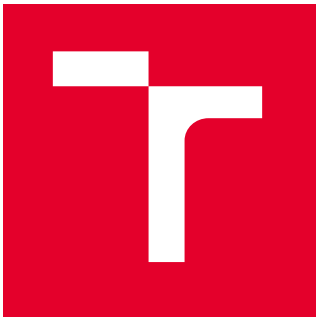


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO DYNAMICKOU ANALÝZU WEBOVÝCH APLIKACÍ

TOOL FOR DYNAMIC ANALYSIS OF WEB APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Patrik Píš**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Ilgner**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Patrik Píš ***ID:*** 219269

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Nástroj pro dynamickou analýzu webových aplikací

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem diplomové práce je návrh a vývoj vlastního nástroje využívajíciho metodického postupu pro provedení dynamické analýzy za účelem bezpečnostního testování webových aplikací. Součástí řešení bude integrace nástroje do platformy Penterep za účelem evidence nálezů analýzy a generování reportů. Výsledky budou v rámci diplomové práce ověřeny v sandbox prostředí.

DOPORUČENÁ LITERATURA:

[1] MYERS, Glenford J., Tom BADGETT a Corey SANDLER. The art of software testing. 3rd ed. Hoboken, New Jersey: Wiley, 2012, xi, 240 s. : il. ISBN 978-1-118-03196-4.

[2] HOFFMAN, Andrew. Web Application security: exploitation and countermeasures for modern web applications. O'Reilly Media, 2020.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 21.5.2024

Vedoucí práce: Ing. Petr Ilgner

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Táto diplomová práca sa zaoberá problematikou penetračného testovania webových aplikácií s primárnym zameraním na využívanie dynamickej analýzy. Práca analyzuje súčasný stav problematiky bezpečnosti webových aplikácií a sústredí sa ako na jednotlivé zraniteľnosti, tak i na protekčné mechanizmy, ktoré webové aplikácie implementujú. Hlavným cieľom práce je navrhnúť a implementovať automatizovaný ofenzívny nástroj, ktorý testuje odolnosť webovej aplikácie voči kybernetickým hrozbám. V porovnaní s inými dostupnými nástrojmi a ich obmedzeniami umožňuje navrhované riešenie efektívne testovanie rate limitingu a zároveň testovanie HTTP hlavičiek, atribútov súborov cookie a direktív content security policy. Na overenie jeho účinnosti pri podpore manuálneho penetračného testovania webových aplikácií bolo vytvorené sandbox prostredie, kde prebiehalo experimentálne testovanie. Nástroj bol testovaný aj v reálnom produkčnom prostredí pri penetračných testoch pre reálnych klientov s pozitívnou odozvou od profesionálnych penetračných testerov, čo poukazuje na jeho praktickosť a použiteľnosť pri penetračnom testovaní webových aplikácií.

KĽÚČOVÉ SLOVÁ

cookies, CSP direktívy, dynamická analýza, HTTP hlavičky, kybernetická bezpečnosť, penetračné testovanie, rate limiting

ABSTRACT

This master's thesis presents matters of penetration testing of web applications with the primary focus on the use of dynamic analysis. The thesis analyzes the current state of the art of web application security and focuses on both individual vulnerabilities and the protection mechanisms implemented by web applications. The main objective of the thesis is to design and implement an automated offensive tool that tests the resilience of a web application to cyber threats. Compared to other available tools and their limitations, the proposed solution enables efficient rate limiting testing while also allowing testing of HTTP headers, cookie attributes, and content security policy directives. To validate its effectiveness in supporting manual penetration testing of web applications, a sandbox environment was created where experimental testing was conducted. The tool was also tested in a real production environment during penetration tests for real clients with positive feedback from professional penetration testers, demonstrating its practicality and usability in web application penetration testing.

KEYWORDS

cookies, CSP directives, cybersecurity, dynamic analysis, HTTP headers, penetration testing, rate limiting

Vysadené pomocou balíčku `thesis` verzie 4.09; `https://latex.fekt.vut.cz/`

PÍŠ, Patrik. *Nástroj pro dynamickou analýzu webových aplikací*. Diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedúci práce: Ing. Petr Ilgner

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Patrik Píš
VUT ID autora:	219269
Typ práce:	Diplomová práca
Akademický rok:	2023/24
Téma závěrečnej práce:	Nástroj pro dynamickou analýzu webových aplikací

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Petrovi Ilgnerovi za odborné vedenie, konzultácie, trpezlivosť a vecné návrhy k práci. Rovnako by som chcel poďakovať aj Willimu Lazarovovi za pomoc a konzultácie pri riešení problémov s integráciou nástroja do platformy Penterep. V neposlednom rade by som chcel poďakovať dlhoročnému kolegovi a priateľovi Tomášovi Vašíčkovi za jeho odborné rady v oblasti penetračného testovania a podporu pri tvorbe tejto diplomovej práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

Oblasť informačnej bezpečnosti je aj napriek postupnému zvyšovaniu povedomia stále pozadu. Štandardný vývoj aplikácií stále naplno neimplementuje do svojich vývojových fáz bezpečnostné testovanie. S postupným zvyšovaním komplexnosti vyvíjaných systémov stúpa aj priestor na chyby a zraniteľnosti. Väčšina moderných spoločností sa verejnosti prezentuje pomocou webových stránok a aj interné firemné procesy a ich management sa postupne presúvajú do cloudu v štýle komplexných webových aplikácií a služieb. Dôraz na bezpečný vývoj a prevádzku webových služieb je tak neustále prísnejší. S narastajúcimi hrozbami sa tak bezpečnostné testovanie webových aplikácií stáva esenciálnym pre bezpečnú prevádzku a znižovanie potenciálneho dopadu na zákazníkov, zamestnancov alebo celú spoločnosť.

Táto diplomová práca sa zameriava na bezpečnostné testovanie webových aplikácií za pomoci dynamickej analýzy. Práca rozoberá možné spôsoby testovania bezpečnosti webových aplikácií, aké protekčné mechanizmy môžu tieto aplikácie implementovať a aký má ich vynechanie potenciálny dopad. Práca analyzuje moderné prístupy a metodiky k bezpečnostnému testovaniu webových aplikácií a využíva ich k vytvoreniu nástroja, ktorý umožní s pomocou dynamickej analýzy efektívne testovať prítomnosť vybraných zraniteľností, chybných konfigurácií, či iných bezpečnostných nedostatkov.

V prvej kapitole je predstavená problematika bezpečnosti webových aplikácií, spôsoby testovania bezpečnosti a odolnosti aplikácie voči hrozbám a platforma Penterep, pre ktorú je v praktickej časti vyvíjaný nástroj. Druhá kapitola je zameraná na bezpečnostné prvky, ktoré webové aplikácie implementujú a dopad, ktorý ich neprítomnosť alebo chybná konfigurácia spôsobuje. V tretej kapitole je predstavená problematika dynamickej analýzy a jej použitie v testovaní bezpečnosti webových aplikácií a metodika, ktorá dynamickú analýzu využíva. Posledná, štvrtá kapitola, obsahuje praktickú časť diplomovej práce a je zameraná na návrh a vývoj nástroja PtWebDA, ktorý pre testovanie využíva poznatky a metodiku analyzovanú v teoretickej časti tejto práce.

Ciele práce

Hlavným cieľom tejto diplomovej práce je navrhnúť a implementovať ofenzívny automatizovaný modulárny nástroj slúžiaci ako rozšírenie platformy Penterep určený pre penetračné testovanie webových aplikácií. Medzi čiastkové ciele vedúce k realizácii nástroja patria nasledovné:

- Analyzovať súčasný stav problematiky zraniteľností webových aplikácií,
- Analyzovať súčasný stav bezpečnostných mechanizmov webových aplikácií,
- Analyzovať a navrhnúť automatizovaný postup využívajúci dynamickú analýzu k testovaniu nedostatkov v zabezpečení webových aplikácií,
- Navrhnúť automatizovaný modulárny nástroj pre penetračné testovanie webových aplikácií,
- Implementovať automatizovaný modulárny nástroj v jazyku Python,
- Implementovať nástroj tak, aby bol kompatibilný s platformou Penterep,
- Otestovať nástroj v kontrolovanom sandbox prostredí a overiť jeho funkčnosť,
- Vytvoriť sprievodné texty pre platformu Penterep ako k jednotlivým testovacím scenárom, tak i k jednotlivým identifikovaným zraniteľnostiam.

1 Bezpečnosť webových aplikácií

Webové aplikácie sa stali populárnymi nástrojmi pre zdieľanie obsahu užívateľom, či poskytovanie služieb z rôznych odvetví akými sú napríklad internetové bankovníctvo, e-commerce, spravodajské portály a mnoho ďalších. Webové aplikácie zjednodušujú užívateľom prístup k službám, bez nutnosti inštalácie proprietárneho software a stávajú sa tak pre firmy najjednoduchším spôsobom ako poskytovať obsah alebo službu širokému spektru zákazníkov. Webové aplikácie sú však kvôli svojej dostupnosti verejnosti vystavené ako potenciálnym zákazníkom, tak potenciálnym útočníkom. Z toho dôvodu je bezpečnosť webových aplikácií kriticky dôležitá. Dopad útokov na webové aplikácie sa môže značne líšiť a je do značnej miery ovplyvnený technológiami, ktoré webová aplikácia používa a samotným charakterom webovej aplikácie. Pokiaľ však dôjde k zneužitiu nájdených zraniteľností vo webovej aplikácii, nedôjde len k narušeniu dôvernosti, integrity, či dostupnosti aktív a infraštruktúry prevádzkovateľa webovej aplikácie, ale môže dôjsť aj k úniku citlivých údajov patriacim používateľom webovej aplikácie, čo môže mať hlbší, individuálny dopad. Webové aplikácie sa neustále vyvíjajú, je nesmierne dôležité venovať dostatočnú pozornosť skúmaniu ich bezpečnosti a správania, nakoľko nárast komplexnosti webových služieb môže viesť k novým zraniteľnostiam, novým taktikám aktérov týchto hrozieb či novým metódam obchádzania už existujúcich obranných mechanizmov webových aplikácií[1].

1.1 Metódy bezpečnostného testovania

V nasledujúcej kapitole budú predstavené ako manuálne, tak automatizované metódy bezpečnostného testovania webových aplikácií. Výber konkrétneho typu bezpečnostného testovania je závislý na viacerých faktoroch akými sú napríklad nasledovné:

- Webová aplikácia nachádza vo fáze prvotného vývoja,
- Vyvíjaná aplikácia prešla už niekoľkými vývojovými cyklami,
- alebo je aplikácia už nasadená v produkčnom prostredí[2].

Výber vhodného typu bezpečnostného testovania ja závislý aj na finančných prostriedkoch objednávateľa. HĺbkovéManuálne testovanie je časovo aj finančne náročnejšie pričom automatizované testovanie je rýchlejšie, lacnejšie. Každá metóda má svoje výhody i nevýhody, kde si musí objednávateľ zvážiť svoj výber na základe vyššie uvedených faktorov. V nasledujúcom texte budú priblížené jednotlivé metódy, ich výhody a nevýhody[2].

1.1.1 Penetračné testovanie

Penetračné testovanie je druh testovania systému s cieľom otestovať jeho bezpečnosť či odolnosť voči kybernetickým hrozbám. Tento proces zahŕňa množstvo metodických postupov a procedúr, ktoré slúžia na evaluáciu stavu bezpečnosti daného systému či infraštruktúry. Hlavným cieľom penetračného testu je odhaliť čo najviac zraniteľností a chybných konfigurácii, ktoré by akýmkoľvek spôsobom viedli k narušeniu dôvernosti, integrity, či dostupnosti aktív klienta. Na rozdiel od skenu zraniteľností, dochádza pri penetračnom testovaní k pokusu o zneužitie nájdených zraniteľností samostatne či zreťazene s inými zraniteľnosťami, aby tak došlo k emulácii reálnej hrozby, ktorá klientovi potenciálne hrozí. So stále narastajúcim množstvom webových služieb je penetračný test webovej aplikácie veľmi populárnou variantnou penetračného testovania. Webové aplikácie často manipulujú s citlivými údajmi, akými sú čísla kreditných kariet, informácie o transakciách, užívateľské mená a heslá, rôzne médiá a pod. Penetračné testovanie sa tak pre prevádzkovateľa webovej aplikácie stáva efektívnym nástrojom pre overenie miery schopnosti zabezpečenia dôvernosti, integrity, či dostupnosti ako jeho aktív, tak i osobných údajov či dát užívateľov webovej aplikácie. Penetračné testovanie je možné rozdeliť do niekoľkých kategórii. Nasledujúci text popisuje rozdiely v možných variantách realizácie penetračného testu[3].

Úroveň znalosti

Black-box je druh penetračného testu, pri ktorom nemá tím realizujúci penetračné testovanie žiadne, prípadne veľmi stručné informácie ohľadom cieľového systému. V prípade penetračného testu webovej aplikácie to teda prakticky znamená, že test je vykonávaný bez prístupu k zdrojovému kódu či prístupu k dokumentácii alebo akýmkoľvek iným informáciám, ku ktorým by potenciálny útočník nemal prístup.

White-box je opakom black-box testovania. Je druh penetračného testu, pri ktorom má tím realizujúci penetračné testovanie kompletný prístup k informáciám o využívaných službách, zdrojovým kódom webovej aplikácie a prípadnej dokumentácie čiastkových komponent webovej aplikácie.

Grey-box je kombináciou testovania black-box a white-box. Je druh penetračného testu, pri ktorom má tím realizujúci penetračné testovanie malé či čiastočné informácie o testovanom systéme[3].

Strana realizácie

Interný penetračný test je test, pri ktorom je predmetom testovania entita alebo množina entít, ktoré sú dostupné z interného perimetru organizácie, ktorej patria

testované entity. Jedná o aktíva, ktoré sú dostupné len z internej siete organizácie a často je toto testovanie vykonávané v rámci scenára assumed breach.

Externý penetračný test je test, pri ktorom je predmetom testovania entita alebo množina entít, ktoré sú dostupné z externého perimetru. Jedná sa o aktíva, ktoré sú verejne dostupné z internetu[3].

Spôsob realizácie

Manuálny penetračný test je test vykonávaný špecializovaným tímom expertov na danú problematiku, pričom využívajú svoje skúsenosti a vedomosti na identifikáciu zraniteľností v aplikáciách či infraštruktúre. Tento typ testovania zahŕňa ručný prístup k skúmaniu a overovaniu zneužiteľnosti nájdených zraniteľností či chybných konfigurácií. Tento druh testovania je často precízny a personalizovaný, čo sa odzrkadľuje na jeho celkovej kvalite a hodnote pre klienta.

Automatizovaný penetračný test je test vykonávaný pomocou automatizovaných nástrojov. Tieto nástroje sa pokúšajú simulovať rôzne útoky za pomoci dynamickej analýzy. Automatizované penetračné testovanie je najčastejšie vykonávané voči webovým aplikáciám. Tento proces je často lacnejší a rýchlejší, avšak o to menej dôkladný a spoľahlivý a môže byť obsluhovaný aj nešpecializovaným personálom. Nástroje pre automatizované penetračné testovanie sú obmedzené množinou testov, ktoré sú v nich implementované.

Poloautomatizovaný penetračný test je kombináciou prvkov manuálneho a automatizovaného testovania. Penetrační testeri môžu použiť automatizované nástroje na identifikáciu zraniteľností, ktoré však stále vyžadujú manuálne overenie a posúdenie jednotlivých nálezov a ich potenciálny reálny dopad[4].

1.1.2 Skenovanie zraniteľností

Skenovanie zraniteľností, tiež známe ako *vulnerability assessment* či *DAST (Dynamic application security testing)*, je formou bezpečnostného testovania, ktorá pomáha identifikovať a klasifikovať zraniteľnosti testovaného systému. Skenovanie zraniteľností je primárne vykonávané automatizovanými nástrojmi akými sú napríklad Nessus či Acunetix. Podobne ako pri automatizovanom penetračnom testovaní, výsledok skenu zraniteľností značne závisí na množnie testov a zraniteľností obsiahnutých v databáze, ktorú daný nástroj používa[5].

1.1.3 Analýza zdrojového kódu

Analýza zdrojového kódu, tiež známa ako statická bezpečnostná analýza zdrojového kódu, je white-box forma testovania bezpečnosti aplikácie, ktorá pracuje výhradne

so zdrojovým kódom testovanej aplikácie, bez jej spustenia. V praxi sa statickou analýzou chápe spúšťanie automatizovaných nástrojov pre statickú analýzu kódu s cieľom odhaliť potenciálne zraniteľnosti s využitím techník ako Taint analýza či Data Flow analýza.

Statická analýza je často vykonávaná v implementačnej fáze bezpečnostného vývojového cyklu, tiež známeho ako SDL (Security Development Lifecycle), čo zabezpečuje, že sa daná aplikácia vyvíja vzhľadom k definovaným bezpečnostným štandardom. Statická analýza môže byť však veľmi nespoľahlivá, preto je skôr braná ako pomôcka pre bezpečnostných analytikov než nástroj, na ktorý sa dá vždy spoľahnúť[6].

1.2 Zraniteľnosti webových aplikácií

V nasledujúcej kapitole budú predstavené vybrané zraniteľnosti webových aplikácií, ich príčina a dopad na dôvernosť, integritu, či dostupnosť aktív webovej aplikácie alebo jej užívateľov.

1.2.1 Cross-site scripting

XSS (Cross-site Scripting) je zraniteľnosť, pri ktorej dochádza k spusteniu útočníkom kontrolovaného škodlivého kódu v kontexte užívateľa webovej aplikácie. Zraniteľnosť vzniká, keď webová aplikácia žiadnym alebo nedostatočným spôsobom ošetruje užívateľský vstup a následne ho vkladá do webovej stránky na zobrazenie užívateľovi. Škodlivý kód útočník vkladá do užívateľského vstupu v podobe JavaScript či HTML kódu, prípadne akýkoľvek iný typ kódu, ktorý môže webový prehliadač spustiť. Zraniteľnosť XSS je možné rozdeliť do nasledovných troch hlavných kategórií[3, 7]:

- Reflected XSS,
- Stored XSS,
- DOM-based XSS.

Reflected XSS je druh zraniteľnosti Cross-site scripting, pri ktorom dochádza k reflektovaniu užívateľského vstupu webovým serverom. Po spracovaní HTTP požiadavky so škodlivým kódom, je tento užívateľský vstup serverom spracovaný a následne zobrazený užívateľovi. Môže sa jednať napríklad o výsledky hľadania v databáze, kde sa reflektuje hľadaná fráza, či kľúčové slovo. Užívateľ môže byť útočníkom dovedený ku kliknutiu na predpripravený odkaz, ktorý je predpripravený tak, aby vykonal HTTP požiadavku so škodlivým kódom a reflektoval ho tak do prehliadača napadnutého užívateľa, čo vyústi spustením škodlivého kódu v jeho kontexte.

Stored XSS je druh zraniteľnosti Cross-site scripting, pri ktorom dochádza k permanentnému alebo dočasnému uloženiu škodlivého kódu zadaného útočníkom v po-

dobe užívateľského vstupu na server napr. do databázy. Najbežnejším príkladom tejto zraniteľnosti je fórum, kde užívatelia prispievajú svojimi komentármi. Ak mechanizmus pridávania komentárov obsahuje zraniteľnosť Stored XSS, môže dôjsť k spusteniu tohto škodlivého kódu u každého užívateľa webovej aplikácie, ktorý si daný komentár zobrazí.

DOM-based XSS je druh zraniteľnosti Cross-site scripting, pri ktorom dochádza k modifikácii tzv. DOM (Document Object Model). DOM je rozhranie pre programovú modifikáciu obsahu webových dokumentov. Pri zraniteľnosti DOM-based XSS dochádza pridaním škodlivého kódu do DOM vo webovom prehliadači napadnutého užívateľa, čo spôsobí jeho interpretáciu a spustenie.

Zraniteľnosť Cross-site scripting je v akejkoľvek forme veľkou hrozbou pre webové aplikácie a ich užívateľov. Dopad týchto hrozieb do značnej miery závisí od druhu webovej aplikácie a oprávneniach napadnutého užívateľa. Možné dopady zraniteľností XSS zahŕňajú krádež súborov cookie, prihlasovacích údajov a iných citlivých informácií, zobrazenie falošného obsahu, presmerovanie na podvodné stránky alebo vykonanie akejkoľvek akcie, ktorú môže vykonať napadnutý užívateľ[3, 7].

1.2.2 Cross-site request forgery

CSRF (Cross Site Request Forgery) je typ kybernetického útoku, pri ktorom útočník donúti webový prehliadač používateľa, aby vykonal nechcenú a neúmyselnú požiadavku na dôveryhodnú stránku. Úspešný útok môže viesť k vykonaniu akcií v mene používateľa bez jeho súhlasu. Presný dopad útoku Cross Site Request Forgery závisí od funkčnosti samotnej webovej aplikácie. V niektorých prípadoch môže útočník donútiť používateľa vykonať neúmyselnú škodlivú akciu na webovej stránke, na ktorej je používateľ aktuálne autentizovaný. Tým môže dôjsť k vymazaniu jeho používateľského účtu, vykonanie platobnej transakcie a mnohé ďalšie. Zraniteľnosť CSRF môže byť často reťazená s inými zraniteľnosťami, ako je napríklad cross-site scripting. Úspešné zneužitie kombinácie zraniteľností CSRF a XSS môže viesť k spusteniu škodlivého kódu v prehliadači obete[8].

1.2.3 Clickjacking

Zraniteľnosť umožňuje útočníkom vykonávať útoky typu *clickjacking*. Clickjacking je technika využívaná na útoky na používateľov webovej aplikácie, pri ktorej útočník využíva nevedomú interakciu používateľa s webovou stránkou. Základnou myšlienkou clickjackingu je skryť škodlivý obsah, napríklad tlačidlo alebo odkaz, na webovej stránke pod iný, zdanlivo neškodný obsah. Keď používateľ klikne na zdanlivo bezpečný prvok, v skutočnosti klikne na škodlivý prvok, ktorý môže vykonať rôzne nežiaduce akcie ako napríklad odoslať rôzne formuláre, zmeniť nastavenia účtu

alebo dokonca nainštalovať škodlivý softvér. Techniku clickjacking môže útočník implementovať rôznymi spôsobmi, vrátane použitia priehľadných prvkov `iframe`, ktoré prekrývajú stránku, alebo použitia skrytých tlačidiel umiestnených na pozadí stránky[9].

MIME type sniffing

MIME (Multipurpose Internet Mail Extensions) je štandard používaný na identifikáciu typu obsahu odosielaného cez internet, napríklad textu, obrázkov, zvuku, videa, či iných súborov. Typy MIME sú definované v HTTP hlavičkách webových požiadaviek a odpovedí. *MIME type sniffing* je operácia, ktorú vykonávajú mnohé webové prehliadače na to, aby zistili typ obsahu tela HTTP odpovede na základe samotného obsahu týchto dát. Pokiaľ webový server odpovie na zdroj požadovaný webovým prehliadačom bez použitia HTTP hlavičky Content-Type, webový prehliadač sa pokúsi typ obsahu identifikovať. Táto funkcionalita sa môže stať nebezpečnou pokiaľ je útočník schopný modifikovať obsah webového servera a tak napríklad namiesto obrázku nahrať JavaScript súbor so škodlivým kódom. Aj keď si užívateľ chce zobraziť obrázok, webový prehliadač vyhodnotí obsah ako JavaScript a daný škodlivý kód spustí. Zraniteľnosť je možné eliminovať použitím HTTP hlavičky X-Content-Type-Options a nastavením jej hodnoty na "nosniff"[10].

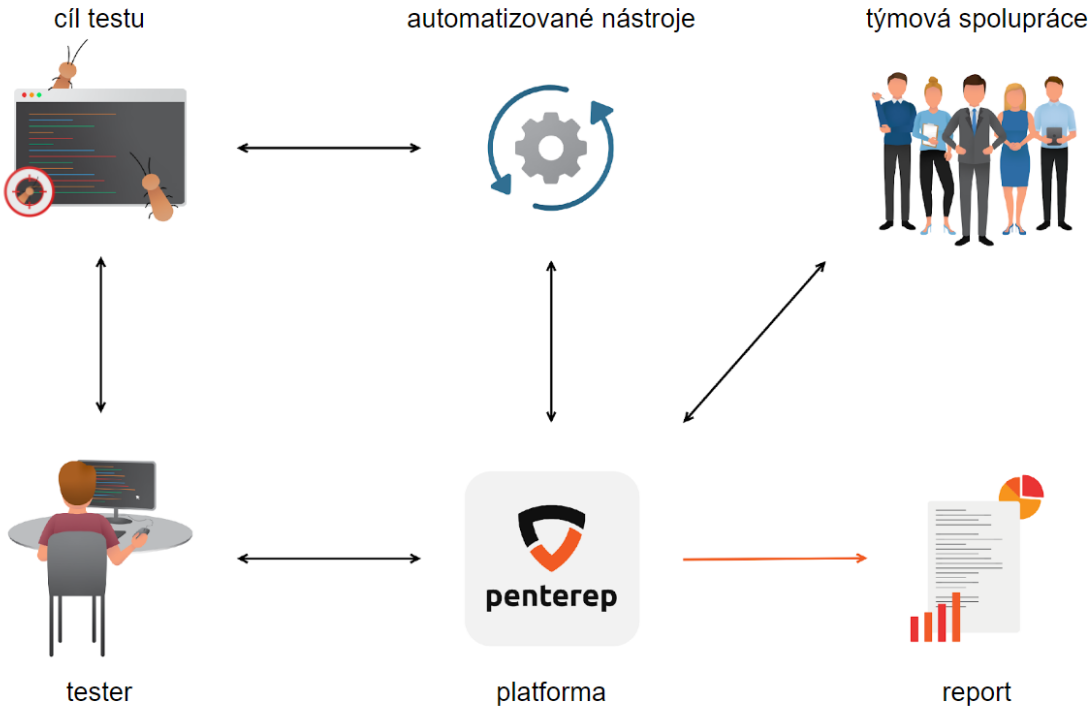
1.3 Platforma Penterep

Penterep je webová, vysoko škálovateľná platforma pre podporu penetračného testovania. Hlavným cieľom platformy je poskytnúť prostredie pre penetračných testerov, ktorí ho budú využívať k efektívnemu testovaniu, komunikácii a spolupráci so svojím, či vývojárskym tímom alebo so samotným vedením. Architektúra platformy Penterep kladie dôraz na vysokú modularitu, škálovateľnosť a prívetivosť prostredia pre jeho užívateľov. Medzi hlavné komponenty platformy patria nasledovné:

- Webová aplikácia,
- Server pre manažment užívateľov,
- Aplikačný server,
- Server pre automatizované testovanie.

