Bezpečnosť webového servera

Bc. András Nagy

Predmet: Bezpečnosť v internete

Cvičiaci: Ing. Rudolf Grežo

Cvičenie: Štvrtok 18:00 - 19:50

Obsah

Úvod

Webové servery sú základnou súčasťou modernej digitálnej infraštruktúry, ktorá umožňuje poskytovanie webového obsahu, aplikácií, rozhraní API a cloudových služieb. Vzhľadom na ich kľúčovú úlohu pri podpore online operácií je bezpečnosť týchto systémov prvoradá. S rastúcim objemom a sofistikovanosťou kybernetických hrozieb sa posilnenie webových serverov stalo ústredným problémom systémových administrátorov, tímov DevOps a bezpečnostných profesionálov.

Zraniteľnosti vyplývajúce zo zastaraného softvéru, nesprávnej konfigurácie alebo nedostatočnej kontroly prístupu môžu vystaviť systémy útokom, ako sú porušenia údajov, eskalácia privilégií alebo odmietnutie služby. Apache HTTP Server (Apache2) a Nginx zostávajú dva z najrozšírenejších webových serverov vďaka svojej flexibilite, výkonu a rozsiahlej komunitnej podpore [1].

Zabezpečenie webového servera zahŕňa niekoľko vrstiev vrátane bezpečných konfiguračných postupov, správnej správy oprávnení a modulov, šifrovania pomocou SSL/TLS a nepretržitého hodnotenia zraniteľnosti. Medzi bežné riziká patrí zneužívanie neopraveného softvéru, povolené nastavenia adresárov, slabé mechanizmy autentifikácie a chyby na úrovni aplikácie, ako napríklad SQL Injection a Cross-Site Scripting (XSS).

Tento článok sa zameriava na bezpečnostnú pozíciu Apache2, analyzuje jeho architektúru, bežné zraniteľnosti a štandardné priemyselné nástroje používané na detekciu a zmierňovanie rizík – ako napríklad Nikto, OpenVAS, OWASP ZAP, Nmap a Acunetix [2]. Nasledujúca časť Analýza poskytne hlbšiu kontrolu konfigurácie zabezpečenia Apache2 spolu s praktickými prístupmi na zvýšenie jeho odolnosti proti útokom.

Analýza

Apache HTTP Server 2.4

Tento webserver je postavený na modulárnej a vysoko rozšíriteľnej architektúre, ktorá umožňuje administrátorom a vývojárom prispôsobiť server širokému spektru potrieb, od jednoduchých statických webových stránok až po zložité dynamické aplikácie [3].

Jadrom Apache 2.4 je jeho modulárny dizajn, ktorý oddeľuje základné funkcie od voliteľných funkcií prostredníctvom systému načítateľných modulov. Samotný server poskytuje ľahkú základňu, zatiaľ čo moduly – označované ako *mod_* – môžu byť načítané alebo vypnuté podľa špecifických požiadaviek. To zaručuje efektívnu prevádzku, zlepšuje výkon a posilňuje bezpečnosť minimalizovaním potenciálnej plochy útoku [4].

Server funguje pomocou modulu pre viacnásobné spracovanie (*Multi-Processing Module*, MPM), ktorý určuje, ako sa spracovávajú požiadavky klientov. Apache 2.4 zaviedol udalosťami riadený MPM ako predvolený pre mnohé systémy, ktorý ponúka lepšiu

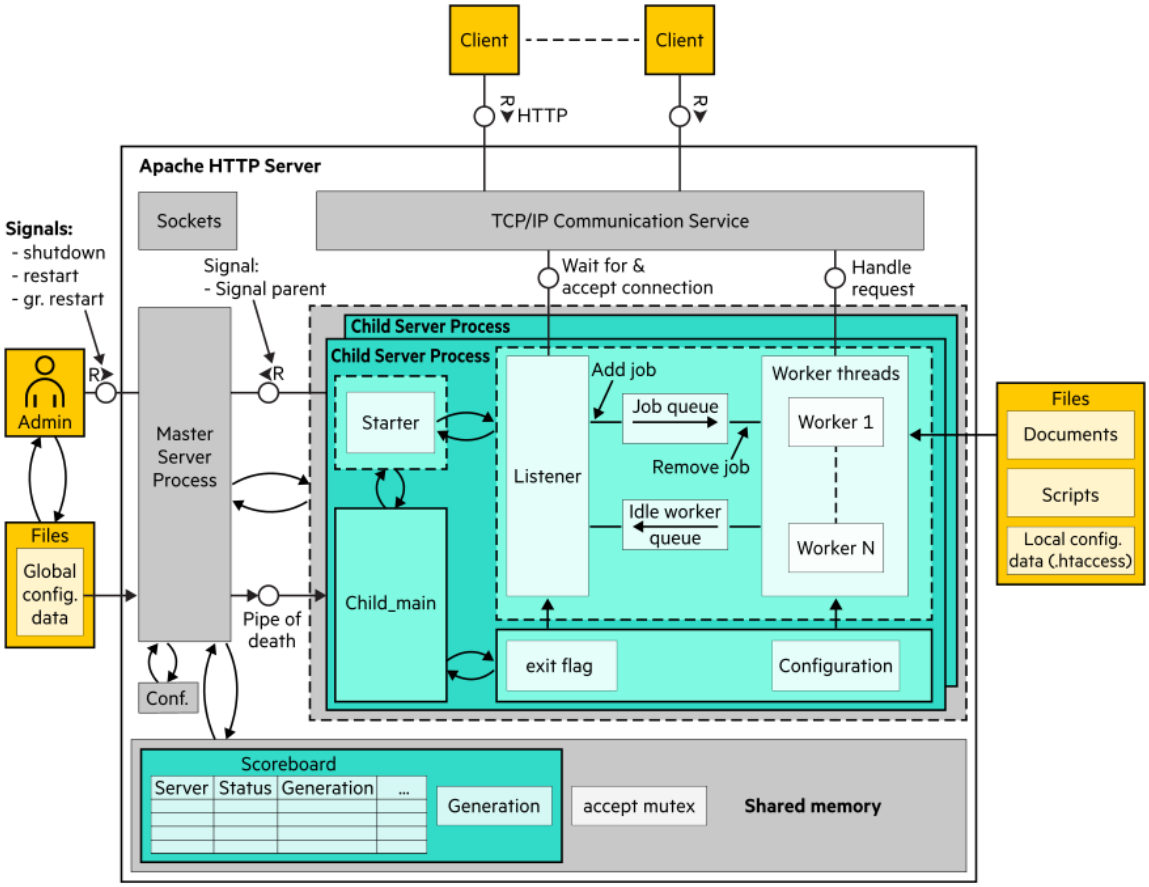
škálovateľnosť a efektívnosť využitia zdrojov v porovnaní s tradičnými MPM, ako sú *prefork* alebo *worker*. Táto zmena v architektúre zlepšuje podporu pre stránky s vysokou návštevnosťou a zabraňuje tomu, aby neaktívne pripojenia (napríklad tie, ktoré vyžaduje funkcia *keep-alive*), zbytočne blokovali cenné serverové vlákna [5] , [6].

Konfiguračný systém Apache je založený na direktívach a podporuje hierarchické vkladanie súborov (napr. *httpd.conf*, *sites-enabled*, *.htaccess*). To ponúka detailnú kontrolu nad správaním servera a zjednodušuje správu zložitých nasadení.

Kľúčové vlastnosti architektúry Apache 2.4 zahŕňajú:

- Podpora dynamických zdieľaných objektov (*Dynamic Shared Object*, DSO), ktorá umožňuje načítanie modulov za behu.
- Input-Output filtre na spracovanie dát pri ich prechode cez serverový zásobník.
- Háčiky a API, ktoré moduly používajú na interakciu s jadrom servera a navzájom medzi sebou.
- Robustný systém spracovania požiadaviek, ktorý podporuje autentifikáciu, prepisovanie URL, proxyovanie a ďalšie.

