VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

BEZPEČNOST APLIKACE MCUXPRESSO WEB

MCUXPRESSO WEB APPLICATION SECURITY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Tomáš Mittaš

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Jan Roupec, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Tomáš Mittaš**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Jan Roupec, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Bezpečnost aplikace MCUXpresso Web

Stručná charakteristika problematiky úkolu:

Téměř všechny dnešní webové aplikace jsou věřejně přístupné a náchylné na různé zranitelnosti proti kterém je potřeba se bránit. Webové aplikace musí být bezpečné a odolné vůči různým druhům útoků. Úkolem práce je analyzovat webové zranitelnosti, existující nástroje pro webovou analýzu zranitelností a navrhnout test plán pro testování webové aplikace mcuxpresso.nxp.com.

Cíle diplomové práce:

1. Seznamte se s webovou aplikací MCUXpresso Web SDK Builder.
2. Prozkoumejte, popište, porovnejte techniky a nástroje analýzy web security.
3. Vybranou techniku aplikujte a analyzujte webovou aplikaci MCUXpresso Web SDK Builder, výběr zdůvodněte.
4. Navrhněte obecný testovací plán bezpečnosti pro MCUXpresso Web SDK Builder z prozkoumaných technik.
5. Vyberte část z test plánu bezpečnosti a v libovolném skriptovacím jazyce implementujte script pro automatizaci.

Seznam doporučené literatury:

HOFFMAN, A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications, O'Reilly Media, Sebastopol CA, 2020, ISBN 1492053112.

MCUXpresso web builder, firemní materiál NXP.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Táto práca sa zaoberá testovaním bezpečnosti aplikácie MCUXpresso Web SDK Builder pomocou techník a nástrojov etického hackovania. Na začiatku práce je stručne spomenutá história etického hackovania a štruktúra webových aplikácií. Ďalej sa v práci rozoberá samotná aplikácia z pohľadu užívateľa, jej časti pred prihlásením a po prihlásení do aplikácie a fungovanie tejto aplikácie. Následne je popísaný zoznam najčastejších zraniteľností a slabín webových aplikácií pre pochopenie prípadných nájdených zraniteľností. Práca sa ďalej zaoberá technikami a nástrojmi bezpečnosti webových aplikácií a ich porovnaniu. Predposledná kapitola sa venuje použitiu techniky Analýza a hľadanie zraniteľností na aplikáciu MCUXpresso Web SDK Builder. Na záver je navrhnutý testovací plán bezpečnosti aplikácie, pričom časť tohto plánu je automatizovaná.

ABSTRACT

This thesis deals with testing of the security of web application MCUXpresso Web SDK Builder using ethical hacking techniques and tools. At the beginning, the history of ethical hacking and structure of web applications are briefly mentioned. The thesis then analyses the application itself from the user's point of view, its parts before logging in and after logging in and the operation of this application. The following is a list of the most common vulnerabilities and weaknesses found in web applications to understand any vulnerabilities found. Furthemore, the thesis deals with the techniques and tools of web application security and compares them. The penultimate chapter deals with the use of Analysis and vulnerability scanning technique on the application MCUXpresso Web SDK Builder. Finally, an application security test plan is designed, while part of this plan is automated.

KĽÚČOVÉ SLOVÁ

Zraniteľnosť, slabina, OWASP, hackovanie, sken, testovanie, CWE, MCUXpresso Web SDK Builder, NXP, webová aplikácia, útok, bezpečnosť, nástroj, technika.

KEYWORDS

Vulnerability, weakness, OWASP, hacking, scan, testing, CWE, MCUXpresso Web SDK Builder, NXP, web application, attack, safety, tool, technique.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2022

BIBLIOGRAFICKÁ CITÁCIA

MITTAŠ, Tomáš. *Bezpečnost aplikace MCUXpresso Web*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 89 s. Diplomová práca. Vedúci práce: doc. Ing. Jan Roupec, Ph.D.

POĎAKOVANIE

Touto cestou by som chcel poďakovať mojej priateľke, bratovi, starému otcovi a rodičom, ktorí pri mne počas celého štúdia stáli a podporovali ma. Ďalej by som chcel poďakovať vedúcemu mojej práce doc. Ing. Jánovi Roupcovi, Ph.D. za jeho cenné rady a pripomienky. V neposlednej rade sa chcem poďakovať aj konzultantom Ing. Jakubovi Cieslarovi a Ing. Ondrejovi Balážovi za usmernenie pri tvorbe práce.

ČESTNÉ PREHLÁSENIE

Prehlasujem, že táto diplomová práca je mojím pôvodným dielom, vypracoval som ju samostatne pod vedením doc. Ing. Jána Roupce, PhD. a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry.

Ako autor uvedenej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil autorské práva tretích osôb, predovšetkým som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovení § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov.

V Brne dňa 20. 5. 2022 .....

Tomáš Mittaš

OBSAH

1. ÚVOD

Bezpečnosť je v dnešnej dobe veľmi dôležitou súčasťou všetkých aplikácií a programov. Veľa spoločností ponúka odmeny za nájdenie nedostatkov a bugov v ich produktoch v programe známom ako Bug Bounty (alebo aj Bug Bounty Hunting). Súčasná ochrana v nemalej miere ťaží zo skúseností získaných v minulosti, avšak hackeri v minulosti pracovali inak, ako dnes. Vo všeobecnosti môžeme ľudí, zaoberajúcich sa bezpečnostným inžinierstvom a hackovaním, rozdeliť do niekoľkých skupín. Najväčšie sú Grey hats, Black hats a White hats. Grey hats sú hackeri, ktorým väčšinou ide o objavenie nedostatkov, čím síce často porušujú zákony, no väčšinou nemajú zlé úmysly. Black hats, nazývaní tiež Crackers, sú hackeri, ktorí sa nedovolenou cestou dostávajú k dátam a informáciám, s plánom ich následne zneužiť alebo predať. White hats alebo aj etickí hackeri sú hackeri zamestnaní na nájdenie chýb a nedostatkov v bezpečnosti a ich následnú opravu. K dátam pristupujú oprávnene a s plným vedomím ich vlastníkov, sú teda presným opakom Black hats hackerov. V každej novo vyvinutej technológii sa spolu s novými prvkami objavujú aj nové nedostatky a slabiny, ktoré sú hackeri pripravení využiť. Bezpečnostné inžinierstvo tak vyžaduje znalosti z mnohých odborov ako napríklad kryptografie, sietí, ekonómie, psychológie, a tiež hardware. Dnes existuje mnoho programov a metód na analýzu a zisťovanie nedostatkov a slabín webových aplikácií. Jednou z popredných technológii využívajúcich sa v oblasti bezpečnostného inžinierstva a etického hackovania je operačný systém Linux, predovšetkým distribúcia Kali. Tá obsahuje početné nástroje, ktoré sa dajú použiť na zistenie zraniteľností využiteľných hackermi a bude použitá v tejto práci. Spolu s metódami na riešenie problémov a nedostatkov bezpečnosti však rastú aj typy útokov, ktoré môžu hackeri používať. Okrem čisto technických prostriedkov sa pri hackovaní uplatňuje aj technika Socíalneho inžinierstva alebo hackerstva, kedy hackeri kontaktujú obete (väčšinou nie osobne) a zmanipulujú ich, aby im vydali osobné informácie. Táto práca popíše najaktuálnejšie skupiny zraniteľností webových aplikácií podľa organizácie OWASP.

Hlavnou úlohou práce je aplikovať techniky a nástroje analýzy webovej bezpečnosti na aplikáciu MCUXpresso Web SDK Builder od spoločnosti NXP Semiconductors a zistiť jej prípadné bezpečnostné nedostatky. Táto aplikácia slúži na vytváranie software development kits (SDK) pre hardware od spoločnosti NXP. NXP Semiconductors je jedna z hlavných spoločností v oblasti mikroprocesorovej techniky.

Ďalej sa táto práca bude zaoberať navrhnutím testovacieho plánu bezpečnosti pre MCUXpresso Web SDK Builder a nakoniec, implementáciou scriptu pre automatizáciu testovacieho plánu bezpečnosti. V práci budú použité termíny, ktoré doposiaľ nemajú český, ani slovenský preklad, a tak budú písané v ich pôvodnom anglickom formáte.

2. BEZPEČNOSŤ WEBOVÝCH APLIKÁCIÍ

Základnými princípmi ochrany informácii sú: [1]

- Dôvernosť (Confidentiality) – je súhrn pravidiel, za pomoci ktorých chránime citlivé informácie pred zneužitím neautorizovanými ľuďmi. Je to napríklad šifrovanie dát či autentizácia.
- Integrita (Integrity) – alebo aj celistvosť, zaisťuje, že sú systémové informácie a procesy chránené pred vedomou či nevedomou modifikáciou. Integritu zaručujú napríklad hashovacie funkcie či kontrolný súčet (checksum).
- Prístupnosť (Availability) – znamená, že autorizovaní užívatelia sú schopní pristúpiť k systémovým dátam vtedy, keď potrebujú a tí, ktorí nespĺňajú podmienky pre prístup, ho majú odopretý. Prístupnosť sa rieši zálohovaním či udržiavaním software a systému aktuálnymi.

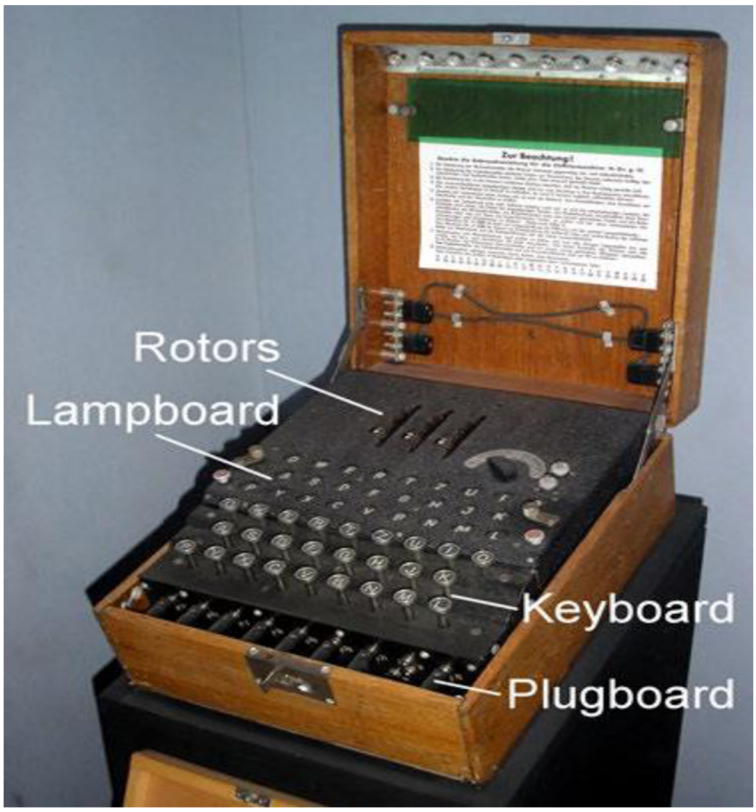
Pre lepšiu predstavu, čo je to hackovanie, si najprv uvedieme jeho históriu. Neskôr sa pozrieme na štruktúru webových aplikácií.

2.1 História hackovania

Za posledné desaťročia sa hackerstvo a hackeri dostali do povedomia ľudí viac ako kedykoľvek predtým. Z toho by sa dalo usúdiť, že hackerstvo je koncept, ktorý vznikol pred nie viac ako 20 až 30 rokmi. To je však iba čiastočne pravdivé. Je síce pravda, že so vznikom služby www (world wide web) sa počet hackerov značne znásobil, ale počiatky hackovania môžeme vidieť už na začiatku 20. storočia (záleží samozrejme na tom, čo si jednotlivec predstavuje pod pojmom hackovanie). Za príklad nám môže slúžiť napríklad manipulácia s prijímačmi a vysielačmi Morzeovho kódu, či zasahovanie do prenosu rádiových vĺn. [2]

Jedným z prvých veľkých pokrokov v bezpečnom prenose dát bolo šifrovacie zariadenie Enigma skonštruované Nemcami na konci 1. svetovej vojny a často používané Nemcami v 2. svetovej vojne. Používalo typ šifrovania, ktorý by sme dnes nazvali ako symetrické (šifrovanie aj dešifrovanie prebiehalo za pomoci rovnakého kľúča). Vynález takéhoto stroja znamenal aj v podstate vznik procesu, ktorý sa neskôr začal nazývať hackovanie. Enigma bola prvýkrát prelomená poľským matematikom Marianom Rejewskim a jeho tímom. Nemci však zložitosť prístroja neustále zvyšovali, až sa pre Mariana a jeho tím stalo nemožným prelomiť šifrovanie v reálnom čase. Tu môžeme pozorovať vzťah hackerov a tých, ktorí sa snažia hackerom ubrániť, ako neustále sa vyvíjajúci proces, kedy jedna aj druhá strana ustavične vytvára nové a lepšie techniky na obranu a útok. Problémom teda začínal byť ľudský faktor, ktorý nedokázal počítať s veľkým počtom kombinácii použitom v šifrovaní. Tento problém sa riešil automatizáciou prelamovania kódu, kedy bol na prelomenie šifrovania použitý prístroj Bombe a pod vedenením Alana Turinga bola Enigma prelomená. Turing staval na informáciách, ktoré mal od Rejewskeho a jeho tímu. Bombe využívalo elektricky

poháňané mechanické rotory, ktoré sa snažili nájsť ekvivalentnú pozíciu rotorov v prístroji Enigma. Vďaka pravidelným správam o počasí, ktoré si Nemci posielali, boli Turing a jeho tím schopní navrhnúť správnu konfiguráciu Bombe spolu s adekvátnym počtom rotorov. Vďaka tomu, že vedeli čo je posielané do správne nakonfigurovanej Enigmy ako vstup, boli schopní omnoho ľahšie algoritmicky určiť možný výstup. Týmto spôsobom Turing určil konfiguráciu stroja spolu so správnym počtom rotorov. Proces, ktorí matematici pri prelamovaní Enigmy používali, sa nazýva reverzné inžinierstvo a stratégia, použitá Turingom a jeho tímom, je dnes známa pod menom Known plaintext attack. Táto stratégia je algoritmus, ktorý sa značne zjednodušuje pokiaľ má prístupné vstupné a výstupné dáta systému/stroja, na ktorý má byť použitá. Hackeri dnes používajú podobné techniky na prelamovanie šifrovania uložených dát alebo šifrovania použitého v software. Prístroj Bombe, ktorý Turing vytvoril a upravil, môžeme považovať za prvý automatizovaný hackerský nástroj. Na obrázku 1. môžeme vidieť prístroj Enigma. [2]



Obr. 1 Prístroj Enigma [2]

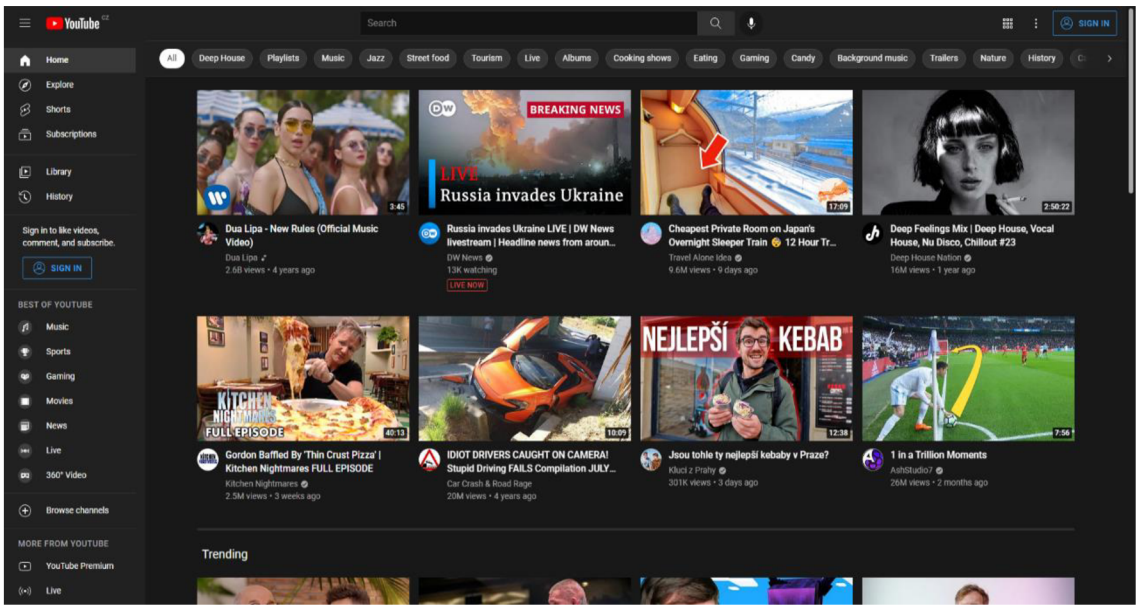
Ďalším miľníkom v hackovaní bolo Telephone phreaking. S hromadným zavedením telefónnej siete a trvajúcim konfliktom Studenej vojny začala narastať potreba odpočúvať rozhovory (napríklad potencionálnych špiónov). Každé číslo na telefóne emitovalo špecifickú frekvenciu, ktorá bola prenesená cez linku do prepájacieho centra. Prepájací prístroj preložil tieto zvuky na čísla a presmeroval hovor odpovedajúcemu príjemcovi. Tento systém bol známy ako vytáčanie tónu (tone dialing), pričom netrvalo dlho a ľudia prišli na to, ako vybudovať systém, ktorý bude tieto zvukové tóny

manipulovať. Spolu so vznikom takýchto systémov a ich väčším rozšírením začali vznikať systémy, ktoré slúžili na obranu proti Telephone phreaking technológii. [2]

O hackovaní počítačov môžeme hovoriť približne od roku 1980. Po značnom rozšírení počítaču Commodore 64 začali byť počítače bežnými prístrojmi používanými firmami, ale aj domácnosťami na spracovanie opakujúcich sa úloh, riadenie financií, účtovníctvo, predaj a podobne. V roku 1983 Fred Cohen vytvoril prvý počítačový vírus. Tento vírus bol schopný vytvárať kópie samého seba a vedel sa rozšíriť z jedného počítača na iný pomocou diskety. Cohen vírus zamaskoval ako legitímny program, čím ho ukryl pred všetkými, ktorí nemali prístup do zdrojového kódu. Ďalším pionierom bol Robert Morris, ktorý vytvoril prvý vírus schopný napádať počítače mimo laboratórium. Dostal názov Morris Worm, pričom sa slovo worm (červ) začalo používať na vírusy schopné replikácie samého seba. Morris bol potrestaný vládou Spojených štátov amerických, čím sa stal prvým potrestaným hackerom v histórii. [2]

Keď vzrástla popularita služby www, znamenalo to novú dobu pre hackerov. Spočiatku sieť slúžila hlavne na zdieľanie dokumentov napísaných v HTML (HyperText Markup Language, v preklade Hypertextový značkovací jazyk). Od počiatku 21. storočia však internet zažil novú éru, kedy stránky začali ukladať dáta poskytnuté užívateľmi a bola možná modifikácia funkcionality stránky na základe užívateľského vstupu. Vďaka tomuto neskôr vznikol Web 2.0, ktorý užívateľom umožňoval navzájom spolupracovať za pomoci zdieľania ich vstupov cez http (hypertext transfer protocol, v preklade hypertextový prenosový protokol) na webový server a ten ich vstupy uložil a tým ich zdieľal s ostatnými užívateľmi na základe žiadostí. Nová ideológia výstavby webových stránok dala vzniknúť sociálným médiám ako ich poznáme aj dnes. Web 2.0 umožnil blogy, wikipédie, stránky na zdieľanie médii a veľa iných. Táto radikálna zmena vo webovej ideológii znamenala zmenu z platformy na zdieľanie dokumentov na platformu distribúcie aplikácií. S ohľadom na túto skutočnosť tiež došlo k zmene spôsobu útoku hackerov na webové aplikácie. Pred tým sa kládol dôraz hlavne na zabezpečenie dvoch hlavných vektorov útokov hackerov, a to serverov a sietí. Pri vzostupe stránok založených na aplikáciách sa stal užívateľ hlavným terčom útoku hackerov. V tejto dobe bolo žiaľbohu implementovaných príliš málo mechanizmov na ochranu užívateľov pred útokmi a navyše len málo užívateľov vtedy chápalo technológii, s ktorou pracovali. Začali sa vo veľkom objavovať útoky DoS (Denial of Service, v preklade Odopretie služby). O pár rokov neskôr začali hackeri široko používať phisingové stránky na ukradnutie údajov. Pretože neexistovali prostriedky ochrany užívateľov proti týmto typom stránok, boli úspešní. Ďalšou veľkou zraniteľnosťou zneužívanou hackermi boli zraniteľnosti XSS (Cross-Site Scripting), ktoré budú v práci popísané neskôr. [2]

Posledná (a aktuálna) etapa hackovania sa počíta od roku 2015. Existencia webových aplikácií ako je Facebook, Google, YouTube a mnohé iné znamená kvanta dát, obrázkov a videí, ktoré je treba zabezpečiť proti hackerom. Veľa dát existuje permanentne skrz relácie a veľké množstvo dátových prvkov je uložených v prehliadači namiesto servera. [2]



Obr. 2 YouTube, typický príklad aplikácie Web 2.0

Ďalším trendom dnešných software spoločností je presúvať svoje produkty na cloudy, čo sú v podstate komplexné siete zložené zo serverov. Ako môžeme vidieť, vo webových aplikáciách je investovaných nemálo peňazí, a preto sú dnes hackeri motivovanejší ako kedykoľvek predtým. Značne pokročili aj webové prehliadače, ktoré majú zakomponovaných veľa bezpečnostných prvkov. Sú značne robustné a nasledujú bezpečnostnú špecifikáciu Same Origin Policy (SOP), ktorá hovorí, že webová stránka A nemôže získať prístup na webovú stránku B, aj keby boli obe otvorené naraz, alebo jedna vbudovaná v druhej. Ďalším konceptom bezpečnosti je napríklad Content Security Policy (CSP), ktorý dovoľuje vývojárom špecifikovať niekoľko úrovní bezpečnosti. [2]

2.2 Štruktúra moderných webových aplikácií

Dnešné webové aplikácie sú značne odlišné od tých, ktoré existovali pred dekádami. Nástroje, ktoré používajú, sú tak pokročilé oproti tým, aké používali staré webové aplikácie, že sa dnešný vývoj webových aplikácií zdá ako kompletne iná špecializácia oproti vývoju pred dekádami. [2]

V minulosti bola väčšina webových aplikácií založená na frameworkoch na strane servera, ktoré rendrovali HTML/JS (Javascript) / CSS (Cascading Style Sheets) stránku a tá bola následne poslaná klientovi. Pri potrebe aktualizácie stránky klient poslal žiadosť na ďalšiu stránku na vyrendrovanie serveru a následné poslanie cez http. Používanie http naďalej vzrástlo so vzostupom AJAX (Asynchronous Javascript and XML, v preklade Asynchrónny Javascript a XML), ktorý umožnil posielať žiadosti siete z page session za pomoci Javascriptu. [2]

Dnes je veľa aplikácií reprezentovaných ako dve alebo viacero aplikácií komunikujúcich spolu cez sieťový protokol, namiesto reprezentácie ako jedna monolitická aplikácia. Toto je jeden z hlavných rozdielov dnešných webových aplikácií oproti webovým aplikáciám používaným v minulosti. Ako už bolo spomínané, dnešné

webové aplikácie sú v podstate viaceré aplikácie spojené spolu pomocou REST (Representational State Transfer) API (Application Proggraming Interface, v preklade aplikačné programové rozhranie). Tieto API sú bezstavové a existujú iba na to, aby vykonali požiadavky z jednej aplikácie do druhej. To znamená, že neukladajú žiadne informácie o žiadateľovi. Mnohé dnešné klientske aplikácie, ktoré bežia v prehliadači pripomínajú tradičné desktopové aplikácie. Riadia si svoje cykly dĺžky života (life cycle loops), vyžiadavajú si vlastné dáta a nepotrebujú opätovné načítanie stránky po počiatočnom nabootovaní. Taktiež je celkom bežný prípad, kedy aplikácia vo webovom prehliadači komunikuje s viacerými servermi súčasne. Na základe predchádzajúcich tvrdení môžeme zhodnotiť, že dnešné moderné webové aplikácie sú kombináciou viacerých samostatných, ale navzájom symbiotických aplikácií, ktoré spolupracujú. Toto je možné vďaka precízne definovaným sieťovým protokolom a architektonickým vzorom API. [2]

Priemerná aplikácia v dnešnej dobe pravdepodobne používa niekoľko z nasledujúcich technológii: [2]

- REST API
- JSON alebo XML
- JavaScript
- SPA framework (React, Vue, EmberJS, AngularJS)
- Autentizačný a autorizačný systém
- Jeden alebo viacero webových serverov (typicky Linux servery)
- Jeden alebo viacero software balíkov webových serverov (ExpressJS, Apache, NginX)
- Jeden alebo viacero databáz (MySQL, MonoDB, atď.)
- Lokálne úložisko dát na strane klienta (cookies, IndexDB)

REST API

REST API je v podstate API, ktoré: [2]

- Musí byť oddelené od klienta (REST API sú jednoduché a silne škálovateľné).
- Musí byť bezstavové (stateless). To znamená, že API neukladá stav klientovho pripojenia, iba preberá vstupy a poskytuje výstupy.
- Musí byť ľahko cachovateľné, pričom by tieto cache mali byť programátorsky vybavené, aby neposkytovali privilegované informácie inému používateľovi.

Kedysi aplikácie používali SOAP (Simple Object Access Protocol, v preklade Jednoduchý objektový prístupový protokol). REST má však oproti SOAP niekoľko výhod. Prvou je, že vyžaduje cielené dáta, nie funkcie. Ďalšou výhodou REST je, že ľahko cachuje požiadavky a je vysoko škálovateľný. SOAP navyše musí používať XML, pričom REST API môže prijímať dáta v akomkoľvek formáte, no typicky sa používa JSON. [2]

JavaScript Object Notation (JSON)

JSON často slúži ako prenosový formát dát pre REST API. Pretože dnešné aplikácie vyžadujú veľa komunikácie medzi klientom a serverom, je nutné tieto dáta štandardizovať. Toto rieši práve JSON, ktorý nie je proprietárny (je open-standard) a má niekoľko zaujímavých vlastností: [2]

- Je veľmi jednoduchý a rýchly (čím redukuje šírku pásma siete, po anglicky network bandwidth).
- Nevyžaduje takmer žiadne parsovanie, čím redukuje zaťaženie hardware klienta aj serveru.
- Je ľahko čitateľný pre ľudí.
- Je hierarchický, čím reprezentuje zložité vzťahy medzi dátami.
- Objekty JSON sú reprezentované veľmi podobne ako JavaScript objekty, čo uľahčuje prácu prehliadača s jeho objektami.

JavaScript

JavaScript je unikátny programovací jazyk spätý s vývojom internetových prehliadačov a ich Document Object Model (DOM). DOM internetového prehliadača je hierarchická reprezentácia dát, ktorá sa používa na riadenie stavov v moderných internetových prehliadačoch. DOM je tiež štandardnou knižnicou JavaScript. [2]

SPA frameworky

Staršie aplikácie nepoužívali škálovateľný model a tým pádom neboli stavané na komplexné funkcie so zložitou logikou. SPA (Single-page application, v preklade Jednostránková aplikácia) frameworky teda poskytujú komplexné aplikácie založené na JavaScripte, ktoré sú schopné uchovávať si ich vnútorný stav, opakovane používať UI (User Interface, v preklade Užívateľské Rozhranie) komponenty, či riadiť si svoju vlastnú dĺžku životného cyklu. Môžeme ich vidieť pri veľkých aplikáciách ako je Facebook, Twitter alebo YouTube, kedy je ich funkcionalita kľúčová a poskytuje takmer rovnakú skúsenosť, ako desktopové aplikácie. [2]

Autentizačné a autorizačné systémy

Tieto systémy zaisťujú, že sa dáta dostanú k správnemu užívateľovi. Autentizačné systémy nám v jednoduchosti hovoria, že užívateľ „Jan145“ je skutočne „Jan145“ a nie „Jozef232“. Autorizácia zasa udáva, k čomu má užívateľ „Jan145“ oproti užívateľovi „Jozef232“ prístup, a tiež udáva, že by užívateľ „Jan145“ nemal mať prístup k dátam užívateľa „Jozef232“ a naopak. [2]

Webové servery

Aplikačná logika na strane servera beží na softwarovo založených balíkoch web serverov (sotware-based web server packages), takže sa vývojár aplikácie nemusí trápiť zaobchádzaním s požiadavkami a riadením procesov. Software web serverov samozrejme beží na operačnom systéme (typicky Linux distribúcie ako Ubuntu, CentOS alebo

RedHat), ktorý beží na fyzickom hardware v nejakom dátovom centre. Takmer polovicu webových stránok na svete obsluhuje server software Apache, čím sa dá predpokladať, že väčšina webových aplikácií taktiež používa Apache. Apache je open-source a beží na takmer každom Linux servery a tiež na niektorých Windows serveroch. Jeho najväčším konkurentom je Nginx obsluhujúci okolo 30% web serverov, pričom toto číslo rýchlo rastie. Nginx sa používa pre veľkoobjemové aplikácie s veľkým počtom jedinečných spojení. Za Nginx sa s popularitou nachádza Microsoft ISS, používaný hlavne pri Microsoft orientovaných technológiách. Toto však môže byť prekážkou, ak chce firma používať iné technológie. Drahá licencia a neodstatok kompatibility s open-source balíkmi založenými na Unix sú ďalšími dôvodmi, ktoré spôsobili úpadok popularity Microsoft IIS. [2]

Databázy na strane servera

Dáta, ktoré klient pošle serveru sa musia uložiť na ďalšiu reláciu (session). Keďže ukladanie dát do pamäte z dlhodobého hľadiska nie je spoľahlivé, napríklad kvôli reštartom, tak takmer všetky dnešné webové aplikácie používajú databázy na ukladanie ich dát. Najpopulárnejšími databázami na všeobecné použitie sú SQL databázy. Medzi najväčšie patria MySQL, PostgreSQL, Microsoft SQL Server a SQLite. Pokiaľ potrebujeme flexibilnejšie úložisko, je možné použiť NoSQL databázy ako MongoDB, DocumentDB či CouchDB, ktoré ukladajú informácie ako voľne štruktúrované dokumenty, možné kedykoľvek modifikovať. Existujú tiež vysoko špecializované databázy ako napríklad Elasticsearch. [2]

Dátové úložiská na strane klienta

V dnešnej dobe veľa webových aplikácií ukladá dôležitý aplikačný stav (application state) na strane klienta, často vo forme konfiguračných dát alebo veľkých skriptov, ktoré by výrazne spomaľovali sieť, ak by mali byť stiahnuté pri každej návšteve klienta. Vo väčšine prípadov sa úložisko riadené prehliadačom nazýva local storage (lokálne úložisko) a slúži na uloženie prístupových kľúčov a hodnotových dát od klienta, pričom nasleduje SOP. Subset local storage je relačné úložisko (session storage), ktoré ukladá dáta rovnako, no drží ich iba pokiaľ sa nezavrie záložka. [2]

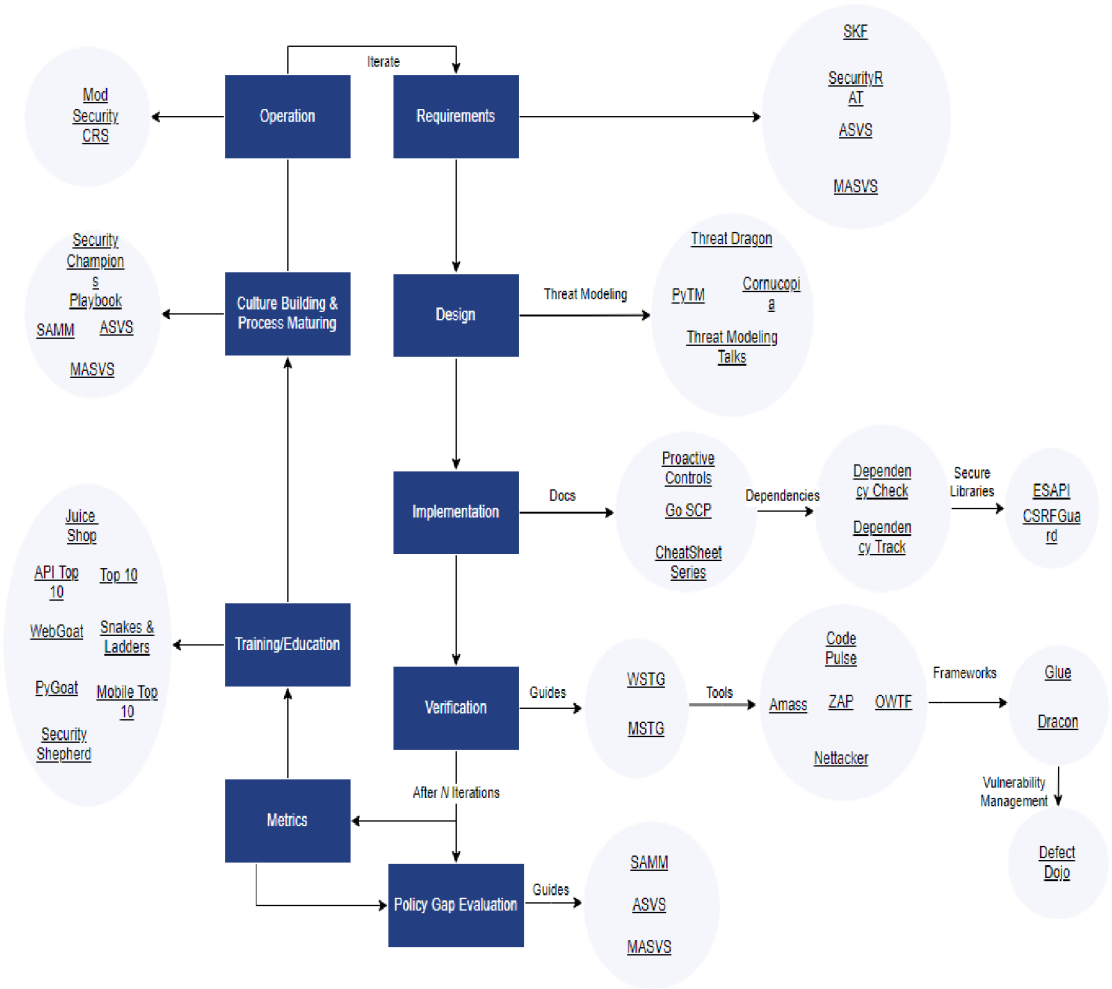
Popis štruktúry webovej aplikácie je dôležitý z hľadiska hľadania a popisu zraniteľností a tiež z hľadiska útokov. Súčasti aplikácie spomenuté v tejto podkapitole často slúžia ako terč útoku hackerov.

