktorými možno webovú aplikáciu implementovať sa stali dostupnejšími a jednoduchšími. S vývojom webových frameworkov sa umožnilo vývojom vytvárať komplexné webové aplikácie a doručovať ich používateľom jednoducho cez webový prehliadač. Obrovská popularita webových aplikácií ale predstavuje závažné bezpečnostné riziko nakoľko spôsoby akým môže byť webová aplikácia prelomená je veľmi veľa bližšie ich rozoberiem v kapitole IV. **Hrozby ktorým čelia webové aplikácie**. V počiatkoch internetu bolo zabezpečovanie webových stránok na developeroch ale s rastúcou komplexnosťou útokov sa táto zodpovednosť presunula na aplikačné firewally. Jedným z nich je aj Webová aplikačná brána ktorej fungovanie si bližšie rozoberieme v tejto kapitole.

Webové aplikačné brány patria do skupiny aplikačných firewall-ov. Operujú na aplikačnej (siedmej) vrstve modelu OSI. Ich hlavná úloha je zabezpečiť webovú aplikáciu bez nutnosti zmeny jej zdrojového kódu. Webové aplikačné brány sú špecifický softvér, ktorý má za úlohu zabezpečiť webové aplikácie pomocou filtrovania aplikačných protokolov ako sú HTTP a HTTPS komunikácie medzi webovým serverom a internetom. Webová aplikačná brána rozumie a pozná aplikačné protokoly, takže je schopná ich analyzovať a vyhodnocovať, ktorá požiadavka obsahuje zlý kód, ktorý chce získať neoprávnený prístup do systému alebo spôsobiť neoprávnený únik dát. Ochrana je definovaná pomocou politiky webovej aplikačnej brány a ak chce klient pristúpiť k serveru, musí spomínanú politiku splniť a následne je presmerovaný na webový server. Správna otázka je akým spôsobom môže byť spomínaná politika webových aplikačných brány

implementovaná. Poznáme **tri modely** ktorými sa dajú klasifikovať zabezpečovacie *politiky webových aplikačných brán*

- 1) Negatívny model
- 2) Pozitívny model
- 3) Zmiešaný model

A. Negatívny model

Tento model využíva princíp čiernej listiny. To znamená že má vopred zadefinované špecifické vektory požiadaviek ktoré budú pri analýze zablokované. Predstavme si situáciu že útočník pošle pri prihlasovaní do webovej aplikácie v jednom zo vstupov požiadavku v ktorej sa bude nachádzať nasledujúci kód "SELECT * FROM users". Webová aplikačná brána by mala túto požiadavku zamietnuť pretože bude mať v čiernej listine vektor ktorému zodpovedá štruktúra danej požiadavky. Po odmietnutí komunikácie nasledujúca aktivita webovej aplikačnej brány je uložiť si útočnickovú IP adresu a požiadavky z jeho strany po určitú dobu zamietat. Táto vlastnosť blokovania sa dá využiť na odhalovanie ktoré požiadavky systém zamietá a týmto spôsobom poskytovať informácie útočníkovi.

Ďalšia nevýhoda tohto modelu je potreba myslieť na všetky možnosti ktorými útočník môže spôsobiť únik citlivých dát. Zoberme si len príklad ktorý som spomenul vyššie. Ak by útočník pri prihlasovaní vložil do vstupu nasledujúci kód "SeLeCt * FrOm users" táto požiadavka by už mohla byť webovou aplikačnou bránou prepustená nakoľko je veľa možností ako takýmto jednoduchým spôsobom môžeme vstup pozmeniť a takto predvídať všetky spôsoby ako môže útočník vniknúť do systému je náročné. Tento spôsob zabezpečuje webovú aplikáciu náhodne a z tohto dôvodu sa už webové aplikačné brány len s týmto modelom ďalej samostatne nevyvíjajú.