Platforma umožňuje penetračným testerom testovať ako manuálne, tak automatizovane. Platforma obsahuje sadu testov, či exploitačných nástrojov, ktoré sa pokúsia danú zraniteľnosť zneužiť a potenciálny nález v platforme zdokumentovať. Webové prostredie platformy umožňuje testerom zadávať nálezy aj manuálne. Zlo-

žitejšie testy, ktoré je potrebné vykonať manuálne a ich výsledky je tak možné jednoducho pridať k automatizovaným nálezom[11].



Obr. 1.1: Schéma používania platformy Penterep [11]

2 Bezpečnostné mechanizmy webových aplikácií

Webové aplikácie a ich zraniteľností je možné klasifikovať do 10 hlavných kategórii podľa zoznamu OWASP Top 10[1]. Napriek tomu, že existuje značné množstvo zraniteľností, ktoré majú na webovú aplikáciu priamy dopad a umožňujú jej kompromitáciu, niektoré zdanlivo menej závažné zraniteľnosti môžu mať väčší dopad na užívateľov a ich súkromie ako na samotnú webovú aplikáciu. V nasledujúcich podkapitolách budú predstavené niektoré bezpečnostné mechanizmy využívané webovými aplikáciami a API k zvýšeniu bezpečnosti a odolnosti voči hrozbám.

2.1 Rate Limiting

Rate limiting je technika, ktorú využívajú webové aplikácie a rozhrania API na limitovanie počtu požiadaviek od klientov. Táto technika je primárne využívaná, k zabráneniu nežiadúcej, automatizovanej aktivity potenciálnych útočníkov alebo automatov. Medzi nežiadúce aktivity, ktoré je možné týmto mechanizmom zmierniť, či dokonca úplne eliminovať patria:

- **Útoky hrubou silou**,
- **Fuzzing** – Fuzzing je technika spočívajúca v opakovanom zasielaní špeciálne upravených požiadaviek za účelom monitorovania reakcie webovej aplikácie a následnej identifikácie jej potenciálnych zraniteľností,
- **DoS** a **DDoS** – DoS a DDoS sú útoky na webové služby, ktorých hlavným cieľom je odopretie služby používateľom danej služby. Útok je často realizovaný zahltením webového servera požiadavkami,
- alebo **Web scraping** – Web scraping je proces extrahovania dát a informácií z webových stránok pomocou automatizovaných nástrojov. Táto technika umožňuje zbierať dáta z cieľových webových stránok a následne ich spracovávať alebo ukladať na ďalšie použitie[12].

Dva hlavné spôsoby, akými je možné implementovať tento obranný mechanizmus, sú implementácia rate limitingu na webovom serveri alebo priamo vo webovej aplikácii. V oboch prípadoch je základným prvkom tohto mechanizmu sledovanie IP adries prichádzajúcich HTTP požiadaviek a časového intervalu medzi týmito požiadavkami. Rate limiting pozoruje ako množstvo času, ktor0 medzi jednotlivými požiadavkami prešlo, tak i množstvo požiadaviek v stanovenom časovom intervale[12, 13, 14].

[1] https://owasp.org/www-project-top-ten/

Okrem prevencie vyššie uvedených útokov, rate limiting obmedzuje útočníkovi enumeračný potenciál. Chýbajúce obmedzenie počtu požiadaviek užívateľa umožňuje útočníkovi efektívne používať techniky ako fuzzing, kde okrem testovania užívateľských vstupov, je pomocou tejto techniky možné napríklad identifikovať adresárovú štruktúru webovej aplikácie. Odhalenie adresárovej štruktúry môže viesť k odhaleniu ďalšej funkcionality a zefektívniť tak snahu útočníka o identifikáciu zraniteľností danej webovej aplikácie. So správne nastaveným rate limitingom sa potenciálnemu útočníkovi práca značne skomplikuje, až úplne znemožní[12, 13, 14].

2.2 HTTP hlavičky

HTTP hlavičky sú prvky HTTP požiadaviek a odpovedí medzi klientom a serverom. Tieto hlavičky poskytujú informácie o danej požiadavke, prípadne odpovedi. V prípade požiadaviek, hlavičky nesú mnohé informácie, medzi ktoré patria napríklad informácie o webovom prehliadači užívateľa, podporovaný jazyk, súbory cookies a pod. V prípade odpovedí, môžu hlavičky niesť informácie napríklad o veľkosti a typu poskytovaného obsahu a akým štýlom by mal webový prehliadač užívateľa daný obsah interpretovať. Okrem informácii potrebných pre riadenie a interpretáciu komunikácie medzi klientom a serverom, môže mať prítomnosť, resp. neprítomnosť určitých hlavičiek značný dopad na bezpečnosť webovej aplikácie a jej užívateľov. V nasledujúcom texte budú predstavené základné bezpečnostné hlavičky a hlavičky, ktorých prítomnosť môže niesť pre webovú aplikáciu potenciálne riziko.

2.2.1 Content-Security-Policy

CSP (Content Security Policy) definuje pravidlá načítavania webového obsahu. Vyžaduje si dôkladné nastavenie a presnú definíciu. Ak je CSP definovaná, má významný vplyv na spôsob, akým webové prehliadače zobrazujú webové stránky. CSP pomáha zisťovať a predchádzať širokému spektru útokov vrátane XSS a ďalších Cross-Site útokov. CSP je implementovaná pomocou HTTP hlavičky s názvom Content-Security-Policy. Táto hlavička umožňuje webovým stránkam definovať, ktorý obsah a ktoré skripty sú povolené na danej stránke. Tabuľka 2.1 popisuje možné direktívy a ich význam pri nastavovaní hlavičky Content-Security-Policy[15, 16].

2.2.2 X-Frame-Options

Hlavička X-Frame-Options definuje webovému prehliadaču, či by mal zobrazovať obsah webovej stránky v HTML elementoch ako `<frame>`, `<iframe>`, `<embed>` alebo `<object>`. Hlavička zabezpečuje, aby nebolo možné vkladať obsah webovej stránky

Tab. 2.1: Tabuľka popisujúca direktívy CSP

Hodnota	**Popis**
default-src	Určuje predvolený zdroj pre obsah, ak nie je špecifikovaný žiadny iný.
script-src	Kontroluje, odkiaľ môžu byť načítané skripty.
img-src	Špecifikuje zdroje, odkiaľ môžu byť načítané obrázky.
style-src	Definuje, kde môžu byť používané štýly a CSS.
font-src	Určuje, odkiaľ sú povolené fonty.
connect-src	Kontroluje, s ktorými zdrojmi môže webová aplikácia komunikovať cez XMLHttpRequest, WebSockets a podobné technológie.
frame-src	Špecifikuje, ktoré stránky môžu byť zobrazené vo frame alebo iframe elementoch.
frame-ancestors	Určuje, ktoré stránky môžu zahrnúť aktuálnu stránku vo frame alebo iframe elementoch.
media-src	Kontroluje, odkiaľ môžu byť načítané médiá, ako audio alebo video súbory.
report-uri alebo report-to	Určuje URL, kam majú byť posielané správy o porušeniach politiky na zabezpečenie monitorovania a riadenia CSP.
child-src	Špecifikuje, ktoré zdroje sú povolené pre resource loading vo frame alebo iframe elementoch.
form-action	Kontroluje, kam môžu byť odosielané formuláre.
object-src	Určuje, odkiaľ môžu byť načítané objekty, ako napríklad Flash.
base-uri	Špecifikuje základný URI pre relatívne adresy v stránke.
manifest-src	Kontroluje, odkiaľ môže byť načítaný manifest aplikácie.
prefetch-src	Definuje zdroje, ktoré môžu byť prednabité pomocou `<link rel="prefetch»`.

Tab. 2.2: Tabuľka popisujúca direktívy X-Frame-Options

Hodnota	**Popis**
deny	Stránka nebude schopná byť zahrnutá do iframe na žiadnej inej stránke, čím sa zabezpečuje úplné zakázanie zahrnutia.
sameorigin	Stránka môže byť zahrnutá iba na stránkach s rovnakým pôvodom (origin), čím sa obmedzuje zahrnutie na doménu.
allow-from	Umožňuje špecifikovať konkrétnu URI adresu, kde môže byť stránka zahrnutá do iframe, čím sa obmedzuje zahrnutie na konkrétnu stránku alebo doménu.

do inej a predchádzať tak útokom typu Clickjacking. Použitie tejto hlavičky je vhodné pokiaľ má užívateľ na danej stránke možnosť interakcie, napríklad pomocou odkazov alebo tlačidiel. Pokiaľ na danú HTTP požiadavku odpovedá server napríklad dátami typu JSON, hlavička X-Frame-Options nepridáva žiadnu vrstvu zabezpečenia. Hlavička X-Frame-Options je však postupne nahradzovaná direktívami v Content Security Policy (CSP), konkrétne frame-ancestors, ktorá funguje na rovnakom princípe. Nie každý webový prehliadač však plne podporuje CSP, preto je použitie hlavičky X-Frame-Options stále silne odporúčané. Tabuľka 2.2 popisuje možné direktívy a ich význam pri nastavovaní hlavičky X-Frame-Options[15, 17].

2.2.3 X-Content-Type-Options

Hlavička X-Content-Type-Options zabraňuje webovému prehliadaču, aby interpretoval prijaté súbory ako iný typ MIME, než ktorý je uvedený v HTTP hlavičke Content-Type. Toto správanie sa odborne nazýva MIME sniffing. Chýbajúce nastavenie tejto hlavičku môže spôsobiť nežiadúcu interpretáciu prijatých dát ako spustiteľných. Užívatelia webovej aplikácie si tak môžu nevedome stiahnuť a spustiť škodlivý kód napr. v podobe skriptov v jazyku JavaScript alebo HTML[15, 18].

2.2.4 Referrer-Policy

Hlavička Referrer-Policy definuje bezpečnostnú politiku webového prehliadača, ktorá upravuje akým spôsobom webová stránka zdieľa informácie o stránkach, z ktorých prišiel používateľ na aktuálnu stránku. Hlavička Referrer-Policy obsahuje adresu predchádzajúcej webovej stránky a môže obsahovať všetky parametre, ktoré boli použité v HTTP požiadavke pre zobrazenie predchádzajúcej webovej stránky. To môže mať na prvý pohľad nevinné implikácie vrátane business analytík, logovania alebo procesov optimalizácie caching služieb. Obsah hlavičky Referrer-Policy môže byť problematický pokiaľ sa v ňom začnú nachádzať citlivé informácie. Ako príklad je

Tab. 2.3: Tabuľka popisujúca direktívy Referrer-Policy

Hodnota	**Popis**
no-referrer	Nezdieľa žiadne informácie o referroch.
no-referrer-when-downgrade	Nezdieľa informácie o referroch pri prechode zabezpečeného (HTTPS) na nezabezpečený (HTTP).
origin	Zobrazí iba základnú URL adresu referra (bez údajov o ceste alebo parametre URL).
strict-origin	Podobne ako "origin", ale poskytne informácie iba, ak referer je zo zabezpečeného zdroja.
same-origin	Zdieľa informácie iba medzi stránkami s rovnakým pôvodom (origin) na tej istej doméne a portovom čísle.
strict-origin-when-cross-origin	Poskytne informácie o referoch, ak sú referer a aktuálna stránka z rôznych zdrojov, ale iba, ak referer pochádza zo zabezpečeného zdroja.

možné uviesť odkaz na reset hesla, na ktorom sa zároveň nachádza odkaz na sociálne médiá. Pokaľ užívateľ pristúpi na tento odkaz a zároveň pristúpi na odkaz na sociálne médiá, môže dôjsť k úniku URL užívateľa pre reset hesla a potenciálne tak kompromitovať jeho užívateľský účet. Tabuľka 2.3 popisuje možné direktívy a ich význam pri nastavovaní hlavičky Referrer-Policy[15, 19].

2.2.5 Strict-Transport-Security

Hlavička Strict Transport Security, tiež všeobecne označovaná skratkou HSTS (HTTP Strict Transport Security), je dôležitým nástrojom pre zabezpečenie HTTP komunikácie prostredníctvom šifrovaného kanálu s použitím kryptografických protokolov akými sú SSL, či TLS. Implementácia tejto hlavičky zabezpečuje, že komunikácia medzi klientom a serverom bude prebiehať výhradne prostredníctvom protokolu HTTPS. Hlavným princípom tohto mechanizmu je použitie hlavičky Strict-Transport-Security v HTTP odpovediach, kde webová aplikácia oznamuje webovému prehliadaču, že sa všetky spojenia na dané webové stránky majú udržiavať len prostredníctvom HTTPS. Po prijatí tejto hlavičky si webový prehliadač zapamätá túto informáciu a pri ďalšom pokuse o prístup na túto stránku vytvorí spojenie len prostredníctvom HTTPS, aj keď používateľ (alebo útočník) zadajú do URL protokol HTTP[15, 20].

Tab. 2.4: Tabuľka popisujúca direktívy Strict-Transport-Security

Parameter	**Popis**
max-age	Určuje, ako dlho by sa malo udržiavať spojenie cez HTTPS v sekundách.
includeSubDomains	Voliteľný parameter, ktorý indikuje, či sa politika HSTS má uplatňovať aj na všetky subdomény hlavnej domény.
preload	Voliteľný parameter, ktorý indikuje, či by sa stránka mala zahrnúť do oficiálneho zoznamu HSTS preload.

Spoločnosť Google ponúka službu HSTS preload[2], kde zaregistrovaním domény a dodržiavaním uvedených pravidiel možno zaručiť, že webové prehliadače budú pristupovať k doméne len prostredníctvom zabezpečeného protokolu HTTPS, a to aj pri prvom pripojení pred odoslaním hlavičky Strict-Transport-Security. Tento preload zoznam poskytuje spoločnosť Google, čo znamená, že HSTS preload nie je oficiálnym prvkom špecifikácie HSTS. Moderné prehliadače ako Chrome, Firefox či Safari však túto službu aktívne využívajú, preto sa odporúča doménu webovej aplikácie do tohto zoznamu zahrnúť. Tabuľka 2.4 popisuje možné direktívy a ich význam pri nastavovaní hlavičky Strict-Transport-Security[15, 20].

2.2.6 Permissions-Policy

Hlavička Permissions-Policy, skôr známa tiež ako Feature-Policy, je bezpečnostná HTTP hlavička špecifikujúca funkcie, ktoré môžu byť alebo nemôžu byť použité v rámci webovej aplikácie. Hlavička definuje množinu politík, ktoré špecifikujú, ku ktorým zdrojom môže webová aplikácia pristupovať. Vynechanie tejto hlavičky môže mať významný bezpečnostný dopad na ochranu súkromia užívateľov. Moderné webové prehliadače využívajú radu špecifických rozhraní API k interakcii so zdrojmi zariadenia, z ktorého užívateľ pristupuje na webové stránky aplikácie. V prípade zneužitia môže dôjsť k interakcii s fyzickými zariadeniami pripojenými k zariadeniu užívateľa akými sú obrazovka, senzory či USB zariadenia a tak pristúpiť k citlivým informáciám, šíriť škodlivý software či pokúsiť sa ukradnúť citlivé informácie zo schránky určenej na kopírovanie a podobne. Tabuľka 2.5 popisuje možné direktívy a ich význam pri nastavovaní hlavičky Permissions-Policy[15, 21].

[2]https://hstspreload.org/

Tab. 2.5: Tabuľka popisujúca direktívy Permissions-Policy

Direktíva	**Popis**
accelerometer	Prístup k senzoru akcelerometra.
camera	Prístup ku kamere zariadenia.
microphone	Prístup k mikrofónu zariadenia.
geolocation	Prístup k geolokačným informáciám.
usb	Prístup k USB zariadeniam.
hid	Prístup k HID (Human Interface Device) zariadeniam.
clipboard-read	Oprávnenie pre čítanie schránky.
clipboard-write	Oprávnenie pre zápis do schránky.
display-capture	Prístup k zachytávaniu obrazovky alebo časti obrazovky.
storage-access	Prístup k API na prístup k lokálnemu úložisku.
sync-xhr	Schopnosť vykonávať synchronné HTTP žiadosti.

2.2.7 Hlavičky poskytujúce informácie o prostredí

Určite webové frameworky, či webové servery pri vychádzajúcich konfiguráciách implementujú do HTTP odpovedí webových aplikácií špecifické HTTP hlavičky nesúce informácie identifikujúce danú použitú technológiu. Prítomnosť týchto hlavičiek poskytuje potenciálnemu útočníkovi užitočnú informáciu o tom, aké softwarové komponenty, či služby sú webovou aplikáciou využívané. Tieto hlavičky najčastejšie obsahujú informácie ako názov použitej technológie a v niektorých prípadoch i konkrétna verzia. Prítomnosť týchto hlavičiek nespôsobuje priamo ohrozenie dôvernosti, integrity či dostupnosti webovej aplikácie, avšak poskytovanie týchto informácii v HTTP hlavičkách umožňuje potenciálnemu útočníkovi efektívnejšie hľadať zraniteľnosti špecifické pre konkrétnu verziu webového servera alebo back-end technológie, ktorú webová aplikácia využíva. HTTP hlavičiek splňujúcich vyššie uvedený popis môže byť veľké množstvo, nakoľko ich názvy či implementácia môžu byť proprietárne, avšak existujú HTTP hlavičky, ktoré sú štandardnou súčasťou webových frameworkov, alebo súčasťou vychádzajúcej konfigurácie webového serveru. Medzi tieto HTTP hlavičky paria nasledové:

- Server,
- X-Powered-By,
- X-AspNet-Version,
- X-AspNetMvc-Version[3, 15].

2.3 HTTP Cookies

Cookies je možné chápať ako súbory informácii, ktoré webový server vygeneruje a zašle webovému prehliadaču klienta, ktorý tieto cookies ukloží na dopredu definovanú dobu. Súbory cookies môžu obsahovať rôzne informácie ohľadom réžie prevádzky webovej aplikácie, ale aj cookies, ktoré slúžia na autentizáciu klienta voči danej webovej aplikácii. Súbory cookies sa prenášajú v rámci hlavičky `Cookie`, ktorá je súčasťou danej HTTP požiadavky a môžu byť webovým serverom nastavované pomocou hlavičky `Set-Cookie` v HTTP odpovedi[22, 23].

2.3.1 Použitie cookies

Použitie súborov cookies, je značne závislé na štýle a charakteru webovej aplikácie a pre správnu interpretáciu dopadu ich chybných konfigurácii, je potrebné plne chápať ich širokospektrálne využitie. Cookies je možné rozdeliť do troch hlavných kategórii:

- Užívateľské relácie,
- Personalizácia,
- Sledovanie aktivity.

Užívateľské relácie

Tieto cookies slúžia na identifikáciu užívateľa a jeho aktivitu v danom čase v danej webovej aplikácii. Relačné cookies zväčša obsahujú náhodný alfanumerický reťazec, ale môžu obsahovať aj *base64* zakódované informácie o danej relácii, ktoré sú na záver digiálne podpísané[3]. S takmer každou HTTP požiadavkou voči webovému serveru, zasiela webový prehliadač tieto cookies v rámci daného HTTP požiadavku. Po prijatí webovým severom dochádza k overovaniu platnosti daného autentizačného cookie a po úspešnom vyhodnotení poskytuje daná webová aplikácia relevantný obsah pre autentizovaného užívateľa.[22]

Personalizácia

Personalizačné cookies často slúžia na spríjemnenie pobytu užívateľa na danej webovej stránke. Sú to často prvky charakteru webovej aplikácie, ktoré si webová stránka "zapamätá", napr. obsah nákupného košíka, a pri ďalšej návšteve webovej stránky webový server tieto informácie automaticky poskytne.[22]

[3]Podobným štýlom fungujú napr. JSON Web Tokens, viď https://jwt.io/introduction

Sledovanie aktivity

Neposledným spôsobom využitia cookies je sledovanie aktivity daného užívateľa webovej aplikácie. Webové stránky umiestňujú tieto cookies do prehliadača používateľa s cieľom zhromažďovať údaje o jeho online aktivitách. Tieto cookies umožňujú webovým stránkam pamätať si preferencie používateľa, udržiavať relácie a sledovať správanie používateľa na rôznych webových stránkach. Hlavným účelom sledovacích súborov cookie je umožniť personalizáciu obsahu, cielenú reklamu a analýzu prevádzky samotných webových stránok[22, 24]. Možno ich rozdeliť na dva hlavné typy:

- Súbory cookie prvej strany – nastavuje ich navštívená webová lokalita,
- Súbory cookie tretej strany – vytvárajú externé služby.

Každý užívateľ má pridelený vlastný identifikátor, ktorý sa ukladá v podobe cookies, ktoré sú následne spolu s ostatnými cookies webovému serveru zasielané. Webový server tieto cookies spracuje a analyzuje a v prípade tretích strán poskytuje tieto cookies ďalej na analýzu tretej strane, najčastejšie službe, ktorá túto analýzu poskytuje[22, 24]. Medzi najznámejšie služby poskytujúce sledovanie aktivity na webových stránkach patria:

- Google Analytics,
- Adobe Analytics,
- Matomo,
- a Mixpanel.

2.3.2 Bezpečnosť cookies

Cookies, ako bolo uvedené v sekciách vyššie, majú tendenciu obsahovať citlivé informácie ako o reláciách užívateľov, tak o ich samotnej aktivite na webových stránkach. Preto je potrebné tieto citlivé informácie chrániť ako pred neautorizovaným prístupom rozšírení webového prehliadača[4], tak i samotného kódu webovej aplikácie na strane klienta. Pri zraniteľnostiach XSS a CSRF sa často stretávame práve s neautorizovaným prístupom k citlivým cookies, ktoré ústia v impersonáciu užívateľa a vykonávanie akcii na webovej stráne v jeho mene bez jeho vedomia.

Súbory cookies je možné chrániť pomocou tzv. cookie atribútov (angl. *cookie attributes* či *cookie flags*), ktoré sú nastavované webovým serverom pri zasielaní hlavičky `Set-Cookie` v HTTP odpovedi. V nasledujúcom texte budú predstavené jednotlivé atribúty a ich význam v kontexte bezpečnosti.

[4]https://www.packetlabs.net/posts/hackers-exploit-cookie-stuffing-chrome-extensions-to-spy-on-users/

Atribút „Secure“

Atribút „Secure“ špecifikuje webovému prehliadaču nutnosť prenosu daného cookie výhradne cez bezpečné HTTPS spojenie. Tým sa zabezpečuje, že daný cookie nebude webovým prehliadačom zasielaný webovému serveru cez nezabezpečené, nešifrované spojenie. Tento mechanizmus zabezpečuje ochranu daného cookie pred útokmi MitM (Man-in-the-Middle), kde dochádza k odposluchu HTTP komunikácie. Pri tomto útoku sa útočník môže pokúsiť vyčítať z komunikácie užívateľské cookies a použiť ich k prihláseniu sa do danej webovej aplikácie. Atribút „Secure“ nezabráni útoku MitM, ale zmierňuje jeho dopad o znemožnenie krádeži daného cookie[25].

Atribút „HttpOnly“

Atribút „HttpOnly“ špecifikuje webovému prehliadaču obmedzenie prístupu k súborom cookie výhradne na operácie na strane servera. Atribút zabezpečuje, že k danému cookie nie je možné pristúpiť zo skriptovacích jazykov ako napr. *JavaScript* na strane klienta. Toto opatrenie je akousi prevenciou dopadu zraniteľnosti XSS, kde sa pri vložení škodlivého kódu do stránky užívateľa útočník pokúša ukradnúť citlivé cookies. Pokiaľ má daný cookie nastavený atribút „HttpOnly“, webový prehliadač dovolí pristupovať k užívateľským cookies len v prípade vykonávania HTTP požiadavky, čo však pri zraniteľnosti XSS nebráni útočníkovi vykonávať HTTP požiadavky a impersonovať tak napadnutého užívateľa. Atribút „HttpOnly“ sa nesmie považovať za ochranu proti útoku XSS, ale len nástrojom na zmiernenie jeho dopadu[25].