2.3 Organizácia OWASP

Organizácia OWASP (Open Web Application Security Project) je nezisková organizácia, zaoberajúca sa zlepšovaním bezpečnosti software. Majú desiatky tisícov členov, veľa projektov a organizujú početné konferencie. Ich projekty, dokumentácia a nástroje sú zadarmo a voľne prístupné každému, kto chce zlepšovať bezpečnosť aplikácií. OWASP Foundation bol spustený 1. decembra 2001. Ich kľúčovými hodnotami sú:

- Otvorenosť – všetko v OWASP je transparentné, od financií až po kód.
- Inovatívnosť – OWASP podporuje inovatívnosť a experimentovanie pri výzvach v software bezpečnosti.
- Globálnosť – ktokoľvek na celom svete sa môže pridať do OWASP komunity.
- Integrita – komunita je otvorená, úprimná a nie je viazaná na žiadneho konkrétneho výrobcu.

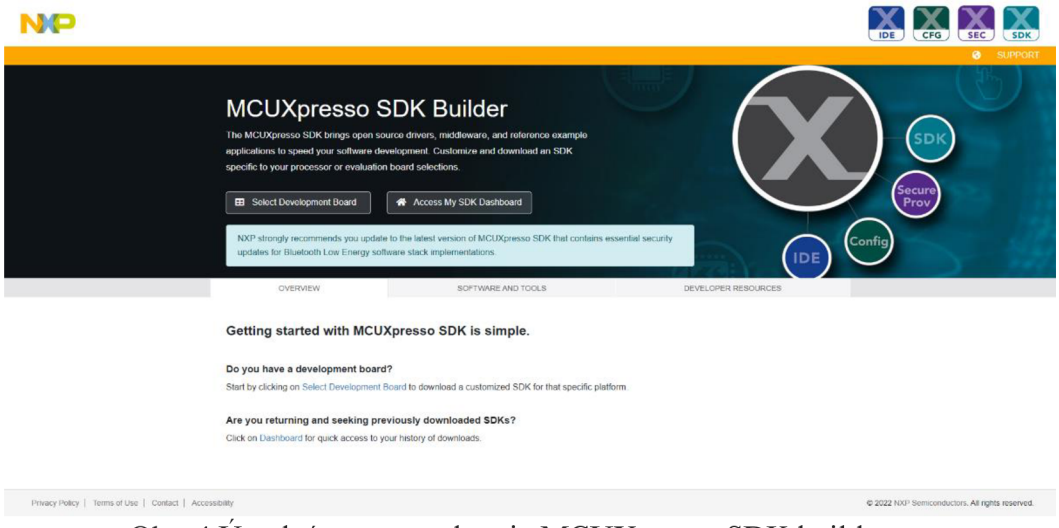
Táto organizácia je dôležitým zdrojom informácii, dokumentácie a nástrojov, ktoré sa budú v rámci tejto práce vyskytovať a používať, a preto si zaslúži osobitné spomenutie. Bude použitý ich model bezpečnostných rizík OWASP TOP 10, testovanie aplikácie prebehne v súladne s ich postupmi a odporúčaniami. Taktiež bude použitý ich nástroj na testovanie webových aplikácií OWASP ZAP. Na obrázku číslo 3 nižšie môžeme vidieť mapu niektorých ich projektov.



Obr. 3 OWASP projekty

3. APLIKÁCIA MCUXpresso Web SDK Builder

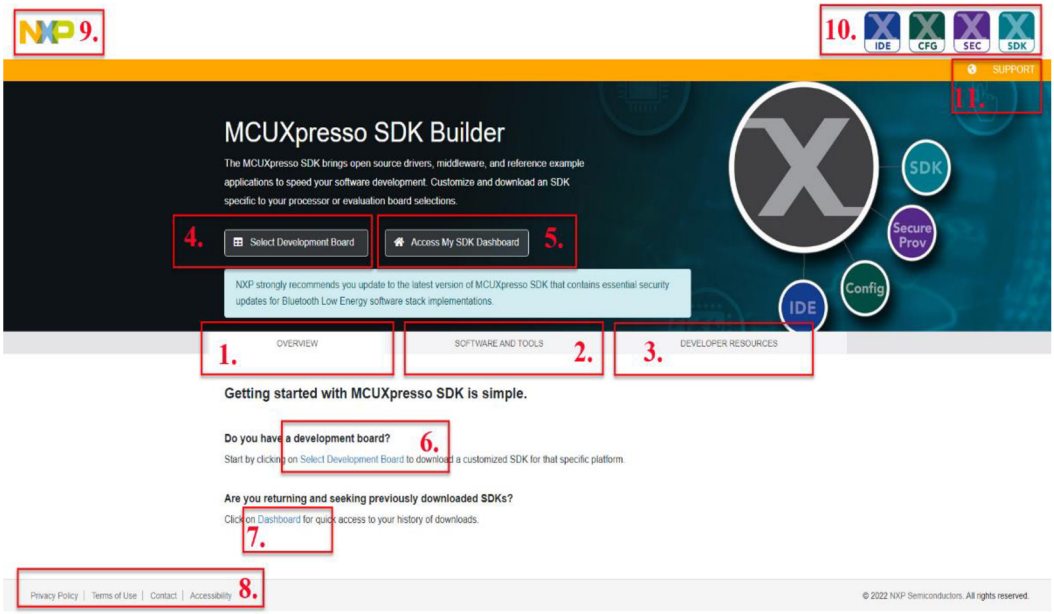
Aplikácia MCUXpresso Web SDK builder slúži na vytvorenie software developement kitov pre procesory firmy NXP Semiconductors. Je to výborný nástroj na zjednodušenie konfigurácie hardware, šetrí a automatizuje prácu. Po otvorení webovej aplikácie v prehliadači sa nám naskytne nasledujúci pohľad (viď obrázok 4):



Obr. 4 Úvodná strana rozhrania MCUXpresso SDK builder

3.1 Súčasti webovej aplikácie MCUXpresso SDK Builder

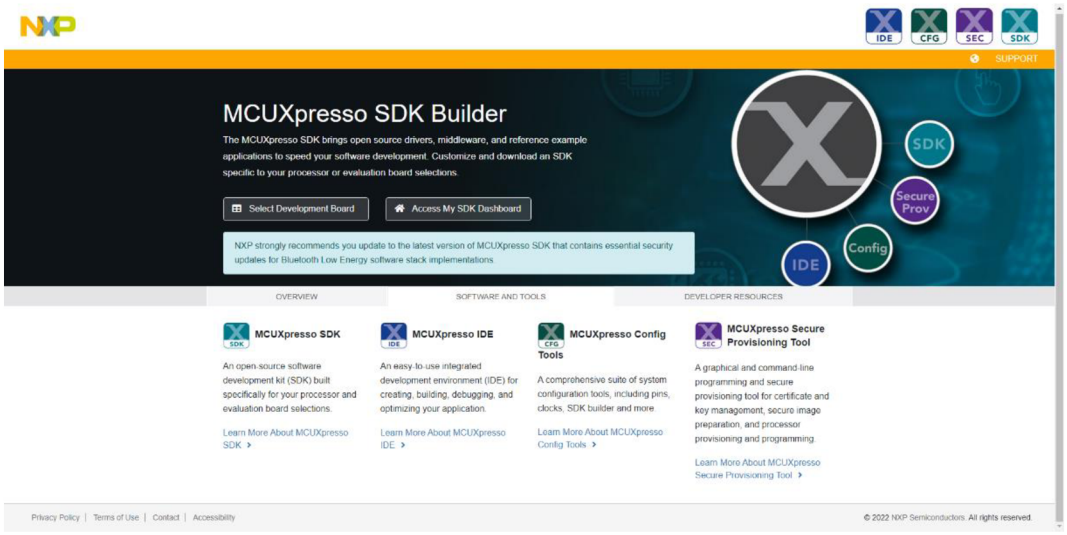
Aplikáciu môžeme na základe najnižšej úrovne zabezpečenia dekomponovať na časť prístupnú pred prihlásením a časť prístupnú po prihlásení. Po označení a vytknutí odkazov aplikácie dostávame nasledovný náhľad, na ktorom môžeme odkazy popísať (viď obrázok 5):



Obr. 5 Dekomponovanie aplikácie na prístupné časti

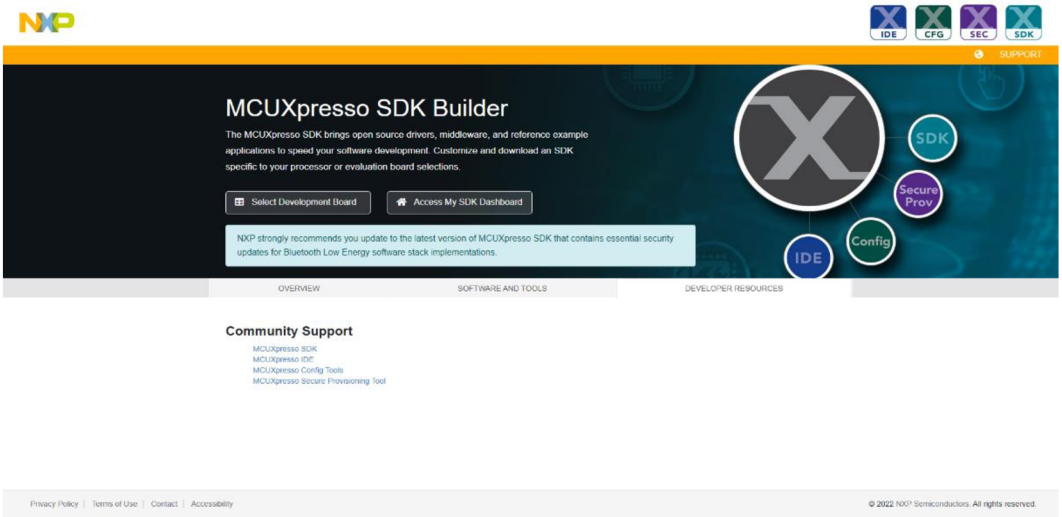
3.2.1 Časť pred prihlásením

Časti pred prihlásením sú časti 1., 2., 3., 8., 9., 10.. Časť 1. s názvom *OVERVIEW* predstavuje aktuálne zobrazenie aplikácie. Ďalšia časť 2., s názvom *SOFTWARE AND TOOLS*, nás odkáže na stránku, kde sú prístupné odkazy na software a nástroje od NXP, ako môžeme vidieť na obrázku 6.



Obr. 6 Software and tools pohľad aplikácie

V *DEVELOPER RESOURCES*, teda časti 3., sú prístupné odkazy pre podporu prostredníctvom komunity NXP, ktoré presmerujú používateľa na stránky pre jednotlivé nástroje od spoločnosti NXP. Rovnako ako aj tieto odkazy na stránky pre podporu sú prístupné bez prihlásenia. Časť 3. môžeme vidieť na obrázku 7.

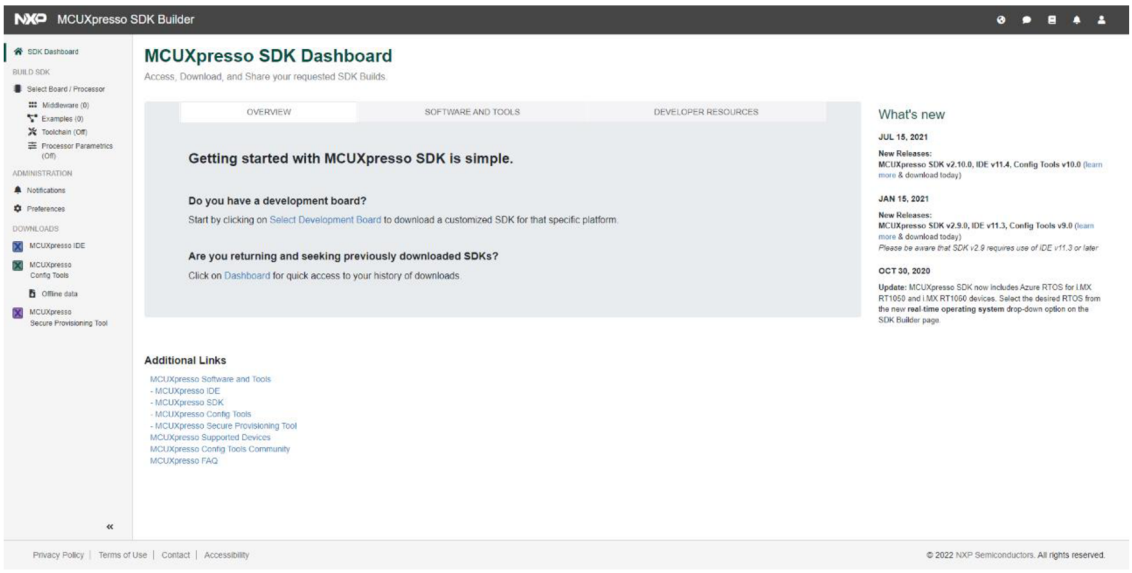


Obr. 7: Developer resources pohľad aplikácie

Časť 10. nás pri zakliknutí jednotlivých nástrojov od NXP odkáže na rovnaké stránky ako časť 2. Časť 9. nás odkáže na domovskú stránku spoločnosti NXP. Takisto odkazy v časti 8. nás odkážu na domovskú stránku spoločnosti NXP, konkrétne na Privacy Policy, Terms of Use, Contact a Accessibillity.

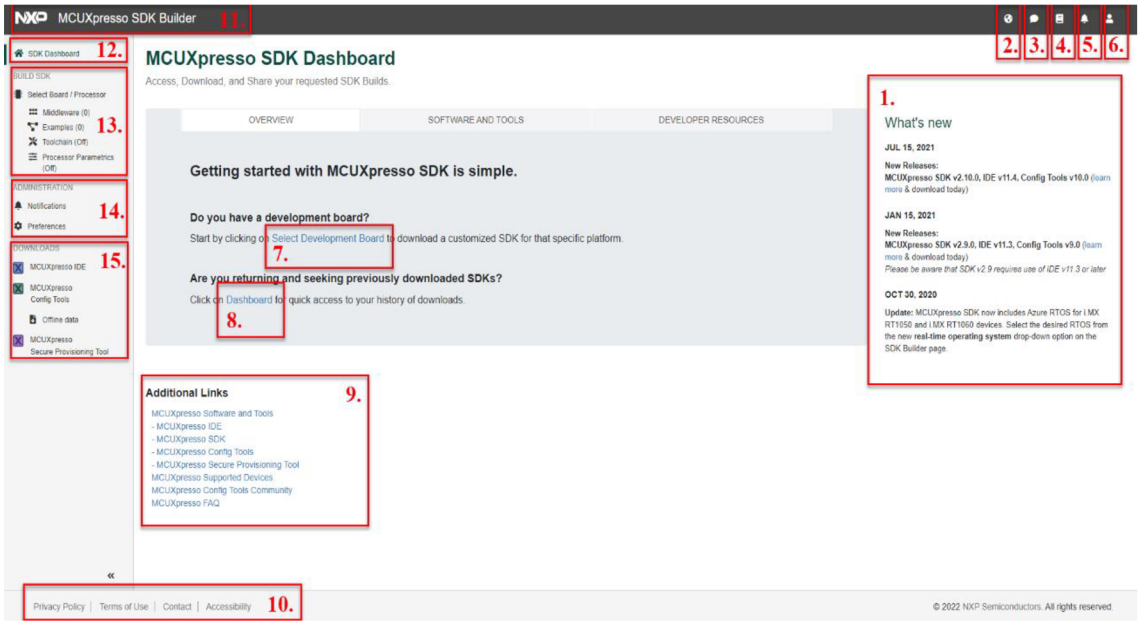
3.2.2 Časť prístupná po prihlásení

Ostatné časti, teda časti 4., 5., 6., 7. nás odkážu na prihlásenie a neoverenému užívateľovi nie sú prístupné. Po zaregistrovaní užívateľa sa teda môžeme dostať do samotnej aplikácie za prihlásením, ako externý užívateľ. Odkazy 5. a 7. nás odkážu na rovnakú stránku v aplikácií. Pohľad, ako aplikácia vyzerá po prihlásení môžeme vidieť na obrázku 8.



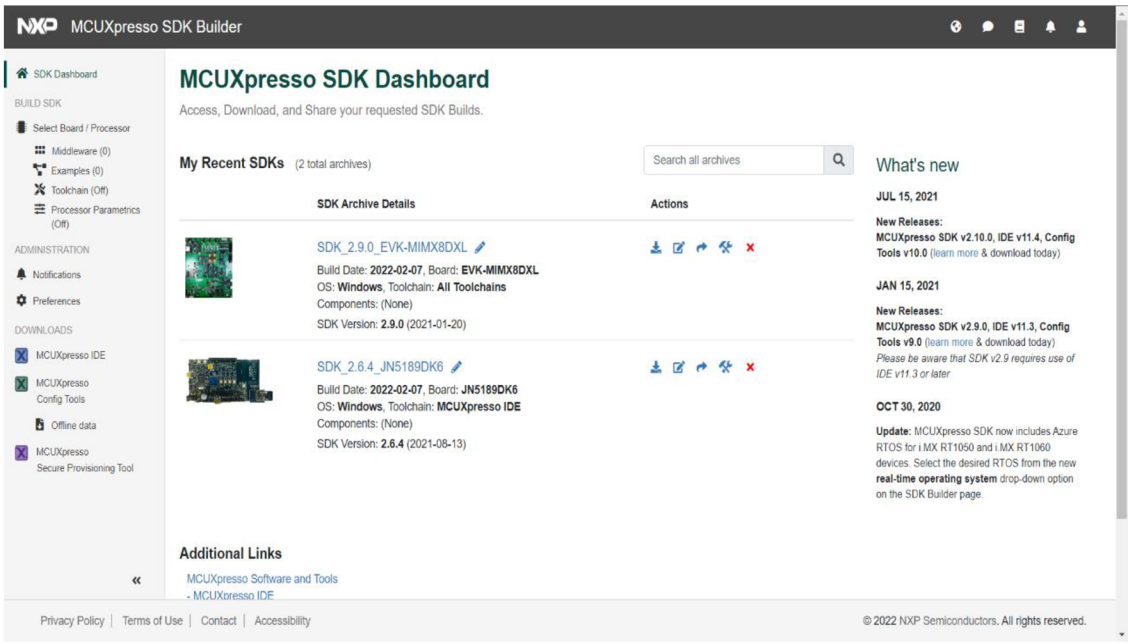
Obr. 8 Pohľad na aplikáciu po prihlásení

Záložky v tomto zobrazení, a teda *OVERVIEW*, *SOFTWARE AND TOOLS* a *DEVELOPER RESOURCES* sú zhodné ako rovnako pomenované záložky v časti pred prihlásením. Opäť je žiadúce aplikáciu po prihlásení dekomponovať na časti a tieto popísať (viď obrázok 9).



Obr. 9 Dekomponovanie aplikácie po prihlásení na časti pre lepšie popísanie

V časti 1. (Obr. 9) sa nachádzajú odkazy na najnovšie releases software od spoločnosti NXP. Na hornej strane Obr. 9 môžeme vidieť časť 2., ktorá slúži na voľbu jazyka (na výber je angličtina a čínština), ďalej časť 3., slúžiaca na poslanie feedbacku (teda spätnej väzby užívateľa) pre aplikáciu MCUXpresso SDK Builder, ďalšou časťou je časť 4., tá nám poskytuje odkazy na dokumentáciu a podporu k software od spoločnosti NXP. Časť 5. (Obr. 6) slúži na oznámenia aplikácie a časť 6. obsahuje lištu s odkazmi na *Preferences* (kde si môžeme nastaviť aké chceme dostávať upozornenia a oznámenia), *Notifications* (rovnaké ako časť 5.) a *Feedback* (rovnaké ako časť 3.). Časť 6. tiež obsahuje možnosť odhlásenia sa z aplikácie. Presunom nášho pohľadu do stredu Obr. 9 vidíme časť 7., ktorá nás odkáže na rovnaké miesto ako časť 13. a bude detailnejšie popísaná pri časti 13.. Časť 8. nás odkáže späť na aktuálny pohľad Obr. 9, no po vybuildení (vystavaní, vyskladaní) SDK nás táto časť odkáže na naše nedávno vybuildené SDK, viď obrázok 10.

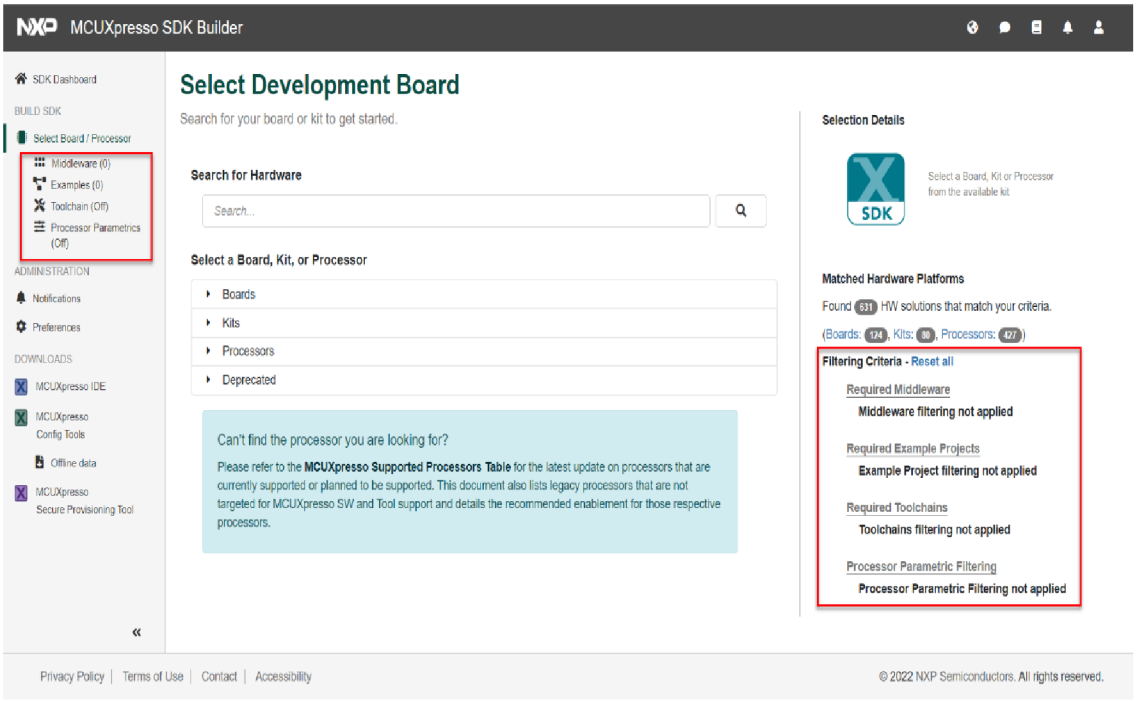


Obr. 10 Pohľad po zakliknutí odkazu v časti 8.

Sem (Obr. 10) nás odkáže aj odkaz z časti 5. z obrázka 2, pokiaľ sme prihlásený užívateľ. Späť na obrázok 6, časť 9. nám ponúka užitočné odkazy na software od NXP a tiež odkazy na stránky komunity pre software NXP, pričom časť 10. je rovnaká ako časť 8. na obrázku 2. aplikácie pred prihlásením. Časť s číslom 11. nás odkáže na pohľad ako na obrázku 1. a časť 12. nás odkáže na pohľad rovnaký ako je obrázok 10. Opäť časť 14. nás len odkáže na už vyššie spomínané funkcie aplikácie, a teda *Notifications* a *Preferences* a časť 15. nás presmeruje pomocou odkazov na stiahnutie software od spoločnosti NXP, máme tu tiež možnosť stiahnúť si Offline data do MCUXpresso Config Tools. Hlavná funkcionalita aplikácie MCUXpresso Web SDK Builder je predstavená časťou 13. Tejto časti sa bude klásť aj väčší dôraz, ako častiam predošlým, pretože bude vysvetľovať prácu s MCUXpresso Web SDK Builder.

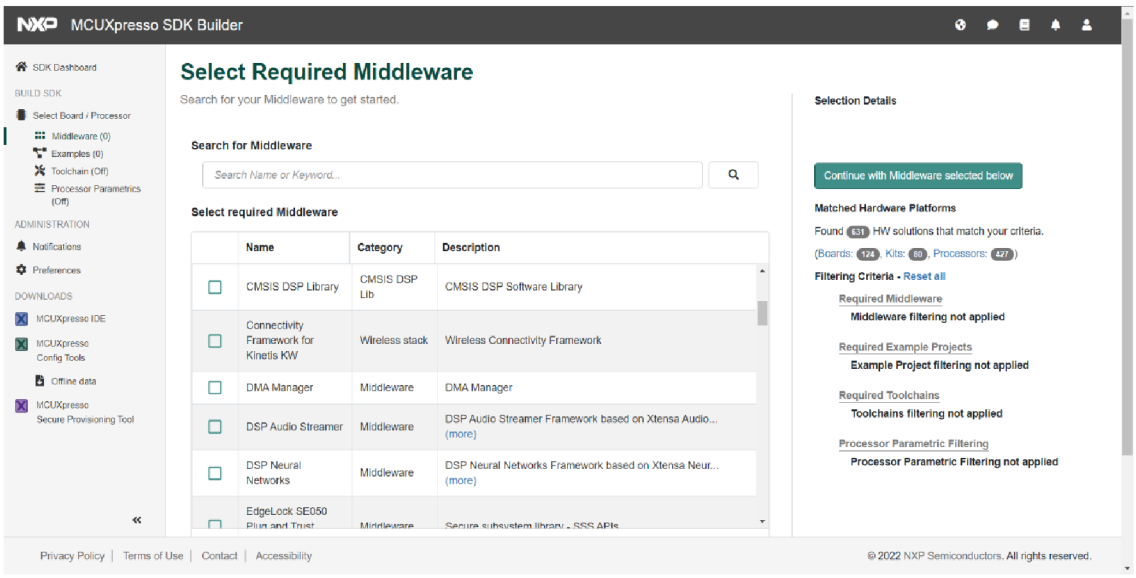
Výstavba SDK pomocou aplikácie MCUXpresso Web SDK Builder

Ako bolo už vyššie spomenuté, aplikácia MCUXpresso Web SDK Builder slúži na buildenie (výstavbu) SDK pre hardware od spoločnosti NXP, a tak šetrí vývojárom čas aj námahu. Úlohou tejto podkapitoly je ukázať funkcionalitu aplikácie a stručný postup, ako vystavať SDK pre náhodne zvolený hardware. Po zakliknutí *Select Board / Processor* v časti 13. v Obr. 10 sa nám naskytne nasledujúci pohľad (viď Obr. 11).