Tento modulárny a flexibilný dizajn robí z Apache 2.4 výkonnú platformu na hosťovanie širokej škály webových služieb, prispôsobiteľnú tradičným IT prostrediam aj moderným cloudovým infraštruktúram [7].



Obrázok č.1: Apache HTTP Server architecture [8].

Tento diagram načrtáva vnútornú architektúru Apache HTTP Servera, konkrétne jeho spracovanie HTTP požiadaviek založené na procesoch s využitím modulárneho a viacprocesorového dizajnu.

Pri spustení hlavný proces servera (*Master Server Process*) načíta globálne konfiguračné súbory a vytvorí viacero podprocesov servera (*Child Server Processes*). Tieto podprocesy sú zodpovedné za spracovanie prichádzajúcich požiadaviek klientov. Každý podproces obsahuje poslucháča (*Listener*), ktorý čaká na pripojenia cez TCP/IP sockety a prijaté požiadavky odovzdáva do frontu úloh (*Job Queue*).

Požiadavky spracovávajú pracovné vlákna (*Worker 1* až *Worker N*), ktoré preberajú úlohy z frontu úloh a po dokončení sa vracajú do frontu nečinných vlákien (*Idle Worker Queue*). Pracovné vlákna sú riadené v rámci životného cyklu podprocesu, ktorý koordinuje funkcia *Child_main*.

Komunikácia a sledovanie stavu medzi procesmi prebieha cez zdieľanú pamäť, najmä prostredníctvom *Scoreboardu*, ktorý zaznamenáva metriky, ako sú stav pripojenia, stav pracovných vlákien a sledovanie generácií. To pomáha serveru monitorovať výkon a zdravie pracovných vlákien.

Administrátor môže posielať riadiace signály (napr. reštart, plynulý reštart, vypnutie) hlavnému procesu, ktorý ich podľa potreby rozšíri na podprocesy. Existuje aj riadiaci mechanizmus nazývaný „*Pipe of Death*“, ktorý slúži na čisté ukončenie podprocesov.

Konfigurácia je hierarchická, umožňuje globálnu konfiguráciu servera aj prepisy na úrovni adresárov cez súbory *.htaccess*. Požiadavky sú nakoniec obsluhované z dokumentových koreňov alebo skriptov na základe konfigurácie servera a požadovaného URI.

Táto architektúra podporuje moduly pre viacprocesorové spracovanie (*Multi-Processing Modules*, MPM), umožňujúce flexibilné modely súbežnosti (*prefork*, *worker*, *event*), a je optimalizovaná pre škálovateľnosť a výkon v malých aj vysoko zaťažených prostrediach.

Bežné bezpečnostné problémy webového servera

Bezpečnosť webového servera závisí nielen od samotného softvéru, ale aj od spôsobu jeho konfigurácie, údržby a ochrany hostovaných aplikácií. Táto časť sa zameriava na najčastejšie bezpečnostné problémy webového servera Apache, ktoré môžu viesť k vážnym bezpečnostným incidentom, ak nie sú včas identifikované a riešené [9].

Zastaraný softvér a neopravené chyby zabezpečenia
Spustenie zastaraných verzií Apache alebo jeho modulov vystavuje systémy známym bezpečnostným zraniteľnostiam. Útočníci často vyhľadávajú systémy, ktoré neaplikovali kritické opravy, čo môže viesť k vzdialenému spusteniu kódu, eskalácii privilégií alebo odmietnutiu služby. Pravidelná aktualizácia Apache a všetkých závislostí je kľúčovou súčasťou udržiavania bezpečného webového servera.

Prechádzanie adresárov, sprístupnenie informácií a nezabezpečené predvolené nastavenia
Nesprávne konfigurácie alebo spoliehanie sa na predvolené nastavenia môžu útočníkom umožniť zneužiť chyby pri prechode cez adresár (napr. použitie ../ na prístup k neoprávneným súborom), odhaliť informácie o serveri alebo odhaliť citlivé údaje, ako sú konfiguračné súbory alebo zdrojový kód. Vypnutie nepotrebných funkcií, dezinfekcia vstupov a obmedzenie prístupu k citlivým adresárom sú nevyhnutné na zmiernenie týchto rizík.

Slabé kontroly prístupu a autentifikačné mechanizmy
Nedostatočné alebo nesprávne nakonfigurované kontroly prístupu môžu umožniť neoprávneným používateľom prístup k chráneným zdrojom. Slabé autentifikačné mechanizmy – ako sú heslá s čistým textom, predvídateľné poverenia alebo chýbajúca viacfaktorová autentifikácia – riziko ešte zvyšujú. Implementácia zásad silných hesiel, používanie bezpečných metód autentifikácie a správne nastavenie príkazov Require pomáhajú vynútiť prísnejšiu kontrolu prístupu.

Nezabezpečená komunikácia (nedostatok HTTPS alebo nesprávne verzie TLS)
Prenos údajov cez HTTP ich vystavuje odpočúvaniu a manipulácii. Nepoužívanie HTTPS alebo spoliehanie sa na zastarané verzie TLS (napr. TLS 1.0 alebo 1.1) oslabuje dôvernosť a integritu komunikácie. Presadzovanie HTTPS na celom webe, presmerovanie všetkej návštevnosti HTTP a konfigurácia Apache na používanie moderných nastavení TLS (TLS 1.2 alebo 1.3) so silnými šiframi je pre zabezpečenú komunikáciu rozhodujúca.

Chyby v hosťovaných aplikáciách (SQLi, XSS, CSRF)
Apache často hostí webové aplikácie, ktoré môžu predstavovať svoje vlastné zraniteľnosti. Medzi bežné problémy patrí SQL injection (SQLi), Cross-Site Scripting (XSS) a Cross-Site Request Forgery (CSRF), ktoré môžu útočníci zneužiť na krádež údajov, odcudzenie identity používateľov alebo manipuláciu so systémami typu backend. Na ochranu pred týmito hrozbami sú nevyhnutné pravidelné kontroly kódu, overovanie vstupu, hlavičky zabezpečenia a udržiavanie aktuálnych aplikácií.

Bežné nesprávne konfigurácie v Apache 2.4

Nesprávne konfigurácie v Apache 2.4 môžu viesť k bezpečnostným chybám a problémom s výkonom [10] [11]. Niektoré z najbežnejších zahŕňajú:

Directory Listing Enabled: Ak je nastavená možnosť Indexy možností, Apache môže odhaliť obsah adresárov, čo používateľom umožní prehliadať citlivé súbory.

Slabé nastavenia AllowOverride: Príliš tolerantné AllowOverride All môže povoliť súborom .htaccess prepísať kritické nastavenia zabezpečenia, čo môže viesť k zraniteľnostiam.

Nezabezpečené alebo nepotrebné moduly: Povolenie nepoužívaných alebo zastaraných modulov (napr. mod_status, mod_info) bez riadnej kontroly prístupu môže spôsobiť únik podrobností o serveri alebo jeho zneužitie.

Nástroje na skenovanie a hodnotenie bezpečnosti

Penetračné testovanie webového servera zahŕňa simuláciu skutočných útokov na identifikáciu zraniteľností v jeho konfigurácii, softvéri a hostovaných aplikáciách. Cieľom je odhaliť bezpečnostné chyby, ako sú nesprávne konfigurácie, zastarané služby, slabé overenie alebo odhalené súbory skôr, ako to urobia zlomyseľní aktéri [12].

K dispozícii je niekoľko automatizovaných nástrojov, ktoré pomáhajú výrazne urýchliť proces testovania a bežne sa používajú pri manuálnom aj nepretržitom testovaní bezpečnosti.

Nikto

Nikto je klasický nástroj na rýchle skenovanie webových serverov na bežné problémy. Kontroluje zastarané verzie softvéru, odhalené konfiguračné súbory, predvolené skripty a iné nenáročné ovocie. Je to skvelé na zachytenie vecí, ako je povolený zoznam adresárov, zabudnuté záložné súbory alebo nesprávne nakonfigurované metódy HTTP, ktoré by mali byť zakázané.

