FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Bezpečnost webových aplikací a její penetrační testování
Student:	Aleš Répáš
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1) Seznamte se s problematikou bezpečnosti webových aplikací, jak ji definuje OWASP Top 10 pro rok 2021.
2) Nastudujte možnosti nástroje OWASP ZAP, popište jeho hlavní vlastnosti a možnosti rozšíření.
3) Navrhněte modul pro OWASP ZAP, který bude detekovat potenciální zranitelnosti typu Cross Site Request Forgery.
4) Tento modul naprogramujte a otestujte. Získejte souhlas majitele vhodné webové aplikace pro otestování nástroje v reálném prostředí.
5) Diskutujte svá zjištění.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 29. září 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Bezpečnost webových aplikací a její penetrační testování

Aleš Répáš

Katedra počítačových systémů
Vedoucí práce: Ing. Josef Kokeš

11. května 2022

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Josefu Kokešovi za pomoc, trpělivost a efektivní komunikaci během psaní této práce. Dále bych chtěl poděkovat Petru Váchalovi za cenné rady ohledně webové bezpečnosti. V neposlední řadě mé poděkování patří Marcele Křížové za veškerou podporu při vytváření této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2022 . . . . . .

Odkaz na tuto práci

Répáš, Aleš. *Bezpečnost webových aplikací a její penetrační testování.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá rozšířením nástroje OWASP ZAP o schopnost detekce zranitelnosti Cross Site Request Forgery. V řešení byla použita metoda manipulace s hlavičkou webové žádosti. Vytvořené řešení poskytuje automatické vyhledávání zranitelných míst ve webové aplikaci. Hlavním výsledkem je přesnější penetrační testování nástrojem ZAP.

Klíčová slova OWASP ZAP, zranitelnost Cross Site Request Forgery, webová aplikace, penetrační testování, kyberbezpečnost

Abstract

This bachelor thesis deals with the extension of the OWASP ZAP tool with the Cross Site Request Forgery vulnerability detection capability. In the solution, the web request header manipulation method was used. The developed solution provides an automatic search for vulnerabilities in a web application. The main result is a more accurate penetration testing with the ZAP tool.

Keywords OWASP ZAP, Cross Site Request Forgery vulnerability, web application, penetration testing, cybersecurity

Obsah

Seznam obrázků

Úvod

V době psaní této bakalářské práce je svět zahlcen informačními technologiemi. Stovky milionů lidí mají k dispozici alespoň jeden počítač. Každé toto zařízení komunikuje s desítkami dalších a cestují mezi nimi data. S nárůstem zdrojových kódů operujících v aplikacích, které běží na těchto zařízeních, vzrůstá i potenciální riziko bezpečnostních slabin. Následné zneužití těchto zranitelných míst už je jen otázkou času.

Webové aplikace v sobě ukládají důležitá osobní data jako e-mail, telefonní čísla, jména, hesla a čísla kreditních karet. Tudíž je jejich zabezpečení nezbytné. Bezpečnost webové aplikace nutně nemusí být během vývoje prioritní. Závadou v aplikaci dokáže útočník poškodit cíl a antivirus nebo firewall tomu nezabrání. V případě zranitelnosti Cross Site Request Forgery (CSRF) útočník může oběti odcizit finanční prostředky nebo uživatelské účty akcí, která se tváří jako legitimní. Penetrační testování nabízí způsob, jak zranitelnost objevit dříve než útočník a tím pádem zamezit její zneužití.

Penetrační testování webových aplikací může být velice časově náročné. Využití času pro zkoumání lze zefektivnit různými nástroji jako je Open Web Application Security Project (OWASP) Zed Attack Proxy (ZAP) nebo BurpSuite od PortSwigger. Tyto nástroje nabízí možnost spuštění automatických testů, díky kterým se lze v mezičase soustředit na objevování jiných, unikátních slabin v testovaném prostředí.

V této bakalářské práci jsem navrhl, naprogramoval a otestoval rozšíření pro nástroj ZAP, které na daném webovém serveru automaticky vyhledává a testuje místa zranitelná na slabinu CSRF. První kapitolu zaměřuji na úvod do problematiky kybernetické bezpečnosti a ukážu události, které obor historicky ovlivnily a zformulovaly. Dále ve druhé kapitole popisuji penetrační testování a používané nástroje. Ve třetí představuji webovou zranitelnost CSRF, způsob jejího zabezpečení a demonstraci v cvičném prostředí. Čtvrtou a pátou kapitolu zaměřuji na analýzu současných způsobů detekce zranitelnosti, návrhu jejího řešení, implementaci rozšíření a testování.

Kapitola 1

Úvod do kybernetické bezpečnosti

Každým dnem se kybernetické útoky v digitálním světě stávají mnohem inovativnější, sofistikovanější a neustále se vyvíjí. Uživatelé vyžadují bezpečnost svých dat a poskytovatelé služeb chtějí chránit svoje systémy. Ochrana dat, systémů, sítí a aplikací vyžaduje znalost kybernetické bezpečnosti a jejích principů. Společnosti svá zařízení připojují k internetu, čímž vzrůstá riziko útoku na ně, a z toho důvodu se musí chovat zodpovědně ohledně otázky bezpečnosti uživatelských dat. [3]

Existuje mnoho druhů kybernetických útoků. První kybernetické útoky se začaly objevovat s počátkem přenosu dat mezi počítači. Od té doby vzniklo mnoho druhů útoků, které dělíme dle určitých kritérií. Tato kritéria nám umožňují nejen popsat průběh útoku, ale i napomáhají prevenci či přípravě na útok.

1.1 První viry a antiviry

Myšlenku o modelu s podobou biologického viru přivedl na svět John von Neumann ve své práci *Theory of self reproducing automata* ve snaze pochopit biologickou evoluci a samoreprodukci. [4]

1.1.1 Virus

Počítačový virus je škodlivý software, který nakazí zařízení připojené k síti, způsobí operační problémy na nich, sám sebe zreplikuje a proces infekce opakuje. Virus na infikovaném zařízení může krást hesla, rozesílat infikované emaily, ničit soubory nebo poškodit celý systém. [5]

1.1.2 Creeper a Reaper

Na počátku sedmdesátých let devatenáctého století se v americké vojenské síti Advanced Research Projects Agency Network (ARPANET), z které se časem vyvinul internet, objevil program Creeper, který množil do vzdálených systémů v síti zprávu motivující k zastavení programu. Reakcí na Creeper se objevil program Reaper s podobnou úlohou. Reaper množil sám sebe do vzdálených systémů a hledal program Creeper, který v případě úspěchu odstranil. [6]

Programy Creeper a Reaper splňují definici počítačového viru a považujeme je za první viry. Program Reaper zašel ještě dále, můžeme jej považovat za předchůdce moderních antivirových programů. A jelikož i sám sebe replikoval, tak byl vůbec prvním případem počítačového červa.

1.1.3 Skrytý virus Brain

V roce 1986 vytvořili pákistánští bratři Farooq Alvi virus Brain, který se šířil po počítačích International Business Machines (IBM) a jejich kompatibilitách. Brain se uložil do boot sektoru počítače, změnil název disku a při pokusu o čtení z infikovaného sektoru zobrazoval původní data. Brain byl navržen jako ochrana proti pirátství zdravotního softwaru a měl ve svém kódu přiložené kontaktní údaje na tvůrce pro případnou opravu nakaženého stroje. Ve své podstatě nic závažného nedělal, ale kvůli své shovívavosti získal reputaci prvního skrytého viru (anglicky stealth virus). [7]

1.1.4 Vienna a antivir

Roku 1987 se objevil virus Vienna. Nakazil zařízení Disk Operating System (DOS) na kterém záměrně poškozoval data a mazal soubory. Byl v tu dobu natolik závažný, že počítačový vědec Brend Fix dostal od svého kolegy Rolfa Burgera úkol najít způsob jak virus zneškodnit. Brend vytvořil vůbec první specializovaný antivirový software s úkolem rozpoznat a zastavit virus Vienna ještě před tím, než pronikne do systému. [8]

Antiviry začaly nabývat na popularitě v následujících desetiletí. Dnes je nezbytné, aby každé zařízení bylo chráněné antivirem.

1.2 Příchod internetu

V době kdy se z ARPANETu začal stávat internet vzrůstala četnost virů a objevily se i trojské koně. V důsledku Studené války se vyvinul obor kybernetické špionáže. Začala se formovat kyberbezpečnost, na trhu se v osmdesátých letech devatenáctého století objevil chránící software - první vydání Ultimate Virus Killer pro Atari ST, první verze československého antiviru NOD a VirusScan od společnosti McAfee. V České Republice byla založena společnost

Avast. Antiviry této doby jednoduše hledaly části virů a rozšířené verze měly zabudován modifikátor programů, který zajišťoval, že virus považoval zařízení za již nakažené. Antiviry byly schopné odrážet útoky pouze známých virů a vyžadovaly objevit způsob, jak je průběžně aktualizovat o nové viry. [9]

1.2.1 Ransomware

Ransomware je škodlivý program, který po nakažení zařízení zašifruje uživatelská data a pro dešifrování požaduje zaplacení nějaké částky. Určité případy dokonce požadují platbu do předem specifikovaného časového intervalu, jinak ransomware data smaže. Používá se asymetrické šifrování, tedy pro každé nakažené zařízení dva různé unikátní klíče speciálně pro šifrování a dešifrování. Šifrovacím klíčem jsou data zašifrována a dešifrovací klíč k němu má k dispozici útočník u sebe. Pokud je částka zaplacena, může oběti vydat dešifrovací klíč pro zachránění dat, avšak ne vždy se tak stane. Dešifrování dat bez dešifrovacího klíče je sice teoreticky možné, ale nesmírně nákladné pro výpočetní techniku a v praxi se tato metoda nepoužívá. [10]

1.2.2 WannaCry a EternalBlue

Dne 12. května 2017 se objevil ransomware *WannaCry*. Rozsahem jeho působení byla celosvětová síť internetu, tedy útok nebyl cílený. Útočil na jednotlivce, státní organizace, nemocnice, školy a různé společnosti. Největší poškození způsobil v britském zdravotnickém systému, kde napadl kolem 600 organizací, včetně specializovaných lékařských služeb, psychiatrickou péči a ambulanci. Zdravotnické přístroje, jako skenery magnetické rezonance byly taktéž napadeny. Útok byl zastaven objevením a spuštěním zabudovaného kill switche, ale v důsledku infikovaných zařízení muselo zdravotnictví přejít na neelektronický způsob komunikace. [11]

Jádrem WannaCry útoku byl hack údajně vyvinutý severoamerickou National Security Agency (NSA) zvaný *EternalBlue* a zadní vrátka k přístupu do zařízení poskytoval nástroj *DoublePulsar*. EternalBlue získal svou reputaci díky protiteroristickým misím a sbíráním inteligence. Trvalo pět let než si Microsoft všiml existence hacku. Hackerská skupina s názvem *Shadow Brokers* získala přístup k EternalBlue a zveřejnila ho 14. dubna 2017 přes svůj twitter účet. Dva měsíce před útokem WannaCry Microsoft vytvořil aktualizaci na ochranu před EternalBlue, avšak organizace neaktualizují svoje systémy pravidelně, čímž přetrvávalo riziku útoku. Zranitelnost EternalBlue využívala slabinu *CVE-2017-0144* v síťovém komunikačním protokolu SMBv1, jelikož protokol chybně zpracovává škodlivé pakety od útočníků. [12]

Ransomware WannaCry zašifroval data a vyžadoval zaplacení tři sta až šest set dolarů v kryptoměně BitCoin za dešifrování a vrácení zařízení do původního stavu. Spojené státy americké a Spojené království trvají na tom, že útok pocházel ze Severní Koreji. [13]

Obrázek 1.1: Zpráva, která se zobrazila na zařízení nakaženém WannaCry [1]

1.3 Současný stav

V současné době je mnoho druhů útoků a samotná kyberbezpečnost se dělí na spoustu odvětví. Známé útoky se zaznamenávají, zkoumají a vytváří se proti nim opatření. Jejich různorodost je vysoká a znát přesně každý útok by bylo pro člověka nesmírně náročné. Řešení nabízí seznam Common Vulnerabilities and Exposures (CVE) publikován od roku 1999 organizací MITRE. Jedná se o bezplatnou databázi identifikující a třídící zranitelnosti v aplikacích a systémech. Díky seznamu můžeme lehce zjišťovat informace ohledně kyberbezpečnosti a lehce aktualizovat bezpečnostní chyby a závady. [14]

CVE od MITRE je široký pojem pro potřeby webových aplikací. Webové aplikace bývají klíčové místo pro útoky a sebemenší závada může implikovat problémy, které je nutné řešit. Pokud je aplikace zranitelná, může být ohrožen i server, na kterém běží. A pokud je možné dostat se skrze zranitelnost na server, může být ohrožena celá síť. Můžeme si tak webové aplikace představit jako přední linii, kde může vzniknout problém. Teoreticky i běžný uživatel dokáže nevědomky objevit zranitelnost, pokud například opakovaně manipuluje s textovým vstupem se soubory. Z tohoto důvodu je dobré již při vývoji webové aplikace mít seznam zranitelností webových aplikací a pohlídat si, že je dostatečně zabezpečena již při návrhu. Jednu takovou možnost nabízí nadace OWASP.