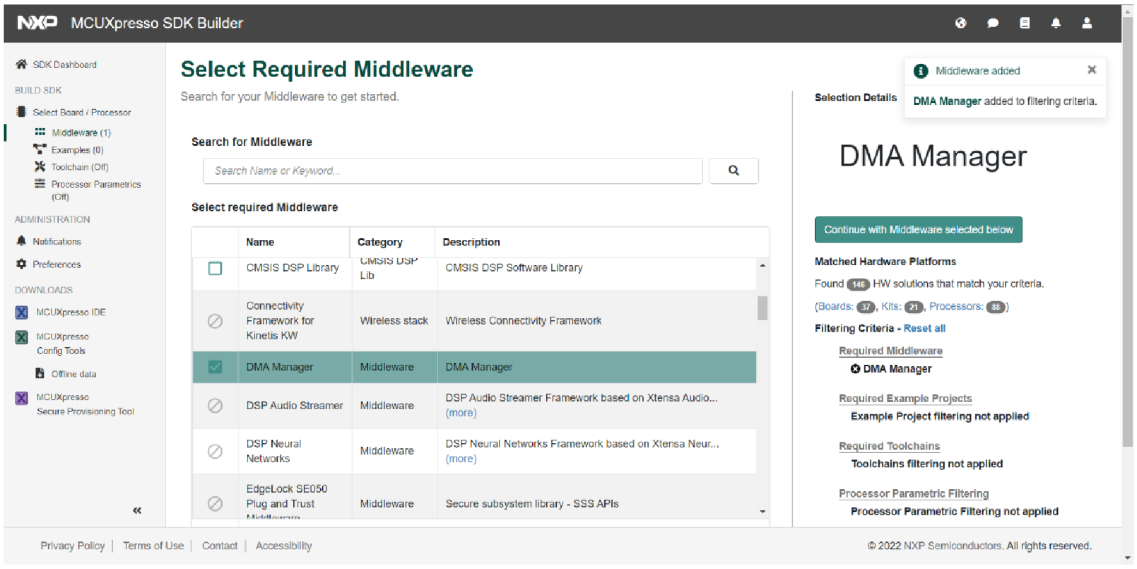
Obr. 11 Výber vývojovej dosky

Odkazy v červených častiach teda *Middleware, Examples, Toolchain* a *Processor Parametrics* majú rovnakú funkcionalitu a slúžia ako filtrovacie kritériá. Hardware môžeme buď to hľadať ručne, cez *Seach for Hardware* alebo môžeme náš výber zúžiť filtrovacími kritériami, alebo hľadaním konkrétneho hardware (*Board* (doska), *Kits* (súpravy), *Processors* (procesory), možné je hľadať aj *Deprecated*, teda zastaralé boards, kits a procesory), poprípade kombináciou oboch kedy aplikujeme rozličné filtre a následne si rozklikneme, aký konkrétny typ hardware hľadáme. Nad filtrovacími kritériami (Obr. 11) môžeme vidieť, koľko daných hardware riešení máme nájdených, vrátane rozpísania jednotlivých typov (board, kit, procesor). Aplikácia nám tiež ponúka odkaz na hľadanie hardware v prípade, že ho tu nenájdeme, odkaz sa nachádza v modrom rámčeku v dolnej strednej časti Obr. 11.. Začneme s nastavovaním filtrov, v tomto prípade začneme od vrchu, teda *Middleware*. Nie je to však nutné a filtre môžeme voliť v rozličnom poradí, pričom nemusia byť nastavené všetky filtre, iba tie ktoré chceme. Po zakliknutí *Required Middleware* v pravom červenom rámčeku na obrázku 8. sa nám zjaví nasledujúci pohľad (pri zakliknutí *Middleware* v rámčeku vľavo hore dostaneme rovnaký pohľad) viď obrázok 12.



Obr. 12 Pohľad po zakliknutí Required Middleware

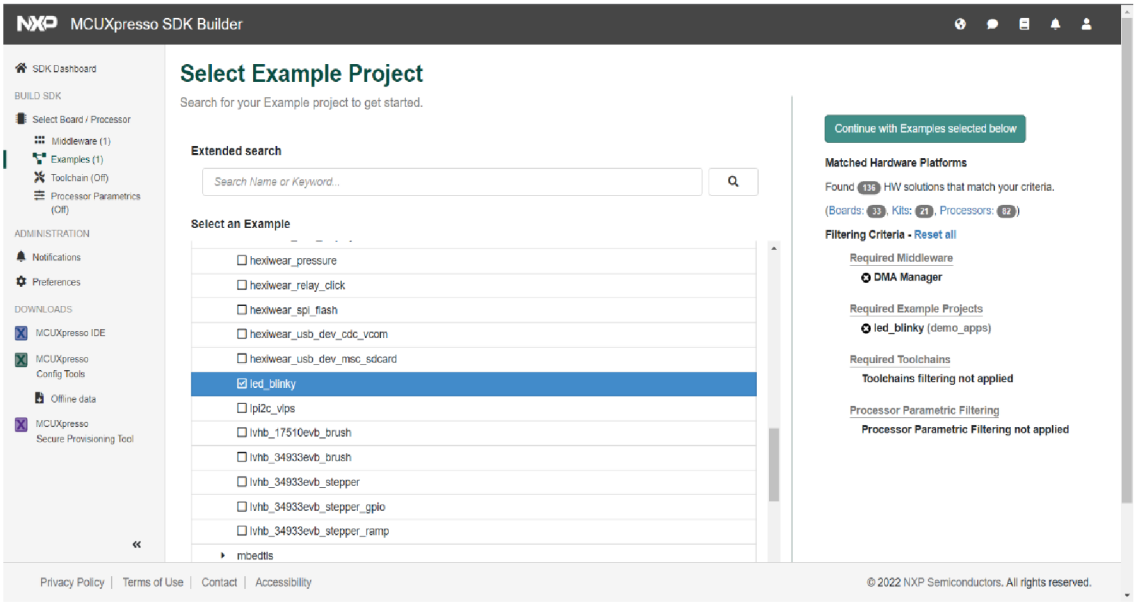
Ako môžeme vidieť na Obr. 12, ukáže sa nám zoznam *Middleware*. Ten, ktorý chceme, môžeme označiť a tým nám vyblednú možnosti nekompatabilné s naším označením (teda žiaden hardware neobsahuje tieto *Middleware* naraz). Zvolíme si napríklad *DMA Manager*, na obrázku 13 môžeme vidieť ako sa nám pohľad aplikácie zmenil, napríklad aj *Selection Details*.



Obr. 13 Vybratý Middleware

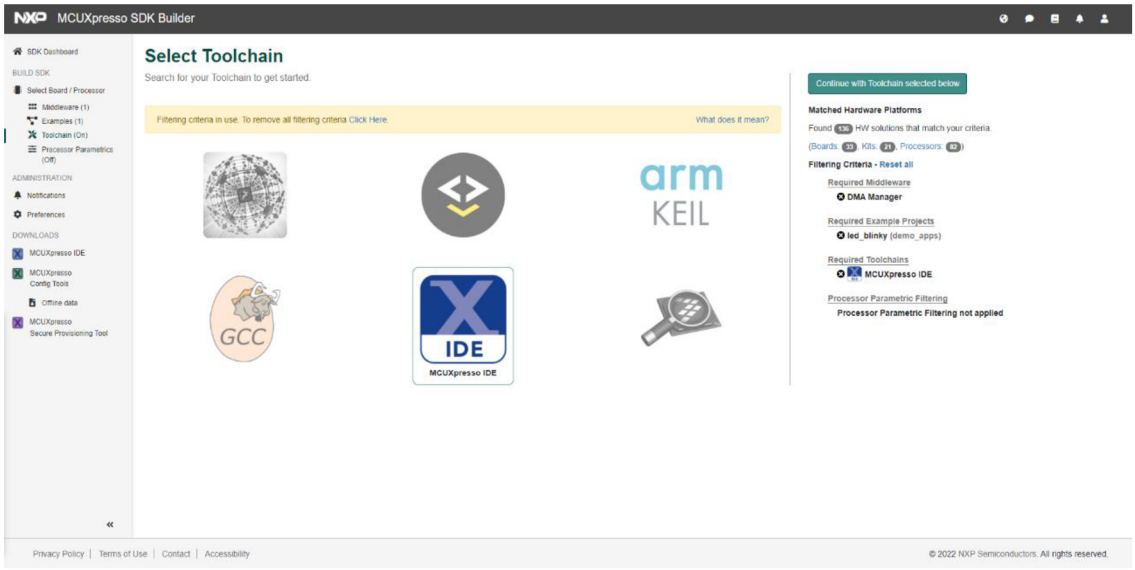
V pravom hornom rohu môžeme vidieť oznam vykonanej akcie, a teda že sme pridali *DMA Manager* ako filtračné kritérium. Ďalšia vec hodná povšimnutia je už vyššie spomenuté vyblednutie označenia *Middleware*, ktorý je s DMA Manager nekompatabilný. V *Selection Details* si môžeme všimnúť názov vybraného Middleware. Zmenil sa nám tiež celkový počet nájdených hardware riešení, ktorý klesol na 146 (odfiltrovali sme tie, ktoré neobsahujú *Middleware DMA Manager*). Ďalším filtrovacím kritériom v poradí je *Required Example Projects* (požadované ukážkové projekty). Toto

kritérium slúži na odfiltrovanie hardware, na ktorom nie sú požadované ukážkové projekty (alebo aj predpripravené projekty, ako napríklad blikanie LED a podobne). Example projektov je veľa, vyberieme náhodne už spomínaný *led_blinky* na blikanie LED. Samozrejme, hľadať môžeme opäť aj za pomoci textboxu (viď obrázok 14).



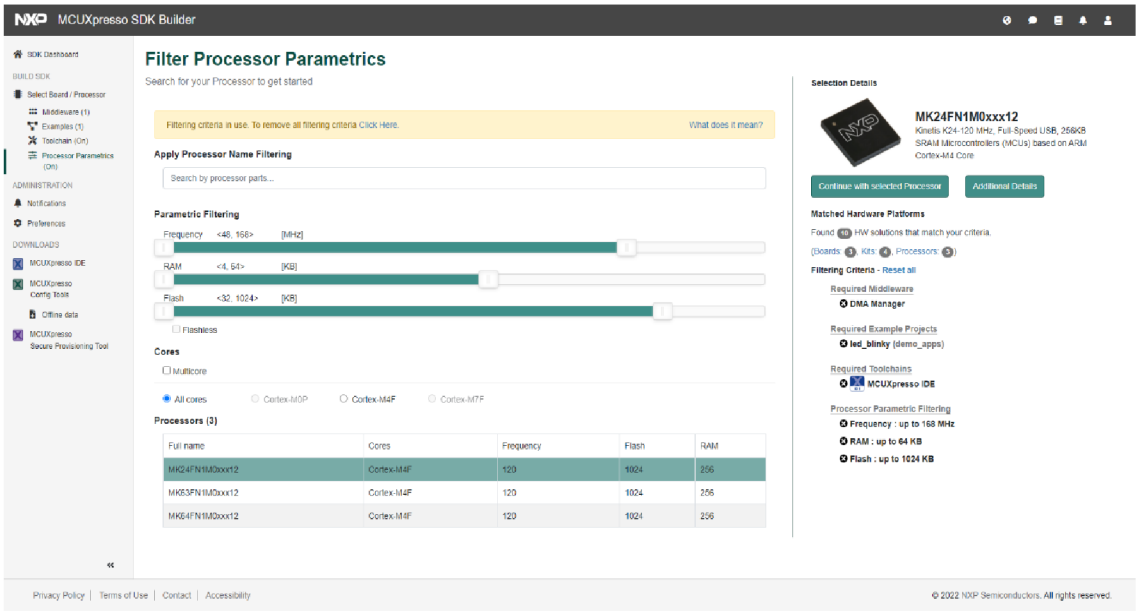
Obr. 14 Vybratý Example Project led_blinky

Pokračujeme filtrovacím kritériom *Required Toolchains* (Požadované krížové prekladače), kde v podstate vyberáme, s akým IDE má byť náš hardware kompatabilný. Na výber sú *Toolchains* Kinetis Design Studio, IAR Embedded Workbench for ARM, Keil MDK, GCC ARM Embedded, MCUXpresso IDE a CodeWarrior Development Studio. Toolchains, ktoré sú nekompatabilné s našimi predošlými filtrami, nebudú prístupné (na obrázku 15 môžeme vidieť, že ich farba vybledne). Pre tento príklad si vyberieme MCUXpresso IDE.



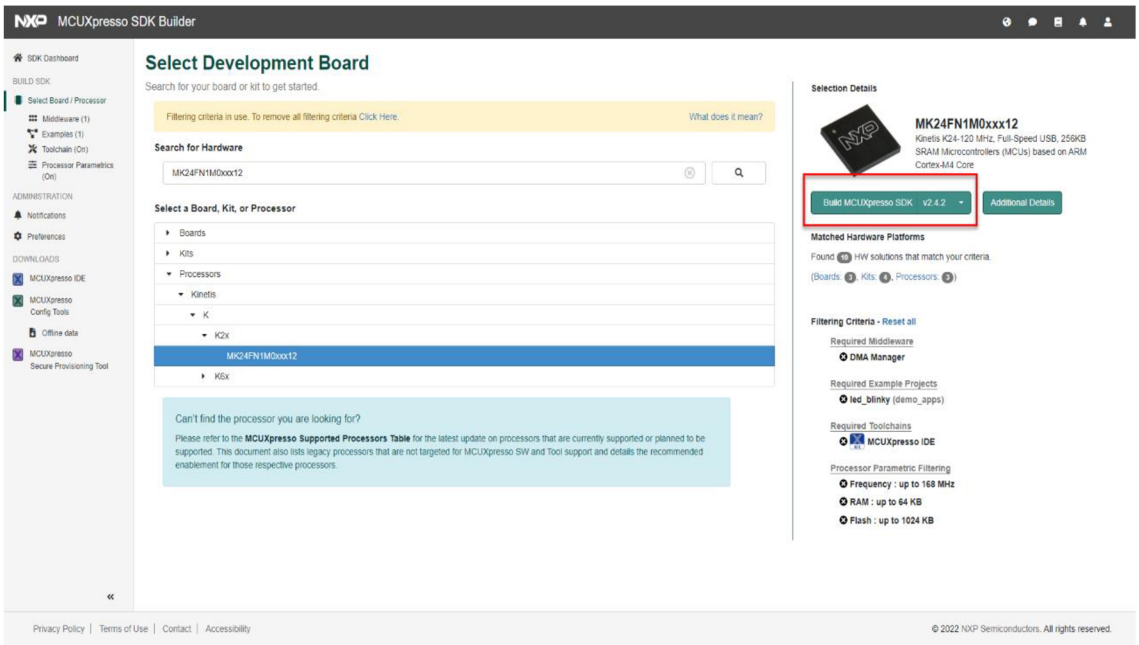
Obr. 15 Výber Toolchain

Posledným filtračným kritériom je *Processor Parametric Filtering*, kde môžeme vybrať parametre procesora ako jeho jadrá, frekvenciu, RAM, Flash pamäť či jadrá. V našom prípade, ktorý je čisto názorný, sme náhodne nastavili parametre a zvolili typ procesoru. Vidieť to môžeme na obrázku číslo 16.



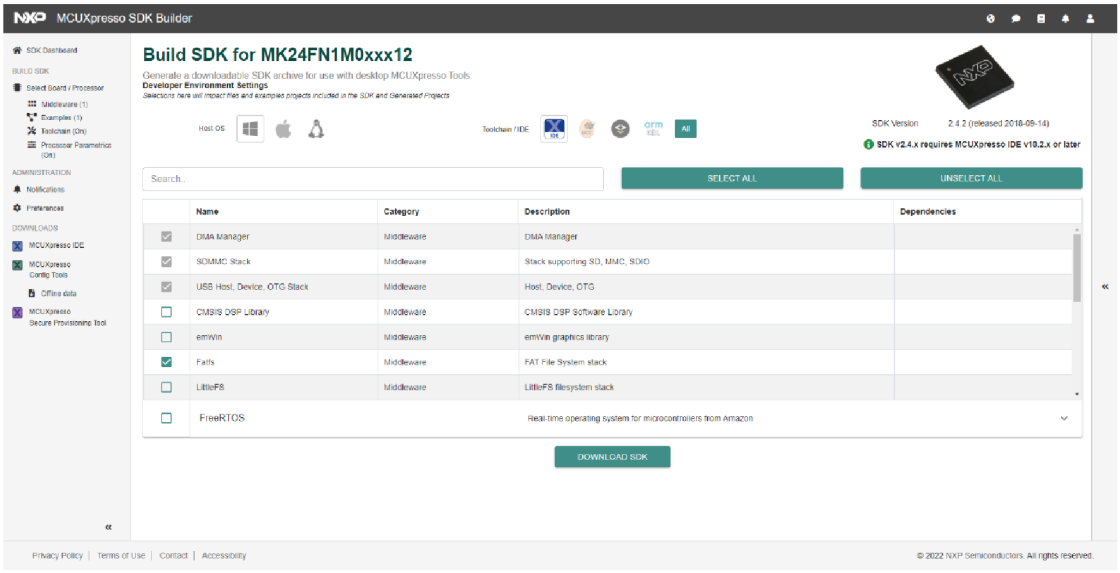
Obr. 16 Filtrovanie parametrov procesora

Po zvolení filtrovacích kritérii nám aplikácia nájde procesor, vyhovujúci naším požiadavkám. Samozrejme, keby sme nezvolili z ponúkaných procesorov, stále môžeme vybrať iné hardware riešenia ako sú boards alebo kits. Ponúknuté hardware riešenie môžeme vidieť na obrázku 17.



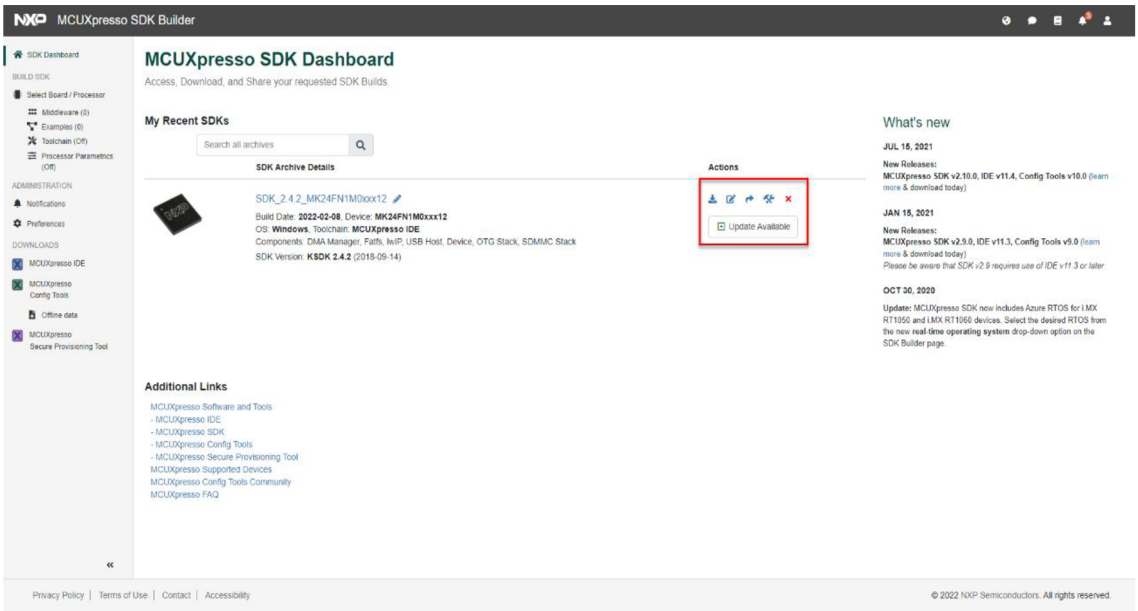
Obr. 17 Vybraný procesor, pre ktorý sa bude vytvárať SDK

Teraz už len ostáva stlačiť Build MCUXpresso SDK (viď Obr. 17) a vytvoriť SDK pre náš procesor. Šípkou si môžeme nastaviť verziu SDK. Dostaneme na výber ešte pridať *Middleware*, my už nič nepridáme a zvolíme *DOWNLOAD SDK* (viď obrázok 18).



Obr. 18 Posledné úpravy pred vystavaním SDK

Objaví sa správa, že výstavba SDK môže trvať niekoľko minút, v závislosti na komplexite konfigurácie. Potom, ako sa naše SDK vytvorí s ním môžeme robiť niekoľko akcii. Môžeme ho stiahnúť, prestavať, zdieľať, pristúpiť pomocou neho k SDK v MCUXpresso Config Tools alebo ho odstrániť z našich nedávno vytvorených SDK. Máme tiež na výber SDK updatnúť, ak je update prístupný. Vyššie zmienené náležitosti môžeme vidieť na obrázku 19.



Obr. 19 Vystavané (vybuildené) SDK pripravené na stiahnutie

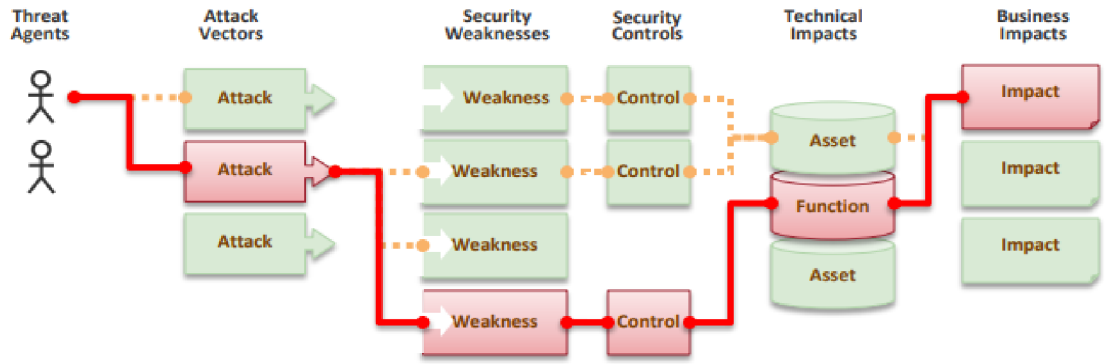
Po vystavaní SDK nám aplikácia pošle mail na e-mail, ktorým sa prihlasujeme ako užívatelia.

4. ZRANITEĽNOSTI, ÚTOKY A SLABINY WEBOVÝCH APLIKÁCIÍ

V aplikáciách existujú nedostatky, nazývané zraniteľnosti (vulnerabilities). Zraniteľnosti sú definované ako slabiny (weaknesses), ktoré je možné zneužiť. Vo všeobecnosti môžu byť zraniteľnosťami napríklad dizajnové vady, či implementačné bugy, dovoľujúce útočníkovi, aby napáchal škodu vlastníkom aplikácie. Príklady zraniteľností: [3]

- Nedostatok kontroly vstupu od užívateľa
- Neprítomnosť postačujúceho prihlasovacieho mechanizmu
- Otváranie chybových hlášok
- Nesprávne uzatváranie spojenia s databázou

Útoky (Attacks) sú techniky, ktoré používajú útočníci na zneužitie zraniteľností v aplikáciách a často bývajú zamieňané so zraniteľnosťami. Podstatou pri definovaní útoku je povedať kroky alebo postup, ktorý by útočník zvolil, zatiaľ čo pri zraniteľnostiach sa sústredíme na nedostatky v aplikácii. [3]



Obr. 20 Postup útoku [4]

Bezpečnostné riziká pre webové aplikácie popisuje dokument OWASP TOP 10. Na obrázku 21 môžeme vidieť zmenu v najčastejších kategóriách bezpečnostných rizík od roku 2017 do roku 2021. [5]

Obr. 21 OWASP TOP 10 2021 [5]

Riziká sú rozdelené do skupín, pričom každá skupina popisuje nejaké typy zraniteľností a s nimi spojené útoky. Budú prejdené postupne.

4.1 A01: Broken Access Control (Prelomená kontrola prístupu)

Táto skupina obsadila v roku 2021 prvé miesto ako najčastejšie bezpečnostné riziko. Zaraďujeme sem nasledujúce zraniteľnosti podľa CWE (Common Weakness Enumeration): [5]

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Vystavenie citlivých informácii neoprávnenému užívateľovi)
- CWE-201: Exposure of Sensitive Information Through Sent Data (Vystavenie citlivých informácii skrz poslané dáta)
- CWE-352: Cross-Site Request Forgery (Cross-site falzifikácia požiadavky)

Access Control (Kontrola prístupu) vyžaduje, aby užívatelia nekonali mimo ich povolení. Pri zlyhaní väčšinou dôjde k neautorizovanému odhaleniu informácii, modifikácii alebo zničeniu všetkých dát, poprípade k vykonaniu nejakej obchodnej funkcie mimo užívateľských právomocí. Zraniteľnosti bežnej kontroly prístupu sú napríklad: [5]

- Dovolenie prezerania alebo úpravy účtu niekoho iného za poskytnutia jeho jedinečného identifikátora (unique identifier).
- Zvýšenie privilégií, teda vystupovanie ako užívateľ bez prihlásenia alebo vystupovanie ako admin pri prihlásení ako užívateľ.
- Manipulácia metadát, ako zmena či prehrávanie JSON Web Token či Access control Token, poprípade zmena cookies, alebo nejaký Hidden field upravený na vyzdvihnutie privilégií.
- Force Browsing (Násilné prehľadávanie) na stránkach s overením ako neoverený užívateľ, alebo prístup k privilegovaným obsahom stránky ako štandardný užívateľ.
- Obídenie Access Control Check úpravou URL (Uniform Resource Locator) alebo použitím nejakého útočného nástroja na modifikovanie API (Application Programming Interface) Request.
- Prístup k API pomocou chýbajúcej kontroly prístupu pre POST, PUT a DELETE.
- Zlá konfigurácia CORS (Cross-Origin Resource Sharing), ktorá dovoľuje prístup k API z neoverených alebo neautorizovaných zdrojov.
- Porušenie princípu najnižšieho privilégia alebo deny by default, kedy by mal byť prístup povolený len pre určitých užívateľov alebo pozície, ale je prístupný pre všetkých.

Access Control je efektívne iba pri trusted server side code alebo server-less API, kde útočník nemôže modifikovať Access Control Check alebo Metadata. Broken Access Control môžeme zabrániť nasledujúcimi krokmi: [5]

- Okrem Public Resources (Verejných Zdrojov), deny by default.
- Zaznamenávanie Access Control zlyhaní, upozornenie adminov, keď je to nutné (napríklad opakované zlyhania).

- Implementovanie Access Control mechanizmov raz a opätovne ich použiť skrz aplikáciu, vrátane minimalizácie používania CORS.
- Limitne ohodnotiť API a prístup kontroléru na minimalizáciu poškodení spôsobených automatizovanými útočnými nástrojmi.
- Vypnutie vypisovanie priečinkov webového servera a zaistenie, aby sa súborové metadáta (ako .git) nenachádzali vo webových rootoch.
- Unikátne obchodné limitné požiadavky aplikácie by mali byť vynútené doménovými modelmi.
- Modelové Access Control by mali vynútiť zaznamenávanie vlastníctva namiesto toho, aby mohol užívateľ vytvárať, čítať, aktualizovať alebo zmazávať nejaké záznamy.

Príkladom útoku je napríklad situácia, kedy aplikácia používa neoverené dáta v SQL volaní. Toto volanie pristupuje k informáciám užívateľského účtu. Iným príkladom útoku je Force Browsing na cielenej URL, pričom na prístup k admin stránke sú potrebné admin práva. Príklad tohto je napríklad https://example.com/app/getappInfo alebo https://example.com/app/admin_getappInfo. [5]

4.2 A02: Cryptographic Failures (Kryptografické zlyhania)

Hlavným zameraním tejto skupiny rizík sú zlyhania v kryptografii, alebo jej absencia, čo vedie k úniku citlivých dát. Podľa CWE sem patria zraniteľnosti: [5]

- CWE-259: Use of Hard-Coded Password (Použitie pevne nakódovaného hesla)
- CWE-327: Broken or Risky Crypto Algorithm (Prelomený alebo riskantný šifrovací algoritmus)
- CWE-331: Insufficient Entropy (Nedostatočná entropia)

Najprv je potrebné zhodnotiť ochranné požiadavky dát v prenose a v príprave na prenos. Príkladmi dát sú napríklad heslá, čísla kreditných kariet, zdravotné záznamy, osobné informácie či firemné tajomstvá. Tieto dáta potrebujú väčšiu ochranu, hlavne z dôvodu, že môžu spadať pod zákony o ochrane súkromia ako napríklad GDPR (európske General Data Protection Regulation). Pre všetky tieto dáta by sme sa mali opýtať nasledujúce otázky: [5]

- Sú prenášané dáta v čistom texte? Toto sa týka protokolov ako http, SMTP, FTP ale tiež TLS. Platí, že external internet traffic (externá internetová komunikácia) je nebezpečná. Všetku vnútornú komunikáciu treba preverovať, napr. web servery alebo back-end systémy.
- Používajú sa nejaké zastaralé alebo slabé kryptografické algoritmy alebo protokoly?
- Sú využívané východzie kryptografické kľúče? Sú generované alebo opakovane používané slabé kryptografické kľúče, poprípade chýba rotácia kľúčov a tým aj ich správne manažovanie?
- Máme vynútené šifrovanie (encryption) ?
- Používame vyradené hash funkcie ako MD5 alebo SHA1? Používame nekryptografické hash funkcie tam, kde sú potrebné kryptografické hash funkcie?

- Sú kryptografické chybové hlášky alebo vedľajšie informácie kanálu zneužiteľné?
- Sú inicializované vektory ignorované, opakovane používané alebo negenerované dostatočne bezpečné pre kryptografický operačný mód?
- Je použitá náhodnosť pre kryptografické účely, ktorá sa nezhoduje s požiadavkami kryptografie?
- Sú heslá použité ako kryptografické kľúče pri absencii hesla kľúčovej derivačnej funkcie?

Na obranu proti tejto skupine rizík môžeme zvoliť nasledovné body: [5]

- Je potrebné klasifikovať dáta, ktoré sú spracúvané, ukladané alebo prenášané aplikáciou. Dôležitým krokom je identifikovať, aké dáta sú citlivé vzhľadom na zákony o súkromí, firemné požiadavky a tak podobne.
- Citlivé dáta by sa nemali ukladať, pokiaľ to nie je naozaj nutné. Pokiaľ je to možné, je potrebné sa ich zbaviť čo najrýchlejšie. Často platí, že údaje, ktoré sa neuchovávajú, nie je možné ukradnúť.
- Všetky citlivé dáta musia byť zašifrované.
- Musíme zaistiť silné a aktuálne štandardizované algoritmy, protokoly a kľúče. Nesmieme zabudnúť na správnu prácu a manažovanie kľúčov.
- Všetky dáta v prenose by mali byť šifrované pomocou zabezpečených protokolov ako je TLS with Forwad Secrecy (FS).
- Cachovanie odpovedí pri citlivých dátach by malo byť vypnuté.
- Pri klasifikácii dát by sme mali aplikovať potrebné bezpečnostné kontroly.
- Na prenos citlivých dát by sme nemali používať zastaralé protokoly ako FTP (File Transfer Protocol) alebo SMTP (Simple Mail Transport Protocol).
- Heslá by mali byť uložené za použitia silne adaptívnych hashovacích funkcii s faktorom oneskorenia ako Argon2, scrypt a podobne.
- Pre mód operácie musia byť zvolené odpovedajúce inicializačné vektory.
- Namiesto obyčajného šifrovania je lepšie použiť šifrovanie s overením.
- Kľúče by sa mali ukladať do pamäte ako byte polia a mali by byť generované kryptograficky náhodne.
- Kde je to vhodné, je potrebné zaistiť použitie kryptografickej náhodnosti.
- Vyhýbať sa používaniu starých kryptografických funkcii ako MD5, SHA1 a podobne.
- Osobitne overiť efektivitu konfigurácie a nastavenia.