Nástroj funguje tak, že odosiela vytvorené požiadavky do cieľa a porovnáva odpovede s veľkou databázou známych problémov. Je to dosť rýchle a môže byť užitočné pre rutinné skenovanie alebo skoré štádiá penetračného testu.

To znamená, že to nie je kradmé - Nikto je veľmi hlasný a ľahko ho hlásia bezpečnostné systémy. Má tiež tendenciu vytvárať falošné pozitíva a netestuje aplikačnú logiku, takže nenahrádza hlbšie testovanie. Napriek tomu je to solídny nástroj na rýchle zistenie zjavných chýb na webových serveroch [13].

Nmap

Nmap je často vnímaný ako sieťový mapovač – ale je tiež neuveriteľne užitočný na prieskum na úrovni servera. Keď skenujete server, Nmap vám rýchlo povie, ktoré porty sú otvorené, aké služby sú spustené a aké verzie tieto služby používajú. To samo o sebe môže veľa odhaliť (zastaranú verziu Apache, odhalenú službu SSH alebo otvorený panel správcu).

Pomocou skriptovacieho nástroja Nmap (NSE) môžete ísť ešte hlbšie. Existujú skripty na testovanie známych zraniteľností, nesprávnych konfigurácií, slabých stránok SSL/TLS a ďalších. Môžete ho použiť napríklad na:

- Skontroluje slabé šifry SSL na webovom serveri
- Zistití, či server stále používa protokol SMBv1
- Identifikuje zraniteľné verzie Apache alebo MySQL

Nmap vám poskytuje vysokoúrovňový pohľad na exponovaný povrch vášho servera a môže fungovať ako systém včasného varovania pre nesprávne nakonfigurované alebo zabudnuté služby. Je to užitočné najmä pri nastavovaní nového servera alebo posilňovaní existujúcej infraštruktúry [13] [14].

OWASP ZAP

OWASP ZAP je jedným z najlepších bezplatných nástrojov na testovanie webových aplikácií. Dokáže automaticky vyhľadávať bežné zraniteľnosti, ako je XSS, SQL injection, nezabezpečené súbory cookie a nefunkčná autentifikácia. Má však aj manuálny režim s nástrojmi, ako sú editory požiadaviek / odpovedí, fuzzery a proxy na zachytenie návštevnosti – veľmi užitočné na vlastné testovanie.

ZAP môžete spustiť v plne automatizovanom režime (skvelé pre vývojárov alebo CI/CD), alebo môžete prejsť na praktické hlbšie manuálne testovanie. Je to užitočné najmä vtedy, keď potrebujete otestovať, ako sa aplikácia správa pri autentifikácii, alebo keď máte čo do činenia s rozhraniami s vysokým obsahom JavaScriptu.

Tiež sa veľmi dobre integruje do kanálov CI/CD pomocou Docker alebo API. S každou novou požiadavkou na zostavenie alebo stiahnutie môžete spustiť skenovanie, čo pomôže vývojárom zachytiť problémy na začiatku vývojového cyklu [15], [16].

Open VAS

OpenVAS je ťažší skener zraniteľnosti – je navrhnutý tak, aby vám poskytol úplný prehľad známych problémov na základe CVE. Je to pomalšie ako nástroje ako Nikto alebo Nmap, ale oveľa komplexnejšie. Pripája sa k pravidelne aktualizovanému zdroju zraniteľností a prehľadáva vaše systémy, či neobsahujú softvérové chyby, zastarané balíky a nezabezpečené konfigurácie.

To, čo robí OpenVAS obzvlášť užitočným, je jeho podrobný reporting. Nielenže vám povie, že niečo nie je v poriadku – často priamo odkazuje na záznamy CVE, poskytuje kroky na nápravu a poskytuje skóre závažnosti. Dokáže vykonávať neoverené aj overené kontroly, čo znamená, že môže preniknúť do vašich systémov hlbšie, ak poskytnete platné poverenia.

Výborne zapadá aj do väčšieho procesu riadenia zraniteľnosti. Môžete naplánovať opakujúce sa kontroly, sledovať problémy v priebehu času a integrovať ich s inými nástrojmi, ako sú SIEM alebo systémy predaja lístkov [17].

Acunetix

Acunetix je komerčný skener zraniteľnosti webu, ktorý sa zameriava na hľadanie problémov, ako je SQL injection, XSS, nesprávne nakonfigurované hlavičky a chyby v autentifikácii. Dobre si poradí s modernými webovými aplikáciami, vrátane tých, ktoré sú vytvorené s rámcami JavaScriptu, ako sú React alebo Angular.

Podporuje overené skenovanie, testovanie API a môže sa integrovať do kanálov CI/CD pre automatizované bezpečnostné kontroly. Acunetix je rýchly, presný a poskytuje jasné správy, čo z neho robí solídnu voľbu pre bezpečnostné tímy aj vývojárov.

Aj keď je výkonný, je to platený nástroj a stále môže produkovať nejaké falošné pozitíva – preto sa po skenovaní odporúča manuálna kontrola [18].

Ciele projektu

Hlavným cieľom tohto projektu je analyzovať a implementovať bezpečnostné opatrenia pre webový server s cieľom minimalizovať riziká spojené s jeho prevádzkou a zvýšiť jeho odolnosť voči kybernetickým hrozbám. Projekt je zameraný na webový server Apache2, ktorý patrí medzi najrozšírenejšie open-source HTTP servery, a preto je častým cieľom útokov. V rámci projektu sa budú vykonávať aktivity spojené so skúmaním architektúry servera, identifikáciou zraniteľností, nastavovaním bezpečnostných mechanizmov a testovaním ich efektivity v reálnom prostredí.

Cieľ 1 - Preskúmať architektúru Apache2 a jeho natívne bezpečnostné mechanizmy. Táto časť sa zameria na pochopenie vnútornej štruktúry servera, jeho modulárneho dizajnu, spôsobu spracovania požiadaviek a dostupných možností konfigurácie z hľadiska bezpečnosti.

Cieľ 2 - Identifikovať bežné konfiguračné chyby a zraniteľnosti, ktoré sa vyskytujú pri prevádzke webových serverov. Na tento účel budú využité automatizované nástroje ako napr. Nikto, Nmap alebo OWASP ZAP ako aj manuálne penetračné testy, ktoré poskytnú detailnejší pohľad na potenciálne slabé miesta. Hlavné aspekty z pohľadu testovacieho softvéru, ktoré môžu byť zaujímavé pri výbere nástrojov, zahŕňajú popularitu (Nikto), možnosť prispôsobenia (Nmap) a kvalitné používateľské rozhranie alebo grafické prostredie (OWASP ZAP).

Cieľ 3 - Na základe identifikovaných nedostatkov sa budú konfigurovať a implementovať konkrétne bezpečnostné opatrenia, vrátane úpravy konfiguračných súborov, obmedzenia prístupu, aktivácie bezpečnostných modulov a ďalších rozširujúcich komponentov, ktoré posilnia ochranu servera.

Cieľ 4 - Automatizovať opravu bezpečnostných slabín zistených na základe bezpečnostnej kontroly pomocou skriptu. Cieľom je aby sa podarilo v jednej skripte spustiť aj scan aj opravenie slabín podľa scanu (parsovanie bezpečnostných slabín).

Cieľ 5 - V poslednej fáze sa uskutoční testovanie implementovaných opatrení, pričom bude hodnotená ich účinnosť pri odvracaní typických útokov. Výsledky budú analyzované a interpretované, s dôrazom na praktické odporúčania a možnosti ďalšieho zlepšenia bezpečnostnej úrovne webového servera.

Metodológia

1. **Analýza literatúry a dostupných zdrojov:**

   Cieľom tejto časti je teoreticky analyzovať fungovanie webového servera Apache 2.4, jeho architektúru, známe bezpečnostné slabiny a dostupné nástroje na ich detekciu. Táto analýza vytvára základ pre praktickú časť projektu, v ktorej sa budú aplikovať bezpečnostné odporúčania.