B. Pozitívny model

Tento model využíva princíp bielej listiny. To znamená že má vopred definované povolené operácie a tým pádom je nezávislý na škodlivých vektoroch. Počiatočný stav tohto typu je že zamietá všetky pokusy o komunikáciu. Z tejto vlastnosti vyplýva že daný typ webovej aplikačnej brány

musí poznať všetky spôsoby akými aplikácia interaguje a tieto spôsoby sa musia pri návrhu zohľadniť. Tento typ modelu je veľmi efektívny dokáže zablokovať veľa časť škodlivej premávky lenže pri zlom nastavení sa môže stať že zablokuje aj normálnu prevádzku. Táto vlastnosť je veľmi reštriktívna a z tohto dôvodu sa tento typ využíva najmä pri webových aplikáciach ktoré nie sú určené pre širokú verejnosť ale len pre užšiu skupinu.

C. Zmiešaný model

Hybridný alebo zmiešaný model je kombinácia oboch predchádzajúcich spôsobov. Väčšina komerčne známych webových aplikačných brán využíva tento typ modelu. Nakoľko v praxi sa ukázalo že kombinácia oboch prístupov je pre bežné webové aplikačné brány najlepší spôsob implementácie nakoľko sa dajú jednoduchšie nakonfigurovať pre rôzne druhy webových aplikácií.

Každá webová aplikačná brána implementuje svoje vlastné bezpečnostné funkcionality ale na to aby garantovala potrebnú úroveň bezpečnosti musí obsahovať minimálne tento zoznam funkcionalít. Nesmieme ale zabúdať ani na správnu konfiguráciu webovej aplikačnej brány, pretože ak nie je správne nakonfigurovaná rozhodne sa nespráva podľa očakávaní a môže blokovať aj nechcenú premávku. Konfigurácia webovej aplikačnej brány je rovnako dôležitá a hrá kľúčovú rolu v jej nasadovaní. Zoznam minimálnych požiadaviek na webovú aplikačnú bránu podľa [2]:

- Podpora známych štandardov pre hrozby ako OWASP Top 10
- Kontrola vstupov a výstupov webovej aplikácie založenej na preddefinovaných bezpečnostných pravidlách.
- Implementácia oprácií zablokuj, povol, upozorni a zrozumiteľné výpisy aktivít ktoré nastanú
- Zabrániť úniku citlivých dát, tým že budú kontrolovať výstupy webovej aplikácie
- Implementácia bezpečnostných pravidiel organizácie pomocou Pozitívneho, negatívneho alebo hybridného modelu
- Vedieť kontrolovať obsah webovej stránky, analýza HTML, CSS a dynamického HTML

- Vedieť analyzovať prenosné aplikačné protokoly ako HTTP, HTTPS, TLS
- Zabrániť útokom a neoprávnenému vstupu do webovej aplikačnej brány
- Podpora SSL a TLS
- Jednoduchý update systému

D. Schéma zapojenia webovej aplikačnej brány

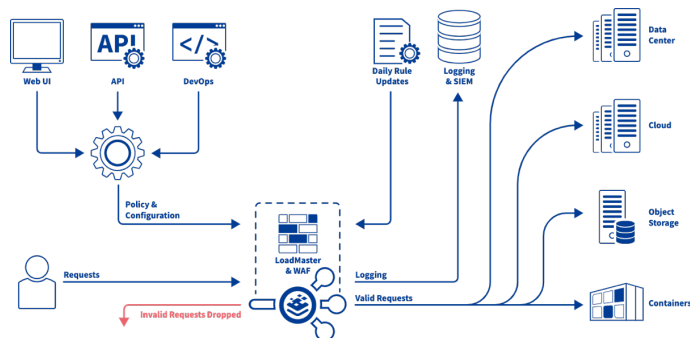


Fig. 2. Blokové zapojenie webovej aplikačnej brány [11]

Hlavná nevýhoda Webovej aplikačnej brány je, že spomaľuje rýchlosť odozvy webového servera nakoľko sa premávka musí duplikovať a posielat' skrz webovú aplikačnú bránu. Na druhú stranu je tu výhoda vyššej bezpečnosti webového produktu a úrýchlenie vývoja, nakoľko developeri nemusia implementovať základné obranné techniky. Webová aplikačná brána ale nie je zázračné riešenie, ktoré zastaví každý útok a pri vývoji produktu sa musí brať ohľad na princípy bezpečného vývoja. [1] [12]