Uveďme si príklad útoku. Aplikácia šifruje čísla kreditných kariet v databáze za použitia automatického šifrovania databáz. Avšak, tieto dáta sú automaticky dešifrované, keď sa načítajú a toto dovoľuje SQL Injection na získanie čísel kreditných kariet v čistom texte. [5]

4.3 A03: Injection (Injekcia)

Do tejto skupiny zaraďujeme nasledujúce zraniteľnosti: [5]

- CWE-79: Cross-Site Scripting
- CWE-89: SQL Injection
- CWE-73: External Control of File Name or Path (Externá kontrola názvu súboru alebo cesty)

Najprv si detailnejšie popíšeme zraniteľnosti Cross-Site Scripting a SQL Injection. Neskôr si uvedieme všeobecný popis a postupy, ako týmto zraniteľnostiam a útokom zabrániť.

4.3.1 Cross-site Scripting (XSS)

Vďaka rozsiahlej interakcii užívateľov s webovými aplikáciami v dnešnej dobe sú XSS zraniteľnosti častým problémom na internete. XSS útoky zneužívajú fakt, že webové aplikácie spúšťajú skripty na strane užívateľovho prehliadača. Ide teda v podstate o vloženie (injection) vlastného skriptového kódu útočníka (napríklad JavaScript alebo HTML) do zraniteľnej webovej aplikácie. Pri použití HTML sa útok tiež nazýva HTML Injection. Môžeme teda povedať, že akýkoľvek dynamicky vytvorený skript, ktorý je spustený, predstavuje riziko pre webové aplikácie, pokiaľ tento skript môže byť nejako modifikovaný (predovšetkým koncovým užívateľom). XSS útoku sa aplikácia môže vystaviť vždy, keď prijíma vstup od užívateľa a následne vstup užívateľovi spätne zobrazuje. [2]

Keby si teda zosumarizujeme informácie, môžeme popísať XSS útok nasledovnými bodmi: [2]

- Spustenie skriptu v prehliadači, ktorý nebol napísaný vlastníkom webovej aplikácie.
- Môže bežať v pozadí, neviditeľný a bez nutnosti užívateľského vstupu na jeho spustenie.
- Oplýva schopnosťou voľne posielať a získavať dáta zo škodlivého servera.
- Vyskytuje sa ako dôsledok nesprávne sanitovaného užívateľského vstupu zakomponovaného v užívateľskom rozhraní.
- XSS útoky môžu byť použité na krádež session tokenov (relačných tokenov), čím spôsobia prebratie účtu (napr. užívateľského).
- XSS útok je schopný vykresliť DOM (Document object model) cez užívateľské rozhranie (UI – user interface), čo vedie na perfektné phising útoky, ktoré technicky menej zdatný užívateľ nie je schopný rozpoznať.

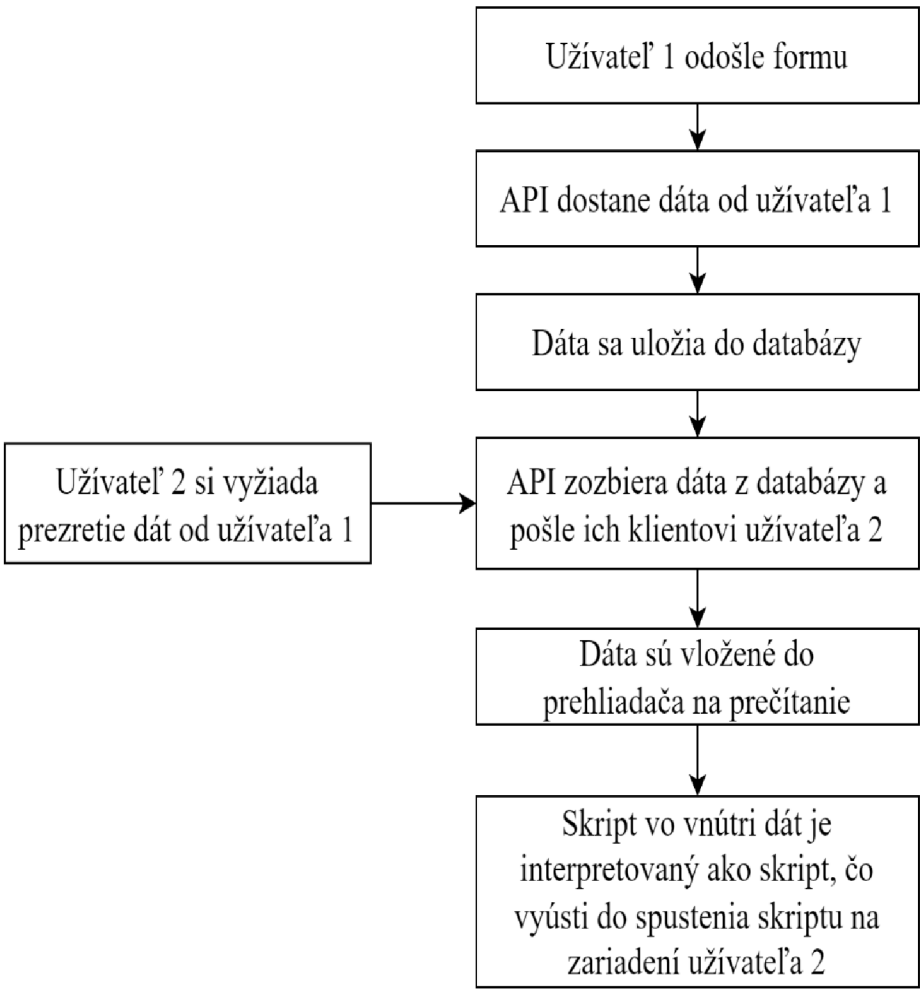
XSS útoky sa dajú rozdeliť do 3 najväčších skupín a to: [2]

- Stored (kód je uložený do databázy pred jeho spustením)
- Reflected (kód nie je uložený do databázy, ale je reflektovaný, čiže odrazený serverom)
- DOM-based (kód je aj uložený v databáze, aj spúšťaný v prehliadači)

Stored XSS (Uložené XSS)

Najbežnejší, najjednoduchší na rozpoznanie, ale často aj najškodlivejší útok kvôli tomu, že postihuje najviac užívateľov. Nazývaný tiež Persistent XSS. Môže napadnúť všetkých užívateľov aplikácie, pretože je uložený v databáze pomocou servera. Užívatelia sú napadnutí pri každom navštívení napadnutej aplikácie. Zraniteľnosti pre Stored XSS útok sú časté pri forum software, hlavne v často komentovaných sekciách a recenziách produktov. [2]

Na obrázku číslo 22 môžeme vidieť postup Stored XSS útoku:



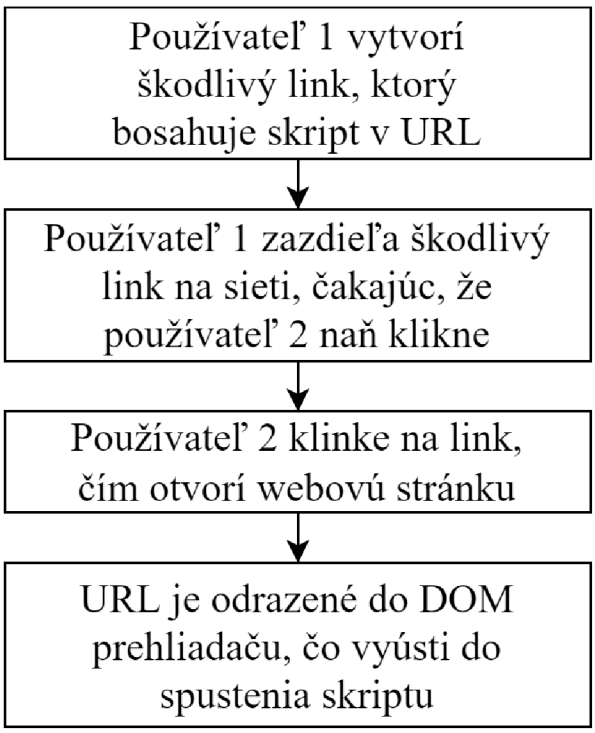
Obr. 22 Postup Stored XSS útoku [2]

Ako už bolo spomenuté, Stored XSS je ľahko zistiteľné a to kvôli ich permanentnej nature. Regulárne skenovanie databázových vstupov je jedna z lacných a efektívnych metód nájdenia uloženého skriptu na Stored XSS. [2]

Reflected XSS (Odrazené XSS)

Reflected XSS útoky nie sú uložené na serveri, ale sú vytvorené posielaním žiadostí (requests) spolu s XSS útokom samotným. Útok sa udeje, keď je užívateľov vstup zahrnutý v odpovedi servera, napríklad v chybových hláseniach či vo výsledkoch

hľadania. Server teda napríklad prečíta URL so zakomponovaným skriptom a pošle ho späť klientovi. [2]
Na obrázku 23 môžeme vidieť postup Reflected XSS útoku.

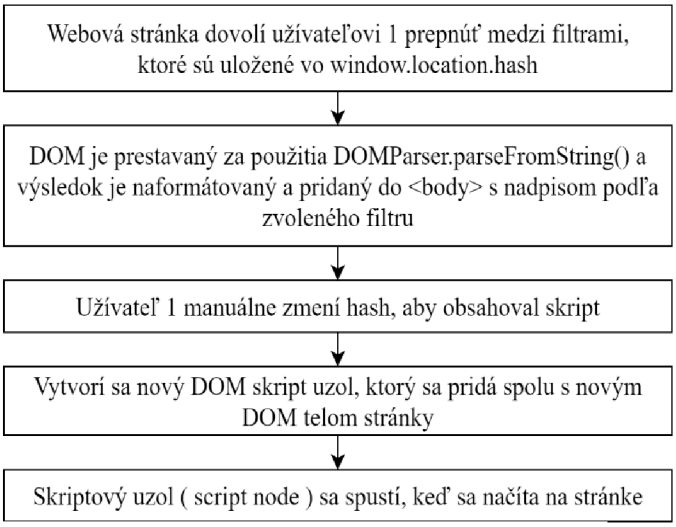


Obr. 23 Postup Reflected XSS útoku [2]

Keďže Reflected XSS útoky počítajú s tým, že užívateľ pošle požiadavku so zakomponovaným XSS útokom, je možné pri útoku uvažovať komponent sociálneho inžinierstva. V podstate môžeme povedať, že tento typ XSS útoku zvyšuje úspech formou sociálneho inžinierstva, pretože môžeme vytvoriť URL, ktoré sa bude chovať ako súčasť reálnej webovej stránky a použiť XSS útok k tomu, aby sme užívateľa presmerovali na škodlivú stránku. Zistiť zraniteľnosti aplikácie na XSS útok sa dá aj jednoduchou formou, kedy do URL cielenej webovej aplikácie pridáme napríklad tagy s úmyslom hrubo vyznačiť text v tagoch. Upravený dotaz teda môže vyzerať nasledovne:
Support.mega-bank.com/search?query=open+checking+account
Pokiaľ je zraniteľnosť prítomná, uvidíme na stránke hrubo vyznačený text, ktorý bol vybraný v URL pomocou query. [2]

DOM-Based XSS (XSS založené na DOM)
DOM XSS môžu byť buď Reflected alebo Stored, pričom v oboch prípadoch používajú DOM Sinks a Sources prehliadaču pre spustenie. Kvôli rozdielom v implementácii DOM jednotlivých prehliadačov môžu byť niektoré prehliadače zraniteľnejšie, zatiaľ čo iné nie. Tieto typy útokov sú ťažšie zistiteľné, ale sú tiež ťažšie využiteľné ako samotné Reflected a Stored XSS útoky, keďže vyžadujú hlbšie znalosti DOM prehliadača a JavaScriptu. [2]

Na obrázku číslo 24 je znázornený postup DOM-Based XSS útoku:

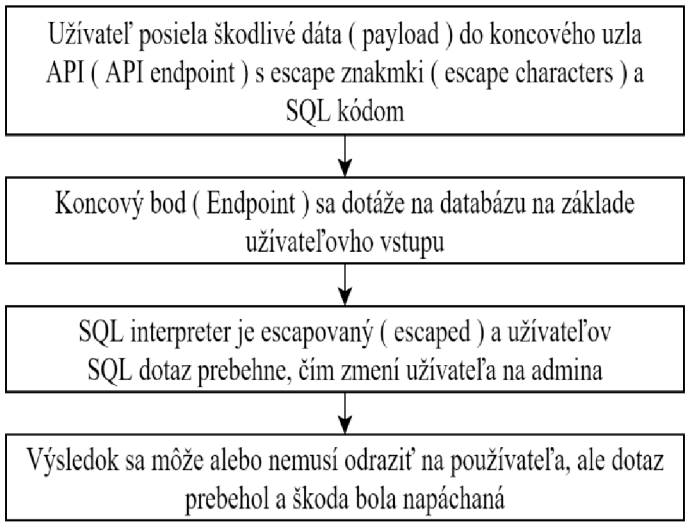


Obr. 24 Postup útoku DOM-Based XSS [2]

Hlavným rozdielom medzi predošlými typmi XSS je, že DOM-Based XSS útoky nepotrebujú interakciu so serverom. Preto ich môžeme zaradiť do podkategórie nazývanej Client-side XSS (teda XSS na strane klienta). Tieto útoky kvôli rozdielnosti prehliadačov môžu napríklad fungovať na prehliadači z roku 2015, ale novší, aktualizovaný prehliadač, už na nich náchylný byť nemusí. [2]

4.3.2 SQL Injection (SQL Injekcia)

Tento typ útoku využíva SQL (Structured Query Language). Je najčastejšie referencovanou formou útoku Injection kategórie. Ide o prenos SQL reťazca (string) v http dátovom obsahu, čo vedie k vykonaniu náhodných SQL dotazov v mene koncového užívateľa. Tieto útoky sú možné skrz vstup užívateľa (napríklad užívateľské meno a heslo), ktoré nie sú vhodne sanitované. [2]
Na obrázku číslo 25 je zobrazený postup útoku SQL injekciou:



Obr. 25 Postup útoku SQL Injection [2]

Vďaka striktnejšej štandardizácii jazykov ako je PHP sa počet útokov SQL Injection znížil. Tieto útoky vedia vyberať informácie z databáz alebo úplne prebrať kontrolu nad serverom. [2]

Ďalším typom Injection útoku je napríklad Code Injection vykonaný pomocou CLI (Command Line Interface), teda príkazového riadka. V CLI volanom pomocou API sa v dôsledku nevhodnej sanitácie vykonajú neočakávané príkazy. [2]

Aplikácia je teda náchylná k zraniteľnostiam a útokom Injection kategórie ak: [5]

- Dáta poskytnuté užívateľom nie sú filtrované, kontrolované alebo sanitované aplikáciou.
- Dynamické dotazy (queries) alebo neparametrizované volania bez Context Aware Escaping sú použité priamo v interpreteri.
- Nebezpečné dáta sú použité v prehľadávacích parametroch Object-Relational Mapping (ORM) na extrakciu citlivých záznamov.
- Nebezpečné dáta sú použité buď priamo alebo sú zreťazené. SQL alebo príkaz obsahuje štruktúru a škodlivé dáta v dynamických dotazoch, príkazoch alebo uložených procedúrach.

Jeden z najčastejších foriem injekcii (injections) sú napríklad SQL, NoSQL, OS command, Object Relational Mapping, LDAP a Expression Language (EL). Koncept je rovnaký pri všetkých interpreteroch. Revízia zdrojového kódu je najlepšou metódou zistenia, či je aplikácia náchylná na injekcie. Taktiež je silne odporúčané automatizované testovanie všetkých parametrov, hlavičiek (headers), URL, cookies, JSON, SOAP a XML dátových vstupov. [5]

Proti injekcii sa bránime tak, že držíme dáta oddelené od dotazov (querries) a príkazov (commands): [5]

- Používanie bezpečného API je preferovanou možnosťou, vďaka ktorej sa vyhneme použitiu interpretera a poskytne nám aj parametrizované rozhranie (interface). Pokiaľ však PL/SQL alebo T-SQL zreťazí dotazy a dáta, alebo spustí nebezpečné dáta pomocou EXECUTE IMMEDIATE alebo exec(), je SQL Injection stále spustiteľný aj keď máme parametrizované procedúry.
- Je žiadúce použiť pozitívne overenie vstupu na strane servera (server side input validation). Keďže ale veľa aplikácií potrebuje špeciálne charaktery ako textové plochy, a tiež API pre mobilné aplikácie, neslúži pozitívny server-side input validation ako kompletná ochrana.
- Pre akékoľvek reziduálne dynamické dotazy by sme sa mali vyhnúť špeciálnym charakterom za použitia špecifickej escape syntaxe pre daný interpreter. SQL štruktúram ako sú názvy tabuliek, názvy stĺpcov sa nie je možné vyhnúť a preto je užívateľský vstup názvov štruktúr nebezpečný.
- Použitie LIMIT a iných SQL Control v rámci dotazov na prevenciu masového úniku záznamov v prípade SQL Injection.

4.4 A04 Insecure Design (Nezabezpečený dizajn)

Nasledujúcou kategóriou v poradí je Nezabezpečený dizajn. Táto kategória sa zameriava na riziká spojené s dizajnom a architektonickými nedostatkami. Medzi CWE hodné spomenutia patria nasledujúce: [5]

- CWE-209: Generation of Error Message Containing Sensitive Information (Generovanie chybových hlásení obsahujúcich citlivé informácie)
- CWE-256: Unprotected Storage of Credentials (Nezabezpečené úložisko osobných dokladov)
- CWE-501: Trust Boundary Violation (Narušenie hraníc dôvery)
- CWE-522: Insufficiently Protected Credentials (Nedostatočne ochránené osobné informácie)

Nezabezpečený dizajn (Insecure design) je široká kategória popisujúca rozličné zraniteľnosti. Treba pochopiť, že existuje rozdiel medzi nezabezpečeným dizajnom (insecure design) a nezabezpečenou implementáciou (insecure implementation). Dizajnové nedostatky a implementačné chyby majú rozličné príčiny a nápravy, a preto ich treba rozlišovať. Zabezpečený dizajn môže mať napríklad stále implementačné chyby, ktoré vedú k zneužiteľným zraniteľnostiam. Nezabezpečený dizajn zasa nemôže byť napravený perfektnou implementáciou, teda potrebné bezpečnostné ochrany (Security Controls) neboli navrhnuté proti špecifickým útokom. Jedným z faktorov, ktoré prispievajú k nezabezpečenému dizajnu, je nedostatok profilovania firemných rizík zakotvených v software, a teda neschopnosť určiť, aká úroveň ochranného dizajnu je potrebná. [5]

Na analýzu rizík v tejto kategórii je potrebné uvážiť, ako veľmi bude aplikácia exponovaná a či je potrebná segregácia užívateľov. Musia sa spracovať technické požiadavky, tak isto ako funkčné a nefunkčné bezpečnostné požiadavky. Potrebné je tiež zhodnotenie firemných požiadaviek v rámci aplikácie, vrátane ochranných požiadaviek ako dôvernosť (confidentiality), integrita (integrity) a prístupnosť (availability). [5]

Zabezpečený dizajn (Secure Design) je v podstate metodológia vyhodnocujúca nebezpečenstvo a zaisťujúca, že kód je navrhnutý robustne a testovaný tak, aby predišiel známym útokom. Nie je to add-on ani nástroj, ktorý sa dá pridať do software. Aby bol dizajn skutočne zabezpečený, je potrebné analyzovať predpoklady a podmienky pre možné chyby toku. Výsledky by sa mali dokumentovať v user story. Je potrebné sa poučiť z chýb a navrhnúť vylepšenia. [5]

Útokom v tejto skupine sa môžeme vyhnúť nasledovne: [5]

- Vytvoriť a použiť zabezpečený vývojový životný cyklus (Secure Development Lifecycle) s AppSec (Application Security) profesionálmi, aby bolo možné lepšie vyhodnotiť a navrhnúť bezpečnosť a kontrolu súkromia.
- Vytvoriť a používať knižnicu zabezpečených dizajnových vzorov.
- Používať modelovanie nebezpečenstva pre kritické overenia (Critical Authentication), kontrolu prístupu (Access Control), obchodnú logiku (Buisness logic) a kľúčové toky.

- Integrovať bezpečnostný jazyk a kontroly do User Stories.
- Integrovať kontroly vierohodnosti v každom stupni aplikácie (od front-end do back-end).
- Napísať Unit a Integration testy na validáciu tak, že všetky kritické toky sú odolné voči modelu nebezpečenstva. Dobré je tiež zostaviť use-cases a misuse-cases pre každý stupeň aplikácie.
- Rozdeliť vrstvy stupňov v systéme a sieťových vrstvách v závislosti na vystavaní a potrebe ochrany.
- Robustne oddeliť užívateľov navrhovaním skrz všetky stupne.
- Obmedziť spotrebu zdrojov na užívateľa alebo servis.

Príkladom nezabezpečeného dizajnu je napríklad nasledovný scenár. Povedzme, že reťazec kín ponúka zľavy na rezerváciu skupín miest a je obmedzený na 15 účastníkov na rezerváciu pred tým, než vyžaduje vklad. Útočník túto vadu zistí pomocou modelovania útoku tak, že si otestuje či je možné rezervovať 600 miest vo viacerých kinách naraz za pomoci pár žiadostí, čím môže spôsobiť veľkú stratu zisku. [5]

4.5 A05: Security Misconfiguration (Chybná konfigurácia zabezpečenia)

Pri dnešnom často vysoko konfigurovateľnom software táto kategória stúpa vyššie každým rokom. Medzi CWE patria nasledovné zraniteľnosti: [5]

- CWE-16: Configuration (Konfigurácia)
- CWE-611: Improper Restriction of XML External Entity Reference (Nevhodné obmedzenie referencií externej entity XML)

Aplikácie sú zraniteľné útokmi z tejto skupiny pokiaľ: [5]

- Aplikácii chýba odpovedajúce Security Hardening (Kalenie) cez akúkoľvek časť Application Stack (spojenie viacerých software programov, ktoré spolupracujú na dosiahnutie rovnakého cieľa), alebo sú v aplikácii prítomné nesprávne nakonfigurované povolenia na cloud servisoch.
- Aplikácia má povolené alebo nainštalované nepotrebné funkcie (features), ako napríklad nepotrebné porty, služby, účty či privilégiá.
- Predvolené (default) účty a ich heslá sú stále povolené a nezmenené.
- Error Handling (Zaobchádzanie s chybami) v aplikácii odhalí priveľmi informatívne chybové hlášky užívateľom.
- Pre aktualizované systémy sú najnovšie bezpečnostné funkcie vypnuté, alebo nie sú bezpečne nakonfigurované.
- Bezpečnostné nastavenia na serveroch aplikácie, frameworkoch (aplikačných rámcoch) aplikácie (napr. ASP.NET), knižniciach alebo databázach nie sú nastavené na bezpečné hodnoty.
- Server neposiela Security Headers (bezpečnostné hlavičky) alebo direktívy, alebo tieto nie sú nastavené na bezpečné hodnoty.
- Software nie je aktuálny alebo je zraniteľný (podľa nadchádzajúcej kategórie A06).

Bez pevného a opakovateľného konfiguračného procesu pre bezpečnosť aplikácie sú systémy vo väčšom nebezpečenstve. Security Misconfiguration môžeme predísť implementáciou zabezpečených inštalačných procesov a to vrátane nasledujúcich opatrení: [5]

- Za použitia opakovaného hardening procesu (stvrdzovacieho procesu) sa uľahčí a urýchli nasadenie prostredia, ktoré je odpovedajúco uzamknuté. Prostredia (enviroments) vývoj, zabezpečovanie kvality (Quality Assurance) a produkciu by mali byť nakonfigurované identicky, s rôznymi osobnými informáciami (credentials) použitými v každom prostredí. Tento proces je žiadúce automatizovať, aby sa tak minimalizovalo úsilie pri nastavovaní nového zabezpečeného prostredia.
- Zaistená by mala byť minimálna platforma bez nepotrebných komponentov, prvkov, dokumentácie a vzoriek. Preto by malo byť zabránené nainštalovaniu nepotrebných prvkov a frameworkov, poprípade by mali byť odstránené, ak už sú nainštalované.
- Implementovaná by mala byť tiež úloha (task), ktorá preskúma a aktualizuje konfigurácie tak, aby to odpovedalo všetkým bezpečnostným poznámkam, aktualizáciám a patchom. Preskúmané by mali byť tiež povolenia cloudových úložísk (Cloud Storage Permissions).
- Skrz aplikáciu, ktorej architektúra je rozdelená (segmentovaná) a poskytuje efektívne a bezpečné oddelenie medzi komponentami a užívateľmi, či už to za pomoci segmentácie, kontajnerizácie (containerization, napr. pomocou Docker) alebo pomocou cloudových bezpečnostných skupín (Cloud Security Groups – ACL, alebo aj Access-control List).
- Posielaním bezpečnostných direktív klientom, napríklad bezpečnostné hlavičky (Security Headers).
- Vytvorením automatizovaného procesu, určeného na verifikáciu efektivity konfigurácie a nastavení vo všetkých prostrediach.

Príkladom situácie, kedy môže dôjsť k útoku, je napríklad nasledovný scenár. Vypisovanie adresárov (Directory Listing) nie je na serveri vypnuté, takže útočník zistí, že môže ľahko vypísať adresáre. Útočník následne nájde a stiahne skompilované Java triedy, ktoré dekompiluje a pomocou reverzného inžinierstva si môže prezrieť kód. Takýmto spôsobom útočník pomerne ľahko nájde značné nedostatky v kontrole prístupu v aplikácii. [5]

4.6 A06: Vurnerable and Outdated Components (Zraniteľné a neaktuálne komponenty)

Medzi bežné zraniteľnosti podľa CWE sem patrí: [5]

- CWE-1104: Use of Unmaintained Third-Party Components (Použitie neudržovaných komponent tretích strán)

Aplikácia je pravdepodobne zraniteľná, ak: [5]

- Nie je známa verzia všetkých používaných komponentov (aj na strane klienta aj na strane servera). Toto zahŕňa priamo používané komponenty, ale aj vnorené dependencie (dependencies).
- Software je zraniteľný, nepodporovaný alebo neaktuálny (out of date). Týka sa to operačného systému, web/application serverov, Database Management Systems (databázové riadiace systémy), API a všetky ich komponenty, Runtime Enviroments (behové prostredia) a knižnice.
- Absencia pravidelného hľadania zraniteľností a prikláňanie sa k bezpečnostným bulletinom (Security Bulletins) súvisiacich s používanými komponentami.
- Neupevnenie alebo neaktualizovanie základnej platformy (Underlying platform), frameworkov a dependencii. K tomuto problému dochádza v prostrediach, kde patching prebieha ako mesačná, alebo štvrťročná úloha ponechávajúc organizácie otvorené zbytočnému nebezpečenstvu odhalenia opravených zraniteľností.
- Software vývojári netestujú kompatibilitu aktualizovaných, vylepšených (upgraded), či patchovaných knižníc.
- Nie je zaistená bezpečnosť konfigurácie komponentov.

Proti týmto zraniteľnostiam sa dá brániť pomocou riadenia aktualizačného procesu (patch management process): [5]

- Je potrebné odstrániť nepoužívané dependencies, nepotrebné prvky, komponenty, súbory a dokumentáciu.
- Zaznamenávať verzie komponentov na strane klienta a na strane servera (napríklad frameworky a knižnice) a ich dependencie za použitia nástrojov ako verzií, napríklad OWASP Dependency Check alebo retire.js. Neustále monitorovať zdroje ako Common Vulnerability and Exposure (CVE) a National Vulnerability Database (NVD) kvôli zraniteľnostiam v komponentoch. Proces je možné automatizovať pomocou Software Composition Analysis nástrojov. Zaistiť si pravidelné emailové upozornenia pre bezpečnostné nedostatky v použitých komponentoch.
- Komponenty získavať iba z oficiálnych zdrojov cez zabezpečené odkazy. Lepšie je preferovať podpísané balíčky (packages), čím sa redukuje šanca zahrnutia modifikovanej alebo škodlivej komponenty.
- Sledovať knižnice a komponenty, ktoré nie sú udržiavané alebo nemajú vytvárané bezpečnostné aktualizácie pre staršie verzie. Ak aktualizácia nie je možná, je vhodné použiť virtuálnu aktualizáciu na sledovanie, detekciu či ochranu proti objaveniu tohto problému.

Každá organizácia by mala mať zaistený plán na monitorovanie, triedenie, aktualizáciu a konfiguračné zmeny pre celý život aplikácie alebo portfólia. Príkladom útočného scenáru môže byť napríklad situácia, kedy komponenty bežia s rovnakými privilégiami ako samotná aplikácia. To znamená, že akýkoľvek nedostatok v komponentoch môže mať veľmi vážny dopad na celú aplikáciu. Tieto nedostatky môžu byť neúmyselné (napríklad programovacia chyba), alebo zámerné (napríklad zadné vrátka v komponente). Medzi problémy so zistením zraniteľností komponent patrí napríklad aktualizácia IoT

(Internet of Things). IoT je často ťažké alebo nemožné aktualizovať, pričom dôležitosti aktualizácii sú tu veľmi veľké (napríklad pri biomedicínskych zariadeniach). [5]

4.7 A07: Identification and Authentication Failures (Identifikačné a autentizačné zlyhania)

CWE hodné spomenutia pri tejto kategórii sú napríklad: [5]

- CWE-297: Improper Validation of Certificate with Host Mismatch (Nesprávna validácia certifikátu s nezhodou hostiteľa)
- CWE-287: Improper Authentication (Nesprávne overenie)
- CWE-384: Session Fixation (Fixácia relácie)

Potvrdenie užívateľovej identity a overenie riadenia relácie (Session Management) sú kritické pri obrane proti útokom zameraným na overenie. Aplikácia môže obsahovať zraniteľnosti overenia ak: [5]

- Dovoľuje automatizované útoky ako napríklad Credential Stuffing (Napĺňanie osobnými údajmi), kedy má útočník prístupný list validných uživateľských mien a hesiel.
- Umožňuje Brute Force Attack (Útok hrubou silou), alebo iné automatizované útoky.
- Povolí predvolené, slabé alebo často známe heslá ako napríklad Heslo1, admin, admin1 a tak podobne.
- Používa slabý alebo neefektívny spôsob obnovenia osobných údajov (Credentials Recovery) a procesov zabudnutia hesla. Za takéto spôsoby sa napríklad považujú otázky založené na vedomosti (Knowledge-based answers).
- Využíva čisto textové, nedostatočne zašifrované alebo slabo zahashované úložíská hesiel.
- Neoplýva Multi-factor Authentication (mnohofaktorové overenie) alebo ho má neefektívne.
- Odhaluje identifikátor relácie (Session Identifier, Session ID) v URL.
- Využíva opätovné používanie identifikátoru relácie po úspešnom prihlásení.
- Nevyužíva správne zneplatnenie relačných identifikátorov. To znamená, že užívateľské relácie alebo tokeny overenia nie sú správne zneplatnené počas odhlásenia alebo dlhšej doby neaktivity.