2. **Konfigurácia a nasadenie serverov**: Inštalácia Apache2 prostredie.
3. **Identifikácia zraniteľností**: Skúmanie bezpečnostných hrozieb pomocou automatizovaných nástrojov:
   a. Nikto – To je najčastejšie používaný nástroj na testovanie webových serverov, preto bude vyskúšaný, ako sa dá používať v operačnom systéme Windows.
   b. Nmap – To je dobre prispôsobiteľný nástroj, umožňuje vytvárať vlastné testovacie skripty (.nse), preto bude vyskúšaný na tvorbu vlastných testovacích skriptov pre špecifické scenáre.
   c. OWASP ZAP – Tento nástroj má dobré používateľské rozhranie, zaujímavé riešenie na testovanie webových serverov.
4. **Automatizácia procesu skenovania a opráv (skriptovanie):**
   V rámci projektu bude vytvorený **PowerShell skript**, ktorý:
   A. Spustí naplánované skenovanie (napr. Nmap)

   B. Vyextrahuje zraniteľnosti z výstupných súborov (napr. .txt logy)

   C. Automaticky aplikuje preddefinované **bezpečnostné opravy** podľa toho, aké bezpečnosnté slabiny boli nájdené, alebo odporúčané zmeny v konfigurácii.

5. **Testovanie bezpečnosti:**

   Po implementácii opatrení budú vykonané **penetračné testy**, ktoré overia ich účinnosť. Testovanie bude pokrývať bežné typy útokov (XSS, SQLi, LFI, CSRF), slabiny v konfigurácii servera, a overenie odolnosti voči neautorizovanému prístupu.

6. **Vyhodnotenie a dokumentácia**: Spracovanie zistení, tvorba odporúčanií.

Experimenty

Experimenty je možné rozdeliť do dvoch hlavných skupín s dvoma odlišnými výstupmi. Prvá skupina experimentov sa zameriava na odhaľovanie bezpečnostných rizík, pričom testovanie bude prebiehať pomocou rôznych nástrojov s cieľom čo najpresnejšie identifikovať bezpečnostné hrozby, pričom sa zohľadnia aj ďalšie relevantné aspekty. Výstupom tohto experimentu bude zoznam bezpečnostných rizík.

Druhá fáza experimentov sa zameriava na odstránenie zistených rizík a posilnenie bezpečnosti servera. V tejto fáze bude vykonaný pokus o automatizované zabezpečenie, ako aj o manuálne odstránenie bezpečnostných hrozieb.

Experiment 1 - Kontrola zraniteľnosti

Pri testovaní bezpečnosti webových serverov máme k dispozícii rôzne nástroje. V tomto experimente je preskúmaných niekoľko rôznych nástrojov, ktoré môžu byť zaujímavé z rôznych hľadísk, ako je popularita, prispôsobiteľnosť alebo používateľské rozhranie. V experimente je použitý Nikto pre svoju popularitu, Nmap pre svoju prispôsobiteľnosť a OWASP ZAP vďaka svojmu prehľadnému používateľskému rozhraniu. Napriek tomu je najdôležitejším parametrom to, ako dobre sú tieto nástroje použiteľné na odhaľovanie bezpečnostných slabín. Na záver získame zoznam zraniteľností a bezpečnostných rizík nakonfigurovaného webového servera Apache 2.4.

Nikto

V prípade tohto nástroja ide o populárny skener zraniteľností, lebo skenuje viac ako 6700 zranitelností. Je dobre použiteľný v operačnom systéme Linux. Avšak v tomto experimente som narazil na problém pri používaní vo Windows prostredí, keďže sa mi nepodarilo zistiť bezpečnostné riziká na mnou nakonfigurovanom Apache webovom serveri bežiacom na localhoste (127.0.0.1).

```
PS C:\nikto\program> perl nikto.pl -h http://127.0.0.1:8080
- Nikto v2.5.0
---------------------------------------------------------------------------
---------------------------------------------------------------------------
+ Unable to connect to 127.0.0.1:8080.
+ 0 host(s) tested
```

Obrázok č.2: Beh keď Nikto nenašiel bežiaci server na localhost a nevedel skenovať.

```
PS C:\nikto\program> curl http://127.0.0.1:8080


StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE html>
```

Obrázok č.3: Ukážka, že server bol dostupný.

Obrázok č.4: Ukážka, že server počúval na porte 8080.

Napriek tomu, že server bol na localhoste dostupný na správnom porte, nástroj nebol schopný sa k nemu pripojiť a otestovať bezpečnostné riziká. V prípade servera, ktorý nebežal lokálne, ako napríklad webová stránka google.com sa však nástroj dokázal pripojiť a spustiť testovací proces.

Obrázok č.5: Ukážka, že Nikto našiel a analizoval google.com.

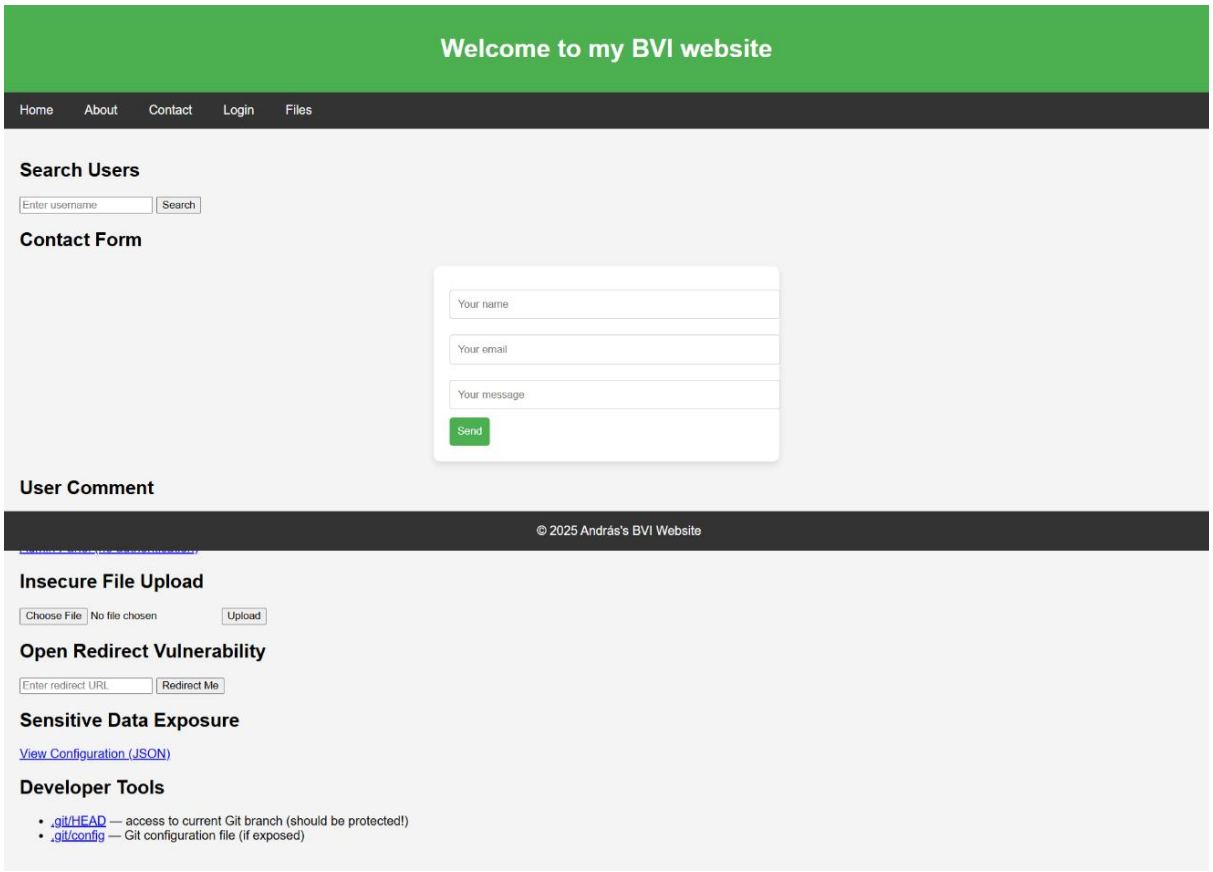
Experimentovanie pokračovalo hľadaním chyby, no rovnaký problém pretrvával aj pri použití rôznych verzií Perl a Nikto. Počas skúmania som musel dospieť k záveru, že pravdepodobne existuje určitá chyba vo verziách pre Windows pri testovaní lokálne bežiacich webových serverov.