Atribút „SameSite“

Atribút „SameSite“ špecifikuje webovému prehliadaču, akým spôsobom má zasielať danej webovej aplikácii cookies. Tento atribút vznikol ako ochrana, či zmiernenie dopadu zraniteľnosti CSRF, kedy dochádza k zneužitiu “automatického“ zasielania cookies webovým prehliadačom. Aktuálne môže atribút „SameSite“ nadobúdať tri hodnoty: Strict, Lax, a None.

Hodnota *Strict* špecifikuje, že daný cookie bude odoslaný webovým prehliadačom len pokiaľ je cieľová destinácia HTTP požiadavky rovnaká, ako pre ktorú bol daný cookie uložený. Hodnota *Lax* povoľuje zasielanie cookies z tretích strán, pokiaľ sa jedná o tzv. *top-level navigation*. Hodnota *None* je zo všetkých najpermisívnejšia a zároveň najnebezpečnejšia. Povoľuje zasielanie cookies ako z prvej strany, tak z tretích strán, avšak vyžaduje, aby daný cookie obsahoval atribút *Secure*[25].

3 Dynamická analýza webových aplikácií

Dynamickú analýzu môžeme chápať ako testovanie, či ohodnotenie funkcionality alebo bezpečnosti aplikácie počas jej behu. Na rozdiel od statickej analýzy sa pri dynamickej analýze nepracuje so zdrojovým kódom aplikácie, ale skúma sa správanie ako pri bežnom, tak pri nezvyčajnom, často útočníkom vynúteného behu aplikácie[26].

Na dynamickú analýzu je možné pozerať ako na metódu testovania kvality software a zisťovania chýb pri vývoji danej aplikácie, ktoré môžu narušiť jej bežný chod. Značné množstvo chýb v aplikáciách môže spôsobiť korupciu pamäti a prerušiť tak chod aplikácie a ohroziť dáta, s ktorými užívateľ pracuje. Ďalším príkladom chýb, ktoré je nutné podchytiť sú chyby, ktoré vznikajú pri multivláknových či multiprocesových aplikáciách, ktoré paralelne pristupujú k určitým zdrojom. Vyššie uvedené príklady chýb v aplikáciách je možné detegovať ako statickou, tak i dynamickou analýzou, avšak, ani jedna metóda nedokáže odhaliť všetky chyby, preto sa v praxi kombinujú oba prístupy[27].

Ďalším, v bezpečnosti veľmi populárnym, uplatnením dynamickej analýzy je analýza malware. Malware, ako akýkoľvek iný spustiteľný program, nesie pri svojom behu charakteristické behaviorálne signatúry, ktoré môžu viesť k jeho detekcii. Okrem samotnej detekcie a odstránenia malware z napadnutého zariadenia, je možné dynamickú analýzu vnímať aj ako nástroj pre zistenie dopadu, ktorý beh daného malware môže spôsobiť. Dynamická analýza malware môže tiež viesť k identifikácii tzv. callback Command&Control serverov, z ktorých sú ofenzívne operácie iniciované a manažované. Identifikácia týchto serverov, môže pomôcť orgánom verejnej moci pri vyšetrovaní, konkrétne priblíženie sa k páchateľovi či organizácii, ktoré je za kybernetický útok zodpovedná[28].

Neposledným, pre túto prácu najpodstatnejším, uplatnením dynamickej analýzy je dynamická analýza za účelom zisťovania zraniteľností danej aplikácie, ktoré by bolo možné útočníkom zneužiť. Pri situáciách, kedy útočník nemá prístup k zdrojovému kódu, napr. pri webovej aplikácii, je útočník odkázaný na dynamickú analýzu. Ako útočník, tak aj penetračný tester pri black box penetračnom teste skúma správanie danej aplikácie. Tester využíva na dynamickú analýzu ako celú sadu automatizovaných nástrojov, tak aj manuálnu inšpekciu. Bezpečnostné testovanie webových aplikácií je rozsiahlou, neustále sa vyvíjajúcou sa tematikou. Existuje niekoľko spôsobov ako efektívne testovať zabezpečenie webových aplikácií. Cez analýzu zdrojového kódu, skenovanie zraniteľností až po penetračné testovanie, je možné tieto metódy rozdeliť do dvoch všeobecnejších, vyššie uvedených kategórii. Dynamická a statická bezpečnostná analýza sú dva kľúčové prístupy k hodnoteniu bezpečnosti aplikácií, pričom každý z nich poskytuje jedinečné perspektívy[3, 26].

3.1 Dynamická vs. statická analýza

Dynamická i statická analýza majú rovnaký cieľ a to odhaliť čo najväčší počet zraniteľností. Hlavný rozdiel medzi dynamickou a statickou analýzou určuje stav, v ktorom sa testovaná aplikácia nachádza v momente vykonávania analýzy. Statická analýza je charakteristická tým, že počas tejto analýzy nedochádza k behu samotnej aplikácie, ale je skúmaný jej zdrojový kód alebo skompilovaná forma. Dynamická analýza v kontraste so statickou analýzou vyžaduje aktívny beh testovanej aplikácie. Pri dynamickej analýze, jednotlivé nástroje nemajú prístup k zdrojovým kódom. Tieto nástroje sa snažia simulovať koncového užívateľa a majú rovnaký prístup k zdrojom aplikácie ako jej potenciálny užívateľ.

Medzi výhody statickej analýzy patrí možnosť odhalenia takmer všetkých vstupov, čí výstupov webovej aplikácie. Pri náleze objaveného statickou analýzou je možné presne identifikovať miesto, kde došlo k chybe, alebo ktorý fragment kódu spôsobuje danú zraniteľnosť. To umožňuje efektívne zraniteľnosti z kódu odstraňovať. Medzi nevýhody statickej analýzy je jej veľká závislosť na programovacom jazyku, v ktorom je webová aplikácie napísaná. Aj keď existujú na trhu riešenia, ktoré sú schopné pracovať s viacerými programovacími jazykmi, táto problematika je v praxi veľmi komplexná.

Medzi výhody dynamickej analýzy patrí skutočnosť, že pre testovanie tohto typu nie je potrebný prístup k zdrojovým kódom testovanej aplikácie a skutočnosť, že dynamická analýza je nezávislá na programovacom jazyku, v ktorom bola táto aplikácia vyvinutá. Medzi nevýhody patrí ťažšia identifikácia konkrétneho fragmentu kódu, ktorý spôsobuje danú zraniteľnosť, čo spôsobí jej časovo náročnejšiu nápravu[29, 30].

3.2 Manuálna vs. automatizovaná dynamická analýza

Webové aplikácie je možné počas ich behu skúmať dvoma hlavnými spôsobmi. Manuálna i automatizovaná dynamická analýza webových aplikácií, sú aplikovateľné v mnohých testovacích scenároch, avšak výber správneho prístupu je silne závisí na povahe a zadaní testu. Oba prístupy však vyžadujú black-box prístup bez akéhokoľvek náhľadu do internej štruktúry webovej aplikácie s cieľom simulovať reálny útok. V praxi sa tak často kombinujú obidva prístupy. Manuálna analýza je úspešnejšia v odhaľovaní komplexnejších bezpečnostných nedostatkov či exploitačných scenárov, zatiaľ čo automatizovaná analýza môže byť lepšou voľbou pre opakujúce sa testy na veľmi veľkom množstve testovaných entít.

3.2.1 Automatizovaná dynamická analýza

Na rozdiel od manuálnej dynamickej analýzy, vyžaduje automatizovaný prístup programovú simuláciu užívateľskej interakcie. Bežný užívateľ je schopný otvoriť link, vypĺňať formuláre a pod. Nástroj vykonávajúci dynamickú analýzu tak musí byť schopný prechádzať stránky webovej aplikácie, identifikovať užívateľské vstupy a pod. Nástroje na dynamickú analýzu používajú celú radu techník pre mapovanie prostredia ako napr. *fuzzing* či *crawling*. Crawling je technika prechádzania stránok a odkazov v rámci stránok webovej aplikácie a je veľmi dôležitou súčasťou dynamickej analýzy. Po identifikácii vstupov a celkovom zmapovaní prostredia, začne nástroj vykonávať sadu testov na verejne známe zraniteľnosti, testovať rôzne druhy injekčných útokov, kontrolovať nájdené konfiguračné súbory a mnoho ďalších[31, 32].

3.2.2 Manuálna dynamická analýza

Najväčším problémom automatizovanej dynamickej analýzy je fakt, že v súčasnom stave techniky, nie je schopná odhaliť každú zraniteľnosť. Určité komplexnejšie zraniteľnosti, či reťazenie zraniteľností, stále vyžaduje ľudské myslenie a kreativitu. Aj keď je automatizovaná dynamická analýza silným nástrojom pre bezpečnostné testovanie, manuálny prístup je stále úspešnejší v nachádzaní zraniteľností a elimináciu falošných pozitív[33].

Manuálny prístup k dynamickej analýze silne využíva princíp zachytávania HTTP požiadaviek proxy serverom a ich následnú inšpekciu. Tento prístup umožňuje testerom skúmať HTTP prevádzku webovej aplikácie a tým skúmať správanie webovej aplikácie. Webové aplikácie okrem formulárov a odkazov obsahujú aj asynchrónne volania z jazyka JavaScript, ktoré sú volané ako reakcia na určitý užívateľský úkon ako napríklad vpisovanie kľúčového slova do vyhľadávania či scrollovanie v rámci webovej stránky. Všetky tieto akcie majú za následok HTTP požiadavku voči serveru. Každá HTTP požiadavka obsahuje niekoľko vstupov, ktoré môže tester skúmať. Zachytením požiadavky do proxy je tester schopný požiadavku modifikovať a pokúsiť sa vyvolať vo webovej aplikácii chybu a tak skúmať jej správanie na rôzne podnety. Webová aplikácia môže na nečakané vstupy reagovať rôzne. Zachytávanie HTTP prevádzky do proxy umožňuje zachytávať ako požiadavky tak odpovede. Tester je tak schopný rozpoznávať jednotlivé prejavy aplikácie na jeho podnety a identifikovať tak zraniteľnosti, chybné konfigurácie, či logické chyby. Využívanie proxy serveru pre inšpekciu HTTP prevádzky je tak neodlúčiteľnou súčasťou manuálnej dynamickej analýzy webových aplikácií[33, 32].

3.3 Metodológia dynamickej analýzy webových aplikácií

Dynamická analýza a penetračné testovanie sú dva veľmi úzko spojené pojmy. Z princípu black-box penetračného testovania webových aplikácií vyplýva, že dynamická analýza je akýmsi stavebným kameňom celého testovania. Pre vykonávanie black-box penetračného testu musí byť daná webová aplikácia spustená, buď v testovacom alebo produkčnom prostredí a tester v reálnom čas skúma jej reakcie na rôzne podnety. Existuje niekoľko verejne dostupných metodológií, ktoré slúžia ako referenčný bod pri bezpečnostnom testovaní, či vývoji webových aplikácií. Za najznámejšiu a hlavnú metodológiou pri penetračnom testovaní a dynamickej analýze je považovaná metodika OWASP Top 10[1] a ASVS (Application Security Verification Standard)[2][34].

OWASP Top 10 je zoznam desiatich kategórií, ktoré obsahujú zoznamy najbežnejších zraniteľností danej kategórie vo webových aplikáciách. Tento zoznam zostavuje organizácia OWASP (Open Web Application Security Project). OWASP je nezisková organizácia zameraná na zvyšovanie úrovne bezpečnosti webových aplikácií. Zoznam OWASP Top 10 sa pravidelne aktualizuje, aby reflektoval aktuálne trendy a hrozby v oblasti bezpečnosti aplikácií. Zoznam zahŕňa rôzne typy zraniteľností či chybných konfigurácií. Od chýb v autentizácii a autorizácii až po zraniteľnosti spôsobené chybným spracovaním užívateľských vstupov a zneužívanie verejne známych zraniteľností neaktualizovaných komponent webovej aplikácie. Tabuľka 3.1 obsahuje zoznam prvkov zoznamu OWASP Top 10 z roku 2021 a ich stručný popis[35].

ASVS (Application Security Verification Standard) je ďalším projektom organizácie OWASP. Projekt sa zameriava na poskytovanie praktických odporúčaní pre testovanie bezpečnosti webových aplikácií. ASVS ja akási množina odporúčaní a kontrolných zoznamov, ktorých účelom je poskytnúť štandard pre overovanie bezpečnosti na troch úrovniach. Vzhľadom k širokému záberu ASVS a špecifickému zameraniu implementovaného nástroja, bude nasledujúci text obmedzený na nasledujúce časti:

- V2.2 General Authenticator Security,
- V3.4 Cookie-based Session Management,
- V14.3 Unintended Security Disclosure,
- V14.4 HTTP Security Headers[36].

[1] https://owasp.org/www-project-top-ten/

[2] https://owasp.org/www-project-application-security-verification-standard/

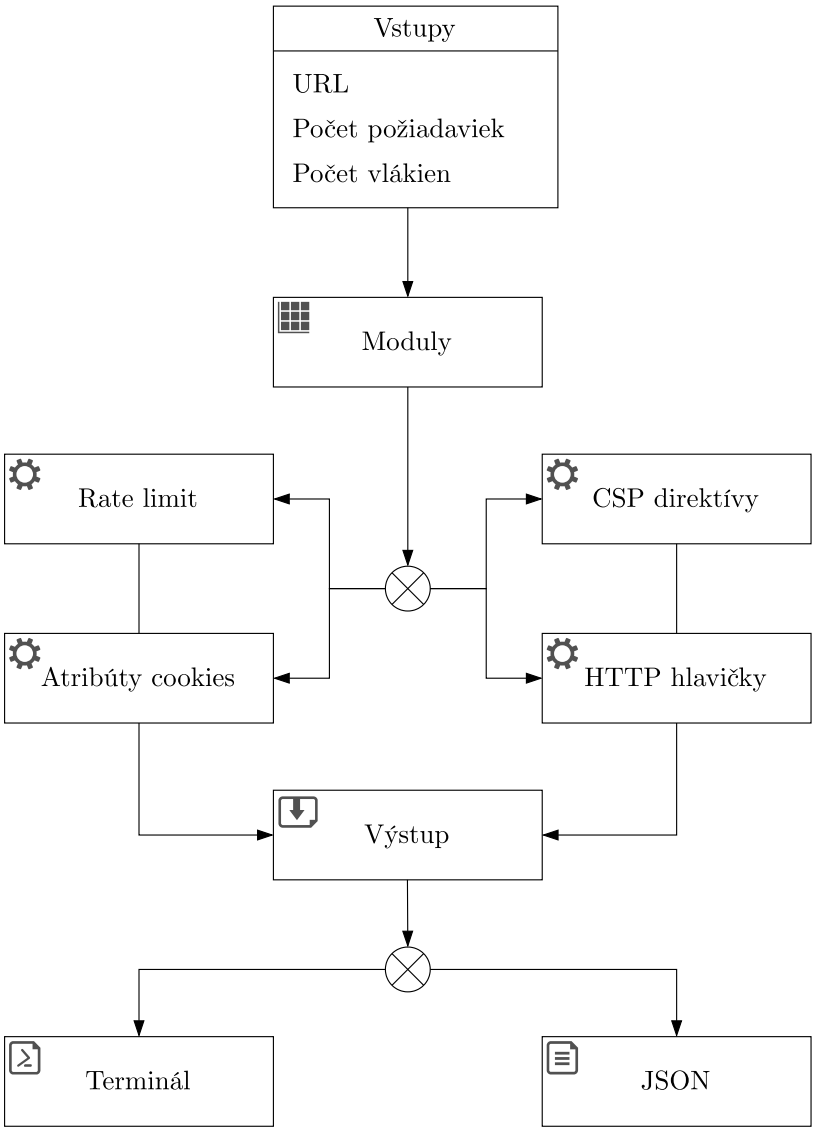
Vyššie uvedené kapitoly z dokumentu ASVS, obsahujú okrem iného kontrolu nedostatkov bezpečnostných mechanizmov z kapitoly 2, ktoré sú kontrolované nástrojom vyvíjaným v rámci praktickej časti diplomovej práce.

Tab. 3.1: OWASP Top 10

Kategória	Technické zameranie
A01:2021-Broken Access Control	Chyby v kontrole prístupu, umožňujúce neoprávnený prístup k funkcionalitám alebo údajom v systéme.
A02:2021-Cryptographic Failures	Chyby v používaní kryptografických metód, vedúce k vystaveniu citlivých údajov alebo kompromitácii systému.
A03:2021-Injection	Chyby v spracovaní vstupov, s dôrazom na injekčné útoky, kde útočník vkladá nebezpečné dáta do aplikácie.
A04:2021-Insecure Design	Riziká spojené s chybami v návrhu aplikácie, zdôrazňuje potrebu threat modelingu a bezpečného návrhu.
A05:2021-Security Misconfiguration	Chyby v konfigurácii systému alebo aplikácie, vedúce k neoprávnenému prístupu alebo iným bezpečnostným problémom.
A06:2021-Vulnerable and Outdated Components	Používanie zastaraných alebo zraniteľných komponentov, s dôrazom na riziká spojené s takýmito komponentami.
A07:2021-Identification and Authentication Failures	Chyby v identifikácii a autentifikácii používateľov, vedúce k neoprávnenému prístupu k účtom alebo funkcionalitám.
A08:2021-Software and Data Integrity Failures	Chyby spojené s predpokladmi týkajúcimi sa integrity softvéru, kritických údajov a procesov CI/CD.
A09:2021-Security Logging and Monitoring Failures	Chyby v logovaní a monitorovaní systému, obmedzujúce schopnosť odhaliť a reagovať na bezpečnostné udalosti.
A10:2021-Server-Side Request Forgery	Útoky, pri ktorých útočník prepúšťa neoprávnené požiadavky na servery zo strany servera, s rôznymi bezpečnostnými problémami.

4 Nástroj pre dynamickú analýzu

Súčasťou zamerania tejto diplomovej práce je implementácia nástroja pre dynamickú analýzu webových aplikácií. Na základe rešerše v teoretickej časti, bol implementovaný nástroj, ktorého účelom je otestovanie prítomnosti podmnožiny bezpečnostných mechanizmov webových aplikácií. V nasledujúcich kapitolách budú popísané návrh a architektúra tvoreného nástroja, implementácia jednotlivých testov, ktoré je nástroj v aktuálnom stave schopný realizovať a porovnanie s aktuálne dostupnými riešeniami. Nástroju bol pridelený pracovný názov *PtWebDA* a podporuje test prítomnosti rate limitingu, test bezpečnostných HTTP hlavičiek, test konfigurácie direktív CSP, a test bezpečnosti cookies. Obrázok 4.1 zobrazuje navrhovanú architektúru nástroja.



Obr. 4.1: Architektúra nástroja PtWebDA

4.1 Modulárny dizajn

Nástroj PtWebDA je vyvíjaný v programovacom jazyku Python vo verzii 12 a žiadna z jeho komponent nie je závislá na konkrétnom operačnom systéme, prípadne software (s výnimkou použitých externých knižníc), ktorý je potrebné inštalovať. Nástroj je konštruovaný tak, aby bolo v rámci platformy Penterep jednoduché implementovať volania jednotlivých podčastí nástroja, vzhľadom na priebeh a vývoj penetračného testu. Z toho dôvodu je nástroj rozdelený na moduly, ktoré sú volané samostatne a poskytujú jedinečné výstupy vzhľadom k predmetu testovania. Všetky moduly zdieľajú podobné prerekvizity, primárne týkajúce sa prípravy a konfigurácie, z toho dôvodu bol implementovaný bázový modul, ktorý slúži ako rodičovská, bázová trieda pre všetky moduly.

Tento návrh, okrem jednoduchej integrácie do platformy Penterep, umožňuje nekonfliktný a kontinuálny vývoj ďalších modulov a poskytuje dostatočný prehľad pri ich vývoji a nasadzovaní do produkčného prostredia. Každý modul je vo svojej podstate samostatný podprogram, ktorý nie je závislí na žiadnom z ostatných modulov. Celý nástroj PtWebDA je tak kolekciou samostatných nástrojov, skombinovaných to jednotného rozhrania v príkazovom riadku.

4.1.1 Bázový modul

Vzhľadom na charakter všetkých testovacích scenárov, všetky moduly zdieľajú prvotnú konfiguráciu. Testovaná entita je nástroju predávaná v dvoch variantách; URL danej webovej aplikácie alebo súbor, v ktorom je obsiahnutá celá HTTP požiadavka vrátanie hlavičiek a parametrov. Pri manuálnom testovaní, napr. pomocou nástroja Burp Suite[1], tak tester môže použiť už zachytenú HTTP požiadavku a použiť ju pri testovaní pomocou nástroja PtWebDA. Tento prístup je efektívny, pokiaľ daná HTTP požiadavka obsahuje značné množstvo cookies, potrebných k správnemu prijatiu požiadavky webovým serverom, alebo ak daná požiadavka obsahuje značné množstvo parametrov či dát v tele požiadavky. Medzi ďalšie konfiguračné parametre patrí konfigurácia proxy a určenie, či daný test má interagovať s testovanou entitou bezpečným spojením používajúcim TLS/SSL.

Okrem konfigurácie testovanej entity a spôsobu interakcie, bázová trieda deklaruje abstraktné metódy, ktorých implementácia je závislá na predmete testovania daného modulu. S použitím abstraktných tried a metód je tak zaručené, že všetky moduly budú vyvíjané dopredu stanoveným štandardom, čo výrazne sprehľadňuje kód celého nástroja. Kostru bázového modulu je možné pozorovať vo výpise 4.1.

[1] https://portswigger.net/burp

Výpis 4.1: Skrátený výpis kódu bázovej triedy

```python
class BaseModule[T](ABC):
    def __init__(
        self,
        target: str | None,
        request_file_path: str | None,
        proxy: str | None,
        https: bool = True,
    ) -> None:
        # Konštruktor slúžiaci na konfiguráciu daného testu
        .
        .
    # Privátne pomocné funkcie bázového modulu
    .
    .
    @abstractmethod
    def run(self) -> bool:
        ...

    @abstractmethod
    def print_info(self) -> None:
        ...

    @abstractmethod
    def test(self) -> T | None:
        ...

    @abstractmethod
    def print_results(self) -> None:
        ...

    @abstractmethod
    def json(self) -> str | None:
        ...

    @staticmethod
    @abstractmethod
    def add_subparser(subparsers: argparse._SubParsersAction) ->
   None:  # type: ignore
        ...
```

Tab. 4.1: Tabuľka popisujúca metódy implementované jednotlivými modulami

Metóda	**Popis**
`run`	Hlavná rutina modulu, ktorá volá metódy vykonávajúce samotný test.
`print_info`	Funkcia zobrazuje informácie o konfigurácii penetračnému testerovi v príkazovom riadku.
`test`	Funkcia vykonávajúca test.
`print_results`	Funkcia zobrazuje informácie o výsledkoch testu penetračnému testerovi v príkazovom riadku.
`json`	Funkcia serializuje výsledky testu namapované na jedinečné identifikátory zraniteľností do JSON štruktúry kompatibilnej s platformou Penterep.
`add_subparser`	Metóda zabezpečujúca parsovanie parametrov z príkazového riadku pre daný modul.

Každý modul dedí z vyššie uvedeného bázového modulu a implementuje všetky povinné metódy označené direktívou `@abstractmethod`. Tieto metódy je možné rozdeliť do niekoľkých kategórii a to metódy pre beh testu, metódy pre zobrazovanie priebehu testu v príkazovom riadku a na metódy zabezpečujúce kompatibilitu a integráciu do platformy Penterep. Tabuľka 4.1.1 obsahuje stručný popis funkcionality jednotlivých metód.

Návratovou hodnotou metódy `test` každého modulu je dátová štruktúra reprezentujúca surové výsledky testu. Táto štruktúra je špecifická pre každý individuálny test, preto bolo nutné použiť pri tvorení bázového modulu okrem abstraktných tried aj generický dátový typ `T`. Generický dátový typ, ktorý je vrátený abstraktnou metódou `test` opäť zabezpečuje prehľadnosť a konzistenciu kódu.

4.2 Implementované moduly

V nasledujúcich podkapitolách budú popísané jednotlivé moduly, ktoré sú v rámci súčastného stavu vývoja nástroja PtWebDA funkčné a otestované. Jednotlivé moduly, či testy, sú implementované ako samostatné triedy s definovanými atribútmi a metódami. Každá trieda má okrem pomocných metód metódu `run`, ktorá je hlavnou rutinou daného testu. Parametre každého testu sú definované v konštruktoch daných testov. V nasledujúcom texte budú popísané samotné testy či moduly, ich mechanizmus a implementácia v jazyku Python. Moduly boli v rámci vývoja testované lokálne v pripravenom, kontrolovanom prostredí vo virtuálnom stroji s operač-

ným systémom *Linux Ubuntu*[2] a vyvinutou zraniteľnou webovou aplikáciou v jazyku Python s použitím frameworku *Flask*[3].