Brániť sa proti problémom tejto skupiny je možné nasledovnými spôsobmi: [5]

- Tam, kde je to možné, použiť Multi-factor overenie na predídenie problémom ako Automated Credential Stuffing, Brute Force, či opätovnému používaniu už ukradnutých osobných údajov.
- Nenasadzovať predvolené osobné údaje, hlavne pre admin užívateľov.
- Vhodné je implementovať aspoň slabú kontrolu hesiel, ako napríklad testovanie nových či zmenených hesiel oproti 10 000 najhorším heslám z nejakého listu.
- Združiť dĺžku hesla, jeho zložitosť a rotačné postupy s guidlines 800-63B od National Institute of Standards and Technology (NIST), alebo s inými modernými, evidence-based postupmi pre heslá.

- Zaistenie spevnenia registrácie, obnovenia osobných údajov a API ciest (API pathways) proti Account Enumeration Attacks používaním rovnakých správ pre všetky možné výsledky.
- Limitovanie nesprávnych prihlásení, alebo zvyšovanie času medzi opätovnými prihláseniami pri nesprávnom prihlásení. Treba si však dať pozor, aby nedošlo k DoS (Denial of Service). Dobrým zvykom je zaznamenávať všetky zlyhania a upozorniť administrátora, keď dôjde k Credential Stuffing, Brute Force alebo inému útoku.
- Používanie vbudovaného relačného manažéra (Built-in Session Manager) na strane servera, ktorý generuje nový, náhodný relačný identifikátor so silnou entropiou po prihlásení. Relačné identifikátory by nemali byť v URL. Mali by byť bezpečne uložené a zneplatnené po odhlásení, čase nečinnosti a vypršaní času.

Príkladom útoku je napríklad už vyššie spomínaný Credential Stuffing, ktorý používa zoznam známych hesiel. Predpokladajme, že aplikácia nemá implementovanú ochranu proti automatizovanému nebezpečenstvu (Automated Threat), alebo proti Credential Stuffing. V tomto prípade môže byť aplikácia použitá ako overovač hesiel, kedy určí, či sú osobné údaje platné. [5]

4.8 A08: Software and Data integrity Failures (Zlyhania software a integrity dát)

Táto kategória sa zameriava na vytváranie predpokladov spojených so software aktualizáciami, kritickými dátami, CI/CD pipelines (Continuous Integration / Continuous Deployment alebo Continuous Delivery) bez overenia integrity. Do tejto skupiny podľa CWE patria: [5]

- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (Začlenenie funkcionality z neoverenej sféry kontroly)
- CWE-494: Download of Code Without Integrity Check (Stiahnutie kódu bez kontroly celistvosti)
- CWE-502: Deserialization of Untrusted Data (Deserializacia neoverených dát)

Chyby integrity software a dát súvisia s kódom a infraštruktúrou, ktoré nechránia proti narušeniam celistvosti. Príkladom je situácia, kedy je aplikácia závislá na zásuvných moduloch (Plugins), knižniciach alebo moduloch z neoverených zdrojov a repozitárov. Nezabezpečená CI/CD pipeline môže spôsobiť neautorizovaný vstup, škodlivý kód či ohrozenie systému. Mnohé dnešné aplikácie obsahujú autoaktualizačnú funkciu, kde sú aktualizácie stiahnuté bez dostatočnej overovacej integrity a sú aplikované na doposiaľ dôveryhodnú aplikáciu. Takto môžu útočníci nahrať ich potencionálnu aktualizáciu, ktorú následne distribuujú a spustia na všetkých inštaláciách. Iným príkladom je situácia, kedy sú objekty alebo dáta kódované alebo serializované do štruktúry, ktorú môže útočník vidieť a modifikovať. [5]

Na riešenie problémov so zraniteľnosťami a útokmi tejto skupiny môžeme urobiť nasledujúce kroky: [5]

- Použitie digitálnych podpisov alebo podobných mechanizmov na overenie, či sú software a dáta z očakávaného zdroja a neboli pozmenené.
- Zaistiť, že knižnice a dependencie, ako napríklad npm alebo Maven sú dôveryhodné repozitáre. Ak máme vysoko rizikový profil, je vhodné uvážiť vytvorenie vnútorného repozitára, ktorý je plne overený.
- Zaistiť, že Software Supply Chain tool (software nástroj pre zásobovací reťazec), ako napríklad OWASP Dependency Check alebo OWASP CycloneDx sú použité na overenie, či komponenty neobsahujú známe zraniteľnosti.
- Zaistiť, aby CI/CD pipeline mala vhodnú segregáciu, konfiguráciu a kontrolu prístupu na zaistenie integrity kódu prúdiaceho cez build (výstavbové) a deploy (nasadzovacie) procesy.
- Postarať sa, aby unsigned (nepodpísané) alebo nešifrované serializované dáta neboli posielané neovereným klientom bez nejakej formy kontroly integrity alebo digitálneho podpisu na rozpoznanie falšovania či prehrávania serializovaných dát.

Viacero domácich routerov, set-top boxov, firmware zariadení a iných neoverujú aktualizácie cez signed (podpísaný) firmware. Unsigned firmware je narastajúci cieľ pre útočníkov a predpokladá sa, že sa situácia iba zhorší. Toto je závažný problém, pretože veľakrát nie sú prítomné mechanizmy na nápravu chýb a ostáva iba napraviť nedostatky v budúcich verziách a čakať na vyradenie predošlých verzií. Príkladom útoku v tejto kategórii bol aj nedávno často skloňovaný útok na SolarWinds Orion. [5]

4.9 A09: Security Logging and Monitoring Failures (Bezpečnostné zaznamenávanie a monitorovanie zlyhaní)

Problémy so zabezpečeným zaznamenávaním (logging) a zlyhaniami monitorovania sú ťažké na otestovanie. Ich testovanie často zahŕňa otázky, či bol útok zaznamenaný počas penetračného testu. Táto kategória obsahuje podľa CWE: [5]

- CWE-778: Insufficient Logging (Nedostatočné zaznamenávanie)
- CWE-117: Improper Outpu Neutralization for Logs (Nesprávna neutralizácia výstupu pre záznamy)
- CWE-223: Omission of Security -relevant Information (Vypustenie zabezpečenia relevantných informácii)
- CWE-532: Insertion of Sensitive Information into Log File (Vkladanie zabezpečených informácii do záznamových súborov)

Popis tejto kategórie slúži na pomoc s detekciou, eskaláciou a odpoveďou na aktívne prelomenia (Active Breaches). Bez zaznamenávania a monitorovania nie je možné detegovať prelomenie (Breach). Nedostatočné zaznamenávanie, detekcia, monitorovanie a nedostatočná aktívna odpoveď sa môžu vyskytnúť v prípadoch ako: [5]

- Auditovateľné udalosti, ako napríklad prihlásenia, zlyhania prihlásenia a transakcie s vysokou hodnotou nie sú zaznamenávané.

- Varovania a chyby generujú neadekvátne alebo nejasné zaznamenávacie správy, alebo ich negenerujú vôbec.
- Záznamy aplikácií a API nie sú monitorované na podozrivú aktivitu.
- Záznamy sú uložené iba lokálne.
- Odpovedajúce varovné prahové hodnoty (Alerting Tresholds) a procesy eskalácie odozvy sú neefektívne alebo neexistujúce.
- Penetračné testy a skeny pomocou Dynamic Application Security Testing (DAST, dynamické bezpečnostné testovanie aplikácií), nástrojov (ako OWASP ZAP) nespôsobujú upozornenia.
- Aplikácia nedokáže rozpoznať, eskalovať alebo upozorniť na aktívne útoky v reálnom čase alebo v takmer reálnom čase.

Aj proti útokom a zraniteľnostiam tejto kategórie sa dá brániť. Vývojár by mal implementovať niektoré, alebo najlepšie všetky nasledujúce kontroly, v závislosti na riziku, ktorému sa aplikácia vystavuje:

- Zabezpečiť, aby boli všetky prihlásenia, kontroly prístupu a validácie zlyhaní vstupu na strane servera zaznamenávané s dostatočným kontextom o užívateľovi, pre možnú identifikáciu podozrivých či zlomyseľných účtov, a tiež aby boli tieto udržané na tak dlhú dobu, aby sa stihla spraviť detailná forenzná analýza.
- Zaistiť generovanie všetkých záznamov vo formáte, ktorý je schopný ľahko spracovať riadenie zaznamenávania.
- Uistiť sa o tom, že sú dáta zakódované správne na prevenciu injekcii alebo útokov na zaznamenávacie alebo monitorovacie systémy.
- Presvedčiť sa, že transakcie s vysokou hodnotou majú kontrolu priebehu (Audit Trail) s kontrolou integrity na prevenciu falšovania či vymazávanie. Príkladom sú napríklad append-only databázové tabuľky.
- DevSecOps tímy by mali byť schopné zaistiť efektívne monitorovanie a upozorňovanie tak, aby boli všetky podozrivé aktivity rozpoznané a vyriešené čo najrýchlejšie.
- Zriadiť alebo prevziať plán odpovede na incidenty a obnovu (Incident Response and Recovery plan), napríklad podľa National Institute of Standards and Technology 800-61r2.

Existujú komerčné ale aj open-source frameworky na ochranu (Open-Source Application Protection Frameworks) ako OWASP ModSecurity Core Rule Set. Medzi open-source zaznamenávací korelačný software patrí napríklad Elasticsearch alebo Logstash. [5]

Príkladom útoku v tejto kategórii je nasledujúci scenár. Operátor stránky na poskytovanie zdravotného plánu pre deti nebol schopný rozpoznať narušenie (breach) kvôli nedostatku monitorovania a zaznamenávania. Externá skupina informovala poskytovateľa zdravotných plánov, že útočník modifikoval tisíce citlivých zdravotných záznamov vyše 3 a pol milióna detí. Následná revízia incidentu ukázala, že vývojári stránky nezhodnotili závažné zraniteľnosti. Keďže nebol prítomný žiadny zaznamenávací alebo monitorovací systém, narušenie sa mohlo odohrávať od roku 2013, teda v období vyše 7 rokov. [5]

4.10 A10: Server-Side Request Forgery (SSRF, v preklade Zneužitie požiadavky na strane servera)

Momentálne podľa OWASP nie sú väčšie CWE kategórie na priradenie. SSRF problémy sa vyskytujú všade, kde webová aplikácia preberá (fetching) vzdialené zdroje (Remote Resources) bez overenia URL poskytnutého užívateľom. Toto dovoľuje útočníkovi donútiť aplikáciu, aby poslala žiadosti (requests) do neočakávaných cieľov, aj keď je chránená firewallom, VPN alebo iným typom ACL (Access Control List). Keďže moderné webové aplikácie poskytujú užívateľom vhodné funkcie, preberanie URL sa stáva bežným scenárom. Výsledkom je, že sa incidencia SSRF zvyšuje. Ďalším dôvodom zvyšovania výskytu SSRF je čím ďalej tým častejšie používanie cloud servisov a komplexita architektúr. [5]

Útokom a zraniteľnostiam z tejto skupiny sa môžeme vyhnúť implementáciou nasledujúcich ochranných hĺbkových kontrol (Defense in Depth Control). Postupovať môžeme z: [5]

Sieťovej vrstvy (Network layer)

- Rozdeliť fukncionalitu prístupu vzdialených zdrojov (Remote Resource Access Functionality) na oddelenie siete za úmyslom redukovať dopad SSRF.
- Vynútiť deny by default pre firewall stratégie (policies) alebo vynútiť pravidlá kontroly prístupu siete (Network Access Control Rules) na zablokovanie všetkej prevádzky okrem nevyhnutnej intranetovej. Dobré je tiež založiť vlastníctvo (Ownership) a životnú dobu (Lifecycle) pre pravidlá firewallu (Firewall Rules) založených na aplikáciách. Ďalšou užitočnou technikou je zaznamenávať všetky prijaté a blokované toky sietí firewallom.

Aplikačnej vrstvy (Application layer)

- Sanitácia a validácia všetkých vstupných dát poskytnutých klientom.
- Vynútiť URL schéma, port a destináciu s pozitívnym prípustným listom (Allow List).
- Pre frontend s dedikovateľnými a riadiacimi užívateľskými skupinami používať šifrovanie siete (Network Encryption, napríklad VPN) na nezávislých systémoch.

Ako už bolo spomenuté, SSRF útoky môžu útočníci použiť na napadnutie systémov ochránených pomocou Web Application firewallov, firewallov alebo sieťových ACL, napríklad za použitia nasledujúceho scenára, kedy máme port sken interných serverov. Ak architektúra siete nie je segmentovaná, útočník môže zmapovať internú sieť a zistiť, či sú porty otvorené alebo uzavreté na interných serveroch pomocou výsledkov pripojenia (Connection Results) alebo uplynutého času na pripojenie alebo odmietnutie SSRF Payload pripojení (SSRF Payload Connections). [5]

Ďalšími kategóriami, ktoré sa nedostali do OWASP TOP 10, ale sú tiež patrične dôležité sú Code Quality Issues, Denial of Service a Memory Management Errors.

4.11 Code Quality issues (Problémy s kvalitou kódu)

Tieto problémy zahŕňajú známe bezpečnostné nedostatky alebo vzory, opätovne používané premenné na viac účelov, odhalenie citlivých informácii pri výstupe debuggeru, chyby konverzie typov signed alebo unsigned a iné. Tieto nedostatky môžu byť často identifikované pomocou prísnych kompilátorových vlajok, nástrojmi statickej analýzy kódu a pluginmi linter IDE. Mnoho moderných jazykov väčšinu týchto nedostatok eliminovalo. [5]

Zabrániť im môžeme tak, že povolíme statickú analýzu kódu v našom editore. Vhodné je tiež použitie nástroja na statickú analýzu kódu. Ďalšou možnosťou ako týmto problémom zabrániť je používanie jazykov alebo frameworkov, ktoré sú voči ich nedostatkom imúnne ako Rust alebo Go. [5]

Príkladom útoku je situácia, kedy sa útočník pokúsi získať citlivé informácie zneužitím race condition (súbehu) za použitia staticky zdieľanej premennej skrz viacero vlákien. [11]

4.12 Denial of Service (Odopretie služby)

Odopretie služieb je možné vždy, pokiaľ sú dostupné dostatočné zdroje. Avšak návrh a správne programovacie postupy vedia dopad DoS značne znížiť. [5]

Brániť sa proti odopretiu služby môžeme kódom na testovanie výkonnosti procesora, vstupov/výstupov, použitia pamäte alebo optimalizácie. Je potrebné zvážiť kontrolu prístupu pre veľké objekty a tým zaistiť, že k veľkým súborom majú prístup len overený užívatelia. [5]

Ako útok môžeme považovať situáciu, kedy útočník zistí, že určitá operácia trvá 5-10 sekúnd. Keď sú použité 4 súbežné vlákna, tak server prestáva odpovedať. Útočník použije 1000 vlákien naraz, a tak zhodí celý systém offline. [5]

4.13 Memory management errors (Problémy s riadením pamäte)

Webové aplikácie bývajú písane v jazykoch a frameworkoch, ktoré majú riadenú pamäť, ako napríklad Java, C#, JavaScript, TypeScript a podobne. Avšak tieto jazyky samotné sú písané v systémových jazykoch, ktoré môžu mať problém s riadením pamäte, ako napríklad buffer overflow (pretečenie vyrovnávacej pamäte) alebo heap overflow (pretečenie na halde). [5]

Obrana proti týmto chybám je v dnešnej dobe realizovaná pomocou moderných API, písaných v pamäťovo bezpečných (memory safe) jazykoch ako je Rust alebo Go. Pre už existujúci kód je prospešné napríklad fuzz testovanie na identifikáciu pretečenia pamäte a polí, statická analýza kódu a silné typovanie. [5]

Scenárom útoku môže byť pretečenie vyrovnávacej pamäte alebo pretečenie na halde. Útočník posiela dáta do programu, ktorý si ich ukladá do poddimenzovanej vyrovnávacej pamäte. Výsledkom je prepísanie informácii v zásobníku, vrátane návratového ukazovateľa funkcie. Dáta potom nastavia hodnotu návratového ukazovateľa tak, aby keď funkcia vrátila hodnotu, tak prenesie kontrolu na škodlivý kód obsiahnutý v útočníkových dátach. [5]

Popísať skupiny zraniteľností je v práci dôležité z dôvodu, že tieto zraniteľnosti sú základom techník útokov, ktoré môžu aplikáciu ohroziť. Navyše sa výstupy mnohých nástrojov analýzy webovej bezpečnosti odvolávajú práve na skupiny zraniteľností podľa OWASP, alebo rovno na jednotlivé slabiny v CWE, či na zraniteľnosti v CVE. Posledným dôvodom je fakt, že pred tým, než aplikáciu začneme analyzovať, mali by sme mať rozumný prehľad, čo sa snažíme nájsť a čo narúša jej bezpečnosť.

5.0 TECHNIKY A NÁSTROJE NA ANALÝZU WEBOVEJ BEZPEČNOSTI

V predošlej kapitole sme si popísali najčastejšie zraniteľnosti, slabiny a nedostatky webových aplikácií. Tieto zraniteľnosti, slabiny a nedostatky dokážeme a chceme nájsť. Pomocou hľadania / zisťovania zraniteľností (vulnerability scanning) vieme riziká pre webovú aplikáciu nájsť a pomocou vyhodnotenia zraniteľností (vulnerability assesment) vieme riziká ohodnotiť, podľa toho, ako veľmi sú nebezpečné. Pravidelné vyhodnotenie zraniteľností je dôležité pre zaistenie bezpečnosti webovej aplikácie. Vieme ním zistiť aj nedostatky ako sú chýbajúce aktualizácie (patches), zastaralé protokoly, certifikáty a služby a mnohé iné. [3, 6]

Penetračné testy (Penetration tests) v stručnosti slúžia na zneužitie zraniteľností v aplikáciách (legálne, ako etický hacker), aby sme tak zistili, do akej miery je útočník schopný zraniteľnosti zneužiť a k čomu tým dostane prístup. [3]
Tieto a ďalšie techniky si bližšie popíšeme v tejto kapitole.
Testovací model pozostáva z: [3]

- Testera, teda toho, kto vykonáva testovacie aktivity
- Nástrojov a metodológie (postupu)
- Aplikácie

Než si popíšeme nástroje a techniky, je dobré si v stručnosti popísať pár termínov, s ktorými sa dá pri testovaní bezpečnosti webových aplikácií stretnúť.

Black box testovanie
Etický hacker nemá prístup k žiadnym informáciám, okrem toho, čo ide testovať. Je potrebná kompletná identifikácia aplikácie a systému. [6]

Grey box testovanie
Etický hacker má prístup k základným informáciám, ako prítomnosť IDS / IPS (Intrusion Detection System / Intrusion Prevention System, v preklade systémy na prevenciu a detekciu narušenia), operačné systémy, ciele, značky vstavaných zariadení. Je to najčastejší typ testovania. [6]

White box testovanie
Etický hacker má prístup ku všetkým detailom ako sú IP adresy, dizajn infraštruktúry, čo je na sieti a ako to pracuje, dokonca možné slabiny systému. [6]

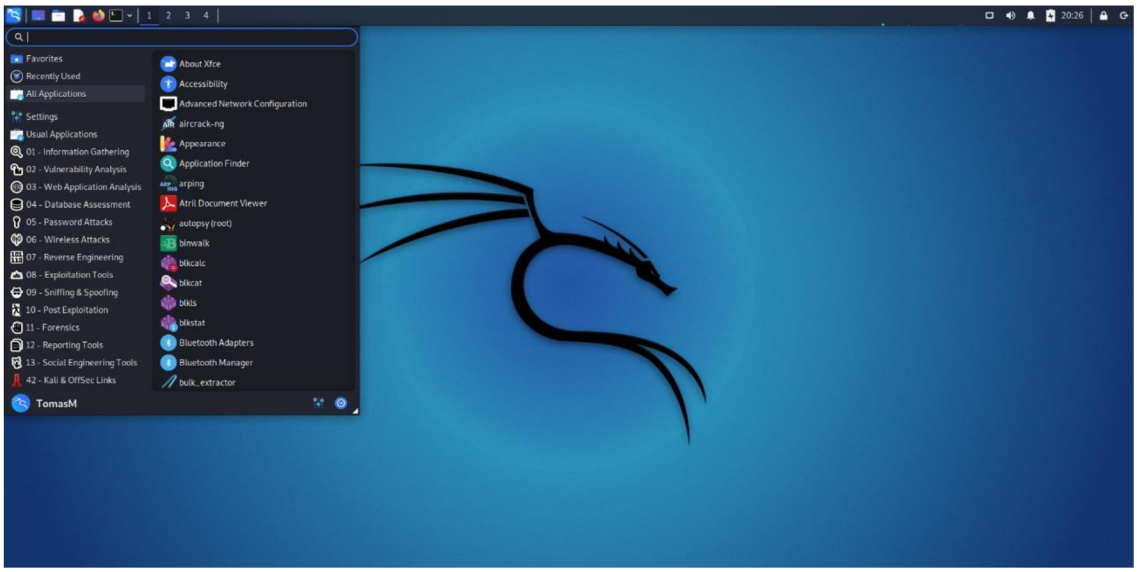
Red Teams (Červené tímy)
Sú vnútorné alebo externé entity, ktoré slúžia na testovanie efektívnosti bezpečnosti programu napodobňovaním útočníkov nástrojmi alebo manuálnym spôsobom. Ich práca je značne podobná, no nie identická s penetračným testovaním. Je možné sa aj stretnúť

s tvrdením, že červené tímy postupujú rovnako ako black hats hackeri, no s dobrými úmyslami. [7]

Blue Teams (Modré tímy)
Je interný bezpečnostný tím, ktorý sa bráni proti skutočným útočníkom, ale aj proti red teams. Sú v nepretržitej pohotovosti obrany proti útokom, čo ich odlišuje od štandardných bezpečnostných tímov organizácii. [7]

5.1 Kali Linux

Na testovanie aplikácií nám v tejto práci poslúži distribúcia operačného systému Linux s názvom Kali. Kali Linux je založený na linuxovej distribúcii Debian. Je špeciálne určený na informačnú bezpečnosť, penetračné testy, vyhodnocovanie a skenovanie zraniteľností a podobne. Na tieto účely má predinštalované rozličné nástroje, pričom niektoré budú v tejto práci použité. Nástroje je možné, samozrejme, doinštalovať podľa potreby. Práca s Kali Linuxom bola realizovaná za použitia VirtualBox. Na obrázku číslo 26 môžeme vidieť výpis skupín nástrojov slúžiacich na testovanie bezpečnosti aplikácií. Na pozadí je možné povšimnúť si logo Kali Linux.



Obr. 26 Kali Linux

Okrem použitia automatizovaných nástrojov je možné si napísať aj vlastné nástroje, napríklad v jazyku Python. V týchto nástrojoch sa často stáva, že aj sami voláme niektoré automatizované nástroje. Iný spôsob je manuálne testovať webové aplikácie. Ako sa neskôr dozvieme, manuálne testovanie má svoje výhody aj nevýhody a niektoré techniky sú naň odkázané viac ako iné. Manuálne testy sú sada postupov a príkazov, ktoré používajú aj automatizované nástroje. Pri testovaní sa dajú použiť aj webové aplikácie na to určené, ako napríklad Damn Vulnerable Application, vulnweb.com a mnohé iné. Tieto aplikácie majú zámerne viaceré nedostatky, chyby a zraniteľnosti, aby sa na nich mohli etickí hackeri učiť pracovať s nástrojmi a vykonávať penetračné testy. Tam, kde to bude

možné, sa bude práca sústrediť predovšetkým na automatizované nástroje, pretože jedným z jej cieľov je automatizovať proces testovania bezpečnosti aplikácie.

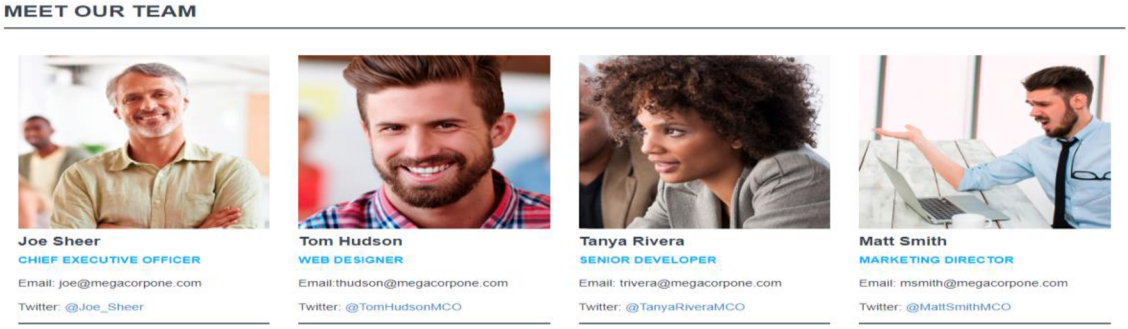
5.2 Získavanie informácii (Information gathering)

Prvá technika, ktorú preskúmame, je získavanie informácii. Nazýva sa tiež prieskum (reconnaissance) alebo fingerprinting (používa sa aj footprinting). Jedná sa o získavanie verejne dostupných informácii, informácii o zamestnancoch (napr. osobné informácie, pracovné pozície a podobne), ale aj zisťovanie sieťových blokov, identifikácia hostiteľov (hosts), významných firemných lokalít, typov serverov, používaných technológii a iných užitočných vecí. Touto technikou vieme potencionálne aj vyfiltrovať použiteľné informácie, ktoré môžeme ďalej použiť v iných technikách. Získavanie informácii sa delí na pasívne a aktívne. [1, 3]

5.2.1 Pasívne získavanie informácii

Pasívne získavanie informácii je proces získavania voľne prístupných informácii ohľadom cieľa bez priameho kontaktu s cieľom samotným. Používame nejaký medzičlánok, aby sme k nemu pristúpili. [3]

Prehľadávanie webovej stránky (Website recon)
Najbežnejším pasívnym získavaním informácii je prehľadávanie webovej stránky klienta/cieľu. Vieme pri ňom získať veľa informácii. Menšie spoločnosti majú väčšinou jednu stránku, väčšie môžu mať stránok viac, pričom niektoré z nich nemusia byť udržiavané. Zo stránky cieľu vieme takto zistiť napríklad mená a emaily tímu, čo nám aj umožní nájsť si členov tímu na sociálnych sieťach (Facebook, Twitter, LinkedIn) a zistiť ďalšie informácie, ktoré sa dajú použiť pri sociálnom hackerstve. Pokiaľ napríklad zistíme, že má niekto z členov tímu veľa CISCO certifikátov, je možné predpokladať, že firma, v ktorej pracuje, používa CISCO zariadenia. Na obrázku 27 nižšie môžeme vidieť potencionálne užitočné informácie. [3]



Obr. 27 Zneužiteľné informácie

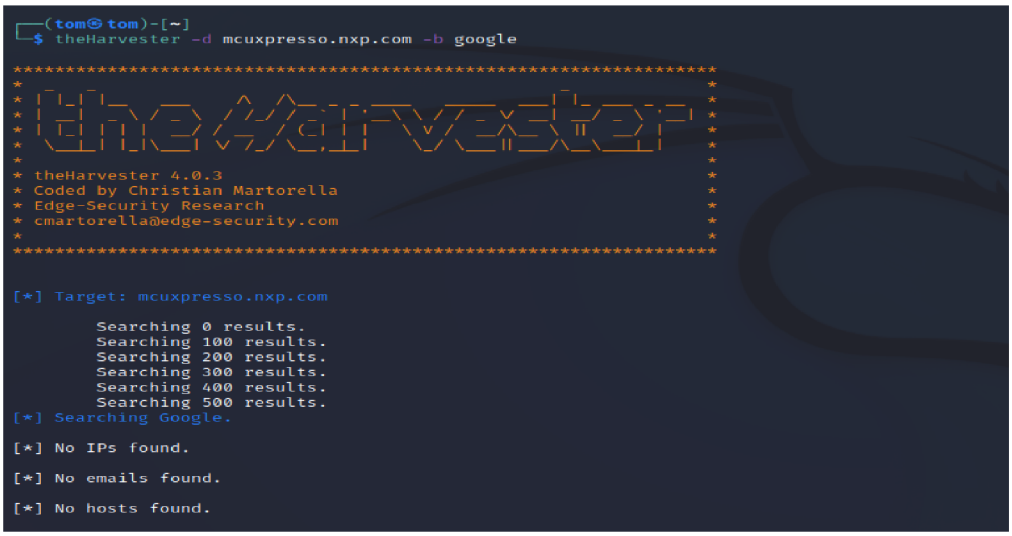
Okrem osobných informácii môžeme zo stránky zistiť aj používané technológie. Koncovky URL nám môžu odhaliť, aké programovacie jazyky stránka používa ako napr.

php alebo .do či .jsp. Ďalším populárnym spôsobom prehľadávania webovej stránky je prezeranie si zdrojového kódu stránky. Pri každom prehliadači je mierne odlišný. V kapitole 2. tejto práce získavame informácie o aplikácii ako užívateľ. [3]

Získavanie informácii o užívateľoch (User information gathering)

Získavanie informácii o užívateľoch je v podstate aj prehľadávanie webovej stránky, o ktorom sa písalo vyššie. Zahŕňa však aj niektoré iné techniky či nástroje, hodné spomenutia. Cieľom takéhoto získavania informácii je zistiť údaje na vytvorenie zoznamov hesiel a užívateľských mien, a tak získať základ pre útok credential stuffing (prepĺňanie osobnými údajmi). Vďaka týmto údajom je možné vykonať vylepšenú verziu techniky nazývanej phising, takzvaný vylepšený phising a celkovo nám tieto získané údaje umožnia vykonávať silnejšie a efektívnejšie útoky na strane klienta. V neposlednom rade nám tieto informácie uľahčia už vyššie spomínané útoky pomocou sociálnych sietí. Na získanie emailov (email harvesting) môžeme použiť nástroj *theHarvester*. Ten okrem emailov vyhľadáva mená a s nimi spojené IP adresy, subdomény, či URL z verejných dátových zdrojov. Iné možnosti na získavanie informácii o užívateľoch zahŕňajú napríklad *Password Dumps*, obsahujúce výpis často používaných hesiel. [3, 6]

Po spustení *theHarvester* nástroja na doménu mcuxpresso.nxp.com aplikácie MCUXpresso SDK Web Builder vidíme, že stránka neobsahuje žiadne vyššie spomínané informácie, ktoré by sa dali získať a boli by užitočné. Možnosť *-d* udáva cieľ nástroja a možnosť *-b* udáva zdroj dát.