Nmap

V prípade tohto nástroja ide o dobre prispôsobiteľný skener bezpečnostných rizík. Pomocou Nmapu bol vykonaný cielený sken s využitím bash skriptu, ktorý v rámci jedného skriptu testuje súbor špecifických zraniteľností, ktoré som vopred definoval. Tieto skeny využívajú konkrétne .nse skripty, ktoré obsahujú preddefinovaný kód na detekciu bezpečnostných rizík. V ďalšom kroku som vytvoril vlastný .nse skript, pomocou ktorého je skener schopný detegovať nezabezpečený prístup k zdrojovým kódom testovaním dostupnosti git súborov. Nakoniec bol spustený úplný Nmap sken, ktorý skúma všetky možné scenáre a vykonáva komplexnú analýzu bezpečnosti webového servera.

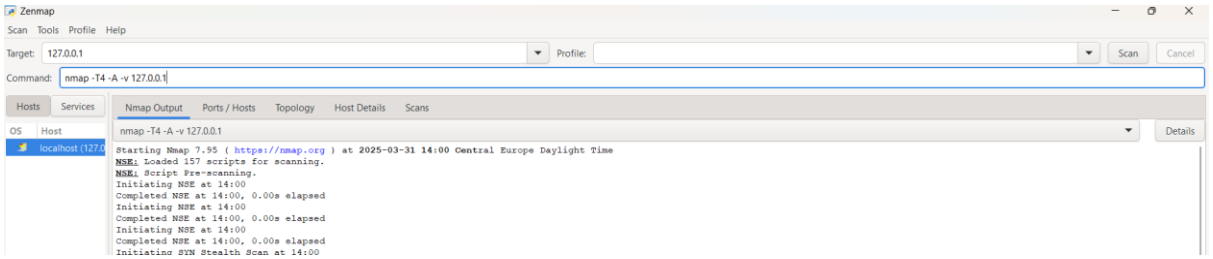
Pre tieto experimenty som implementoval jednoduchú webovú stránku na demonštráciu bezpečnostných rizík. Implementácia, bezpečnostné skenovania, skripty a riešenia sú dostupné vo verejnom GitHub repozitáriu uvedenom v citáciách [19].

Základná webová stránka:



Obrázok č.6: Vytvorená základná webová stránka na demonštráciu bezpečnostných slabín.

Používateľské rozhranie, je dobre používatelné ale nie je veľmi interaktívna a intuitívna.



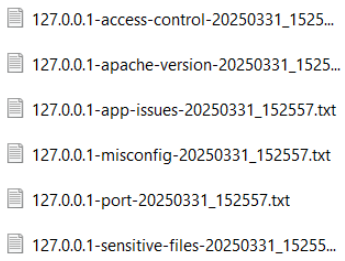
Obrázok č.7: Používateľské rozhranie Nikto.

Vytvorený script ***apache_nmap_scan.sh*** kontroluje určité prípady, určité možné bezpečnostné riziká spustením konkrétnych príkazov Nmap.

```bash
#!/bin/bash

# target is provided?
if [ -z "$1" ]; then
    echo "Usage: $0 <apache-server-ip>"
    exit 1
fi

TARGET=$1
PORT=8080  # Apache is running on 127.0.0.1:8080
OUTPUT_DIR="apache_scan_results"
mkdir -p $OUTPUT_DIR
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")

# checking Apache port
echo "[+] Checking if Apache is running on $TARGET:$PORT..."
nmap -p $PORT --open -oN "$OUTPUT_DIR/$TARGET-port-$TIMESTAMP.txt" $TARGET

# Apache version check and Outdated Software
echo "[+] Checking for outdated Apache versions..."
nmap -p $PORT -sV --script=http-server-header -oN "$OUTPUT_DIR/$TARGET-apache-version-$TIMESTAMP.txt" $TARGET

# security Misconfigurations (Headers & SSL Issues)
echo "[+] Checking for security misconfigurations..."
nmap -p $PORT --script http-security-headers,http-headers -oN "$OUTPUT_DIR/$TARGET-misconfig-$TIMESTAMP.txt" $TARGET

# access Control Issues (Weak Authentication & Methods)
echo "[+] Checking for access control vulnerabilities..."
nmap -p $PORT --script http-auth-finder,http-methods -oN "$OUTPUT_DIR/$TARGET-access-control-$TIMESTAMP.txt" $TARGET

# web Application Vulnerabilities (XSS, SQL Injection, LFI)
echo "[+] Scanning for web application vulnerabilities..."
nmap -p $PORT --script http-sql-injection,http-xssed,http-phpself-xss,http-lfi-spider -oN "$OUTPUT_DIR/$TARGET-app-issues-$TIMESTAMP.txt" $TARGET

# sensitive Files (Backup Files, Robots.txt)
echo "[+] Scanning for sensitive files..."
nmap -p $PORT --script http-config-backup,http-robots.txt,http-enum -oN "$OUTPUT_DIR/$TARGET-sensitive-files-$TIMESTAMP.txt" $TARGET

echo "Apache Security Scan completed. Results saved in $OUTPUT_DIR/"
```

Obrázok č.8: Bash skript na skenovanie špecifických problémov.

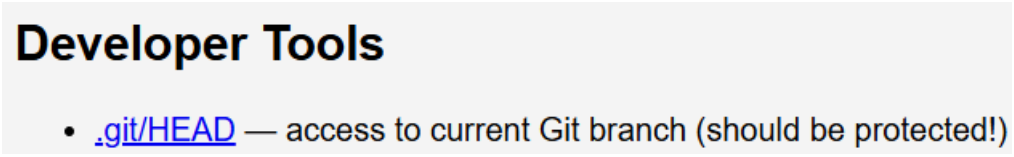
Vo výstupe sa nachádza priečinok, ktorý obsahuje výstupné logy jednotlivých Nmap príkazov. Na základe týchto logov je možné identifikovať zistené bezpečnostné riziká.



Obrázok č.9: Bash script výstupné reporty.

V nasledujúcom kroku bude predstavená definícia vlastného .nse skriptu a jeho fungovanie. Pridaním skriptu ***http-check-gitdir.nse*** [19] k existujúcim vstavaným .nse skriptom dosiahneme rozšírenie skenovacích schopností. Novo dostupná kontrolná jednotka zisťuje, či je dostupný nejaký *.git* adresár, prostredníctvom ktorého by mohlo dôjsť k ohrozeniu zdrojových kódov.

Dotknuté bezpečnostné riziko na webovej stránke:



Obrázok č.10: Problémová časť na webovej stránke.

```
PS C:\Program Files (x86)\Nmap\scripts> nmap -p 8080 --script ./http-check-gitdir.nse 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-31 17:43 Central Europe Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0010s latency).

PORT     STATE SERVICE
8080/tcp open  http-proxy
| http-check-gitdir: [VULNERABLE] Exposed .git directory detected!
| URL: /.git/HEAD
|_Response: ref: refs/heads/main

Nmap done: 1 IP address (1 host up) scanned in 0.42 seconds
```

Obrázok č.11: Úspešné detekovanie zranitelnosti s vlastným .nse scriptom.

Nakoniec je viditeľný výsledok spustenia takého Nmap príkazu, ktorý využíva všetky dostupné .nse skripty, vrátane novo definovaného skriptu, ktorý kontroluje dostupnosť *.git* zdrojov. Ide o oveľa dlhšie trvajúci sken. V tomto prípade trval 2 hodiny a 17 minút, takže nie je taký efektívny ako bash skript zameraný na kontrolu špecifických oblastí (za 1-2 sekúnd), no poskytuje oveľa podrobnejšie výsledky a odhaľuje nielen vopred určené, ale všetky bezpečnostné riziká.

```
C:\Users\nagya>nmap -p 8080 --script http-* -T4 -A -v 127.0.0.1
```

Obrázok č.12: Príkaz na úplný Nmap sken.