4.2.1 Rate limit

Testovanie rate limitingu v rámci dynamickej analýzy spočíva v zostavení modelu, ktorého cieľom bude vyhodnocovať reakcie webovej aplikácie na značný počet požiadaviek v stanovenom čase. Skutočnosť, že webová aplikácia alebo webový server skutočne disponuje mechanizmom obmedzovania užívateľských požiadaviek, nemusí byť však okamžite jednoznačná. Pre nástroj vyhodnocujúci túto skutočnosť je značnou výzvou diferencovanie medzi spomalením či úplným zastavením požiadaviek skutočným mechanizmom obmedzovania užívateľských požiadaviek a nedostatočnými výpočtovými či pamäťovými zdrojmi testovaného systému. Pred predstavením implementovaného modelu, je potrebné poukázať na všeobecný, matematický princíp rate limitingu.

Matematický aparát pre kontrolu podmienky spĺňajúcej preddefinované pravidlá obmedzovania počtu požiadaviek v časovom intervale $[t_1, t_2]$ môžeme vyjadriť nasledujúcim vzťahom:

$$\int_{t_1}^{t_2} R(t)\,dt \leq L \times (t_2 - t_1) \tag{4.1}$$

kde,

$R(t)$ predstavuje rýchlosť HTTP odpovedí v čase t,
L predstavuje maximálny počet požiadaviek za minútu,
t_1, t_2 predstavujú časový interval samotného testu.

Z nerovnice 4.1 je možné vyčítať, že rate limit je definovaný vzťahom, kde počet úspešne odoslaných požiadaviek a prijímaných odpovedí musí byť menší alebo rovný maximálnej povolenej frekvencii odosielania požiadaviek L. Pri testovaní rate limitingu však prístup k informácii o rate limite L nie je dostupná, preto je pre algoritmus testovania rate liminigu nerovnica 4.1 nepoužiteľná. Slúži však ako základ pre vytvorenie vhodnej metodológie. Pre testovanie rate limitingu bez prístupu k informácii o rate limite L je potrebné zaviesť ďalšiu premennú a to počet zamietnutých či chybových HTTP požiadaviek. Vzťah medzi úspešnými a neúspešnými HTTP požiadavkami je možné vyjadriť nasledujúcim vzťahom:

$$\int_{t_1}^{t_2} F(t)\,dt < \frac{1}{E} \int_{t_1}^{t_2} R(t)\,dt \tag{4.2}$$

[2]https://ubuntu.com/download/desktop
[3]https://flask.palletsprojects.com/en/3.0.x/

kde,

$R(t)$ predstavuje rýchlosť úspešných HTTP odpovedí v čase t,
$F(t)$ predstavuje rýchlosť zamietnutých HTTP odpovedí v čase t,
E predstavuje maximálnu akceptovateľnú chybovosť HTTP požiadaviek,
t_1, t_2 predstavujú časový interval samotného testu.

Daný výraz môžeme upraviť nasledovným spôsobom.

$$\frac{\int_{t_1}^{t_2} F(t)\,dt}{\int_{t_1}^{t_2} R(t)\,dt} < \frac{1}{E} \qquad \frac{1}{E} \simeq T \tag{4.3}$$

kde,

T predstavuje hraničný pomer neúspešných a úspešných HTTP požiadaviek.

Z vyššie odvodených vzťahov vyplýva, že úspešnosť testu bez znalosti rate limitu L značne závisí od nájdenia vhodného hraničného pomeru T. Určiť tento pomer nie je však triviálna úloha. Nastavenie rate limitu vo webových aplikáciách značne záleží od ich funkcionality. Kladie sa dôraz na to, či je testovaný koncový bod prihlasovací formulár, stránka zobrazujúca statické dáta alebo stránka zobrazujúca dynamický obsah. Každý prípad vyžaduje starostlivé nastavenie. Pre testovanie to znamená, že výber vhodného hraničného pomeru T zavisí na samotnom testerovi, prípadne nástroji, ktorý bude testovanie vykonávať. Chýbajúci univerzálny spôsob výberu vhodného hraničného pomeru T pre túto prácu znamená vytvorenie vlastnej metodológie na základne dostupných informácií a tzv. *best practices*.

Z analýzy *best practices* od spoločností ako napríklad CloudFare[37] vyplýva, že pre efektívne uplatnenie rate limitingu sa používajú relatívne nízke hodnoty maximálneho počtu požiadaviek za stanovený čas. Efektívna obrana pred útokmi na prihlasovacie formuláre webových aplikácií tak môže predstavovať rate limit o hodnote napríklad 4 požiadavky za minútu. Vzhľadom k nízkym hodnotám používaných v praxi, je možné vzťah 4.2 zjednodušiť nasledovným spôsobom:

$$\int_{t_1}^{t_2} F(t)\,dt < \frac{1}{E}\int_{t_1}^{t_2} R(t)\,dt, \quad kde \quad \frac{1}{E} = 1 \quad a \quad T = 1 \tag{4.4}$$

z čoho vyplýva,

$$\int_{t_1}^{t_2} F(t)\,dt < \int_{t_1}^{t_2} R(t)\,dt \tag{4.5}$$

Po zjednodušení bude nástroj brať hraničný pomer neúspešných a úspešných HTTP požiadaviek T ako 1:1. Pre nástroj to tak v praxi znamená, že počet chybných či zahodených požiadaviek, nesmie počas testu prekročiť počet úspešných požiadaviek. Pokiaľ sa tak stane, test vyhodnotí prítomnosť rate limitingu. Ak test vyhodnotí prítomnosť rate limitingu, nerovnica 4.5 musí nadobudnúť nasledujúcu formu rovnice:

$$\int_{t_1}^{t_2} F(t)\,dt = \int_{t_1}^{t_2} R(t)\,dt \tag{4.6}$$

Detekcia rate limitingu a počet chybných požiadaviek a úspešných požiadaviek poskytujú dostatočné množstvo informácií pre vyriešenie nerovnice 4.1 a umožňujú tak vypočítať teoretickú hodnotu maximálneho počtu požiadaviek L za stanovený časový interval.

Implementácia v jazyku Python

Nástroj v rámci testovania rate limitingu využíva vyššie navrhovaný matematický model. Nakoľko rate limit môže byť implementovaný v desiatkach až tisíckach požiadaviek za stanovený čas a interakcia s webovým serverom je I/O operácia, nástroj využíva multivláknový prístup k efektívnemu odosielaniu HTTP požiadaviek webovému serveru. K tomu bola využitá Python knižnica *concurrent.futures*[4]. Pri spúšťaní modulu má užívateľ možnosť špecifikovať počet vlákien, pričom východzia hodnota počtu vlákien je nastavená na 10, avšak je na uvážení testera, akú intenzitu zvolí. Výpis 4.4 nižšie obsahuje hlavnú rutinu testu, ktorá inicializuje všetky vlákna, zbiera ich výsledky a analyzuje pomer úspešných a neúspešných požiadaviek. Dôležitým prvkom rutiny zobrazenej vo výpise 4.4 je návratová hodnota. Dátová štruktúra `RateLimitResult` je zobrazená vo výpise 4.2.

Výpis 4.2: Štruktúra reprezentujúca návratovú hodnotu testu rate limitingu

```
1 class RateLimitResult(NamedTuple):
2     rate_limited: bool
3     request_threshold: int | None
```

Návratová hodnota popisuje, či bol detekovaný rate limiting a v prípade pozitívneho nálezu, približnú hodnotu počtu úspešných HTTP požiadaviek, ktoré boli odoslané pred aplikovaním pravidiel charakteristických pre daný systém obmedzovania požiadaviek. Táto hodnota je pre penetračného testera veľmi užitočná, nakoľko

[4]https://docs.python.org/3/library/concurrent.futures.html

indikuje maximálnu rýchlosť, či záťaž, akú môže tester na webovú aplikáciu vyvinúť bez toho, aby došlo k obmedzeniu zo strany servera a tak znemožneniu, či skomplikovaniu jeho práce.

Pri samotnom testovaní dochádza k paralelnému spusteniu parametrom definovaného počtu vlákien, ktoré volajú internú metódu `_make_request`, ktorá zabezpečuje interakciu s webovým serverom a interpretuje jeho odpovede. Dôležitú časť kódu tejto funkcie je možné pozorovať vo výpise 4.3. Hlavným princípom tejto metódy, je vyhodnocovať úspešnosť danej požiadavky a aktualizovať interné čítače, ktoré sú neskôr využívané k indikácii prítomnosti rate limitingu.

Výpis 4.3: Interná metóda `_make_request`

```
try:
    start_time = time.time()

    # Send the final prepared request in the constructor
    response = requests.Session().send(
        self.prepared_request.prepare(), proxies=self.proxies,
   verify=self.verify
    )

    end_time = time.time()
    response_time = (end_time - start_time) * 1000
    status_code: int = response.status_code

    if status_code == 509 or status_code == 429:
        self.failed_req += 1
    else:
        self.success_req += 1
    return Response(status_code, response_time)
except requests.exceptions.RequestException:
    self.failed_req += 1
    return None
```

K evaluácii interného stavu dochádza práve v hlavnej metóde samotného testu a teda v metóde `test`, kde dochádza k inicializácii vlákien a kontinuálnej evaluácii interných premenných, ktoré sú využívané k testovaniu vzťahu definovaného v rovnici 4.6. Časť kódu hlavnej metódy `test` a spôsob evaluácie interného stavu je možné pozorovať vo výpise 4.4. Po ukončení testu je metódou vrátená dátová štruktúra popísaná vo výpise 4.2, ktorá je neskôr využívaná k evaluácii výsledkov a ich serializácii pre zaistenie kompatibility s platformou Penterep alebo následnému zobrazeniu penetračnému testerovi do príkazového riadku.

Výpis 4.4: Multivlánkové testovanie rate limitingu v metóde `test`

```
with ThreadPoolExecutor(max_workers=self.num_threads) as executor:
    self.futures = [
        executor.submit(self.__make_request) for _ in range(self.
    total_requests)
    ]

    for future in concurrent.futures.as_completed(self.futures):
        .
        .
        result = future.result()

        if result is None:
            continue

        if self.failed_req > self.success_req:
            for future in self.futures:
                future.cancel()

            return RateLimitResult(True, self.success_req)

        self.meta_resutls.append(result)

    concurrent.futures.wait(self.futures)

return RateLimitResult(False, None)
```

4.2.2 HTTP hlavičky

Testovanie prítomnosti HTTP hlavičiek spočíva v zachytení odpovede na požiadavku voči webovému serveru. Test s využitím Python knižnice requests[5] vykoná HTTP požiadavku s metódou GET na špecifikovanú URL, ktorá bola testu predaná formou parametru pri inicializácii objektu samotného testu. Aj keď je tento test vo svojej podstate veľmi jednoduchý a jednoducho implementovateľný, sila nálezov, ktoré môže poskytnúť, môžu mať značný dopad na celkové zabezpečenie webovej aplikácie či webového serveru. HTTP hlavičky, ktoré sú predmetmi testu, sú rozdelené podľa kapitoly 2.2 do nasledovných troch kategórií, ktorých konkrétne hodnoty je možné pozorovať vo výpise 4.5:

- Hlavičky poskytujúce potenciálne citlivé informácie,
- Chýbajúce bezpečnostné hlavičky,

[5] https://pypi.org/project/requests/

- Hlavičky poskytujúce informácie o caching[6] mechanizme webového serveru[7].

Výpis 4.5: HTTP hlavičky určtené k testovaniu

```
INFO_HEADERS: list[str] = [
    "server",
    "x-powered-by",
    "x-aspnet-version",
    "x-aspnetmvc-version",
]

MISSING_HEADERS: list[str] = [
    "content-security-policy",
    "x-frame-options",
    "x-content-type-options",
    "referrer-policy",
    "strict-transport-security",
    "permissions-policy",
]

CACHE_HEADERS: list[str] = [
    "cache-control",
    "pragma",
    "last-modified",
    "expires",
    "etag",
]
```

Podobne ako pri teste rate limitingu v kapitole 4.2.1, hlavná metóda testu má ako svoju návratovú hodnotu dátovú štruktúru reprezentujúcu výsledky testu. Štruktúra primárne pozostáva z troch zoznamov reprezentujúcich vyššie uvedené tri kategórie sledovaných HTTP hlavičiek. Jednotlivé hlavičky sú reprezentované triedou `Header`, ktorá v sebe obsahuje názov hlavičky, kód reprezentujúci zraniteľnosť v platforme Penterep[8], a hodnotu danej hlavičky. Dátové štruktúry `Header` a `HeadersResults` je možné pozorovať vo výpise 4.6.

[6]Caching vo webových aplikáciách je proces ukladania dočasnej kópie dát alebo stránok na klientovi alebo serveri, aby sa zvýšila rýchlosť a výkonnosť aplikácie. Tento mechanizmus umožňuje znížiť množstvo dátového prenosu a znižuje záťaž na server, čím zlepšuje celkový užívateľský zážitok.

[7]Aj keď tieto hlavičky nie sú zakomponované do platformy Penterep ani hlásené ako zraniteľnosť, pretože priamo nepredstavujú žiadnu zraniteľnosť a ich správna interpretácia vyžaduje širší kontext ako je tento test schopný poskytnúť, je prítomnosť týchto hlavičiek testerovi dostupná vo výstupe do príkazového riadku, nakoľko môžu byť pre testera stále užitočné.

[8]Bližšie informácie k týmto kódom je možné nájsť v kapitole 4.4.

Výpis 4.6: Dátové štruktúry `Header` a `HeadersResults`

```
class Header(NamedTuple):
    name: str
    code: str | None
    value: str | None

class HeadersResults(NamedTuple):
    missing_headers: list[Header]
    headers_leaking_info: list[Header]
    cache_headers: list[Header]
```

Test začína odoslaním GET požiadavky na URL špecifikovanú pri vytváraní objektu testu. Odpoveď na túto požiadavku je následne zachytená a analyzovaná pomocou knižnice Python requests. Iteráciou cez HTTP hlavičky obsiahnuté v odpovedi, prebehne normalizácia týchto hlavičiek. Niektoré webové aplikácie, či frameworky, nedodržujú rovnaký štandard veľkých a malých písmen. Normalizáciou všetkých hlavičiek na malé písmená, je možné predísť preskočeniu potenciálnemu nálezu. Proces normalizácie prijatých HTTP hlavičiek je možné pozorovať vo výpise 4.7.

Výpis 4.7: Normalizácia prijatých HTTP hlavičiek

```
# Send the final prepared request in the constructor
response: requests.Response = requests.Session().send(
    self.prepared_request.prepare(), proxies=self.proxies, verify=
   self.verify
)
.
.
# Normalize the response headers to lowercase
for key, value in response.headers.items():
    lowercase_headers[key.lower()] = value
```

Po normalizácii prijatých HTTP hlavičiek, je program pripravený porovnávať jednotlivé hlavičky. Porovnávanie hlavičiek prebieha v troch samostatných cykloch. Pri porovnávaní chýbajúcich hlavičiek program upozorňuje užívateľa, ktoré hlavičky chýbajú. Avšak, pri hlavičkách, ktoré potenciálne poskytujú citlivé informácie o prostredí webovej aplikácie a pri hlavičkách týkajúcich sa konfigurácie caching mechanizmu webovej aplikácie, poskytuje test užívateľovi obsah týchto hlavičiek vo forme konzolového výstupu, pre ich hlbšiu, manuálnu analýzu. Analýzu HTTP hlavičiek je možné pozorovať vo výpise 4.8. Test porovnávaním HTTP hlavičiek končí a dochádza k spracovaniu výsledkov. Po ukončení testu, je hlavnou metódou testu vrátená

dátová štruktúra popísaná vo výpise 4.6, ktorá je podobne ako pri teste rate limitingu v kapitole 4.2.1 neskôr využívaná k evaluácii výsledkov a ich následnému zobrazeniu penetračnému testerovi alebo serializácii pre výstup kompatibilný s platformou Penterep.

Výpis 4.8: Porovnávanie chýbajúcich a prítomných HTTP hlavičiek

```
# Collect headers that are missing and should be implemented
for header in MISSING_HEADERS:
    if header not in lowercase_headers:
        res_missing_headers.append(Header(header, PT_VULN_CODES[
    header], None))

# Collect headers that are present and potentially contain
    sensitive infomarion
for header in INFO_HEADERS:
    if header in lowercase_headers:
        res_headers_leaking_info.append(
            Header(header, PT_VULN_CODES[header], lowercase_headers
    [header])
        )
# Collect headers that are present and potentially contain useful
    caching information
for header in CACHE_HEADERS:
    if header in lowercase_headers:
        res_cache_headers.append(Header(header, None,
    lowercase_headers[header]))
```

4.2.3 Analýza direktív CSP

Modul analýzy direktív CSP funguje na podobnom princípe ako testovanie bezpečnostných HTTP hlavičiek. Jedná sa o interakciu s webových serverom na báze zaslania HTTP požiadavky a následnej analýze odpovede webového servera. Webové aplikácie môžu implementovať direktívy dvoma spôsobmi; Obsah direktív CSP je obsiahnutý v špecifických HTTP hlavičkách alebo sú direktívy špecifikované priamo v rámci HTML kódu danej webovej stránky. Pri analýze CSP sa sledujú pravidlá a obmedzenia, ktoré tieto direktívy stanovujú pre zdroje, s ktorými môže webová stránka komunikovať. To zahŕňa určenie povolených zdrojov pre skripty, štýly, obrázky, média a ďalšie aktivity vykonávané na webovej stránke. Samotný modul analýzy CSP vyhodnocuje, či sú tieto direktívy správne nastavené a či neexistujú žiadne nebezpečné alebo nedostatočné konfigurácie, ktoré by mohli umožniť útoky typu

XSS, CSRF a iné bezpečnostné hrozby. V prípade detekcie nebezpečnej konfigurácie, nástroj vyhodnotí nález v zmysle nedostatočnej, či nebezpečnej konfigurácie Content Security Policy direktív a tieto nebezpečné direktívy sú penetračnému testerovi zobrazené v rozhraní príkazového riadku.

Modul, podobne ako v predošlých testoch, obsahuje hlavnú metódu testu s názvom `test`. Táto metóda zasiela webovému serveru HTTP požiadavku, ktorá je následne analyzovaná vyššie uvedenými dvoma spôsobmi. Existujú dva hlavné zdroje CSP direktív; HTTP hlavičky a HTML kód webovej stránky. Pre uloženie výsledkov testu je opäť vytvorená dátová štruktúra, obsahujúca zoznamy CSP direktív rozdelených do vyššie uvedených kategórií. K reprezentácii CSP direktív bola vytvorená pomocná dátová štruktúra `CSPDirective`, ktorá obsahuje názov direktívy, jej obsah, a indikáciu, či sa jedná o nebezpečnú konfiguráciu alebo nie. Táto dátová štruktúra je neskôr využívaná k tvorbe vyššie uvedenej dátovej štruktúry `CSPResult`. Tieto dátové štruktúry je možné pozorovať vo výpise 4.9.

Výpis 4.9: Dátová štruktúra `CSPDirective`

```
class CSPDirective(NamedTuple):
    name: str
    content: str
    dangerous: bool

class CSPResult(NamedTuple):
    csp_headers: list[CSPDirective] | None
    csp_html: list[CSPDirective] | None
```

Samotné testovanie, ako bolo uvedené v úvode tejto podkapitoly, spočíva v analýze odpovede webového servera. Pred samotnou analýzou je však potrebné dané direktívy z odpovede webového servera extrahovať. O extrakciu CSP direktív z vyššie uvedených zdrojov sa starajú dve interné metódy ktorých sa každá venuje práve jednému zdroju. Pri analýze HTTP hlavičiek sú kontrolované tri možné deklarácie CSP; content-security-policy, x-content-security-policy, a x-webkit-csp. Aj keď sú hlavičky *x-content-security-policy* a *x-webkit-csp* už zastaralé, niektoré staršie systémy a webové prehliadače ich stále podporujú a dôslednosť tohto modulu nie je limitovaná len na najnovšie technológie. Pri testovaní direktív CSP v HTML kóde webovej stránky je použitá knižnica beautifulsoap4[9], ktorá je využívaná na parsovanie HTML kódu a na extrakciu žiadúcich HTML tagov. V prípade CSP sú jej direktívy definované v HTML tagu `<meta>` a príklad definície CSP v HTML kóde je možné pozorovať vo výpise 4.10.

[9]https://pypi.org/project/beautifulsoup4/

Výpis 4.10: Konfigurácia CSP v HTML kóde

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'">
```

Hlavná metóda testu, okrem zasielania požiadavky, musí odpoveď pred analýzou vhodne pripraviť. V prípade hľadania direktív CSP v HTML kóde webovej stránky, je potrebné najskôr pomocou vyššie uvedenej knižnice zparsovať odpoveď webového servera. K tomu je využitá dátová štruktúra `Html`, ktorá v rámci svojho konštruktora, vytvorí objekt typu `BeautifulSoup`, pomocou ktorého je neskôr možné pristupovať k jednotlivým elementom HTML kódu. Túto štruktúru je možné pozorovať vo výpise 4.11. Útržok kódu funkcie `test`, ktorá využíva vyššie uvedené princípy a dátové štruktúry, je možné pozorovať vo výpise 4.12.

Výpis 4.11: Dátová štruktúra `Html`

```
class Html:
    def __init__(self, url: str, content: str) -> None:
        self.url: str = url
        self.content = content
        self.soup_body = BeautifulSoup(self.content, "html.parser")
```

Výpis 4.12: Hlavná metóda testu direktív CSP

```
try:
    response: requests.Response = requests.Session().send(
        self.prepared_request.prepare(), proxies=self.proxies,
    verify=self.verify
    )
    .
    .
    r = Response(response.headers, Html(response.url, response.text
    ))

    csp_headers = self._check_csp_headers(r)
    csp_html = self._check_csp_html(r)
except requests.exceptions.RequestException as e:
    Log.error(f"Error occurred: {e}")
    return None

return CSPResult(csp_headers, csp_html)
```

Po extrakcii CSP direktív zo všetkých zdrojov, dochádza k analýze a evaluácii týchto direktív. Funkcia `_eval_directives` vyhodnocuje CSP direktívy na základe prítomnosti špecifických kľúčových slov, ktoré indikujú prítomnosť nesprávne definovanej politiky CSP. Na základe výskumu vykonanom v tejto diplomovej práci boli zvolené nasledujúce testovacie scenáre:

- Prítomnosť kľúčového slova „unsafe“,
- Prítomnosť tzv. *wildcard*[10],
- Neprítomnosť direktívy „object-src“,
- Neprítomnosť direktívy „default-src“,
- Neprítomnosť direktívy „base-uri“.

Prítomnosť kľúčového slova „unsafe“

Kľúčové slovo *unsafe* môže byť súčasťou definície viacerých direktív. Môže sa jednať napr. o zdroje skriptov `script-src` či zdroje štýlov `style-src` a pod. Použitie direktívy obsahujúcej kľúčové slovo „unsafe“ zvyčajne znamená, že webová aplikácia povoľuje určité nebezpečné operácie, ako sú inline skripty[11], použitie `eval()` funkcie a pod.

Wildcard

Prítomnosť *wildcard* typicky označuje veľmi permísívne možnosti načítania zdrojov v rámci webovej aplikácie. V prípade použitia wildcard napríklad v direktíve `script-src` tak útočník nie je takmer vôbec limitovaný, nakoľko webová aplikácia povoľuje spúšťať akýkoľvek JavaScript kód z akéhokoľvek zdroja, čo je pre útočníka veľmi žiadúca situácia.

Neprítomnosť direktív „object-src“ a „default-src“

Direktíva *object-src* definuje možné zdroje obsahu pre HTML elementy `<object>`, `<embed>`, a `<applet>`. Táto direktíva spoločne s direktívou *default-src* sú v praxi často zabúdané a je možné ich neprítomnosť zneužiť k úspešnému útoku XSS a spusteniu ľubovoľného JavaScript kódu aj pri prísnej konfigurácii direktívy *scrip-src*.

Neprítomnosť direktívy „base-uri“

Direktíva *base-uri* definuje základnú URI, ktorá bude použitá pre všetky relatívne URL adresy v dokumente. Pokiaľ webová aplikácia načítava JavaScript súbory s relatívnou cestou, je možné túto direktívu zneužiť zmenou bázovej URI na útočníkovú

[10] Wildcard je typicky reprezentovaný symbolom „*“.

[11] Inline skripty sú skripty, ktoré sú definované priamo v HTML kóde v tagoch `<script>`.

URI a načítať tak skripty s rovnakým menom z webového serveru útočníka.

Obchádzanie bezpečnostných direktív CSP

Nesprávne definovanie direktív CSP môže v prípade prítomnosti zraniteľnosti XSS umožniť obídenie tohto bezpečnostného nastavenia a tak úspešne zneužiť zraniteľnosť XSS. Pre penetračného testera, či útočníka, je práve z toho dôvodu veľmi dôležité identifikovať nedostatočnú, či príliš permisívnu, konfiguráciu CSP, aby tak mohol aplikovať vhodnú techniku obchádzania direktív CSP. Existuje celá rada rôznych techník, ktoré môže tester k obchádzaniu CSP použiť a ich popis je nad rámec tejto práce. Zoznam rôznych techník použiteľných v praxi[12], je možné skúmať na webových stránkach ako napr. *Hacktricks*[13].

4.2.4 Analýza atribútov cookies

Modul pre analýzu atribútov cookies je posledným modulom, ktorý je v rámci tejto diplomovej práce a aktuálneho stavu nástroja dostupný a funkčný. Principiálne je modul opäť podobný modulom analýzy CSP a HTTP hlavičiek. Jedná sa teda primárne o zasielanie HTTP požiadavky a analýzy odpovede webového servera. Samotná analýza sa však líši. Cookies nastavované webovým serverom sa do webového prehliadača klienta dostanú prostredníctvom HTTP hlavičky *Set-Cookie* a práve tieto hlavičky sú pri odpovedi webového servera pozorované. Príklad HTTP odpovedí, ktorá nastavuje príslušné cookies je možné pozorovať vo výpise 4.13.

Výpis 4.13: Ukážka HTTP odpovede obsahujúca nastavenie cookies