Obr. 28 Výstup nástroja *theHarvester*

Ďalej sa dostávame k nástrojom na analýzu a prehľadávanie sociálnych sietí. *Social-Searcher* je vyhľadávací engine, pomocou ktorého dokážeme zistiť, aké sociálne siete daná spoločnosť preferuje. Potom môžeme prejsť na nástroje špecifické pre danú sociálnu sieť. *Social-Searcher* je platený, ale je možné si zaregistrovať účet zadarmo s obmedzeným počtom prehľadávaní za deň. Nástroje špecifické pre určité sociálne siete sú napríklad *Twofi* a *linkedin2username*. *Twofi* zoskenuje príspevky užívateľa na Twitteri

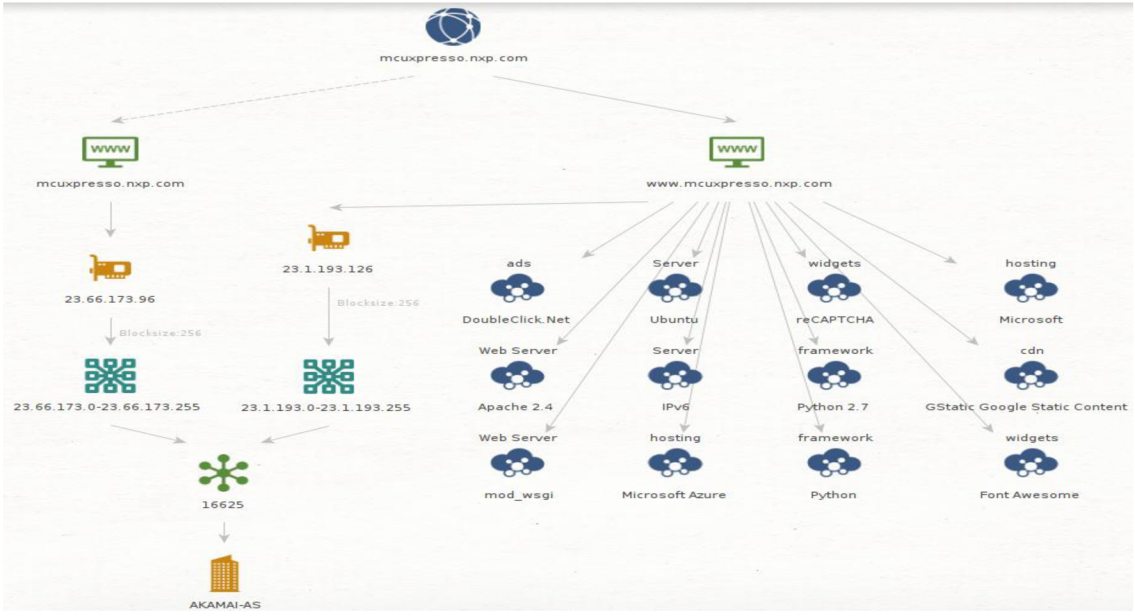
a na ich základe vygeneruje list hesiel, ktorý je možné použiť na útok. *Twofi* potrebuje na správne fungovanie platný Twitter API kľúč. Na sieť LinkedIn môžeme použiť nástroj *linkedin2username*. Ten generuje užívateľské mená založené na dátach z tejto siete. Tento nástroj však potrebuje platné užívateľské údaje a LinkedIn spojenie s užívateľmi, od ktorých chceme dáta získať. Do tejto kategórie získavania informácii sa zaraďuje aj sociálne inžinierstvo / hackerstvo (social engineering / hacking). [3, 6]

Poslednou technikou získavania informácii o užívateľoch je prehľadávanie Stack Overflow. Stack Overflow je stránka, kde sa vývojári a iní technici pýtajú alebo odpovedajú na rôzne problémy spojené s programovaním, sieťami, informatikou a podobne. Pokiaľ sa dá zistiť, pre akú spoločnosť užívateľ pracuje, vieme podľa jeho otázok a odpovedí určiť, aké technológie pravdepodobne táto spoločnosť využíva.

Frameworky na získavanie informácii

Rozsiahlym zdrojom dát je *OSINT Framework*, ponúkajúci informácie a odkazy na bezplatné (alebo aspoň s bezplatnou edíciou) nástroje a zdroje nie len pasívneho, ale aj aktívneho získavania informácii, vyhodnocovania zraniteľností a mnohých iných techník. Grafickým analytickým nástrojom *OSINT Framework* je napríklad *Maltego*. Prehľadáva tisícky online dát a používa prepracované „transformácie“ na konvertovanie jednej informácie na inú. Príkladom takejto transformácie je, že po zadaní emailovej adresy vykoná automatické vyhľadávania a túto adresu pretransformuje na odpovedajúce telefónne číslo alebo číslo ulice. [3, 6]

Maltego je platený nástroj, ale je možné použiť jeho bezplatnú verziu *Maltego Community Edition*. Okrem manuálneho pridávania a vypĺňania blokov a ich následných transformácii môžeme pustiť aj automatické získavanie dát. Na obrázku číslo 29 môžeme vidieť vygenerovaný graf po fingerprintingu úrovne L2. Sú tri úrovne, L1, L2 a L3, pričom každá vyššia úroveň poskytuje viac informácii (avšak je aj časovo náročnejšia na vykonanie).



Obr. 29 Graf informácii vygenerovaný pomocou *Maltego*

Google Hacking

Hackovanie pomocou *Google* je technika, kedy s jeho pomocou odhalíme nejaké kritické informácie či zle nakonfigurované stránky. Používame pri tom vyhľadávacie reťazce (search strings) a operátory, umožňujúce filtrovať nezaujímavé výsledky. Bežné operátory prehľadávania sú napríklad *site:* (použijeme ako site:owasp.org), *inurl:*, *intitle:*, *intext:* alebo *inbody:* a *filetype:*. Užitočný operátor je aj *cache:*, ktorý môže ukazovať už zmenený alebo zneprístupnený obsah od poslednej indexácie tohto obsahu. Je možné používať aj iné prehľadávacie nástroje ako *Bing* či *Shodan*. *Shodan* okrem webových stránok zahŕňa aj IoT zariadenia či routery, v podstate hľadá na internete zariadenia, zatiaľ čo napríklad *Google* hľadá obsah webového serveru. *Google Hacking Database* (GHDB) obsahuje viaceré kreatívne vyhľadávania, na ktorých je možné vidieť silu tejto techniky. [1, 6]

Netcraft

Na pasívne získavanie informácii môžeme použiť aj internetovú spoločnosť *Netcraft*. Zaujímavá je napríklad ich DNS seach page (Vyhľadávacia stránka DNS), ktorou vieme získať rozličné užitočné informácie, ako typ web servera a jeho operačného systému, IPv4 a IPv6 adresy, poskytovateľa sieťových služieb a mnohé iné. Na obrázku číslo 30 môžeme vidieť výstup *Netcraft* DNS search na webovú aplikáciu MCUXpresso Web SDK Builder. [6]

Background

Site title	Welcome | MCUXpresso SDK Builder	Date first seen	November 2016
Site rank	874771	Netcraft Risk Rating	1/10
Description	The MCUXpresso SDK brings open source drivers, middleware, and reference example applications to speed your software development.	Primary language	English

Network

Site	http://mcuxpresso.nxp.com	Domain	nxp.com
Netblock Owner	Akamai Technologies	Nameserver	ns1.nxp.net
Hosting company	Akamai Technologies	Domain registrar	corporatedomains.com
Hosting country	EU	Nameserver organisation	whois.corporatedomains.com
IPv4 address	95.101.251.36 (VirusTotal)	Organisation	NXP B.V., High Tech Campus, Eindhoven, 5656 AG, NL
IPv4 autonomous systems	AS16625	DNS admin	ip@domains.nxp.com
IPv6 address	2a02:26f0:9d00:191:0:0:0:1acc	Top Level Domain	Commercial entities (.com)
IPv6 autonomous systems	AS20940	DNS Security Extensions	unknown
Reverse DNS	a95-101-251-36.deploy.static.akamaitechnologies.com		

IP delegation

IPv4 address (95.101.251.36)

Obr. 30 Výstup Netcraft (IPv4, IPv6, DNS informácie)

Ďalšie významné informácie, ktoré sme získali, sú už spomínaný typ webového servera a jeho operačný systém. Môžeme ich vidieť na obrázku 31. Vďaka týmto informáciám môže útočník zamerať svoje útoky na konkrétny typ webového servera a operačného systému. Okrem už vyššie spomínaných dát nám *Netcraft* vráti napríklad aj rozsah IPv6 a IPv4 adries.

Hosting History

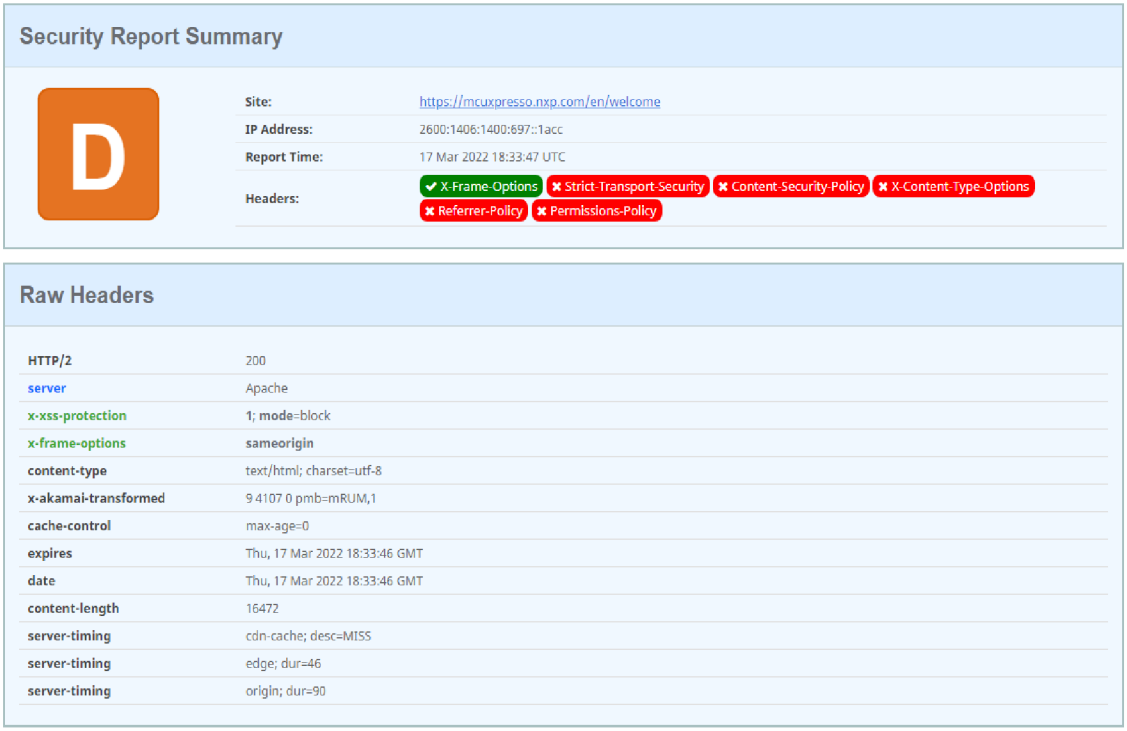
Netblock owner	IP address	OS	Web server	Last seen
Akamai Technologies, Inc. 145 Broadway Cambridge MA US 02142	23.204.227.21	Linux	Apache	15-Mar-2022

Obr. 31 Informácie o serveri a jeho operačnom systéme

Využiť môžeme aj webovú stránku *Exploit Database* (exploit-db.com), kde sa dajú vyhľadať zraniteľnosti a slabiny technológii, nájdených pomocou *Netcraft*.

Security Headers Scanner

Skener Bezpečnostných hlavičiek nám analyzuje http response headers (v preklade odpoveďové hlavičky http). Vieme ním zistiť bezpečnostné a programovacie praktiky spoločnosti. Na obrázku číslo 32 môžeme vidieť výstup, spolu s ohodnotením skenu.



Security Report Summary

D

Site:	https://mcuxpresso.nxp.com/en/welcome
IP Address:	2600:1406:1400:697::1acc
Report Time:	17 Mar 2022 18:33:47 UTC
Headers:	✔ X-Frame-Options ✖ Strict-Transport-Security ✖ Content-Security-Policy ✖ X-Content-Type-Options ✖ Referrer-Policy ✖ Permissions-Policy

Raw Headers

HTTP/2	200
server	Apache
x-xss-protection	1; mode=block
x-frame-options	sameorigin
content-type	text/html; charset=utf-8
x-akamai-transformed	9 4107 0 pmb=mRUM,1
cache-control	max-age=0
expires	Thu, 17 Mar 2022 18:33:46 GMT
date	Thu, 17 Mar 2022 18:33:46 GMT
content-length	16472
server-timing	cdn-cache; desc=MISS
server-timing	edge; dur=46
server-timing	origin; dur=90

Obr. 32 Výstup skenu Security Headers

Podľa výsledku skenu chýbajú niektoré defensive headers (v preklade obranné hlavičky). Tieto chýbajúce hlavičky nemusia nutne znamenať zraniteľnosť alebo slabinu, ale môžu naznačovať nedostatok server hardening.

SSL Server Test

Táto stránka analyzuje SSL/TLS konfigurácie a porovnáva ich s aktuálnymi odporúčanými postupmi. Vie tiež identifikovať niektoré SSL/TLS zraniteľnosti ako Heartbleed alebo Poodle. Na obrázku 33 nižšie sa dá povšimnúť výstup SSL Server testu pre mcuxpresso.nxp.com, spolu s ohodnotením ochrany.

SSL Report: mcuxpresso.nxp.com

	Server	Test time	Grade
1	23.59.197.178 a23-59-197-178.deploy.static.akamaitechnologies.com Ready	Thu, 17 Mar 2022 18:36:47 UTC **Duration:** 44.511 sec	A
2	2600:1407:5800:586:0:0:0:1acc g2600-1407-5800-0586-0000-0000-0000-1acc.deploy.static.akamaitechnologies.com Ready	Thu, 17 Mar 2022 18:37:31 UTC **Duration:** 55.692 sec	A
3	2600:1407:5800:596:0:0:0:1acc g2600-1407-5800-0596-0000-0000-0000-1acc.deploy.static.akamaitechnologies.com Ready	Thu, 17 Mar 2022 18:38:27 UTC **Duration:** 55.743 sec	A

Obr. 33 Výstup SSL Server testu

Ďalšie možné služby, použiteľné na pasívne získavanie informácii, je napríklad stránka *host.io*, ktorá získava údaje o doménach a subdoménach a stránka *hunter.io*, slúžiaca na vyhľadávanie emailov a osobných informácií (podobne ako spomínaný *theHarvester*).

5.2.2 Aktívne získavanie informácii

Aktívne získavanie informácii používa priamy postup pri kontakte s cieľom. Zahŕňa vytvorenie spojenia medzi naším strojom a cieleným systémom a sieťou. Dokážeme ním zistiť špecifické dáta ako živých hostiteľov (live hosts), bežiace služby, verzie aplikácií, súbory zdieľané na sieti a informácie o užívateľských účtoch. Živí hostitelia nám poskytnú informáciu, koľko zariadení je práve online. Znalosť operačného systému a bežiace služby nám pomôžu určiť úlohu daného zariadenia v sieti a to, aké zdroje klientom poskytuje. Napríklad pokiaľ zistíme, že na cielenom systéme sa nachádza veľa zdieľaných súborov, môže to znamenať, že tento systém je súborovým serverom (file server). [3, 6]

Pri aktívnom získavaní informácii stroj útočníka posiela špeciálne dotazy potencionálnemu cieľu, pričom útočník dúfa, že stroj cieľa odpovie prostredníctvom nejakých dôverných informácii. Pri aktívnom získavaní informácii hrozí riziko odhalenia cieľom. [6]

DNS enumerácia

DNS enumerácia je technika prehľadávania špecifických DNS záznamov na nájdenie domény cielenej organizácie. Každá doména používa iné typy DNS záznamov, medzi najznámejšie patria NS (Nameserver Records), A (Host Record), MX (Mail Exchange Records), PTR (Pointer Records), CNAME (Canonical Name Records) a TXT (Text Records). Vďaka množstvu informácii obsiahnutých v DNS je táto technika často používaná. Najbežnejšie vyhľadávanie IP adries je pomocou linuxového príkazu *host*. Použitie napríklad ako host -t mx mcuxpresso.nxp.com, kde -t slúži na špecifikovanie

záznamu, ktorý hľadáme, v tomto prípade MX. Príkaz *host* je možné použiť aj pri prehľadávaní hrubou silou, kedy príkazu poskytneme zoznam možných názvov hostiteľov (hostnames). Tiež je možné tento príkaz použiť reverzne, teda poskytnutím IP adresy a vyhľadaním doménového mena. Iným linuxovým príkazom s podobnou funkciou je *nslookup*. [1, 6]

V rámci tejto techniky prehľadávania je možné použiť DNS zone transfer, čo je v podstate databázová replikácia združeného DNS serveru, kedy sa zone file prekopíruje z master DNS serveru na slave server. Zone file obsahuje zoznam všetkých DNS mien nakonfigurovaných pre danú zónu. Pokiaľ administrátor zle nakonfiguruje jeho DNS server (napríklad tak, že kópiu DNS zone servera dostane každý, kto si o ňu požiada) tak hacker dostane prístup k celému rozloženiu siete danej spoločnosti. [1, 6]

Na DNS enumeráciu existuje aj niekoľko automatizovaných nástrojov. Patria sem nástroje ako *dnsenum*, *dnsrecon*, *Fierce* či *dnsmap*, ktorý slúži na nájdenie subdomén. Na hľadanie subdomén slúži aj nástroj *knockpy*. Na obrázku 34 je znázornený výstup nástroja *dnsrecon*, ostatné nástroje majú výstup podobný. [1, 6]

```
dnsrecon -d  mcuxpresso.nxp.com -t axfr
Checking for Zone Transfer for mcuxpresso.nxp.com name servers
Resolving SOA Record
     SOA n0dscb.akamaiedge.net 88.221.81.192
     SOA n0dscb.akamaiedge.net 2600:1480:e800::c0
Resolving NS Records
NS Servers found:
Removing any duplicate NS server IP Addresses...

Trying NS server 88.221.81.192
88.221.81.192 Has port 53 TCP Open
Zone Transfer Failed (Zone transfer error: REFUSED)

Trying NS server 2600:1480:e800::c0
Zone Transfer Failed for 2600:1480:e800::c0!
Port 53 TCP is being filtered
```

Obr. 34 DNS enumerácia a DNS zone transfer pomocou *dnsrecon*

Skenovanie

Skenovanie je ďalšia široko používaná technika pri aktívnom získavaní informácii. Skenovaním portov si vie hacker uľahčiť prácu a nesústrediť sa na všetky porty, ale napríklad iba na porty 80 a 443. Môže mať rôzne podoby, ako: [6]

- Ping sweep/Network sweep
- Detekcia operačného systému, služieb a verzii
- Skenovanie hostiteľských zariadení, ktoré majú vypnutý ICMP
- Tajné/Skryté skenovanie (Stealth scanning)
- TCP/UDP skenovanie
- Skenovanie portov

Úlohou je teda identifikovať živých hostiteľov na sieti, zistiť otvorené a zatvorené porty zariadenia, identifikovať bežiace služby na cieľovom systéme a vytvoriť si infraštruktúru

siete nášho cieľa. Na skenovanie slúžia rôzne nástroje, patria sem napríklad *Wireshark*, *massscan*, *Hping3* alebo *Nmap*. [6]

Nmap je rozsiahly nástroj, ktorý si popíšeme bližšie. Je bezplatný a dokáže množstvo vecí vrátane nasledujúcich: [6]

- Vytvorenie sieťového inventáru (network inventory)
- Hľadanie živých hostiteľov
- Identifikácia zraniteľností na hostiteľovi
- Detekcia snifferov
- Schopnosť určiť, či je v sieti prítomný firewall

Existuje aj grafická verzia, nazývaná *Zenmap*. Je určená pre začiatočníkov a navrhnutá na jednoduché používanie. V *Nmap* aj *Zenmap* je možné používať predpripravené skripty pomocou NSE (Nmap Scripting Engine), a tak vykonať rôzne operácie ako DNS Zone Transfer, automatické vyhľadávanie zraniteľností a mnohé ďalšie. Keďže sa sústredíme na testovanie webovej aplikácie, budú nás zaujímať hlavne verzie bežiacich služieb, určenie operačného systému, zraniteľnosti a prítomnosť firewallu. Zraniteľnosti budú popísané v samostatnej podkapitole. [6]

Medzi aktívne získavanie informácii sa zaraďujú aj skenery webového obsahu ako *Dirb* a *Dirbuster*. Pri týchto nástrojoch je dôležité spomenúť, že sú len tak dobré, ako zoznamy priečinkov a súborov, ktoré obsahujú a porovnávajú s nájdenými na webovom serveri.

Dirb

Dirb hľadá existujúce (aj skryté) webové objekty. Funguje tak, že vykonáva priečinkovo založené útoky (directory based attacks) proti webovému serveru a analyzuje jeho odpovede. [8]

Dirbuster

Je nástroj od organizácie OWASP. Na rozdiel od *Dirb* obsahuje aj grafické rozhranie. Podobne ako *Dirb* používa útoky hrubou silou na priečinky a mená súborov na webových serveroch. *Dirbuster* má 9 rozličných zoznamov, vďaka čomu vie efektívne vyhľadať aj skryté súbory a priečinky. Pokiaľ by tieto zoznamy nepostačovali, je stále možné vykonať čistý útok hrubou silou. [6, 9]

Keďže nechceme zahltiť ani zhodiť server, obmedzujeme počet žiadostí na 1 za sekundu. Tým však výrazne zväčšujeme čas skenovania (až niekoľko dní), a preto skenery webového obsahu na aplikáciu MCUXpresso Web SDK builder nepoužijeme.

Whatweb

Whatweb je mocný nástroj na získavanie informácii, ale slúži aj na zistenie zraniteľností (na tento účel bude spomenutý v ďalšej podkapitole). Vie rozpoznať viaceré webové technológie ako systémy riadiace obsah (v preklad content management systems, skratka CMS), blogovacie platformy, štatistické a analytické balíky, knižnice JavaScript, webové servery a vstavané zariadenia (v preklade embedded devices). Obsahuje vyše 900

pluginov, každý určený na identifikáciu niečoho iného. Vie tiež identifikovať čísla verzii, emailové adresy, ID účtov, web framework moduly, SQL chyby a mnohé iné. [10]

V tejto kapitole použijeme *Whatweb* iba na získanie informácii o cieli. Použijeme teda najmenej agresívny sken (takzvaný Stealth scan). Na obrázku 35 môžeme vidieť výstup *Whatweb*. Dozvedeli sme sa, že http server je Apache a že aplikácia obsahuje nezvyčajné hlavičky (uncommon headers), čo môže byť odkaz na podobné informácie ako sme dostali z *Security Headers*. Je však zjavné, že väčšina informácii je zabezpečená.

```
WhatWeb report for https://mcuxpresso.nxp.com/en/welcome
Status    : 200 OK
Title     : Welcome | MCUXpresso SDK Builder
IP        : 23.37.33.215
Country   : UNITED STATES, US

Summary   : UncommonHeaders[x-akamai-transformed,server-timing], Script, X-XSS-Protection[1; mode=block], Apache, X-Frame-Option
s[sameorigin], X-UA-Compatible[ie=edge], HTTPServer[Apache], Google-Analytics[Universal][UA-71560721-2], HTML5
```

Obr. 35 Výstup nástroja *Whatweb*

Whois

Tento nástroj niektoré zdroje uvádzajú ako pasívne, iné zasa ako aktívne získavanie informácii. *Whois* nám vráti informácie o poskytovateľovi internetových služieb, mená adminov, pracovné telefónne čísla, emaily, či IP adresy. Nedokázal však nájsť aplikáciu MCUXpresso Web SDK Builder pomocou jej domény mcuxpresso.nxp.com. [6]

5.3 Posudzovanie a vyhľadávanie zraniteľností

Ďalšiu techniku, ktorú použijeme na testovanie bezpečnosti webovej aplikácie MCUXpresso Web SDK Builder je posudzovanie a vyhľadávanie zraniteľností (vulnerability assesment and scanning). Nadväzuje na získavanie informácii, kedy získané informácie vieme použiť napríklad na lepší výber nástrojov a cielenejšie skenovanie konkrétnych zraniteľností (napríklad zraniteľnosti pre používané technológie). Dôvodom použitia tejto techniky je skutočnosť, že zraniteľnosti je možné zneužiť a tým spôsobiť nemalé škody vlastníkovi aplikácie. Túto techniku môžeme vykonávať manuálne ale aj automatizovanými nástrojmi. Ako bolo spomenuté už v predošlej podkapitole, práca sa bude sústrediť na automatizované nástroje. Veľká časť nástrojov tejto techniky je bohužiaľ platených ako napríklad *Nessus*, *Nexpose* a *Burp suite*. Tieto sú pochopiteľne najprecíznejšie, najsilnejšie a dokážu odhaliť najviac slabín a skupín zraniteľností. Majú však aj bezplatné verzie, s ktorými sa zoznámime. Do bezplatných nástrojov patria napríklad *Nikto*, *OWASP ZAP* a *OpenVAS*. Ďalšou skupinou nástrojov sú také, ktoré slúžia na špecifický typ zraniteľností a slabín, ako *Wapiti* alebo *Sqlmap*. Existujú aj celé frameworky ako *Metasploit* alebo *Legion*, ktoré vďaka početným pluginom združujú viacero funkcii a dokonca sú schopné v sebe spúšťať iné nástroje. [3, 6]

Známe zraniteľnosti a slabiny môžeme nájsť v databázach ako *CWE* (*Common Weakness Enummeration*) alebo *CVE* (*Common Vulnerabilities and Exposures*).

Obsahujú detailný popis zraniteľností vrátane ich príkladov, následkov ich zneužitia, obrany a detekčných metód. Na ohodnotenie závažnosti rizika, ktoré zraniteľnosti predstavujú slúži napríklad *CVSS* (*Common Vulnerability Scoring System*), ale jednotlivé spoločnosti majú aj vlastné ohodnotenia, ako napríklad *Nessus* na tenable.com. Je potrebné tiež spomenúť, že všetky najbežnejšie skupiny zraniteľností podľa OWASP TOP 10 nie je možné vyhľadať automaticky žiadnym skenerom a v mnohých prípadoch je potrebný namáhavý manuálny prístup. [1, 6]

Väčšina automatizovaných skenerov zraniteľností postupuje zhruba nasledovne: [3]

1. Zistia, či je cieľová aplikácia zapnutá a beží.
2. Spravia celkový alebo čiastočný sken portov (záleží na konfigurácii).
3. Identifikujú operačný systém pomocou bežných fingerprinting techník.
4. Pokúsia sa identifikovať bežiace služby pomocou techník ako banner grabbing, service behavior identification alebo nájdenia súborov (v preklade file discovery).
5. Vykonajú signature-matching proces (porovnanie súčastí a náležitostí aplikácie s databázou, kde sú k nim priradené zraniteľnosti) na nájdenie zraniteľností.

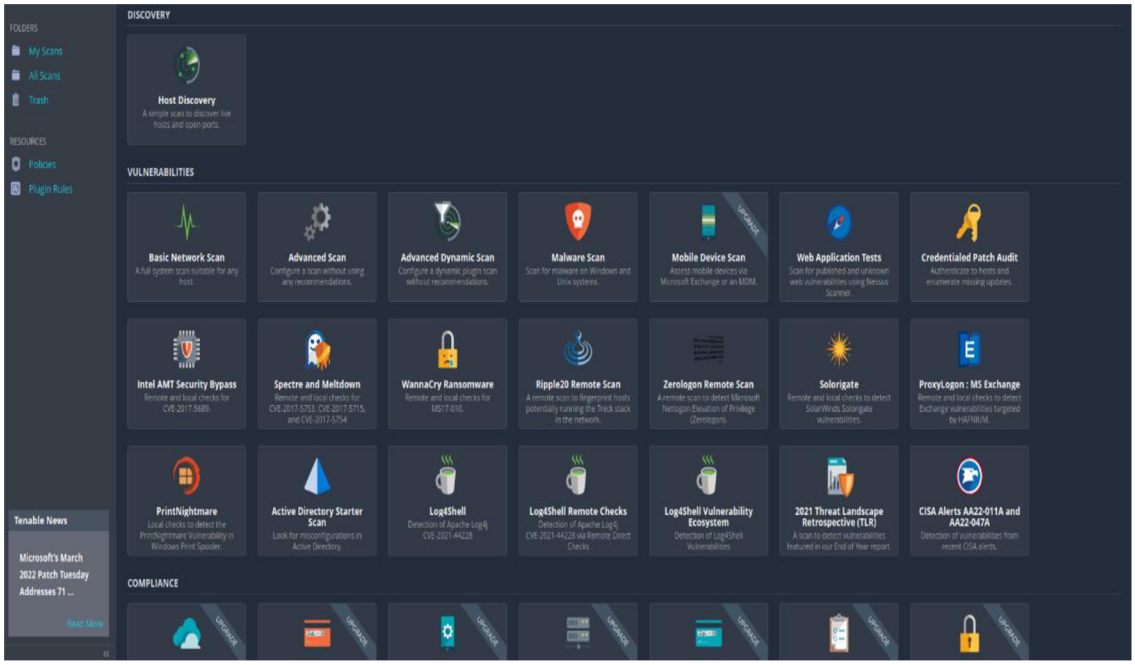
Týmto v podstate vykonajú postupne všetky kroky, ktoré by etický hacker robil manuálne. Napriek silnému signature-matching procesu nemáme úplnú garanciu prítomnosti nájdenej zraniteľnosti. Automatizované skenery nám teda môžu vrátiť výsledky: [1]

- Falošne pozitívne – Skener identifikoval niečo, čo si myslí, že je zraniteľnosť, no po bližšom preskúmaní zistíme, že to zraniteľnosť nie je.
- Falošne negatívne – Skener neidentifikoval zraniteľnosť, ktorá bola prítomná.
- Pravdivo pozitívne – Skener identifikoval zraniteľnosť, ktorá sa po manuálnom prehľadávaní ukázala ako legitímna.
- Pravdivo negatívne – Skener neidentifikoval zraniteľnosť tam, kde sa žiadna nenachádzala.