III. ANALÝZA VOĽNE DOSTUPNÝCH RIEŠENÍ WEBOVÝCH APLIKAČNÝCH BRÁN

Web sa stal perfektnou lokalitou pre poskytovanie služieb zákazníkom, je to hlavne z dôvodu prenositeľnosti webových aplikácií nakoľko na otvorenie webovej aplikácie stačí mať nainštalovaný browser a pripojenie na internet. Túto výhodu na poskytovanie obsahu zákazníkom využili mnohé firmy pričom spoiatku za bezpečnosť webovej aplikácie zodpovedali programátori webovej aplikácie. Pričom nedostatočné zabezpečenie webovej aplikácie môžu viesť k miliónovým škodám. Avšak s rastúcou frekvenciou a komplexnosťou útokov sa táto zodpovednosť preniesla na webové aplikačné brány ktoré zodpovedajú za bezpečnosť a chránia webové aplikácie pred hrozbami ako skriptovanie naprieč sieťou, sql injekcie a tak ďalej. Viac sa tejto téme venujem v **IV. Hrozby ktorým čelia webové aplikácie.**

Zabezpečujú však voľne dostupné webové aplikácie brány webové aplikácie proti všetkým hrozbám? Na internete je dostupné veľké množstvo webových aplikačných brán. V tejto časti práce sa pozrieme bližšie na tri najznámejšie:

- 1) ModSecurity
- 2) WebKnight
- 3) Guardian

Vyhodnocovanie efektivity podľa [9]:

- 1) Počet útokov ktoré webová aplikačná brána zastavila
- 2) Počet chýb z hľadiska zablokovanej bežnej prevádzky

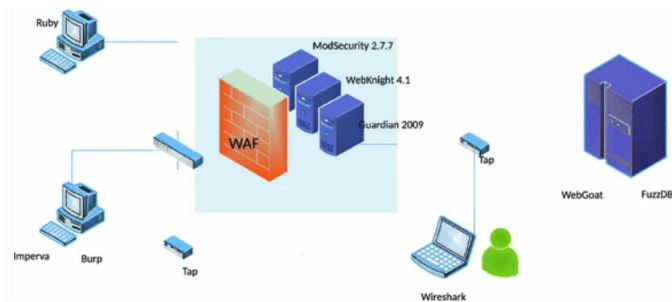


Fig. 3. Zapojenie testovacích zariadení [9]

A. Popis testovania

Webové aplikačné brány boli nastavené podľa defaultnej konfigurácie. Testy pozostávali z viacerých častí. Útoky boli najskôr spúšťané z frameworku Imperva ktorý poskytuje širokú škálu generovania zlomyselnej premávky, tento framework bol využitý aj na generovanie bežnej premávky na otestovanie schopnosti webových aplikačných brán rozlišovať nebezpečnú a bežnú premávku. Ďalej sa testovali Fuzz-DB útoky, pomocou Burp suite boli testované útoky ako skriptovanie naprieč sieťou a SQL injekcie[9]

B. Generovanie zlomyselnej prevádzky

WAF Tests	Mod Security	WebKnight	Guardian@Jumperz
Total XSS tests	223	223	223
XSS Bypassed	3	1	89
XSS Blocked	220	222	134
Total SQLi tests	43	43	43
SQLi Bypassed	0	1	24
SQLi Blocked	43	42	19
Total RFI/LFI tests	26	26	26
RFI/LFI Bypassed	0	0	26
RFI/LFI Blocked	26	26	0
Total Debug/Admin Flag Attacks	40	40	40
Debug/Admin Flag Bypassed	40	40	40
Debug/Admin Flag Blocked	0	0	0
Total Directory Disclosure Attacks	10	10	10
Directory Disclosure Bypassed	9	5	9
Directory Disclosure Blocked	1	5	1
Total HTTP Manipulation Attacks	115	115	115
HTTP Manipulation Bypassed	25	0	0
HTTP Manipulation Blocked	98	115	115
Total Integer Overflow Attacks	36	36	36
Integer Overflow Bypassed	13	28	36
Integer Overflow Blocked	23	8	0
Total LDAP Attacks	27	27	27
LDAP Bypassed	16	0	15
LDAP Blocked	11	27	12

Fig. 4. Tabuľka s výsledkami vyhodnocovania nebezpečnej premávky [9]