1.3.1 Open Web Application Security Project a webové aplikace

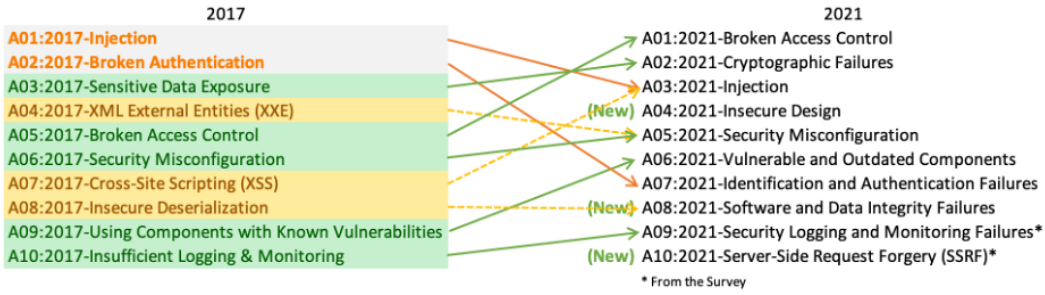
OWASP je nezisková nadace, která vydává doporučení ohledně vývoje, nakupování, udržování důvěryhodnosti a bezpečnosti webových softwarových aplikací. Je známá díky svému publikovanému seznamu *Top 10 nejčastějších zranitelností ve webových aplikací*. Nadace působí jako platforma shromažďující informace ohledně rizicích nejčastějších bezpečnostních chyb webových aplikací a umožňuje navazovat kontakty mezi odborníky na počítačové vědy, informační technologie a kyberbezpečnost. [15]

OWASP také vydává jednoduchý zaškrtávací seznam orientovaný na požadavky dodavatelů pro penetrační testování zabezpečení webových aplikací, pokrývající větší rozsah typů zranitelností, než je seznam Top 10. Aplikace otestované podle požadavků seznamu zaručují, že jsou jejich služby dostatečně zabezpečené. [16]

Nadace také nabízí veřejně dostupný penetrační nástroj ZAP.

1.3.2 Seznam OWASP Top 10

Top 10 je seznam prvně vydaný v roce 2003 představující nejzávažnější bezpečnostní rizika pro webové aplikace seřazená podle velikosti závažnosti chyby a četnosti opakování. Ke každé slabině je uveden postup k její nápravě. Obsah listu je vytvořen podle zpráv bezpečnostních analytiků z celého světa a seznam je po uplynutí dvou až tří let periodicky aktualizován. Primárním účelem je nabídnout vývojářům a testerům nejaktuálnější přehled o zranitelnostech, aby je mohli obsáhnout ve svých projektech a tím snížit množství jejich výskytů. [17]



Obrázek 1.2: OWASP Top 10 2021[2]

Naposledy vydaný list je z roku 2021. V textu níže je krátce popsána každá kategorie s její prevencí. Text je seřazen od nejčetnější závady po nejméně častou podle OWASP. Přestože OWASP vydává tento list, nejedná se zdaleka o všechny zranitelnosti týkající se webové bezpečnosti. Zranitelností a chyb je mnohem více, jedná se však o výčet deseti nejčastějších.

1.3.2.1 Porušené řízení přístupu

Nejzávažnější chybu představuje chybné nastavení přístupu. Na webové aplikaci má uživatel přiřazené přístupy k datům, se kterými může pracovat. V případě chyby se může stát, že uživatel se i přes neautorizovaný přístup dostane k údajům ostatních uživatelů nebo modifikuje či smaže data mimo uživatelská práva. Chybu může představovat chybějící Application Programming Interface (API) definice pro žádosti typu POST, PUT a DELETE, zvýšení privilegií na administrátorský účet během přihlášení jako běžný uživatel nebo přístup k účtu jiného uživatele. Na ochranu přispějí integrační a funkcionální přístupové testy během vývoje. [18]

1.3.2.2 Kryptografická selhání

Druhou nejzávažnější chybou se staly *kryptografické chyby*. V předchozím vydání OWASP top 10 nesly název *vystavení citlivých dat*. Jedná se o chyby kryptografického původu, kde je spíš kladen důraz na nedostatečnou ochranu dat. Data jako hesla, uživatelské údaje, čísla kreditních karet, zdravotní údaje a tajemství společnosti vyžadují extra krok ochrany, což je dokonce požadavek General Data Protection Regulation (GDPR). Odpovědnost za tuto chybu může nést již prolomený nebo neaktualizovaný šifrovací software nebo knihovna, přenášení dat v plaintextu, nedostatečná opatření při práci se šifrovacími klíči či opakované používání šifrovacích klíčů. Pro případ prevence je nutné najít místa, kde se pracuje s citlivými daty, a ta dostatečně ošetřit a šifrovat nebo dokonce i zvážit, zda je nutné data ukládat. Hesla je nutné hashovat opakovaně silnou funkcí s dynamickou solí. V případě šifrovacích algoritmů je mnohem lepší použít veřejně známé knihovny než si psát vlastní, kde je riziko nedostatečného otestování. [19]

1.3.2.3 Injekce

Pro rok 2021 OWASP usadil injekce na třetí místo. Známým příkladem injekcí je Structured Query Language (SQL), kdy v případě formuláře je do textového pole vložen SQL dotaz, který přeruší zamýšlené provádění zdrojového kódu a vynutí chování, které může zobrazit či v horším případě zlikvidovat data v databázi, se kterou webová aplikace pracuje. Chybu způsobuje aplikací nedostatečné ověření uživatelem odeslaných dat nebo pokus vývojáře odesílat dynamické dotazy do databáze bez dostatečného ověření možnosti ukončení takového dotazu vstupem. Alternativní řešení při vývoji nabízí API nebo Object Relational Mapping (ORM), které lze přímo nalinkovat na práci s daty v databázi. [20]

1.3.2.4 Nezabezpečený design

Již ve vývoji webové aplikace je nutné myslet na její zabezpečení. Avšak pokud se v návrhu vyskytne chyba, tak implementace jí těžko vyřeší. Jedná se o místa, kde se pracuje s vyššími oprávněními, než je nutné, nebo neohraničení uživatelského vstupu, což může způsobit přetečení bufferu. Jelikož tato zranitelnost je spíše teoretická, je celé její řešení již na prvním kroku před implementací aplikace, a to je návrh. Tedy pokud se stane, že aplikace není navržena bezpečně, může se stát, že budeme muset celou aplikaci předělat od návrhu. [21]

1.3.2.5 Chybná konfigurace zabezpečení

Při nasazování webových aplikací většina přichází ve stavu, který dovoluje provozovatelům se seznámit s nově nainstalovaným prostředím, nastavit si ho podle požadavků a vytvořit si vlastní účty s potřebnými privilegii. V momentě nasazení webové aplikace by provozovatel měl zajistit deaktivování výchozího administrátorského účtu a zkontrolovat nastavení aplikace dle potřeb. [22]

1.3.2.6 Zranitelné a zastaralé komponenty

Aplikace jsou tvořené komponentami, které jim přidávají funkcionalitu. Komponenty mohou být aktualizacemi zlepšovány a rozšiřovány nebo jim může být přidáno zabezpečení v případě, že se v nich objevila závada. Webové aplikace jsou v mnoha případech tvořené více komponentami a pokud neznáme jejich verze, nemusí nutně být bezpečné pro používání. Na ošetření je třeba aktualizace komponent na nejnovější verzi. Provozovatelé webových aplikací by ideálně měli pravidelně provádět aktualizace komponent a odstranit komponenty, které již nejsou potřebné pro chod aplikace. [23]

1.3.2.7 Selhání identifikace a ověření

V případě přihlašovacích údajů se silně doporučuje nesdílet nikde svá uživatelská jména a hesla. Téměř všechny využívané aplikace mají po instalaci vytvořený výchozí administrátorský účet, který slouží pro nastavení a správu prostředí. Přihlašovací údaje je možné najít ve veřejném manuálu nebo dokumentaci aplikace. Mnohé z nich se opakují mezi výrobci a jsou známy i mezi běžnými uživateli. Po nastavení aplikace nestačí jenom změnit heslo tohoto výchozího účtu, jelikož pro útočníka znalost uživatelského jména vyznačuje značnou výhodu, která mu dovoluje heslo účtu hádat hrubou silou. Je nezbytné vytvořit si nový administrátorský účet, ten výchozí odstranit a pokud to aplikace nabízí, nastavit ověření více faktory, jako je například Short Message Service (SMS) zprávou. [24]

1.3.2.8 Selhání softwaru a integrity dat

Tato kategorie chyb sdílí podobnost s zranitelnými a zastaralými komponentami, avšak liší se původem. V tomto případě se jedná o chyby v přídavných modulech a knihovnách z online zdrojů za přítomnosti automatických aktualizací z cílových úložišť a chybného nebo žádného ověření integrity dat. Závislost modulů na automatických aktualizacích může přinést riziko neoprávněného přístupu, manipulaci útočníkem, a ve výsledku kompromitaci systému. Následně mohou nastat situace, kdy odeslaná data modulu jsou serializována do nezabezpečené podoby, kterou dokáže útočník zobrazit, čímž může data modifikovat a vynutit tak závažnou deserializaci na cílovém bodě. [25]

1.3.2.9 Selhání protokolování a monitorování

Digitální svět je příliš dynamický na to, aby stačilo nainstalovat a nastavit webovou aplikaci a dále jí nespravovat. Pokud se již útočník dostal do systému, může být včasná reakce způsob, jak útočníka zastavit. Monitorování a protokolování nabízí možnost, jak zjistit, jakým způsobem se útočník do systému dostal a co v něm dělal. Tím pádem se naskytne možnost zabránit opakovanému narušení systému a nebo nahlášení zranitelnosti v aplikaci. Jestliže aplikace neprovádí zaznamenávání uživatelských akcí a není monitorována pro podezřelé aktivity, tak existuje riziko, že útočník již v systému je. [26]

1.3.2.10 Podvrh na straně serveru

Tato zranitelnost nastane v případě, kdy útočník donutí server k vytvoření požadavku ze strany serveru na nechtěné místo. Ohrožení se vztahuje na přístup k datům nebo akce v síti i aplikaci. Útočník může provést neautorizované činnosti, jelikož je zneužita důvěra systému webové aplikace s vnitřní komunikační sítí. V případě závady může útočník spustit libovolné příkazy na straně aplikace či vnitřní síť napojit na svůj vnější třetí bod. Pro ošetření je nezbytné v síti monitorovat a blokovat komunikaci, jejíž provádění není potřebné pro chod aplikace. [27]

Kapitola 2

Penetrační testování webových aplikací

Představme si situaci, kde provozujeme webovou aplikaci se stovkami aktivních uživatelů denně. S rostoucí aktivitou se zvyšuje popularita webu a přináší nám finanční odměny. Uživatelé na tento web nahrávají svá data a mají důvěru v jeho bezpečnost. Při situaci, kdy by aplikace byla napadena útočníkem, který by data ukradl či smazal, tak by se v ní snížila důvěra uživatelů, což by snížilo aktivitu, a to by snížilo finanční odměny. Útočník navíc může naší aplikaci použít k dalším útokům na aplikace jiné a zakrýt tím vlastní stopu. V obou případech navíc hrozí trestní stíhání.

A co když se na situaci podíváme z očí uživatele? Co v případě, že by šlo o bankovní aplikaci, která provádí finanční transakce a ukládá čísla kreditních karet? Chceme skutečně riskovat možné odcizení našich dat? A co v případě, že mi někdo vykradl bankovní účet? Ve všech případech jsme nenávratně poškozeni.

Jak se v tomto případě chová útočník? Záleží na typu útočníka, pokud je za cíleným útokem někdo zkušený, bude hledat slabinu a způsob, jak ji využít. Jestliže se mu podaří objevit zranitelnost a dostane se do systému aplikace, pokusí se povýšit svá privilegia na administrátorský účet. Dosažením administrátorských práv získá plnou kontrolu nad systémem. Co udělá dál je neznámé a závisí na jeho motivaci. Může si udělat zadní vrátka pro opakovaný přístup do systému, zamazat stopy, manipulovat s daty, využít výpočetní sílu stroje nebo stroj přidat do botnetu. Nutno podotknout, že se celou dobu jedná o nelegální chování.