Kvôli týmto výsledkom je žiadúce každú zraniteľnosť manuálne vyhodnotiť a preskúmať. Napriek tomu sú signature-matching procesy značne efektívne a mnohonásobne rýchlejšie ako manuálny prieskum, vďaka čomu sú automatické skenery výbornou voľbou pri prvotnom vyhodnocovaní zraniteľností aplikácie. Slúžia tak ako perfektný doplnok k manuálnemu prieskumu. Postupne si teda prejdeme niektoré nástroje zo skupiny skenerov zraniteľností. [3, 6]

Nessus

Nessus je jeden z najpopulárnejších profesionálnych skenerov zraniteľností. Podporuje cez 130 000 pluginov na overenie zraniteľností. Do roku 2005 bol licencovaný ako open-source. Fork tohto open-source projektu viedol k vzniku *OpenVAS*, ktorý je stále bezplatný (no voči *Nessus* aj menej efektívny). Považuje sa za priemyselný štandard v hľadaní zraniteľností a ich vyhodnocovaní. *Nessus* má prístupnú aj bezplatnú verziu *Essentials*, ktorá však nesmie slúžiť na komerčné účely. Môžeme ju však použiť na edukačné účely. Ponúka nám veľké množstvo rôznych skenov, ako je možné vidieť na obrázku 36. [6, 11]



Obr. 36 Prístupné skeny nástroja Nessus

Niektoré prvky sú prístupné iba po zakúpení platenej licencie. Na obrázku 36 ich môžeme vidieť medzi *VULNERABILITIES* prelepené páskou s nápisom *UPGRADE*. Nás však bude predovšetkým zaujímať sken *Web Application Tests*, ktorý slúži na hľadanie zraniteľností vo webovej aplikácii a je na to špeciálne nakonfigurovaný. V pravom hornom rohu obrázka 36 môžeme vidieť nadpis *FOLDERS* a položky *My Scans* a *All Scans*, kde sa nachádzajú skeny vytvorené a nakonfigurované užívateľom. Je možné si tiež všimnúť položky *Trash*, v ktorej sa nachádzajú vymazané / zahodené skeny. Pod ním sa nachádza nadpis *RESOURCES* a položky *Policies* a *Plugin Rules*. *Policies* slúži na vytvorenie vzoru konfigurácie skenu, ktorý môžeme opakovane používať a *Plugin Rules* slúži na zmenu dôležitosti daného pluginu alebo jeho vypnutie. Po vytvorení a nakonfigurovaní skenu je možné ho púšťať aj opakovane, napríklad v dopredu určených intervaloch (pre bezplatnú verziu je však obmedzenie na automatizáciu iba jedného skenu). Ovládanie je intuitívne. Výstupom skenu / testu sú zraniteľnosti s popismi, rozdelenie do niekoľkých kategórii podľa závažnosti (najnižší stupeň je najmenšia závažnosť): [11]

- Info (nepredstavuje žiadne riziko)
- Low (predstavuje nízke riziko)
- Medium (predstavuje strednú úroveň rizika)
- High (predstavuje vysoké riziko pre aplikáciu)
- Critical (predstavuje kriticky vysoké riziko s vážnymi dopadmi v prípade zneužitia)

Na aplikáciu MCUXpresso SDK Web Builder sme použili niekoľko typov skenov podľa hĺbky prehľadávania a nastavených parametrov. Prvým bol rýchly sken typu Web Application Tests, skenujúci všetky známe zraniteľnosti webových aplikácií. Ďalším bol komplexný sken typu Web Application Tests, kedy bolo rozšírené skenovanie portov

a boli použité všetky http metódy, vrátane pokusu o http parameter pollution. Nakoniec sme použili pokročilý sken typu Advanced Scan, ktorý mal všetky parametre predošlých spomínaných skenov spolu s pridanými pluginmi (za cenu dlhšieho skenovania). Tento sken je schopný aj útoku hrubou silou a skenovania malware. Ani jeden sken nenašiel rizikové zraniteľnosti.

OWASP ZAP

Je open-source skener zraniteľností od spoločnosti OWASP. Pracuje ako man-in-the-middle proxy, pretože sa nachádza medzi používateľovým prehliadačom a webovou aplikáciou. Okrem automatického skenovania zraniteľností je možne použiť ho aj na skenovanie manuálne, poprípade ním zneužiť samotné zraniteľnosti vo forme penetračného útoku. Poskytuje užitočné techniky ako aktívne skenovanie, pasívne skenovanie a crawling pomocou pavúkov (v preklade spiders). Tento nástroj je vhodnou bezplatnou voľbou, ponúkajúcou široké spektrum možností analýzy bezpečnosti webovej aplikácie. Má tiež množstvo rozšírení (add-ons), ktoré slúžia na pridanie nových techník analýzy. [12]

Nikto

Nikto je konfigurovateľný, open-source webový skener, ktorý vie vyhľadať mnohé bezpečnostné nedostatky, ako napríklad zraniteľné verzie serverov, problémy s konfiguráciou, potencionálne nebezpečné súbory a programy a mnohé iné. Nájdenie zraniteľností detailne referencuje pomocou OSVDB (Open Source Vulnerability Database), čo je nezávislá a open-source databáza obsahujúca informácie o webových zraniteľnostiach. Je však potrebné spomenúť, že Nikto nie je tichý (stealth-oriented) nástroj a pri jeho používaní na seba útočník ľahko upozorní. Oskenuje všetky stránky webovej aplikácie, ktoré dokáže nájsť, preto pri veľkých webových aplikáciách môže jeho sken zabrať veľa hodín. Jeho skenovanie sa dá, samozrejme, obmedziť. [13]

Whatweb

Tento nástroj bol spomínaný už v predošlej podkapitole pri aktívnom získavaní informácii. Podľa stupňa agresie vieme určiť, do akej hĺbky chceme danú webovú aplikáciu skúmať. Pri nastavení agresie na úroveň 4 bude nástroj hľadať aj zraniteľnosti webovej aplikácie. Je treba poznamenať, že čim vyššia úroveň skenu, tým pomalší je a tým viac na seba útočník upozorní. [10]

Nmap

Ďalší nástroj, s ktorým sme sa už zoznámili pri aktívnom získavaní informácii. Nmap a *Zenmap* môžu slúžiť na vyhľadávanie zraniteľností pomocou NSE, kedy spúšťajú skripty písané v programovacom jazyku Lua. Je možné použiť vlastnoručne napísané skripty, alebo tie, ktoré ponúka *Nmap* (tieto je taktiež možné ľubovoľne zmeniť). Po oskenovaní aplikácie nám Nmap vráti zraniteľnosti spolu s odkazmi na ich dokumentáciu na stránkach ako OWASP a CVE. [14]

Sqlmap

Sqlmap slúži na automatizáciu procesu nájdenia a zneužitia zraniteľnosti SQL Injekcie a prevzatia databázových serverov. Má plnú podporu mnohých SQL serverov ako MySQL, Oracle, PostgreSQL, Microsoft SQL Server a mnohé ďalšie. [15]

Wapiti

Wapiti vykonáva black-box skeny. To znamená, že neštuduje zdrojový kód aplikácie, ale skenuje jej webové stránky a hľadá v nich skripty a formy, do ktorých by mohol vložiť dáta. Tým efektívne testuje, či je aplikácia náchylná na skupinu útokov Injekcie. Má aj viaceré moduly na testovanie iných zraniteľností ako buster na hľadanie zneužiteľných súborov. Je bezplatný a dokáže spustiť aj nástroj *Nikto*. [16]

Burp Suite

Tento nástroj je značne obsiahly a oplýva mnohými spôsobmi, ako analyzovať webovú aplikáciu. Je často známy ako najlepší proxy nástroj. Podporuje automatické aj manuálne hľadanie zraniteľností či penetračné testy na ich zneužitie. Jeho nedostatkom je však licencia, ktorá je platená. Bezplatná licencia sa nehodí na automatické skenovanie zraniteľností. Manuálne testovanie je v bezplatnej *Burp Suite Community Edition* dobré. Tento nástroj má tiež dobre prepracovanú dokumentáciu a spomenutý je aj preto, že jeho platené licencie sú naozaj silným nástrojom webovej bezpečnosti.

Existuje veľa nástrojov na vyhľadávanie zraniteľností. V tejto kapitole boli uvedené iba niektoré vybrané, na základe aktuálnosti, popularity, dostupnosti či osobnej preferencie (ktorá býva pri voľbe nástroja často silný faktor).

5.4 Iné techniky testovania bezpečnosti webových aplikácií

Popisovať bližšie všetky techniky bezpečnosti webových aplikácií by bolo príliš obsiahle, preto sme sa sústredili v predošlých kapitolách na najdôležitejšie techniky. Pred tým, než si detailnejšie spomenieme poslednú dôležitú techniku penetračné testovanie, je rozumné v krátkosti popísať iné, taktiež používané a dôležité techniky.

Modelovanie hrozby (Threat modeling)

Táto technika je white-box typu. Vieme teda všetky dôležité informácie o systéme alebo aplikácií a na ich základe sa snažíme vytvoriť model, ktorý by pre nich posudzoval možné riziká. Má nasledovné kroky: [3]

- Dekomponovanie aplikácie – manuálna inšpekcia na pochopenie, ako aplikácia funguje.
- Definovanie a klasifikovanie prostriedkov – vrátane ohodnotenia na základe dôležitosti pre firmu.
- Preskúmanie možných zraniteľností – či už to technických, operačných alebo riadiacich.

- Preskúmanie možných hrozieb – vytvoriť realistický pohľad potencionálneho útočníka z jeho perspektívy za použitia útočných scenárov alebo útočných stromov (attack trees).
- Vytvorenie stratégii na zmiernenie – vytvoriť opatrenia na zmiernenie hrozieb.

Výstupom je súhrn zoznamov a diagramov. Výhodou je flexibilita a praktický pohľad systému z útočníkovej perspektívy, no nevýhodou je, že dobrý model neznamená automaticky dobrý software. [3]

Revízia zdrojového kódu (Source code review)

Je proces manuálnej kontroly zdrojového kódu webovej aplikácie s úmyslom nájsť problémy s bezpečnosťou. Mnohé závažné zraniteľnosti nedokážeme nájsť iným spôsobom analýzy alebo testovania. Vďaka prístupu k zdrojovému kódu vie tester presne určiť čo sa deje (alebo čo sa má diať) a vyhne sa tak hádaniu ako pri black-box technikách. Revíziou často vieme zistiť problémy s kontrolou prístupu, kryptografické slabiny, trójske kone či iné nedostatky. Väčšina bezpečnostných expertov sa zhoduje, že revízia zdrojového kódu sa nedá ničím nahradiť. Výhodami sú efektivita, presnosť a rýchlosť (pri kompetentných revíznych technikoch). Nevýhodou je potreba vysoko kvalifikovaných vývojárov v oblasti bezpečnosti, nemožnosť jednoducho zistiť runtime chyby a možnosť prehliadnutia problémov v už skompilovaných knižniciach. [3]

5.5 Penetračné testy

Poslednou technikou, ktorú si detailnejšie priblížime, sú penetračné testy. Patria do skupiny black-box testov. Táto technika nie je plne automatizovateľná, a tak sa penetračné testy vykonávajú predovšetkým manuálne. Hlavný rozdiel oproti zisťovaniu a vyhodnocovaniu zraniteľností je, že penetračné testovanie sa zameriava na zneužitie týchto zraniteľností. Typicky sa tím penetračných testerov snaží pristupovať k aplikácií z pohľadu používateľa. Ide teda o to zistiť, ako veľmi je možné danú zraniteľnosť zneužiť a kam sa až dokážeme týmto zneužitím dostať, teda k čomu získame prístup. Technika penetračného testovania je posledná spomenutá preto, lebo sama zahŕňa predošlé spomenuté techniky ako získavanie informácii, hľadanie zraniteľností a modelovanie hrozby. Penetračné testovanie je metodológia, ktorá pozostáva z nasledujúcich sekcii: [3, 17]

- Pasívne získavanie informácii
- Aktívne získavanie informácii (skenovanie, enumerácia)
- Modelovanie hrozby
- Zisťovanie a vyhodnocovanie zraniteľností (manuálne, manuálne s pomocou nástrojov alebo čisto za použitia automatizovaných nástrojov)
- Zneužitie zraniteľností a získanie prístupu
- Pokiaľ sme prístup získali, udržiavanie tohto prístupu
- Pokiaľ je to žiadúce, zametanie stôp. V našom prípade nepoužívame žiadne nástroje na ukrývanie identity alebo zametanie stôp. Dôvodom je fakt, že v našom prípade

je dôležité aby vlastník aplikácie poznal identitu testera, a tak sa chceme vyhnúť nedorozumeniam. Sú však prípady, kedy je zametanie stôp vyslovene žiadané, napríklad keď chceme zistiť, či sa vie útočník dostať do systému nepozorovane a nepozorovane z neho aj ukradnúť dáta (napríklad testovanie IDS alebo IPS).

- Správa o penetračnom teste.

Tieto body zahŕňajú techniky používané v penetračnom teste vo všeobecnosti. Je ich možné rozdeliť aj na dielčie techniky a tak pridať špecifickejšie kroky. Záleží aj na individuálnom postupe penetračného testera. Vo všeobecnosti sa penetračné testovanie vykonáva menej pravidelne ako napríklad hľadanie zraniteľností, keďže je náročnejšie a pomalšie. V správe o penetračnom teste sa nachádzajú popisy všetkých objavených zraniteľností alebo problémov, vrátane závažnosti rizika. Ďalej obsahuje popis ako a do akej miery je možné tieto zraniteľnosti využiť. Spoločnosti ako OWASP ponúkajú dlhý a detailný zoznam, ako vykonať celkový penetračný test webovej aplikácie. Podľa tohto zoznamu sa testujú technológie, ktorých zraniteľnosti boli popísané v kapitole 4., ako testovanie autentizácie, autorizácie, kryptografie, konfigurácie či validácie vstupu. [3, 17]

5.6 Porovnanie techník a nástrojov webovej bezpečnosti

Kedže technika ako penetračné testovanie obsahuje iné spomenuté techniky, nie je možné ju úplne rovnocenne porovnať s inými technikami. Každopádne je možné porovnať, na čo sa jednotlivé techniky zameriavajú a spolu s nimi porovnať ich nástroje.

Pomocou techniky získavania informácii vieme o cieli zistiť verejne prístupné informácie ako mená a emailové adresy zamestnancov, technológie, ktoré používajú (do určitej miery) ale tiež IP a DNS adresy, ochranu proti enumeráciu a zone transfer. Aj keď aktívne získavanie informácii mierne zasahuje do vyhodnocovania a hľadania zraniteľností, samotné získavanie informácii sa nezameriava na nájdenie zraniteľností a jeho metódy sú často riešené cez tretiu stranu. Zaužívané spoločnosti ako *Netcraft*, *Security Headers* či *SSL Server Test* vedia pomerne jednoducho získať užitočné informácie o technológiach či praktikách používaných firmou. Veľmi silným nástrojom na zistenie adries a technológii je *Maltego*. Na uľahčenie techniky sociálneho hackerstva nám zasa slúži prehľadávanie webovej stránky či získavanie informácii o užívateľoch, napríklad aj pomocou *theHarvester*. Mnohé zo získaných informácii môžeme použiť pri vyhľadávaní a vyhodnocovaní zraniteľností, napríklad IP adresy zariadení. Vďaka získavaniu informácii je môžné zúžiť rozsah technológii, ktorých nedostatky hľadáme pomocou techniky vyhľadávania a vyhodnocovania zraniteľností.

Vyhľadávanie a vyhodnocovanie zraniteľností sa sústredí čisto na nájdenie, popísanie a ohodnotenie nebezpečnosti zraniteľnosti. Netestuje, ako hlboko sa dokážeme do systému alebo aplikácie dostať zneužitím zraniteľností, ani k čomu ich zneužitím dostaneme prístup. Manuálne zisťovanie zraniteľností je časovo náročnejšie a vyžaduje hlbšie znalosti, ale často vie skutočnú zraniteľnosť nájsť s vysokou pravdepodobnosťou. Niektoré zraniteľnosti alebo skupiny zraniteľností sa nedajú nájsť inak, ako manuálnym zisťovaním zraniteľností. Pri automatickom zisťovaní býva často pravidlom, že platené

nástroje sú lepšie a dokážu nájsť oveľa viac zraniteľností. Špecializované nástroje ako *Sqlmap* alebo *Wapiti* slúžia lepšie na nájdenie istej formy slabín. Nástroje ako *OWASP ZAP* alebo *Burp Suite* zasa slúžia ako perfektný pomocník pri manuálnom testovaní. Nástroje so všeobecnejším zameraním a bezplatnou verziou ako *Whatweb* alebo *Nmap* nemusia podávať relevantné výsledky pri všetkých nájdených problémoch.

Modelovanie hrozby slúži predovšetkým na zistenie častí, kam by útočník mohol potencionálne zaútočiť a navrhnúť model obrany aplikácie. Nezameriava sa na zistenie doposiaľ neznámych zraniteľností alebo ich zneužitie.

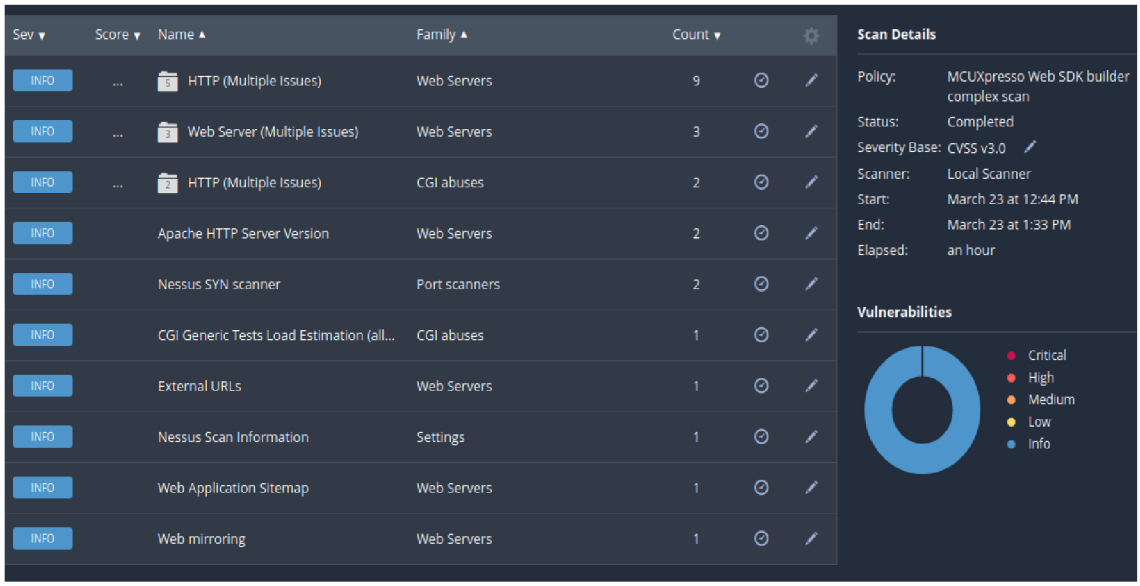
Revízia zdrojového kódu sa zameriava na samotný zdrojový kód aplikácie s účelom odhaliť zakázané praktiky či nedokonalosti v implemetácii. Nesústredí sa na testovanie bežiacej aplikácie, ani na vyhľadávanie či zneužitie zraniteľností.

Penetračné testovanie pokrýva väčšinu vyššie spomenutých techník. V penetračnom testovaní ide predovšetkým o zneužitie nájdených zraniteľností a vypísaní správy o podrobnom postupe tohto zneužitia (aby bolo možné postup zopakovať). Toto je hlavný rozdiel oproti samotnému posudzovaniu a vyhľadávaniu zraniteľností.

6. ANALÝZA A HĽADANIE ZRANITEĽNOSTÍ APLIKÁCIE MCUXPRESSO SDK WEB BUILER

Z popísaných techník bolo vybrané vyhľadávanie a vyhodnocovanie zraniteľností. Dôvodov je niekoľko. Prvým je, že sa táto technika dá pomerne dobre automatizovať, aj vďaka jej automatizovaným nástrojom. Druhým dôvodom je fakt, že táto technika je kľúčová aj pri penetračnom teste. V neposlednom rade je dôležitosť zabezpečeného systému bez zraniteľností a slabín. Analýzu vykonáme pomocou automatizovaných nástrojov *Nikto, Whatweb, Wapiti, Sqlmap, Nessus* a *OWASP ZAP*. Z týchto nástrojov použijeme za prihlásením *Wapiti* a *OWASP ZAP*, ktorý použijeme aj na manuálne testovanie. To sa bude sústrediť na skupinu Injekcie (A03), ktorá je stále pomerne vysoko v rebríčku OWASP TOP 10. Niektoré nástroje, ako napríklad *Nessus*, nám poslúžia na prehľadávanie viacerých skupín zraniteľností. Je treba mať na pamäti, že žiadny nástroj nedokáže automaticky vyhľadať všetky zraniteľnosti a manuálne prehľadávanie všetkých skupín OWASP TOP 10 je vysoko pracné a nad rámec tejto práce.

Plné výstupy nástrojov budú pripojené k práci ako príloha, keďže celkové záznamy z testovania majú veľa strán. V obrázkoch sa môže vyskytnúť *User-agent* nastavený na hodnotu „Diplomka“. Toto je z dôvodu identifikácie vlastníkmi aplikácie, pretože niektoré typy skenov, ktoré boli použité sú „hlasné“ (ochranné systémy ich ľahko odhalia a vlastníka aplikácie upozornia, že sa niekto snaží zistiť, či je jeho aplikácia zraniteľná). Pozrime sa teda na prvý výstup skenu od nástroja *Nessus*, ktorý môžeme vidieť na obrázku číslo 37.

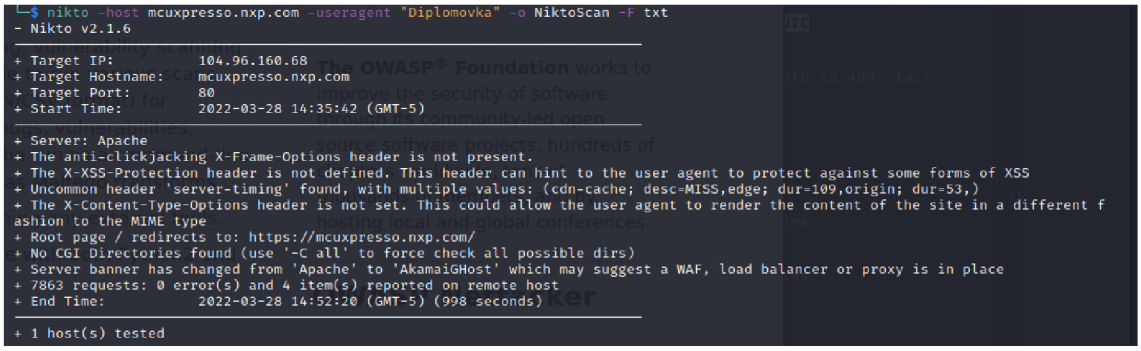


Obr. 37 Výstup komplexného skenu nástroja *Nessus*

Podľa popisu v podkapitole 5.3 už vieme, že *Nessus* má úrovne nebezpečnosti zraniteľností. Tu sú všetky označené iba ako *Info*, čo naznačuje, že žiadna nie je „skutočnou“ zraniteľnosťou. Sú to len odporúčania, ktoré by mohol vývojár nasledovať.

Skener tiež zistil, že aplikácia používa server APACHE. V prílohe 1. a 2. sa nachádzajú detailnejšie správy o skenoch pomocou nástroja *Nessus*.

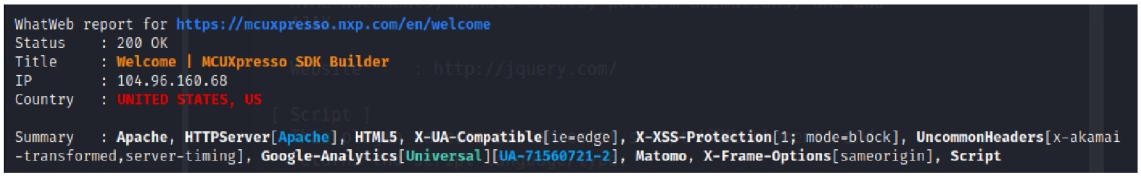
Ďalším použitým nástrojom bol *Nikto*. Na obrázku 38 môžeme vidieť jeho výstup.



Obr. 38 Vystup nastroja *Nikto*

Zo skenu sme sa dozvedeli niekoľko vecí. *X-XSS-Protection header* označil za neprítomný, čo je však v rozpore s výstupom Security Headers (obrázok 32) a tak isto s výstupom *Whatweb* (obrázok číslo 39). Oba tvrdia, že hodnota hlavičky je *X-XSS-Protection:1;mode=block* (to znamená, že prehliadač zamedzí vygenerovaniu stránky, ak deteguje XSS útok). Naznačuje to falošne pozitívny výsledok nástroja *Nikto*. Chýbajúci *anti-clickjacking X-Frame-Options header* môže naznačovať možnosť clickjacking útoku (zobrazenie neviditeľného HTML prvku na inom prvku, ktorý používateľ vidí a môže naň kliknúť). *Whatweb* zistil, že *X-Frame-Options* je nastavený na *sameorigin*. To znamená, že aplikácia dovolí zobrazenie stránky v rámci (frame) na inej stránke, no iba v aktuálnej doméne. To obmedzuje možnosť *clickjacking* útoku. Najbezpečnejšie je nastaviť hodnotu tejto hlavičky na deny, pokiaľ to neobmedzuje špecifické potreby pre rámcovanie (framing). Nenastavený *X-Content-Type-Options header* naznačuje, že by stránka mohla byť zraniteľná na *MIME sniffing*, čo je technika prehľadávania obsahu prúdu bitov. To môže vytvoriť príležitosť pre XSS útok. XSS patrí do skupiny A03 podľa OWASP TOP 10.

Na obrázku 39 môžeme vidieť už spomínaný výstup nástroja *Whatweb*. Podrobnejší výstup skenu *Whatweb* sa nachádza v prílohe 3..

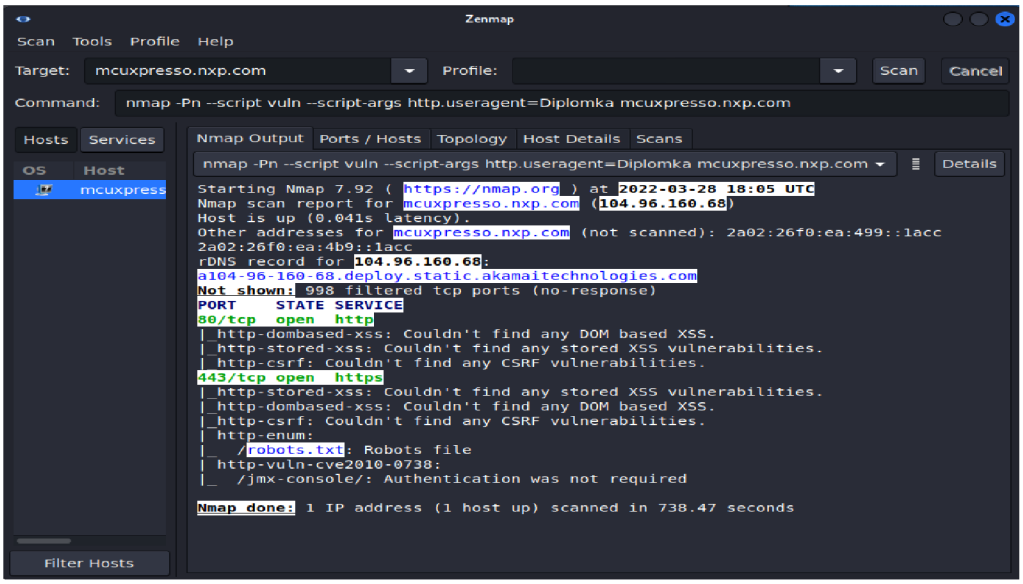


Obr. 39 Výstup nástroja *Whatweb*

Whatweb okrem už známych vecí odhalil napríklad použitie HTML5, či Matomo (open-source software na zaznamenávanie návštevy webov).

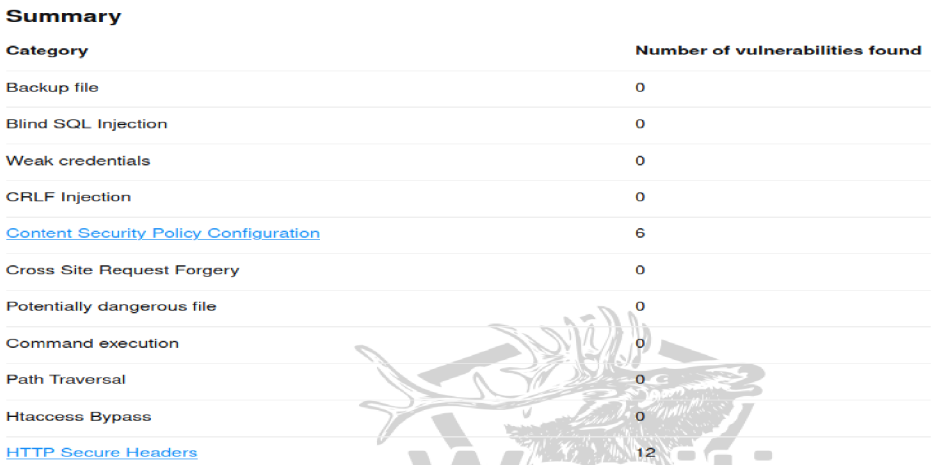
Nasledujúcim použitým nástrojom bol *Nmap*, v tomto prípade jeho grafická verzia *Zenmap*. Príkaz, ktorý je použitý, by fungoval rovnako aj pri použití v príkazovom riadku s rovnakým výstupom. Bolo využité NSE pre zraniteľnosti v argumente *–script vuln*. Výstup nástroja Zenmap môžeme vidieť na obrázku 40. Z obrázka je zrejmé, že *Zenmap*

nenašiel žiadne zraniteľnosti, v tomto prípade sa zameriaval na XSS a CSRF (Cross-Site request forgery).



Obr. 40 Výstup nástroja *Zenmap*

Sken pomocou nástroja *Wapiti* neodhalil žiadne závažné zraniteľnosti. *Wapiti* sa sústredí predovšetkým na zraniteľnosti skupiny Injekcie. Tak ako nástroje pred ním, aj on detegoval problémy s bezpečnostnými hlavičkami. Našiel tiež neprítomnosť konfigurácie CSP (Content Security Policy). CSP slúži na zabezpečenie aplikácie proti XSS útoku. Plný záznam skenu Wapiti je v prílohe 4. pre sken prihlásený a v prílohe 5. pre sken neprihlásený vo forme HTML. Vo formáte .json pre neprihlásený sken je záznam o skene *Wapiti* v prílohe 6.. Prihlásenie bolo riešené predaním cookies s prihlasovacími údajmi na zákaznícky účet. Tieto cookies sme získali pomocou *wapiti-getcookie*. Oba skeny dopadli takmer rovnako. Na obrázku číslo 41.