Výstup je zoznam nájdených rizík ***Nmap_Vulnerability_report.csv***:

#	Issue	Severity	Description	Fix
1	Slowloris DoS (CVE-2007-6750)	High	Apache vulnerable to Slowloris	Reverse proxy or mod_reqtimeout
2	TRACE method enabled	Medium	Can lead to XST attacks	TraceEnable off
3	Proxy may be open	Medium	Proxy redirecting requests	Secure/disable mod_proxy
4	CSRF on multiple forms	Low	Forms lack tokens	Implement CSRF protection
5	Server banner disclosure	Low	Apache version exposed	Use ServerTokens Prod
6	Missing security headers	Low	XFO, CSP, HSTS missing	Use mod_headers
7	Broken/missing endpoints	Info	Many 404s on key paths	Clean up or implement
8	Accepts known bots	Info	Allows scraper UAs	Filter or rate-limit
9	Public .git/ folder exposed	Critical	Source code leakage	Remove or deny access

Tabuľka č.1: Zoznam detekovaných bezpečnostných zraniteľností.

1. **Slowloris DoS (CVE-2007-6750)**
   Útok typu DoS (Denial of Service), ktorý vyčerpáva dostupné spojenia webového servera tým, že ich zámerne udržiava otvorené čo najdlhšie pomocou neúplných požiadaviek. Apache bez ochranných opatrení je na tento typ útoku zraniteľný.
   Riešenie: Použiť reverzný proxy server alebo modul mod_reqtimeout na obmedzenie trvania požiadaviek.

2. **TRACE method enabled**
   Povolená HTTP metóda TRACE môže byť zneužitá pri XST (Cross Site Tracing) útokoch, ktoré kombinujú túto funkciu s inými zraniteľnosťami (napr. XSS) na získanie citlivých údajov ako cookies.
   Riešenie: Zakázať metódu pomocou TraceEnable off.

3. **Proxy may be open**
   Server môže umožňovať požiadavky cez mod_proxy, čo môže útočník využiť na anonymné pripojenie, útoky alebo prístup k interným sieťam.
   Riešenie: Zakázať alebo zabezpečiť mod_proxy modul.

4. **CSRF on multiple forms**
   Webové formuláre nemajú ochranné CSRF tokeny, čo umožňuje útočníkovi vykonať akciu v mene autentifikovaného používateľa bez jeho vedomia.
   Riešenie: Zaviesť ochranu proti CSRF pridaním unikátnych tokenov do formulárov.

5. **Server banner disclosure**
   Server odhaľuje svoju verziu (napr. Apache/2.4.x), čo poskytuje útočníkovi informácie na cielený útok.
   Riešenie: Nastaviť ServerTokens Prod, čím sa skrýva verzia a znižuje sa informačná stopa.

6. **Missing security headers**
   Chýbajú hlavičky ako X-Frame-Options (XFO), Content-Security-Policy (CSP) alebo HTTP Strict Transport Security (HSTS), ktoré sú kľúčové pre obranu proti rôznym webovým útokom (XSS, clickjacking, atď.).
   Riešenie: Pridať tieto hlavičky cez mod_headers.

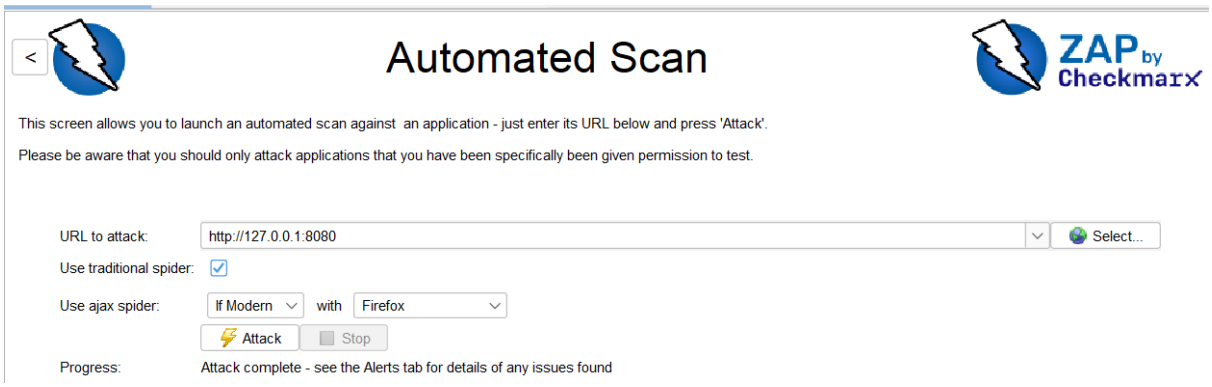
7. **Broken/missing endpoints**
   Mnohé požiadavky končia chybou 404, čo môže znamenať buď chýbajúce funkcie, alebo zlé smerovanie. Zároveň môžu útočníci takto získať informácie o vnútornej štruktúre webu.
   Riešenie: Opraviť alebo odstrániť neexistujúce cesty.

8. **Accepts known bots**
   Server nefiltruje známe boty (napr. scraping nástroje), čo umožňuje automatizované zbieranie dát z webu.
   Riešenie: Nastaviť filtre, blokovanie alebo rate-limity na základe user-agentov.

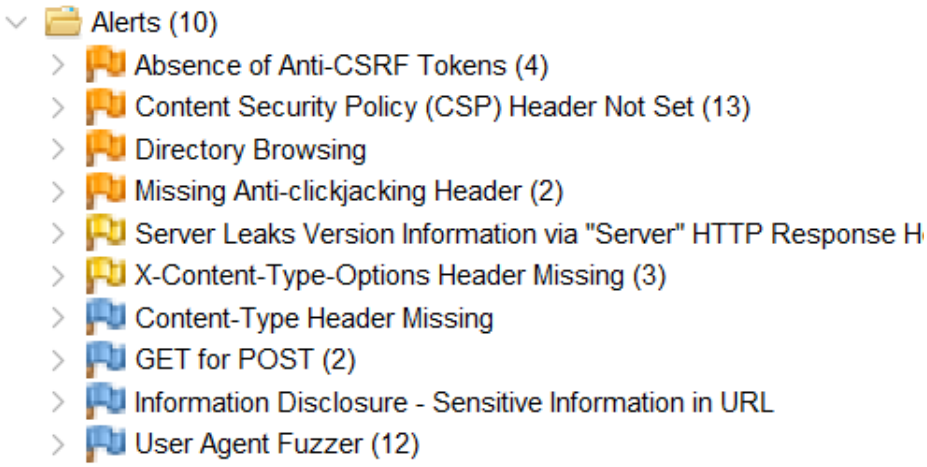
9. **Public .git folder exposed**
   Kritická zraniteľnosť – ak je .git priečinok verejne dostupný, útočník môže získať celý zdrojový kód vrátane citlivých informácií.
   Riešenie: Prístup k .git adresáru by mal byť zakázaný alebo priečinok úplne odstránený z verejnej časti servera.

OWASP ZAP

V tomto prípade sa stretávame s používateľsky prívetivejším rozhraním, kde sú rôzne funkcie usporiadané do intuitívneho interaktívneho grafického rozhrania (GUI). Podobne ako pri Nmap, aj tu je potrebné spúšťať skenovacie príkazy v závislosti od toho, čo a v akom rozsahu chceme kontrolovať. Následne softvér poskytne štruktúrovanú vizualizáciu, v ktorej informuje používateľa o zistených bezpečnostných rizikách a označuje ich rôznymi spôsobmi podľa ich závažnosti. Boli dosiahnuté podobné výsledky ako v prípade Nmap, avšak dodatočné kritérium, ktoré bolo pridané do konfigurácie Nmapu, v tomto prípade detegované nebolo.