V roli provozovatele chceme vědět, zda je naše webová aplikace odolná vůči takové operaci. Nabízí se situace vytvořit kopii prostředí aplikace a nechat v ní působit uživatele, který přijme roli útočníka a pokusí se různými způsoby zmocnit aplikace. Jestliže se mu to podaří, předá provozovateli zprávu popisující objevená slabá místa v aplikaci a jak je zabezpečit. Člověk v roli útočníka

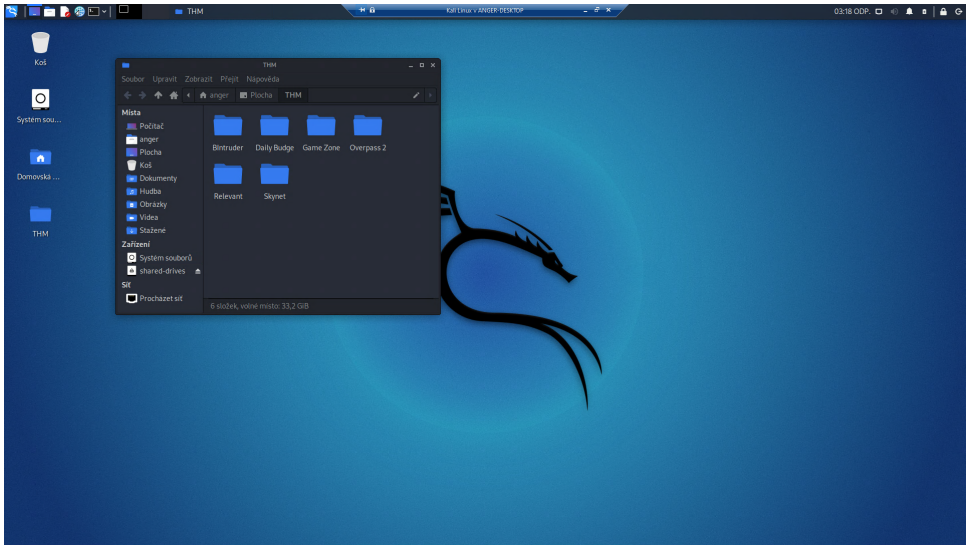
se nazývá bezpečnostní analytik a proces zkoumání slabin aplikace se nazývá *penetrační testování*, jehož cílem je posílení zabezpečení. [28]

Penetrační testování můžeme rozdělit do čtyř fází: průzkum, výčet, zneužití a hlášení. Bezpečnostní analytik dostane od zadavatele izolované prostředí, na které útočí, a časové období, během kterého vykoná útok. [29]

2.1 Operační systémy

2.1.1 Kali Linux

Existují specifické operační systémy, které jsou bezpečnostními analytiky s oblibou používané. Jedním příkladem je Kali. Jedná se o open source linux založený na Debianu se zaměřením na penetrační testování a bezpečnostní audit. Jeho síla spočívá v tom, že ve své instalaci obsahuje spousty různých nástrojů užívaných během testování, tedy vše, co bezpečnostní analytik potřebuje pro provedení testu, je již v celkové instalaci. Systém je volitelně rozšiřitelný, průběžně aktualizovaný a podporuje více jazyků. [30]

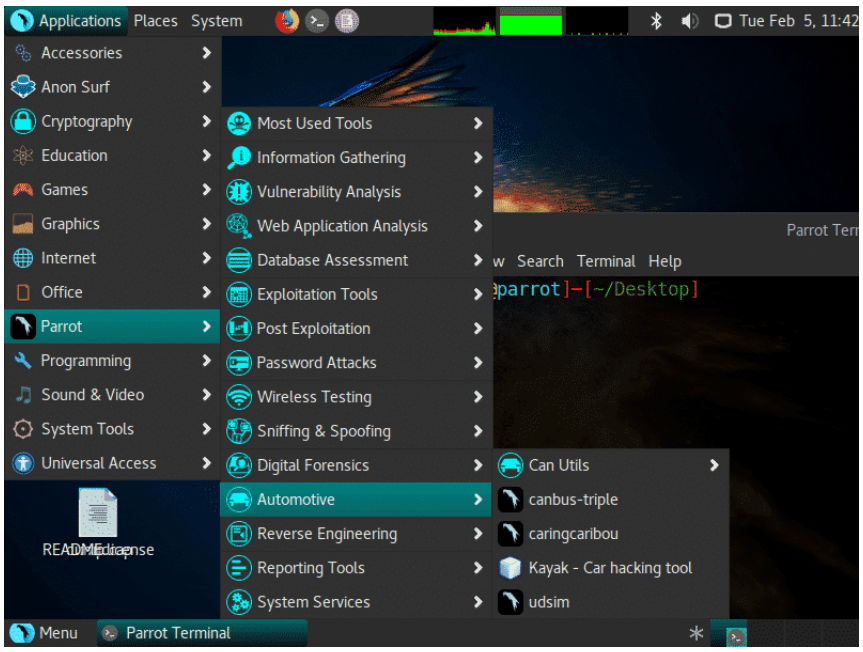


Obrázek 2.1: Prostředí Kali Linux v Hyper-V

2.1.2 Parrot Security OS

Alternativou Kali je open source nekomerční Parrot Security Operating System (OS). Založen na Debianu se soustředí na penetrační testování a bezpečnost, avšak přidává dimenzi zajištění anonymity na internetu s předinstalovanými nástroji The Onion Router (TOR) a AnonSurf, které Kali v základní

instalaci nemá. Systém disponuje též forenzním módem, ve kterém nepřipojuje diskové zařízení, čímž se v počítači skryje. [31]



Obrázek 2.2: Prostředí Parrot Security OS

2.2 Průzkum

Průzkum zahrnuje zkoumání perimetru před jakýmkoliv zásahem. Využívá se princip Open Source Intelligence (OSINT). V této fázi bezpečnostní analytik má k dispozici pouze identifikaci cíle na který útočí, a snaží se získat užitečné *veřejně dostupné* informace. Proto je tato fáze prakticky nerozeznatelná provozovatelem systému. Identifikátor může být adresa Internet Protocol (IP), web nebo název společnosti. Nástrojem této fáze je obyčejný webový prohlížeč. Zdrojem informací jsou sociální sítě, vyhledávače, WHOIS a Domain Name System (DNS) systémy. [32]

Manuální procházení webů poskytovatele, pokud nějaké jsou, je běžná praxe. Bezpečnostní analytik se chová jako standardní uživatel. Může se podívat na webovou stránku a proklikat jí, vytvořit si uživatelský účet nebo se podívat na zdrojový kód stránky. Informace mohou také nastínit starší verze webových stránek uložené v internetových archivech.

Fázi odmění i znalost a aplikace sociálního inženýrství. Jedná se o získávání informací od zaměstnanců a uživatelů webové aplikace. Bezpečnostní analytik zaútočí za pomoci falešné identity přímo na zaměstnance organizace s účelem vylákat z nich užitečné informace. Cílem jsou jména zaměstnanců, telefonní

čísla, hesla, emailové adresy, přístupové údaje, systémové informace, popis pracovního prostředí, využívané služby a software.

Na konci této fáze má analytik více informací o tom cíli, na který útočí.

2.3 Výčet

Tato fáze se soustředí na nalezení vektorů útoku na systém, které lze využít pro kompromitaci systému. Využívá informace nasbírané ve fázi průzkumu a přidává k nim nové, které se získají z aktivního spojení s cílem. Na dotazování cíle je možné použít nástroje, který proces provedou automaticky, a tím může bezpečnostní analytik využít čas na hledání unikátních zranitelností v prostředí, které automatizované nástroje neodhalí. Nástrojů existuje mnoho a jejich využití může záležet na situaci.

2.3.1 Nmap

Typickým prvním krokem v této fázi bývá výčet nástrojem Network Mapper (Nmap). Nástroj na vstupu vyžaduje IP adresu cíle a umožňuje zjistit, které porty jsou na cílovém stroji otevřené. Pokud je port otevřený, je na něm služba, která očekává připojení klientským zařízením pro výměnu dat. Otevřený port odpoví na žádost svým stavem. Nmap tedy na specifikované porty pošle žádosti a zaznamenává si k nim odpovědi od stroje. Způsob, jakým Nmap dotazuje porty je možné nastavit parametrem. [33]

```
Starting Nmap 7.91 ( https://nmap.org ) at 2022-05-09 00:12 CEST
Nmap scan report for px01.svethostingu.cz (83.167.244.201)
Host is up (0.015s latency).
Not shown: 997 filtered ports
PORT    STATE SERVICE
21/tcp  open  ftp
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 4.45 seconds
```

Obrázek 2.3: Výstup nástroje Nmap

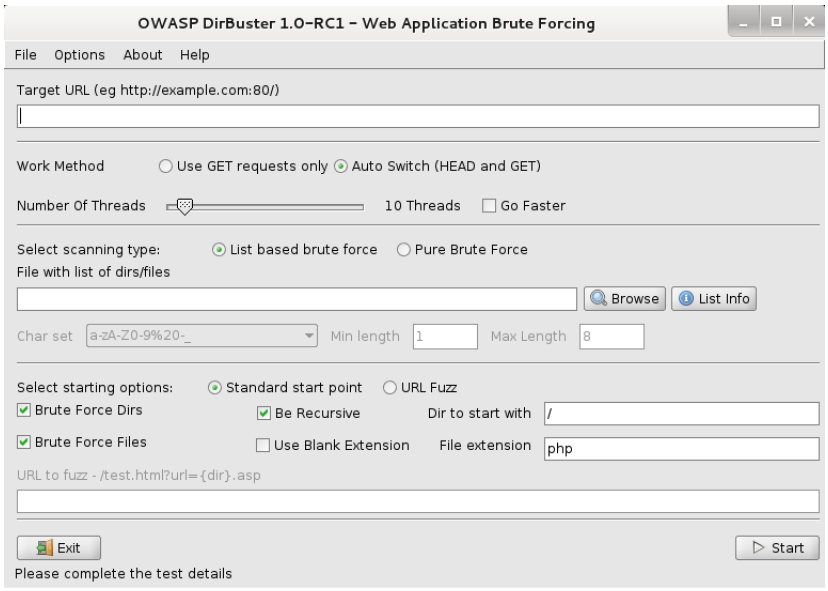
Nmap taktéž nabízí zjištění možného operačního systému na zařízení. Různé operační systémy odpovídají na žádosti jiným formátem zprávy a ve výchozím nastavení jsou na nich otevřené jiné porty. Nástroj ale upozorňuje, že tato funkcionalita může být nepřesná a nemusí být nutně možná.

Další dovednost, kterou nástroj disponuje, je spuštění výčtových skriptů na cíli. Nmap s sebou instaluje přibližně deset výchozích a nabízí možnost vytvoření dalších. Za zmínku stojí skript vuln, který cíl zkontroluje pro známé zranitelnosti.

Výstupem nástroje spuštěním s moduly výše je list s otevřenými porty a službami, které na nich běží, operační systém zařízení a objevené zranitelnosti.

2.3.2 Dirbuster

Tento nástroj lze použít, pokud na cíli jsou otevřené porty 80 nebo 443, tedy Hypertext Transfer Protocol (HTTP) a Hypertext Transfer Protocol Secure (HTTPS), což znamená, že na zařízení se nachází webová stránka. Webové stránky nemusí nutně mít všechny své cesty dosažitelné pouhým klikáním na úvodní stránce. Mohou existovat části a soubory, které se tváří jako skryté, ale znalostí cesty se k nim lze dostat. Příkladem je přihlašovací stránka pro správce webu. Dirbuster je nástroj vytvořený komunitou OWASP. [34]



Obrázek 2.4: Grafické prostředí Dirbuster

Dirbuster na vstupu vyžaduje adresu stránky a cestu ke slovníku, ve kterém jsou definované cesty, které otestuje. Validní cesty rozpozná podle odpovědi webu, které po skončení vypíše na výstup. Nástroj má též své grafické prostředí.

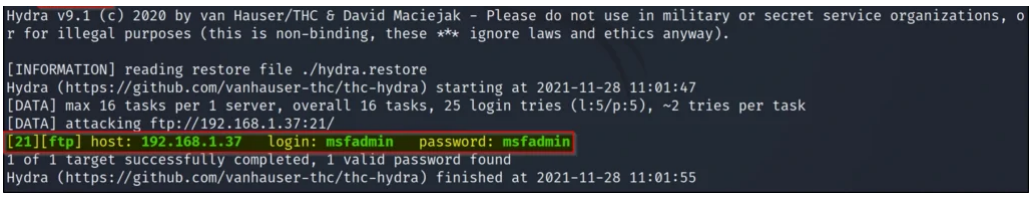
2.3.3 Nástroje na prolamování hesel

Na světě je spoustu varování ohledně síly hesel a jejich opakování. Přesto uživatelé často používají jedno heslo, které si lehce zapamatují. Pokud takový případ útočníkovi napomáhá, neboť po získání hesla se zmocní všech účtů, které uživatel vlastní. Na útok hrubou silou postačí útočníkovi znát uživatelské

jméno. Nástrojů na prolamování hesel existují spousty, avšak pro penetrační testování stačí jeden.