Summary

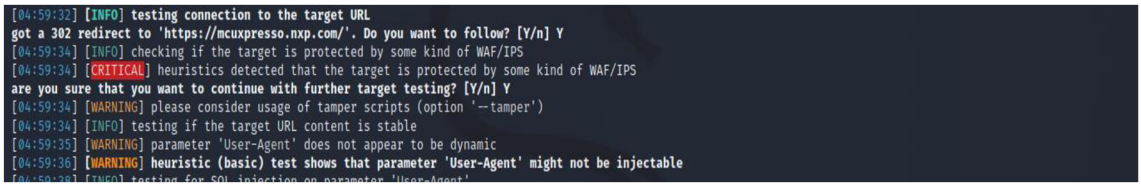
Category	Number of vulnerabilities found
Backup file	0
Blind SQL Injection	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	6
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Htaccess Bypass	0
HTTP Secure Headers	12

Obr. 41 Orezaný výstup *Wapiti*

Zabezpečenie proti skupine útokov Injekcie sme si potvrdili aj nástrojom *Sqlmap*. Vedľa hlásenia *CRITICAL* môžeme vidieť, že aplikácia je proti SQL Injekcíam zabezpečená. Relevantnú časť jeho výstupu je možné vidieť na obrázku 42.

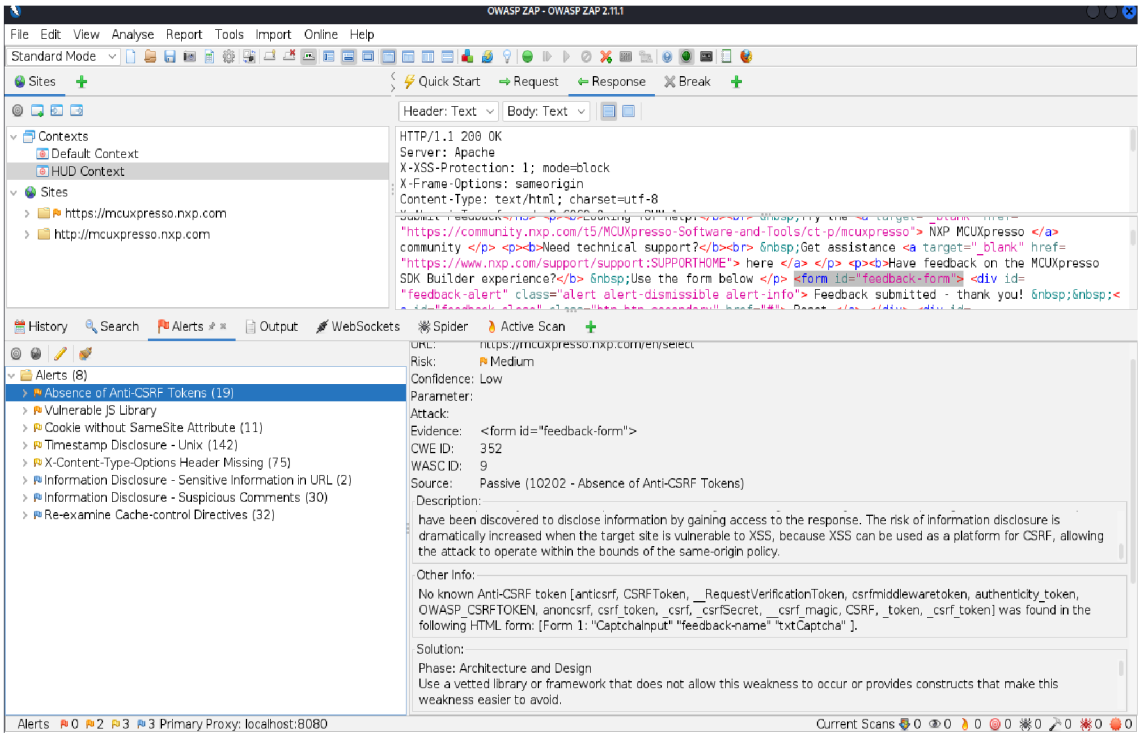
Obr. 42 Výstup nástroja *Sqlmap*

Sqlmap tiež zistil, že aplikácia je chránená nejakou formou WAF (Web application firewall) alebo IPS (Intrusion prevention system). To môžeme vidieť na obrázku 43 vedľa hlásenia *CRITICAL*.



Obr. 43 Hlásenie o ochrane pomocou WAF / IPS

Posledným nástrojom, ktorý sme použili na analýzu bezpečnosti aplikácie, je *OWASP ZAP*. Tento nástroj sme použili na automatické hľadanie zraniteľností, aj sme si ním vypomohli pri manuálnom testovaní aplikácie. Na obrázku 44 môžeme vidieť výsledky automatického skenu zraniteľností v prihlásenej aplikácii. Detailnejší záznam o teste zraniteľností pomocou *OWASP ZAP* je uvedený v prílohe 7.



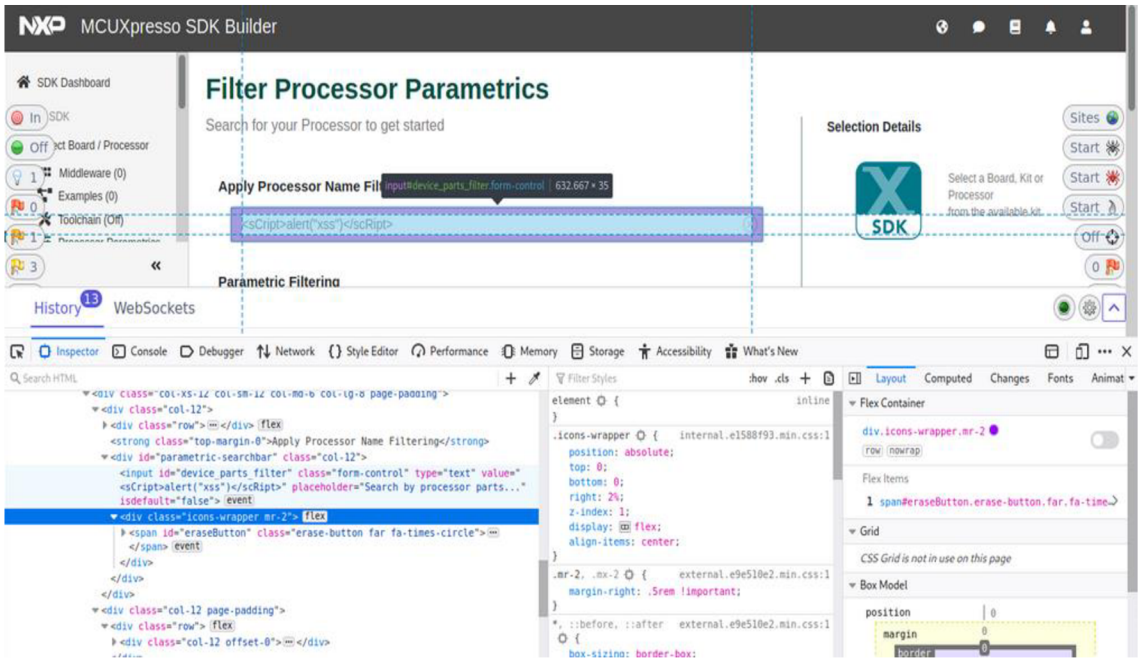
Obr. 44 Výsledok skenu zraniteľností pomocou *OWASP ZAP*

Na obrázku 44 si ďalej môžeme povšimnúť mnoho zaujímavých vecí. *OWASP ZAP* nám konkrétne popíše, kde v kóde (HTML kód v strede aplikácie) našiel zraniteľnosť a vyznačí ju (v tomto *prípade <form id =”feedback.form”>*). Ďalej nám poskytne krátky popis zraniteľnosti, riešenie, ako sa tohto problému zbaviť a ID pre dohľadanie v databáze CWE. Aktuálne označená zraniteľnosť (pravý dolný roh pod adresárom Alerts) Cross-Site Request Forgery (CSRF) je označená s nebezpečnosťou médium. Tu sa však naskytá otázka, či je táto zraniteľnosť skutočne zraniteľnosťou, alebo iba slabina. Cross-Site Request Forgery patrí skupiny A01 podľa OWASP TOP 10 a funguje tak, že útočník sfalšuje kód stránky, doplní do neho svoj vlastný, škodlivý kód a pošle ho obeti (spojenie so sociálnym hackerstvom). Presvedčí ju napríklad, že si musí zmeniť heslo a pošle odkaz na svoju sfalšovanú stránku, ktorá sa tvári úplne rovnako ako stránka pôvodná. Obeť, v domienke že sa jedná o skutočnú stránku, do nej vyplní svoje prihlasovacie údaje a tým ich útočník získa. V tomto prípade ale máme k dispozícii iba textové boxy na filtrovanie kritérii pri procesoroch a formu na zadávanie spätnej väzby, keďže sme už za prihlásením a účet si môže vytvoriť akýkoľvek cudzí človek (čo vystupuje napríklad ako zákazník). Je preto možné zhodnotiť, že takouto formou sa nedajú ukradnúť nejaké relevantné dáta a tým pádom je absencia Anti-CSRF tokenu skôr slabinou, ako zraniteľnosťou. Zraniteľná knižnica Javascriptu (Vulnerable JS Library) patrí do skupiny A06 podľa OWASP TOP 10 a upozorňuje nás na zastaralú verziu tejto knižnice, s odporúčaním na jej okamžitú aktualizáciu. Ostatné upozornenia sú skôr informačné, niektoré boli už rozobraté (ako X-Content-Type-Options Header) a nepredstavujú hrozbu. Číslo vedľa názvu slabiny alebo zraniteľnosti udáva, koľkokrát sa v aplikácii opakuje. Po rozkliknutí zraniteľnosti/slabiny je možné vidieť metódu, ktorou program zraniteľnosť/slabinu zistil a miesto, kde ju objavil. Bližší detail je možné vidieť na obrázku 45.

Obr. 45 Detail nájdených zraniteľností/slabín

Presuňme sa k manuálnemu testovaniu. V tomto prípade sme použili *OWASP ZAP* ako Man-in-the-middle klienta, aby sme mohli analyzovať odpovede stránky. Skúmané boli zraniteľnosti XSS a SQL Injekcia. Na ich testovanie slúži viacero spôsobov. Popíšme si stručne manuálny test na zraniteľnosť XSS. Snažíme sa vložiť časť kódu do nejakého vstupného okna aplikácie (najlepšie takého, ktoré niečo vracia ako odpoveď) a sledujeme, či sa niečo udeje. Pokiaľ aplikácia odpovie na vloženie kódu tým, že tento kód vykoná, znamená to, že by mohla byť napadnuteľná pomocou XSS útoku. V našom prípade bol použitý jednoduchý kód *<sCript>alert(‘xss’)</scRipt>*. Po jeho prevedení by malo v aplikácii vyskočiť okno s nápisom xss. Niektoré písmená sú veľkými preto, lebo stránky môžu implenetovať filter kedy sledujú isté slová ako *script* a odstránia ich zo vstupu. Zväčšenie jedného alebo viacerých písmen je veľmi jednoduchý prípad, ako tento filter

obísť. Na obrázku číslo 46 môžeme vidieť, že skript bol vložený do kódu stránky (medzi <input> tagmi nad modro vyznačenou časť kódu), ale po jeho zadaní sa nič nestalo. Tento proces bol zopakovaný pri všetkých textových boxoch aplikácie s rovnakým výsledkom pri *Middleware*, *Examples* a *Processor Parametrics* oknách, pričom pri oknách *Notifications* a *Offline data* ani len neuložilo skript do kódu. Iná forma testu je vložiť tento kus kódu do URL aplikácie, no v tomto prípade bola reakcia rovnaká, nič sa neudialo. Napriek uloženiu kódu bol kód neresponsívny, a tak môžeme zhodnotiť, že stránka je zabezpečená proti XSS útokom (podložené automatickými skenermi).



Obr. 46 Testovanie na XSS zraniteľností

Posledným testom na aplikácii MCUXpresso SDK Web Builder bolo manuálne vyskúšanie SQL Injekcie. Pokiaľ je z URL jasné, že aplikácia používa napríklad php, je možné skúšať vkladať SQL príkazy rovno do URL a sledovať odpovede stránky. Často sa používa napríklad 1=1 a 1=0 na overenie, či je stránka rovnaká po vyhodnotení porovnania na pravdu, alebo true (1 = 1) a na nepravdu, alebo false (1 = 0). U nás URL neobsahuje php, preto sme sa pokúsili zistiť, či vyhľadávacie textboxy nevyhodia nejakú chybu, ktorá by nám mohla napovedať, či je stránka zraniteľná na SQL Injekciu. To sme sa snažili docieliť vložením znakov ako je ‘ alebo “ do textového boxu. Aplikácia žiadnu chybu nevrátila. Spolu s tvrdením automatických skenerov, ktoré rovnako nenašli žiadnu zraniteľnosť typu SQL Injekcia, môžeme tvrdiť, že aplikácia je v tomto ohľade bezpečná.

Podľa výstupu nástrojov, ich záznamov o testovaní a vyššie spomenutých informácii môžeme zhodnotiť, že aplikácia MCUXpresso Web SDK Builder netrpí žiadnymi závažnými bezpečnostnými nedostatkami, ktoré by boli schopné použité schopné odhaliť.

7. TESTOVACÍ PLÁN BEZPEČNOSTI APLIKÁCIE MCUXpresso Web SDK Builder

Posledná kapitola je zameraná na navrhnutie testovacieho plánu bezpečnosti aplikácie MCUXpresso Web SDK Builder. Časť tohto plánu sa pokúsime zautomatizovať.

Preskúmané techniky nám ponúkajú veľa možností testovania aplikácie. Vo všeobecnosti sa dá test plán bezpečnosti nazvať penetračným testom, keďže penetračné testovanie obsahuje väčšinu techník. Plán bude zahŕňať osobitne jednotlivé techniky, ale bude sa v ňom nachádzať aj krok penetračné testovanie. Je to z dôvodu voľnosti testera, podľa toho, do akej hĺbky chce aplikáciu skúmať a ktoré všetky kroky chce použiť (v tomto prípade slúži zahrnutie penetračného testovania v pláne ako možné rozšírenie práce o podrobnejšie skúmanie aplikácie). Navrhnutý testovací plán bezpečnosti je teda nasledujúci:

- Analýza zdrojového kódu aplikácie (napríklad pre nedodržiavanie noriem bezpečného programovania).
- Získavanie informácii o firme pasívne – pomocou *Netcraft*, *Maltego*, *Security Headers* a iných vyššie spomínaných techník.
- Získavanie informácii o firme aktívne – *Whatweb*, *DNSenum*, vrátane testu na ochranu proti DNS Zone Transfer.
- Automatické hľadanie a analýza zraniteľností aplikácie – napríklad pomocou *OWASP ZAP*, *Wapiti*, *Nessus* či *Nikto*.
- Manuálne hľadanie a analýza zraniteľností aplikácie – preskúmanie prípadných nájdených zraniteľností a overenie pravdivosti záznamu o testovaní automatizovaných nástrojov.
- Penetračné testovanie – zahŕňajúce viaceré predošlé kroky. Možnosť brať ako príležitosť rozšíriť testovací plán, napríklad oveľa rozsiahlejším manuálnym hľadaním zraniteľností.
- Zhodnotenie výsledkov, vypracovanie správy o bezpečnosti aplikácie.
- Prijatie patričných opatrení na vyriešenie nájdených bezpečnostných nedostatkov.

7.1 Automatizácia časti testovacieho plánu bezpečnosti

Na automatizáciu bola zvolená časť Automatické hľadanie a analýza zraniteľností aplikácie. Dôvodov je hneď niekoľko. Prvým je fakt, že by sa táto časť mala opakovať v pravidelných intervaloch počas roka. Tým zabránime mnohým útokom zameraným napríklad na neaktuálne verzie použitých prostriedkov. Druhým dôvodom je, že hľadanie zraniteľností je kľúčové pri akejkoľvek aplikácii a tvorí v podstate pevný základ pre ďalšie penetračné testovanie. Tretím dôvodom je skutočnosť, že táto časť je najlepšie automatizovateľná, aj vďaka samotným automatizovaným nástrojom.

Vďaka automatizovaným nástrojom je možné vytvoriť shell skript ScanAll.sh (príloha 8.), v ktorom sú tieto nástroje volané s príslušnou konfiguráciou postupne

a generujú nám správu o výsledkoch ich testovania. V skripte sa volajú všetky nástroje z kapitoly 6., okrem nástroja *Nessus*. Dôvodom je nutnosť prihlásenia do osobného účtu, a teda poskytnutie osobných údajov. Okrem toho je možné automatizáciu rozšíriť pomocou technológii CI / CD ako Atlasian Bamboo alebo Jenkins, ktoré by shell skrip púštali automaticky v daný čas. Pokiaľ nemáme k dispozícii zariadenie s Kali Linuxom, môžeme shell skript púšťať napríklad pomocou Docker Image Kali. Automatizáciu je tiež možné navrhnúť pomocou Python API pre *OWASP ZAP* a napísať si tak vlastný sken tohto nástroja. Nachádza sa v prílohe číslo 9. (ZAP_Scanner.py). Keďže je potrebné, aby pri tomto skeneri bežal ZAP daemon (implementovaný v prílohe 11. ako daemon_runzap.sh), je ZAP_Scanner.py púšťaný pomocou runScan.py (príloha číslo 10.), ktorý púšťa zároveň ZAP daemona aj skener. OWASP ZAP používa ako východzí port 8080, takže je nutné ho mať otvorený a pripravený počúvať. Pri písaní skeneru bolo použité API dokumentácie pre ZAP dostupnej z [12]. Iný spôsob automatizácie je implementácia celého vlastného skenera, napríklad v jazku Python. V práci sú však použité profesionálne nástroje určené na testovanie bezpečnosti webových aplikácií, a tak je implementácia vlastného, potencionálne menej presného nástroja zbytočná.

8. ZÁVER

V tejto práci bola testovaná bezpečnosť aplikácie MCUXpresso Web SDK Builder od spoločnosti NXP.

Najprv bolo popísané fungovanie samotnej aplikácie, jej časti za prihlásením a pred prihlásením, spolu s krátkym návodom vytvorenia SDK.

Práca sa ďalej zaoberala najčastejšími zraniteľnosťami a slabinami podľa spoločnosti OWASP a ich rebríčka OWASP TOP 10. Toto bolo dôležité z dôvodu pochopenia, čo vlastne hľadáme, ako vznikajú problémy so zabezpečením aplikácie a ako sa proti nim brániť.

Nasledovalo preskúmanie techník a nástrojov bezpečnosti webových aplikácií. V práci bol na testovanie aplikácie použitý operačný systém Linux, konkrétne jeho distribúcia Kali, slúžiaca na etické hackovanie, informačnú bezpečnosť a mnohé iné oblasti. Techniky, ktoré boli popísané boli predovšetkým získavanie informácii, vyhľadávanie a analýza zraniteľností a penetračné testovanie. V krátkosti boli spomenuté aj iné techniky. Ku každej detailnejšie popísanej technike boli spomenuté aj jej nástroje. Veľa z týchto nástrojov je navrhnutých pre prácu na Kali Linuxe, niektoré sú v ňom aj predinštalované. Spomenutiahodné nástroje pri technike Získavania informácii sú napríklad *Maltego*, *theHarvester*, *Netcraft*, *Dnsrecon* či *Security Headers*. Získali sme pomocou nich napríklad IP adresy, technológie, ktoré firma a aplikácia používajú, názvy DNS serverov a chýbajúce bezpečnostné hlavičky. Podrobnejšie bola aplikácia testovaná technikou analýzy a vyhľadávania zraniteľností. Tu bola za použitia nástrojov *Nessus*, *Nikto*, *Wapiti*, *Sqlmap*, *OWASP ZAP*, *Nmap* a *Whatweb* aplikácia zoskenovaná a ich výstupné správy vyhodnotené. Aplikáciu môžeme zhodnotiť ako bezpečnú, žiadne závažné zraniteľnosti sa nenašli. Nájdená bola zastaralá verzia knižnice Javascriptu a chýbajúca ochrana proti Cross-Site Request Forgery. Nebezpečnosť tejto chýbajúcej ochrany je však diskutabilná, vzhľadom k použitiu textových boxov v aplikácii, kedy nemôže dôjsť k zneužitiu údajov pomocou Cross-Site Request Forgery. Pri testovaní sme sa snažili vyhnúť útokom hrubou silou (brute force attacks), aby sme nevyvolali výpadok služieb aplikácie. Na záver techník je vhodné spomenúť, že penetračné testovanie obsahuje takmer všetky zmienené techniky, pričom je značne obsiahle.

Posledná kapitola práce sa venovala návrhu test plánu bezpečnosti pre aplikáciu MCUXpresso Web SDK Builder. Časť tohto plánu bola zautomatizovaná, a to konkrétne Automatické hľadanie a analýza zraniteľností. Automatizácia bola prevedená automatickým výstupom nástrojov, ktoré boli volané v shell skripte ScanAll.sh. Ďalším možným rozšírením automatizácie je použitie Continuous Integration / Continuous Development nástrojov ako je Atlassian Bamboo či Jenkins, ktoré by daný bash script púštali automaticky v daný čas. Pokiaľ nemáme k dispozícii zariadenie s Kali Linuxom, môžeme shell skript púšťať napríklad pomocou Docker Image Kali. Druhý spôsob automatizácie bol ukázaný na implementácii python skriptu pomocou API pre nástroj OWASP ZAP. Hľadanie a analýzu zraniteľností je možné automatizovať aj za pomoci

vlastného skeneru zraniteľností, napríklad v jazyku Python. Keďže sa ale v práci používajú profesionálne nástroje určené na testovanie bezpečnosti webovej aplikácie, je implementácia vlastného skeneru s potencionálne menej presnými výsledkami zbytočná.

ZOZNAM POUŽITEJ LITERATÚRY

[1] MESSIER, Ric. *CEH v10: Certified Ethical Hacker Study Guide*. Indianapolis, Indiana: John Wiley & Sons, 2019. ISBN 978-1-119-53319-1.

[2] *HOFFMAN, A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. 1005 Gravenstein Highway North, Sebastopol: O'Reilly Media, 2020. ISBN 978-1-492-08796-0.*

[3] OWASP. *Web Application Security Testing Guide* [online]. [cit. 2022-05-02]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/stable/

[4] OWASP. *Obrázok postupu útoku* [online]. [cit. 2022-05-02]. Dostupné z: https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf

[5] OWASP. *OWASP TOP 10*: 2021 [online]. [cit. 2022-05-02]. Dostupné z: https://owasp.org/Top10/

[6] SINGH, Glen D. *Learn Kali Linux 2019: Perform Powerful penetration testing using Kali Linux, Metasploit, Nessus, Nmap, and Wireshark*. Liverly Place, 35 Liverly Street, Birmingham, B3 2PB, UK: Packt Publishing, 2019. ISBN 978-1-78961-180-9.

[7] Miessler, Daniel. *The Difference Between Red, Blue, and Purple Teams* [online]. [cit. 2022-05-02]. Dostupné z:

https://danielmiessler.com/study/red-blue-purple-teams/

[8] OFFSEC SERVICES. *Kali Linux Tools. Dirb* [online]. [cit. 2022-05-02]. Dostupné z: https://www.kali.org/tools/dirb/

[9] OFFSEC SERVICES. *Kali Linux Tools. Dirbuster* [online]. [cit. 2022-05-02]. Dostupné z: https://www.kali.org/tools/dirbuster/

[10] OFFSEC SERVICES. *Kali Linux Tools. Whatweb* [online]. [cit. 2022-05-02]. Dostupné z: https://www.kali.org/tools/whatweb/

[11] Tenable. *Nessus Dokumentácia* [online]. [cit. 2022-05-03]. Dostupné z:

https://docs.tenable.com/nessus/10_1/Content/GettingStarted.htm

[12] *OWASP ZAP Dokumentácia.* [online]. [cit. 2022-05-03]. Dostupné z:

https://www.zaproxy.org/docs/

[13] Github: Sullo, Chris. *Nikto Wiki.* [online]. [cit. 2022-05-03]. Dostupné z: https://github.com/sullo/nikto/wiki

[14] *Nmap Dokumentácia* [online]. [cit. 2022-05-03]. Dostupné z:

https://nmap.org/book/man.html

[15] *Sqlmap: Automatic SQL Injection and Database Takeover Tool* [online]. [cit. 2022-05-03]. Dostupné z: https://sqlmap.org/

[16] Github: Surribas, Nicolas. *Wapiti: A Free and Open-Source Web Applicatoin Vulnerability Scanner in Python* [online]. [cit. 2022-05-03]. Dostupné z: https://wapiti-scanner.github.io/

[17] *Penetration Testing Guidance* [online]. [cit. 2022-05-03]. Dostupné z: https://www.pcisecuritystandards.org/documents/Penetration-Testing-Guidance-v1_1.pdf?agreement=true&time=1651604083297

[18] *Common Weakness Enumeration* [online]. [cit. 2022-05-03]. Dostupné z: https://cwe.mitre.org/

[19] DUFFY, Christopher. *Python: Penetration Testing for Developers*. Liverly Place, 35 Liverly Street, Birmingham, B3 2PB, UK: Packt Publishing, 2016. ISBN 978-1-78712-818-7.

[20] A. Al Anhar and Y. Suryanto, "Evaluation of Web Application Vulnerability Scanner for Modern Web Application," *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)*, 2021, pp. 200-204, doi: 10.1109/ICAICST53116.2021.9497831.

[21] MCDONALD, Malcom. *Web Security for Developers: Real Threats, Practical Defense*. 245 8th Street, San Francisco, CA 94103: No Starch Press, 2020. ISBN 978-1-5932-7994-3.

[22] PortSwigger. *Burp Suite testing for OWASP TOP 10* [online]. [cit. 2022-05-05]. Dostupné z:

https://portswigger.net/support/using-burp-to-test-for-the-owasp-top-ten

[23] WEIDMAN, Georgia. *Penetration testing: A Hands-on Introduction to Hacking*. 245 8th Street, San Francisco, CA 94103: No Starch Press, 2014. ISBN 978-1-59327-564-8.

[24] *Common Vulnerability and Exposure* [online]. [cit. 2022-05-05]. Dostupné z: https://www.cve.org/

[25] *Common Vulnerability Scoring System Version 3.1* [online]. [cit. 2022-05-05]. Dostupné z: https://www.first.org/cvss/v3-1/

[26] D. Yadav, D. Gupta, D. Singh, D. Kumar and U. Sharma, "Vulnerabilities and Security of Web Applications," *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, 2018, pp. 1-5, doi: 10.1109/CCAA.2018.8777558.

[27] M. Agreindra Helmiawan, E. Firmansyah, I. Fadil, Y. Sofivan, F. Mahardika and A. Guntara, "Analysis of Web Security Using Open Web Application Security Project 10," *2020 8th International Conference on Cyber and IT Service Management (CITSM)*, 2020, pp. 1-5, doi: 10.1109/CITSM50537.2020.9268856.

[28] NXP. *MCUXpresso web builder, firemní materiál* [online]. [cit. 2022-05-02]. Dostupné z: https://mcuxpresso.nxp.com/en/welcome

[29] HERTZOG, Raphaël, Jim O'GORMAN a Mati AHARONI. *Kali Linux Revealed: Mastering the Penetration Testing Distribution*. 19701 Bethel Church Road, #103-253 Cornelius NC 28031 USA: Offsec Press, 2017. ISBN 978-0-9976156-0-9.

[30] SINHA, Sanjib. *Beginning Ethical Hacking with Kali Linux: Computational Techniques for Resolving Security Issues*. Apress, 2018. ISBN 978-1-4842-3890-5.

[31] CISCO. *Penetration Testing and Network Defense: Performing Host Reconnaissance* [online]. [cit.2022-05-05] .Dostupné z:

https://www.ciscopress.com/articles/article.asp?p=469623&seqNum=2

[32] *Penetration Testing Framework* [online]. [cit. 2022-05-05]. Dostupné z: http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html

[33] Github: *Sqlmap Dokumentácia* [online]. [cit. 2022-05-05]. Dostupné z: https://github.com/sqlmapproject/sqlmap/wiki/Introduction

[34] Tenable. *Nessus Plugins* [online]. [cit. 2022-05-05]. Dostupné z: https://www.tenable.com/plugins

[35] OWASP. *Zoznam Útokov* [online]. [cit. 2022-05-05]. Dostupné z: https://owasp.org/www-community/attacks/

[36] PortSwigger. *Burp suite WebSecurity Academy* [online]. [cit. 2022-05-05]. Dostupné z: https://portswigger.net/web-security

ZOZNAM SKRATIEK

API – Application Programming Interface
SDK – Software Development Kit
URL – Uniform Resource Locator
JSON – JavaScript Object Notation
CWE – Common Weakness Enumeration
OWASP – Open Web Application Security Project
GDPR – General Data Protection Regulation
SQL – Structured Query Language
SMTP – Simple Mail Transfer Protocol
FTP – File Transfer Protocol
MD5 – Message Digest 5
SHA1 – Secure Hash Algorithm 1
TLS – Transport Layer Security
FS – Forward Secrecy
HTML – HyperText Markup Language
DOM – Document Object Model
ORM – Object Relational Mapping
NIST – National Institute of Standards and Technology
XSS – Cross-Site Scripting
DoS – Denial of Service
SOP – Same Origin Policy
REST – Representational State Transfer
JS – JavaScript
CSS – Cascading Style Sheets
XML – Extensible Markup Language
AJAX – Asynchronous JavaScript and XML
SPA Framework –
SOAP – Simple Objec Access Protocol
LED – Light Emitting Diode
CVE – Common Vulnerability and Exposure
CVSS – Common Vulnerability Scoring System
CORS – Cross-Origin Resource Sharing
UI – User Interface
CLI – Command Line Interface
LDAP – Lightweight Directory Access Protocol
EL – Expression Language
PL/SQL – Proceduarl Language extensions to the Structured Query Language
T-SQL – Transact Structured Query Language
ACL – Access Control List

NVD – National Vulnerability Database
CI / CD – Continuous Integration / Continuous Delivery alebo Deployment
DAST – Dynamic Application Security Testing
VPN – Virtual Private Network
SSRF – Server-Side Request Forgery
IDE – Integrated Development Enviroment
IDS / IPS – Intrusion Detection System / Intrusion Prevention System
GHDB – Google Hacking Database
DNS – Domain Name System
IPv4 / IPv6 – Internet Protocol version 4 / Internet Protocol version 6
SSL – Secure Sockets Layer
OSVDB – Open Source Vulnerability Database
CSRF – Cross-Site Request Forgery
WAF – Web Application Firewall

ZOZNAM PRÍLOH

Príloha č.1 Nessus_Web_Application_Tests_(Complex scan).pdf
Príloha č.2 Nessus_Advanced_Scan.pdf
Príloha č.3 WhatwebScan.txt
Príloha č.4 Wapiti_Prihlaseny_HTML
Príloha č.5 Wapiti_Neprihlaseny_HTML
Príloha č.6 Wapiti_Neprihlaseny.json
Príloha č.7 OWASP_ZAP_Report
Príloha č.8 ScanAll.sh
Príloha č.9 ZAP_Scanner.py
Príloha č.10 runScan.py
Príloha č.11 daemon_runzap.sh