```
1 HTTP/1.1 302 FOUND
2 Content-Type: text/html; charset=utf-8
3 Location: /
4 Set-Cookie: username=JohnDoe; Path=/
5 Set-Cookie: session=9f86d081884c7d659a2feaa0c; Max-Age=3600; Path=/
6 Date: Sat, 01 May 2024 12:00:00 GMT
```

HTTP cookies, ako bolo popísané v kapitole 2.3.2, môžu mať nastavených hneď niekoľko atribútov. Nastavenie týchto atribútov je možné pozorovať aj vo vyššie uvedenom výpise 4.13, kde sa nastavujú cookies `username`, `session` a ich atribúty. Pri

[12]Zneužiteľnosť direktív CSP je značne ovplyvnená spôsobom, akým dochádza k zraniteľnosti CSP. Tester tak musí veľmi dobre poznať testované prostredie a neexistuje univerzálny spôsob, ako obísť konfiguráciu CSP.

[13]https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

nastavovaní cookies je potrebné dbať na ich funkcionalitu, nakoľko nie všetky cookies vyžadujú prísne nastavenie. Penetračný tester tak musí rozoznať, ktorý cookie má akú funkcionalitu a na základe toho interpretovať závažnosť daných nálezov.

Testovanie prítomnosti bezpečnostných atribútov cookies začína analýzou prítomných HTTP hlavičiek. Medzi hlavičkami vrátenými webovým serverom, ako bolo uvedené vyššie, musí byť prítomná hlavička `Set-Cookie`. Pre hlavnú metódu `test` je opäť definovaná jej návratová hodnota `CookieResult`, ktorá v sebe obsahuje zoznam objektov typu `Cookie`. Objekt reprezentujúci cookie sa skladá z jeho mena, atribútov a indikácie, či je jeho konfigurácia bezpečná alebo nie. Vo výpise 4.14 je možné pozorovať štruktúry, ktoré dohromady tvoria návratovú hodnotu testu bezpečnostných atribútov cookies.

Výpis 4.14: Štruktúry tvoriace návratovú hodnotu testu cookies

```
class SameSiteAttr(NamedTuple):
    present: bool
    value: str | None

class Cookie(NamedTuple):
    name: str
    secure: bool
    http_only: bool
    same_site: SameSiteAttr
    path: str | None
    domain: str | None
    dangerous: bool

class CookieResult(NamedTuple):
    cookies: list[Cookie] | None
```

Po sprarsovaní HTTP hlavičiek odpovede webového servera a extrakcii definícií cookies, dochádza k ich analýze. Pri jednotlivých cookies sa kontrolujú bezpečnostné atribúty uvedené a popísané v kapitole 2.3.2. Nález prítomnosti, či neprítomnosti danej bezpečnostnej konfigurácie je zároveň vyhodnotený ako nebezpečný a uložný ako boolean hodnota premennej `dangerous`, ktorá je následne využitá k vytvoreniu dátovej štruktúry `Cookie` uvedenej vo výpise 4.14. Proces analýzy bezpečnostných atribútov cookies je relatívne jednoduchý. Zložitejšia je však ich interpretácia. Webové aplikácie často využívajú vlastné, neštandardizované názvy cookies a pokus o identifikáciu a oddelenie relačných a prevádzkových cookies je tak značne zťažený a jeho programová implementácia by tak bola do veľmi veľkej miery nespoľahlivá. Z toho dôvodu sú výsledky prezentované penetračnému testerovi v podobe konzo-

lového výstupu so zvýraznenými nebezpečnými konfiguráciami. Táto časť nástroja je tak do istej miery len pomôckou pre penetračného testera a samotná evaluácia nálezov je tak na jeho úsudku a osobných skúsenostiach. Vo výpise 4.15 je možné pozorovať úryvok kódu analyzujúceho bezpečnostné atribúty cookies.

Výpis 4.15: Úryvok kódu analyzujúceho atribúty cookies

```
for cookie in cookies:
    attribs = cookie.split("; ")
    attribs = [attr.lower() for attr in attribs]
    name = attribs[0].split("=")[0].strip()

    if "secure" in attribs:
        secure = True
    else:
        dangerous = True

    if "httponly" in attribs:
        http_only = True
    else:
        dangerous = True

    for attr in attribs:
        if "samesite" in attr:
            same_site = True
            same_site_value = attr.split("=")[1].strip()
        else:
            dangerous = True

    all_cookies.append(
        Cookie(
            name,
            secure,
            http_only,
            SameSiteAttr(same_site, same_site_value),
            path,
            domain,
            dangerous,
        )
    )
```

4.3 Testovanie nástroja

V nasledujúcej kapitole bude popísané testovacie prostredie a jeho vývoj. K vyvíjaniu a následnému testovaniu nástroja PtWebDA bola vyvinutá zraniteľná webová aplikácia obsahujúca zraniteľnosti, ktoré sú relevantné pre testovanie funkcionality nástroja. Súčasťou kapitoly sú tiež testy jednotlivých modulov, ich používanie a ich výstupy. Nástroj bol okrem lokálneho testovania v kontrolovanom prostredí testovaný aj v praxi pri penetračnom testovaní webových aplikácií pre reálnych klientov. Záverom kapitoly budú predstavené výsledky experimentálneho testovania v kontrolovanom prostredí a popis a výsledky testovania nástroja v produkčnom komerčnom prostredí pri realizácii penetračných testov pre reálnych klientov.

4.3.1 Vývoj zraniteľnej webovej aplikácie

Pre otestovanie nástroja PtWebDA bola vyvinutá jednoduchá webová aplikácia. Táto webová aplikácia obsahuje štyri koncové body, či "routy", kde má každý z nich iné nastavenie rate limitingu, HTTP hlavičiek a cookies. Hlavným cieľom testovania je preskúmať reaktivitu nástroja na jednotlivé bezpečnostné nedostatky a jeho efektivitu pri ich prezentovaní penetračnému testerovi. Webová aplikácia je naprogramovaná v programovacom jazyku Python s použitím webového frameworku Flask[14]. Webová aplikácia je spúšťaná vo virtuálnom stroji s použitím virutalizačného prostredia VMware Workstation Pro 17[15] na operačnom systéme Ubuntu 22.04.3 LTS.

Rate limiting

Prvým krokom pri vývoji zraniteľnej webovej aplikácie bolo definovať, na ktoré koncové body sa budú aplikovať aké pravidlá obmedzovania požiadaviek. Webová aplikácia obsahuje celkovo štyri koncové body. Koncový bod `/`, ktorý nemá nastavený žiadny rate limiting, koncový bod `/login`, ktorý emuluje prihlasovanie užívateľov, má nastavený prísny rate limit 4 požiadavky za minútu. Koncový bod `/menu`, ktorý reprezentuje bežnú stránku webovej aplikácie, má nastavený rate limit na 30 požiadaviek za minútu. Posledný koncový bod `/api`, ktorý má nastavený žiadny rate limit na 100 požiadaviek za minútu. Tieto limity boli vybrané na základe výskumu v teoretickej časti v kapitole 2.1 a v praktickej časti v kapitole 4.2.1.

Rate limiting je pri použití fameworku Flask možné implementovať rôznymi spôsobmi. Všeobecne je možné implementovať rate limiting na úrovni samotného webového servera, proxy alebo priamo vo webovej aplikácii. Všetky vyššie uvedené prístupy majú výhody aj nevýhody, avšak, pre potreby testovania nástroja

[14]https://flask.palletsprojects.com/en/3.0.x/

[15]https://www.vmware.com/products/workstation-pro.html

Tab. 4.2: Koncové body a ich konfigurácia rate limitingu

Koncový bod	Max. počet požiadaviek za minútu
`/`	∞
`/login`	4
`/menu`	30
`/api`	100

bol zvolený prístup rate limitingu priamo vo webovej aplikácii. Rate limiting bol vo webovej aplikácii implementovaný pomocou knižnice *flask_limiter*[16]. Táto knižnica umožňuje pridávať obsluhám jednotlivých koncových bodov indikácie, resp. definície podmienok, za akých majú danú požiadavku obslúžiť. Webový framework Flask umožňuje definovať obslužné funkcie koncových bodov pomocou direktívy `@app.route("/názov")` a v spojení s direktívou implementovanou vyššie uvedenou knižnicou, *flask_limiter* a podmienkou `@limiter.limit("Podmienka")`, je možné definovať ako obsluhu koncového bodu, tak obmedzenie počtu požiadaviek, ktoré môže užívateľ odoslať webovému serveru za stanovený čas. Príklad využitia vyššie uvedených direktív je možné pozorovať vo výpise 4.16. Podobným spôsobom sú tak nakonfigurované všetky koncové body, ktoré sú v testovacej webovej aplikácii vyvinuté a popísané v nasledujúcich sekciách. Tabuľka 4.3.1 obsahuje zoznam koncových bodov a ich príslušnú konfiguráciu rate limitingu.

Výpis 4.16: Príklad konfigurácie rate limitingu pomocou knižnice *flask_limiter*

```
1 @app.route("/menu")
2 @limiter.limit("30 per minute")
3 def medium():
4     ...
```

HTTP hlavičky a konfigurácia CSP

HTTP hlavičky, podobne ako pri rate limitingu, je možné nakonfigurovať viacerými spôsobmi. Hlavičky je možné nakonfigurovať ako na samotnom webovom servery, proxy, tak na úrovni samotnej webovej aplikácii. Pre účely testovania nástroja bol opäť zvolený spôsob definície HTTP hlavičiek na úrovni webového servera. Test HTTP hlavičiek testuje primárne hlavičky definované v kapitole 2.2 a cieľom testu je otestovať schopnosti detekcie neprítomnosti každej z nich. Z toho dôvodu majú rôzne koncové body nastavené rôzne HTTP hlavičky. Z toho dôvodu je konfigurácia

[16]https://flask-limiter.readthedocs.io/en/stable/

HTTP hlavičiek v koncovom bode / ponechaná v základných nastaveniach bez manuálnej konfigurácie. Ukážku koncového bodu bez akejkoľvek konfigurácie je možné pozorovať vo výpise 4.17.

Výpis 4.17: Ukážke funkcie obsluhujúcej koncový bod /

```
@app.route("/")
def default():
    return "<h1>No headers, no cookies, no rate limit</h1>"
```

Koncovému bodu `/login` boli pri testovaní modulu hlavičky definované veľmi prísne. Vrámci HTTP odpovedí boli definované všetky HTTP hlavičky prítomné v zozname `MISSING_HEADERS` z výpisu 4.5. Avšak, bola ponechaná hlavička `Server`, ktorá v sebe obsahuje informácie ohľadom webového servera. HTTP hlavičky je možné nastaviť s použitím webového frameworku Flask pomocou vytvorenia objektu typu `flask.Response`, ktorý v sebe obsahuje dátovú štruktúru typu `Dictionary`. Táto dátová štruktúra v sebe obsahuje zoznam a príslušné hodnoty HTTP hlavičiek, ktoré budú odoslané spolu s odpoveďou na danú HTTP požiadavku. Tento slovník je tak možné naplniť jednotlivými HTTP hlavičkami podľa potreby. Implementáciu hlavičiek a zároveň rate limitingu pre koncový bod `/login` je možné pozorovať vo výpise 4.18.

Výpis 4.18: Konfigurácia HTTP hlavičiek pre koncový bod `/login`

```
@app.route("/login")
@limiter.limit("4 per minute")
def login():
    resp = flask.Response("<h1>Login page</h1>")
    resp.headers['Content-Security-Policy'] = "Content-Security-
   Policy: default-src 'none'; script-src 'self'; style-src 'self';
    img-src 'self'; connect-src 'self'; font-src 'self'; object-src
    'none'; media-src 'self'; frame-src 'none'; frame-ancestors '
   none'; form-action 'self'; base-uri 'self';"
    resp.headers['X-Frame-Options'] = "DENY"
    resp.headers['X-Content-Type-Options'] = "nosniff"
    resp.headers['Referrer-Policy'] = "strict-origin-when-cross-
   origin"
    resp.headers['Strict-Transport-Security'] = "max-age=31536000;
   includeSubDomains"
    resp.headers['Permissions-Policy'] = "accelerometer=(), camera
   =()"
    .
    .
```

Ďalším koncovým bodom je `/menu`. Tento koncový bod má oproti koncovému bodu `/login` značne voľnejšiu konfiguráciu CSP a neimplementuje niektoré bezpečnostné HTTP hlavičky ako *Strict-Transport-Security* alebo *Referrer-Policy*. Táto nebezpečná konfigurácia by v prípade výskytu zraniteľnosti XSS nezabránila tomuto útoku. Konfiguráciu tohto koncového bodu je možné pozorovať vo výpise 4.19.

Výpis 4.19: Konfigurácia HTTP hlavičiek pre koncový bod `/menu`

```
@app.route("/menu")
@limiter.limit("30 per minute")
def menu():
    resp = flask.Response("<h1>Main menu</h1>")
   resp.headers["Content-Security-Policy"] = (
        "object-src 'none'; style-src 'self' 'unsafe-inline';
   script-src 'self' 'unsafe-inline'; img-src * data:; connect-src
   https://www.test.cz https://cert.test.cz https://reg.test.cz;"
    )
    resp.headers["X-Frame-Options"] = "DENY"
    resp.headers["X-Content-Type-Options"] = "nosniff"
    return resp
```

Posledným koncovým bodom je `/api`. Tento koncový bod má okrem vyššej tolerancii počtu požiadaviek užívateľa nakonfigurovanú len jednu HTTP hlavičku a to *X-Frame-Options*. Konfiguráciu tohto koncového bodu je možné pozorovať vo výpise 4.20. Súhrn konfigurácii HTTP hlavičiek je možné pozorovať v tabuľke 4.3.1.

Výpis 4.20: Konfigurácia HTTP hlavičiek pre koncový bod `/api`

```
@app.route("/api")
@limiter.limit("100 per minute")
def api():
    resp = flask.Response("<h1>API</h1>")
    resp.headers['X-Frame-Options'] = "DENY"
    return resp
```

Cookies

Poslednou fázou vývoja bola konfigurácia cookies na jednotlivých koncových bodoch. Podobne ako pri HTTP hlavičkách a rate limitingu, rôzne koncové body majú definovanú rôznu konfiguráciu bezpečnostných atribútov cookies. Pre konfiguráciu cookies je pri použití webového frameworku Flask dostupná funkcia `set_cookie()`.

Tab. 4.3: Koncové body a ich konfigurácia HTTP hlavičiek

Koncový bod	/	/login	/menu	/api
Content-Security-Policy	✗	✓	✓	✗
X-Frame-Options	✗	✓	✓	✓
X-Content-Type-Options	✗	✓	✓	✗
Referrer-Policy	✗	✓	✗	✗
Strict-Transport-Security	✗	✗	✗	✗
Permissions-Policy	✗	✓	✗	✗
Server	✓	✓	✓	✓
X-Powered-By	✗	✗	✗	✓
X-AspNet-Version	✗	✗	✗	✗
X-AspNetMvc-Version	✗	✗	✗	✗

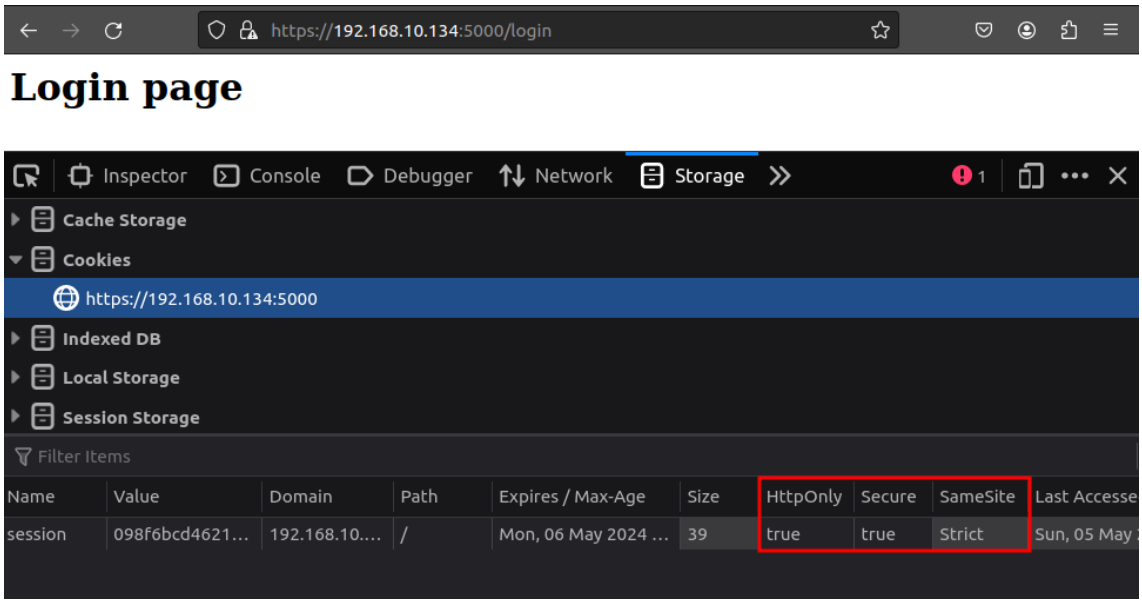
Medzi parametre funkcie patrí názov cookie, jeho hodnota, a bezpečnostné a iné atribúty. Koncový bod `/` je, podobne ako pri ostatných bezpečnostných mechanizmoch z predošlých sekcii, tou najjednoduchšou variantou a nepoužíva žiadne cookies. Je to z dôvodu, aby bolo možné overiť schopnosť nástroja upozorniť penetračného testera ako na prítomnosť, či neprítomnosť bezpečnostných atribútov, tak aj na neexistenciu cookies. Príklad konfigurácie cookies pomocou metódy `set_cookie()` je možné pozorovať vo výpise 4.21. Rovnako je možné pozorovať dané nastavenie aplikované vo webovom prehliadači *Firefox*[17] na obrázku 4.2. Obdobným spôsobom sú tak konfigurované všetky koncové body, ktorých bližší popis je možné nájsť v nasledujúcom texte.

Výpis 4.21: Príklad konfigurácie cookies pomocou metódy `set_cookie`

```
@app.route("/login")
@limiter.limit("4 per minute")
def login():
    resp = flask.Response("<h1>Login page</h1>")
    .
    .
    resp.set_cookie("sessionid", "098f6bcd4621d373cade4e832627b4f6"
   , secure=True, httponly=True, samesite='Strict')

    return resp
```

[17] https://www.mozilla.org/en-US/firefox/new/



Obr. 4.2: Ukážka konfigurácie cookies vo webovom prehliadači Firefox

Najprísnejšiu konfiguráciu má opäť koncový bod `/login`. Cookies na tomto koncovom bode spĺňajú všetky bezpečnostné požiadavky, ktoré nástroj testuje a daný cookie je chránený pred zraniteľnosťami XSS, CSRF a pred pred odosielaním cookie cez nezabezpečený kanál. Cieľom je opäť vytvoriť veľký kontrast medzi jednotlivými konfiguráciami a tak otestovať reaktivitu a detekčné schopnosti nástroja. Koncový bod `/menu` má atribúty cookies nastavené tak, aby bolo znemožnené ich ukradnutie pomocou zraniteľnosti XSS, avšak cookie neobsahuje atribút *Secure*, čo umožňuje daný cookie odosielať aj cez nezabezpečený kanál. Posledný koncový bod, `/api`, má naopak implementovaný atribút *Secure* a *SameSite*, čo daný cookie cháni pred odosielaním cez nezabezpečený kanál a zraniteľnosti CSRF, avšak nie je chránený pred útokmi typu XSS kvôli chýbajúcemu atribútu *HttpOnly*. Prehľad konfigurácie atribútov cookies na jednotlivých koncových bodoch je možné pozorovať v tabuľke 4.3.1.

Tab. 4.4: Koncové body a ich konfigurácia atribútov cookies

Koncový bod	**/**	**/login**	**/menu**	**/api**
HttpOnly	✗	✓	✓	✗
Secure	✗	✓	✗	✓
SameSite	✗	✓	✓	✗

4.3.2 Testovanie modulov

Táto sekcia sa bude venovať testovaniu modulov popísaných v kapitole 4.2 na zraniteľnej webovej aplikácii popísanej v kapitole 4.3.1 a to konkrétne popisu ich používania, konfigurácie, rozhrania príkazového riadku, a výstupu pre penetračného testera. Nástroj je možné spúšťať pomocou príkazového riadku, kde užívateľ musí špecifikovať názov modulu a jeho príslušné argumenty. Všetky moduly zdieľajú rovnaké vstupné parametre s výnimkou modulu testujúceho prítomnosť rate limitingu, kde sú navyše k zdieľaným parametrom pridané parametre ovplyvňujúce intenzitu modulu. Nástroj disponuje správou s nápoveďou pre jednotlivé moduly, z ktorej je možné vyčítať bližšie informácie k používaní daného modulu a jeho parametrov.

Test rate limit modulu

Modul rate limitingu vyžaduje pre svoj beh niekoľko vstupných parametrov. Jedná sa primárne o definíciu cieľa, proxy, protokolu a intenzity zasielania HTTP požiadaviek v podobe počtu vlákien a maximálneho celkového počtu zasielaných požiadaviek. Okrem cieľa majú všetky parametre prednastavené východzie hodnoty; proxy je vo východzej konfigurácii vypnuté, protokol je nastavený na HTTPS, počet vlákien je nastavený na 10 a celkový počet požiadaviek na 1000. Nakoľko efektivita modulu je značne závislá na testovanom prostredí, je na uvážení testera ako tieto hodnoty nastaví. Ako príklad je možné uviesť rozdiel medzi prihlasovacím formulárom, pri ktorom by malo byť nastavenie rate limitingu prísne, a medzi rozhraniami API, ktoré majú spravidla vyššiu toleranciu voči záťaži. Správu s nápoveďou pre tento modul je možné pozorovať vo výpise 4.22.

Výpis 4.22: Správa s nápoveďou modulu testujúceho rate limiting

```
usage: ptwebda.py ratelimit [-h] [-u URL] [-f FILE] [-p PROXY] [-s]
    [-t THREADS] [-n NUM_REQUESTS]

options:
  -h, --help            show this help message and exit
  -u URL, --url URL     URL to check headers for
  -f FILE, --file FILE, -f FILE
                        Path to the file used by the modules (
   optional)
  -p PROXY, --proxy PROXY, -p PROXY
                        Proxy URL to use (e.g., http
   ://127.0.0.1:8080)
  -s, --https           Use HTTPS. (only used with -f)
  -t THREADS, --threads THREADS
                        Number of threads to use. (default 10)
```

```
  -n NUM_REQUESTS, --num-requests NUM_REQUESTS
                        Number of requests to use. (default 1000)
```

Okrem detekcie prítomnosti rate limitu, je pre penetračné testovanie dôležitou informáciou aj približný počet požiadaviek, ktorý spustí vo webovej aplikácii alebo v API rate limiting. Správne nastavenie rate limitu môže byť náročné a v prípade nedostatočného obmedzenia požiadaviek, môže útočník spomaliť svoj útok tak, aby sa držal v limite bez obmedzenia. Pri testovaní modulu boli použité všetky koncové body zraniteľnej aplikácie popísanej v kapitole 4.3.1. Vo výpisoch 4.23, 4.24, 4.25 a 4.26 je možné pozorovať výsledky implementovaného testu rate limitingu na všetkých štyroch koncových bodoch testovacej webovej aplikácie.

Výpis 4.23: Test koncového bodu / bez obmedzenia

```
 ____            _                        _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:


        Test name:       : RateLimitTest
        Target:          : https://192.168.10.134:5000/
        HTTPS            : True
        Proxies          : None

        Threads          : 10
        Total requests   : 1000
        Display interval : 0.1

[*] Running module
[+] No rate limit detected! Total requests: 1000.00. Failed
   requests: 0.00 Progress: [====================>] 100.00%
[+] Module finished successfully
```

Výpis 4.24: Test koncového bodu `/login` s obmedzením pre 4 požiadavky za minútu

```
 ____            _                          _____          _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__    ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:


        Test name:       : RateLimitTest
        Target:          : https://192.168.10.134:5000/login
        HTTPS            : True
        Proxies          : None

        Threads          : 10
        Total requests   : 1000
        Display interval : 0.1

[*] Running module
Requests per second: 7.67. Total requests: 8.00. Failed requests:
   4.00 Progress: [>--------------------] 0.40%
[!] Too many failed requests (15 / 19)
[!] Basic heuristics suggests that the web server has rate limit
   configured
[i] Approximate threshold: 4 requests
[i] Elapsed time: 0.52
[i] Requests terminated by connection error = 15
[+] Module finished successfully
```

Výpis 4.25: Test koncového bodu `/menu` s obmedzením pre 30 požiadaviek za minútu