2.3.3.1 John the Ripper a Hashcat

John the Ripper vznikl jako program na testování síly hesla. Využití nalézá i jako nástroj na prolomení zaheslovaných souborů. Nabízí možnosti slovníkového útoku, hrubou silou generující posloupnost specifikovaných znaků a rozpoznávání hashů. Typicky na vstupu dostane formát hashe, slovník hesel a hash, který se pokusí rozluštit. Pokud se mu podaří najít heslo ve slovníku takové, jehož hash byl zadán na vstupu, vypíše ho na výstup. Hashcat je nástroj, který v základu funguje stejně jako John the Ripper. Hlavní rozdíl mezi nimi spočívá ve využití výpočetní síly, avšak oba nástroje jsou si v podstatě rovny. John the Ripper byl primárně vyvinut k maximálnímu využití síly Central Processing Unit (CPU) a Hashcat naopak Graphics Processing Unit (GPU). [35]



Obrázek 2.5: Výstup nástroje Hydra

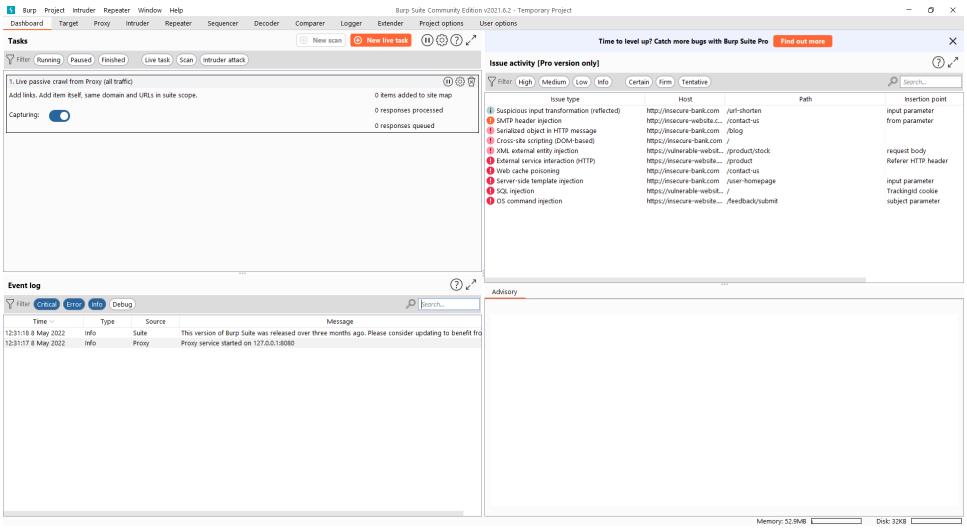
2.3.3.2 Hydra

V případě, že nemáme hash hesla či zaheslovaný soubor a chceme prolomit zabezpečení služby v kombinaci uživatelské jméno a heslo, můžeme využít jiné nástroje. Takový případ nabízí nástroj Hydra. Na vstupu dostane uživatelské jméno, slovník hesel a cíl, na který útočí. V případě úspěchu dostane Hydra odpověď a vypíše na výstup heslo.

2.3.4 BurpSuite

PortSwigger nabízí svůj nástroj BurpSuite pro testování webových aplikací. Zdarma k dispozici je komunitní verze a placená enterprise edice s více možnostmi. Nástroj má spoustu funkcionalit a i samotný v komunitní verzi je velice užitečný. BurpSuite umožňuje manuální procházení webu a ukládá navštívené cesty, k tomu obdobně jako nástroj Dirbuster má možnost automatizovaného procházení webu za pomoci slovníku a konečně strojové surfování webu překlikáváním. [36]

Nástroj taktéž přes nastavení proxy umí zachytit webové žádosti, které pak umožňuje upravit a odeslat nebo přesunout do jiných částí nástroje. V závis-



Obrázek 2.6: Grafické prostředí BurpSuite Community Edition

losti na potřebě tyto části umí žádosti replikovat a odeslat znovu, prolamovat hesla nebo odesílat payload.

Další dovedností nástroje je dekodér, ve kterém lze kódovat a dekódovat data. Dekódování lze provést manuálně či nechat BurpSuite rozpoznat kódování.

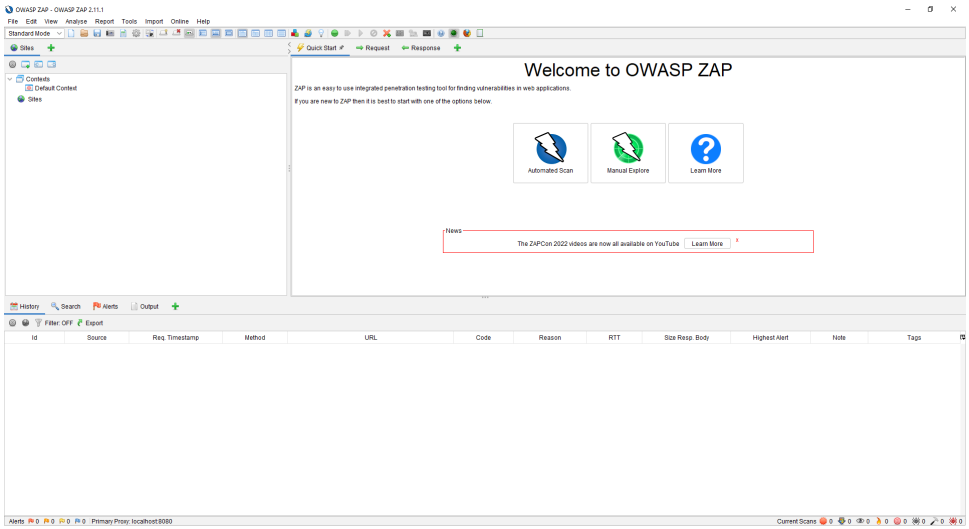
Nástroj má svoje rozšiřitelné API skrz svůj modul Burp Extender, který instaluje moduly z BApp Store.

2.3.5 OWASP ZAP

Alternativou k BurpSuite je bezplatný java nástroj OWASP ZAP, který má téměř shodnou funkcionalitu. ZAP nabízí možnost procházet web manuálně za pomoci proxy nebo automatický sken pavoukem. Zachycené webové žádosti prohledává, ukládá a dovoluje jejich modifikaci nebo opakované odesílání. Nástroj je volitelně rozšiřitelný a nabízí API ke spuštění v pravidelných testech.

2.3.5.1 Procházení

Nástroj zaznamenává všechna místa, kam se analytik podívá, avšak k tomu potřebuje nakonfigurovat. Při zapnutí vytvoří na zařízení v dostupném portu proxy server, na který lze připojit webový prohlížeč, aby mohl ZAP zachytit komunikaci a případně jí modifikovat. Další způsob je zabudovaný webový prohlížeč v nástroji, který ještě disponuje svým vlastním Heads Up Display (HUD) pro lepší pracování s nástrojem během penetračního testování. Poslední způsob pochází ze skenování nástroje, kdy si sám nástroj pamatuje cesty, které navštívil, a ukládá si je.



Obrázek 2.7: Grafické prostředí ZAP

2.3.5.2 Pasivní sken

Jedním ze skenů, co nástroj nabízí, je pasivní sken. Jedná se o sken, který ze zadané webové adresy pouze zkoumá odpovědi a hledá v nich zranitelnosti. Nijak nemodifikuje odeslaná data.

2.3.5.3 Aktivní sken

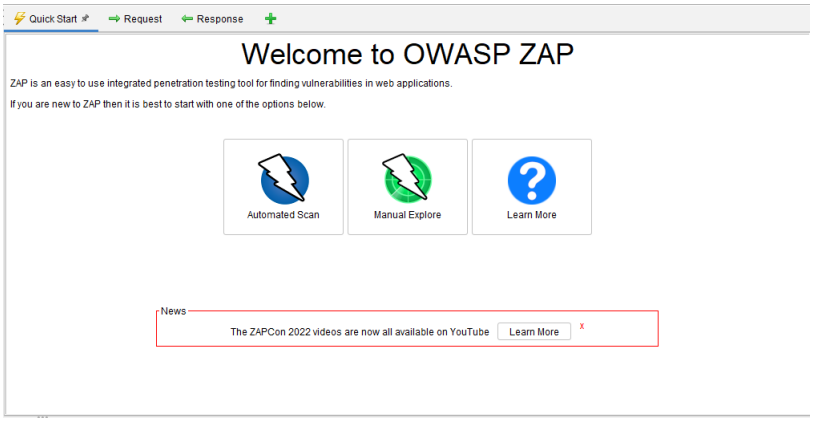
Aktivní sken je proces prohledávání zadané webové adresy, ve které dochází k modifikaci vstupních dat pro hledání závažnějších zranitelností. Tento sken dokáže též nahrávat i škodlivé skripty na stránku. Během aktivního skenu bezpečnostní analytik odhalí, že s cílem komunikuje a může být odhalen bezpečnostními procesy na straně aplikace. Proto by aktivní sken měl být spuštěn pouze se souhlasem provozovatele webové stránky.

2.3.5.4 Módy

V levém horním rohu ZAP nabízí možnost přepínání mezi módy. Na výběr je standardní, útočný, bezpečný a chráněný. Nástroj se spouští automaticky v módu standardním, který dovoluje uživateli prakticky provést cokoliv na jakékoliv stránce. Útočný mód aktivně skenuje webové stránky. Bezpečný vypíná škodlivé prvky během skenování a konečně chráněný mód skenování vymezí pouze na specifikované webové stránky.

2.3.5.5 Stránky

V levém horním okně nástroj ukládá navštívené stránky skrze proxy. Taktéž zde umišťuje cesty, které prošel během skenování. Okno lze nastavit tak, aby zobrazovalo pouze zvolené cesty.



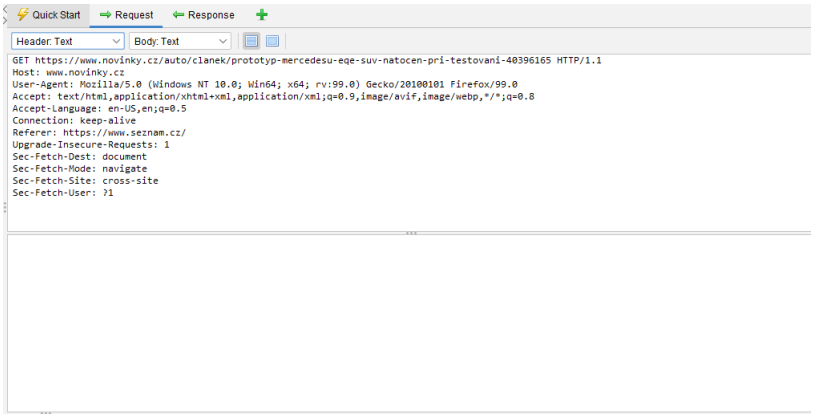
Obrázek 2.8: Pracovní okno ZAP s kartou rychlého startu

2.3.6 Pracovní okno

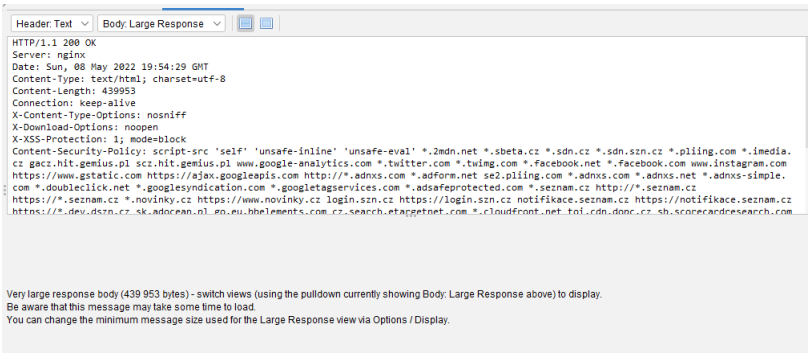
Vpravo nahoře se nachází okno pro práci s nástrojem. Na první kartě po startu je okno rychlého spuštění, ve kterém nástroj nabízí skeny, ve kterých nejprve dochází k automatizovanému prolezení celé webové stránky a potom se spustí aktivní sken. Vedle skenu se nachází tlačítko pro spuštění zabudovaného prohlížeče nástroje, který umožňuje manuální procházení internetu. Častěji se ale využívají dvě další karty, žádost a odpověď webové aplikace. V obou případech se okno ještě rozdělí na dvě. V horním okně můžeme vidět hlavičku s cookies a dole parametry v případě žádosti. Pro odpověď nahoře vidíme hlavičku a dole tělo.

2.3.6.1 Dolní okno

Dolní okno ukazuje komunikaci, tedy převážně GET a POST žádosti, které se posílaly. Dále ještě zde jsou karty výstup a upozornění, které je zde nejdůležitější a nejpoužívanější. V této kartě můžeme najít všechny zranitelnosti, které nástroj našel na webové stránce za pomoci skenování. Například v této bakalářské práci je možné najít tzv. anti-CSRF token. Když si spustíme sken na stránce, která tyto parametry nemá, ZAP to rozpozná a v upozorněních vypíše, že na zadané webové stránce token nenašel. V tom případě předpokládá, že je stránka zranitelná na CSRF. V upozornění můžeme také nalézt popis zranitelnosti a její zabezpečení, a také místo, ve kterém byla zranitelnost objevena.