Obrázok č.13: Používateľské rozhranie OWASP ZAP.



Obrázok č.14: Výstup OWASP ZAP.

Zhrnutie

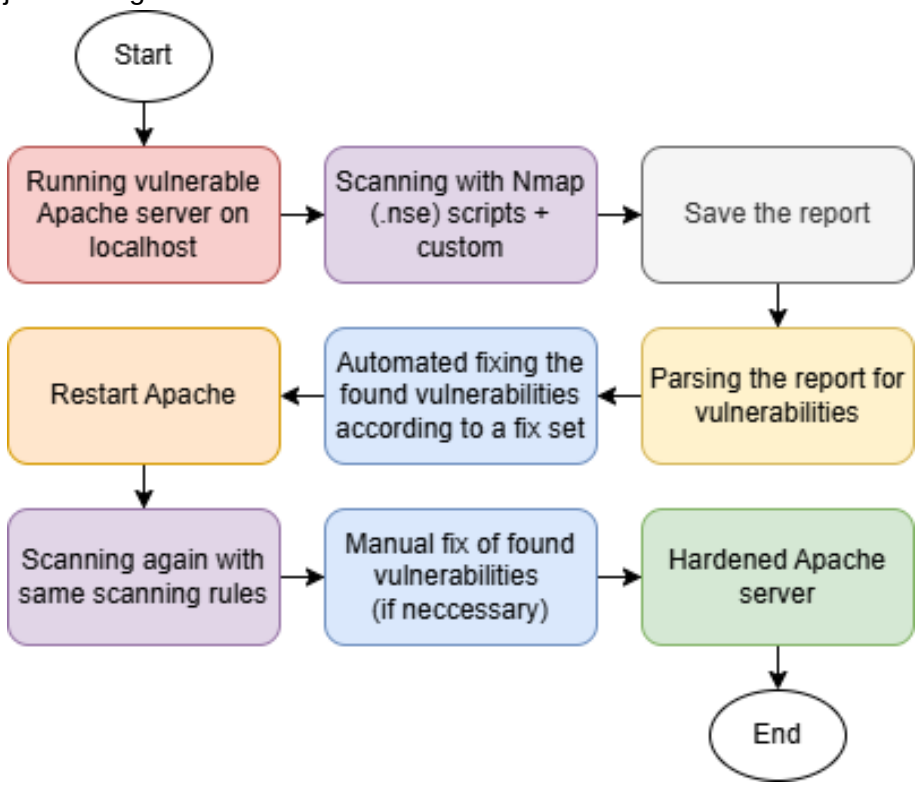
V experimente boli testované tri rôzne nástroje na detekciu zraniteľností webových serverov Nikto, Nmap a OWASP ZAP. Každý z nich bol vybraný na základe iného kritéria: Nikto pre svoju popularitu, Nmap pre prispôsobiteľnosť a OWASP ZAP pre používateľsky prívetivé rozhranie. Nikto sa ukázal ako menej spoľahlivý v prostredí Windows, kde nevedel otestovať lokálny server, zatiaľ čo externý web fungoval. Nmap umožnil cielené aj úplné skenovanie

pomocou .nse skriptov vrátane vlastného skriptu na detekciu nezabezpečených git súborov. Hoci úplný sken trval vyše dvoch hodín, poskytol najpodrobnejšie výsledky. OWASP ZAP ponúkol jednoduchšie ovládanie a podobné výsledky ako Nmap, no vlastný skript na kontrolu .git adresárov nedokázal použiť. Výsledkom experimentu bol zoznam zistených bezpečnostných rizík nakonfigurovaného Apache 2.4 servera.

Experiment 2 - Fixovanie zranitelnosti

V druhej fáze experimentovania je predstavený automatizovaný skript na testovanie a opravu zraniteľností, avšak s ohľadom na to, že nie všetky bezpečnostné anomálie je možné opraviť automatizovane, v určitých prípadoch je potrebný manuálny zásah.

Najprv sa spustí úplný Nmap sken na lokálne bežiacom webovom serveri Apache 2.4, ktorý identifikuje a uloží logy, čím vytvorí report o zistených bezpečnostných rizikách. Z tohto reportu sa automaticky vyparsujú konkrétne bezpečnostné hrozby – na základe detailov sa určí ich typ – a z rozsiahleho a zložitého výstupu vznikne zoznam obsahujúci iba problémy a ich špecifikácie. Na základe tohto zoznamu skript automaticky aplikuje potrebné úpravy do konfiguračných súborov podľa vopred definovanej množiny riešení a následne reštartuje server. Týmto krokom sa končí automatizovaná časť procesu.
V ďalšom kroku prebehne nový bezpečnostný sken, ktorý slúži na overenie, či boli problémy skutočne odstránené. V prípade, že niektoré riziká síce boli detegované, no automatizácia ich nedokázala správne alebo vôbec opraviť, je potrebný manuálny zásah. Výsledkom celého procesu je robustný a bezpečný webový server. Tento postup je dobre znázornený na nasledujúcom diagrame.



Obrázok č.15: Hardening webového servera.

Skript ***scan_and_fix.ps1*** napísaný v PowerShelli predstavuje automatizovaný prístup, ktorý dokáže spustiť kompletný Nmap sken vrátane vlastnej definície kontrolnej jednotky a následne na základe uložených logov automaticky vyparsovať bezpečnostné riziká zo reportu. Potom pomocou vopred definovaného súboru riešení dokáže podľa potreby tieto zraniteľnosti opraviť. Tento prístup výrazne urýchľuje a zjednodušuje proces zabezpečenia webového servera.

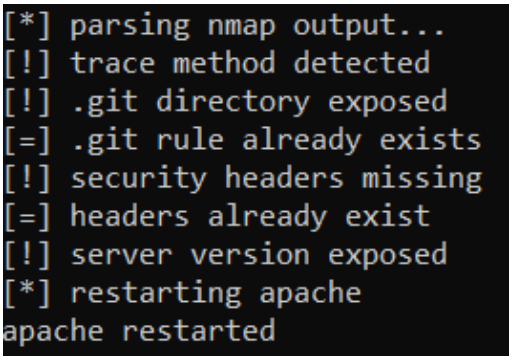
```
# trace
if ($nmapOutput -match "TRACE is enabled") {
    write-host "[!] trace method detected"
    if (Test-Path $apacheDefaults) {
        if (-not (Select-String -Path $apacheDefaults -Pattern "TraceEnable off" -Quiet)) {
            Add-Content -Path $apacheDefaults -Value "`nTraceEnable off"
            write-host "[+] trace disabled in httpd-default.conf"
        }
    } else {
        if (-not (Select-String -Path $apacheConf -Pattern "TraceEnable off" -Quiet)) {
            Add-Content -Path $apacheConf -Value "`nTraceEnable off"
            write-host "[+] trace disabled in httpd.conf"
        }
    }
}
```

Obrázok č.16: Príklad opravy zo súboru pravidiel automatizácie.

```
[*] parsing nmap output...
[!] trace method detected
[+] trace disabled in httpd-default.conf
[!] .git directory exposed
[+] .git access blocked
[!] security headers missing
[=] headers already exist
[!] server version exposed
[!] bad user-agents allowed
[=] bad bot rule already exists
[!] slowloris vulnerability detected
[+] request timeout settings added
[!] open proxy detected
[+] proxy access blocked
[*] restarting apache
```

Obrázok č. 17: Proces automatizovanej opravy webového servera.

Je dôležité spomenúť problém, na ktorý som narazil počas automatizovanej opravy – skript je potrebné spustiť v **režime správcu (administrátora)**. Ak sa spustí bez príslušných oprávnení, opravy síce prebehnú, ale v skutočnosti sa nezmení konfigurácia servera kvôli nedostatku prístupových práv. Výsledkom je, že pri následnom bezpečnostnom skene nie sú viditeľné žiadne opravené zraniteľnosti. Rozdiel medzi prvým a druhým skenom je teda pozorovateľný len vtedy, ak bol skript spustený s administrátorskými právami.