```
 ____            _                          _____          _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__    ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:
```

```
11
12
13          Test name:       : RateLimitTest
14          Target:          : https://192.168.10.134:5000/menu
15          HTTPS            : True
16          Proxies          : None
17
18          Threads          : 10
19          Total requests   : 1000
20          Display interval : 0.1
21
22  [*] Running module
23  Requests per second: 11.55. Total requests: 59.00. Failed requests:
        29.00 Progress: [=>--------------------] 3.00%
24  [!] Too many failed requests (41 / 71)
25  [!] Basic heuristics suggests that the web server has rate limit
       configured
26  [i] Approximate threshold: 30 requests
27  [i] Elapsed time: 2.60
28  [i] Requests terminated with 'HTTP 429 Too Many Requests' = 28
29  [i] Requests terminated by connection error = 13
30  [+] Module finished successfully
```

Výpis 4.26: Test koncového bodu `/api` s obmedzením pre 100 požiadaviek za minútu

```
1   ____             _                        _____           _
2  |  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
3  | |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
4  |  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
5  |_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
6                                    |_|               ptwebda v1.0
7                                          https://www.penterep.com
8
9
10 [*] Test info:
11
12
13          Test name:       : RateLimitTest
14          Target:          : https://192.168.10.134:5000/menu
15          HTTPS            : True
16          Proxies          : None
17
18          Threads          : 10
19          Total requests   : 1000
20          Display interval : 0.1
21
```

```
[*] Running module
Requests per second: 11.55. Total requests: 59.00. Failed requests:
    29.00 Progress: [=>--------------------] 3.00%
[!] Too many failed requests (41 / 71)
[!] Basic heuristics suggests that the web server has rate limit
   configured
[i] Approximate threshold: 30 requests
[i] Elapsed time: 2.60
[i] Requests terminated with 'HTTP 429 Too Many Requests' = 28
[i] Requests terminated by connection error = 13
[+] Module finished successfully
```

Vo vyššie uvedených výpisoch je možné pozorovať úspešnosť detekčných schopností modulu a odhalenie všetkých koncových bodov, kde bol alebo nebol nakonfigurovaný rate limiting. Rovnako modul úspešne odhadol približný počet požiadaviek, ktorý je potrebný na spustenie tohto protekčného mechanizmu. Pre overenie konzistentnosti nástroja boli dané testy opakované 20-krát a spriemerované výsledky je možné pozorovať v tabuľke 4.3.2.

Tab. 4.5: Výsledky testovania modulu rate limitingu

Koncový bod	**Limit**	**Detekcia**	**Odhad**
/	-	✗	-
/login	4	✓	4.5
/menu	30	✓	30
/api	100	✓	100

Test modulu HTTP hlavičiek

Podobne ako pri testovaní modulu rate limitingu, modul testovania bezpečnostných HTTP hlavičiek vyžaduje podobné vstupné parametre. Jedná sa primárne o definíciu cieľa, proxy a indikácie, či sa má modul pokúsiť naviazať spojenie cez zabezpečený kanál HTTPS. Nakoľko je počas samotného testu zasielaná len jedna HTTP požiadavka, nie je potrebné špecifikovať ani počet vlákien, ani celkový počet požiadaviek ako pri teste rate limitingu. Správu s nápoveďou pre tento modul je možné pozorovať vo výpise 4.27.

Výpis 4.27: Správa s nápoveďou pre modul HTTP hlavičiek

```
usage: ptwebda.py headers [-h] [-u URL] [-f FILE] [-p PROXY] [-s]

options:
  -h, --help            show this help message and exit
  -u URL, --url URL     URL to check headers for
  -f FILE, --file FILE, -f FILE
                        Path to the file used by the modules (
   optional)
  -p PROXY, --proxy PROXY, -p PROXY
                        Proxy URL to use (e.g., http
   ://127.0.0.1:8080)
  -s, --https           Use HTTPS. (only used with -f)
```

Testované boli opäť všetky koncové body ako v predošlom teste. Vo výpisoch 4.28, 4.29, 4.30 a 4.31 je možné pozorovať výstupy nástroja pri testovaní jednotlivých koncových bodov webovej aplikácie. Tabuľka 4.3.2 zobrazuje výsledky experimentálneho testovania a úspešnosť nástroja pri detegovaní bezpečnostných nedostatkov v podobe HTTP hlavičiek.

Výpis 4.28: Test HTTP hlavičiek pre koncový bod /

```
 ____            _                        _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : HeadersTest
        Target:   : https://192.168.10.134:5000/
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Missing headers:
        content-security-policy
        x-frame-options
        x-content-type-options
        referrer-policy
```

```
        strict-transport-security
        permissions-policy
[i] Headers potentially leaking info:
        server: Werkzeug/3.0.2 Python/3.10.12
[+] Module finished successfully
```

Výpis 4.29: Test HTTP hlavičiek pre koncový bod `/login`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|                ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : HeadersTest
        Target:   : https://192.168.10.134:5000/login
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Missing headers:
[i] Headers potentially leaking info:
        server: Werkzeug/3.0.2 Python/3.10.12
[+] Module finished successfully
```

Výpis 4.30: Test HTTP hlavičiek pre koncový bod `/menu`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|                ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : HeadersTest
        Target:   : https://192.168.10.134:5000/menu
```

```
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Missing headers:
        referrer-policy
        strict-transport-security
        permissions-policy
[i] Headers potentially leaking info:
        server: Werkzeug/3.0.2 Python/3.10.12
[+] Module finished successfully
```

Výpis 4.31: Test HTTP hlavičiek pre koncový bod `/api`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : HeadersTest
        Target:   : https://192.168.10.134:5000/api
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Missing headers:
        content-security-policy
        x-content-type-options
        referrer-policy
        strict-transport-security
        permissions-policy
[i] Headers potentially leaking info:
        server: Werkzeug/3.0.2 Python/3.10.12
[+] Module finished successfully
```

Tab. 4.6: Výsledky testovania modulu HTTP hlavičiek

Koncový bod	**Počet chýbajúcich hlavičiek**	**Počet hlavičiek odhaľujúcich informácie**	**Počet úspešných detekcií**
/	6	1	7
/login	1	1	2
/menu	3	1	4
/api	5	1	6

Test modulu analýzy CSP

Modul testujúci konfiguráciu CSP má rovnaké vstupné parametre ako modul testujúci HTTP hlavičky. Jedná sa teda o definíciu cieľa, proxy a indikácie, či sa má modul pokúsiť naviazať spojenie cez zabezpečený kanál HTTPS. Test opäť vyžaduje zaslanie len jednej HTTP požiadavky, preto nie je multivláknový prístup potrebný. Správu s nápoveďou pre tento modul je možné pozorovať na obrázku 4.32.

Výpis 4.32: Správa s nápoveďou pre modul analýzy CSP

```
usage: ptwebda.py csp [-h] [-u URL] [-f FILE] [-p PROXY] [-s]

options:
  -h, --help            show this help message and exit
  -u URL, --url URL     URL to check headers for
  -f FILE, --file FILE, -f FILE
                        Path to the file used by the modules (
   optional)
  -p PROXY, --proxy PROXY, -p PROXY
                        Proxy URL to use (e.g., http
   ://127.0.0.1:8080)
  -s, --https           Use HTTPS. (only used with -f)
```

Podobne ako v predošlých testoch boli testované všetky koncové body. Dva koncové body neobsahujú konfiguráciu CSP a nástroj by ich mal úspešne označiť a vyhodnotiť situáciu ako chýbajúcu konfiguráciu CSP. Koncový bod `/login` je podobne ako pri iných moduloch konfigurovaný veľmi prísne, avšak koncový bod `/menu` má umelo zavedené miskonfigurácie CSP a modul by mal byť schopný vyhodnotiť príliš perimisívnu konfiguráciu direktív `style-src`, `script-src` a `img-src`. Vo výpisoch 4.33, 4.34, 4.35 a 4.36 je možné pozorovať výstupy nástroja pri testovaní jednotlivých

koncových bodov webovej aplikácie. Tabuľka 4.3.2 zobrazuje výsledky experimentálneho testovania a úspešnosť nástroja pri detegovaní miskonfigurácií CSP.

Výpis 4.33: Test konfigurácie CSP pre koncový bod /

```
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __   |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \    | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |   | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/    |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CSPTest
        Target:   : https://192.168.10.134:5000/
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Server did not respond with any CSP headers!
[i] HTML content does not include any CSP configuration!
[!] Results cannot be printed! Value of results is None
[+] Module finished successfully
```

Výpis 4.34: Test konfigurácie CSP pre koncový bod /login

```
 ____            _                        _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __   |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \    | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |   | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/    |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CSPTest
        Target:   : https://192.168.10.134:5000/login
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] HTML content does not include any CSP configuration!
```

```
[i] CSP directies:
[i] default-src 'none'
[i] script-src 'self'
[i] style-src 'self'
[i] img-src 'self'
[i] connect-src 'self'
[i] font-src 'self'
[i] object-src 'none'
[i] media-src 'self'
[i] frame-src 'none'
[i] frame-ancestors 'none'
[i] form-action 'self'
[i] base-uri 'self';
[+] Module finished successfully
```

Výpis 4.35: Test konfigurácie CSP pre koncový bod `/menu`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|                ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CSPTest
        Target:   : https://192.168.10.134:5000/menu
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] HTML content does not include any CSP configuration!
[i] CSP directies:
[!] default-src not defined
[!] base-uri not defined
[i] object-src 'none'
[!] style-src 'self' 'unsafe-inline'
[!] script-src 'self' 'unsafe-inline'
[!] img-src * data:
[i] connect-src https://www.test.cz https://cert.test.cz https://
   reg.test.cz;
[+] Module finished successfully
```

Výpis 4.36: Test konfigurácie CSP pre koncový bod `/api`

```
 ____             _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CSPTest
        Target:   : https://192.168.10.134:5000/api
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] Server did not respond with any CSP headers!
[i] HTML content does not include any CSP configuration!
[!] Results cannot be printed! Value of results is None
[+] Module finished successfully
```

Tab. 4.7: Výsledky testovania modulu analýzy CSP

Koncový bod	**Počet zavedených miskonfigurácií**	**Počet detegovaných miskonfigurácií**
/	1	1
/login	0	0
/menu	5	5
/api	1	1

Test modulu analýzy atribútov cookies

Posledným bodom testovania nástroja bol modul testujúci bezpečnostné atribúty cookies. Modul zdieľa vstupné parametre s testom HTTP hlavičiek a s testom analýzy CSP a podobne ako tieto moduly nevyužíva multivlánkový prístup, ale k analýze atribútov cookies vyžaduje len jednu HTTP požiadavku. Medzi parametre teda patrí definícia cieľa, proxy a indikácie, či sa má modul pokúsiť nadviazať spojenie cez zabezpečený kanál HTTPS. Správu s nápoveďou pre tento modul je možné pozorovať na obrázku 4.37.

Výpis 4.37: Správa s nápoveďou pre modul analýzy cookies

```
usage: ptwebda.py cookies [-h] [-u URL] [-f FILE] [-p PROXY] [-s]

options:
  -h, --help            show this help message and exit
  -u URL, --url URL     URL to check headers for
  -f FILE, --file FILE, -f FILE
                        Path to the file used by the modules (
   optional)
  -p PROXY, --proxy PROXY, -p PROXY
                        Proxy URL to use (e.g., http
   ://127.0.0.1:8080)
  -s, --https           Use HTTPS. (only used with -f)
```

Testovanie má rovnaký rozsah ako pri testovaní predošlých modulov a teda v zábere testovania atribútov cookies sú zakomponované všetky koncové body. Vo výpisoch 4.38, 4.39, 4.40 a 4.41 je možné pozorovať výstupy nástroja pri testovaní jednotlivých koncových bodov webovej aplikácie. Tabuľka 4.3.2 zobrazuje výsledky experimentálneho testovania a úspešnosť nástroja pri detegovaní nedostatočného zabezpečenia cookies.

Výpis 4.38: Test cookies pre koncový bod /

```
 ____             _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __    |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \     | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ |  __/ |_) |    | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/     |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CookiesTest
        Target:   : https://192.168.10.134:5000/
        HTTPS     : True
        Proxies   : None

[*] Running module
[i] HTTP response did not set any cookies!
[+] Module finished successfully
```

Výpis 4.39: Test cookies pre koncový bod `/login`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CookiesTest
        Target:   : https://192.168.10.134:5000/login
        HTTPS     : True
        Proxies   : None

[*] Running module
[*] Checking cookies returned from the server:
[+] Found cookie: session
[i] Cookie 'session' has SameSite attribute set to: 'strict'
[+] Module finished successfully
```

Výpis 4.40: Test cookies pre koncový bod `/menu`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|               ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CookiesTest
        Target:   : https://192.168.10.134:5000/menu
        HTTPS     : True
        Proxies   : None

[*] Running module
[*] Checking cookies returned from the server:
[+] Found cookie: session
[!] Cookie 'session' does not have the secure flag set!
```

```
[!] Cookie 'session' has SameSite attribute set to: 'lax'
[+] Module finished successfully
```

Výpis 4.41: Test cookies pre koncový bod `/api`

```
 ____            _                          _____           _
|  _ \ ___ _ __ | |_ ___ _ __ ___ _ __     |_   _|__   ___ | |___
| |_) / _ \ '_ \| __/ _ \ '__/ _ \ '_ \      | |/ _ \ / _ \| / __|
|  __/  __/ | | | ||  __/ | |  __/ |_) |     | | (_) | (_) | \__ \
|_|   \___|_| |_|\__\___|_|  \___| .__/      |_|\___/ \___/|_|___/
                                 |_|                ptwebda v1.0
                                       https://www.penterep.com


[*] Test info:

        Test name : CookiesTest
        Target:   : https://192.168.10.134:5000/api
        HTTPS     : True
        Proxies   : None

[*] Running module
[*] Checking cookies returned from the server:
[+] Found cookie: session
[!] Cookie 'session' is not HttpOnly!
[!] Cookie 'session' has SameSite attribute set to: 'none'
[+] Module finished successfully
```

Tab. 4.8: Výsledky testovania modulu analýzy cookies

Koncový bod	**Počet zavedených miskonfigurácií**	**Počet detegovaných miskonfigurácií**
/	0	0
/login	0	0
/menu	2	2
/api	2	2

4.3.3 Výsledky testovania

Ako je opísané v časti 4.2, nástroj sa vo svojej prvej verzii skladá zo štyroch modulov. Pred testovaním v produkčnom prostredí bola vykonaná séria testov popísaných v kapitole 4.3.2, pre preukázanie účinnosti a spoľahlivosti každého modulu. Hlavným cieľom experimentálneho testovania bolo zabezpečiť, aby navrhovaný nástroj dokázal odhaliť a efektívne upozorniť na dôležité poznatky v konfiguráciách webových serverov a webových aplikácií, a pokúsiť sa identifikovať potenciálne úzke hrdlá a nedostatky nástroja, ktoré boli v priebehu testovania opravené.

Kontrolované prostredie poskytlo silnú pozíciu na interpretáciu výstupov testovacích prípadov jednotlivých modulov a ich porovnanie s umelo zavedenými chybnými konfiguráciami webovej aplikácie. Keďže všetky moduly, s výnimkou obmedzovania rýchlosti, pracujú na veľmi jednoduchých princípoch, výsledky experimentálneho testovania boli veľmi presné. Nástroj dokázal efektívne identifikovať všetky chýbajúce hlavičky a poskytol výstup týkajúci sa hlavičiek, ktoré odhaľujú potenciálne citlivé informácie. Pri testovaní modulu obmedzovania rýchlosti nástrojom produkované výsledky boli tiež presné až na niekoľko výnimiek. Modul obmedzovania rýchlosti využíva mutlivláknový prístup a dokáže odhadnúť nastavenia obmedzenia rýchlosti webovej aplikácie s presnosťou približne 5 požiadaviek.

Experimentálne testovanie modulu rate limitingu poskytlo poznatky o tom, ako zaobchádzať s niektorými okrajovými prípadmi, ktoré môžu zmeniť výstup nástroja a poskytnúť nepresné informácie. Medzi okrajové prípady patria napríklad nasledovné: Nástroj je príliš rýchly a limit rýchlosti je pomerne malý, alebo limit rýchlosti je veľmi povolený a nástroj nie je dostatočne rýchly na spustenie limitu rýchlosti, alebo nástroj neposiela dostatok požiadaviek na spustenie limitu rýchlosti definovaného testovanou aplikáciou.

Nástroj bol okrem testovania v kontrolovanom, lokálnom prostredí testovaný aj na reálnych projektoch komerčných penetračných testov. Nástroj bol testovaný celkovo na viac ako desiatich projektoch pre klientov z rôznych sfér. Jednalo sa o webové aplikácie verejne dostupné z internetu a o interné webové aplikácie či SAP systémy. Nástroj sa pri používaní reálnymi penetračnými testermi javil ako veľmi prínosný a značne pomáhal s rutinnými kontrolami k identifikácii zraniteľností.

4.4 Integrácia nástroja do platformy Penterep

Nástroj popísaný v predošlých kapitolách bol navrhovaný a vyvíjaný na dve primárne použitia. Nástroj je konštruovaný tak, aby sa s ním dalo pracovať ako s konzolovou aplikáciu, ktorá poskytuje výstupy späť do konzoly, tak s nástrojom, ktorý je volaný iným externým nástrojom a vyžaduje striktné dodržiavanie formátu

výstupu. Platforma Penterep[18] využíva desiatky rôznych nástrojov, ktoré poskytujú dáta pre penetračných testerov do webového rozhrania a vyžaduje, aby nástroje poskytovali svoje výstupy vo formáte JSON s dodržaním preddefinovanej štruktúry. Základnú štruktúru definovanú internými smernicami platformy Penterep je možné pozorovať vo výpise 4.42.

Výpis 4.42: Základná JSON štruktúra požadovaného formátu výstupu nástroja

```
{
    "guid": "",
    "status": "",
    "message": "",
    "result": {
        "nodes": [],
        "properties": {},
        "vulnerabilities": []
    }
}
```

4.4.1 Použitie knižnice ptlibs

Aby mohlo dôjsť k serializácii výsledkov jednotlivých testov, je potrebné použiť knižnicu vyvinutú spoločnosťou Penterep *ptlibs*[19], ktorá má v sebe zakomponované metódy pre prácu s JSON štruktúrou ukázanou vo výpise 4.42. Knižnica obsahuje metódy na komplexné operácie akými sú napríklad; vkladanie nových uzlov do platformy, vkladanie rôznych tzv. *properties*, či vkladanie samotných nálezov. Nástroj PtWebDA primárne využíva funkcionalitu vkladania jednotlivých nálezov, ktoré boli počas testovania odhalené. Jednotlivé zraniteľnosti sú v platforme označované jedinečným identifikátorom vo formáte `PTV-XXX-XXXXXX` a na vkladanie jednotlivých nálezov do vyššie popísanej JSON štruktúry je využívaná funkcia `add_vulnerability`. Príklad použitia funkcie `add_vulnerability` a výslenú JSON štruktúru s ukážkovým nálezom je možné pozorovať vo výpisoch 4.43 a 4.44[20]. Nástroj PtWebDA využíva túto funkcionalitu k serializácii výsledkov testov, čím zabezpečuje kompatibilitu s platformou Penterep.

[18] https://www.penterep.com/
[19] https://pypi.org/project/ptlibs/
[20] Zdrojom tohto výpisu je interná dokumentácia platformy Penterep

Výpis 4.43: Príklad volania metódy `add_vulnerability`

```
self.ptjsonlib.add_vulnerability("PTV-WEB-CROSSDOMAINXML",
    request_text, response_text)
```

Výpis 4.44: Praktický príklad požadovaného formátu výstupu nástroja

```
{
    "satid": "",
    "guid": "",
    "status": "finished",
    "message": "",
    "results": {
        "nodes": [],
        "properties": {},
        "vulnerabilities": [
            {
                "code": "PTV-WEB-CROSSDOMAINXML",
                "request": "GET /crossdomain.xml HTTP/1.1\r\nHost:
    www.example.com\r\nUser-Agent: Penterep Tools\r\nAccept-Encoding
    : gzip, deflate, br\r\nAccept: */*\r\nConnection: keep-alive\r\n
    \r",
                "response": "HTTP/1.1 200 OK\r\nDate: Fri, 15 Mar
    2024 07:00:06 GMT\r\nContent-Type: application/xml\r\nContent-
    Length: 267\r\nConnection: keep-alive\r\nX-Me: nvp1-fapi-vmjsk\r
    \nLast-Modified: Wed, 06 Mar 2024 18:02:30 GMT\r\nCache-Control:
     max-age=7776000, public\r\nExpires: Thu, 13 Jun 2024 07:00:06
    GMT\r\nContent-Encoding:....."
            }
        ]
    }
}
```

4.4.2 Serializácia výsledkov testu

Nakoľko výledky jednotlivých testov implementovaných nástrojom PtWebDA nie sú konzistentné v ich podobe, bolo potrebné implementovať funkcioinalitu, ktorá prevedie všetky výsledky na jednotný formát podporovaný platformou Penterep. V kapitole 4.1.1 je možné nájsť stručné popisy metód bázového modulu, ktoré musia jednotlivé moduly implementovať zvlášť. Jednou z týchto metód je práve metóda `json`. Nakoľko je výstup každého modulu iný, metóda `json` sa tak medzi jednotlivými

modulmi do značnej mieri líši. V nasledujúcom texte budú opísané implementácie serializácie výsledkov jednotlivých modulov.

Serializácia výsledkov modulu rate limitingu

Modul rate limitingu je spomedzi všetkých implementovaných modulov najviac špecifický pri serializácii jeho výsledkov. Metóda `add_vulnerability` dostupná z knižnice *ptlibs* umožňuje okrem zadania kódu zraniteľnosti poskytnúť ako zadanie HTTP požiadavky, ktorá je pri teste zasielaná, tak HTTP odpovede webového servera. Nakoľko pri testovaní rate limitingu sa zasielajú tisíce požiadaviek, je táto funkcionalita metódy `add_vulnerability` využitá k predaniu informácií ohľadom rate limitingu penetračnému testerovi, k jeho manuálnej inšpekcii vo webovom rozhraní platformy Penterep. Na rozdiel od ostatných modulov, kde je k vykonaniu testu potrebná len jedna HTTP požiadavka, je tak táto funkcionalita využitá efektívnejšie. Útržok funkcie `json` je možné pozorovať vo výpise 4.45.

Výpis 4.45: Serializácia výsledkov modulu rate limitingu

```
def json(self) -> str | None:
    response_text_no_detection = "NO RATE LIMIT DETECTED!"
    response_text_detection = (
        "RATE LIMIT DETECTED!"
        + f"Too many failed requests ({self.failed_req} / {self.success_req + self.failed_req})"
        + f"Approximate threshold: {self.success_req} requests"
        + f"Elapsed time: {self.elapsed_time:.2f}"
    )

    if self.result.rate_limited:
        self.ptjsonlib.add_vulnerability(
            "PTV-WEB-RATELIMITDETECTED", self.request_text.decode(), response_text_detection
        )
    else:
        self.ptjsonlib.add_vulnerability(
            "PTV-WEB-NORATELIMIT", self.request_text.decode(), response_text_no_detection
        )
    return self.ptjsonlib.get_result_json()
```

Serializácia výsledkov modulu HTTP hlavičiek

Modul HTTP hlavičiek je v rámci množstva možných nálezov najproduktívnejší. Celkovo je modul schopný odhaliť až desať rôznych nálezov, preto bol potrebné navrhnúť serializáciu čo najefektívnejšie. V rámci modulu bola zavedená dátová štruktúra s názvom `PT_VULN_CODES`, ktorá v sebe obsahuje názov hlavičky, ktorej sa miskonfigurácia týka a jej príslušný kód v platforme Penterep. Túto dátovú štruktúru je možné pozorovať vo výpise 4.46.

Výpis 4.46: Dátová štruktúra `PT_VULN_CODES` pre modul HTTP hlavičiek

```
PT_VULN_CODES: dict[str, str] = {
    # Missing headers:
    "content-security-policy": "PTV-WEB-MISSINGHEADER-CSP",
    "x-frame-options": "PTV-WEB-MISSINGHEADER-XFRAMEOPTIONS",
    "x-content-type-options": "PTV-WEB-MISSINGHEADER-
   XCONTENTTYPEOPTIONS",
    "referrer-policy": "PTV-WEB-MISSINGHEADER-REFERRERPOLICY",
    "strict-transport-security": "PTV-WEB-MISSINGHEADER-HSTS",
    "permissions-policy": "PTV-WEB-MISSINGHEADER-PERMISSIONSPOLICY"
   ,
    # Headers leaking information:
    "server": "PTV-WEB-LEAKINGHEADER-SERVER",
    "x-powered-by": "PTV-WEB-LEAKINGHEADER-XPOWEREDBY",
    "x-aspnet-version": "PTV-WEB-LEAKINGHEADER-XASPNETVERSION",
    "x-aspnetmvc-version": "PTV-WEB-LEAKINGHEADER-XASPNETMVCVERSION
   ",
}
```

Pri serializácii výsledkov testu je vyššie uvedená dátová štruktúra použitá k extrakcii príslušného kódu zraniteľnosti. Vo výpise 4.8 z kapitoly 4.2.2 je možné pozorovať prístup k dátovej štruktúre `PT_VULN_CODES` a vloženie kódu zraniteľnosti do dátovej štruktúry `Header`. Obsah dátovej štruktúry `Header` je možné pozorovať vo výpise 4.47. Ku kódu zraniteľnosti je tak možné pristupovať v rámci dátovej štruktúry `Header` a ukážku samotného procesu serializácie je možné pozorovať vo výpise 4.48.

Výpis 4.47: Dátová štruktúra `Header`