Obrázek 2.9: Pracovní okno ZAP s kartou žádosti



Obrázek 2.10: Pracovní okno ZAP s kartou odpovědi

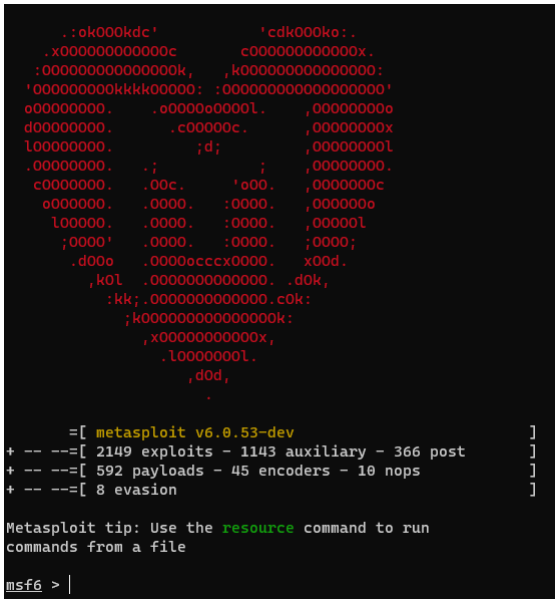
Obrázek 2.11: Upozornění zranitelností v ZAP

2.4 Zneužití

Analytik se dostal do cílového systému a nyní je cílem získat plnou kontrolu nad prostředím, jinak řečeno povýšení privilegií. K tomu opět může využít spousty nástrojů, ale existuje i manuální způsob výpisu systémového nastavení a pokus o jeho zneužití. Pokud se analytikovi podařilo povýšit se na správce systému, může se porozhlédnout po vnitřní síti skrytou před internetem a identifikovat nové cíle, čímž by pro ně začal nový cyklus penetračního testování.

2.4.1 PEAS

Jednu alternativu nabízí skripty povýšení administrátorských prav pod názvem Privilege Escalation Awesome Script (PEAS). Stačí nahrát soubor na cílové zařízení a spustit ho. Skript se sám pokusí identifikovat způsob, jak povýšit uživatelská práva. Principiálně hledá chybné nastavení operačního systému. Na stránce Github existují dva skripty: WinPEAS pro výčet operačního systému Windows a LinPEAS pro platformu Linux. [37]



Obrázek 2.12: Metasploit msfconsole

2.4.2 Metasploit

Podobně jako ZAP a BurpSuite pro webové aplikace existuje i velmi silný open source nástroj Metasploit. Nabízí možnost vyhledat skripty v online databázích, stáhnout, nakonfigurovat a spustit je. Před spuštěním skriptu umožňuje

analytikovi blíže specifikovat útok. Nástroj má svoje vlastní grafické prostředí zlehčující celkovou práci s ním a je celkově velice snadný na užívání. Metasploit má ve svém základu zabudovaný Meterpreter, což je vzdálená dynamická příkazová řádka na cílovém stroji schopna komunikovat s Metasploitem na útočníkově zařízení. Další nástroj používaný v tandemu pro vytvoření vzdáleného připojení je msfvenom, který pro cílový operační systém vygeneruje vzdálený bod, na který se Metasploit dokáže připojit. [38]

2.5 Hlášení

Poslední část penetračního testování je podání zprávy o výsledku. V této zprávě bývá detailně popsán celý proces od průzkumu po zmocnění se systému, všechna nalezená kritická data, objevené zranitelnosti a důkaz zranitelnosti. Vývojářský tým by po přečtení zprávy měl mít znalost toho, kde je v systému chyba, jak jí lze využít a měl by pracovat na zabezpečení.

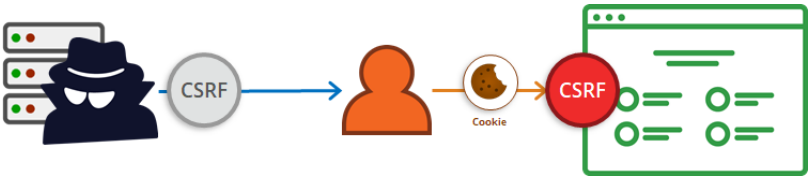
Kapitola 3

Zranitelnost Cross Site Request Forgery

Cross Site Request Forgery (CSRF) (česky podvržení křížového požadavku na webu) je útok, který donutí uživatele provést nežádoucí akci. Útočník identifikuje místo na webové stránce, která ověřenému uživateli provede určitý typ akce, například změnu hesla nebo platbu na účet, a zkonstruuje skrytý webový dotaz, který odešle oběti. Typický příklad je odeslání webové stránky s obrázkem, jehož odkaz je prakticky neplatný, jelikož na jeho konci se neskrývá obrázek. Webový prohlížeč se na jeho adresu podívá a bez vědomí oběti odešle důvěryhodná data včetně autentizace. Webová aplikace zpracovávající zranitelný dotaz nemá žádný způsob, jak zjistit, zda uživatel skutečně chce provést akci nebo zda je neúmyslná. Útočník ve své podstatě nemusí nutně vědět, zda již útok proběhl, a zda vůbec úspěšně. Využití zranitelnost nalézá v aplikacích, kde je možné vytvořit si vlastní uživatelský účet, tedy vyžaduje autentizaci.

Přestože se zranitelnost neumístila v seznamu OWASP Top 10, jedná se o zranitelnost, která je stále aktivní. Poslední významný útok touto slabinou se stal v roce 2020 na platformě TikTok. Zranitelnost se také objevila roku 2014 v bezpečnostním produktu McAfee Network Security Manager, kde dovolila útočníkovi modifikovat uživatelské účty ostatních. V roce 2008 byla zranitelná platforma YouTube. Zranitelné byly téměř všechny uživatelské akce, včetně upravování videí a odesílání zpráv. Ve stejném roce se závada objevila i ve webové stránce mezinárodní bankovní společnosti ING, ve které umožnila útočníkovi přesun finančních prostředků. Přestože stránka ověřovala uživatele přes protokol SSL, neměla žádnou ochranu proti CSRF. [39]

Pokud oběť otevře ve svém prohlížeči útočníkův skrytý webový dotaz, odešle její prohlížeč na stránku i obsah cookies. V těch bývají uložené uživatelské údaje napomáhající s navigací na stránce včetně autentizace a otevřené relace. V tomto případě je zneužita důvěra mezi prohlížečem oběti a webovou aplikací, kdy se komunikace tváří jako validní a zamýšlená.



Obrázek 3.1: Vizualizace CSRF

```
1 <img src="http://stranka.cz/platba?ucet=jan&castka=10000" />
```

V praktických případech se často používá zneužití žádostí GET. Webové aplikace by v žádných případech neměly měnit svůj stav na základě takového požadavku. Požadavek GET by měl sloužit pouze k odeslání dat do uživatelského prohlížeče nebo aplikace, se kterou web umí komunikovat přes vhodné API. Avšak určité stránky stále používají žádosti GET k provedení určitých změn. Pokud se v prohlížeči oběti objeví obrázek nebo iframe s odkazem na cestu GET požadavku, tak se akce provede s autentizací oběti.

Ve skutečném prostředí útočník použije jiné žádosti než GET. Zranitelný nutně nemusí být pouze POST, avšak jedná se o nejčastěji používaný způsob vynucení určité akce ve webové aplikaci. Provést CSRF přes žádost typu POST už je složitější, avšak stále možné. V tomto případě útočník může zkonstruovat webové stránky, které ve svém zdrojovém kódu vytvářejí POST žádosti na cíl, nebo použít JavaScript, který běží ve webových prohlížečích.

Tato zranitelnost poškozuje uživatele webových aplikací. V případě zranitelnosti dokáže útočník oběť poškodit způsoby, které závisí na typu a obsahu aplikace. Obvykle jsou ohroženy uživatelské účty, neboť díky chybě může útočník ukrást data nebo samotný účet bez znalosti původního hesla. V případě e-shopů a bankovních služeb závada představuje ohrožení finančních prostředků.

Tato zranitelnost je specifická tím, že se může na stránce objevit i v případě naprosto jiné zranitelnosti, jako je například Cross Site Scripting (XSS). Pokud se ve webové aplikaci objeví tato zranitelnost, všechny pokusy o zabezpečení CSRF mohou být invalidovány. [40]

3.1 Ochrana

Na zabezpečení existuje mnoho způsobů. Žádný z nich však nezaručuje kompletní ochranu, ale kombinováním se jí můžeme přiblížit. Například v hlavičce webových žádostí zkontrolovat, zda požadavek vygenerovala stejná stránka, která na něj odpovídá. Tímto však docílíme pouze přidáním extra dimenze zabezpečení, které útočník dokáže eventuálně obejít. Nejpopulárnější metodou ochrany jsou anti-CSRF tokeny.

3.1.1 Origin/Referer

Jedním způsobem ochrany je přítomnost parametrů origin a referer v hlavičkách žádostí. Tyto parametry musí mít v sobě adresu stránky, ze které jsou odeslány. Pokud na webovou stránku přijde žádost, ve které je cizí webová stránka nebo parametry kompletně chybí, měla by zpracování úplně zamítnout, zastavit a neprovést žádnou manipulaci s daty. Problém ale nastane v případě, kdy aplikace skutečně žádá stránky z jiného zdroje.

3.1.2 Cross Site Cookies

Toto zabezpečení funguje pouze v prohlížečích, které implementují možnost funkce same-site cookie. Nastavení efektivně zabraňuje zranitelnosti CSRF. V případě navštívení třetím bodem zablokuje webovému prohlížeči odeslání cookies identifikující sezení uživatele. Cookie nedokáže samo chránit před útokem, proto se používá v kombinaci s tokeny. [41]

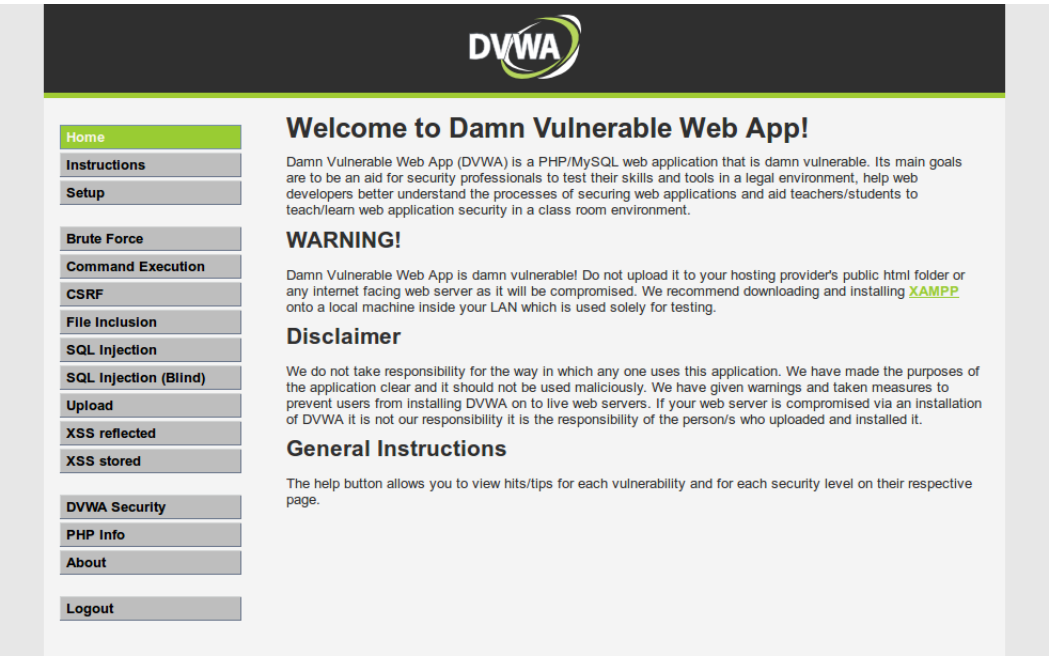
3.1.3 Anti-CSRF token

Token je unikátní parametr vygenerovaný webovou aplikací a vložen uživateli do formulářů jako skrytý pro ověření při odeslání, zda nedošlo k replikaci webové žádosti. Může to být v podstatě cokoliv, například číslo, které webová stránka pouze inkrementuje při každém vygenerování nové GET žádosti. V takovém případě samozřejmě není zabezpečí dostatečné a útočník, který si všimne vzoru zvyšování, snadno token prolomí. Požadavkem je, aby se při každé nové žádosti měnil, jestliže je neměnný a pevně vložen do zdrojového kódu stránky, opět nepřidává žádnou novou úroveň bezpečnosti. Myšlenkou je mít náhodný generátor dat, který odesílá data do hashovací funkce. Tím pádem je zajištěna nepředvídatelnost tokenu, který z hashovací funkce je uložen do paměti webové aplikace pro ověření, zda již byl použit, a zároveň odeslán do webového prohlížeče oběti v formuláři jako skrytý parametr, aby byl využit ve webové žádosti, kterou v budoucnu vykoná. Pokud tedy token, který přijde v žádosti je shodný s tím, který je v paměti stroje, můžeme předpokládat, že nedošlo k útoku. V opačném případě byl použit duplikát, upraven nebo smazán a žádost zahodíme.

3.1.4 Dvojite odeslání cookies

Alternativou k tokenům je použití techniky dvojitého odeslání cookies. Jedná se o bezstavovou techniku, která sdílí podobnost s tokeny. Aplikace během komunikace s webovým prohlížečem odešle náhodně vygenerovaný parametr v cookies a při práci s formuláři vyžaduje jeho hodnotu. V případě zvýšení bezpečnosti lze parametr zašifrovat. [42]

Bližší přiblížení zranitelnosti a jejího zabezpečení v praxi nabízí níže popsaná Damn Vulnerable Web Application (DVWA).