Z tabuľky môžeme vidieť že ModSecurity a WebKnight boli schopné zastaviť väčšinu pokusov o útok, naopak Guardian nezastavil väčšinu útokov. Z výsledkov vychádza že WebKnight ma najlepšiu efektivitu s 85.6% zablokovaných útokov, druhý najlepší skončil ModSecurity ktorý zachytil 79.7% útokov, najhoršie skončil Guardian ktorý bol schopný zachytiť 56% vygenerovaných útokov. [9]

C. Chybné vyhodnotenie zlomyselnej prevádzky

Imperva Friendly Tests	Mod Security	WebKnight	Guardian@Jumperz
Total Friendly tests	148	148	148
Friendly Blocked	76.5	147	6.1
Friendly Passed	71.5	1	141.9

Fig. 5. Tabuľka s výsledkami vyhodnocovania bežnej premávky [9]

Jedna z najdôležitejších vlastností webových aplikačných brán je rozlišovanie bežnej od nebezpečnej premávky. Nakoľko táto vlastnosť dokáže veľmi ovplyvniť používateľský zážitok. Z výsledkov testovania môžeme vidieť že najlepšie prešiel Guardian z 5% blokovanvej bežnej premávky, druhý najlepší bol ModSecurity ktorý vyhodnotil falošne pozitívne 52% bežnej premávky a najhoršie dopadol WebKnight ktorý zablokoval 98.54 bežnej premávky.[9]

D. Zhrnutie

Ako sme mohli vidieť ani jedno z riešení nebolo úplne bezpečné a stále niektoré útoky ostali nepovšimnuté systémami ktoré ich majú chrániť. Ak však prihliadneme na to že tieto systémy sú voľne dostupné tak svoju rolu čiastočne splnili. V práci [9] testovali webové aplikačné brány podľa viacerých kritérií ktoré som vo svojej práci nezahrnul a zo všetkých testov vyšiel najlepšie ModSecurity ktoré sa vyznamenalo ako najlepšie voľne dostupné riešenie. Pričom WebKnight dokázal zablokovať viaceré pokusy o útoky ale tak tiež zablokoval oveľa väčšiu časť bežnej premávky ako ostatné riešenia takže z testovania vyšiel najhoršie.

IV. HROZBY, KTORÝM ČELIA WEBOVÉ APLIKÁCIE

Webové aplikácie čelia útokom na dennej báze. Útočníci sú stále kreatívnejší v hľadaní zraniteľností webových aplikácií. Tu prichádza do hry OWASP, Open Web Application Security Project, nezisková organizácia, ktorej hlavným cieľom je zabezpečovanie webových aplikácií. Ich najznámejší projekt je OWASP Top 10, ktorý opakujú v pravidelných cykloch. Tento projekt sa zameriava na vyhľadávanie najčastejších zraniteľností, ktorým podliehajú webové aplikácie. [5]

- 1) Zlyhanie kontroly prístupu
- 2) Kryptografické zlyhania
- 3) SQL injekcie
- 4) Nezabezpečený dizajn
- 5) Nesprávna konfigurácia zabezpečenia
- 6) Zraniteľné a zastarané komponenty
- 7) Zlyhania pri identifikácii a autentifikácii
- 8) Poruchy softvéru a integrity údajov
- 9) Zlyhanie monitorovania bezpečnostných logov
- 10) Falšovanie požiadaviek na strane servera

A. Zlyhanie kontroly prístupu

Toto je veľmi jednoduchá a veľmi častá chyba pri vývoji webových aplikácií. V dobre navrhnutom softvéri by mali byť vytvorené rôzne triedy privilégii ktoré garantujú používateľom vykonávanie akcií a získavanie dát na ktoré majú oprávnenia v rámci ich skupiny. Na tento fakt sa však často zabúda pričom zanedbanie môže spôsobiť vážne škody.