Obrázok č. 18: Druhý beh automatizovanej opravy.

V tomto prípade automatizácia úspešne opravila väčšinu bezpečnostných zraniteľností, pričom len v niekoľkých prípadoch bolo potrebné vykonať manuálnu opravu. Súhrn týchto prípadov je uvedený v tabuľke:

Zranitelnosti	Po automatizácií	Po manuálnom zásahu
TRACE method enabled	Opravená	Nebolo treba
Server banner disclosure	Opravená	Nebolo treba
Missing security headers	Opravená	Nebolo treba
Missing endpoints	Opravená	Nebolo treba
Accept known bots	Opravená	Nebolo treba
Slowloris DoS	Neopravená	Opravená
Proxy may be open	Neopravená	Opravená
Public .git/ folder exposed	Neopravená	Opravená
CSRF on multiple forms	Neopravená	Stále neopravená

Tabuľka č.2: Automaticky a manuálne opravené zranitelnosti.

Zhrnutie

V druhej fáze experimentu bol testovaný automatizovaný skript na detekciu a opravu zraniteľností webového servera Apache 2.4. Skript dokázal automaticky spustiť Nmap sken, vyparsovať zraniteľnosti z logov a aplikovať príslušné opravy. Väčšina problémov bola úspešne odstránená, no niektoré, ako napríklad CSRF ochrana si vyžadovali manuálny zásah alebo ostali nevyriešené. Dôležitým zistením bolo, že skript musí byť spustený s administrátorskými právami, inak zmeny v konfigurácii neprebehnú. Výsledkom je bezpečný server, pričom kombinácia automatizácie a manuálnej kontroly sa ukázala ako efektívny prístup.

Z dôvodu zdĺhavého testovacieho procesu a problematickej implementácie PowerShell skriptu neboli manuálne opravy zahrnuté do automatizácie, no sú spomenuté v časti budúca práca.

CSRF je ťažké opraviť automaticky, pretože ochrana musí byť implementovaná priamo v kóde webovej aplikácie, nie na úrovni servera. Vyžaduje generovanie a overovanie tokenov vo formulároch, čo závisí od použitého frameworku a autentifikácie. Preto si oprava vyžaduje manuálny zásah do aplikácie. Tomu problému som ďalej nevenoval v rámci experimentu, lebo to už nie je blízko k téme bezpečnosť webového servera.

Konklúzia

Po komplexnej analýze problematiky cieľom projektu bolo analyzovať a zvýšiť úroveň bezpečnosti webového servera Apache 2.4 pomocou kombinácie manuálnych a automatizovaných techník. V prvej fáze boli identifikované bežné zraniteľnosti servera pomocou nástrojov ako Nikto, Nmap a OWASP ZAP, pričom každý z nich ponúkol inú výhodu, či už rýchlosť, prispôsobenie alebo používateľské rozhranie. Nová kontrolná logika bola implementovaná a pridaná do .nse set, tým spôsobom Nmap skenovacia logika bola zlepšovaná.
Výsledkom bol zoznam reálnych bezpečnostných slabín, ktoré predstavovali riziko pre prevádzku servera.

V druhej fáze bol vyvinutý PowerShell skript, ktorý dokázal automaticky spustiť bezpečnostné skenovanie, vyparsovať nájdené hrozby a aplikovať riešenia podľa definovaného zoznamu. Skript úspešne opravil väčšinu problémov, pričom len niekoľko zraniteľností, ako napríklad aj CSRF vyžadovalo manuálny zásah, keďže ich riešenie presahuje rámec konfigurácie servera a vyžaduje zásah priamo do aplikácie.
Celkovo projekt preukázal, že aj dostupné open-source nástroje a jednoduchá skriptovacia logika môžu výrazne prispieť k zvýšeniu bezpečnosti serverovej infraštruktúry, ak sú správne nasadené a kombinované s odborným dohľadom.

Budúca práca

Pokiaľ ide o pokračovanie projektu, bolo by žiaduce vylepšiť automatizáciu takým spôsobom, že po vykonaní úplného skenovania a oprave zistených bezpečnostných rizík by systém automaticky spustil nový sken – nie však kompletný, ale zameraný len na predtým identifikované problémy, čím by sa zjednodušilo a urýchlilo overenie aplikovaných opráv.

Akonáhle bude táto optimalizácia zavedená, bude vhodné rozšíriť funkcionalitu automatizovaného skriptu o ďalšie bezpečnostné opravy, čím sa dosiahne komplexnejšie a širšie pokrytie bezpečnostných opatrení automatizovaným spôsobom.

Napokon by bolo vhodné túto automatizáciu ďalej rozvinúť a umožniť real-time monitorovanie bezpečnosti a automatickú bezpečnostnú konfiguráciu webových serverov. Takéto riešenie by bolo obzvlášť užitočné pre servery, ktoré hostujú dynamicky sa meniace webové stránky.

Bibliografia

[1] D. S. e. a. Wibowo, "Apache web server security with security hardening," *Journal of Soft Computing Exploration,* vol. 4, no. 4, p. 213–221, 2023.

[2] M. e. a. Aydos, "Security testing of web applications: A systematic mapping of the literature," *Journal of King Saud University-Computer and Information Sciences,* vol. 34, no. 9, p. 6775–6792, 2022.

[3] B. Laurie and P. Laurie, Apache: The definitive guide, O'Reilly Media, Inc., 2003.

[4] Apache Software Foundation, "Apache HTTP Server Version 2.4 Documentation," 2024. [Online]. Available: https://httpd.apache.org/docs/2.4/.

[5] T. Hubinek, "Apache 2.4 Evaluation and Integration with the Existing Middleware Infrastructure," 2013.

[6] N. Kew, The Apache modules book: application development with Apache, Prentice Hall Professional, 2007.

[7] Apache software foundation, "Apache HTTP Server," [Online]. Available: https://httpd.apache.org/.

[8] Hewlett Packard Enterprise, "NonStop HTTP Server 2.4 Reference Manual," [Online]. Available: https://support.hpe.com/hpesc/public/docDisplay?docId=a00099613en_us&page=GUID-E2F0455C-403C-48AA-AC1F-5CE6C96896C7.html&docLocale=en_US.

[9] J. Varghese, "Web Server Security - Beginner's Guide," [Online]. Available: https://www.getastra.com/blog/security-audit/web-server-security/.

[10] A. Welekwe, "Apache web server: Security guide," [Online]. Available: https://www.comparitech.com/net-admin/apache-web-server-security/.

[11] J. e. a. Shaid, "A comparative study of web application security parameters: Current trends and future directions," *Applied Sciences,* vol. 12, no. 8, p. 4077, 2022.

[12] J. T. Santoso and B. Raharjo, "Performance evaluation of penetration testing tools in diverse computer system security scenarios," *JURNAL TEKNOLOGI INFORMASI DAN KOMUNIKASI,* vol. 13, no. 2, p. 132–159, 2022.

[13] Y. e. a. Muhyidin, "Perbandingan Tingkat Keamanan Website Menggunakan Nmap Dan Nikto Dengan Metode Ethical Hacking," *Jurnal Teknologika,* vol. 12, no. 1, p. 80–89, 2022.

[14] A. Orebaugh and B. Pinkard, Nmap in the enterprise: your guide to network scanning, Elsevier, 2011.

[15] O. R. Laponina and S. A. Malakhovsky, "Using the ZAP vulnerability scanner to test web applications," *International Journal of Open Information Technologies,* vol. 5, no. 8, p. 18–26, 2017.

[16] Y. Makino and V. Klyuev, "Evaluation of web vulnerability scanners," 2015.

[17] Greenbone AG., "OpenVAS – Open Vulnerability Assessment Scanner," [Online]. Available: https://www.openvas.org/.

[18] Invicti Security., "Acunetix – Web Application Security Scanner," 2025. [Online]. Available: https://www.acunetix.com/.

[19] A. Nagy, "Github repository - BVI," 2025. [Online]. Available: https://github.com/xnagya/BVI.