```
class Header(NamedTuple):
    name: str
    code: str | None
    value: str | None
```

Výpis 4.48: Serializácia výsledkov modulu HTTP hlavičiek

```
def json(self) -> str | None:
    for header in self.results.missing_headers:
        if header.code is None:
            Log.error(f"Header in findings {header.name} does not
   have a PT_VULN_CODE!")
            continue

        self.ptjsonlib.add_vulnerability(
            header.code, self.request_text.decode(), self.
   response_text.decode()
        )
    # Podobným štýlom ako vyššie sa cyklí aj cez ostatné prvky dá
   tovej štruktúry "self.results"
    .
    .
    return self.ptjsonlib.get_result_json()
```

Serializácia výsledkov modulu CSP

Modul CSP je pri serializácii značne jednoduchší. Nakoľko samotné vyhodnocovanie direktív CSP vyžaduje manuálny prístup a kontext, ktorý nástroj nie je schopný sám o sebe získať, sú všetky potenciálne nálezy nástrojom vyhodnotené, označené ako jeden nález. Tým je telo metódy `json` značne zjednodušené na len jedno volanie metódy `add_vulnerability`. Urývok metódy `JSON` je možné pozorovať vo výpise 4.49.

Výpis 4.49: Serializácia výsledkov modulu CSP

```
def json(self) -> str | None:
    self.ptjsonlib.add_vulnerability(
        "PTV-WEB-CSPMISCONFIG", self.request_text.decode(), self.
   response_text.decode()
    )

    return self.ptjsonlib.get_result_json()
```

Serializácia výsledkov modulu cookies

Podobne ako pri module HTTP hlavičiek, dochádza v tomto module k serializácii podobným spôsobom. Modul analýzy cookies môže vyhodnotiť až 4 rôzne nálezy a opäť

bola definovaná dátová štruktúra `PT_VULN_CODES`, ktorú je možné pozorovať vo výpise 4.50. Rovnakým štýlom tak dochádza aj k volaniu metódy `add_vulnerability` a skrátenú verziu kódu funkcie `json` je možné pozorovať vo výpise 4.51.

Výpis 4.50: Dátová štruktúra `PT_VULN_CODES` pre modul analýzy cookies

```
PT_VULN_CODES: dict[str, str] = {
    "httponly": "PTV-WEB-COOKIENOTHTTPONLY",
    "secure": "PTV-WEB-COOKIENOTSECURE",
    "samesite": "PTV-WEB-COOKIEWITHOUTSAMESITE",
    "samesite-none": "PTV-WEB-COOKIESAMESITENONE",
}
```

Výpis 4.51: Serializácia výsledkov modulu cookies