Táto zraniteľnosť zahŕňa [5]:

- Porušenie princípu najnižšej privilegovanosti alebo predvoleného odmietnutia kde by mali byť akcie, dáta zobrazené iba určitej triede používateľov ale je zobrazený hocikomu
- Zobrazenie a modifikácia iného účtu poskytnutím jedinečného identifikátora účtu do požiadavky
- Prístup k aplikačnému serveru a endpointom bez potrebného overenia používateľa pre vykonanie POST, PUT, DELETE požiadavky
- Elevácia práv, možnosť vydávania sa za používateľa bez toho aby prebehle prihlásenie alebo vydávanie sa za admina ak som prihlásený iba ako používateľ

Ako zabrániť tejto zraniteľnosti podľa [5]:

- Pokiaľ nejde o verejne dostupný zdroj, predvolene odmietaj prístu
- Prístup k modelom by mal vynútený na základe vlastníctva záznamu
- Zaznamenávať zlyhanie autentifikácie a notifikovať adminov ak je to vhodné
- Vymedziť rozsah na kontroller čím sa dá zamedziť škoda od automatických skenovacích nástrojov
- Zabezpečenie aby metadáta súborov ako .git neboli v priečinku bežiackej webovej aplikácie

Príklady útokov

V tomto útoku útočník jednoducho zadá identifikátor ktorý neidentifikuje jeho účet a ak daný účet existuje a tento prípad nie je ošetrovaný, útočník môže získať prístup k účtu iného používateľa

```
https://example.com/app/accountInfo?acct=notmyacct
```

Fig. 6. Příklad možného útoku[5]

Útočník jednoduchým spôsobom donúti browser na získanie URL a týmto jednoduchým spôsobom môže získať neautorizovaný prístup

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

Fig. 7. Příklad možného útoku[5]

B. SQL injekcie

Táto zraniteľnosť spočíva v tom že útočník zadá SQL kód do inputu formulára vo webovej aplikácii a ak webový aplikačný server pri tvorbe dynamického SQL dopytu neošetrí vstup od používateľa táto nebalosť môže spôsobiť že útočníkovi budú zobrazené informácie ktoré mu neboli určené. V princípe je to technika napadnutia databázovej vrstvy aplikácie.

Kedy je aplikácia zraniteľná podľa: [5]

- Používateľom poskytnuté dáta nie sú validované, filtrované aplikáciou
- Ak sa dáta s dopytom priamo používajú alebo spájajú do SQL dopytu
- Ak sa dáta z dopytov používajú v objektovo relačnom mapovaní (ORM) ako vyhľadávacie

parametre na extrahovanie ďalších citlivých údajov

Ako zabrániť zraniteľnosti podľa: [5]

- V SQL dopytoch používať zabudované funkcie limitovanie výsledkov dopytu
- Používanie objektovo relačných frameworkov (ORM)
- Časti SQL dopytov ktoré odkazujú na názvy tabuliek, názvy stĺpcov by sa nemali nahrádzať premennými
- Používanie parametrizovaných dopytov namiesto dynamických. Parametrizované dopytu donúti programátora zdefinovať všetky časti SQL dopytu a neskôr vložiť používateľské dáta. Táto metóda dovoľí databáze rozlíšiť čo je SQL dopyt a čo sú data poskytnuté používateľom.

Príklady útokov: Takto môže vyzeráť príklad tvorby dynamického SQL dopytu na strane aplikácie servera.

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Fig. 8. Tvorba dynamického SQL dopytu[5]

V takomto prípade stačí jednoducho modifikovať URL adresu a namiesto používateľského unikátneho identifikátora vložiť SQL kód a aplikácia server by vrátil všetky záznamy v tabuľke používateľov. Čo by mohlo byť horšie že SQL dopyt by sa dal modifikovať na to aby modifikoval záznamy alebo dokonca vymazal záznamy z databázy