Obrázek 3.2: Prostředí DVWA

3.2 Damn Vulnerable Web Application

Damn Vulnerable Web Application (DVWA) je zranitelná webová aplikace vytvořená a spravovaná organizací OWASP určená pro cvičné praktikování penetračního testování legálním způsobem. Jelikož vyžaduje vlastní instalaci, tak aplikace není nikde hostována, je ale k dispozici ke stažení ze stránky Github. Po vytvoření uživatelského účtu nabízí řadu zranitelností a umožňuje nastavení jejich obtížnosti - lehké, střední, těžké a nemožné. Aplikace u každé zranitelnosti nabízí možnost nahlédnout do zdrojového kódu k identifikaci slabého místa. [43]

V aplikaci je pro tuto práci důležitá sekce CSRF. Cílem útoku je změna uživatelského hesla. Obětí i útočníkem je jedna a totéž osoba.

3.2.0.1 Lehká obtížnost

V lehkém případě se jedná o základní případ zranitelnosti. Jako útočník si najdeme stránku s formulářem měnící uživatelské heslo a podíváme se, jaké parametry odesílá. Cestu této stránky pak zkopírujeme a jelikož se jedná o GET request, stačí data měnící heslo připojit k cestě. V tomto momentě máme připravený dotaz, jehož akci chceme vynutit u oběti. Zde existuje více způsobů, avšak tradiční je cestu vložit jako zdroj obrázku v HTML a následně oběť donutit zobrazit si vytvořený kód ve webovém prohlížeči. Jakmile se tak stane, bez vědomí oběti si změní heslo. Útočník sice vyžaduje znalost uživatelského

jména, ale pro tento případ ho již víme, jelikož je to náš vlastní účet. V tomto případě se stačí odhlásit a přihlásit pod novým heslem a úkol je hotov.

3.2.0.2 Střední obtížnost

Střední obtížnost už si trvá na řetězení slabin. Předchozí metoda je již zabezpečná a tedy nefunkční. Stačí ale menší úprava, a bude zase funkční. Server si musí myslet, že jde o skutečnost žádost. Musíme provést manuální editaci žádosti v aplikaci, a to přidáním odkazu (ang. referer) do hlavičky. Tuto editaci lze provést v příkazové řádce, v nástroji BurpSuite při zapnuté funkci Intercepter nebo v ZAP. Odesláním tohoto požadavku docílíme úspěchu, avšak my jako útočníci bychom chtěli lehčí řešení pro naší oběť než modifikaci žádosti. Tady již je nutné využít řetězení zranitelnosti v kategorii reflected XSS.

3.2.0.3 Těžká obtížnost

Obtížnost těžká přidává nový, dynamický prvek, který běžná uživatel nevidí, avšak s ním pracuje. Jedná se o náhodně generovaný anti-CSRF token, který se vkládá do žádosti a aplikace tak podle něj pozná, zda byla žádost zreplikovaná, nebo se jedná o originální. Typicky nesprávný přístup je přidat do původní žádosti tento parametr s momentální hodnotou.

Token má svojí délku a sílu. Můžeme zkoumat jeho různost a pokusit se předvídat jeho další hodnotu. Tím pádem bychom vlastně útočili na náhodný generátor ve webové aplikaci. Přijatelná možnost, avšak pro potřeby DVWA moc složité, a v určitých případech i příliš časově a výpočetně náročné.

V tomto případě tedy pro úspěšný útok musíme nejprve nechat aplikaci vygenerovat nový token. Ten pak nakopírujeme do žádosti a donutíme oběť provést akci. Jelikož token bude naprosto nový, aplikace bude důvěřovat tomu, že se jedná o jednorázovou zamýšlenou žádost, a provede změnu hesla. V případě DVWA je třeba opět řetězit zranitelnosti a to slabinu v nahrávání souborů na aplikaci. Jde o složitější provedení, avšak postup je téměř stejný jako v předchozím případě, jenom je třeba získat token a vložit ho do žádosti.

3.2.0.4 Nemožná obtížnost

Nápad, který aplikace propaguje jako absolutní zabezpečení stránky před CSRF je v podstatě další přidání dat v žádosti. Jelikož cílem tohoto útoku je neznámé heslo změnit na známé, můžeme při změně hesla vlastně původní heslo vyžadovat, a pak bez jeho znalosti nelze heslo změnit. Tedy webová žádost v tomto případě požaduje původní heslo. Samozřejmě můžeme pomocí různých technik jako sociální inženýrství nebo phishing heslo zjistit a pak ho odeslat, ale k čemu nám je měnit známé heslo na známe, když už ho vlastně známe.

DVWA tedy zkráceně říká: CSRF je zranitelnost, kdy je útočník schopný donutit oběť vykonat nežádoucí akci. Potřebuje k tomu znalost webové apli-

kace, na kterou útočí, a určitý typ kontaktu s obětí. Faktor ochrany na webu je důvěřovat a ověřovat přicházející žádosti, že je skutečně odeslaná uživatelem s cílovou záminkou. K tomu slouží přidání specifických parametrů, které toto chování mají zajistit. Maximální zajištění lze dosáhnout, když pro provedení akce je třeba znalost vstupu, který je pro každého uživatele rozdílný, a pro útočníka klíčový znát. Zranitelnost není izolovaná a může souviset i s jinou zranitelností na webu.

Kapitola 4

Rozšíření

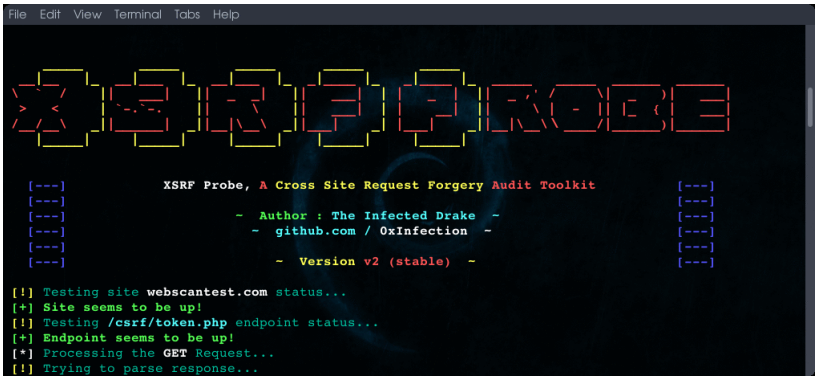
Pro implementaci CSRF rozšíření do ZAP je vhodné podívat se, zda již existuje nějaký způsob, jak zranitelnost testovat. Případně jaká řešení existují.

Validní, avšak pracný a časově náročný způsob je manuální psaní žádostí s payloady. Během penetračního testování se stejně musí manuálně otestovat, zda se ve webové aplikaci tato zranitelnost skutečně nachází. Jelikož CSRF má určitý vzor, tak existují nástroje, které hledají potenciální slabé místo a hlásí ho.

4.1 Analýza existujících nástrojů

4.1.1 xsrfprobe

Populární nástroj *xsrfprobe* nabízí možnost komplexního automatizovaného testování v širokém rozsahu. Je napsaný v jazyce Python a zcela dostupný na stránce Github. Je velice detailně dokumentovaný. Na vstupu vyžaduje cíl, který skenuje na zranitelnost, případně cookies a sessions z webové stránky. Nástroj v celé své fázi spouští celkem dvanáct testů. V prvních dvou kontroluje, zda stránka reaguje na modifikaci hlaviček žádostí. V dalších čtyřech hledá a blíže zkoumá anti-CSRF token. Testuje jeho sílu, náhodnost, kódování a zda aplikace reaguje na jeho přítomnost. Následně se podívá na hodnoty v cookies z důvodu manipulace s nimi, jejich validaci a reakci na ně. Pak nástroj proleze celou webovou stránku, najde všechny formuláře, které si uloží a následně otestuje, zda mění token. Během této fáze se nástroj tváří jako tři uživatelé a ukládá si odpovědi webu, které pak kontroluje na rozdíl délky pomocí algoritmu Ratcliff-Obershelp. V následujícím testu nástroj odesílá žádosti, ve kterých manipuluje s tokeny, jako třeba smazání nebo zaměnění jednoho charakteru. V předposledním testu se použije buď manuální nástroj k ověření existence chyby CSRF ve formulářích nebo automatický s výchozími hodnotami. Finální test se zaměřuje na zkoumání Levenshteinovi vzdálenosti v nasbíraných tokenech a rozeznání vzorce použitého při jejich generování.



Obrázek 4.1: Terminálové prostředí xsrfprobe

4.1.2 OWASP ZAP

OWASP ZAP nabízí na testování manuální modifikaci odeslaných HTML requestů. Můžeme upravovat hlavičku žádosti nebo do ní vkládat payload. Nástroj automaticky detekuje a hlásí přítomnost anti-CSRF tokenu, které pak umožňuje generovat webovou stránkou a odesílat. ZAP nabízí desítky rozšíření. Avšak žádné z nich se nesoustředí primárně na zranitelnost CSRF. Nástroj také vyhledává cookies na přítomnost same-site hodnoty.

4.1.3 BurpSuite

BurpSuite nabízí velice podobnou metodu testování. Pomocí funkce intercept můžeme chytit žádosti, upravit je a sledovat jejich odpovědi. V alternativním případě je možnost request odeslat do funkce Intruder a vložit množinu payloadů, které chceme otestovat. Komerční verze BurpSuite umí webovou stránku testovat na zranitelnost CSRF

4.2 Návrh

Návrh vychází z dokumentace OWASP ZAP zaproxy, kde je popsáno nastavení nástroje pro vývojářské účely a vytváření nového modulu. Zdrojový kód modulů i nástroje je dostupný na stránce Github.

Nástroj již během skenování prochází web a ukládá si odeslané webové žádosti a jejich odpovědi.

Testovat každou stránku přímo na zranitelnost CSRF automaticky je obtížné. Aplikace mají různé cesty, používají různé pojmenování parametrů i v cizích jazycích. Sice existují payloady na otestování, ale jejich základ je třeba editovat v závislosti na webové stránce. Bezpečnostní analytik však i při objevení zranitelnosti automatickým nástrojem musí manuálně zkontrolovat jeho existenci během penetračního testu. Nástroj již reaguje na přítomnost anti-CSRF

tokenu. Nabízí se tedy možnosti reakce na jeho manipulaci, přítomnost nebo zkoumání jeho náhodnosti a pokusu najít v jeho generování vzor a tím pádem se naučit ho vytvořit. Druhý způsob je ověření dat hlavičky žádostí a reakce na jejich manipulaci, modifikaci a přítomnost. Třetí možnost je sledovat řetězení zranitelností, které umožňují CSRF útok i v případě, že je webová stránka zabezpečená tokenem a ověřením hlavičky. Pro tuto práci byla zvolena druhá možnost, a to manipulace s daty hlavičky žádosti.

4.2.1 Rozšíření pasivního skenování

V nástroji ZAP je možnost ukládat navštívené stránky v aplikaci, které sleduje koncové body v pavoucích, při skenování, v zabudovaném prohlížeči i při použití proxy serveru. K těmto cestám jsou navázané webové žádosti. V případě pasivního skenování dochází nástrojem ke čtení žádostí, které následně procházejí kontrolními pravidly.

V pasivních pravidlech došlo k přidání nových. Pro webové žádosti se vyhledávají v hlavičkách parametry origin a referer. Jestliže nejsou spolu přítomné, nástroj vypíše upozornění se středním rizikem. V případě přítomnosti se porovná, zda obě žádosti přichází od stejného zdroje. Pokud není zdroj shodný, vypíše se informativní upozornění ke zkontrolování podoby parametrů.

4.2.2 Rozšíření aktivního skenování

V případě aktivních pravidel došlo k podobnému zásahu jako v pasivních pravidlech. Přítomnost hodnot origin a referer kontrolují pasivní pravidla, tudíž aktivní sken manipuluje s obsahem parametrů. V první řadě pravidla kontrolují, zda jsou hodnoty přítomny v původní webové žádosti, která byla zpracována během skenování nebo procházení stránky. Nejsou-li přítomny, nespouští se žádná další kontrola. V opačném případě je odeslán požadavek, ve kterém obsah parametru chybí, a sleduje se reakce aplikace. Následně je proces zopakován s tím, že do parametru je vložena falešná hodnota. Jestliže webová stránka ani v jednom případě nereaguje na změnu parametrů, je vyhodnocena jako středně zranitelná.

4.3 Implementace

V implementaci došlo k zásahu do modulů ascanrules a pscanrules. Oba tyto moduly jsou bodem, kam ZAP posílá webové žádosti, které skrze něj prochází.