```
def json(self) -> str | None:
    for cookie in self.results.cookies:
        if not cookie.secure:
            self.ptjsonlib.add_vulnerability(
                PT_VULN_CODES["secure"],
                self.request_text.decode(),
                self.response_text.decode()
            )
        if not cookie.http_only:
            self.ptjsonlib.add_vulnerability(
                PT_VULN_CODES["httponly"],
                self.request_text.decode(),
                self.response_text.decode(),
            )
        if not cookie.same_site.present:
            self.ptjsonlib.add_vulnerability(
                PT_VULN_CODES["samesite"],
                self.request_text.decode(),
                self.response_text.decode(),
            )
        else:
            if cookie.same_site.value is None:
                continue
            if cookie.same_site.value.lower() == "none":
                self.ptjsonlib.add_vulnerability(
                    PT_VULN_CODES["samesite-none"],
                    self.request_text.decode(),
                    self.response_text.decode(),
                )
    return self.ptjsonlib.get_result_json()
```

4.4.3 Sprievodné texty k implementovaným testom a zraniteľnostiam

Moduly nástroja PtWebDA popísané v predošlých sekciách reprezentujú v platforme Penterep konkrétne testy, ktoré sa majú vykonať na základe prechádzania kontrolných zoznamov, ktoré sú v platforme zavedené. V rámci kontrolných zoznamov má každý test vlastný textový popis. V prílohe A tejto diplomovej práce sa nachádzajú popisy jednotlivých textov pre konkrétne testy, ktoré sú vykonávané nástrojom navrhovaným v tejto diplomovej práci. Celkovo boli vytvorené 4 texty k testom; test rate limitingu, test HTTP hlavičiek, test konfigurácie CSP, a test cookies. Textový popis jednotlivých testov je zložený z nasledujúcich sekcií:

- Úloha (test) – krátky a výstižný popis testu,
- Popis testu – podrobnejší popis testu a bližšia špecifikácia jeho cieľov,
- Náročnosť testu – subjektívne hodnotenie náročnosti vykonávaného testu,
- Ako vykonať test – detailný popis postupu testovania obsahujúci inštrukcie a potrebné príkazy,
- Testom je možné odhaliť tieto zraniteľnosti – zoznam zraniteľností, ktoré je možné testom odhaliť,
- Testom je možné sprístupniť nadväzujúce testy – zoznam testov, pre ktoré je pozitívny nález vykonávaného testu prerekvizitou.

Testy popísané vyššie odhaľujú rôzne zraniteľnosti, ktoré majú v platforme taktiež vlastné popisy. V rámci tejto diplomovej práce bolo k týmto zraniteľnostiam napísaných celkovo 16 textov, z toho 1 patrí testu rate limitingu, 10 patrí testu HTTP hlavičiek, 1 patrí testu konfigurácie CSP, a 4 patria testu cookies. Popisy jednotlivých textov pre konkrétne zraniteľnosti je možné nájsť v prílohe B tejto diplomovej práce. Textový popis jednotlivých zraniteľností je zložený z nasledujúcich sekcií:

- Názov zraniteľnosti
- Popis zraniteľnosti – výstižný popis zraniteľnosti
- Príčiny – dôvod výskytu zraniteľnosti
- Prejavy – správanie, ktoré napovedá o výskyte zraniteľnosti
- Dopady – bezpečnostný dopad, ktorý daná zraniteľnosť má po jej zneužití aktérom hrozby
- Náprava / doporučenie – kroky vedúce k náprave zraniteľnosti
- Závažnosť – objektívne hodnotenie zraniteľnosti podľa hodnoty *Common Vulnerability Scoring System* (CVSS)[21]

[21] CVSS: `https://nvd.nist.gov/vuln-metrics/cvss`

- CVSS score – číselná hodnota získaná z kalkulačky CVSS[22]
- CVSS string – textový reťazec (tzv. vektor) získaný z kalkulačky CVSS

4.5 Porovnanie s dostupnými riešeniami

Dynamickou analýzou sa zaoberajú mnohé ako verejne dostupné *open source*, tak i platené nástroje. Aj keď pojmy dynamická analýza, DAST a sken zraniteľností sú v definícii rozdielne, v praxi sa často tieto pojmy zamieňajú a používajú nesprávne. V nasledujúcich tabuľkách budú porovnávané nástroje, ktoré v rámci svojho behu umožňujú automatizované alebo manuálne DAST testovanie. Prvotné porovnanie bude porovnávať jednotlivé nástroje na základe nasledovných kritérií:

- Nástroj umožňuje vykonávať fuzzing,
- Nástroj podporuje manuálne testovanie,
- Nástroj podporuje automatizované testovanie,
- Nástroj je dostupný v *open source* podobe.

Následne budú dané nástroje porovnané s plánovanými funkciami navrhovaného nástroja PtWebDA. Medzi porovnávané funkcionality patria nasledovné:

- Analýza rate limitingu,
- Analýza HTTP hlavičiek,
- Analýza atribútov cookies,
- Analýza CSP direktív.

Tabuľky 4.9 a 4.10 zobrazujú výsledky vyššie uvedených porovnaní. Z výsledokv v tabuľke 4.10 vyplýva, že testovanie rate limitingu neobsahuje žiadny z porovnávaných nástrojov. Tento výsledok bol očakávaný vzhľadom k tomu, že väčšina automatizovaných nástrojov dbá na čo najväčšie pokrytie preddefinovaných testov a nie na tzv. *opreation security*[23]. Nástroj je tak možné implementáciou testovania rate limitingu považovať za unikátny. Nástroj PtWebDA spolu so svojím modulárnym dizajnom, ľahkou možnosťou pridávania rôznych rozšírení v podobe ďalších modulov a jeho unikátnosťou, predstavuje kompaktný a praktický nástroj pre penetračných testerov. V neposlednom rade je výhodou nástroja PtWebDA tiež fakt, že jeho implementácia je *open source*.

[22]Kalkulačka CVSS 3.1: `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator`

[23]Pod pojmom *operation security* alebo *opsec* sa rozumie dbanie na bezpečnosť daného testovania čo znamená obmedzovanie rýchlosti nástrojov alebo invazívnych testov za účelom vyhnutiu sa detekcie danej aktivity.

Tab. 4.9: Nástroje pre dynamickú analýzu

Nástroje	Fuzzing	Manuálne testovanie	Automatizované testovanie	Open source
Burp Suite CE[1]	✓	✓	✓	✗
OWASP ZAP	✓	✓	✓	✓
ffuf	✓	✓	✗	✓
Wireshark	✗	✓	✗	✗
Netsparker	✗	✗	✓	✗
Acunetix	✓	✓	✓	✗
W3af	✓	✗	✓	✓
Nikto	✗	✗	✓	✓
Rapid7 AppSpider	✗	✗	✓	✗
WebInspect	✗	✗	✓	✗

[1] Burp Suite Community Edition

Tab. 4.10: Porovnanie nástrojov s funkciami implementovaného nástroja PtWebDA

Test	Rate limit	HTTP hlavičky	Atribúty cookies	Analýza CSP
Burp Suite CE	✗	✓	✓	✗
OWASP ZAP	✗	✓	✓	✓
ffuf	✗	✓	✓	✗
Wireshark	✗	✗	✗	✗
Netsparker	✗	✓	✓	✓
Acunetix	✗	✓	✓	✓
W3af	✗	✓	✓	✗
Nikto	✗	✓	✓	✗
Rapid7 AppSpider	✗	✓	✓	✓
WebInspect	✗	✓	✓	✗
PtWebDA	✓	✓	✓	✓

Záver

Táto diplomová práca sa venovala problematike dynamickej analýzy za účelom efektívneho testovania bezpečnosti webových aplikácií. V teoretickej časti boli predstavené hrozby, ktorým webové aplikácie a služby čelia, metódy testovania bezpečnosti webových aplikácií, a princípy a metodika použitia dynamickej analýzy k efektívnemu testovaniu bezpečnosti webových aplikácií či API. Dôraz bol kladený primárne na bezpečnostné HTTP hlavičky, schopnosti webovej aplikácie chrániť sa pred útokmi vyžadujúcich vysoko nadštandardne veľké množstvo HTTP požiadaviek, analýzu direktív CSP chrániacich webovú aplikáciu pred útokmi XSS, a na bezpečnostné atribúty cookies chrániace samotných užívateľov webovej aplikácie.

V rámci praktickej časti bol navrhnutý a implementovaný automatizovaný modulárny nástroj PtWebDA slúžiaci k odhaleniu vybraných zraniteľností, či chybných konfigurácií webových aplikácií a serverov za pomoci dynamickej analýzy. Ďalej bolo tiež vytvorené experimentálne pracovisko v podobe zraniteľnej webovej aplikácie, kde prebiehalo experimentálne testovanie funkcionality a efektivity navrhovaného nástroja. Okrem experimentálneho testovania v kontrolovanom prostredí bol nástroj testovaný aj v reálnom produkčnom prostredí pri penetračných testoch pre reálnych klientov. Praktická časť taktiež obsahovala integráciu nástroja do platformy Penterep, ktorej súčasťou bolo aj zostavenie sprievodných textov pre jednotlivé testy a zraniteľnosti. Na záver praktickej časti bolo vykonané porovnanie aktuálne dostupných nástrojov a ich funkcionality s implementovaným nástrojom PtWebDA. Nástroj oproti porovnávaným nástrojom disponuje jedinečnou schopnosťou efektívne testovať rate limiting a zároveň testovať bezpečnosť HTTP hlavičiek, atribútov súborov cookies a direktív content security policy v jednom kompaktnom riešení aplikovateľnom v praxi pri práci penetračných testerov.

Výsledky tejto diplomovej práce boli prezentované na študentskej konferencii EEICT, kde článok v kategórií „M4 – Communication and Information Systems, Network Security II“ získal 1. miesto a špeciálnu cenu sponzora od spoločnosti NXP Semiconductors Czech Republic s.r.o.

Literatúra

[1] KUNDA, Mohammed Ali and ALSMADI, Izzat Practical web security testing: Evolution of web application modules and open source testing tools. In *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. 2022. s. 152–155. doi:10.1109/IDSTA55301.2022.9923130. [online]. [cit. 2023-05-29].

[2] PORTSWIGGER *What is DevSecOps? A guide from PortSwigger*. [online]. [cit. 2023-12-6]. Dostupné z: <`https://portswigger.net/solutions/devsecops/guide-to-devsecops`>

[3] BALOCH, Rafay Ethical hacking and penetration testing guide. CRC Press: Taylor & Francis Group. 2014. 531 s. ISBN 978-1-4822-3161-8.

[4] KUPSCH, James A. and MILLER, Barton P. Manual vs. Automated vulnerability assessment: A case study. ročník 469. 01 2009. Dostupné z: <`https://www.researchgate.net/publication/228910960_Manual_vs_Automated_vulnerability_assessment_A_case_study`>

[5] HACKERONE *What Is Vulnerability Assessment? Benefits, Tools, and Process*. [online]. [cit. 2023-05-29]. Dostupné z: <`https://www.hackerone.com/knowledge-center/what-vulnerability-assessment-benefits-tools-and-process`>

[6] OWASP *Static Code Analysis*. [online]. [cit. 2023-05-29]. Dostupné z: <`https://owasp.org/www-community/controls/Static_Code_Analysis`>

[7] OWASP *Cross-Site Scripting (XSS)*. [online]. [cit. 2023-05-29]. Dostupné z: <`https://owasp.org/www-community/attacks/xss/`>

[8] OWASP *Cross-Site Request Forgery (CSRF)*. [online]. [cit. 2023-05-29]. Dostupné z: <`https://owasp.org/www-community/attacks/csrf`>

[9] OWASP *Clickjacking*. [online]. [cit. 2023-05-29]. Dostupné z: <`https://owasp.org/www-community/attacks/Clickjacking`>

[10] KEYCDN *What Is MIME Sniffing?* [online]. [cit. 2023-05-29]. Dostupné z: <`https://www.keycdn.com/support/what-is-mime-sniffing`>

[11] LAZAROV, W.; MARTINÁSEK, Z. Web Platform for Comprehensive Penetration Testing. In *Proceedings II of the 28th Conference STUDENT EEICT 2022*. Brno University of Technology, Faculty of Electrical Engineering and

Communication. april 2022. s. 88–91. doi:10.13164/eeict.2022.88. [online]. [cit. 2023-12-15].

[12] CLOUDFLARE *What is Rate Limiting?* [online]. [cit. 2023-05-29]. Dostupné z: <https://www.cloudflare.com/learning/bots/what-is-rate-limiting/>

[13] PATIL, Rachana Yogesh and RAGHA, Lata A rate limiting mechanism for defending against flooding based distributed denial of service attack. In *2011 World Congress on Information and Communication Technologies*. 2011. s. 182–186. doi:10.1109/WICT.2011.6141240. [online]. [cit. 2023-06-29].

[14] FIRMANI, Donatella and LEOTTA, Francesco and MECELLA, Massimo On Computing Throttling Rate Limits in Web APIs through Statistical Inference. In *2019 IEEE International Conference on Web Services (ICWS)*. 2019. s. 418–425. doi:10.1109/ICWS.2019.00075. [online]. [cit. 2023-06-29].

[15] OWASP *HTTP Security Response Headers Cheat Sheet*. [online]. [cit. 2023-05-29]. Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html>

[16] OWASP *Content Security Policy (CSP)*. [online]. [cit. 2023-05-29]. Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html#content-security-policy-csp>

[17] OWASP *Clickjacking Defense Cheat Sheet*. [online]. [cit. 2023-05-29]. Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html>

[18] MOZILLA DEVELOPER NETWORK *MIME Types*. [online]. [cit. 2023-05-29]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#mime_sniffing>

[19] MOZILLA DEVELOPER NETWORK *Referer Header: Privacy and Security Concerns*. [online]. [cit. 2023-05-29]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/Security/Referer_header:_privacy_and_security_concerns>

[20] OWASP *HTTP Strict Transport Security Cheat Sheet*. [online]. [cit. 2023-05-29]. Dostupné z: <https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html>

[21] MOZILLA DEVELOPER NETWORK *Permissions Policy (formerly Feature Policy)*. [online]. [cit. 2023-05-29]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy>

[22] CLOUDFLARE *What are cookies? | Cookies definition*. [online]. [cit. 2024-04-20]. Dostupné z: <https://www.cloudflare.com/learning/privacy/what-are-cookies/>

[23] MOZILLA DEVELOPER NETWORK *Using HTTP cookies*. [online]. [cit. 2024-04-20]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

[24] COOKIEBOT *What are tracking cookies and how do they work?* [online]. [cit. 2024-04-20]. Dostupné z: <https://www.cookiebot.com/en/tracking-cookies/>

[25] INVICTI *Cookie security flags*. [online]. [cit. 2024-04-20]. Dostupné z: <https://www.invicti.com/learn/cookie-security-flags/>

[26] NIDECKI, Tomasz Andrzej *Vulnerability Assessment and Penetration Testing of Web Application*. [online]. [cit. 2023-05-29]. Dostupné z: <https://www.acunetix.com/blog/web-security-zone/dynamic-static-code-analysis-web-security/>

[27] MYERS, Glenford J. and BADGETT, Tom and SANDLER, Corey The Art of Software Testing. Hoboken, New Jersey: Wiley. třetí vydání. 2012. xi, 240 s. ISBN 978-1-118-03196-4.

[28] AFIANIAN, Amir and NIKSEFAT, Salman and SADEGHIYAN, Babak Malware Dynamic Analysis Evasion Techniques: A Survey. 11 2018. [online]. [cit. 2023-09-20]. Dostupné z: <https://www.researchgate.net/publication/328758559_Malware_Dynamic_Analysis_Evasion_Techniques_A_Survey>

[29] YAN, Xuexiong and MA, Hengtai and WANG, Qingxian *A static backward taint data analysis method for detecting web application vulnerabilities*. [online]. [cit. 2023-05-29]. Dostupné z: <https://ieeexplore.ieee.org/document/8230288>

[30] CircleCI Blog *SAST vs DAST: When to Use Them*. [online]. [cit. 2023-05-29]. Dostupné z: <https://circleci.com/blog/sast-vs-dast-when-to-use-them/#c-consent-modal>

[31] PELLEGRINO, Giancarlo and TSCHÜRTZ, Constantin and BODDEN, Eric and ROSSOW, Christian *Using Dynamic Analysis to Crawl and Test Modern Web Applications*. [online]. [cit. 2023-05-29]. Dostupné z: <https://link.springer.com/chapter/10.1007/978-3-319-26362-5_14>

[32] FOX, Jacob *Manual versus Automated Penetration Testing*. [online]. [cit. 2023-05-29]. Dostupné z: <https://www.cobalt.io/blog/manual-vs-automated-pentesting>

[33] PORTSWIGGER *Dynamic Application Security Testing (DAST)*. [online]. [cit. 2023-05-29]. Dostupné z: <https://portswigger.net/burp/application-security-testing/dast>

[34] PETRANOVIĆ, Teodora and ŽARIĆ, Nikola *Effectiveness of Using OWASP TOP 10 as AppSec Standard*. [online]. [cit. 2023-06-29]. Dostupné z: <https://ieeexplore.ieee.org/document/10078626>

[35] OWASP *OWASP Top Ten*. [online]. [cit. 2023-05-29]. Dostupné z: <https://owasp.org/www-project-top-ten/>

[36] OWASP *OWASP Application Security Verification Standard 4.0*. [online]. [cit. 2023-05-29]. Dostupné z: <https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf>

[37] CLOUDFARE *Rate limiting best practices*. [online]. [cit. 2024-04-29]. Dostupné z: <https://developers.cloudflare.com/waf/rate-limiting-rules/best-practices/>

Zoznam symbolov a skratiek

API Application Programming Interface

ASVS Application Security Verification Standard

CD/CI Continuous Integration and Continuous Delivery/Continuous Deployment

CSP Content Security Policy

CSS Cascading Style Sheets

CSRF Cross-Site Request Forgery

CVSS Common Vulnerability Scoring System

DAST Dynamic Application Security Testing

DDoS Distributed Denial of Service

DOM Document Object Model

DoS Denial of Service

HID Human Interface Device

HSTS HTTP Strict Transport Security

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

MIME Multipurpose Internet Mail Extension

OWASP Open Web Application Security Project

SAST Static application security testing

SDL Security Development Lifecycle

SSL Secure Sockets Layer

TLS Transport Level Security

URI Uniform Resource Identifier

URL Uniform Resource Locator

XSS Cross-Site Scripting

Zoznam príloh

A Sprievodné texty k testom

Test prítomnosti rate limitingu

- **Úloha (test)** – Implementuje webová aplikácia obmedzovanie počtu požiadaviek užívateľa?
- **Popis testu** – Úlohou tohto testu je overiť, či webová aplikácia alebo API implementuje rate limiting a obmedzuje tak útočníka vo vykonávaní fuzzingu alebo brute force útokov.
- **Náročnosť testu** – Jednoduchá
- **Ako vykonať test** – Test je možné vykonať pomocou automatizovaných nástrojov, ktoré používajú techniku fuzzingu či brute force a pozorovať, či a ako webová aplikácia na zvýšenú záťaž reaguje. Pokiaľ po určitom počte odoslaných požiadaviek webová aplikácia neobmedzuje počet zasielaných požiadaviek od užívateľa, jedná sa o chýbajúcu prítomnosť rate limitingu. Pokiaľ k obmedzeniu počtu požiadaviek dôjde, tester by mal vyhodnotiť efektivitu tohto obmedzenia a pokúsiť sa nastaviť parametre svojich nástrojov tak, aby sa rate limitingu vyhol. Pokiaľ sa rate limitingu úspešne vyhne, je potrebné rate limiting vyhodnotiť ako nedostatočný.
- **Testom je možné odhaliť tieto zraniteľnosti** – Chýbajúca prítomnosť rate limitingu
- **Testom je možné sprístupniť nadväzujúce testy** – Žiadne

Test HTTP hlavičiek

- **Úloha (test)** – Obsahujú HTTP odpovede webového servera požadované bezpečnostné HTTP hlavičky?
- **Popis testu** – Úlohou tohto testu je overiť, či webová aplikácia alebo API zahŕňa vo svojich HTTP odpovediach bezpečnostné HTTP hlavičky, ktoré poskytujú vyššiu úroveň zabezpečenia a ochrany pred rôznymi útokmi.
- **Náročnosť testu** – Jednoduchá
- **Ako vykonať test** – Test je možné vykonať zachytením HTTP odpovede na požiadavku na testovaný koncový bod a inšpekciou HTTP hlavičiek tejto odpovede.
- **Testom je možné odhaliť tieto zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky X-Frame-Options, Chýbajúca implementácia HTTP hlavičky X-Content-Type-Options, Chýbajúca implementácia HTTP hlavičky Referrer-Policy, Chýbajúca implementácia HTTP hlavičky Strict-Transport-Security, Chýbajúca implementácia HTTP hlavičky Strict-Transport-Security, Chýbajúca implementácia HTTP hlavičky Content-Security-Policy, Hlavička

X-Powered-By v HTTP odpovedi odhaľuje potenciálne citlivé informácie, Hlavička X-AspNet-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie, Hlavička X-AspNetMvc-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie, Hlavička Server v HTTP odpovedi odhaľuje potenciálne citlivé informácie
- **Testom je možné sprístupniť nadväzujúce testy** – Žiadne

Test direktív CSP

- **Úloha (test)** – Implementuje webová aplikácia dostatočne bezpečnú konfiguráciu CSP?
- **Popis testu** – Úlohou tohto testu je overiť, či webová aplikácia implementuje bezpečnú konfiguráciu CSP a zabraňuje tak úspešnosti zraniteľnosti XSS.
- **Náročnosť testu** – Jednoduchá
- **Ako vykonať test** – Test je možné vykonať zachytením HTTP odpovede na požiadavku na testovaný koncový bod a inšpekciou HTTP hlavičiek tejto odpovede. Hlavička Content-Security-Policy v sebe obsahuje jednotlivé direktívy, ktorých bezpečnostný dopad je potrebné individuálne preskúmať v spojení s aktuálnym kontextom webovej aplikácie.
- **Testom je možné odhaliť tieto zraniteľnosti** – Potenciálne nebezpečná konfigurácia CSP
- **Testom je možné sprístupniť nadväzujúce testy** – Žiadne

Test cookies

- **Úloha (test)** – Nastavuje webová aplikácia atribúty cookies bezpečným spôsobom?
- **Popis testu** – Úlohou tohto testu je overiť, či webová aplikácia alebo API nastavuje cookies s bezpečnými atribútmi a obmedzuje tak útočníka pri vykonávaní útokov ako XSS či CSRF.
- **Náročnosť testu** – Jednoduchá
- **Ako vykonať test** – Test je možné vykonať zachytením HTTP odpovede na požiadavku na testovaný koncový bod a inšpekciou HTTP hlavičiek tejto odpovede. Hlavička Set-Cookie v sebe obsahuje ako názov, či hodnotu daného cookie, tak jeho atribúty. Vzhľadom na použitie daného cookie je potrebné určiť, či sú jeho atribúty nastavené bezpečným spôsobom a jeho aktuálne nastavenie neohrozuje užívateľa webovej aplikácie.
- **Testom je možné odhaliť tieto zraniteľnosti** – Cookie nemá nastavený bezpečnostný atribút Secure, Cookie nemá nastavený bezpečnostný atribút

HttpOnly, Cookie nemá nastavený bezpečnostný atribút SameSite, Cookie so SameSite nastaveným na hodnotu None

- **Testom je možné sprístupniť nadväzujúce testy** – Žiadne

B Sprievodné texty k zraniteľnostiam

Zraniteľnosť neprítomnosti rate limitingu

- **Názov zraniteľnosti** – Chýbajúca prítomnosť rate limitingu
- **Popis zraniteľnosti** – Webová aplikácia neobmedzuje počet HTTP požiadaviek, ktoré môže útočník odoslať aplikácii v danom časovom intervale.
- **Príčiny** – Webová aplikácia, či webový server neimplementuje mechanizmus rate limitingu.
- **Prejavy** – Webová aplikácia nereaguje na zvýšenú intenzitu zasielania HTTP požiadaviek obmedzovaním klienta na určitý počet požiadaviek za stanovený čas.
- **Dopady** – Útočník môže vykonávať techniky enumerácie, ako je fuzzing, útoky hrubou silou a iné podobné útoky. Útoky hrubou silou môžu byť zamerané napríklad na prihlásenie do aplikácie alebo odhalenie adresárovej štruktúry webovej aplikácie a jej súborov.
- **Náprava / doporučenie** – Webová aplikácia by mala zaznamenávať počet požiadaviek prichádzajúcich z konkrétnych IP adries. Ak počet požiadaviek prekročí vopred stanovený počet požiadaviek v danom časovom intervale, IP adresa by mala byť dočasne zablokovaná, čím sa preruší potenciálny prebiehajúci útok.
- **Závažnosť** – Stredná
- **CVSS score** – 4.8
- **CVSS string** – AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:L

Zraniteľnosť chýbajúcej HTTP hlavičky X-Frame-Options

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky X-Frame-Options
- **Popis zraniteľnosti** – Hlavička X-Frame-Options HTTP je bezpečnostná hlavička používaná v HTTP odpovediach, ktorá umožňuje webovým serverom obmedziť spôsob vkladania ich stránok do prvkov iframe na iných stránkach. Táto hlavička bola navrhnutá na ochranu pred clickjackingom a podobnými útokmi, ktoré využívajú tento mechanizmus.
- **Príčiny** – Webová aplikácia neposiela v odpovediach HTTP hlavičku X-Frame-Options.
- **Prejavy** – Chýbajúca HTTP hlavička X-Frame-Options v HTTP odpovedi webovej aplikácie.
- **Dopady** – Zraniteľnosť umožňuje útočníkom vykonávať útoky typu clickjacking. Clickjacking je technika využívaná na útoky na používateľov webových

stránok, pri ktorej útočník využíva nevedomú interakciu používateľa s webovou stránkou. Táto technika je známa aj ako "presmerovanie používateľského rozhrania". Základnou myšlienkou clickjackingu je skryť škodlivý obsah (napríklad tlačidlo alebo odkaz) na webovej stránke pod iný, zdanlivo neškodný obsah. Keď používateľ klikne na zdanlivo bezpečný prvok, v skutočnosti klikne na škodlivý prvok, ktorý môže vykonať rôzne nežiaduce akcie, napríklad odoslať formuláre, zmeniť nastavenia účtu alebo dokonca nainštalovať škodlivý softvér. Clickjacking sa môže vykonávať rôznymi spôsobmi vrátane použitia priehľadných prvkov iframe, ktoré prekrývajú stránku, alebo skrytých tlačidiel umiestnených na pozadí stránky.

- **Náprava / doporučenie** – Odporúčame vám nakonfigurovať webový server tak, aby v HTTP odpovediach odosielal hlavičku X-Frame-Options.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – CVSS:3.1AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:L/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky X-Content-Type-Options

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky X-Content-Type-Options
- **Popis zraniteľnosti** – Hlavička X-Content-Type-Options zabraňuje webovému prehliadaču interpretovať prijaté súbory ako typ MIME iný, ako je uvedený v hlavičke Content-Type. Toto správanie sa nazýva "MIME sniffing". Chýbajúce nastavenie tejto hlavičky môže spôsobiť neúmyselnú interpretáciu prijatých údajov ako spustiteľných (napr. neúmyselná interpretácia typu MIME ako text/html môže viesť k zraniteľnosti Cross-Site Scripting).
- **Príčiny** – Webová aplikácia neposiela v odpovediach HTTP hlavičku X-Content-Type-Options.
- **Prejavy** – Chýbajúca HTTP hlavička X-Content-Type-Options v HTTP odpovedi webovej aplikácie.
- **Dopady** – Používatelia webovej aplikácie môžu bez svojho vedomia stiahnuť a spustiť škodlivé spustiteľné údaje (napr. JavaScript alebo súbor HTML).
- **Náprava / doporučenie** – Odporúčame nakonfigurovať webovú aplikáciu alebo server tak, aby v odpovediach HTTP odosielali hlavičku X-Content-Type-Options s hodnotou "nosniff".
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – CVSS:3.1AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:L/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky Referrer-Policy

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky Referrer-Policy
- **Popis zraniteľnosti** – HTTP hlavička Referrer-Policy HTTP určuje, aké informácie budú obsiahnuté v hlavičke Referer. Hlavička Referer odosiela informácie o pôvode požiadavky (adresa URL, z ktorej bola požiadavka odoslaná).
- **Príčiny** – Webová aplikácia neposiela v odpovediach HTTP hlavičku Referrer-Policy.
- **Prejavy** – Webová aplikácia neposiela v HTTP odpovediach hlavičku Referrer-Policy, ktorá určuje, aké informácie sa budú posielať v hlavičke Referer.
- **Dopady** – Kvôli chýbajúcej konfigurácii Referrer-Policy môže potenciálne dôjsť k úniku citlivých informácií prostredníctvom hlavičky Referer.
- **Náprava / doporučenie** – Odporúčame nastaviť Referrer-Policy na hodnotu strict-origin-when-cross-origin. Táto hodnota zachováva užitočnosť hlavičky Referer a zároveň zabraňuje úniku údajov.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – CVSS:3.1AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:L/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky Permissions-Policy

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky Strict-Transport-Security
- **Popis zraniteľnosti** – Hlavička Permissions-Policy, predtým známa ako Feature-Policy, je bezpečnostná HTTP hlavička, ktorá špecifikuje funkcie, ktoré sa môžu alebo nesmú používať v rámci webovej aplikácie. Hlavička definuje súbor politík, ktoré určujú, ku ktorým zdrojom môže webová aplikácia pristupovať.
- **Príčiny** – Webová aplikácia neposiela v odpovediach HTTP hlavičku Permissions-Policy.
- **Prejavy** – Webová aplikácia neodosiela v HTTP odpovediach hlavičku Permissions-Policy.
- **Dopady** – Vynechanie tejto hlavičky môže mať významný bezpečnostný vplyv na súkromie používateľa. Moderné webové prehliadače používajú na interakciu so zdrojmi zariadenia, z ktorého používateľ pristupuje k webovým stránkam aplikácie, niekoľko špecifických rozhraní API. V prípade zneužitia môže útočník komunikovať s fyzickými zariadeniami pripojenými k počítaču používateľa, ako je obrazovka, geolokácia, mikrofón alebo zariadenia USB, a tak môže pristúpiť k citlivým informáciám, šíriť škodlivý software alebo sa pokúsiť

ukradnúť citlivé informácie zo schránky na kopírovanie.

- **Náprava / doporučenie** – Odporúčame implementovať HTTP hlavičku Permissions-Policy v HTTP odpovediach aplikácie. Implementácia hlavičky Permissions-Policy do veľkej miery závisí od potrieb aplikácie, vo všeobecnosti však odporúčame zakázať nepotrebný prístup.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky Strict-Transport-Security

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky Strict-Transport-Security
- **Popis zraniteľnosti** – HTTP Strict Transport Security (HSTS) je mechanizmus, ktorý pomáha chrániť používateľov webovej aplikácie pred útokmi ako je zmena protokolu z HTTPS na HTTP alebo krádež súborov cookie. Táto hlavička umožňuje webovému serveru zabezpečiť, aby používateľskí agenti komunikujúci s webovou aplikáciou používali iba zabezpečené pripojenie HTTPS (namiesto nezabezpečeného protokolu HTTP).
- **Príčiny** – Webová aplikácia neposiela v odpovediach HTTP hlavičku Strict-Transport-Security.
- **Prejavy** – Webová aplikácia neodosiela v HTTP odpovediach hlavičku Strict-Transport-Security (HSTS).
- **Dopady** – Absencia hlavičky HTTP Strict-Transport-Security zvyšuje riziko bezpečnostných incidentov a ohrozuje súkromie používateľov a integritu dát.
- **Náprava / doporučenie** – Túto zraniteľnosť možno odstrániť implementáciou hlavičky HTTP Strict-Transport-Security. Ako ďalšie odporúčanie je možné uviesť službu HSTS preload. Spoločnosť Google ponúka službu "HSTS preload". Zaregistrovaním domény a dodržiavaním pravidiel môžete zaručiť, že webové prehliadače budú pri prvom pripojení (pred odoslaním hlavičky Strict-Transport-Security) pristupovať k doméne len prostredníctvom zabezpečeného protokolu HTTPS. Tento preload zoznam poskytuje spoločnosť Google, takže nie je oficiálnym prvkom špecifikácie HSTS, avšak používajú ho všetky webové prehliadače.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky Content-Security-Policy

- **Názov zraniteľnosti** – Chýbajúca implementácia HTTP hlavičky Content-Security-Policy
- **Popis zraniteľnosti** – Content Security Policy (CSP) vyžaduje starostlivé nastavenie a presnú definíciu. Ak je CSP implementovaná, má významný vplyv na spôsob, akým webové prehliadače zobrazujú webové stránky. CSP pomáha detegovať a predchádzať širokému spektru útokov vrátane XSS a ďalších Cross-Site útokov.
- **Príčiny** – HTTP hlavička Content-Security-Policy nie je v rámci webovej aplikácie zasielaná v HTTP odpovediach.
- **Prejavy** – Chýbajúca HTTP hlavička Content-Security-Policy v HTTP odpovedi webovej aplikácie.
- **Dopady** – Chýbajúca implementácia CSP nemá priamy vplyv na bezpečnosť. Ak však vo webovej aplikácii existuje zraniteľnosť XSS, správna konfigurácia CSP môže zabrániť jej úspešnému zneužitiu. Chýbajúca implementácia CSP teda pripravuje webovú aplikáciu o ďalšiu vrstvu zabezpečenia.
- **Náprava / doporučenie** – Odporúčame implementovať konfiguráciu CSP do HTTP odpovedí webovej aplikácie.
- **Závažnosť** – Nízka
- **CVSS score** – 6.1
- **CVSS string** – AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky X-Powered-By

- **Názov zraniteľnosti** – Hlavička X-Powered-By v HTTP odpovedi odhaľuje potenciálne citlivé informácie
- **Popis zraniteľnosti** – Hlavička X-Powered-By v HTTP odpovedi odhaľuje potenciálne citlivé informácie.
- **Príčiny** – HTTP hlavička X-Powered-By je v rámci webovej aplikácie zasielaná v HTTP odpovediach.
- **Prejavy** – Prítomnosť HTTP hlavičky X-Powered-By v HTTP odpovedi webovej aplikácie.
- **Dopady** – Táto zraniteľnosť umožňuje útočníkovi efektívnejšie vyhľadávať zraniteľnosti špecifické pre konkrétny webový server či software, ktorý webová aplikácia využíva. Ak by sa útočníkovi podarilo objaviť takúto zraniteľnosť, mohol by server kompromitovať. Konkrétny bezpečnostný dopad by závisel od konkrétnej zraniteľnosti danej verzie webového servera či software.
- **Náprava / doporučenie** – Odporúčame úplne odstrániť hlavičku X-Powered-By zo všetkých odpovedí HTTP.

- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky X-AspNet-Version

- **Názov zraniteľnosti** – Hlavička X-AspNet-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie
- **Popis zraniteľnosti** – Hlavička X-AspNet-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie.
- **Príčiny** – HTTP hlavička X-AspNet-Version je v rámci webovej aplikácie zasielaná v HTTP odpovediach.
- **Prejavy** – Prítomnosť HTTP hlavičky X-AspNet-Version v HTTP odpovedi webovej aplikácie.
- **Dopady** – Táto zraniteľnosť umožňuje útočníkovi efektívnejšie vyhľadávať zraniteľnosti špecifické pre konkrétny webový server či software, ktorý webová aplikácia využíva. Ak by sa útočníkovi podarilo objaviť takúto zraniteľnosť, mohol by server kompromitovať. Konkrétny bezpečnostný dopad by závisel od konkrétnej zraniteľnosti danej verzie webového servera či software.
- **Náprava / doporučenie** – Odporúčame úplne odstrániť hlavičku X-AspNet-Version zo všetkých odpovedí HTTP.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky X-AspNetMvc-Version

- **Názov zraniteľnosti** – Hlavička X-AspNetMvc-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie
- **Popis zraniteľnosti** – Hlavička X-AspNetMvc-Version v HTTP odpovedi odhaľuje potenciálne citlivé informácie.
- **Príčiny** – HTTP hlavička X-AspNetMvc-Version je v rámci webovej aplikácie zasielaná v HTTP odpovediach.
- **Prejavy** – Prítomnosť HTTP hlavičky X-AspNetMvc-Version v HTTP odpovedi webovej aplikácie.
- **Dopady** – Táto zraniteľnosť umožňuje útočníkovi efektívnejšie vyhľadávať zraniteľnosti špecifické pre konkrétny webový server či software, ktorý webová aplikácia využíva. Ak by sa útočníkovi podarilo objaviť takúto zraniteľnosť, mohol by server kompromitovať. Konkrétny bezpečnostný dopad by závisel od konkrétnej zraniteľnosti danej verzie webového servera či software.

- **Náprava / doporučenie** – Odporúčame úplne odstrániť hlavičku X-AspNetMvc-Version zo všetkých odpovedí HTTP.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť chýbajúcej HTTP hlavičky Server

- **Názov zraniteľnosti** – Hlavička Server v HTTP odpovedi odhaľuje potenciálne citlivé informácie
- **Popis zraniteľnosti** – Hlavička Server v HTTP odpovedi odhaľuje potenciálne citlivé informácie.
- **Príčiny** – HTTP hlavička Server je v rámci webovej aplikácie zasielaná v HTTP odpovediach.
- **Prejavy** – Prítomnosť HTTP hlavičky Server v HTTP odpovedi webovej aplikácie.
- **Dopady** – Táto zraniteľnosť umožňuje útočníkovi efektívnejšie vyhľadávať zraniteľnosti špecifické pre konkrétny webový server či software, ktorý webová aplikácia využíva. Ak by sa útočníkovi podarilo objaviť takúto zraniteľnosť, mohol by server kompromitovať. Konkrétny bezpečnostný dopad by závisel od konkrétnej zraniteľnosti danej verzie webového servera či software.
- **Náprava / doporučenie** – Odporúčame úplne odstrániť hlavičku Server zo všetkých odpovedí HTTP.
- **Závažnosť** – Nízka
- **CVSS score** – 5.8
- **CVSS string** – AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Zraniteľnosť nebezpečnej konfigurácie CSP

- **Názov zraniteľnosti** – Potenciálne nebezpečná konfigurácia CSP
- **Popis zraniteľnosti** – Content Security Policy (CSP) vyžaduje starostlivé nastavenie a presnú definíciu. Ak je CSP implementovaná, má významný vplyv na spôsob, akým webové prehliadače zobrazujú webové stránky. CSP pomáha detegovať a predchádzať širokému spektru útokov vrátane XSS a ďalších Cross-Site útokov. Implementovaná konfigurácia CSP nedostatočne chráni webovú aplikáciu pred útokmi XSS.
- **Príčiny** – Webová aplikácia špecifikuje direktívy CSP príliš permisívne.
- **Prejavy** – HTTP hlavička Content-Security-Policy zasielaná v rámci HTTP odpovedí obsahuje chybne konfigurované direktívy.

- **Dopady** – Chybná implementácia CSP nemá priamy vplyv na bezpečnosť. Ak však vo webovej aplikácii existuje zraniteľnosť XSS, správna konfigurácia CSP môže zabrániť jej úspešnému zneužitiu. Chybná implementácia CSP teda pripravuje webovú aplikáciu o ďalšiu vrstvu zabezpečenia.
- **Náprava / doporučenie** – Odporúčame implementovať konfiguráciu CSP do HTTP odpovedí webovej aplikácie menej permisívnym spôsobom.
- **Závažnosť** – Nízka
- **CVSS score** – 3.1
- **CVSS string** – AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Zraniteľnosť cookie bez atribútu Secure

- **Názov zraniteľnosti** – Cookie nemá nastavený bezpečnostný atribút Secure
- **Popis zraniteľnosti** – Jeden alebo viac súborov cookie nemajú nastavený atribút Secure. Keď je súbor cookie nastavený s atribútom Secure, dáva prehliadaču pokyn, že daný cookie je možné použiť len v prípade, že sa jedná o zabezpečený prenos protokolom HTTPS.
- **Príčiny** – Cookie nedisponuje bezpečnostným atribútom Secure.
- **Prejavy** – Hlavička Set-Cookie v HTTP odpovedi nenastavuje danému cookie atribút Secure.
- **Dopady** – Pokiaľ webová aplikácia používa protokol HTTP, tento súbor cookie bude prenesený cez spojenie HTTP v nezašifrovanej forme. Ak je tento súbor cookie citlivý (napríklad session cookie), útočník ho môže pri útoku man-in-the-middle zachytiť a ukradnúť tak reláciu obete.
- **Náprava / doporučenie** – Odporúčame citlivým cookies nastavovať bezpečnostný atribút Secure.
- **Závažnosť** – Nízka
- **CVSS score** – 3.1
- **CVSS string** – AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Zraniteľnosť cookie bez atribútu HttpOnly

- **Názov zraniteľnosti** – Cookie nemá nastavený bezpečnostný atribút HttpOnly
- **Popis zraniteľnosti** – Jeden alebo viac súborov cookie nemajú nastavený atribút HttpOnly. Keď je súbor cookie nastavený s atribútom HttpOnly, dáva prehliadaču pokyn, že k súboru cookie môže pristupovať len server a nie skripty na strane klienta.
- **Príčiny** – Cookie nedisponuje bezpečnostným atribútom HttpOnly.

- **Prejavy** – Hlavička Set-Cookie v HTTP odpovedi nenastavuje danému cookie atribút HttpOnly.
- **Dopady** – Keď má daný cookie nastavený atribút HttpOnly, webový prehliadač obmedzí prístup k obsahu cookie len na strane servera a nepovolí prístup prostredníctvom klienta cez skriptovacie jazyky ako napr. JavaScript. Pokiaľ cookie týmto atribútom nedisponuje, zvýši sa tak potenciálny dopad útokov typu Cross-Site Scripting (XSS), pretože útočníci môžu potenciálne získať prístup k obsahu cookie klienta a ukradnúť citlivé informácie ako napríklad identifikátory užívateľských relácií či iné citlivé údaje.
- **Náprava / doporučenie** – Odporúčame citlivým cookies nastavovať bezpečnostný atribút HttpOnly.
- **Závažnosť** – Nízka
- **CVSS score** – 3.1
- **CVSS string** – AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Zraniteľnosť cookie bez atribútu SameSite

- **Názov zraniteľnosti** – Cookie nemá nastavený bezpečnostný atribút SameSite
- **Popis zraniteľnosti** – Cookie nedisponuje bezpečnostným atribútom SameSite.
- **Príčiny** – Hlavička Set-Cookie v HTTP odpovedi nenastavuje danému cookie atribút SameSite.
- **Prejavy** – Keď je súbor cookie nastavený s atribútom SameSite, dáva prehliadaču pokyn, akým spôsobom sa má zachovať pri cross-origin požiadavkách. Pri chybnej konfigurácii atribútu SameSite môže byť cookie náchylný k zneužitiu zraniteľnosťou CSRF.
- **Dopady** – Odporúčame jednotlivým cookies nastavovať bezpečnostný atribút SameSite. Predvolené správanie webových prehliadačov sa môže líšiť pri spracovaní súborov cookie pri cross-origin požiadavkách, takže konečné rozhodnutie o odoslaní súboru cookie je v tomto kontexte nepredvídateľné. Atribút SameSite by mal byť nastavený v každom súbore cookie, aby vývojári vynútili očakávaný výsledok.
- **Náprava / doporučenie** – Odporúčame jednotlivým cookies nastavovať bezpečnostný atribút SameSite. Predvolené správanie webových prehliadačov sa môže líšiť pri spracovaní súborov cookie pri cross-origin požiadavkách, takže konečné rozhodnutie o odoslaní súboru cookie je v tomto kontexte nepredvídateľné. Atribút SameSite by mal byť nastavený v každom súbore cookie, aby vývojári vynútili očakávaný výsledok.

- **Závažnosť** – Nízka
- **CVSS score** – 3.1
- **CVSS string** – AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Zraniteľnosť cookie s atribútom SameSite nastaveným na None

- **Názov zraniteľnosti** – Cookie so SameSite nastaveným na hodnotu None
- **Popis zraniteľnosti** – Jeden alebo viac súborov cookie majú nastavený atribút SameSite na hodnotu None.
- **Príčiny** – Cookie disponuje bezpečnostným atribútom SameSite s potenciálne nebezpečnou hodnotou None.
- **Prejavy** – Hlavička Set-Cookie v HTTP odpovedi nastavuje danému cookie atribút SameSite na hodnotu None.
- **Dopady** – Keď je súbor cookie nastavený s atribútom SameSite, dáva prehliadaču pokyn, akým spôsobom sa má zachovať pri cross-origin požiadavkách. Pri chybnej konfigurácii atribútu SameSite môže byť cookie náchylný k zneužitiu zraniteľnosťou CSRF.
- **Náprava / doporučenie** – Odporúčame jednotlivým cookies nastavovať bezpečnostný atribút SameSite. Predvolené správanie webových prehliadačov sa môže líšiť pri spracovaní súborov cookie pri cross-origin požiadavkách, takže konečné rozhodnutie o odoslaní súboru cookie je v tomto kontexte nepredvídateľné. Atribút SameSite by mal byť nastavený v každom súbore cookie, aby vývojári vynútili očakávaný výsledok.
- **Závažnosť** – Nízka
- **CVSS score** – 3.1
- **CVSS string** – AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

C Obsah elentronickej prílohy

Obsahom elektronickej prílohy je obsah adresára obsahujúceho zdrojový kód nástroja PtWebDA, ktorý bol v rámci tejto diplomovej práci vyvíjaný. Obsahom tejto elektronickej prílohy je taktiež zdrojový kód zraniteľnej webovej aplikácie a samotný text tejto diplomovej práce vo formáte PDF.

```
/ ..... Koreňový adresár priloženého archívu
├── ptwebda ..... Adresár s nástrojom PtWebDA
│   ├── modules ..... Adresár s jednotlivými modulmi
│   │   ├── __init__.py
│   │   ├── basemodule.py ..... Bázový modul
│   │   ├── cookies.py ..... Modul bezpečnostných HTTP hlavičiek
│   │   ├── csp.py ..... Modul analýzy CSP
│   │   ├── headers.py ..... Modul analýzy HTTP hlavičiek
│   │   ├── ratelimit.py ..... Modul rate limitingu
│   │   └── utils ..... Zložka s pomocnými nástrojmi
│   │       ├── helpers.py .... Súbor s funkciami pre podporu formátovanie výstupu
│   │       └── http.py ..... Súbor s funkciami parsovania HTTP požiadaviek
│   ├── ptwebda.py ..... Hlavný súbor nástroja obsahujúci main metódu
│   └── requirements.txt ..... Zoznam knižníc potrebných pre chod nástroja
├── vulnserver ..... Adresár so zraniteľnou webovou aplikáciou
│   └── server.py ..... Skript obsahujúci zdrojový kód zraniteľnej webovej aplikácie
└── DP_Pis.pdf ..... Diplomová práca (PDF)
```