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--
```

Fig. 9. Príklad modifikácie URL adresy[5]

V. TESTOVANIE

Na testovanie som zvolil iný postup ako som uvádzal v zámere projektu. Nakoniec som si vybral jednu webovú aplikáciu ktorú budem testovať a tou je ModSecurity. Ako webový server som si vybral nginx nakoľko je predinštalovaný na linuxovej distribúcii a voľne dostupný. Posledný kúsok skladačky je webová aplikácia ktorú som vytvoril pre tento účel. Najskôr si podrobnejšie rozoberieme všetky entity testovania. Ako ďalšie napíšem ako som postupoval pri konfigurácii jednotlivých entít. A v poslednej časti budú samotné útoky a správanie webovej aplikácie brány.

A. ModSecurity

ModSecurity je open source webová aplikácia brána ktorá je podporovaná pre webové servery ako Apache, nginx a IIS. ModSecurity poskytuje flexibilné pridávanie pravidiel na vykonávanie jednoduchých a zložitých operácií pri vyhodnocovaní.



Fig. 10. Logo ModSecurity firewallu[5]

ModSecurity poskytuje Core Rule Set ktorý obsahuje širokú škálu pravidiel na detekciu útokov vrátane OWASP top ten. Taktiež nás tieto pravidlá chránia pred útokmi ako SQL injekcie, Cross Site Scripting, Local File Inclusion atď. Túto webovú aplikáciu som si vybral pretože podľa zdroja [9] je to jedno z najlepších voľne dostupných riešení na trhu.

ModSecurity poskytuje dva druhy logov pričom jeden z nich je **DebugLog** ktorý sa dá nastaviť na rôzne úrovne výpisov a poskytuje veľmi podrobný náhľad do toho ako ModSecurity funguje na pozadí. Ja v tejto práci budem analyzovať len

Audit logy ktoré poskytujú sumár informácií ako bola požiadavka vyhodnotená

Vysvetlenie konfigurácie:

- SecAuditEngine: nastavuje ktoré požiadavky sa budú vypisovať do logov
 - RelevantOnly: vypíše len požiadavky ktoré sa vyhodnotia ako nebezpečné
 - on: vypisuje všetky požiadavky, vhodné na monitorovanie všetkej premávky
 - off: audit logy nebudú vypisované
- SecAuditLogParts: definuje štruktúru audit logu, tieto časti bližšie popíšem neskôr
- SecAuditLogType: rozhoduje ako sa budú logy ukladať
 - Serial: všetky logy sú súčasťou jedného súboru
 - Concurrent: jeden log = jeden súbor
- SecAuditLog: rozhoduje do akého súboru sa budú logy ukladať

```
# -- Audit log configuration -----
# Log the transactions that are marked by a rule, as well as those that
# trigger a server error (determined by a 5xx or 4xx, excluding 404,
# level response status codes).
#
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?:?!04))"

# Log everything we know about a transaction.
SecAuditLogParts ABFZ

# Use a single file for logging. This is much easier to look at, but
# assumes that you will use the audit log only occasionally.
#
SecAuditLogType Serial
SecAuditLog /home/matus/pib.log

# Specify the path for concurrent audit logging.
#SecAuditLogStorageDir /opt/modsecurity/var/audit/
```

Fig. 11. Moja konfigurácia audit logov

Štruktúra audit logu:

- A: Hlavička audit logu
- B: Hlavička HTTP požiadavky
- C: Telo HTTP požiadavky
- E: Telo HTTP odpovede
- F: Hlavička HTTP odpovede
- Z: Finálne vyhodnotenie audit logu

Aké pravidlá som použil v main.conf:

```
Include /etc/nginx/modsec/modsecurity.conf
Include /usr/local/modsecurity-crs/crs-setup.conf
Include /usr/local/modsecurity-crs/rules/*.conf
```

Fig. 12. Použité pravidlá

B. Nginx

Je to voľne dostupný a voľne šíriteľná technológia ktorá sa preslávila ako webový server ale v dnešnej dobe sa používa ako reverse proxy, vyrovnávacia pamäť pre HTTP a vyrovnávač zaťaženia. Narozdiel od ostatných webových serverov nginx namiesto nevytvára nový proces pre každú webovú požiadavku ale namiesto toho používa asynchrónne a udalostami riadené programovanie kde viaceré požiadavky môžu byť spracované jedným thread-om.

Nginx obsahuje jeden master proces ktorý kontroluje viacere worker procesy pričom tieto procesy vykonávajú spracovávanie požiadaviek. Keďže Nginx je asynchrónny tak viaceré požiadavky môžu byť spracovávané worker threadom súčasne bez toho aby sa blokovali

Nginx podporuje:

- Reverse Proxy Caching
- IPv6
- Load Balancing
- WebSockets
- Servovanie statických suborov
- TLS/SSL



Fig. 13. Logo nginx [4]

Postup konfigurácie nginx:

- Skontroloval som inštaláciu nginx
- Vytvoril som 2 servery:
 - zabezpečený s ModSecurity na porte 80
 - nezabezpečený na porte 81
- Nakonfiguroval som servery ako reverse proxy pre moju aplikáciu
- Nakonfiguroval som ModSecurity a pridal ho do konfigurácie jedného zo serverov

Môžeme vidieť že konfigurácie serverov sú takmer totožné až na dva riadky ktoré aktivujú Mod-Security.

Konfigurácia zabezpečeného servera:

```
## SECURE WEB APPR
server {
    listen 80 ;

    modsecurity on;
    modsecurity_rules_file /etc/nginx/modsec/main.conf;

    location / {
        proxy_request_buffering off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass http://localhost:3000/;
        proxy_redirect http://localhost:3000/ https://$server_name/;
    }
}
```

Fig. 14. Konfigurácia zabezpečeného web servera

Konfigurácia nezabezpečeného servera:

```
## UNSECURE WEB APP
server {
    listen 81;

    location / {
        proxy_request_buffering off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass http://localhost:3001/;
        proxy_redirect http://localhost:3001/ https://$server_name/;
    }
}
```

Fig. 15. Konfigurácia zabezpečeného web servera

C. Webová aplikácia

Provizorný webový aplikačný server som naprogramoval v expressJs. Ako databázu som si vybral Postgres kde som vytvoril jedného používateľa. S používateľským menom pib a heslom supertajne. **Screenshot prihlasovacieho formulára:**

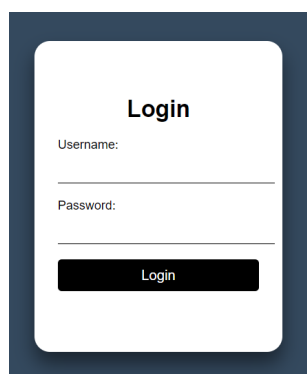


Fig. 16. Prihlasovací formulár

Screenshot úspešného prihlásenia:

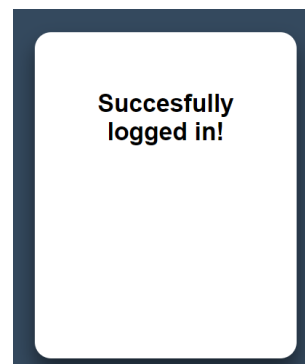


Fig. 17. Prihlásenie do systému

Screenshot neúspešného prihlásenia:

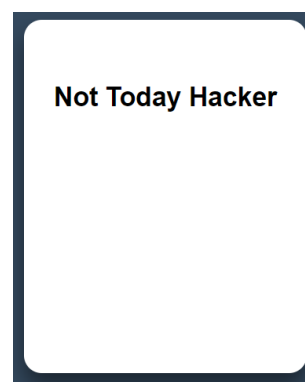


Fig. 18. Odmietnutie prihlásenia

D. Realizácia útokov

V tejto práci som sa rozhodol na moju webovú aplikáciu zaútočiť pomocou:

- Vykonávania vzdialeného kódu
- Pomocou SQL injekcie extrahovať dáta

Vykonávanie vzdialeného kódu:

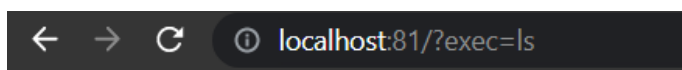


Fig. 19. Simulácia vykonávania vzdialeného kódu

Reakcie aplikácií:

Ako sa dalo očakávať webový server bežiaci na porte 81 daný útok nezastavil a požiadavka prešla až na aplikačný server. Naopak webový server zabezpečený s ModSecurity danú požiadavku označil za nebezpečnú a vrátil HTTP odpoveď so statusom 403 forbidden (Zakázaný)

Zachytenie útoku:

Nakoľko Audit logy sú veľmi rozsiahle a obsahujú informácie ktoré nie je potrebné zdôrazniť z logu som vybral len časť ktorá upozorňuje na útok.

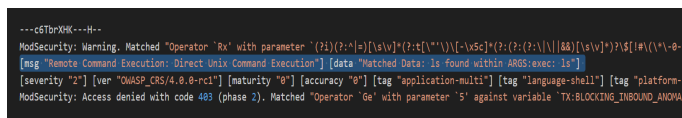


Fig. 20. Audit log detail zachytenia útoku



Fig. 21. Odpoveď na dopyt

SQL injekcia:

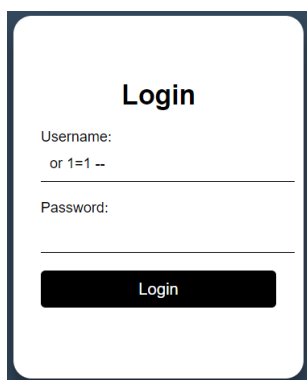


Fig. 22. Simulácia SQL injekcie

Reakcie aplikácií:

Zabezpečený webový server na porte 80 vyhodnotil požiadavku ako nebezpečnú a vrátil 403 ako v predošlej simulácii útoku. Prekvapivé ale bolo že nezabezpečená aplikácia na porte 81 nevrátila žiadne dáta. Po krátkej analýze som prišiel na to že express používa parametrický SQL dopyt o ktorom som písal v IV. Akým hrozbám čelia webové aplikácie takže neunikli žiadne dáta. Odpoveď servera bola rovnaká ako v predošlom príklade takže ju tu nebudem uvádzať v grafickej forme.

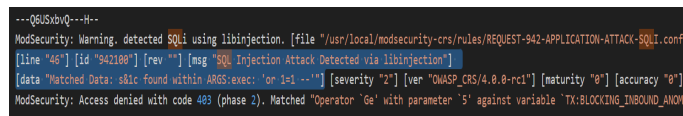


Fig. 23. Audit log detail zachytenia útoku

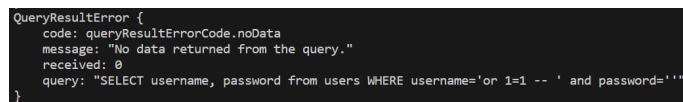


Fig. 24. Výpis na aplikačnom servery

VI. ZÁVER

V práci som sa venoval problematike webových aplikačných brán. Zanalyzoval som prečo vznikli a pred čím nás chránia. Jedno z riešení som si vyskúšal nakonfigurovať a otestoval som ako sa správa pri prípadných útokoch.

REFERENCES

- [1] *Cloudflare WAF*.
<https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/>. [Online; accessed 2023-03-02].
- [2] Z. Ghanbari et al. "Comparative approach to web application firewalls". In: *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. 2015, pp. 808–812. DOI: 10.1109/KBEI.2015.7436148.
- [3] *IPS/IDS*. <https://www.vmware.com/topics/glossary/content/intrusion-prevention-system.html>.
- [4] *Nginx*. <https://kinsta.com/knowledgebase/what-is-nginx/>.
- [5] *OWASP Top Ten — OWASP Foundation*.
<https://owasp.org/www-project-top-ten/>.
- [6] *PacketFiltering*.
<https://www.techtarget.com/searchsecurity/feature/The-five-different-types-of-firewalls>. [Online; accessed 2023-03-03].
- [7] *PacketFiltering*.
<https://www.sunnyvalley.io/docs/network-security-tutorials/what-is-packet-filtering-firewall>. [Online; accessed 2023-03-03].
- [8] *PacketFiltering*.
https://www.researchgate.net/figure/The-principle-of-packet-filtering_fig1_30831979. [Online; accessed 2023-03-03].
- [9] Stefan Prandl, Mihai Lazarescu, and Duc-Son Pham. "A Study of Web Application Firewall Solutions". In: *Information Systems Security*. Ed. by Sushil Jajoda and Chandan Mazumdar. Cham: Springer International Publishing, 2015, pp. 501–510. ISBN: 978-3-319-26961-0.
- [10] *The 5 Different Types of Firewalls Explained*.
<https://www.techtarget.com/searchsecurity/feature/The-five-different-types-of-firewalls>. Jan. 2021.
- [11] *wafImg*. <https://kemptechnologies.com/blog/waf-web-application-firewall-101>. [Online; accessed 2023-03-04].
- [12] *What is a Web Application Firewall (WAF)?* <https://www.f5.com/glossary/web-application-firewall-waf>.