Pro případ pscanrules byly vytvořené nové třídy rozšiřující třídu *PassivePluginScanner*, v nichž dochází ke kontrole hlaviček webových žádostí. Do nástroje přibylo nové pravidlo, které otevře hlavičky webových žádosti a zkontroluje, zda se v nich nachází parametr origin nebo referer. Absence obou

parametrů je vyhodnocena jako středně závažná, tudíž pravidlo vytvoří upozornění.

```
1  public void scanHttpResponseReceive(HttpMessage msg, int id,
       Source source) {
2    List<String> origin = msg.getRequestHeader().getHeaderValues(
       ORIGIN);
3    List<String> referer = msg.getRequestHeader().getHeaderValues(
       HttpHeader.REFERER);
4
5    boolean missingOrigin = origin.isEmpty();
6    boolean refererMissing = referer.isEmpty();
7
8    // if both are missing, the request could be vulnerable
9    if (missingOrigin && refererMissing) {
10     this.raiseAlert(msg, id);
11   }
12 }
```

Zdrojový kód 4.1: Kontrola přítomnosti origin/referer v hlavičce

Metoda newAlert nabízí možnost vytvoření upozornění. Převezme parametry specifikující zranitelnost a v momentě zavolání se automaticky zobrazí v dolním okně. Metoda raiseAlert byla vytvořena za účelem vytvoření nového upozornění a je volána ve chvíli, kdy bylo detekováno slabé místo. Ve svých metodách umožňuje nastavení konkrétního popisu události, včetně webové žádosti, ve které došlo k rozpoznání.

```
1  private void raiseAlert(HttpMessage msg, int id) {
2  // creates new alert
3
4  newAlert()
5    .setName(getName())
6    .setRisk(getRisk())
7    .setConfidence(Alert.CONFIDENCE_HIGH)
8    .setDescription(getDescription())
9    .setParam(ORIGIN + ", "  + HttpHeader.REFERER)
10   .setSolution(getSolution())
11   .setReference(getReference())
12   .setCweId(getCweId())
13   .setWascId(getWascId())
14   .raise();
15 }
```

Zdrojový kód 4.2: Metoda vytvářející upozornění

Jestliže v hlavičce žádosti jsou nalezeny parametry origin nebo referer, projdou ověřením, zda odkazují na správnou webovou adresu. Z hlavičky je extrahován obsah Uniform Resource Identifier (URI) a porovnán s obsahem v parametrech. Nejedná-li se o shodu, je vytvořeno informativní upozornění s textem motivujícím k manuálnímu ověření, zda hodnoty jsou skutečně rozdílné.

```
1    // check if scope is equal
```

```
2   URI refURI;
3   String refURL = referer.get(0);
4   try {
5     refURI = new URI(refURL, false);
6   } catch (URIException | NullPointerException e) {
7     return;
8   }
9   try {
10    if (!baseURI.getHost().equals(refURI.getHost())) {
11      this.raiseAlert(msg, id, "Base URI: " + baseURI.getHost() + "\
            nReferer URI: " + refURI.getHost());
12    }
13  } catch (URIException e) {
14    return;
15  }
```

Zdrojový kód 4.3: Ověření hodnoty referer

Během skenování aktivními pravidly jsou celkem vytvořeny tři nové požadavky. První slouží ke kontrole, zda je parametr origin nebo referer přítomen a je z něj vyjmut návratový kód stránky. Nedochází v něm k žádné editaci, pouze v případě, že parametr chybí, je proces ukončen.

```
1     HttpMessage msg = sourceMsg.cloneRequest();
2
3   try {
4     sendAndReceive(msg, false);
5   } catch (URIException e) {
6     log.debug("Failed to send HTTP message, cause: {}", e.getMessage
          ());
7     return;
8   }
9
10  if (isStop()) {
11    return;
12  }
13
14  List<String> origin = msg.getRequestHeader().getHeaderValues(
        ORIGIN);
15
16  // origin is not present
17  if(origin.isEmpty()) {
18    // raiseAlert(msg, param, "Origin was not found during active
          testing."); Do not raise an alert, passive scanning already
          did that
19    return; // application doesn't react to origin tampering
20  }
21  verifyCode = msg.getResponseHeader().getStatusCode();
```

Zdrojový kód 4.4: První fáze aktivního skenu parametru origin

V druhé fázi je vytvořena nová žádost, do jejíž hlavičky je zkopírován obsah z původní zprávy s tím rozdílem, že je testovaný parametr odstraněn. Žádost je následně odeslána a sleduje se odpověď. Jestliže webová stránka nereaguje

na odstranění parametru, dojde k vytvoření upozornění se střední závažností a zprávou informující o důvodu hlášení. Třetí fáze je téměř identická jako druhá s tím rozdílem, že se do testovaného parametru vloží obsah falešné webové stránky.

```
1            // detect if website reacts to fake website in origin
2       HttpMessage msg3 = sourceMsg.cloneRequest();
3       msg3.getRequestHeader().setHeader(ORIGIN, FAKE_WEBSITE);
4
5       try {
6         sendAndReceive(msg3, false);
7       } catch (URIException e) {
8         log.debug("Failed to send HTTP message, cause: {}", e.
              getMessage());
9         return;
10      }
11
12      if (isStop()) {
13        return;
14      }
15
16      retCode = msg3.getResponseHeader().getStatusCode();
17
18      if(verifyCode == retCode) {
19        raiseAlert(sourceMsg, param, "Origin was set to fake website
              , website didn't react to change.");
20      }
```

Zdrojový kód 4.5: Ověření reakce webové aplikace na falešný obsah parametru origin

Všechny tři fáze jsou spuštěny v metodě scan, která je součástí třídy AbstractAppParamPlugin. Metoda se spouští pro každou nalezenou webovou žádost a zahrnuje v sobě kopii původní zprávy s odpovědí. Během každého odeslání nové žádosti se kontroluje, zda v ZAPu nebyla provedena akce k zastavení průběhu testů. Po odeslání žádostí se metoda ukončí, nedochází zde k dalšímu ověřování nebo manipulaci s odpověďmi.

Kapitola 5

Testování

Ve fázi testování byly vybrány dvě webové aplikace pro porovnání výsledků. V prvním případě se jednalo o webovou aplikaci, ve které byla zranitelnost CSRF přítomná. V druhém případě nebyly známé žádné podrobnosti kromě neověřitelné informace od provozovatele, že je kompletně zabezpečená. V pravidlech došlo k izolaci tak, že se během testování spouštěla pouze nová aktivní a pasivní pravidla skenování.

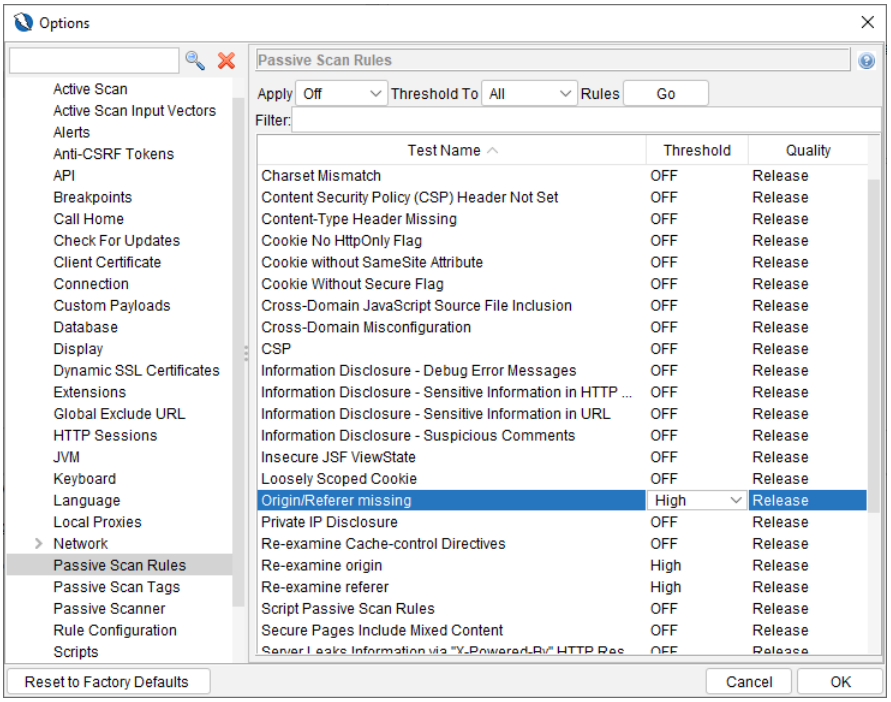
V nastavení pasivního skenování byla aktivována pouze pravidla Origin/Referer missing, Re-examine origin a Re-examine referer. V aktivním skenování se aplikovala nově vytvořená skenovací zásada, v níž jsou aktivní pouze pravidla Origin a Referer Tampering.

5.1 Cvičné prostředí

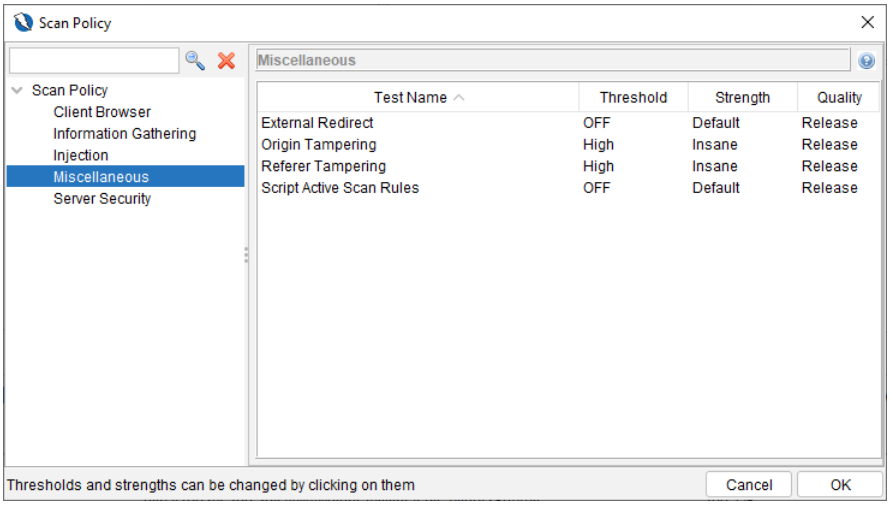
Spuštění pasivního skenování ve cvičném prostředí hlásilo detekování střední závažnosti. Žádosti v sobě neměli přítomné parametry origin/referer a nedocházelo k jejich validaci. Aktivním skenováním bylo odhaleno, že určité koncové body nereagují na odstranění parametru referer. Jelikož parametr origin není ve webové aplikaci vůbec přítomen, nedošlo k žádné modifikaci a testování reakce na jeho změnu. Z důvodu znalosti zranitelnosti ve webové aplikaci a hlášení výsledku aktivního skenování lze potvrdit přítomnost zranitelnosti CSRF a usoudit, že jí nástroj nyní detekuje i z nového úhlu.

5.2 Skutečné prostředí

Po dohodě s provozovatelem webové aplikace byl spuštěn penetrační test pouze s novými pravidly. Z důvodů mlčenlivosti a bezpečnosti nelze v této práci odhalit webovou adresu cíle. Provozovatel během dohody tvrdil, že aplikace je kompletně zabezpečená. V tomto případě se jedná o black box test, během kterého není žádná znalost o cíli.

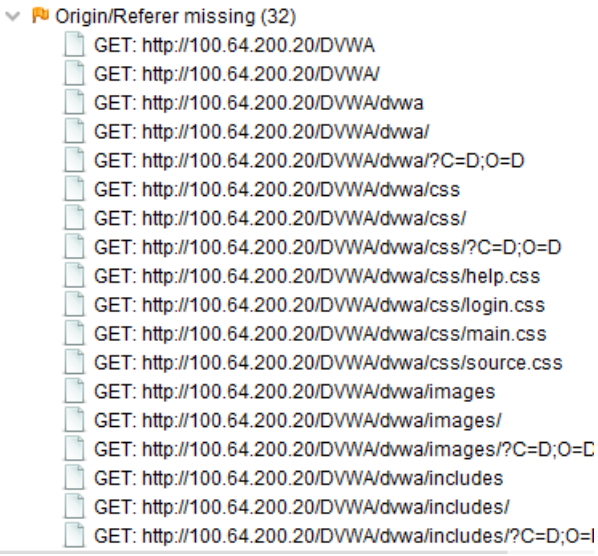


Obrázek 5.1: Izolace nově vytvořených pasivních pravidel

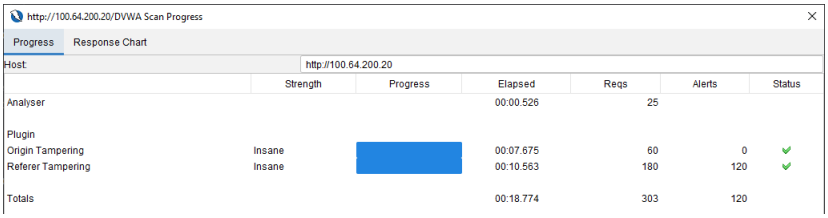


Obrázek 5.2: Izolace nově vytvořených aktivních pravidel

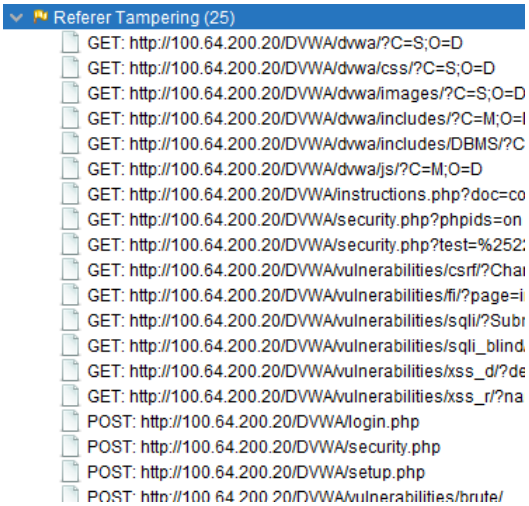
Penetrační test hlásil upozornění ve všech skenovaných kategorií. Detailnějším prozkoumáním bylo zjištěno, že se z většiny jedná o žádosti typu GET, které odkazovaly na hlavní stránku a soubory v aplikaci. Zbytek byl vyhodnocen jako falešná pozitivita, jelikož webová aplikace v sobě obsahuje zabezpe-



Obrázek 5.3: Závady detekovány pasivními pravidly ve cvičném prostředí



Obrázek 5.4: Průběh aktivního testování ve cvičném prostředí



Obrázek 5.5: Závady detekovány aktivními pravidly ve cvičném prostředí

čení proti zranitelnosti způsoby anti-CSRF tokeny a same-site cookies, které nebyly otestovány, jelikož byly předem deaktivované v pravidlech. Výsledkem penetračního testu je nepřítomnost zranitelnosti CSRF ve webové aplikaci. Následně s provozovatelem proběhla schůzka, v níž byl prozrazen výsledek penetračního testu.

Obrázek 5.6: Závady po kompletním skenování ve skutečném prostředí

5.3 Výsledek

Rozšíření objevilo místa v obou webových aplikacích, které označilo za zranitelná. Bližším zkoumáním došlo k závěru, že nástroj hlásí spoustu falešných pozitivit, neboť pravidla skenování spoléhají na chybnou nebo chybějící konfiguraci ve webových žádostech a netestují další sounáležitosti zabezpečení vůči zranitelnosti CSRF. Zapnutím ostatních skenovacích pravidel v nástroji lze vyloučit četnost falešných pozitiv a tím pádem získat přesnější odhad, kde by se zranitelnost mohla nacházet. Výsledek je tedy považován za úspěšný, neboť nástroj skutečně dokázal vyhledat zranitelnost ve webové aplikaci.

Závěr

Cílem bakalářské práce byl návrh, implementace a otestování rozšíření pro nástroj OWASP ZAP na vyhledání zranitelnosti CSRF ve webové aplikaci.

V rámci bakalářské práce došlo k seznámení se se zranitelností CSRF, jejím dopadem a ochranou. Na základě analýzy útoku a nástrojů, které zranitelnost vyhledávají, byl zvolen přístup řešení přes modifikaci a ověření webových žádostí v aplikaci. V případě pasivního skenování byla nástroji přidaná funkcionalita kontroly parametrů v hlavičkách žádostí a v případě aktivního skenování dochází k jejich manipulaci a sledování reakce webové aplikace. Pro implementaci byl použit programovací jazyk Java.

Rozšíření bylo prvně otestováno ve cvičném prostředí se zranitelností a následně ve skutečném se souhlasem provozovatele webové aplikace. Nástroj v nich úspěšně dokázal díky rozšíření vyhledat potenciální slabá místa. Výsledek testování byl nahlášen a prokonzultován s provozovatelem. Rozšíření je připravené pro využití během penetračního testování webových aplikací, kde napomáhá automatizované detekci zranitelnosti.

V analýze došlo k popsání alternativních způsobů vyhledávání slabých míst, které nástroj ZAP nemá implementované. Jejich vytvořením lze rozšířit schopnost detekovat zranitelnosti a tím pádem snížit četnost falešných pozitiv během penetračního testování.

Literatura

[1] WannaCry ransomware used in widespread attacks all over the world. [online], květen 2017, [cit. 2022-05-08]. Dostupné z: https://securelist.com/wannacry-ransomware-used-in-widespread-attacks-all-over-the-world/78351/

[2] OWASP Top Ten Web Application Security Risks | OWASP. [online], [cit. 2022-04-01]. Dostupné z: https://owasp.org/www-project-top-ten/

[3] Kumar, S.: An Introduction to Cyber Security Basics for Beginner. [online], červenec 2021, [cit. 2022-04-01]. Dostupné z: https://geekflare.com/understanding-cybersecurity/

[4] John von Neumann's Cellular Automata | The Embryo Project Encyclopedia. [online], červen 2010, [cit. 2022-03-30]. Dostupné z: https://embryo.asu.edu/pages/john-von-neumanns-cellular-automata

[5] Johansen, A. G.: What Is A Computer Virus? [online], červenec 2020, [cit. 2022-04-02]. Dostupné z: https://us.norton.com/internetsecurity-malware-what-is-a-computer-virus.html

[6] 1970s. [online], [cit. 2022-04-02]. Dostupné z: https://encyclopedia.kaspersky.com/knowledge/years-1970s/

[7] 1980s. [online], [cit. 2022-04-02]. Dostupné z: https://encyclopedia.kaspersky.com/knowledge/years-1980s/

[8] Regan, J.: A Brief History of Computer Viruses. [online], červen 2020, [cit. 2022-03-26]. Dostupné z: https://www.avg.com/en/signal/history-of-viruses

[9] Chadd, K.: The history of cybersecurity. [online], listopad 2020, [cit. 2022-03-22]. Dostupné z: https://blog.avast.com/history-of-cybersecurity-avast

[10] What Is Ransomware? | Trellix. [cit. 2022-04-13]. Dostupné z: https://www.trellix.com/en-us/security-awareness/ransomware/what-is-ransomware.html

[11] Ghafur, S.; Kristensen, S.; Honeyford, K.; aj.: A retrospective impact analysis of the WannaCry cyberattack on the NHS. ročník 2, č. 1, říjen 2019: s. 1–7, ISSN 2398-6352, doi:10.1038/s41746-019-0161-6, [cit. 2022-04-13]. Dostupné z: https://www.nature.com/articles/s41746-019-0161-6

[12] Burdova, C.: What Is EternalBlue and Why Is the MS17-010 Exploit Still Relevant? [online], březen 2022, [cit. 2022-04-13]. Dostupné z: https://www.avast.com/c-eternalblue

[13] Cyber-attack: US and UK blame North Korea for WannaCry. prosinec 2017, [cit. 2022-04-13]. Dostupné z: https://www.bbc.com/news/world-us-canada-42407488

[14] What is a CVE? Common Vulnerabilities and Exposures Explained | UpGuard. [online], [cit. 2022-04-14]. Dostupné z: https://www.upguard.com/blog/cve

[15] Loshin, P.: What is OWASP? What is the OWASP Top 10? All You Need to Know. [online], [cit. 2022-04-04]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/OWASP

[16] AppCheck & the OWASP Penetration Testing Checklist | AppCheck. [online], září 2021, [cit. 2022-03-24]. Dostupné z: https://appcheck-ng.com/appcheck-the-owasp-penetration-testing-checklist/

[17] What Is the OWASP Top 10 and How Does It Work? | Synopsys. [online], [cit. 2022-04-07]. Dostupné z: https://www.synopsys.com/glossary/what-is-owasp-top-10.html

[18] A01 Broken Access Control - OWASP Top 10:2021. Dostupné z: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

[19] Secure against the OWASP Top 10 for 2021 | Chapter 2: Cryptographic failures (A2). [online], únor 2022, [cit. 2022-04-03]. Dostupné z: https://support.f5.com/csp/article/K00174750

[20] Muscat, I.: What Are Injection Attacks. [online], duben 2019, [cit. 2022-04-04]. Dostupné z: https://www.acunetix.com/blog/articles/injection-attacks/

[21] Bathla, S.: A04:2021-Insecure Design. [online], září 2021, [cit. 2022-04-05]. Dostupné z: https://medium.com/@shivam_bathla/a04-2021-insecure-design-9e16449f29ef

[22] Sundar, V.: What is Owasp Security Misconfiguration and How to Prevent it? [online], srpen 2014, [cit. 2022-04-15]. Dostupné z: https://www.indusface.com/blog/owasp-security-misconfiguration/

[23] Secure against the OWASP Top 10 for 2021| Chapter 6: Vulnerable and outdated components (A6). [online], únor 2022, [cit. 2022-04-15]. Dostupné z: https://support.f5.com/csp/article/K17045144

[24] Hierseman, N.: OWASP Top 10 Deep Dive: Identification and Authentication Failures | Rapid7 Blog. [online], prosinec 2021, [cit. 2022-04-15]. Dostupné z: https://www.rapid7.com/blog/post/2021/12/01/owasp-top-10-deep-dive-identification-and-authentication-failures/

[25] Making sense of OWASP A08:2021 – Software & Data Integrity Failures. [online], říjen 2021, [cit. 2022-04-05]. Dostupné z: https://www.securityjourney.com/post/making-sense-of-owasp-a08-2021-software-data-integrity-failures

[26] Kiprin, B.: Insufficient Logging and Monitoring: Ultimate Guide 2022. [online], srpen 2021, section: Vulnerability Prevention. Dostupné z: https://crashtest-security.com/insufficient-logging-monitoring-guide/

[27] What is SSRF (Server-side request forgery)? Tutorial & Examples | Web Security Academy. [online], [cit. 2022-04-15]. Dostupné z: https://portswigger.net/web-security/ssrf

[28] What is the purpose of a Penetration Test? [online], březen 2019, [cit. 2022-04-18]. Dostupné z: https://www.packetlabs.net/posts/purpose-of-a-penetration-test/

[29] Understanding the Phases of the Penetration Testing Process. [online], [cit. 2022-04-18]. Dostupné z: https://eccouncil.org/cybersecurity-exchange/penetration-testing/phases-penetration-testing-process/

[30] What is Kali Linux? | Kali Linux Documentation. [online], [cit. 2022-05-01]. Dostupné z: https://www.kali.org/docs/introduction/what-is-kali-linux/

[31] Azad, U.: Parrot Security OS: Product Review. [online], [cit. 2022-05-08]. Dostupné z: https://linuxhint.com/parrot_security_os_review/

[32] RedTeam Pentesting GmbH - Penetration Test / Reconnaissance: Information Gathering before the Attack. [online], [cit. 2022-04-18]. Dostupné z: https://www.redteam-pentesting.de/en/pentest/reconnaissance/-penetration-test-reconnaissance-information-gathering-before-the-attack

[33] Thelberg, S.: Nmap - what is it and how does it work? [online], leden 2021, [cit. 2022-04-18]. Dostupné z: https://www.holmsecurity.com/resources/what-is-nmap

[34] Bintahin, M.: What is DirBuster and How to Use it? [online], březen 2022, [cit. 2022-04-18]. Dostupné z: https://cyber-today.com/what-is-dirbuster-and-how-to-use-it/

[35] Sharma, A.: John the Ripper explained: An essential password cracker for your hacker toolkit. [online], červenec 2020, [cit. 2022-05-08]. Dostupné z: https://www.csoonline.com/article/3564153/john-the-ripper-explained-an-essential-password-cracker-for-your-hacker-toolkit.html

[36] How to use Burp Suite for penetration testing. [online], [cit. 2022-04-20]. Dostupné z: https://portswigger.net/burp/documentation/desktop/penetration-testing

[37] Gierach-Pacanek, K.: Linux local Privilege Escalation Awesome Script (linPEAS) analysis. [online], květen 2021, [cit. 2022-04-20]. Dostupné z: https://blog.cyberethical.me/linpeas

[38] What is Metasploit: Overview, Framework, and How is it Used | Simplilearn. [online], [cit. 2022-04-20]. Dostupné z: https://www.simplilearn.com/what-is-metaspoilt-article

[39] Dizdar, A.: CSRF Attacks: Real Life Attacks and Code Walkthrough. [online], únor 2021, [cit. 2022-05-10]. Dostupné z: https://brightsec.com/blog/csrf-attack/

[40] CSRF Attacks: Anatomy, Prevention, and XSRF Tokens. [online], [cit. 2022-05-08]. Dostupné z: https://www.acunetix.com/websitesecurity/csrf-attacks/

[41] Defending against CSRF with SameSite cookies | Web Security Academy. [online], [cit. 2022-05-10]. Dostupné z: https://portswigger.net/web-security/csrf/samesite-cookies

[42] Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series. [online], [cit. 2022-05-09]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[43] DVWA Ultimate Guide - First Steps and Walkthrough. [online], duben 2021, [cit. 2022-04-24]. Dostupné z: https://bughacking.com/dvwa-ultimate-guide-first-steps-and-walkthrough/

Příloha A

Seznam použitých zkratek

API Application Programming Interface.

ARPANET Advanced Research Projects Agency Network.

CPU Central Processing Unit.

CSRF Cross Site Request Forgery.

CVE Common Vulnerabilities and Exposures.

DNS Domain Name System.

DOS Disk Operating System.

DVWA Damn Vulnerable Web Application.

GDPR General Data Protection Regulation.

GPU Graphics Processing Unit.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

HUD Heads Up Display.

IBM International Business Machines.

IP Internet Protocol.

Nmap Network Mapper.

NSA National Security Agency.

ORM Object Relational Mapping.

OS Operating System.

OSINT Open Source Intelligence.

OWASP Open Web Application Security Project.

PEAS Privilege Escalation Awesome Script.

SMS Short Message Service.

SQL Structured Query Language.

TOR The Onion Router.

URI Uniform Resource Identifier.

XSS Cross Site Scripting.

ZAP Zed Attack Proxy.

Příloha B

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF