
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

## ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS


# NÁVRH AUDITNÍHO SYSTÉMU PRO KONTROLU SYSTÉMŮ A SLUŽEB V POČÍTAČOVÉ SÍTI

DESIGN OF AN AUDIT SYSTEM FOR THE CONTROL OF SYSTEMS AND SERVICES IN A COMPUTER NETWORK.

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Peter Šimkovič**
AUTHOR

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



**BRNO 2021**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

# Zadání diplomové práce

| | |
|---|---|
| Ústav: | Ústav informatiky |
| Student: | **Bc. Peter Šimkovič** |
| Studijní program: | Systémové inženýrství a informatika |
| Studijní obor: | Informační management |
| Vedoucí práce: | **Ing. Petr Sedlák** |
| Akademický rok: | 2020/21 |

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

## Návrh auditního systému pro kontrolu systémů a služeb v počítačové síti

**Charakteristika problematiky úkolu:**

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

**Cíle, kterých má být dosaženo:**

Analyzovat bezpečnostní hrozby a možná rizika, která působí na počítačové sítě. Na základě analýzy vytvořit analytický nástroj pro kontrolu počítačové sítě a podle výstupu zpracovat bezpečnostní opatření a pravidla pro nápravu možných bezpečnostních slabin.

**Základní literární prameny:**

ČSN ISO/IEC 27001, Informační technologie - Bezpečnostní techniky - Systémy managementu bezpečnosti informací - Požadavky, Praha: Český normalizační institut, 2014.

HORÁK, J. a M. KERŠLÁGER. Počítačové sítě pro začínající správce. 5., aktualiz. vyd. Brno: Computer Press, 2011. ISBN 978-80-251-3176-3.

HUNT, C. TCP/IP network administration. 3rd ed. Sebastopol: O'Reilly & Associates, 1992. ISBN 0-937175-82-X.

JORDÁN, V. a V. ONDRÁK. Infrastruktura komunikačních systémů I: Univerzální kabelážní systémy. Brno: Akademické nakladatelství CERM, 2013. ISBN 978-80-214-4839-1.

KUROSE, J., K. ROSS a J. JONÁK. Počítačové sítě. 1. vyd. Brno: Computer Press, 2014. ISBN 978-80-251-3825-0.

ONDRÁK V., P. SEDLÁK a V. MAZÁLEK. Problematika ISMS v manažerské informatice. Brno: CERM, Akademické nakladatelství, 2013. ISBN 978-80-7204-872-4.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně dne 28.2.2021

L. S.

| | |
|---|---|
| Mgr. Veronika Novotná, Ph.D.<br>ředitel | doc. Ing. Vojtěch Bartoš, Ph.D.<br>děkan |

**Abstrakt**

Táto diplomová práca sa zaoberá návrhom auditného systému, pomocou ktorého bude možné detegovať bezpečnostné medzery v počítačovej sieti. Detekcia bude prebiehať na základe kontroly vopred určených portov a služieb na nich bežiacich. Na základe výstupu z auditného systému, dôjde k navrhnutiu možných opatrení a pravidiel, nutných k náprave bezpečnostných medzier. Samotný systém bude navrhnutý tak, aby bola zabezpečená jeho jednoduchá implementácia u akéhokoľvek zákazníka.

**Kľúčové slová**

Počítačová sieť, audit, detekcia, skenovanie, bezpečnosť, služby, porty, kontajner, docker, nmap, virtualizácia

**Abstract**

This diploma thesis deals with the design of an audit system with which it will be possible to detect security gaps in a computer network. Detection will take place on the basis of checking predefined ports and services running on them. Based on the output of the audit system, possible measures and rules necessary to correct security gaps will be proposed. The system itself will be designed to ensure its easy implementation for any customer.

**Keywords**

computer network, audit, detection, scanning, security, services, ports, container, docker, nmap, virtualization

**Bibliografická citácia**

ŠIMKOVIČ, Peter. *Návrh auditního systému pro kontrolu systémů a služeb v počítačové síti* [online]. Brno, 2021 [cit. 2021-05-09]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/133633. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Petr Sedlák.

**Čestné prehlásenie**

Prehlasujem, že predložená diplomová práca je pôvodná a spracoval som ju samostatne. Prehlasujem, že citácia použitých prameňov je úplná, že som vo svojej práci neporušil autorské práva (v zmysle Zákona č. 121/2000 Sb., o práve autorskom a o právach súvisiacich s právom autorským).

V Brne dňa 09. Mája 2021

……

podpis študenta

**Poďakovanie**

Rád by som týmto spôsobom poďakoval pánovi Ing. Sedlákovi za jeho vedenie nie len počas písania diplomovej práce, ale počas celého štúdia. Taktiež by som chcel poďakovať zamestnancom z firmy C SYSTEM CZ a.s., za odborné konzultácie a sprostredkovanie podkladov a informácii. Úprimné ďakujem patrí všetkým, ktorí mi pomohli sa dostať k riešenej problematike, náležite ju pochopiť a následne dané znalosti aj zužitkovať. Taktiež ďakujem za veľkú dôveru, s ktorou mi bolo umožnené využívať informačnú infraštruktúru spoločnosti.

# Obsah

# ÚVOD

Hacking, a teda hľadanie zraniteľností, je v rámci IT odvetvia takmer rovnako staré ako samotná počítačová sieť. Prvá experimentálna sieť ARPANET vznikla v roku 1969 a v danej dobe prepájala len 4 uzly. Už dva roky na to vzniká prvý počítačový červ „Creeper Worm“, ktorý bol schopný autonómneho šírenia pomocou ARPANET na ďalšie počítače. Odpoveďou na to bol „Reaper“, ktorý sa taktiež dokázal šíriť sieťou podobne ako Creeper, avšak s tým rozdielom, že jeho účelom bolo odstraňovanie autonómne sa rozširujúceho červa. V podobnom duchu to pokračuje až dodnes.

Motivácia pre kybernetické útoky býva rôzna, niekedy je to vidina finančného zisku pomocou vydierania, inokedy je to získanie citlivých dát pre dosiahnutie výhody nad obeťou a niekedy je to samotná deštrukcia a poškodenie obete. Nech je motivácia akákoľvek, tak sa dotýka každého z nás. Hacking, ako už bolo spomínané, je hľadanie zraniteľností, či už je to zraniteľnosť spojená s ľudským, technologickým alebo iným faktorom.

Ľudský faktor býva často najzraniteľnejší a prináša najväčšie riziká. Či už je to neúmyselné zavinenie, napríklad v dôsledku phishingu alebo úmyselné a jedná sa o crackera. Vplyv prvého ľudského faktoru môže do značnej miery zamedziť vhodné školenie. Čo však môže zamedziť obom vplyvom, či už úmyselnému zavineniu človeka z vnútra alebo zneužitiu zraniteľnosti využívanej technológie, je dobre nastavená bezpečnostná politika a nadstavenie pravidiel a oprávnení.

Základným pravidlom pre nastavovanie oprávnení je zakázať všetko, čo nie je nevyhnutné. Pre takéto nastavenie, a taktiež pre kontrolu dodržiavania, je nutné dokázať spracovávať audit siete a prvkov, ktoré sa na nej nachádzajú. Práve z dôvodu aktuálnosti danej problematiky je táto diplomová práca zameraná práve na tému vytvorenia nástroja a prípadnej metodiky pre skenovanie a audit siete.

Ďalšie detaily o zámere, metodike a štruktúre práce budú spracované v časti „Ciele práce, metódy a postupy spracovania“.

# CIELE PRÁCE, METÓDY A POSTUPY SPRACOVANIA

Hlavným cieľom tejto diplomovej práce je vytvoriť program umožňujúci kontrolu počítačovej siete a prvkov na nej implementovaných. Na základe kontroly bude vypracovaná metodika nápravy odhalených zraniteľností a nadstavenie vhodných pravidiel pre to, aby sa zamedzilo ich opätovnému výskytu. Tvorba ako samotného programu, tak metodiky bude založená na spracovanej analýze bezpečnosti počítačovej siete u klienta spoločnosti C SYSTEM CZ a.s., ďalej len CSYSTEM. Viac informácii o spoločnosti CSYSTEM a spomínanom klientovi je dostupných v úvode kapitoly 2.

Metodický postup pre spracovanie diplomovej práce je zložený z viacerých krokov. Prvým krokom je pomocou literatúry získať rozhľad v problematike bezpečnosti počítačových sietí a analýzy rizík. Ďalšou časťou je praktická časť, ktorá sa skladá z analýzy súčasného stavu. Táto analýza sa zameriava primárne na stav vývojového a technologického prostredia spoločnosti. Vďaka danej analýze je možné nadstaviť základné požiadavky na tvorbu prostredia a samotného skenovacieho skriptu. Ďalšou časťou analytickej časti je analýza rizík, ktorá sa skladá z identifikácie aktív, hrozieb, zraniteľností a ich zhodnotenie. Výstupom je hodnota rizík pre jednotlivé aktíva. Vďaka tomu je možné vytvoriť a nastaviť skript tak, aby čo najlepšie spĺňal podmienky pre špecifickú sieť v spoločnosti. Cieľom poslednej, návrhovej časti, je na základe analýz spracovať zariadenie tak, aby bolo ľahko prenosné, zabezpečené a taktiež aj v čo najväčšej miere komplexné pre budúce úpravy. Druhým cieľom návrhovej časti je nastavenie skriptu tak, aby čo najlepšie spĺňal požiadavky analyzovanej siete. Taktiež sa v rámci danej časti práce odprezentuje príkladný výstup zo skenovania a predstavia sa možné opatrenia na nápravu.

# 1 TEORETICKÉ VÝCHODISKA PRÁCE

Táto časť práce je určená na poučenie o význame jednotlivých pojmov, ktorým je potrebné rozumieť pre správnu orientáciu a porozumenie vo zvyšku diplomovej práce.

## 1.1 Monitorovanie a audit

Jedná sa o pravidelné preskúmavanie účinnosti opatrení, pri čom sa berie ohľad na výsledky bezpečnostných auditov, merania účinnosti opatrení, incidentov a podnetov zainteresovaných strán (1, s. 73).

K určeniu toho, či je predmet skúmania primeraný, efektívny a v konečnom dôsledku vhodný k dosiahnutiu stanovených cieľov slúži **Preskúmanie** (Review) (1, s. 73).

Následne, na základe auditu je možné usúdiť to, či sú splnené požiadavky. V prípade nesplnenia požiadavkou, dochádza k stavu **nezhody**. Opatrenie ktoré je určené na odstránenie nezhody sa nazýva **náprava** (1, s. 73).

## 1.2 Definícia a klasifikácia aktív

Aktívum je slovo, ktoré reprezentuje majetok, či už hmotný alebo nehmotný. V skratke ide o čokoľvek, čo má pre danú spoločnosť hodnotu. Prvým a základným krokom k vhodnej správe a hodnotení aktív je ich identifikácia (1, s. 82).

Pre úspešné spracovanie tohto kroku, alebo inak aj etapy, je odporúčané:

- Najskôr je vhodné zoskupiť všetky aktíva, ktoré spadajú pod jednotlivé logické množiny (obchodné aktíva, bezpečnostné aktíva, programové aktíva, služby a podobne) (1, s. 82).
- Následne je potrebné identifikovať a priradiť vlastníka ku každému aktívu. Osoba označená ako vlastník aktíva je priamo zodpovedná nie len za dané aktívum, ale taktiež aj za určenie konkrétnej hodnoty daného aktíva (1, s. 82).

### 1.2.1 Hodnotenie aktív

K hodnoteniu aktív sa odporúča používať softwarové nástroje. Ideálne je využitie špecializovaných programov, ako je napríklad CRAMM metodika. V prípade absencie

prostriedkov, alebo spracovávania jednoduchého hodnotenia je možné použiť aj voľne dostupné prostriedky, ako je na príklad MS Excel a podobne (1, s. 82).

Po navolení softwarového nástroju je ďalším krokom stanovanie stupnice a hodnotiacich kritérií. Tieto kritéria budú následne využité pre priradenie ohodnotenia určitého aktíva. Stupnica samotná, môže byť vyjadrená kvantitatívne, napríklad peniazmi, alebo inými kvalitatívnymi hodnotami. Príkladom kvalitatívneho ohodnotenia môže byť na príklad stupnica, vyjadrujúca hodnotu v škále od veľmi nízka až po kritická. To, či bude zvolená kvantitatívna metóda, kvalitatívna metóda, poprípade ich kombinácia, závisí na rozhodnutí organizácie (1, s. 82.).

Príklad tabuľky vhodnej pre kvalitatívne hodnotenie:

**Tabuľka č. 1 Príklad hodnotenia aktív**
(Zdroj: Vlastné spracovanie podľa 1, s. 82)

| | | | |
|---|---|---|---|
| 1 | | žiaden dopad na organizáciu | bezvýznamné riziko |
| 2 | | zanedbateľný dopad na organizáciu | akceptovateľné riziko |
| 3 | | problémy alebo finančné straty | nízke riziko |
| 4 | | vážne problémy alebo citeľné finančné straty | nežiadúce riziko |
| 5 | | existenčné problémy | neprijateľné riziko |

Najväčší dôraz sa pri hodnotení aktív berie na náklady vzniknuté v dôsledku porušenia hlavných informačných kritérií. Tými sú:

- **Dôvernosť** – zaistenie toho, že informácie sú prístupné len pre oprávnené subjekty (1, s. 83).
- **Dostupnosť** – zaistenie toho, že informácia, poprípade nástroj je pre oprávneného užívateľa ihneď dostupný (1, s. 83).
- **Integrita** – zaistenie toho, že informácie sú správne a úplne. To znamená, že nimi nebolo nijak neoprávnene manipulované (1, s. 83).

### 1.2.2 Výpočet hodnoty aktíva

K výpočtu hodnoty aktíva existuje množstvo rôznych postupov. Najjednoduchším a najpoužívanejším je súčtový algoritmus. Ten je založený na matematickom princípe súčtu podeleného počtom prvkov: (x + y + z)/3 - (dostupnosť + dôvernosť + integrita)/3 (1, st. 83).

Ako je možné vidieť v tabuľke č. 1, tak hodnotiaca škála je v rozmedzí 1-5 pre každé informačné kritérium. Ku príkladu:

- Dôvernosť – 5
- Dostupnosť – 3
- Integrita – 4

Pri aplikácii výpočtu: (5+3+4)/3=4. Z tabuľky č. 1 opäť vieme vyčítať že sa jedná o „*vážne problémy alebo citeľné finančné straty“.* Výslednou hodnotou je taktiež definovaný aj **dopad** (1, st. 84).

## 1.3 Bezpečnostné hrozby

Hrozbou je možné nazvať akúkoľvek udalosť, ktorá má potenciál ohroziť bezpečnosť, a tým spôsobiť nežiadúci incident. Následkom takéhoto ohrozenia môže byť poškodenie aktív spoločnosti (1, st. 86).

### 1.3.1 Delenie hrozieb

Z hľadiska pôvodu je možné deliť na zavinenie ľudským faktorom, ako je na príklad chyba užívateľa, poprípade úmyselné poškodenie, a zavinené prírodným faktorom. U zavinenia prírodným faktorom figurujú vplyvy ako na príklad povodne, zemetrasenie, požiar, silný vietor a podobné (1, st. 86).

Z hľadiska úmyslu dochádza k jednoduchému deleniu na úmyselné a neúmyselné, čo znamená náhodné. Ďalšie delenie je možné z pohľadu zdroja a to na vonkajšie a vnútorné (1, st. 86).

Ako už bolo v kapitole Definícia a klasifikácia aktív uvádzané, tak je ďalej potrebné deliť hrozby aj podľa toho, na aké aktíva pôsobia. Takéto delenie je založené na logickom zoskupení aktív:

- sieť
- operačný systém
- aplikácie
- databáza
- klient (1, st. 86).

## 1.4 Analýza rizík

Základom pre proces znižovania rizík je ich analýza. V rámci analýzy rizík sa pracuje s definovanými hrozbami, pravdepodobnosťami ich realizácie a taktiež aj so samotným dopadom na aktíva. V princípe ide o stanovenie rizík a ich závažnosti (9, s. 93).

Proces analýzy rizík sa skladá z nasledujúcich krokov:

1.) Identifikácia aktív – jedná sa o definíciu aktív (9, s. 94).
2.) Stanovenie hodnoty aktíva – klasifikácia a výpočet hodnoty aktíva (9, s. 94).
3.) Identifikácia hrozieb a slabín – identifikácia a definícia udalostí ohrozujúcich bezpečnosť, poprípade inak znižovať hodnotu aktíva (9, s. 94).
4.) Stanovenie a ohodnotenie závažnosti hrozieb a miery zraniteľnosti – ohodnotenie pravdepodobnosti výskytu hrozby a toho, ako je dané aktívum voči nej zraniteľné (9, s. 94).

### 1.4.1 Pojmy spojené s analýzou rizík

Medzi základné pojmy môžeme zaradiť **Aktívum**, **hrozba**, **zraniteľnosť** a **riziko**. Aktívum a hrozba sú popísané vyššie, z toho dôvodu sa tu budem venovať už len zraniteľnosti a riziku.

#### 1.4.1.1 Zraniteľnosť

Zraniteľnosťou je možné nazvať nedostatok, slabinu alebo iný analyzovaný stav aktíva, ktorý je potencionálne zneužiteľný hrozbou, čo má za následok uvedenie aktíva do nežiaduceho stavu. Zraniteľnosť, sa dá v skratke definovať ako veličina, ktorá vyjadruje to, ako je citlivé aktívum na pôsobenie danej hrozby (9, s. 95).

Zraniteľnosť vzniká pri interakcii aktíva a hrozby. Danú veličinu ohodnocujeme na základe úrovne zraniteľnosti. Tá závisí na dvoch faktoroch, ktorými sú citlivosť a kritičnosť aktíva (9, s. 95).

#### 1.4.1.2 Riziko

Riziká sú spôsobené vzájomným pôsobením hrozieb a aktív. Z toho dôvodu sa pri analýze neberú do úvahy hrozby, ktoré nijakým spôsobom nepôsobia na aktívum.

*„…riziko vyjadřuje míru ohrožení aktiva, míru nebezpečí, že se uplatní hrozba a dojde k nežádoucímu výsledku vedoucímu ke vzniku škody.“* (9, s 96)

Riziko, podobne ako zraniteľnosť, je merané pomocou úrovne rizika. Úroveň rizika je definovaná hodnotou aktíva, úrovňou hrozby a zraniteľnosťou aktíva. Túto úroveň je možné znižovať len za pomoci protiopatrení. Ich náklady by však nemali byť vyššie, ako škoda, ktorá by mohla byť napáchaná dopadom hrozby (9, s 96).

## 1.5 Internet

V dnešnej, informáciami nabitej dobe, berieme technologické zdroje týchto informácii ako samozrejmosť. Ku príkladu, chcete vedieť kedy ide autobus, ako sa dostať do najbližšej predajne s elektronikou alebo si chcete kúpiť lístok na vlak? Tieto a ďalšie pre nás úplne bežné a prirodzené činnosti bývajú založené práve na internete a technológiách s ním spojené (10, s. 3).

Internet môžeme považovať za aktuálne najväčší technický systém, aký ľudstvo vytvorilo. Jeho súčasťou sú milióny počítačov, notebookov, telefónov, tabletov, herných konzol, kamier a v neposlednej rade miliardy užívateľov ktoré ich využívajú (3, s. 23).

Internet je počítačová sieť, prepojujúca stovky miliónov počítačových zariadení rozmiestnených po celom svete. Pôvodne počítačové siete tvorili predovšetkým veľké pracovné stanice, ktoré boli založené na operačnom systéme Linux, a servery slúžiace na úschovu a distribúciu informácii. V prítomnosti začínajú dominovať zariadenia uľahčujúce úplne bežné a základné činnosti ako sú notebooky, tablety, smartphony, kamery, automobily, zabezpečovacie systémy a taktiež aj zariadenia určené na zábavu. Takéto koncové uzly, alebo inak aj systémy, sú prepojené prostredníctvom komunikačných liniek. Takéto linky môžu byť tvorené fyzickými médiami, ako sú kabely rôznych prevedení, či už optické alebo metalické, alebo médiami ako sú ku príkladu rádiové vlny (3, s. 24).

### 1.5.1 Prostriedky prepojujúce sieť

Koncové uzly bývajú prepojené prostredníctvom komunikačných liniek a paketových prepínačov. **Komunikačné linky** sa dajú deliť nie len podľa spomínaného prevedenia fyzického média, ale taktiež aj podľa prenosovej rýchlosti. Takáto rýchlosť sa meria v bitoch za sekundu. Komunikačné linky prenášajú takzvané **pakety**, ktoré obsahujú potrebnú informáciu. Tieto pakety sú pomocou linky predávané prepínačom paketov.

Existuje veľké množstvo druhov **prepínačov paketov**, avšak najčastejšími typmi sú smerovače, inak aj routery, a prepínače, inak aj switche. Tieto zariadenia následne, po spracovaní a prípadnom nasmerovaní pakety posielajú na ďalšiu komunikačnú linku a tak dookola. Takéto prepojenie medzi odosielajúcim uzlom, a prijímajúcim, koncovým uzlom sa nazýva **trasa** (3, s. 26).

### 1.5.2 Protokoly

Koncové uzly a systémy využívajú k spracovaniu paketov protokoly. Medzi najdôležitejšie protokoly patrí TCP (Transmission Control Protocol), a IP (Internet Protocol). Internet Protocol ma na starosti určenie formátu paketov pudiacich medzi smerovačmi a koncovými uzlami. Z dôvodu dôležitosti práve týchto dvoch protokolov začalo dochádzať k súhrnnému označeniu TCP/IP protokol (3, s. 26).

Protokol v podstate definuje „*formát a pořadí zpráv vyměňovaných mezi dvěma nebo více komunikujícími subjekty a také akce, které budou provedeny při odeslání nebo přijetí zprávy nebo při výskytu jiné události*“. (3, s. 29)

Bližšie informácie o TCP/IP a protokoloch sú uvedené v časti TCP/IP a náležitých podkapitolách.

Každá činnosť, ktorá sa udeje na internete, a prebieha za účasti viac ako jedného uzlu, sa riadi protokolom. Protokoly sú implementované na každej vrstve. Ku príkladu, na fyzickej vrstve, čiže na spojení dvoch fyzicky pripojených počítačov, riadi protokol tok bitov medzi danými zariadeniami. Na vrstve linkovej zasa dochádza k riadeniu rýchlosti prenosu paketov medzi uzlami a na vrstve sieťovej dochádza k určovaniu cesty paketov k cieľu (3, s. 29).

Viac sa téme vrstiev a ich delenia podľa referenčného ISO/OSI modelu budem venovať v ďalších častiach práce.

Dôležitou úlohou protokolov je štandardizácia a kompatibilita zariadení, a tým umožnená ich komunikácia. Z toho dôvodu vznikli štandardy a normy, ktoré zaisťujú, že systémy budú skutočne kompatibilné (3, s. 26).

### 1.5.3 Sieťové služby

Internet je nutné popisovať taktiež z hľadiska infraštruktúry, ktorá umožňuje poskytovanie služieb aplikáciám. Medzi takéto aplikácie na príklad patrí elektronická pošta, sociálne siete, IP telefonovanie, streamovanie, zdieľanie súborov a obrovské množstvo ďalších. Takéto aplikácie sa nazývajú **distribuované aplikácie**. Takýto názov dostali práve vďaka tomu, že obsluhujú viacero koncových uzlov, medzi ktorými „distribuujú“ dáta a informácie (3, s. 31).

Viac informácii o sieťových službách bude obsahovať téma TCP/IP a jej podkapitoly.

## 1.6 Porty

Internetové, prenosové protokoly, ako sú TCP a UDP, používajú pre identifikáciu aplikácii čísla portov. Takéto čísla portov sú následne po spojení s IP adresou zariadenia uzavreté do takzvaného socketu. Tieto prvky následne zaisťujú unikátnu identifikáciu pre odosielajúce zariadenie, ale taktiež aj to, o akú aplikáciu ide (10, s. 477).


**Obrázok č. 1 TCP/UDP socket**
(Zdroj: Vlastné spracovanie podľa 10, s. 477)

Čísla portov, ako také môžeme rozdeliť do troch častí. Tými sú **Dobre Známe Porty**, **Registrované Porty** a **Dynamické Porty**, inak aj **Privátne Porty**. Každá z týchto častí existuje na určitom rozsahu portov. U Dobre Známych Portov je to rozsah čísiel portov v rozmedzí 0 až 1023. Registrované Porty fungujú v rozmedzí 1024 až po 49151. Na záver, Dynamické/Privátne Porty zase používajú čísla portov od 49152 až po 65535. Príkladom Dobre Známych Portov a protokolov, ktoré ich bežne používajú sú porty 21 – ftp, 22 – ssh, 23 – telnet, 80 – http a podobne (10, s. 477).

Viac sa téme konkrétnych protokolov venujem v kapitole TCP/IP a priľahlých podkapitolách.

## 1.7 Vrstvy ISO/OSI modelu

Sieť a internet obecne tvorí obrovské množstvo prvkov, ktoré spolu tvoria veľmi komplikovaný systém. Pre potrebu spolupráce a prepojenia veľkého množstva heterogénnych systémov je nutné siete usporadúvať do štandardizovanej architektúry. Za týmto účelom vznikol ISO/OSI model, ktorý siete rozdeľuje do vrstiev.

Referenčný model ISO/OSI je obecným modelom definujúcim sieťovú komunikáciu. Tvorí ho sedem vrstiev, ktoré navzájom na seba nadväzujú. Vrstvy vieme rozdeliť na dve časti. Prvé štyri vrstvy sú určené na prenos signálu. Zvyšné, vrchné tri vrstvy sú zase orientované aplikačne (4, s. 13).

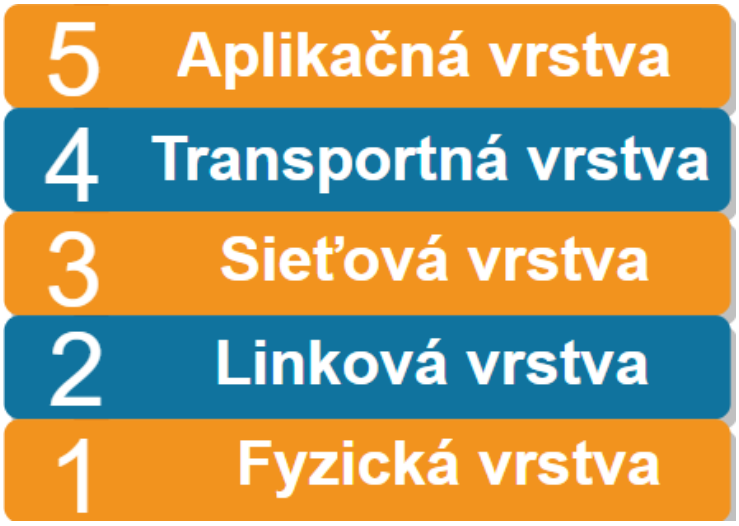

**Obrázok č. 2 ISO OSI model**
(Zdroj: Vlastné spracovanie podľa 4, s. 13)

Takáto vrstvená architektúra nám poskytuje podrobne definované konkrétne súčasti veľkého a komplexného systému ako je internet. Rovnako tak umožňuje modularitu ktorá závisí na poskytovaných službách vrstvy pod sebou, a na službách ktoré poskytuje vrstve nad sebou. Vďaka tomu je možné pri zachovaní takýchto **služieb** na vstupoch a výstupoch meniť samotnú implementáciu jednej vrstvy bez zmeny vrstiev ostatných (3, s. 59).

### 1.7.1 Vrstvenie protokolov

Z dôvodu potreby zaistenia štruktúry návrhu sieťových protokolov boli nielen protokoly ale aj hardware a software ktorý ich implementuje, usporiadané do vrstiev. Vrstvy sú totožné s vrstvami ISO/OSI modelu a každý protokol patrí práve do jednej vrstvy (3, s. 59).

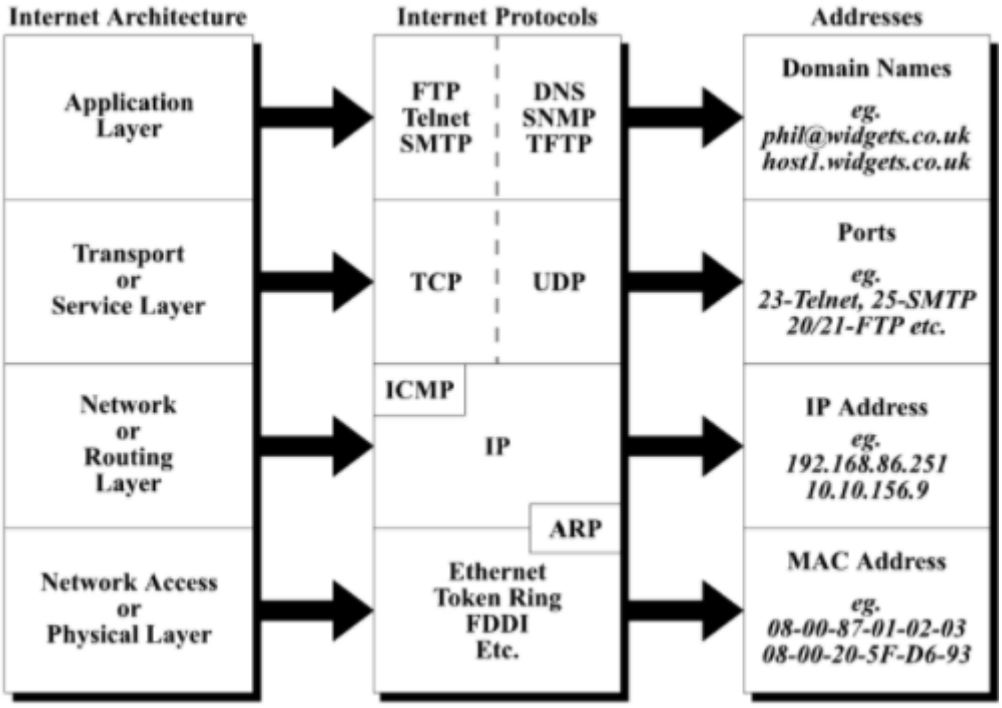
Pre potreby internetových protokolov vznikol päťvrstvový model, ktorý explicitne nevyjadruje vrstvu prezentačnú a relačnú (3, s. 60).




**Obrázok č. 3 Päťvrstvový model internetových protokolov**
(Zdroj: Vlastné spracovanie podľa 3, s. 60)

Samotnú komunikáciu medzi vrstvami zaisťujú **služby**, ktoré vrstva poskytuje vrstve nad sebou (3, s. 60).

Priame naviazanie protokolov na jednotlivé vrstvy je možné vidieť na obrázku nižšie. Obrázok obsahuje aj príklady toho, aké adresovanie dané protokoly používajú.




**Obrázok č. 4 Internetové protokoly vo vzťahu s väzbami a adresovaním**
(Zdroj: 11, s. 539)

## 1.8 TCP/IP

TCP/IP, alebo inak aj súprava internetového protokolu, sa zameriava na poskytovanie zmysluplnej komunikácie medzi počítačmi. IP (Internet Protocol) je bez debát najdôležitejším protokolom v danej oblasti. Bez preháňania sa dá povedať, že všetky ostatné protokoly sú závislé práve na IP protokole. Ďalším, veľmi dôležitým protokolom, je TCP (Transmission Control Protocol), na ktorom závisí spoľahlivé doručovanie dát a informácii (10, s. 3).

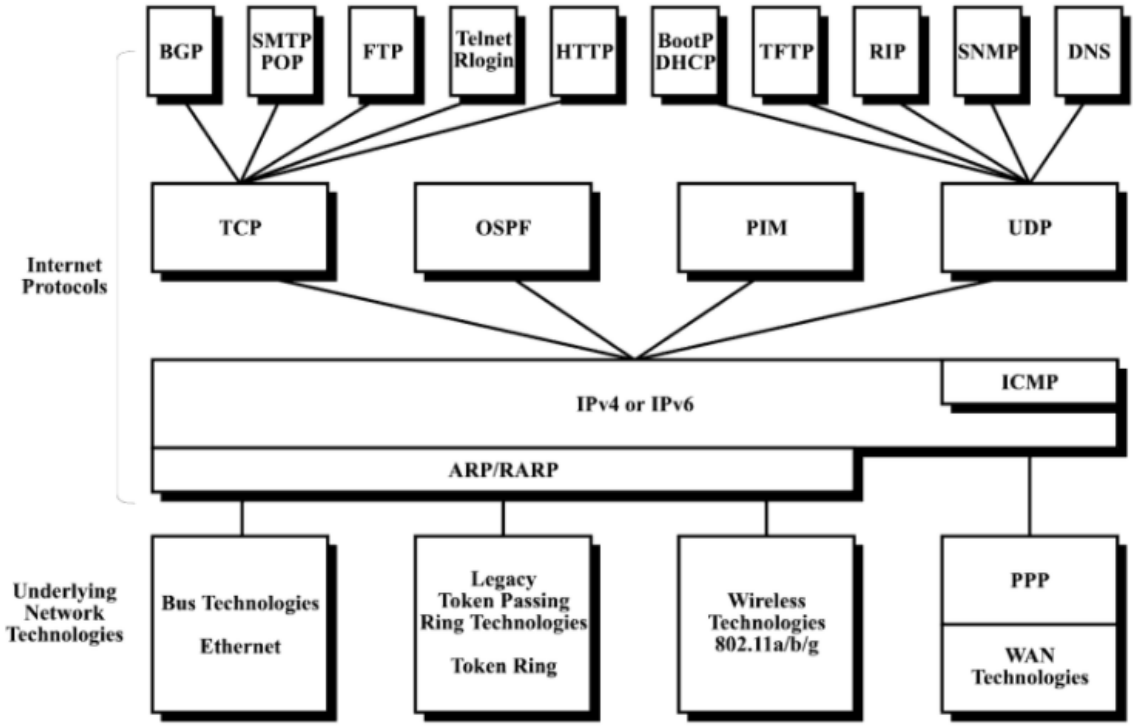
TCP/IP vznikol ako následok snahy vývojárov o spoľahlivý ale zároveň jednoduchý spôsob komunikácie medzi počítačovými sieťami a zariadeniami v nich. Medzi skupinou, ktorá na vývoji danej sady pracovala, bola aj University of California. Práve to bol kľúčový element k tomu, aby došlo k rýchlemu a rozsiahlemu zavedeniu TCP/IP do obehu. Dôvodom bolo to, že väčšina univerzít, čo znamená aj väčšina užívateľov počítačového prepojenia v osemdesiatych rokoch, používala operačný systém UNIX. Práve začlenením TCP/IP do systému UNIX došlo k tomu, že dochádzalo k masívnemu testovaniu danej súpravy na univerzitách, a to dokonca často aj bez toho aby o tom vedeli.

Vďaka tomu došlo k zaisteniu rýchleho, efektívneho a lacného začlenenia TCP/IP medzi už existujúce technológie tej doby (10, s. 4).

### 1.8.1 TCP/IP protokoly

Sada internetového protokolu sa skladá z veľkého množstva rôznych protokolov. Protokoly ako také, bývajú obecne zoskupené podľa funkcií, aplikácii, služieb a tak podobne. Pre upresnenie je treba uviesť, že sada internetového protokolu obecne nedefinuje nové podliehajúce sieťové technológie. Práve naopak, sada používa už existujúce, štandardizované a overené sieťové technológie (10, s. 4).

Náležitý príklad protokolov, vrstiev a ich prepojenia je možné vidieť na nasledujúcom obrázku.




**Obrázok č. 5 Sada internetových protokolov**
(Zdroj: 10, s. 5)

Jednotlivým protokolom a ich definícii sú venované nasledujúce podkapitoly.

#### 1.8.1.1 DNS

Obecne sa dá prehlásiť, že pamäť človeka a jeho schopnosť uchovávania je pomerne obmedzená. A to je tvrdenie, ktoré sa vzťahuje už na bežné veci, ktorým rozumieme a dávajú nám zmysel. Teraz si predstavte že by sme boli nútení si pamätať nie len nám

prirodzené alfanumerické názvy webových stránok, ale ich IP adresy. To znamená, že miesto google.com by ste si museli pamätať adresu 172.217.23.238.

Práve z vyššie spomenutých dôvodov vznikol protokol DNS (Domain Name System), ktorý má na starosti preklad doménových mien na IP adresy, ktorým naše počítače rozumejú. Takýto proces býva zastrešovaný Name Serverom (11, s. 539).

#### 1.8.1.2 Telnet

Telnet protokol používa pre svoje funkcie TCP spojenie, ktoré bežne beží na porte 23. Takéto spojenie zaisťuje spoľahlivé prostredie pre zdieľanie dát. Základnou funkciou, ktorú Telnet poskytuje je obojstranná komunikačná cesta, ktorá spája terminálové zariadenia s terminálovo-orientovanými procesmi na hostiteľskom zariadení. To umožňuje prácu s danými procesmi aj bez toho, aby mali užívatelia fyzický prístup k hostiteľskému zariadeniu. Takéto prepojenie následne vytvára ilúziu, že je užívateľ lokálne so zariadením spojený (11, s. 585).

Telnet protokol definuje štandardizovanú metódu pre spojenie viacerých systémov dokopy. V dňoch vzniku daného protokolu a internetu celkovo v podstate existovali len dva protokoly, a tými boli práve Telnet a FTP protokoly. Úprimne sa dá prehlásiť, že tieto dva protokoly tvoria hlavné aplikačné protokoly, ktoré boli základom pre vznik HTTP protokolu, ktorý je základom pre „World-Wide Web“, ako ho v dnešnej dobe poznáme (11, s. 605).

#### 1.8.1.3 FTP

Jedným zo základných motivátorov pre vznik internetu bola túžba po zdieľaní dát. Dáta ako také tvoria súbory (files), z toho dôvodu časom vznikol File Transfer Protocol (FTP) (11, s. 607).

FTP, ako už bolo aj v popise Telnetu spomenuté, bol jedným zo základných protokolov v internetovej sade protokolov. Vznikol za účelom zdieľania súborov a taktiež aj ako podpora pre používanie vzdialených počítačov (11, s. 614).

#### 1.8.1.4 HTTP

HTTP (Hypertext Transfer Protocol) začal byť používaný v roku 1990 ako základný stavebný kameň pre „World-Wide Web“. Obecne funguje HTTP v server-klient prostredí. Bežne býva takéto prostredie zastúpené na príklad webovým prehliadačom (klientom) na strane užívateľa a serverom, ktorý uchováva všetky dáta a informácie.

Tieto dáta môžu byť distribuované na príklad na restovom rozhraní, ktoré je založené na systéme dotaz/odpoveď (11, s. 645).

Na obrázku nižšie je možné vidieť základný princíp komunikácie dotaz-odpoveď. UA zastupuje tzv. „User Agent“, a teda na príklad prehliadač a OS zastupuje server.


**Obrázok č. 6 Jednoduchý dotaz/odpoveď mechanizmus**
(Zdroj: 11, s. 645)

Najčastejšie býva HTTP implementované cez TCP a používa port 80. Takéto spojenie medzi klientom a serverom požaduje spoľahlivý prenos dát, čo zaisťuje práve TCP. Taktiež aj to, že spojenie býva väčšinou pomerne krátke a žije len počas konkrétneho dotaz/odpoveď reťazca (11, s. 646).

Pôvodný protokol HTTP neposkytuje šifrovanú komunikáciu. To znamená, že útočník, ktorý má priestor a možnosť odpočúvať našu komunikáciu, tak má priamy prístup k dátam ktoré zasielame, a to v čitateľnej forme. Medzi takýmito dátami si predstave svoje rodné číslo, číslo kreditnej karty a ďalšie súkromné informácie. Z toho dôvodu vznikol protokol HTTPS, ktorý je v podstate to isté ako HTTP, avšak využíva prepojenie cez TLS (Transport Layer Security). TLS poskytuje end-to-end šifrovanie medzi serverom a klientom a tým spôsobom zaisťuje to, že napriek tomu že útočník získa vašu komunikáciu, tak nebude pre neho čitateľná. Takéto spojenie väčšinou požíva port 443 (11, s. 652).

#### 1.8.1.5 SMTP

Ako už boli protokoly popísané vyššie, Telnet je kritickou súčasťou sieťového manažéra, ktorému umožňuje vzdialenú správu zariadení a serverov, FTP zase poskytuje obrovské možnosti v oblasti zdieľania dát a súborov, HTTP nám umožňuje komunikáciu cez internet tak ako ho v dnešnej dobe poznáme. Netreba však zabúdať na jednu z najzakladanejších a najrozsiahlejšie používaných funkcii, a tým je mail. Určite sa nájde niekto kto by mohol nesúhlasiť, a použil by argumentáciu že v dnešnej dobe sa čím ďalej tým viac prechádza na tzv. „četovú“ komunikáciu, poprípade priamo video hovory. Preto

treba brať dotyčné tvrdenie skôr z historického hľadiska a toho ako sa internet rokmi vyvíjal.

Email a danú technológiu môžeme rozdeliť na dve hlavné časti – mechanizmus na zasielanie mailov a mechanizmus na ich prijímanie. Jedna z daných častí zaisťuje doručenie mailu z nášho zariadenia na server. Práve táto časť je zastúpená SMTP (Simple Mail Transfer Protocol). Druhá časť zase zaisťuje roztriedenie danej pošty na serveri a distribúciu na správne mailové adresy. Tento proces je zastúpený pomocou POP (Post Office Protocol), poprípade alternatívne IMAP (Internet Message Access Protocol) (11, s. 655).

Na obrázku nižšie je znázornená SMTP komunikácia medzi odosielateľom, zväčša klientom a prijímateľom, ktorého zastupuje server. Takéto spojenie funguje vďaka tomu, že server počúva na porte 25, ktorý umožňuje klientom TCP spojenie.


**Obrázok č. 7 Príklad jednoduchej SMTP výmeny**
(Zdroj: 11, s. 655)

#### 1.8.1.6 SNMP

SNMP (Simple Network Management Protocol), ako už názov napovedá, jedná sa o pomerne jednoduchý protokol, určený na spravovanie sieťových zariadení. SNMP berie zariadenie ako skupinu objektov, ktoré majú určité vlastnosti a nastavenia. Komunikácia medzi takzvaným agentom manažérom a spravovaným zariadením je opäť založená na báze dotaz/odpoveď. V prípade potreby sa vie manažér dopytovať priamo na požadované zariadenie. To mu následne zašle všetky požadované informácie. V prípade že sa manažér rozhodne meniť nastavenie zariadenia, tak sa opäť manažér dopytuje na dané zariadenie, avšak pridáva aj identifikátor objektu, ktorý chce zmeniť. Spolu s identifikátorom prirodzene zasiela aj hodnotu, s ktorou chce daný objekt modifikovať (11, s. 713).

Pre úplnosť je taktiež nutné uviesť, že SNMP nepracuje len na princípe dotaz/odpoveď, ale taktiež používa aj na príklad takzvané „Traps“. Trap v slovenskom jazyku znamená

pasca, a ako už názov napovedá, tak sa jedná o správy, ktoré sú manažérovi zasielané na základe nezvyčajných aktivít, poprípade priamo aktivít, na ktoré sú pasce nastavené aby reagovali (11, s. 713).

## 1.9 Nmap

Nmap, alebo inak aj „Network Mapper“ je bezplatná open-source funkcionalita, slúžiaca na prieskum a audit siete. Využitie taktiež poskytuje v oblasti manažovania a spravovania sieťových zariadení. Taktiež poskytuje nástroje podporujúce pravidelné updaty zariadení a ich služieb (7).

Nmap funguje na princípe zasielania IP paketov, vďaka ktorým dokáže identifikovať to, aké zariadenia sú dostupné na sieti, aké služby a v akých verziách poskytujú, na akom OS bežia, aké filtre poprípade firewall sú implementované a celkovo poskytuje náhľad na veľké množstvo ďalších užitočných informácii (7).

Nmap vznikol primárne za účelom skenovania veľkých sietí, avšak umožňuje aj skenovanie jednotlivých zariadení. Taktiež bol vyvinutý pre všetky známejšie operačné systémy, ako sú na príklad Windows, Linux alebo v neposlednej rade Mac OS (7).

## 1.10 Kali Linux

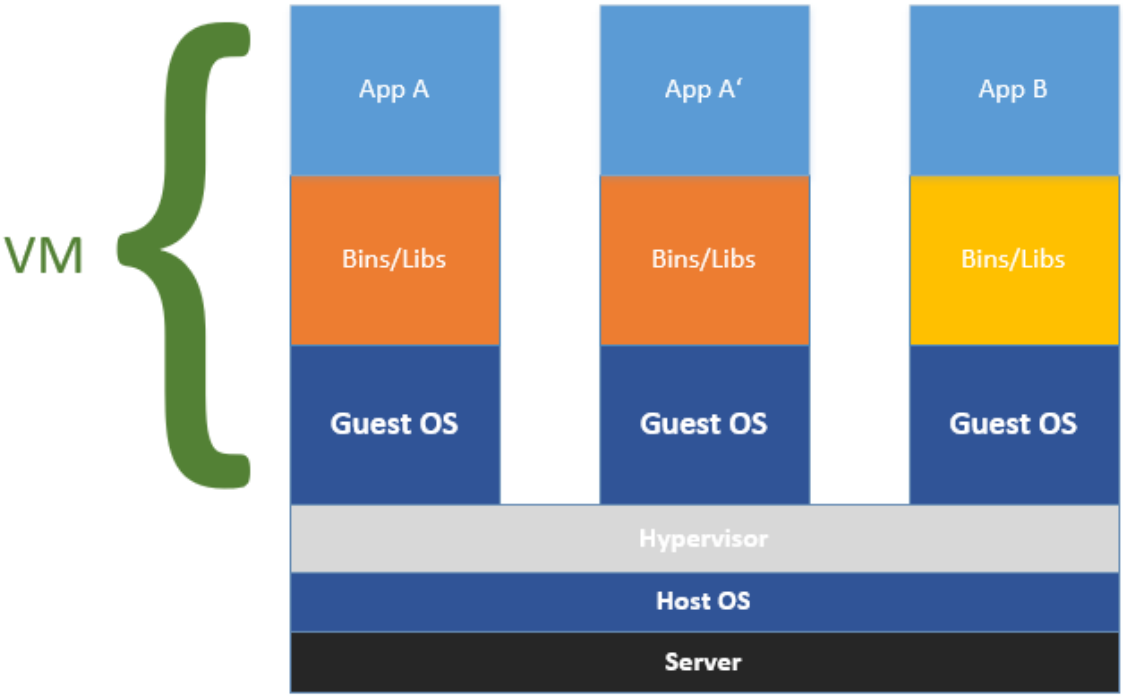
Kali Linux je open-source linux založený na báze Debianu. Primárne je zameraný na rozsiahlejšie a pokročilejšie penetračné testy a bezpečnostné audity v oblasti ICT. Kali Linux obsahuje veľké množstvo prostriedkov a nástrojov, zameraných práve na činnosti ako sú penetračné testovanie, bezpečnostné prieskumy, audity a podobne. Ako už z pojmu open-source vychádza, jedná sa o voľne dostupný a bezplatný multiplatformový nástroj (8).

## 1.11 Virtuálny stroj

Virtuálny stroj (Virtual machine), ďalej len VM, je virtuálne prostredie, ktoré funguje ako virtuálny počítač s vlastným procesorom, pamäťou, internetovým rozhraním a úložiskom, vytvoreným na fyzickom hardware počítača. Proces rozdeľovania a distribúcie zdrojov z hardware má na starosti software nazvaný hypervisor (5, 9-11).

Fyzický stroj, na príklad počítač, ktorý poskytuje všetky fyzické zdroje pre virtualizáciu sa nazýva host. Naopak, VM, ktoré používajú zdroje daného počítača sa nazývajú guests. Hypervisor, ako už bolo spomínané má za úlohu práve rozdelovanie výkonu, ako na príklad z procesoru, alebo pamäte hosta medzi všetky VM a teda guestov. Takéto VM sú od zdrojového systému oddelené a je možné, aby existovalo viacero VM na jednom počítači. Práve toto umožňuje to, aby dokázal jeden stroj, na príklad server, obsluhovať viacero heterogénnych systémov, ktoré nedokážu pracovať na rovnakom operačnom systéme (5, 9-11).

Ďalšou veľkou výhodou je prenositeľnosť celého guesta medzi zariadeniami. Takýto prenos umožňuje preniesť celý systém, ktorý obsahuje všetky potrebné programy, funkcionality a nastavenia nutné pre správne fungovanie celého prostredia a jeho kľúčových aplikácii. Týmto spôsobom je možné zaistiť kompatibilitu za takmer akýchkoľvek podmienok a taktiež to zamedzuje nutnosti konfigurácie a debugovania riešenia po každom prenose a nasadení na novom zariadení (5, 9-11).




**Obrázok č. 8 Grafické zobrazenie schémy fungovania VM**
(Zdroj: Vlastné spracovanie podľa 6)

Ako je možné vidieť na obrázku, tak VM používajú pre každé virtuálne prostredie vlastný operačný systém a taktiež aj knižnice. Týmto spôsobom dochádza k ich častej duplicite,

ktorá zbytočne zahlcuje systém nie len z hľadiska úložiska, ale taktiež aj pamäte a výkonu zariadenia.

## 1.12 Docker

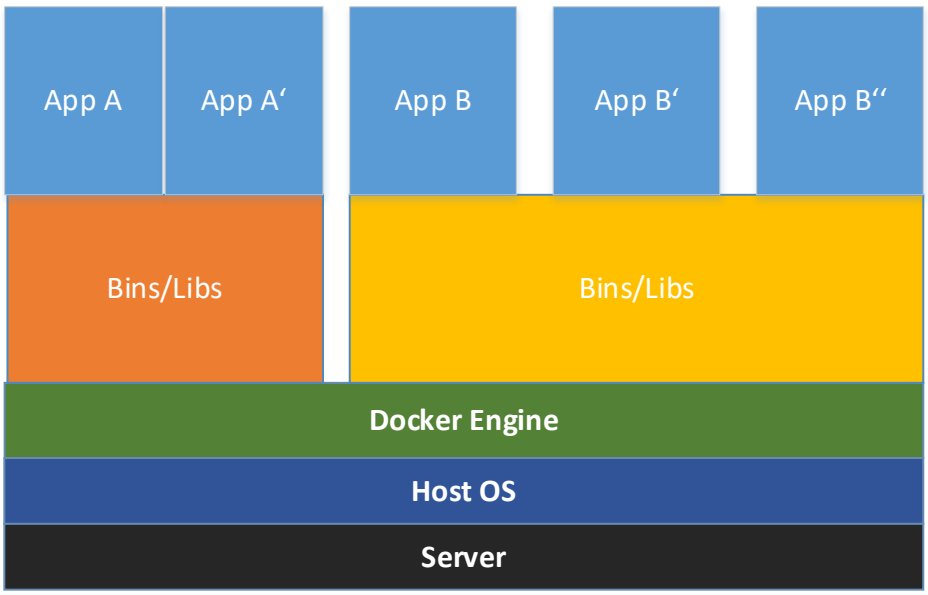
Docker je softwarová open-source platforma vyvinutá pre budovanie aplikácii. Platforma je založená na báze tzv. kontajnerov. Jedná sa o malé individuálne prostredia, ktoré spolu zdieľajú kernel operačného systému, avšak mimo to fungujú všetky samostatne a izolovane. Kernel je počítačový program, ktorý zaisťuje komunikáciu medzi hardwarom a softwarom zariadenia (6).

Pre pochopenie toho, čo je kontajner a na čo slúži, je nutné najskôr pochopiť motiváciu jeho vzniku. V dnešnej dobe sa kladie vysoký dôraz na to, aby bolo možné pri vývoji držať aplikácie oddelené, ale zároveň na jednom zariadení. Cieľom takejto snahy je to, aby nedochádzalo ku konfliktom aplikácii pri vývoji (6).

Takýto konflikt by mohol vzniknúť v prípade, že dve, alebo viacero aplikácii zdieľa jeden podporný program, na príklad NodeJS. Pri novom update v rámci jednej aplikácie dôjde k zmene požiadavkou na takýto program a začne požadovať vyššiu verziu, avšak druhá aplikácia, ktorá rovnako používa NodeJS, nevie pracovať s vyššou verziou. V takomto prípade nemá vývojár možnosť mať obe verzie na jednom zariadení zároveň a teda môže jedna aplikácia obmedzovať aplikáciu druhú.

Riešením takéhoto problému sú na príklad VM, ktoré popisujem v kapitole vyššie. Ako už bolo spomínané v danej kapitole, tak každá VM požaduje vlastný OS, čo spôsobuje razantný nárast potrebnej pamäte a taktiež nárokov na zdroje.

Naopak, ako je možné vidieť na obrázku nižšie, kontajnery síce fungujú na separovanej báze, avšak zdieľajú spoločný OS a taktiež aj kernel. Vďaka tomu zaberajú takéto kontajnery neporovnateľne menej miesta, to zaručuje ľahkú portabilitu a taktiež znižujú požiadavky na zdroje na minimum (6).




**Obrázok č. 9 Grafické zobrazenie schémy fungovania docker kontajnerov**
(Zdroj: Vlastné spracovanie podľa 6)

# 2 ANALÝZA SÚČASNÉHO STAVU

Ako už bolo v úvode spomínané, analýza prebieha u klienta spoločnosti C SYSTEM CZ a.s., ktorý nebude z dôvodov bezpečnosti menovaný. Z toho dôvodu je v celej práci označovaný len ako klient.

Prvú časť tejto kapitoly venujem charakteristike spoločnosti C SYSTEM CZ a.s., ktorá obsahuje základné informácie a taktiež aj zámer firmy s výstupom diplomovej práce. Zároveň sa táto kapitola zameriava na samotnú analýzu počítačovej siete klienta a potencionálnych hrozieb, ktoré by mohli na ňu pôsobiť.

## 2.1 Základné informácie o spoločnosti

**Názov spoločnosti:** C SYSTEM CZ, a.s.

**Právna forma:** Akciová spoločnosť

**Sídlo:** Otakara Ševčíka 56, 636 00

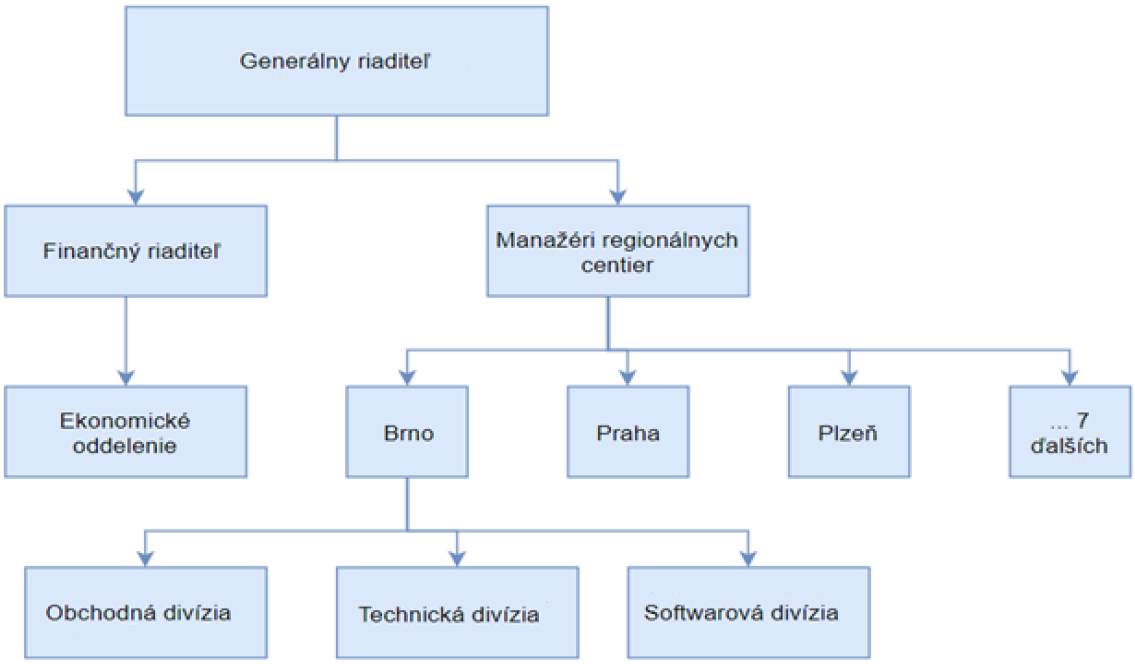
**Rok založenia:** 1996 (2).

**Obrázok č. 10 Logo organizácie**
(Zdroj: 2)

### 2.1.1 Predstavenie organizácie

Spoločnosť vznikla v roku 1996 v Brne. Jej hlavným zámerom bolo dodávanie hardwarových riešení nie len pre podnikateľské subjekty ako sú dodávatelia komunikačných služieb, ale taktiež pre štátnu správu. Časom sa začala spoločnosť výraznejšie zameriavať na dodávanie či už dielčích alebo komplexných informačných systémov. Časom sa CSYSTEM rozrástol o ďalších 8 pobočiek rozmiestnených po celej Českej Republike. V poslednom období bolo pre spoločnosť najväčším krokom vytvorenie novej IT divízie zameranej na zdravotnícke informačné systémy (2).

## 2.2 Organizačná štruktúra




**Obrázok č. 11 Organizačná štruktúra spoločnosti**
(Zdroj: Vlastné spracovanie)

## 2.3 Analýza požiadaviek spoločnosti

Spoločnosť C SYSTEM CZ a.s. poskytuje vlastné softwarové riešenia nie len súkromným subjektom, ale taktiež aj verejným inštitúciám, ako sú na príklad nemocnice. Pri práci na nemocničných systémoch často dochádza k spolupráce viacerých spoločností na jednom produkte a teda je nutné zdieľať často súkromné a citlivé dáta. Keďže spadajú nemocnice pod kritickú infraštruktúru, tak je potrebné aby niesol zodpovednosť za citlivé dáta celý dodávateľský reťazec. Mnoho spoločností to aktuálne rieši len formou zmlúv, SLA (service level agreement) a obdobných dohôd, ktoré majú z legislatívneho hľadiska chrániť spoločnosti v prípade úniku dát a informácii. Spoločnosť C SYSTEM si zakladá na kvalite poskytovaných služieb a preto hľadá aj iné spôsoby, ktoré nechránia len samotnú spoločnosť, ale taktiež aj klientov. Z toho dôvodu sa rozhodli pre hľadanie riešenia, ktoré dokáže zabezpečiť aspoň jednoduchý, rýchlo a ľahko implementovateľný nástroj pre základné skenovanie siete a hľadanie možných zraniteľností, ktoré v danej počítačovej sieti môžu pôsobiť.

## 2.4 Analýza technických prostriedkov

V aktuálnej časti práce budú analyzované možnosti a prostriedky spoločnosti a jej partnerov pre vývoj prostriedku pre analýzu sietí. Primárne sa tu zameriavam na riešenie toho, v akom prostredí a za pomocí akých prostriedkov bude daný nástroj vyvinutý.

### 2.4.1 Voľba skenovacieho softwaru

Pri výbere vhodného nástroju pre skenovanie siete bolo stanovených viacero základných podmienok:

- Flexibilita
- Prenositeľnosť
- Jednoduchosť
- Podpora a dokumentácia
- Opensource prostriedok

Po stanovení podmienok a požiadaviek som začal s výberom vhodného nástroju. Medzi prvými nájdenými a taktiež aj najznámejšími figuruje Nmap. To môže byť odôvodnené viacerými faktormi. Či už to bolo spopularizovaním vo filmoch ako na príklad The Matrix Reloaded, alebo veľkým množstvom získaných ocenení. Ku príkladu Nmap získal ocenenie „Security Product of the Year“ (7).

Z hľadiska vlastností, spĺňa nástroj všetky stanovené podmienky. Jednotlivé podmienky a to, ako ich nástroj spĺňa bude popísané nižšie.

**Flexibilita:** Nmap podporuje desiatky pokročilých metód pre mapovanie sietí. Takéto metódy a techniky umožňujú reagovať a obchádzať prekážky ako sú rôzne IP filtre, firewally, routre a podobne. Podporuje taktiež TCP a UDP skenovanie, zisťovanie OS, verzii služieb a množstvo ďalších úkonov (7).

**Prenositeľnosť:** Nmap je podporovaný väčšinou operačných systémov, to zaručuje to, že ho bude možné používať a prenášať medzi veľkým spektrom zariadení. Taktiež to, že je v základe implementovaný vo viacerých systémoch, ako na príklad Kali linux, poprípade to, že je veľmi ľahko stiahnuteľný z verejnej repository, zaručuje vysokú úroveň portability (7).

**Jednoduchosť:** Nástroj síce umožňuje aj veľmi zložité operácie, avšak taktiež poskytuje veľmi základné a jednoduché príkazy a knižnice pre základné skenovanie a audit (7).

**Podpora a dokumentácia:** Keďže sa jedná o globálne používaný prostriedok, tak k Nmap existuje veľké množstvo zdrojov, návodov a dokumentácii. Podporu zase zaisťuje rozsiahla komunita vývojárov, ktorý aktívne reagujú na nahlasované chyby a taktiež pokračujú vo vývoji (7).

**Opensource prostriedok:** Nmap je opensource nástroj, ktorý vďaka tomu zaisťuje to, že je bezpečný, že nebude dochádzať k únikom alebo zberu dát a taktiež aj to, že bude pokračovať jeho podpora a vývoj (7).

Vďaka týmto všetkým aspektom a taktiež aj vďaka tomu, že plánujem tvoriť len obmedzený nástroj založený na práci s príkazovým riadkom, som došiel k rozhodnutiu že pre analýzu použijem práve nástroj Nmap (7).

### 2.4.2 Voľba platformy

Spoločnosť a taktiež aj jej zákazníci a partneri pracujú prevažne na serveroch na platforme windows. Z toho dôvodu je obmedzená možnosť použitia linuxových prostriedkov, ktoré sú pre používanie NMAP funkcionalít omnoho vhodnejšie.

Vývoj, ktorý neprebieha priamo na serveri, prebieha v súčinnosti s aplikáciou docker. Táto aplikácia uľahčuje zdieľanie riešení medzi jednotlivými vývojármi, testermi a kýmkoľvek ďalším, kto má prístup do firemnej repository. Jedným z nedostatkov tohto riešenia je to, že využíva Hyper-V, ktoré naopak zamedzuje fungovaniu iných vývojárskych prostriedkov, ako je na príklad virtualbox.

Pokiaľ si stanovíme operačný systém linux ako podmienku, tak do úvahy prichádza niekoľko možností. Jednou z nich je využitie prostriedkov ako je spomínaný virtualbox. Ten, ako už bolo spomenuté nefunguje v kombinácii s Hyper-V, ktorý je zase potrebný pre prácu s dockerom. Tento problém je možné vyriešiť pomocou dual-boot systému, kedy máme na výber, či chceme spustiť systém s Hyper-V alebo bez neho. Druhou možnosťou je použitie priamo prostredia docker, čo však opäť prináša mnoho negatívnych aspektov.

Dôslednejšia analýza jednotlivých riešení bude prebiehať v nasledujúcich podkapitolách.

#### 2.4.2.1 Analýza využitia virtualbox

Výhodou použitia virtualbox je grafické rozhranie, a aj to že je možné kali-linux získať už v pomerne dobre nakonfigurovanej forme. Rovnako tak prichádza s implementovanou väčšinou potrebných funkcionalít. Taktiež použitie virtualboxu je pomerne jednoduché, a v prípade že je užívateľ oboznámený s OS linux, tak nebudú nutné nijak vysoké nároky na školenie a oboznamovanie sa s prostredím.

Nevýhodou, ako už bolo spomínané je práve obmedzenie fungovania aplikácie docker, ktorá je pre vývoj elementárne dôležitá. Takýto nedostatok sa dá riešiť pomocou už spomínaného dual-boot systému, ktorý však vyžaduje reštartovanie zariadenia. To by mohlo mať negatívny vplyv na efektívnosť a výkonnosť užívateľov popri inak zvyčajnej náplne práce.

Ďalšou nevýhodou je veľkosť prednastaveného virtuálneho zariadenia, ktoré môže mať už v základe tri až štyri gigabajty. Rovnako tak samotné nasadenie daného riešenia do virtualboxu môže trvať v slabších zariadeniach aj desiatky minút.

Na záver nemožno zabúdať na to, že samotný spustený virtualbox dokáže byť pre slabšie zariadenia pomerne náročný a za následok by mohol mať opäť obmedzenie ďalšej práce na zariadení.

#### 2.4.2.2 Bezpečnosť virtualizácie

Pre problematiku bezpečnosti v rámci virtualizácie vznikla norma ISO/IEC 21878:2018, ktorá sa zaoberá postupmi k bezpečnému návrhu a implementáciu virtualizovaných serverov. V rámci normy sa spomínajú nasledujúce riziká:

- Útok na fyzický server
- Útok na hypervizora
- Spustenie nezabezpečeného systému
- Nedostatočná segregácia serverov
- Vyššia chybovosť
- Zle nastavené práva

Všeobecne sa dá zhrnúť, že problematika pri zabezpečení virtuálnych prostredí pramení prevažne z obáv z nedostatku kontroly a dohľadu nad daným zariadením. Táto kombinácia spolu vytvára pocit neznáma, ktorý môže vyvolávať ďalšie obavy.

Z hľadiska neznáma je vhodné spomenúť základné rozdiely medzi fyzickými a virtuálnymi servermi:

- Existencia novej vrstvy správy, ktorá je tvorená hypervizormi, konzolami a rozhraním pre správu.
- Koncentrácia viacerých serverov na jednom fyzickom stroji.
- Nutnosť presunov virtuálnych počítačov medzi fyzickými servermi.

Tieto aspekty vytvárajú práve spomínaný pocit neznáma a nedostatku dohľadu.

Ďalším aspektom v rámci bezpečnosti virtualizácie je **absencia fyzických kontrolných prvkov zabezpečenia**. To je spôsobené tým, že všetky prvky sú virtuálne, či už server, počítač alebo sieť. Jediné miesto, kde je možné umiestniť fyzické kontrolné body, je perimeter virtuálnej infraštruktúry, kde končí virtuálne prostredie a začína fyzické.

Ďalšími bezpečnostnými problémami sú:

- **Slepé škvrny komunikácie** – medzi rôznymi VM na rovnakom hostiteľovi
- **Vzájomné útoky medzi VM a napadnutie hypervizoru** – jeden VM dokáže preniesť nákazu na ďalšie VM na rovnakom hostiteľovi
- **Rôzny úroveň dôveryhodnosti VM** – v rámci jedného hostiteľa dochádza k miešaniu dát s rôznou úrovňou dôležitosti
- **Súperenie o prostriedky** – na príklad pri spustení antivirových prostriedkov dochádza k antivírovej búrke, ktorá obmedzuje výkon systému

Tieto všetky aspekty vo veľkej miere ovplyvňujú náročnosť implementácie práve virtualizovaného riešenia a vytvárajú vysoké nároky na bezpečnostne korektné riešenie.

#### 2.4.2.3 Analýza využitia docker

Jedným zo základných kladov použitia dockeru je veľmi jednoduché riešenie prenosu a implementácie riešení medzi zariadeniami. To je založené na princípe zdieľaných repositories, a teda úložísk, odkiaľ je možné dané riešenie stiahnúť. Taktiež svojim spôsobom zdieľania zdrojov, OS a knižníc medzi kontajnermi šetrí zariadenie a jeho výkon. To umožňuje nasadenie aj na slabšie zariadenia.

Ďalšia dôležitá výhoda sa viaže priamo na prostredie spoločnosti, v ktorom sa aktuálne práve docker používa. To uľahčuje nie len školenie, ale taktiež eliminuje potrebu nasadzovania nového aplikačného prostredia na zariadenia vývojárov.

Výhod by sa dalo nájsť skutočne ešte mnoho, avšak pre úplnú analýzu je nutné uviesť aj nevýhody daného riešenia. Prvou a len veľmi ťažko kompenzovateľnou je absencia grafického rozhrania. Taktiež pomerne silným nedostatkom je to, že sa jedná o vcelku nové riešenie v kombinácii s kali linux a teda neexistuje veľké množstvo zdrojov z ktorých by sa dalo čerpať. Ďalšou vecou je nutnosť inštalácie väčšiny knižníc a programov a mierne náročnejšia prvotná konfigurácia systému.

#### 2.4.2.4 Bezpečnosť kontajnerizácie

V prípade, že dôjde k použitiu dockeru, a teda kontajnerizácie, tak je nutné brať ohľad na viacero bezpečnostných zásad. Tieto zásady sú spracované na základe normy NIST SP 800-190.

**Bezpečnostné hrozby spojené s image.**

**Použitie v maximálnej miere okresaného operačného systému**. Väčšina zdrojových systémov v rámci kontajnerizácie je poskytovaných aj v takzvanej čistej forme. To znamená že neobsahujú ani základné knižnice a programy, ktoré často berieme ako samozrejmosť. Na takýto systém je následne možné nainštalovať a implementovať len tie nástroje, ktoré budeme reálne potrebovať a používať. Vďaka tomu je možné znížiť možnosti útočníka v prípade infekcie kontajneru.

**Pravidelný update image**. Pri spúšťaní bežného systému dochádza k pravidelnej kontrole dostupný aktualizácii, ktoré v prípade potreby stiahne a nainštaluje. Vďaka týmto aktualizáciám dochádza k oprave identifikovaných chýb, ktoré mali často potenciál pre zneužitie a infekciu zariadenia. V rámci dockeru a images k takémuto ničomu automaticky nedochádza. Pre to je nutné to, aby dochádzalo k pravidelnej inštalácii aktualizácii správcom. Tento postup by sa dal automatizovať každým vytvorením kontajneru, avšak to by zabralo pomerne veľa času a tým by sa vo veľkej miere znížila miera pohodlnosti daného riešenia.

**Nastavenie užívateľa pri spúšťaní kontajneru**. Pri spúšťaní kontajneru je možné vždy navoliť to, s ktorým užívateľom sa spustí. Vďaka tomu je možné zamedziť tomu, aby prípadný útočník mohol upravovať kontajner a jeho náležitosti.

**Inštalácia a implementácia len overených programov**. Tento bod je už z časti zahrnutý v prvom bode tohto zoznamu. Presnejšie v použití okresaného OS. Treba však explicitne spomenúť to, že je nutné sa vyvarovať sťahovaniu a inštalácii balíkov a súborov od

neznámych vydavateľov. V prípade že to nebude dodržané, je možné aby sa medzi takýmito súbormi nachádzal aj infikovaný súbor, ktorý môže nepozorovane napádať ostatné kontajneri a taktiež aj hostiteľské zariadenie.

**Zamedzenie vkladaniu nechránených citlivých dát a údajov do image**. Napriek tomu, že to môže znieť absurdne, aby niekto vkladal do image citlivé údaje a následne ich zdieľal, avšak prax ukazuje že sa to deje. Jedným z príkladov toho, kedy sa to deje je spojenie klient-server. V prípade že je server dobre nastavený, tak automaticky požaduje identifikátor klienta. Takýmto identifikátorom býva na príklad aj client secret, čiže jeho heslo. Pre to, aby mal klient v rámci jedného kontajneru šifrované spojenie so serverom v druhom kontajneri je nutné, aby bol v rámci klientského kontajneru prístupný práve daný kľuč. Tento kľúč by bol následne dostupný akémukoľvek užívateľovi danej image, kto by mal dostatočné oprávnenia. Tým by mohlo dôjsť k jeho úniku a následne by sa ako overený klient mohol tváriť ktokoľvek, a tak získavať dáta zo serveru. V takomto prípade je preferovanou možnosťou šifrovanie daného kľúča pomocou hesla.

**Používanie overených a dôveryhodných images**. Často dochádza k tomu, že z dôvodu pohodlia dochádza k používaniu images od tretích strán. Niekedy je to z toho dôvodu, že daný image už je nakonfigurovaný na naše potreby. V takomto prípade však nie je možné zaistiť to, že daná image neobsahuje škodlivé súbory. Tým spôsobom by mohlo dôjsť k tomu, že vďaka jednému kontajneru môže dôjsť k infikovaniu celého hosta, poprípade až celej siete.

**Bezpečnostné hrozby spojené s registry.**

**Nezabezpečené spojenie s registry**. V rámci image často bývajú zakomponované firemné aplikácie, prístupové kódy a ďalšie citlivé údaje. Síce by sa tomu malo zabraňovať, avšak niekedy to nie je možné. Z toho dôvodu je nutné to, aby bolo pri šírení image zabezpečené spojenie. Takéto spojenie zabezpečuje nutnosť byť na internej firemnej sieti, poprípade na VPN. Taktiež je vhodné to, aby bola samotná komunikácia šifrovaná a bol riadený prístup do registry.

**Neefektívna autentifikácia a autorizácia.** Vďaka tomu že je registry často uložiskom všetkých images s ktorými firma pracuje, tak je veľmi podstatné nastaviť správnu prístupovú politiku. V prípade že by sa dostal útočník k registry, poprípade by bol schopný do nej vkladať upravené images, tak by dokázal nie len získať citlivé firemné dáta a know-how, ale taktiež aj podstrčiť škodlivé programy, poprípade odstrániť alebo

šifrovať jej obsah. Všetci ostatní užívatelia, ktorí by s danými poškodenými images pracovali, tak by nemali ani len tušenia, že je na príklad celá ich aktivita skenovaná, že aktívne vyzrádzajú citlivé dáta, poprípade nevedome napádajú ostatné zariadenia v sieti. Tomuto sa dá zamedziť správnym nastavením autentifikácie voči užívateľom a registry. Ďalším stupňom zabezpečenia je prideľovanie práv k jednotlivým projektom a udeľovanie prístupu len užívateľom, ktorý potrebujú s daným projektom pracovať. Tým je možné zabezpečiť autorizáciu.

**Bezpečnostné hrozby spojené s kontajnermi.**

**Nekontrolovaný bežiaci kontajner**. Jedná sa o problematiku, kedy ostáva kontajner bežať na pozadí počas dlhšieho obdobia. Vďaka tomu má útočník priestor a čas na napadnutie a zneužitie daného kontajneru pre získanie prístupu do zvyšku systému.

**Aplikačné hrozby**. Jedná sa o hrozby spojené s aplikáciami bežiacimi v rámci kontajnerov. Preto je nutné dbať zvýšenú pozornosť aj pri zabezpečovaní samotných aplikácii a softwaru, ktorý bude na kontajneri fungovať.

**Zlá konfigurácia kontajneru**. V rámci spúšťania kontajneru sú umožnené rôzne možnosti ako konfigurovať kontajner. Či už je to užívateľ s ktorým sa spustí, tak aj to, s akými privilégiami poprípade na akej sieti bude kontajner figurovať. Tieto aspekty dokážu vo veľkej miere sprístupniť a uľahčiť infikovanie všetkých ostatných kontajnerov a dokonca aj samotného systému, z jedného zle konfigurovaného kontajneru.

Z vyššie popísaných odstavcov je jasné, že takéto riešenie prináša množstvo bezpečnostných rizík. Niektoré sú však spoločné nehľadiac na to, na akej platforme by bolo riešenie postavené. Čo je však veľmi dôležité je to, že boli dané riziká identifikované a vďaka tomu je možné ich eliminovať a navrhnúť opatrenia.

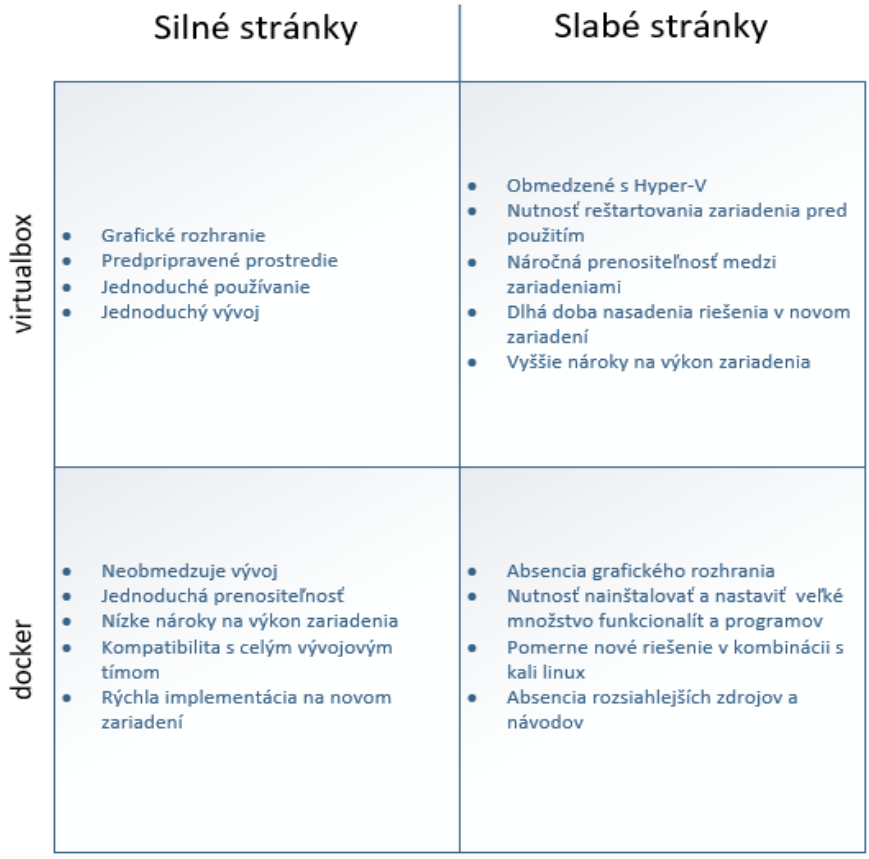
### 2.4.3 Zhodnotenie a výber platformy

Zhrnutie analýz jednotlivých platforiem je možné vidieť na obrázku nižšie.

Virtualbox vedie z hľadiska užívateľského komfortu a taktiež aj z hľadiska náročnosti používania. Jeho slabé stránky pozostávajú prevažne v komplikovanej implementácii v danej spoločnosti a v porovnaní s dockerom taktiež aj náročnejšieho zdieľania a presunu nástroju.

Docker zase vedie vďaka tomu že už je v spoločnosti využívaný. Rovnako tak lepšie spĺňa požiadavku na flexibilitu a portabilitu skenovacieho nástroju. Ďalším bonusom je jeho nenáročnosť na zariadenie na ktorom beží a veľmi rýchla implementácia na novom zariadení. Nevýhody plynú primárne z náročnejšieho vývoju daného skenovacieho zariadenia a jeho prostredia. Z hľadiska bezpečnosti vytvárajú obe riešenia podobne vysoké nároky, avšak u dockeru je riešenie bezpečnosti ľahšie, a to z toho dôvodu, že disponujem väčšími znalosťami v oblasti kontajnerizácie ako v prípade virtualizácie cez virtualbox. Vďaka tomu sa mierne stráca obava z neznáma, nedostatku kontroly a dohľadu.

Po zvážení všetkých kladov a záporov jednotlivých platforiem došlo k rozhodnutiu, že platformou, na ktorej budem skener vyvíjať je docker. Najväčšiu váhu na tomto rozhodnutí má to, že je docker aktívne používaný v spoločnosti a vďaka internej repository poskytuje veľkú možnosť portability. Čo sa týka negatív tak beriem do úvahy to, že pre daný nástroj postačuje príkazový riadok a taktiež aj to že sa jedná o jednorazový vývoj, ktorý nebude požadovať väčšiu údržbu.




**Obrázok č. 12 Graf. zobrazenie silných a slabých stránok jednotlivých riešení**
(Zdroj: Vlastné spracovanie)

## 2.5 Analýza rizík

Veľmi podstatnou časťou analýzy rizík je identifikácia a hodnotenie aktív, v tomto prípade sieťových prvkov a služieb, ktoré na nich bežia. V prvom kroku došlo k analýze samotných sieťových prvkov nachádzajúcich sa na bežnej firemnej sieti. Z dôvodu bezpečnostných opatrení nie sú vypísané skutočné sieťové prvky spoločnosti, ale ide len o obecnú analýzu bežného stavu v spoločnostiach.

V druhom kroku dochádza k identifikácii hrozieb a zraniteľností. Táto analýza je postavená na základe analýzy portov a služieb, ktoré môžu byť na daných prvkoch otvorené a spustené. Opäť ide o obecné zhodnotenie, vďaka ktorému je možné nastaviť skenovací nástroj čo najkomplexnejšie.

### 2.5.1 Hodnotenie aktív

Prvým krokom hodnotenia aktív, je ich identifikácia. Vďaka identifikácii aktív je možné spracovať ich následné ohodnotenie. Aktíva budú členené na štyri základné skupiny, ktorými sú: dáta, hardware, software a služby. Hodnota jednotlivých aktív je vypočítaná na základe troch informačných kritérií. Tými sú: dôvernosť, integrita a dostupnosť. Následný výpočet hodnôt aktív je možné vidieť na vzorci nižšie.

$$Hodnota\ aktíva = \frac{(Dôvernosť + Integrita + Dostupnosť)}{3}$$

Farebné označenie jednotlivých aktív a ich hodnôt, ktoré je vidieť v tabuľke nižšie, vychádza z tabuľky riešenej v teoretickej časti práce.

**Tabuľka č. 2 Identifikácia a hodnotenie aktív**
(Zdroj: Vlastné spracovanie)

| Typ | Aktivum (A) | Zdroj | Dôvernosť | Integrita | Dostupnosť | Hodnota |
|---|---|---|---|---|---|---|
| Dáta | Dáta o zamestnancoch | Server | 4 | 4 | 3 | 4 |
| | Interné dáta | Server, notebook | 4 | 4 | 3 | 4 |
| | Kód | Gitlab, notebook | 5 | 4 | 5 | 5 |
| | Zálohy kódu | NFS, notebook | 5 | 5 | 3 | 4 |
| | Docker images | Docker registry, notebook | 3 | 4 | 4 | 4 |
| Hardware | Gitlab server | | 5 | 5 | 5 | 5 |
| | Gitlab client | | 4 | 3 | 3 | 3 |
| | AD server | | 5 | 5 | 4 | 5 |
| | RDP | | 5 | 3 | 3 | 4 |
| | NFS server | | 5 | 5 | 3 | 4 |
| | SMTP server | | 4 | 4 | 3 | 4 |
| | Notebook | | 4 | 3 | 2 | 3 |
| Software | Operačný systém | Server, notebook | 3 | 3 | 2 | 3 |
| | Vývojová platforma | RDP, Docker registry, notebook | 3 | 4 | 4 | 4 |
| | Vyvinuté aplikácie | RDP, Gitlab, NFS, notebook | 5 | 5 | 5 | 5 |
| | Informačný systém | Server | 4 | 3 | 3 | 3 |
| Služby | Mailová služba | SMTP server | 4 | 4 | 3 | 4 |
| | Autentifikačná služba | AD server | 5 | 5 | 3 | 4 |
| | Zálohovacia služba | NFS server, Gitlab server | 5 | 5 | 3 | 4 |
| | Internetové pripojenie | Sídlo spoločnosti | 5 | 5 | 4 | 5 |
| | Gitové služby | Gitlab server, Gitlab client | 5 | 5 | 5 | 5 |

Medzi bežné sieťové prvky, ktoré sa na sieti nachádzajú patria mailové (SMTP) servery, AD servery, tlačiarne a bežné RDP. Prirodzene býva na sieti viacej prvkov, avšak ako už bolo spomínané pôjde o obecnú analýzu. Ohľad na špecifické prostredie a sieťové zariadenia bude nutné brať už priamo pri vytváraní jednotlivých skenovacích skriptov.

Z výstupu je zrejmé, že všetky aktíva, ktoré sú bežne využívané pri vývoji majú hodnotu vyššiu ako tri. To znamená, že narušenie fungovania ktoréhokoľvek z vymenovaných aktív, bude mať za následok problémy a finančné straty. Niektoré dokonca spôsobia vážne problémy, veľké straty poprípade až existenčné problémy firmy.

### 2.5.2 Identifikácia hrozieb a zraniteľností

V bežnom firemnom prostredí existuje nespočetné množstvo hrozieb a zraniteľností. V diplomovej práci sa však budem venovať len tým, ktoré pramenia z nedostatkov v rámci sieťovej infraštruktúry a nastavenia jednotlivých zariadení v počítačovej sieti. Z toho dôvodu budem v rámci identifikácie hrozieb a zraniteľností nadväzovať priamo na hrozby a zraniteľnosti súvisiacej s jednotlivými portami a službami, ktoré na zariadeniach bežia.

Veľké množstvo hrozieb, ako na príklad zašifrovanie dát, ich odcudzenie poprípade zničenie, pramení práve z toho, že útočník získa prístup do zariadenia. Možností na to,

aby sa do zariadenia dostal je mnoho. Či už sa jedná o osobu z vnútra organizácie, poprípade niekoho, kto zneužije zraniteľnosť pôsobiacu na dané aktívum. Z dôvodu, že sa jedná o naozaj komplexnú a rozsiahlu tému, tak sa pokúsim v nasledujúcich riadkoch popísať aspoň základné hrozby a zraniteľnosti, ktoré na aktíva pôsobia.

Medzi **zraniteľnosti** je možné zaradiť:

- Nedostatočné školenie a podľahnutie phishingu
- Inštalácia neovereného softwaru
- Zlé zabezpečenie siete (firewall)
- Zle nastavenie prístupových práv
- Zle nastavená politika hesiel
- Absencia pravidelnej aktualizácie systémov
- Zle nastavené služby
- Otvorené nežiaduce porty

Opäť niektoré z **hrozieb**, ktoré z vyššie popísaných zraniteľností vyplývajú:

- Krádež dát
- Znehodnotenie dát
- Modifikácia dát
- Zamedzenie prístupu k dátam a službám
- Narušenie fungovania služieb
- Neoprávnené použitie a prístup k dátam a službám
- Zlyhanie služieb a zariadení

V nasledujúcej tabuľke sú znázornené hrozby vo vzťahu k jednotlivým zraniteľnostiam. V rámci tabuľky je možné vidieť aj hodnotu, ktorá znázorňuje možnú pravdepodobnosť, že dôjde k ich naplneniu.

**Tabuľka č. 3 Identifikácia hrozieb vo vzťahu so zraniteľnosťami**
(Zdroj: Vlastné spracovanie)

| Hrozba (T) | P | Príklad zraniteľnosti |
|---|---|---|
| Špionáž | 3 | Nedostatočná ochrana sieťovej infraštruktúry |
| | 4 | Absencia logovania, nesprávne nastavené prístupy, slabá kontrola zamestnancov |
| Znehodnotenie dát | 4 | Napadnutie ransomwarom |
| | 5 | Neodborná manipulácia |
| Stráta dát | 3 | Zlyhanie zálohovania |
| Modifikácia dát útočníkom | 3 | Nedostatočne zabezpečený prístup |
| Zamedzenie prístupu | 3 | Napadnutie AD |
| | 3 | Útočník zmení oprávnenia užívateľov |
| Narušenie dostupnosti služieb | 4 | DDoS útok |
| Neoprávnené použitie a prístup k dátam a službám | 3 | Útočník si vytvorí backdoor na zariadení |
| | 3 | Absencia logovania, nesprávne nastavené prístupy, slabá kontrola zamestnancov |
| Zlyhanie služieb | 3 | Nedostatočná aktualizácia |
| Neschopnosť detekcie problému | 5 | Absencia logov |
| | 4 | Absencia dohľadu nad zariadeniami na sieti |

Ako je možné pozorovať, väčšina hrozieb má priamy vplyv na tri základné kritéria hodnotenia aktív, ktorými sú dôvernosť, integrita a dostupnosť. Množstvo zo zraniteľností, z ktorých vyplývajú dané hrozby, je možné identifikovať vďaka skenovaciemu nástroju, ktorý je tvorený v ďalších častiach práce. Vďaka správnej a včasnej identifikácii je následne možné navrhnúť opatrenia, ktorými sme schopní v čo najväčšej miere eliminovať dané hrozby.

Farebné rozlíšenie jednotlivých pravdepodobností je opäť podobné ako v hodnotení aktív. Pre prehľadnosť pridávam nasledujúcu klasifikačnú schému.

**Tabuľka č. 4 Klasifikačná schéma pravdepodobnosti vzniku hrozieb**
(Zdroj: Vlastné spracovanie)

| Pravdepodobnosť hrozby | Klasifikačný stupeň |
|---|---|
| Náhodná | 1 |
| Nepravdepodobná | 2 |
| Pravdepodobná | 3 |
| Veľmi pravdepodobná | 4 |
| Trvalá | 5 |

### 2.5.3 Matica zraniteľností

Na základe identifikovaných hrozieb a aktív je možné spraviť maticu zraniteľností. Táto matica udáva to, ktorá hrozba má vplyv na ktoré aktívum a taktiež aj to, ako vysoká je zraniteľnosť aktíva voči danej hrozbe.

**Tabuľka č. 5 Matica zraniteľnosti**
(Zdroj: Vlastné spracovanie)

| Zraniteľnosť [V] | | Zdroj | Dáta | | | | | Hardware | | | | | | | Software | | | | Služby | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Aktívum | Dáta o zamestnancoch | Interné dáta | Kód | Zálohy kódu | Docker images | Gitlab server | Gitlab client | AD server | RDP | NFS server | SMTP server | Notebook | Operačný systém | Vývojová platforma | Vyvinuté aplikácie | Informačný systém | Mailová služba | Autentifikačná služba | Zálohovacia služba | Internetové pripojenie | Gitové služby |
| | | A | 4 | 4 | 5 | 4 | 4 | 5 | 3 | 5 | 4 | 4 | 4 | 3 | 3 | 4 | 5 | 3 | 4 | 4 | 4 | 5 | 5 |
| Hrozba | T | | | | | | | | | | | | | | | | | | | | | | |
| Špionáž z vonku | 3 | | 3 | 3 | 4 | 2 | 3 | | | 2 | 3 | | | | 1 | 2 | 3 | 2 | 3 | 3 | | | 3 |
| Špionáž z vnútra | 4 | | 4 | 5 | 4 | 2 | 4 | | | 2 | 5 | | | | 1 | 4 | 4 | 4 | 3 | 3 | | | 4 |
| Znehodnotenie dát - úmyselné | 4 | | 2 | 3 | 3 | 2 | 3 | | | | | | | | | | | | | | | | |
| Znehodnotenie dát - neúmyselné | 5 | | 1 | 4 | 5 | 4 | 2 | | | | | | | | | | | | | | | | |
| Stráta dát | 3 | | 3 | 4 | 3 | 2 | 3 | | | | | | | | | | | | | | | | |
| Modifikácia dát útočníkom | 3 | | 2 | 3 | 4 | 2 | 4 | | | | | | | | | | 5 | 3 | | | | | |
| Zamedzenie prístupu - AD | 3 | | | | | | | 5 | 5 | 5 | 5 | 5 | 5 | | | | | | | 5 | | | |
| Zamedzenie prístupu - oprávnenia | 3 | | | | | | | 4 | 4 | 3 | 3 | 4 | 2 | 3 | | | | | | | | | |
| Narušenie dostupnosti služieb | 4 | | | | | | | | | | | | | | | | | | 4 | 3 | 3 | | 4 |
| Neoprávnené použitie a prístup k dátam a službám - z vonku | 3 | | 3 | 3 | 4 | 2 | 3 | | | | 4 | | | | 1 | 3 | 3 | 3 | 4 | | | 3 | 4 |
| Neoprávnené použitie a prístup k dátam a službám - z vnútra | 3 | | 4 | 5 | 4 | 2 | 4 | | | | 5 | | | | 1 | 4 | 4 | 4 | 4 | | | 5 | 5 |
| Zlyhanie služieb | 4 | | | | | | | | | | | | | | | | | | 4 | 3 | 3 | 5 | 5 |
| Neschopnosť detekcie problému - logy | 5 | | | | | | | 3 | 3 | 4 | 5 | 4 | 4 | 3 | | | | | | | | | |
| Neschopnosť detekcie problému - prehľad zariadení | 4 | | | | | | | 5 | 5 | 5 | 5 | 5 | 5 | 4 | | | | | | | | | |

Hodnoty a farebné rozlíšenie zraniteľnosti odpovedá schéme na tabuľke nižšie.

**Tabuľka č. 6 Klasifikačná schéma pre hodnotenie zraniteľnosti**
(Zdroj: Vlastné spracovanie)

| Zraniteľnosť | Klasifikačný stupeň |
|---|---|
| Veľmi nízka | 1 |
| Nízka | 2 |
| Stredná | 3 |
| Vysoká | 4 |
| Veľmi vysoká | 5 |

### 2.5.4 Matica rizík

Pre znázornenie a výpočet rizík, slúži matica rizík. Táto matica je takmer zhodná s maticou zraniteľností, s tým rozdielom, že miesto hodnôt zraniteľnosti obsahuje hodnotu rizika. Túto hodnotu je možné získať vynásobením troch základných parametrov. Tými sú A (hodnota aktív) x T (pravdepodobnosť hrozby) x V (zraniteľnosť). Výstupom z tohto výpočtu bude hodnota 0-125. Pre klasifikáciu jednotlivých hodnôt rizika pridávam nasledujúcu tabuľku.

**Tabuľka č. 7 Klasifikačná schéma pre hodnotenie rizík**
(Zdroj: Vlastné spracovanie)

| Úroveň rizika | Klasifikačný stupeň |
|---|---|
| Bezvyznamné riziko | 0-10 |
| Akceptovateľné riziko | 11-20 |
| Nizke riziko | 21-30 |
| Nežiadúce riziko | 31-60 |
| Neprijateľné riziko | 61-125 |

V tabuľke nižšie je možné vidieť, že väčšina rizík spadá pod úroveň „Neprijateľné riziko“ a úroveň „Nežiadúce riziko“. To, čím je to zapríčinené a taktiež aj to, akým spôsobom je možné dané rizika eliminovať popíšem v časti „Zhodnotenie“.

**Tabuľka č. 8 Matica rizík**
(Zdroj: Vlastné spracovanie)

| Riziko [R] | | Zdroj | Dáta | | | | | Hardware | | | | | | | Software | | | | Služby | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Aktívum | Dáta o zamestnancoch | Interné dáta | Kód | Zálohy kódu | Docker images | Gitlab server | Gitlab client | AD server | RDP | NFS server | SMTP server | Notebook | Operačný systém | Vývojová platforma | Vyvinuté aplikácie | Informačný systém | Mailová služba | Autentifikačná služba | Zálohovacia služba | Internetové pripojenie | Gitové služby |
| | | A | 4 | 4 | 5 | 4 | 4 | 5 | 3 | 5 | 4 | 4 | 4 | 3 | 3 | 4 | 5 | 3 | 4 | 4 | 4 | 5 | 5 |
| Hrozba | T | | | | | | | | | | | | | | | | | | | | | | |
| Špionáž z vonku | 3 | | 33 | 33 | 56 | 26 | 33 | | | 28 | 33 | | | | 8 | 22 | 45 | 20 | 33 | 39 | | | 45 |
| Špionáž z vnútra | 4 | | 59 | 73 | 75 | 35 | 59 | | | 37 | 73 | | | | 11 | 59 | 80 | 53 | 44 | 52 | | | 80 |
| Znehodnotenie dát - úmyselné | 4 | | 29 | 44 | 56 | 35 | 44 | | | | | | | | | | | | | | | | |
| Znehodnotenie dát - neúmyselné | 5 | | 18 | 73 | 117 | 87 | 37 | | | | | | | | | | | | | | | | |
| Stráta dát | 3 | | 33 | 44 | 42 | 26 | 33 | | | | | | | | | | | | | | | | |
| Modifikácia dát útočníkom | 3 | | 22 | 33 | 56 | 26 | 44 | | | | | | | | | | 75 | 30 | | | | | |
| Zamedzenie prístupu - AD | 3 | | | | | | | 75 | 50 | 70 | 55 | 65 | 55 | | | | | | | 65 | | | |
| Zamedzenie prístupu - oprávnenia | 3 | | | | | | | 60 | 40 | 42 | 33 | 52 | 22 | 27 | | | | | | | | | |
| Narušenie dostupnosti služieb | 4 | | | | | | | | | | | | | | | | | | 59 | 52 | 52 | | 80 |
| Neoprávnené použitie a prístup k dátam a službám - z vonku | 3 | | 33 | 33 | 56 | 26 | 33 | | | | 44 | | | | 8 | 33 | 45 | 30 | 44 | | | 42 | 60 |
| Neoprávnené použitie a prístup k dátam a službám - z vnútra | 3 | | 44 | 55 | 56 | 26 | 44 | | | | 55 | | | | 8 | 44 | 60 | 40 | 44 | | | 70 | 75 |
| Zlyhanie služieb | 4 | | | | | | | | | | | | | | | | | | 59 | 52 | 52 | 93 | 100 |
| Neschopnosť detekcie problému - logy | 5 | | | | | | | 75 | 50 | 93 | 92 | 87 | 73 | 45 | | | | | | | | | |
| Neschopnosť detekcie problému - prehľad zariadení | 4 | | | | | | | 100 | 67 | 93 | 73 | 87 | 73 | 48 | | | | | | | | | |

### 2.5.5 Zhodnotenie analýzy rizík

Ako už som poznamenal nad maticou rizík, veľká časť analyzovaných rizík spadá do kategórie nežiadúcich alebo neprijateľných rizík. To môže byť spôsobené tým, že v rámci hodnotenia aktív identifikujem len tie aktíva, ktoré bývajú priamo naviazané na vývoj v spoločnosti. Takéto aktíva sú prirodzene pre vývojovú spoločnosť kritické, a teda všetky dosahujú vysokú hodnotu. Ďalším aspektom, ktorý tieto hodnoty výrazne ovplyvňuje je to, že vo väčšine bežných spoločností neexistuje dohľadový, alebo iný systém, ktorý by umožňoval detegovať, poprípade eliminovať pokusy o útok na sieť. Z toho dôvodu je nutné vytvoriť systém kontroly, vďaka ktorému bude možné detailnejšie

identifikovať zraniteľnosti jednotlivých aktív, a tak navrhnúť opatrenia znižujúce riziká na prijateľnú úroveň.

## 2.6 Zhodnotenie analýzy

V rámci analýzy došlo k predstaveniu spoločnosti a jej požiadaviek. Z tých vyplynulo, že hlavnými kritériami pre skenovací nástroj sú flexibilita, jednoduchosť a prenositeľnosť.

Na základe definovaných kritérií došlo k rozhodnutiu, že systém bude implementovaný na jednej z virtualizačných platforiem, vďaka čomu bude možné zaistiť flexibilitu a prenositeľnosť. Pri finálnom zhodnotení všetkých kladov a záporov pomohlo k rozhodnutiu to, že v rámci spoločnosti už sa používa docker, čo znamená, že lepšie spĺňa posledné kritérium, ktorým je jednoduchosť.

Taktiež došlo k výberu skenovacieho prostriedku, ktorým je nmap. Ten poskytuje komplexný skenovací nástroj, ktorý sa veľmi ľahko používa a zaberá minimálne množstvo pamäte. Vďaka tomu takisto splňuje všetky požadované kritéria.

Po voľbe prostredia a skenovacích prostriedkov, bolo potrebné identifikovať aktíva, ktoré budú skenované. K tomu pomohla identifikácia a hodnotenie aktív spoločnosti. Na základe identifikovaných aktív došlo k identifikácii hrozieb a zraniteľností vplývajúcich na dané aktíva. Výstupom toho celého je matica rizík, ktorá ukázala, že na veľkú časť aktív vplývajú nežiadúce až neprijateľné riziká. To je spôsobené tým, že došlo k analýze úzkej skupiny aktív, ktoré priamo súvisia s vývojom, a teda sú pre vývojovú spoločnosť kriticky dôležité. Druhým faktorom, ktorý bol identifikovaný ako zdroj tak vysokých rizík, je absencia monitorovacieho zariadenia na sieti. Takýto nástroj poskytuje nie len možnosť reaktívne reagovať na aktuálne hrozby, ale taktiež pôsobí ako jediný prostriedok, akým je možné zhodnotiť skutočný stav zariadení v sieti. Vďaka takémuto zhodnoteniu je následne možné vytvárať opatrenia, a tak proaktívne chrániť aktíva v sieti.

Na základe analýzy rizík je možné vyhodnotiť, že požiadavky spoločnosti na skenovací nástroj sú legitímne. Takýto nástroj skutočne poskytuje prostriedky pre zvýšenie bezpečnosti sieťového prostredia a aktív v sieti prítomných. Taktiež je možné zhodnotiť, že spoločnosť disponuje prostriedkami, vďaka ktorým je možné projekt úspešne uskutočniť.

# 3 VLASTNÉ NÁVRHY RIEŠENIA

Finálna časť práce sa venuje vlastnému návrhu a tvorbe riešenia. Tento proces je založený na výstupoch z analýz. Prostredníctvom analýzy požiadaviek spoločnosti a analýzy technických prostriedkov je možné navrhnúť a vytvoriť prostredie, v rámci ktorého bude nástroj pracovať. Následne, v nadväznosti na analýzu rizík, v ktorej došlo k identifikácii a hodnoteniu aktív, hrozieb, zraniteľností a taktiež aj výpočtu samotných rizík, je možné vytvoriť nástroj, ktorý bude špecifický pre potreby daného sieťového prostredia. Výstupom z nástroju bude výstup s možnými zraniteľnosťami, ku ktorým bude identifikovaných niekoľko príkladov toho, ako ich zabezpečiť pomocou opatrení.

## 3.1 Príprava prostredia

Pri príprave prostredia treba brať ohľad na viacero faktorov. Prvým z nich, je vytvorenie takého docker image, ktorý bude schopný spustiť a ovládať aj užívateľ, ktorý nemá obšírnejšie znalosti či už v oblasti dockeru alebo linuxu.

Druhou vecou na ktorú je treba hľadieť je zaistenie prenositeľnosti a dostupnosti pre všetkých oprávnených užívateľov. K tomu sa pri dockeri používajú docker registry. Takéto registry existujú či už verejné, tak aj interné. Vďaka tomu že spoločnosť používa pre vývoj práve docker, tak je veľmi jednoducho dostupná aj forma interná.

Po získaní, adaptácii a sprístupnení vhodnej image nasleduje potreba automatizácie vytvorenia a zostavenia kontajneru práve z danej docker image. K tomu slúžia prostriedky ako sú súbor yaml, Dockerfile a env file. Takéto súbory spolu s popisom v README je taktiež nutné niekde umiestniť tak, aby boli opäť dostupné pre oprávnených užívateľov. Pre tieto účely existuje platforma git, ktorá je interne využívaná v rámci spoločnosti.

Nutnosťou pre správne fungovanie je nastavenie dockeru na linux kontajnery.

### 3.1.1 Nastavenie image

Ako prvé je nutné stiahnuť kali-linux image. Pre moje potreby poslúži oficiálny image, ktorý vytvoril vývojový tým kalilinux. Jeho stiahnutie a aj všetky úpravy prebiehajú v príkazovom riadku PowerShell.

```
PS C:\Users\peter> docker pull kalilinux/kali-rolling
Using default tag: latest
latest: Pulling from kalilinux/kali-rolling
2da976873d5f: Pull complete
Digest: sha256:c950908e7cde70d846c44fef3e8a4fe0302144e655cfb87017b11580332c499b
Status: Downloaded newer image for kalilinux/kali-rolling:latest
```

**Obrázok č. 13 Stiahnutie základného image**
(Zdroj: Vlastné spracovanie)

Ako je na obrázku vidno, pre stiahnutie sa používa docker príkaz **pull** s názvom image. V prípade že by obsahoval názov aj adresu, na príklad myregistry.local:5000/kalilinux/kali-rolling, tak by automaticky docker vyhľadával danú image na lokálnej registry.

Po stiahnutí je možné z danej image vytvoriť a spustiť kontajner. Pre spustenie sa používa docker príkaz **run**.

```
PS C:\Users\peter> docker run -it kalilinux/kali-rolling /bin/bash
┌──(root㉿ 4424588bcc39)-[/]
└─# Som vo vnutri kontajneru a pisem do prikazoveho riadku
```

**Obrázok č. 14 Spustenie kontajneru v interaktívnom móde**
(Zdroj: Vlastné spracovanie)

Ako je možné vidieť, tak v príkaze používame taktiež **-it**, ktoré umožňuje interakciu a taktiež spustenie terminálového spojenia. Poslednou časťou príkazu je **/bin/bash**, čo umožňuje to, aby sa v rámci kontajneru spustil shell a teda príkazový riadok. Vďaka tomu je možné s kontajnerom interagovať, upravovať ho a inštalovať ďalšie nástroje a knižnice.

Po vytvorení a spustení kontajneru prichádza čas na jeho úpravu. Základom bude inštalácia knižníc a balíčkov. Takáto inštalácia prebieha rovnako ako na bežnom zariadení, ktoré beží na platforme linux. V nasledujúcich riadkoch budú vypísané aspoň niektoré kroky, ktoré som spravil pre to, aby bol kontajner aspoň z minimálnej časti kompletný a prispôsobený pre potreby skeneru. Tieto kroky taktiež prezentujú aj jedno z negatív riešenia v rámci docker, ktoré bolo v analýze spomínané. Tým je absencia väčšieho množstva základných nastavení a nutnosť budovania systému od základu.

**# apt update –** pomocou príkazu si systém stiahne a aktualizuje zoznam balíkov ktoré existujú v rámci systému. Taktiež získa informácie o ich najnovších verziách, vďaka čomu vie zaznamenať to, ktorý balík bude možné aktualizovať.

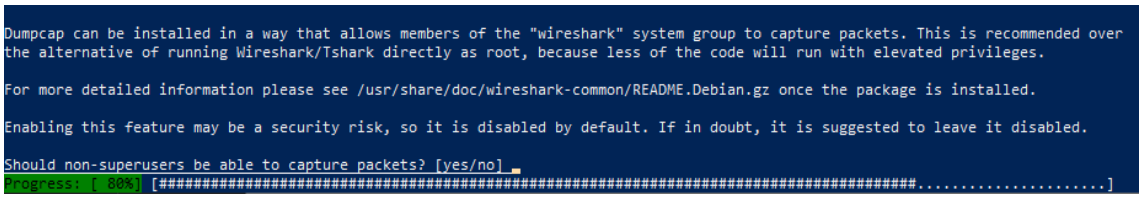
**# apt dist-upgrade –** vďaka zoznamu, ktorý bol získaný pomocou apt update stiahne všetky najnovšie verzie už existujúcich balíkov na danom zariadení. Taktiež vďaka predpone dist, odstráni zastaralé balíky, poprípade nainštaluje ich náhrady.

**# apt autoremove –** pre zachovanie čo najnižšej veľkosti kontajneru je nutné aby neobsahoval žiadne nadbytočné balíčky a software. Pri apt (poprípade apt-get) sa nesťahujú a neinštalujú len požadované balíky, ale taktiež aj software, na ktorom tieto balíky závisia. Vďaka autoremove bude dochádzať k tomu, že keď odstránim jeden balík, alebo aplikáciu, tak sa odstráni aj všetok ostatný software, ktorý je na danom balíku alebo aplikácii závislý.

**# apt clean –** opäť pomáha uchovávať veľkosť kontajneru na minime, a to vďaka tomu, že premaže všetky balíky, ktoré sú držané v cache.

**# apt install kali-tools-top10 –** umožňuje inštalovať desať základných nástrojov pre bezpečnostnú analýzu a audit. Medzi nimi sú na príklad nmap, sqlmap, wireshark, hydra a podobné. V rámci diplomovej práce síce s väčšinou daných nástrojov nebudem pracovať, avšak pre budúci vývoj a rozšírenie daného riešenia tvoria absolútny základ.

Ako je možné vidieť na obrázku, už pri inštalácii wireshark sa berie ohľad na bezpečnosť. Vďaka tomu, že bola zvolená možnosť nie, tak bolo zaistené to, že aj keď sa ku kontajneru dostane niekto neautorizovaný, tak nebude schopný používať dané nástroje bez správcovských oprávnení.




**Obrázok č. 15 Inštalácia wireshark a nastavenie oprávnení**
(Zdroj: Vlastné spracovanie)

Pri prvom teste príkazu nmap začalo dochádzať k chybe s oprávnením. To bolo spôsobené tým, že dané zariadenie, v tomto prípade linuxkit-00155d380106 nebolo evidované medzi hostami. Z toho dôvodu nevedel systém priradiť to, z akého zariadenia sa snažím príkaz nmap vykonať. Chyba sa následne tvorila pri každom sudo príkaze, na obrázku nižšie je možné ju vidieť v kombinácii s príkazom kill, ktorým sa pokúšam ukončiť proces.

```
┌──(root㉿ linuxkit-00155d380106)-[/]
└─# sudo kill 38
sudo: unable to resolve host linuxkit-00155d380106: Name or service not known
```

**Obrázok č. 16 Chyba s chýbajúcim hostom**
(Zdroj: Vlastné spracovanie)

Pre nápravu bolo nutné získať nástroj nano, ktorý umožňuje editovať súbory. Nástroj bol získaný pomocou **apt install nano**. Samotné rozdelenie a priradenie doménových mien k adresám sa nachádza v zložke **/etc/hosts**. Pre úpravu som teda použil príkaz **nano /etc/hosts**, čo umožnilo vidieť obsah súboru a taktiež aj jeho textovú editáciu.

Ako je možné vidieť na obrázku, tak súbor naozaj neobsahuje priradenie adresy k hostovi linuxkit-00155d380106.

```
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

**Obrázok č. 17 Otvorenie hosts pomocou nano bez hosta**
(Zdroj: Vlastné spracovanie)

Pomocou nano som následne súbor upravil a pridal hosta na adresu 127.0.0.1, ktorá figuruje taktiež ako localhost.

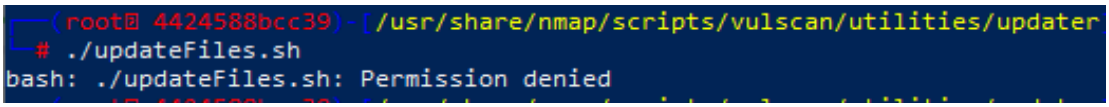
```
  GNU nano 5.4                                          /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1       linuxkit-00155d380106
```

**Obrázok č. 18 Hosts po úprave a doplnení**
(Zdroj: Vlastné spracovanie)

Ďalej som ešte nainštaloval základné knižnice a nástroje, ktoré pravdepodobne budú potrebné pre bežnú činnosť pri práci na skenovaní. Príkladom balíkov sú: iputils-ping, ipcalc, arp-scan, exploitdb a taktiež bolo potrebné získať -y procps. Mimo bežných balíkov som sa ešte rozhodol stiahnuť dva skripty, ktoré je možné použiť v spolupráci s nmap a vo veľkej miere uľahčujú skenovanie. Postup pre sprevádzkovanie daných skriptov je nasledujúci:

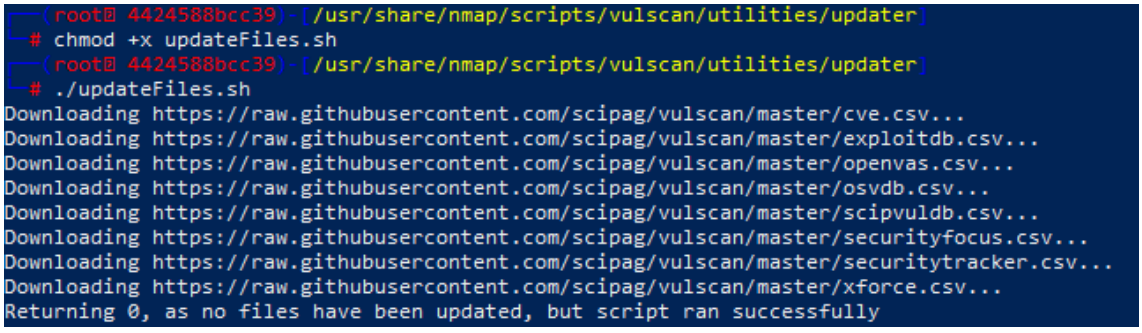
1.) cd /usr/share/nmap/scripts/
    a. Ako prvé je nutné sa dostať do umiestnenia nmap skriptov

2.) git clone https://github.com/vulnersCom/nmap-vulners.git
    a. Následne je možné stiahnúť spomínané skripty. Tie sú dostupné na verejnej github repository
3.) git clone https://github.com/scipag/vulscan.git
4.) cd vulscan/utilities/updater/
    a. Po stiahnutí je vhodné aktualizovať databázy so zraniteľnosťami. K tomu slúži skript updateFiles.sh. Z toho dôvodu sa premiestňujem do zložky, ktorá ho obsahuje
5.) chmod +x updateFiles.sh
    a. Samotný súbor je najskôr nutné povoliť ako spasiteľný program. Na to slúži chmod +x
    b. V prípade že by som to nespravil, tak by som došiel k hláške viditeľnej na nasledujúcom obrázku




**Obrázok č. 19 Znázornenie hlášky bez použitia chmod**
(Zdroj: Vlastné spracovanie)

6.) ./updateFiles.sh
    a. Po povolení spustenia stiahne všetky najnovšie databázy so zraniteľnosťami.

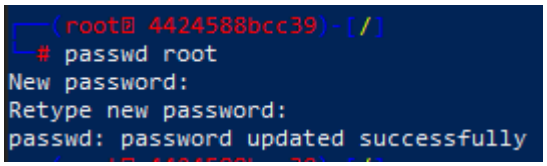



**Obrázok č. 20 Povolenie spustenia skriptu a jeho následné spustenie**
(Zdroj: Vlastné spracovanie)

Po nastavení celého prostredia v roli root, je následne možné vytvoriť užívateľa, ktorý bude mať oprávnenia na spúšťanie daných skriptov, avšak zároveň bude mať zamedzené sťahovanie a úpravu balíkov a programov. Tým spôsobom je možné zaistiť to, že danú image neupraví nijaká neoprávnená osoba. V prípade že by to bolo možné, tak by

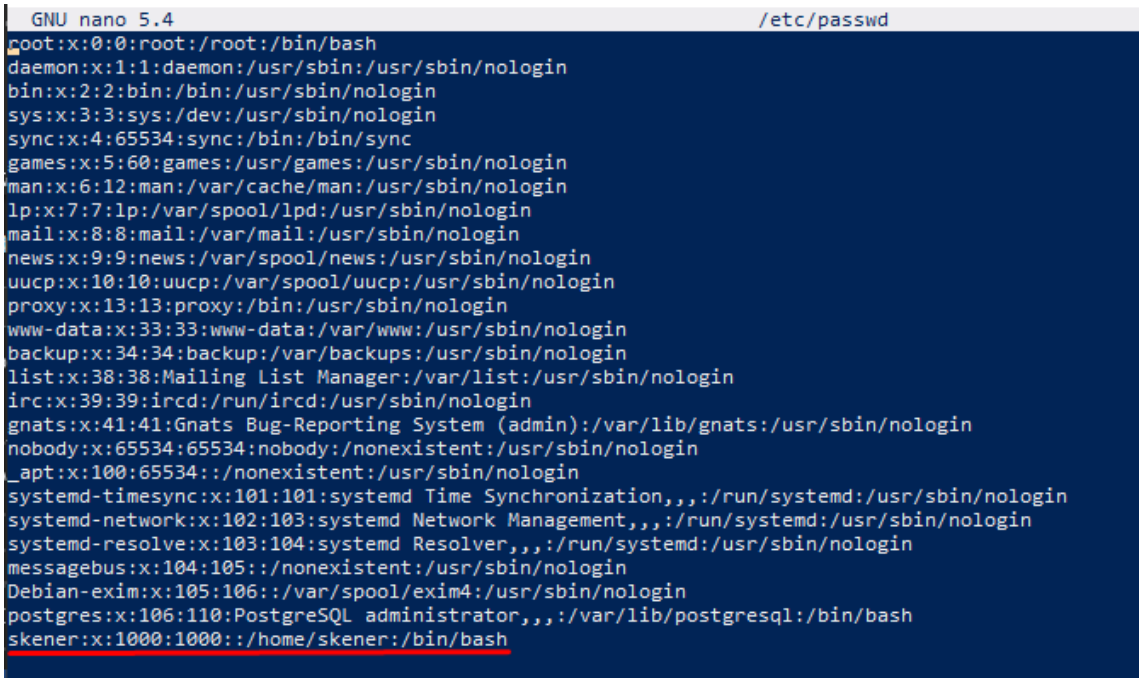
vznikalo enormné riziko toho, že môže byť daný kontajner zneužitý a bolo by umožnené aby unikali citlivé výsledky z auditov tretím stranám.

Ako prvé nastavujem heslo pre root, ktoré bude dostupné len mne, poprípade iným autorizovaným osobám. K zmene hesla používam príkaz **passwd root**, ktorý, vďaka tomu že je spúšťaný z root užívateľa, nepožaduje staré heslo, ale len nové. To je možné vidieť aj na obrázku nižšie. Heslo by malo byť čo najdlhšie, kombinovať znaky, veľké a malé písmená a čísla, tak aby bol v čo najväčšej miere sťažený bruteforce útok.



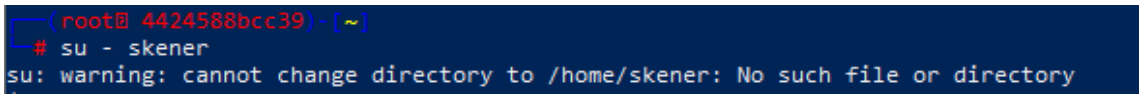

**Obrázok č. 21 Zmena hesla u root užívateľa**
(Zdroj: Vlastné spracovanie)

Následne je možné vytvoriť užívateľa s obmedzenými právomocami. Pre vytvorenie užívateľa používam príkaz **useradd -s /bin/bash skener**, možnosť -s slúži na to aby bol vytvorený so správnym „shellom“ /bin/bash. V prípade že by som ho nezadal, tak by sa nastavil základný „shell“ /bin/sh, ktorý by nemal niektoré funkcionality, ako na príklad históriu a podobne. Posledná časť príkazu – skener, definuje názov užívateľského konta. Po vytvorení môžem opäť meniť heslo pomocou passwd a nastavujem ho na príklad na „Heslo123..“. Jeho vytvorenie je možné overiť pomocou zobrazenia zložky passwd, ktorá sa nachádza v etc. Obsah zložky je vyobrazený na nasledujúcom obrázku.



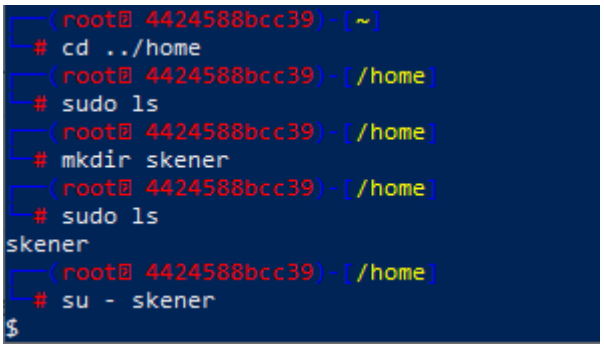

**Obrázok č. 22 Uživatelia v rámci kontajneru**
(Zdroj: Vlastné spracovanie)

Na obrázku je možné vidieť že domácim adresárom pre nového užívateľa je adresa /home/skener. Tento adresár sa však automaticky pri useradd netvorí, a preto je nutné vytvoriť aj danú zložku. Pre príklad čo by sa stalo v prípade že sa prihlásim na užívateľa skener, bez toho aby som mu vytvoril daný adresár je možné vidieť na obrázku nižšie.




**Obrázok č. 23 Prihlásenie bez vytvorenia domovského adresáru**
(Zdroj: Vlastné spracovanie)

Pre vytvorenie adresáru, je nutné ísť do zložky home. Tam je možné vytvoriť novú zložku pomocou príkazu **mkdir** a názvu zložky. Na obrázku nižšie je možné vidieť to, ako prechádzam do zložky home, v nej zobrazujem všetky aktuálne zložky, v tomto prípade žiadne neexistujú, následne tvorím zložku s mkdir a opäť zobrazujem zložky nachádzajúce sa v home. Ako je vidieť, tak daná zložka už tam existuje, preto dochádza k prihláseniu na užívateľa skener, u ktorého už nevypisuje hlášku ako na obrázku nižšie.



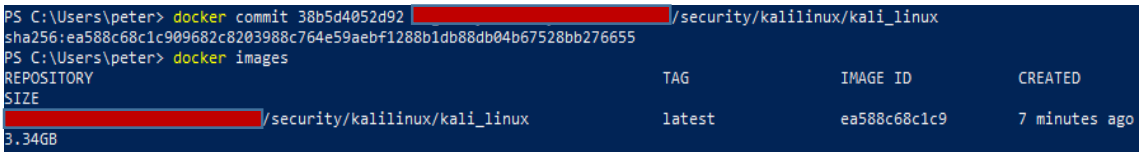

**Obrázok č. 24 Vytvorenie a kontrola domáceho adresáru pre užívateľa skener**
(Zdroj: Vlastné spracovanie)

Novo vytvorený užívateľ môže používať nmap, ipcalc, ping a všetky dostupné nástroje (až na wireshark, u ktorého sme to zakázali), ale zároveň nemôže pridávať nové, alebo upravovať stávajúce balíky a aplikácie. Vďaka tomu dochádza k pomerne dobrej eliminácii toho, aby neoprávnený užívateľ vložil do daného kontajneru nejaké nežiadúce prostriedky.

V tomto bode považujem kontajner ako plne pripravený na použitie. Ďalším a posledným krokom, ktorý je nutné vykonať je pretvorenie kontajneru na image, ktorá sa dá zdieľať. K tomu slúži docker príkaz commit. Príklad príkazu je nasledujúci *„docker commit 38b5d4052d92 registry.server.local/security/kalilinux/kali_linux“*.

- Číslo 38b5d4052d92 reprezentuje ID kontajneru ktorý chcem pretransformovať na image.
- registry.server.local/security/kalilinux/kali_linux zastupuje názov image. Bežne sa zadáva jednoduchší názov, avšak pre potreby zaslania image do registry je nutné, aby sa názov image zhodoval s názvom a adresou adresáru. Táto adresa a aj to ako sa tvorí budú lepšie popísané v nasledujúcej kapitole.

Príklad vytvorenia image a aj jeho zobrazenia v zozname images je vidieť na obrázku nižšie.

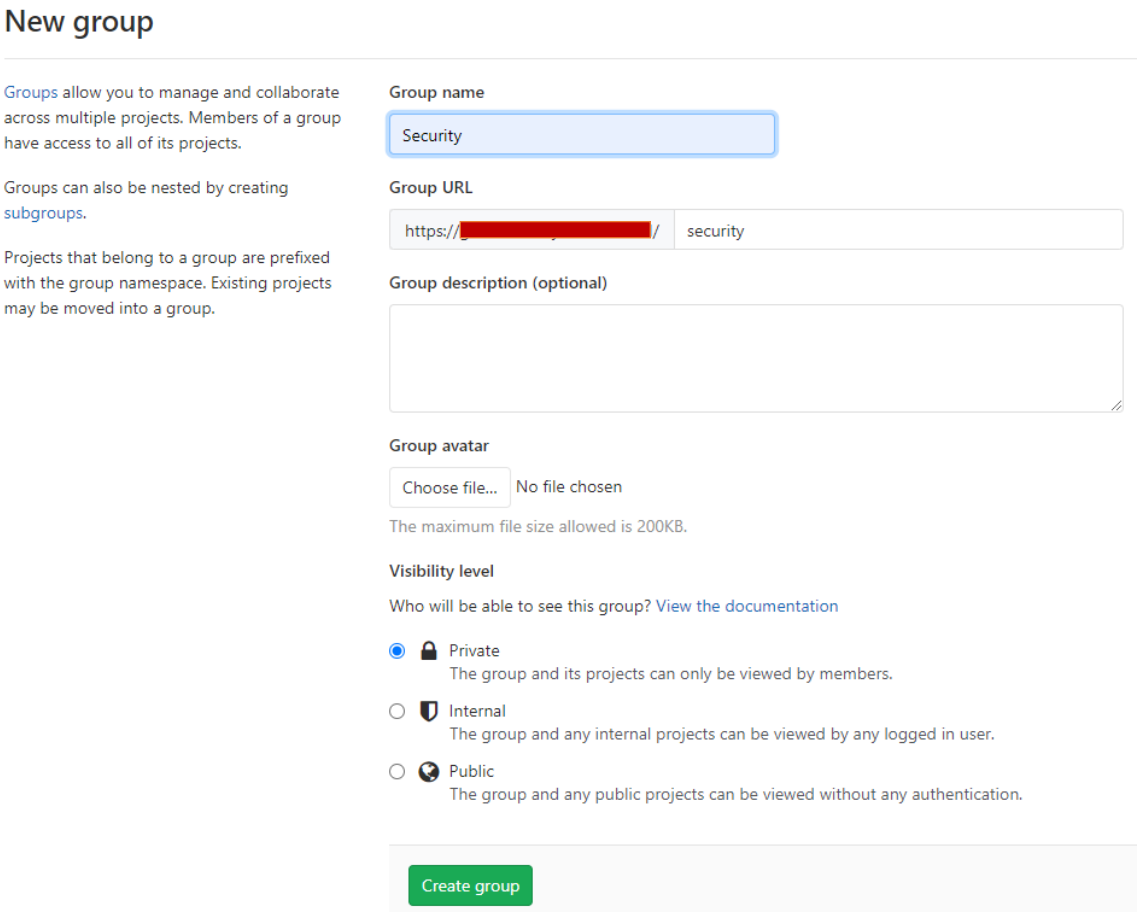



**Obrázok č. 25 Vytvorenie image a jej zobrazenie v zozname s images**
(Zdroj: Vlastné spracovanie)

### 3.1.2 Umiestnenie image v Docker registry

Po vytvorení image prichádza čas na jeho sprístupnenie. Pre takéto sprístupnenie použijem lokálnu Docker registry, ktorá sa používa aj pre uchovávanie iných image, používaných pri vývoji. Práve vďaka tomu, že je lokálna a že je k nej prístup len v rámci lokálnej siete, čiže za použitia VPN, alebo za fyzickej prítomnosti v budove s prístupom k heslu siete zaisťujem základný stupeň bezpečnosti daného riešenia.

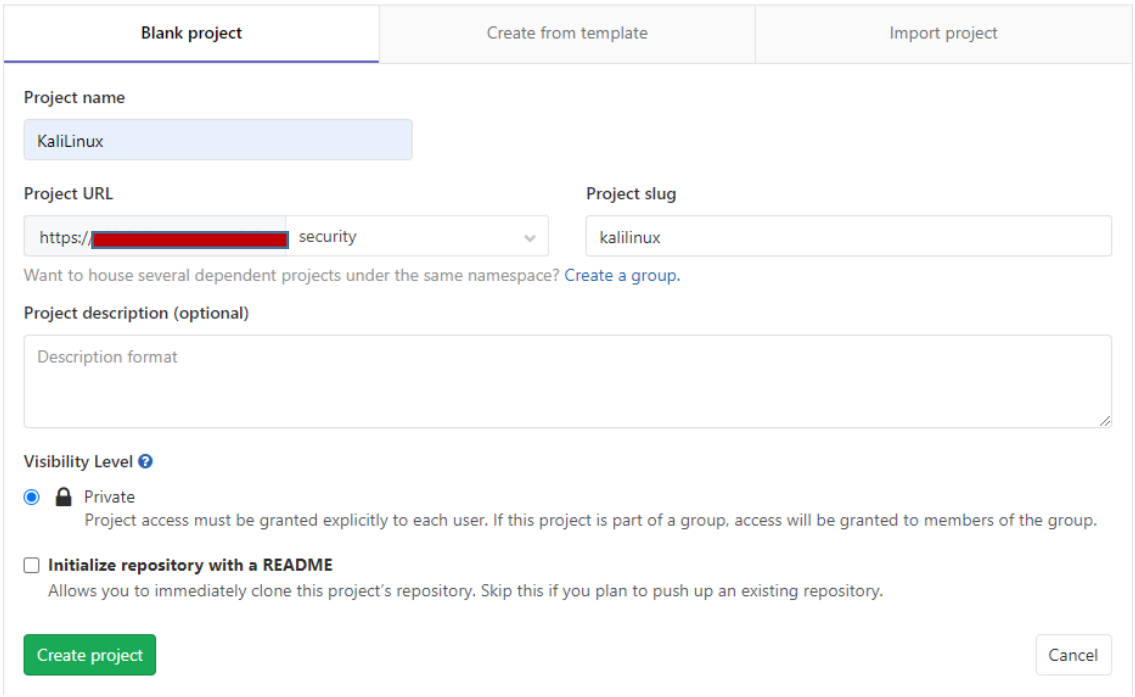
Prvým krokom, ktorý je treba podstúpiť je vytvorenie vlastnej skupiny pre daný projekt. Tento proces prebieha v platforme gitlab, ktorá je v spoločnosti používaná. Tvorba skupiny je znázornená na obrázku. Dávam jej názov Security a taktiež zaškrtávam dôležité nastavenie Private, vďaka čomu bude daná skupina a jej obsah dostupná len pre vybratých užívateľov.




**Obrázok č. 26 Tvorba skupiny pre aktuálny projekt**
(Zdroj: Vlastné spracovanie)

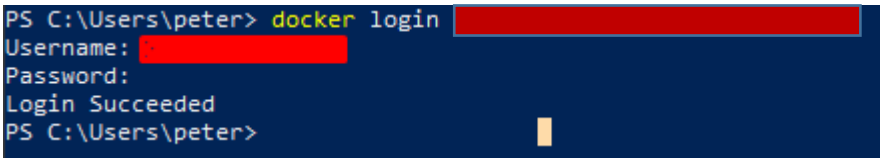
V rámci skupiny je následne nutné vytvoriť projekt. Takýto projekt môže obsahovať nie len image, ale aj iné súbory, ktoré budú dostupné na bežnej gitovej báze. Tvorba projektu

prebieha podobne ako tvorba skupiny. Volíme URL cestu pre projekt a taktiež nastavenie bezpečnosti, ktorá je po skupine zdedená na „Private“.



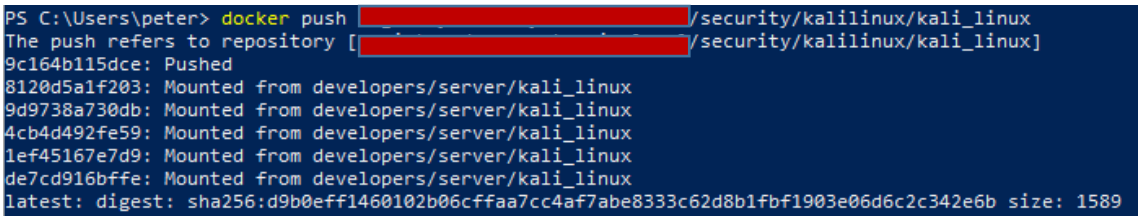

**Obrázok č. 27 Tvorba projektu v gitlabe**
(Zdroj: Vlastné spracovanie)

Následne je možné do daného projektu pridávať nové images, poprípade iné zložky a súbory. Pre prístup do danej registry je nutné prihlásenie. To opäť zvyšuje bezpečnosť daného riešenia a znižuje riziko jeho zneužitia. To, ako sa autentifikovať je zobrazené na obrázku nižšie.



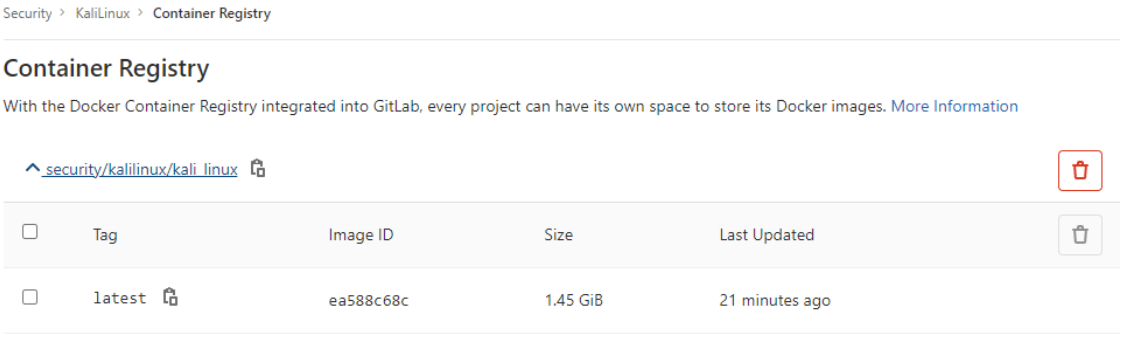

**Obrázok č. 28 Prihlásenie do docker registry**
(Zdroj: Vlastné spracovanie)

Po vytvorení prostredia pre uloženie a taktiež aj samotného kontajneru, môžem daný image zaslať do registry. Tento úkon sa vykonáva pomocou príkazu *„docker push registry.server.local/security/kalilinux/kali_linux“*.
*„registry.server.local/security/kalilinux/kali_linux“* zastupuje názov image a zároveň názov registry aj s miestom pre uloženie. Proces odosielania do registry je možné vidieť aj na obrázku nižšie.



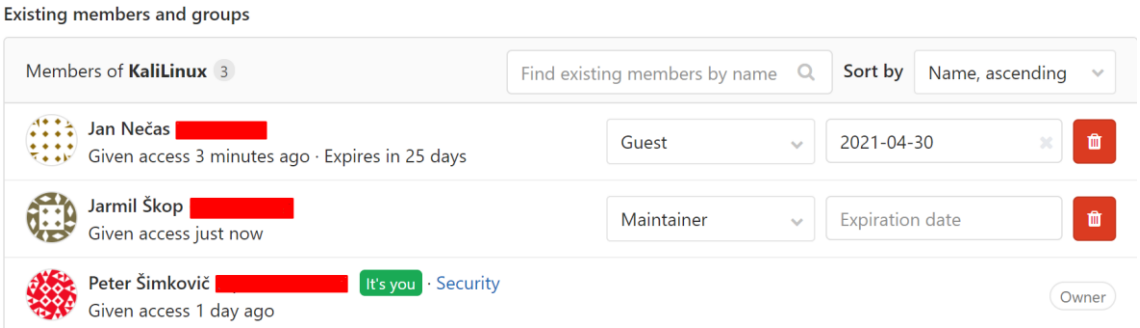

**Obrázok č. 29 Zasielanie image do registry**
(Zdroj: Vlastné spracovanie)

Následne je možné vidieť danú image aj v rámci gitlab, v projekte ktorý som vytvoril. To ako to vyzerá je opäť možné vidieť na obrázku nižšie.




**Obrázok č. 30 Image v gitlabe**
(Zdroj: Vlastné spracovanie)

Ďalším bodom zabezpečenia je nastavenie rol jednotlivým užívateľom. Gitlab už v základe obsahuje role, ktoré majú každá iné oprávnenia. Na príklad rola „Guest“ alebo inak aj hosť, umožňuje sťahovať projekty, avšak už neumožňuje jeho úpravu a správu. Týmto spôsobom je možné zaistiť to, že síce k daným nástrojom bude mať užívateľ prístup, avšak nebude môcť upravovať obsah projektu, čo sa týka aj docker images. Vďaka tomu viem zaistiť to, že nikto neoprávnený nebude do danej image vkladať nežiadúce prostriedky. Na obrázku je vidieť príklad rôznych rol, ako aj to že im je možné nastaviť aj obmedzený čas platnosti.




**Obrázok č. 31 Zobrazenie členov projektu a ich rol**
(Zdroj: Vlastné spracovanie)

### 3.1.3 Nastavenie prostriedkov Docker

Docker poskytuje väčšie množstvo prostriedkov ktoré uľahčujú a pomáhajú automatizovať kontajnerizáciu. Jedným z prvých prostriedkov, ktorý tu opíšem je súbor .env. Pokračovať budem s Docker Compose a Dockerfile. Taktiež budem popisovať aj shell skript ktorý uľahčuje celú automatizáciu. Túto časť zakončím popisom README súboru, ktorý bude slúžiť ako návod pre užívateľov, ktorí sa s daným riešením stretnú. Popri tomto návode taktiež opíšem ako sa daný kontajner spúšťa, a ako je z neho možné získať logové výpisy.

#### 3.1.3.1 Súbor .env

Súbor .env (environment), slúži na deklaráciu základných premenných pre budovanie kontajneru. Vďaka takémuto súboru je možné to, aby si užívateľ nastavil svoju špecifickú konfiguráciu na jednom mieste. Pre lepšie pochopenie prikladám obrázok s .env súborom, ktorý som vytvoril práve pre tento projekt.

```
################SHARED FOLDERS###################
FOLDER_IN=C:\Users\peter\git\security\kalilinux\IOfiles\in
FOLDER_OUT=C:\Users\peter\git\security\kalilinux\IOfiles\out

################NMAP SCRIPT PARTS################
#NMAP_CODE       Subor s kódom pre spustenie
#NMAP_IPs        Subor s IP adresami pre scan
#NMAP_OUTPUTFile        Nazov vystupneho suboru

NMAP_CODE=Code.txt
NMAP_IPs=IP.txt
NMAP_OUTPUTFile=vystup

################DEV ENVIROMENT###################
#Stanovuje to, či spustí kontajner tak, aby sa v ňom dalo pracovať, alebo spustí len NMAP_CODE súbor
IS_DEVELOP=0
```

**Obrázok č. 32 env súbor vytvorený pre potreby skenovacieho kontajneru**
(Zdroj: Vlastné spracovanie)

Ako je možné vidieť na obrázku, tak súbor obsahuje viacero premenných. Riadky, ktoré začínajú so znakom # alebo sú prázdne, slúžia len ako poznámky alebo sú systémom ignorované. Ostatné riadky reprezentujú vzťah <názov premennej>=<hodnota premennej>.

Medzi premenné patrí na príklad cesta k zložke, do ktorej vkladám súbory, ktoré chcem dostať do kontajneru, a taktiež aj cesta k zložke, do ktorej chcem ukladať výstupné súbory. V ďalších častiach vysvetlím práve to, ako sa nadstavuje mapovanie medzi danými súbormi vo vnútri kontajneru a medzi hostiteľským systémom.

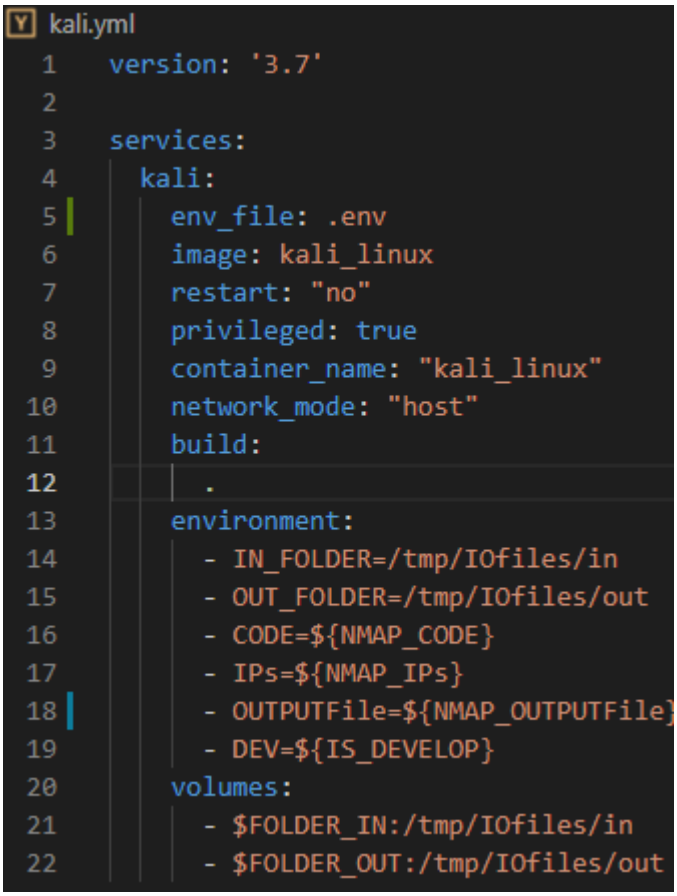
Ďalšie premenné, ako na príklad NMAP_CODE a NMAP_IPs slúžia na definovanie toho, z akého súboru má systém brať nmap kód pre spustenie skenovania a taktiež aj to, odkiaľ má brať systém súbor s IP adresami, ktoré bude skenovať. Takáto premenná existuje pre to, aby bolo možné uchovávať viacero nmap kódov a len si vďaka premennej medzi nimi vyberať. Taktiež NMAP_IPs uľahčí presun riešenia medzi viacerými sieťami bez toho, aby sme stratili prácu z predošlej siete. Ďalšou z premenných je NMAP_OUT, ktorá definuje názov súboru, do ktorého bude ukladaný výstup zo skenovania. Riešenie taktiež obsahuje opatrenia voči tomu, aby nedochádzalo k prepisu výstupu s rovnakým názvom. Dané riešenie bude predstavené v časti run.sh.

Poslednou z premenných je IS_DEVELOP, ktorá stanovuje to, či chce užívateľ spustiť kontajner na editáciu, poprípade na normálnu prácu v ňom, ako v bežnom systéme. V prípade že je zadaná 0, tak dôjde iba k spusteniu kódu z premennej NMAP_CODE a uloženiu výstupu do vopred nadefinovaného súboru.

#### 3.1.3.2 Docker Compose

Ďalším z nástrojov, ktoré poskytuje Docker, je Docker Compose. Docker Compose umožňuje spustenie aplikácii, ktoré sa skladajú z viacerých kontajnerov. Taktiež umožňuje základné nastavenie toho, ako vytvoriť kontajner pri tom, ako sa buduje z image. V mojom prípade ho použijem primárne na to, aby bolo riešenie v čo najväčšej miere zautomatizované. Tým spôsobom je možné to, aby dané riešenie použila aj osoba, ktorá nemá hlboké znalosti a skúsenosti s dockerom.

Nasledujúci obrázok obsahuje moje nastavenie súboru yaml, ktorý slúži práve pre definíciu prostriedkov pre Docker Compose.




**Obrázok č. 33 YAML súbor definujúci základné nastavenia pre budovanie kontajneru**
(Zdroj: Vlastné spracovanie)

Prvým nastavením je **env_file**, ktoré umožňuje meniť súbor, z ktorého systém prevezme premenné .env. Týmto štýlom je možné flexibilne používať viacero prednastavených skenovaní a meniť len tento jeden parameter.

Druhým nastavením je „**image**“, to určuje to, z akého image má vybudovať kontajner. V prípade že image s týmto názvom nenájde, tak ho vybuduje zo zdroju, ktorý je uvedený v Dockerfile a pomenuje podľa danej hodnoty, v tomto riešení kali_linux. Dockerfile bude predstavený o pár strán nižšie.

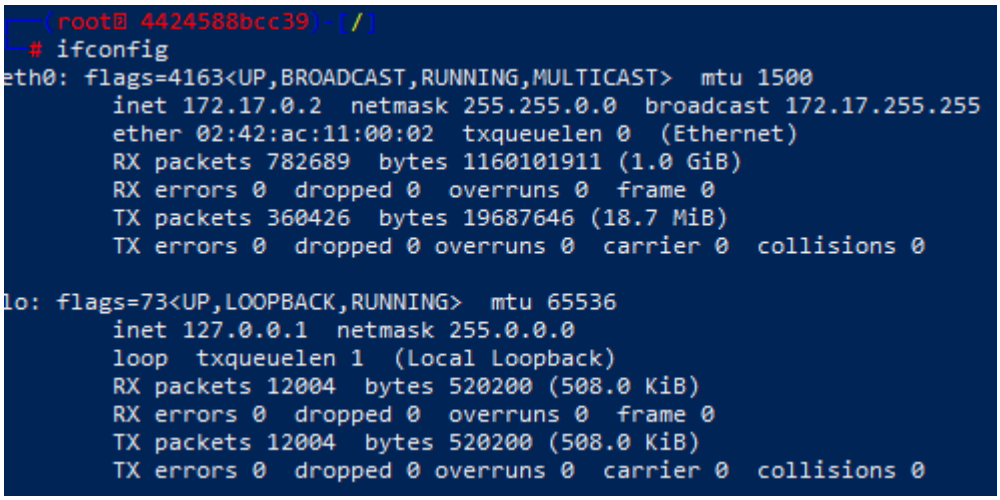
Ďalšími dôležitými nastaveniami je „**restart**“, ktoré stanovuje to, čo má robiť kontajner po vykonaní príkazu, ktorý spúšťa pri štarte. V prípade, kedy by bolo v env súbore zadané IS_DEVELOP=0 a pre restart: always, tak by dochádzalo k stále opakovanému vykonávaniu kódu, ktorý je umiestnený v Code.txt, to znamená k skenovaniu siete. Tomu sa snažím prirodzene zabrániť, a preto nastavujem restart na „no“.

Medzi základné nastavenia nutné pre skenovanie patrí taktiež aj nastavenie „**privileged**“ ako true. Takéto nastavenie dáva kontajneru rovnaké práva ako má hostiteľské zariadenie.

To je potrebné pre to, aby mal kontajner prístup k sieťovým prvkom a podobne. Prirodzene to prináša aj negatívne aspekty, a tými sú možnosť zneužitia a napadnutia zariadenia. To v mojom riešení eliminujem tým, že sa kontajner buduje a spúšťa s užívateľom, ktorý nemá správcovské oprávnenia.

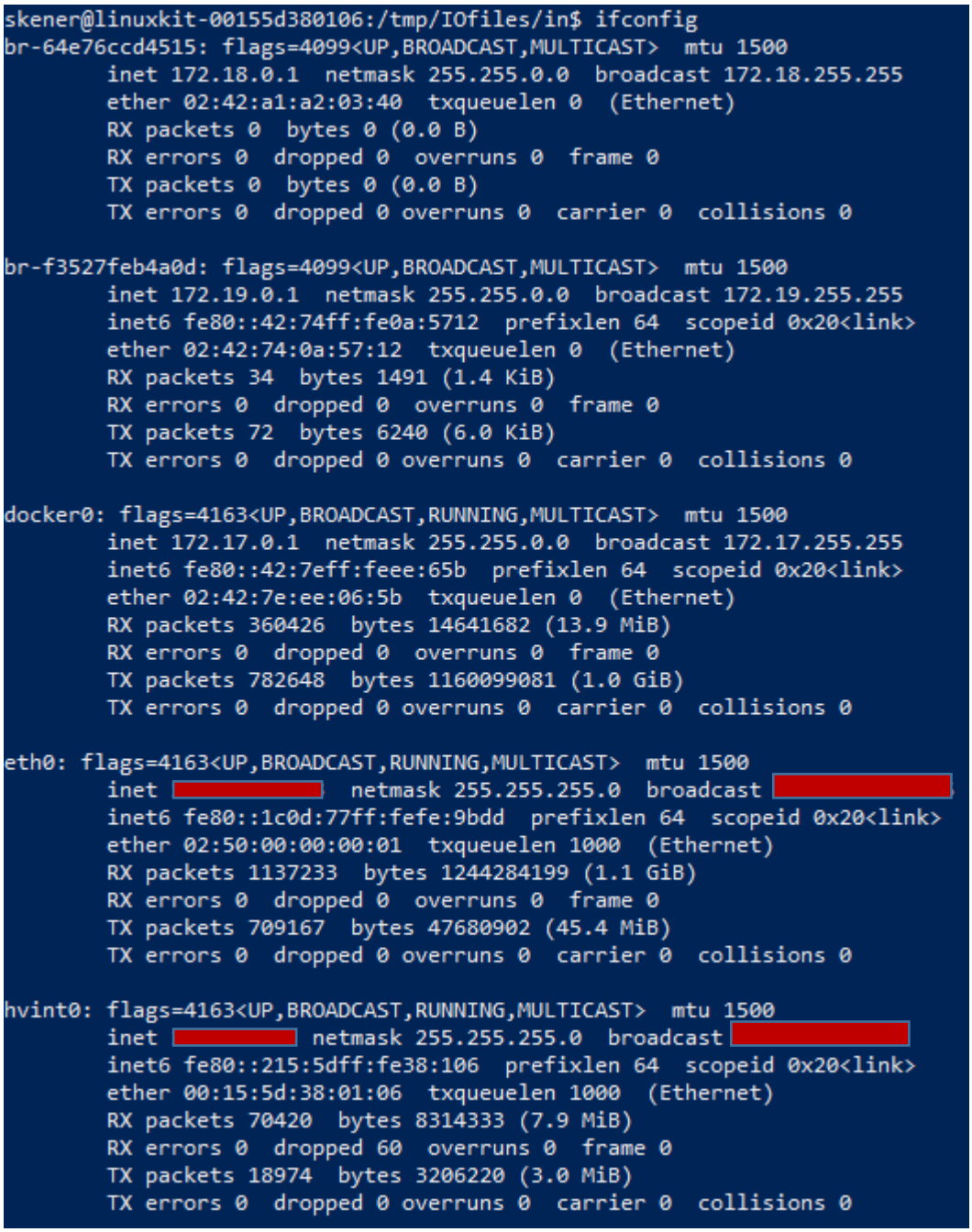
Ďalšími nastaveniami sú „**container_name**“ a „**network_mode**“. Container_name stanovuje to, ako bude kontajner po vybudovaní pomenovaný. Network_mode už je pre riešenie dôležitejší. Vďaka nastaveniu na „host“ dostáva kontajner rovnaké sieťové nastavenie ako hostiteľské zariadenie. To mu umožňuje vidieť aj na siete mimo docker. V prípade že by bolo nastavenie iné, na príklad „bridge“, tak by kontajner neumožňoval skenovanie vonkajších sietí, ale len sietí, ktoré sú vytvorené vo vnútri dockeru. Ako príklad rozdielu medzi nastaveniami prikladám výstup z príkazu ifconfig, z kontajneru spusteného so sieťovým nastavením ako bridge, a z kontajneru spusteného ako host. Pre pochopenie, ifconfig slúži na zobrazenie a konfiguráciu sieťového rozhrania systému.

Na prvom obrázku je možné vidieť výstup z kontajneru nastaveného na „bridge“.




**Obrázok č. 34 ifconfig z kontajneru so sieťovým nastavením bridge**
(Zdroj: Vlastné spracovanie)

Na obrazku druhom, je výstup z kontajneru so sieťovým nastavením host.

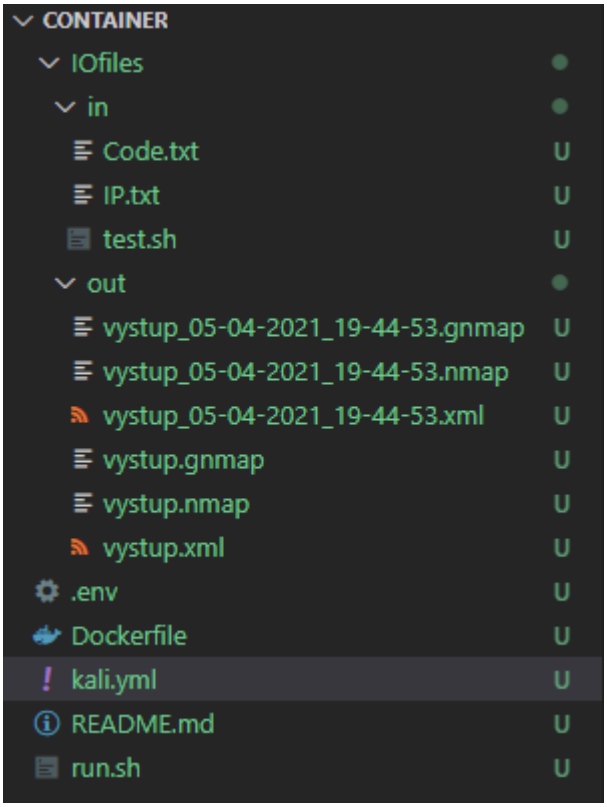



**Obrázok č. 35 Výstup z ifconfig v kontajneri so sieťovým nastavením host**
(Zdroj: Vlastné spracovanie)

Je možné si všimnúť že výstup obsahuje nie len vnútorné siete v rámci dockeru, čiže 172.X.X.X, ale taktiež aj vonkajšie siete, ktoré sú z dôvodu bezpečnosti prekryté.

Medzi ďalšie nastavenia yaml súboru patrí „**build**“, ktorý obsahuje informácie o tom, ako zostaviť image. Tieto informácie sú obsiahnuté v súbore dockerfile, ktorý sa nachádza v rovnakej zložke ako kali.yml. Z toho dôvodu obsahuje len „.“, ktorá definuje umiestnenie daného dockerfile. V inom prípade by mohol build obsahovať aj cestu

a názov súboru, z ktorého má zostaviť image. Na obrázku znázorňujem to, ako sú jednotlivé súbory a zložky v projekte rozmiestnené.



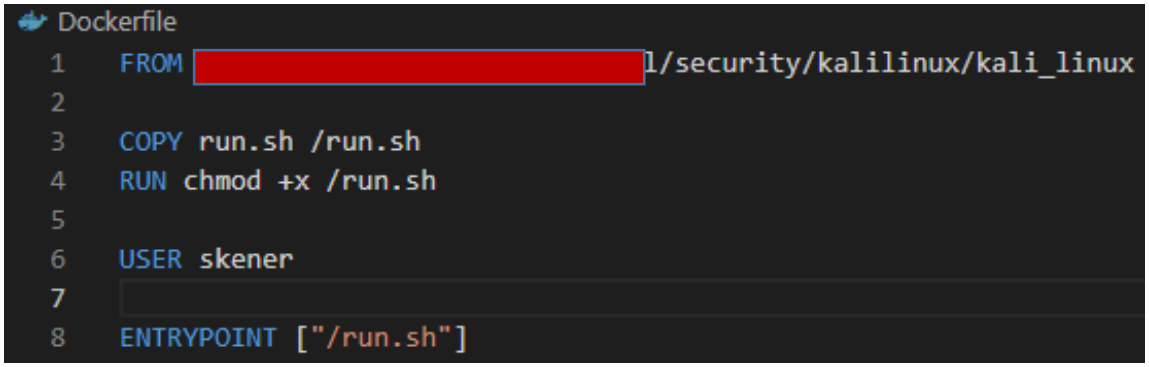

**Obrázok č. 36 Rozmiestnenie súborov a zložiek v projekte**
(Zdroj: Vlastné spracovanie)

Nasleduje nastavenie „**environment**“, ktoré definuje systémové premenné, ktoré budú dostupné aj v rámci spusteného kontajneru. Opäť majú vzťah <Nazov Premennej>=<Hodnota Premennej>. Medzi jednotlivými premennými je možné vidieť aj normálnu hodnotu ako na príklad /tmp/IOfiles/in, ale aj hodnotu, ktorá sa nachádza v premennej. Tieto premenné, na príklad NMAP_CODE, pochádzajú práve zo súboru .env a týmto spôsobom sú ďalej predávané systému.

Posledným nastavením v rámci súboru yaml je „**volumes**“. Dané nastavenie definuje mapovanie zložiek medzi hostiteľským zariadením a kontajnerom. Opäť je možné vidieť spojenie premennej, ktorá pochádza z .env súboru. Na príklad $FOLDER_IN, ktorá obsahuje cestu v rámci hostiteľského zariadenia, a cestou v kontajneri. Takéto nastavenie funguje tak, že čokoľvek sa v rámci nastavených zložiek, či už v kontajneri alebo hostiteľskom zariadení zmení, tak dôjde k zmene aj na druhej strane. Týmto spôsobom je možné pomerne jednoducho predávať súbory zo zariadenia do kontajneru a naopak.

### 3.1.3.3 Dockerfile

Dockerfile, ako už bolo viacej krát spomínané slúži na zostavenie image. Taktiež poskytuje prostriedky pre nastavenie toho, ako sa má zachovať kontajner pri spustení. Pre daný projekt som vytvoril nasledujúci dockerfile.




**Obrázok č. 37 Zobrazenie dockerfile vytvoreného pre potreby projektu**
(Zdroj: Vlastné spracovanie)

Prvou položkou, ktorá sa v dockerfile nachádza je „**FROM**“ a priradená hodnota. FROM definuje to, odkiaľ má čerpať image. V takomto prípade sa jedná o internú registry, do ktorej som nahrával mnou upravenú image. Tej som sa venoval v časti „Nastavenie image“ a nasledujúcej časti „Umiestnenie image v Docker registry“.

Ďalším prvkom je prvok „**COPY**“, ktorý umožňuje kopírovať súbory z hostiteľského zariadenia do kontajneru. Oproti „Volumes“ je rozdiel ten, že sa jedná len o jednorazový proces, ktorý ďalej nereaguje na zmeny súboru, či už v kontajneri, alebo v zariadení. V tomto prípade do kontajneru kopírujem súbor run.sh, ktorý obsahuje skript, ktorý sa má spustiť pri štartovaní kontajneru. Tento skript bude popísaný a zobrazený nižšie.

Po „COPY“ nasleduje „**RUN**“. Ten umožňuje spúšťať príkazy priamo v rámci kontajneru. V danom prípade pomocou chmod +x, ktorý už som popisoval pri tvorbe image, nastavujem súbor run.sh ako spustiteľný.

Jednou z posledný položiek je „**USER**“, práve táto položka umožňuje to, aby sa kontajner spúšťal s užívateľom, ktorý nemá správcovské oprávnenia. Jeho tvorbe a náležitým procesom som sa venoval opäť v časti Nastavenie image.

Poslednou položkou je „**ENTRYPOINT**“, ktorá stanovuje to, čo sa stane po spustení kontajneru. V tomto riešení je položka nastavená na „/run.sh“, to znamená, že hneď po vytvorení kontajneru dôjde k spusteniu súboru so skriptom, ktorý som vytvoril.

### 3.1.3.4 run.sh

Daný súbor už nemôžeme zaradiť medzi prostriedky poskytované dockerom. Jedná sa o bežný súbor obsahujúci shellovský skript. Popisovaný skript som napísal tak, aby som v čo najväčšej miere zautomatizoval spúšťanie skenovania a zároveň bral ohľad na možné komplikácie. Samotný skript s jednotlivými časťami je obsiahnutý na obrázku nižšie.

run.sh

```
#!/bin/bash
if [ $DEV == 1 ]
then
    echo IS_DEVELOP je nastaveny na 1, z toho dovodu spustam kontajner pripraveny na editaciu.
    tail -f /dev/null
elif [ -s $IN_FOLDER/$CODE ]
then
    echo IS_DEVELOP je nastaveny na 0 a zaroven obsah $CODE nie je nulovy, preto spustam scan s kódom z $CODE.
    cd $OUT_FOLDER

    #Nastavovanie vystupneho suboru a jeho nazvu
    if [ -f $OUTPUTFile.xml ] || [ -f $OUTPUTFile.nmap ] || [ -f $OUTPUTFile.gnmap ]
    then
        if [ -s $OUTPUTFile.xml ] || [ -s $OUTPUTFile.nmap ] || [ -s $OUTPUTFile.gnmap ]
        then
            NOW='_'`date +%d-%m-%Y_%H-%M-%S`
            echo $OUTPUTFile uz existuje, preto novy vystup ukladam ako $OUTPUTFile$NOW.
            OUTPUT=$OUTPUTFile$NOW
        else
            echo $OUTPUTFile uz existuje, avsak je prazdny, preto dochadza k jeho prepisu.
            OUTPUT=$OUTPUTFile
        fi
    else
        echo Ako vystupny subor nastavujem $OUTPUTFile
        OUTPUT=$OUTPUTFile
    fi

    #Uprava suboru s Code pre pripad ze by tam boli nadbytocne znaky
    cd $IN_FOLDER
    tr -cd [:print:] < $CODE > CodeTRIM.txt && cp CodeTRIM.txt $CODE && rm CodeTRIM.txt
    #Ziskanie a volani kodu zo suboru
    read line < $CODE
    eval echo Volam $line
    eval $line
else
    printf "$CODE je prazdny, z toho dovodu nie je mozne spustit scan.\nKontajner sa vypina.\n" && exit
fi
```

**Obrázok č. 38 run.sh skript, ktorý sa spúšťa pri štarte kontajneru**
(Zdroj: Vlastné spracovanie)

Prvým riadkom skriptu je „#!/bin/bash“, tento text stanovuje to, o aký typ shellu bude daný kód spracovávať. Samotný skript sa skladá z niekoľkých základných podmienok.

Prvou z nich riešim to, či chce užívateľ spúšťať kontajner na editáciu a normálne použitie, alebo či chce spustiť priamo prednastavený skener. V aktuálnom riešení kontrolujem hodnotu **$DEV**. Jedná sa o premennú, ktorú nastavujem v kali.yml. Tam ju nastavujem z premennej ktorú zadávam v .env súbore. V prípade že užívateľ zadal hodnotu 1, tak vypíše hlášku, že je kontajner spustený pre editáciu.

Taktiež spúšťa príkaz „**tail -f /dev/null**“. Tento príkaz zaisťuje to, že kontajner bude bežať neustále. Bežne by kontajner spracoval nastavený príkaz a došlo by k jeho vypnutiu, poprípade k jeho reštartovaniu. Príkaz tail funguje tak, že zobrazí niekoľko posledných riadkov zo súboru, ktorý identifikujem v rámci príkazu. Parameter -f (--follow), zase slúži na to, aby sa nejednalo o jednorazovú akciu, ale aby dochádzalo k neustálemu sledovaniu a zobrazovaniu obsahu súboru. V kombinácii s /dev/null, ktorý sa v rámci kontajneru nepoužíva dochádza k procesu, ktorý nikdy neskončí, a zároveň nedáva žiaden výstup. Týmto spôsobom som schopný držať kontajner spustený, a tak ho môžem upravovať a pracovať v ňom ako v normálnom systéme. Samotná podmienka a jej vyhodnotenie je lepšie viditeľná na nasledujúcom vloženom kóde:

```
if [ $DEV == 1 ]
then
    echo IS_DEVELOP je nastaveny na 1, z toho dovodu spustam kontajner pripraveny na editaciu.
    tail -f /dev/null
```

**Obrázok č. 39 Podrobnejšie zobrazenie prvej podmienky**
(Zdroj: Vlastné spracovanie)

Príklad výpisu pri spustení kontajneru s nastavenou premennou $DEV na 1 je možné vidieť na obrázku nižšie.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
IS_DEVELOP je nastaveny na 1, z toho dovodu spustam kontajner pripraveny na editaciu.
```

**Obrázok č. 40 Výpis kontajneru pri spustení s nastavenou premennou DEV na 1**
(Zdroj: Vlastné spracovanie)

Obecnejší postup budovania, spúšťania a napájania sa na kontajner je vysvetlený v ďalších častiach práce.

V prípade že podmienka neprejde, a teda že $DEV nie je nastavené na 1, tak dochádza k druhej podmienke „**elif [ -s $IN_FOLDER/$CODE ]**“. Táto podmienka kontroluje to, či existuje súbor s kódom, ktorý sme zadávali do súboru .env, a taktiež aj to či niečo obsahuje a nie je prázdny. Pokiaľ súbor nie je prázdny, a teda podmienka prejde, tak opäť dochádza k výpisu a informovaní o tom, že dôjde k spusteniu skenovania. V rámci tejto podmienky sa taktiež kontroluje premenná, ktorú sme v .env nadefinovali ako názov výstupného súboru. Prvou kontrolou je to, či existujú súbory s daným názvom. K tomu slúži nasledujúci riadok:

```
if [ -f $OUTPUTFile.xml ] || [ -f $OUTPUTFile.nmap ] || [ -f $OUTPUTFile.gnmap ]
```

V prípade že súbor existuje, tak ďalej kontrolujem to, či dané súbory obsahujú nejaké dáta. K tomu slúži nasledujúci riadok:

```
if [ -s $OUTPUTFile.xml ] || [ -s $OUTPUTFile.nmap ] || [ -s $OUTPUTFile.gnmap ]
```

V prípade že je daná podmienka vyhodnotená ako pravdivá, a teda že súbor obsahuje dáta, tak dôjde k automatickému získaniu aktuálneho dátumu a času, a doplní ho k nami predvolenému názvu výstupu. Príklad toho, ako systém na to reaguje je možné vidieť na nasledujúcom obrázku.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
IS_DEVELOP je nastaveny na 0 a zaroven obsah Code.txt nie je nulovy, preto spustam scan s kódom z Code.txt.
vystup uz existuje, preto novy vystup ukladam ako vystup_06-04-2021_13-08-06.
```

**Obrázok č. 41 Výpis systému v prípade že súbor existuje a nie je prázdny**
(Zdroj: Vlastné spracovanie)

Naopak, v prípade že daná podmienka nie je vyhodnotená ako pravdivá, a teda že súbory síce existujú ale neobsahujú žiadne dáta, tak dôjde k ich prepisu. Reakcia na danú situáciu je zachytená na obrázku nižšie.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
IS_DEVELOP je nastaveny na 0 a zaroven obsah Code.txt nie je nulovy, preto spustam scan s kódom z Code.txt.
vystup uz existuje, avsak je prazdny, preto dochadza k jeho prepisu.
```

**Obrázok č. 42 Výpis systému v prípade že súbor existuje a je prázdny**
(Zdroj: Vlastné spracovanie)

Vyššie zmienené scenáre sa uskutočnia len v prípade, že vyhodnotilo podmienku o existencii súborov ako pravdivú. V prípade že podmienku vyhodnotí ako nepravdivú, tak situácia nevyžaduje nijaké zložitejšie procedúry a jednoducho priradí názov výstupného súboru z nami nadefinovanej premennej. Výpis systému v takomto prípade je nasledujúci.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
IS_DEVELOP je nastaveny na 0 a zaroven obsah Code.txt nie je nulovy, preto spustam scan s kódom z Code.txt.
Ako vystupny subor nastavujem vystup
```

**Obrázok č. 43 Výpis systému v prípade že súbor neexistuje**
(Zdroj: Vlastné spracovanie)

Kompletný proces kontroly a nastavovania názvu výstupného súboru, je možné vidieť na nasledujúcom obrázku.

```
#Nastavovanie vystupneho suboru a jeho nazvu
if [ -f $OUTPUTFile.xml ] || [ -f $OUTPUTFile.nmap ] || [ -f $OUTPUTFile.gnmap ]
then
    if [ -s $OUTPUTFile.xml ] || [ -s $OUTPUTFile.nmap ] || [ -s $OUTPUTFile.gnmap ]
    then
        NOW='_'`date +%d-%m-%Y_%H-%M-%S`
        echo $OUTPUTFile uz existuje, preto novy vystup ukladam ako $OUTPUTFile$NOW.
        OUTPUT=$OUTPUTFile$NOW
    else
        echo $OUTPUTFile uz existuje, avsak je prazdny, preto dochadza k jeho prepisu.
        OUTPUT=$OUTPUTFile
    fi
else
    echo Ako vystupny subor nastavujem $OUTPUTFile
    OUTPUT=$OUTPUTFile
fi
```

**Obrázok č. 44 Proces vyhodnocovania a nastavovania názvu výstupného súboru**
(Zdroj: Vlastné spracovanie)

Nehľadiac na to, ako dopadne vyhodnocovanie existencie výstupného súboru s daným názvom, tak dochádza k volaniu kódu. Samozrejme len v prípade, že sa nachádzame v podmienke, ktorá overila to, či existuje súbor s kódom a taktiež aj to, či obsahuje nejaké dáta. Pred volaním pre istotu robím osekávanie súboru, pre prípad že by sa náhodou kód nenachádzal na prvom riadku. Požadované osekávanie zabezpečuje nasledujúci riadok.

```
tr -cd [:print:] < $CODE > CodeTRIM.txt && cp CodeTRIM.txt $CODE && rm CodeTRIM.txt
```

Tento jednoduchý kód najskôr spracuje a oseká nami navolený súbor s kódom a výstup vloží do súboru CodeTRIM.txt. Následne kopírujem obsah CodeTRIM.txt do mnou definovaného súboru s kódom. Na konci mažem pomocný CodeTRIM.txt súbor.

Vyššie spomínaná úprava je nutná z toho dôvodu, že následne načítavam obsah daného súboru do premennej. Takéto načítavanie však prebieha po riadkoch a preto bolo nutné to, aby sa kód nachádzal vždy na prvom riadku. Po načítaní premennej je možné ju spustiť. Problém však nastáva v tom, že kód ktorý volám, obsahuje ďalšie premenné. Z toho dôvodu je nutné k volaniu pridať taktiež príkaz **eval**, ktorý dokáže takéto premenné v rámci premennej spracovať, a fungovať s nimi ako s hodnotami. Príklad volaného kódu v rámci súboru s textom je na nasledujúcom obrázku.

```
IOfiles > in >  Code.txt
  1    nmap -sV --unprivileged -iL ${IN_FOLDER}/${IPs} -oA ${OUT_FOLDER}/${OUTPUT}
```

**Obrázok č. 45 Súbor obsahujúci kód s premennými**
(Zdroj: Vlastné spracovanie)

Na obrázku je možné vidieť že obsahuje viacero premenných. Jednou z premenných je „OUTPUT“, ktorý obsahuje názov výstupného súboru. Daný názov sa nastavoval na riadkoch vyššie. Ďalšou z premenných je na príklad „**IPs**“. Táto premenná obsahuje názov súboru, ktorý v sebe ukladá IP adresy ktoré chceme skenovať.

Pre lepšiu prehľadnosť opäť prikladám obrázok, ktorý obsahuje vyššie popísanú logiku.

```
#Uprava suboru s Code pre pripad ze by tam boli nadbytocne znaky
cd $IN_FOLDER
tr -cd [:print:] < $CODE > CodeTRIM.txt && cp CodeTRIM.txt $CODE && rm CodeTRIM.txt
#Ziskanie a volanie kodu zo suboru
read line < $CODE
eval echo Volam $line
eval $line
```

**Obrázok č. 46 Spracovanie súboru s kódom a volanie daného kódu**
(Zdroj: Vlastné spracovanie)

Pre príklad výpisu takéhoto kompletného volania prikladám nasledujúci obrázok.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
IS_DEVELOP je nastaveny na 0 a zaroven obsah Code.txt nie je nulovy, preto spustam scan s kódom z Code.txt.
Ako vystupny subor nastavujem vystup
Volam nmap -sV --unprivileged -iL /tmp/IOfiles/in/IP.txt -oA /tmp/IOfiles/out/vystup
Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-06 18:58 UTC
```

**Obrázok č. 47 Celkový výpis zo spustenia kontajneru a zavolania nmap skenu**
(Zdroj: Vlastné spracovanie)

Ďalšie riadky by obsahovali samotný výstup zo skenovania. Výstup sa taktiež vygeneruje aj do jednotlivých súborov. Viac informácii a taktiež popis nmap skenovania a jeho výstupov popíšem v kapitole „Tvorba skenovacieho nástroju“.

Ako posledné v rámci run.sh je ešte nutné uviesť a popísať situáciu, kedy systém vyhodnotí nie len podmienku „if [ $DEV == 1 ]“, ale taktiež aj podmienku elif „[ -s $IN_FOLDER/$CODE ]“ ako nepravdivú. To by znamenalo to, že súbor, ktorý sme zadali ako zdroj skenovacieho kódu je buď to prázdny, alebo neexistuje. Pre taký prípad som vytvoril nasledujúcu časť skriptu.

```
else
    printf "$CODE je prazdny, z toho dovodu nie je mozne spustit scan.\nKontajner sa vypina.\n" && exit
```

**Obrázok č. 48 Reakcia na prázdny Code súbor a zároveň DEV nerovnajúci sa 1**
(Zdroj: Vlastné spracovanie)

Toto riešenie neobsahuje žiadnu zložitú logiku. Jeho účelom je primárne informovať užívateľa o tom, z akého dôvodu nedošlo k mienenému skenovaniu. Výstupný výpis je možné vidieť na nasledujúcom obrázku.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
Code.txt je prazdny, z toho dovodu nie je mozne spustit scan.
Kontajner sa vypina.
```

**Obrázok č. 49 Výpis z kontajneru v prípade že súbor Code.txt neobsahuje kód**
(Zdroj: Vlastné spracovanie)

Na obrázku nižšie prikladám taktiež aj príklad toho, ako by to vyzeralo, keby neexistovala daná podmienka. Kontajner by sa jednoducho vypol a užívateľovi by neposkytol o tom žiadnu informáciu.

```
PS C:\Users\peter\git\scan\container> docker logs kali_linux
PS C:\Users\peter\git\scan\container>
```

**Obrázok č. 50 Spustenie kontajneru bez upozornenia na chýbajúci súbor Code**
(Zdroj: Vlastné spracovanie)

#### 3.1.3.5 README.md

Návod toho, ako kontajner spúšťať, ako funguje a ako sa s ním pracuje je obsiahnutý v súbore README.md. Jedná sa o súbor, ktorý sa bežne používa pre základnú dokumentáciu k projektom v rámci gitu. Jeho štruktúra vyzerá nasledovne:

README.md > # Kontajner pre mapovanie siete

```
# Kontajner pre mapovanie siete
## Prerekvizity:
* nastavené hosts pre gitlab a docker registries
* byť na VPN poprípade na internej sieti

## Súbor .env
Súbor slúži na nastavenie základných vstupných parametrov pre systém.
* **FOLDER_IN** - Definuje cestu k súboru, ktorý chceme mapovať do kontajneru ako vstupný
* **FOLDER_OUT** - Definuje cestu k súboru, ktorý chceme mapovať z kontajneru ako výstupný
* **NMAP_CODE** - Definuje názov súboru, z ktorého sa bude brať kód pre mapovanie
    * **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_IN
* **NMAP_IPs** - Definuje súbor s IP adresami ktoré chceme mapovať
    * **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_IN
* **NMAP_OUT** - Definuje názov výstupného súboru
    * V prípade že súbor s daným názvom už existuje a obsahuje data, tak vytvorí nové súbor
    * **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_OUT
* **IS_DEVELOP** - Definuje to, akým spôsobom spustí kontajner
    * Ak zadáme hodnotu 1 - kontajner spustí tak, že je možné ho editovať, poprípade spúšťa
        * **POZOR** kontajner sa spúšťa s non-root užívateľom, takže bude možnosť editácie
    * Ak zadáme inú hodnotu ako 1 - kontajner automaticky spustí kód z NMAP_CODE a vytvorí

## Súbor s nmap kódom
Podmienky pre správne fungovanie súboru s kódom:
* Textový súbor musí obsahovať práve jeden nmap kód
    * V prípade že má užívateľ pripravených viacero rôznych skenov, tak ich doporučujem pre
```

**Obrázok č. 51 Štruktúra README.md**
(Zdroj: Vlastné spracovanie)

Takýto súbor je špecifický hlavne tým, že z vyššie zobrazenej štruktúry, si automaticky gitlab vygeneruje prehľadný popis projektu. Príklad toho, ako zobrazí daný súbor gitlab je nasledujúci:

README.md

## Kontajner pre mapovanie siete

### Prerekvizity:

- nastavené hosts pre gitlab a docker registries
- byť na VPN poprípade na internej sieti

### Súbor .env

Súbor slúži na nastavenie základných vstupných parametrov pre systém.

- **FOLDER_IN** - Definuje cestu k súboru, ktorý chceme mapovať do kontajneru ako vstupný
- **FOLDER_OUT** - Definuje cestu k súboru, ktorý chceme mapovať z kontajneru ako výstupný
- **NMAP_CODE** - Definuje názov súboru, z ktorého sa bude brať kód pre mapovanie
  - **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_IN
- **NMAP_IPs** - Definuje súbor s IP adresami ktoré chceme mapovať
  - **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_IN
- **NMAP_OUT** - Definuje názov výstupného súboru
  - V prípade že súbor s daným názvom už existuje a obsahuje data, tak vytvorí nové súbory s datumovou príponov
  - **POZOR** súbor sa musí nachádzať v zložke, ktorá je definovaná ako FOLDER_OUT
- **IS_DEVELOP** - Definuje to, akým spôsobom spustí kontajner
  - Ak zadáme hodnotu 1 - kontajner spustí tak, že je možné ho editovať, poprípade spúšťať nmap a iné príkazy priamo z neho
    - **POZOR** kontajner sa spúšťa s non-root užívateľom, takže bude možnosť editácie obmedzená
  - Ak zadáme inú hodnotu ako 1 - kontajner automaticky spustí kód z NMAP_CODE a vytvorí výstup do NMAP_OUT

**Obrázok č. 52 Zobrazenie README v gitlabe**
(Zdroj: Vlastné spracovanie)

V rámci daného README opisujem aj to ako sa má kontajner spúšťať a ako fungujú jednotlivé príkazy s tým spojené. Pre porozumenie prikladám obrázok časti z README, ktorá sa práve spúšťaniu kontajneru venuje.

## Spúšťanie kontajneru

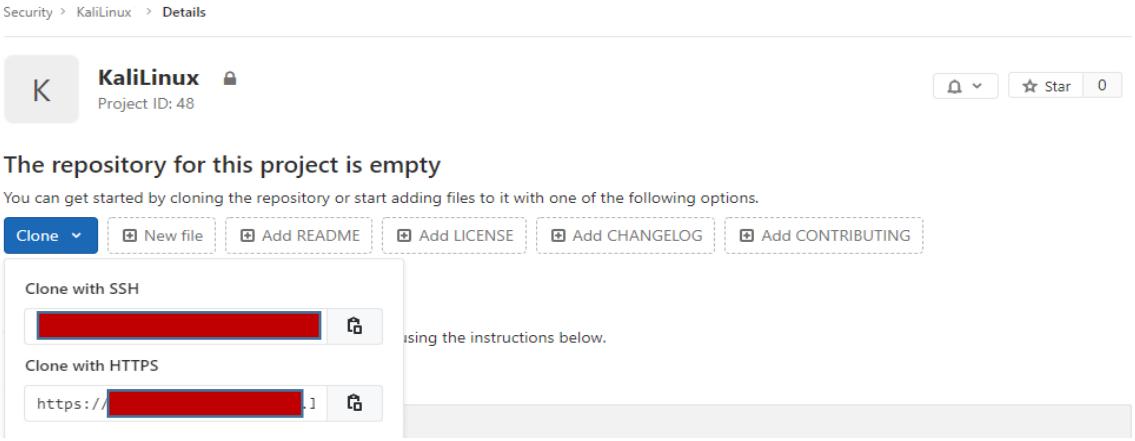
Pri spúšťaní a spravovaní kontajneru postačujú nasledujúce docker príkazy:

- **docker-compose -f kali.yml build --no-cache**
  - Pomocou tohoto príkazu dôjde k vybudovaniu image podľa dockerfile
- **docker-compose -f kali.yml up -d**
  - Pomocou tohoto príkazu dôjde k spusteniu kontajneru
  - príznak **-d** - stanovuje to či ho chceme v detach móde. To znamená, že kontajner sa spustí na pozadí a my môžeme ďalej používať terminál na prácu
    - V prípade že nás zaújima výstup z mapovania v termináli a neplánujeme upravovať kontajner, tak je možné zadať daný príkaz bez -d
- **POZOR** docker-compose príkazy sú viazané priamo na súbor. Z toho dôvodu je nutné to, aby bol užívateľ v rámci terminálu v zložke, kde sa nachádza Dockerfile, Kali.yml, .env a run.sh
- **docker logs kali_linux**
  - Zobrazí informácie z kontajneru, v princípe to isté, čo by nám zobrazilo v prípade že nepoužijeme pri spúšťaní -d
- **docker exec -it kali_linux /bin/bash**
  - V prípade že je IS_DEVELOP nastavený na hodnotu 1, tak tento príkaz umožňuje terminálové spojenie s kontajnerom. Vďaka tomu je možné v ňom pracovať ako v bežnom systéme

**Obrázok č. 53 Popis spúšťania kontajneru v rámci README**
(Zdroj: Vlastné spracovanie)

### 3.1.4 Umiestnenie prostriedkov v GIT repository

Pre zachovanie ľahkej prenositeľnosti je nutné, aby boli vyššie popisované prostriedky dostupné všetkým oprávneným užívateľom. Projekt, ktorý som vytvoril v časti „Umiestnenie image v Docker registry“ umožňuje nie len uloženie a distribúciu docker image, ale taktiež aj bežného kódu a súborov. V rámci projektu v gitlabe stačí prejsť do úvodnej časti projektu. V rámci danej časti existuje tlačidlo, ktoré poskytuje adresu nutnú pre klonovanie danej repository zariadenia. Príklad takejto úvodnej stránky s tlačidlom je možné vidieť na obrázku nižšie.




**Obrázok č. 54 Úvodná projektová stránka v rámci gitlab**
(Zdroj: Vlastné spracovanie)

Vďaka danej adrese je možné to, aby som si v rámci zariadenia vytvoril lokálnu kópiu. To je možné pomocou nasledujúceho príkazu:

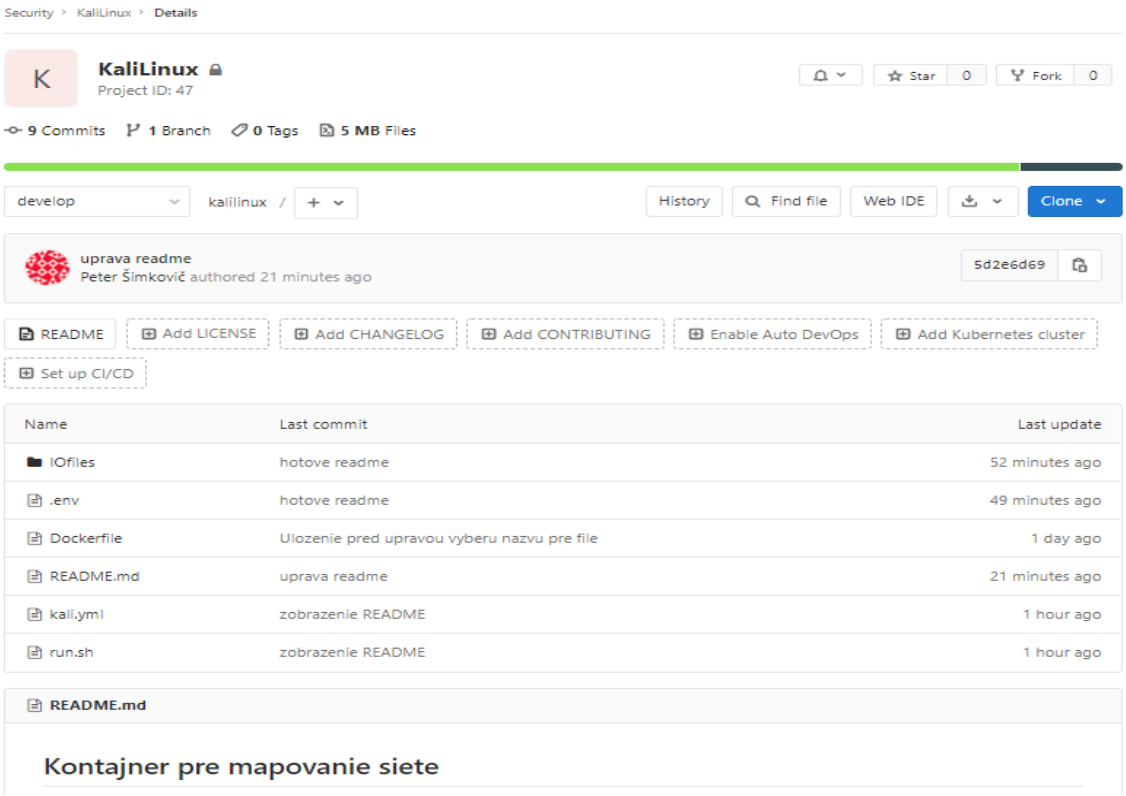

**Obrázok č. 55 Prvotné klonovanie gitového projektu**
(Zdroj: Vlastné spracovanie)

Výstupom z tohto príkazu je vytvorená prázdna zložka s názvom daného príkazu. Následne je možné danú zložku vyplniť so súbormi, ktoré som vytvoril v predošlých častiach. Po vyplnení je nutné to, aby sa nahrali všetky súbory do lokálnej kópie gitu. Na to slúži nasledujúci postup:

1.) git add .
   - „.“ Znázorňuje to, že chcem pridať všetky súbory, ktoré sa nachádzajú v danej zložke a jej pod-zložkách
   - Samotné add slúži na prípravu vybraných súborov na zapísanie

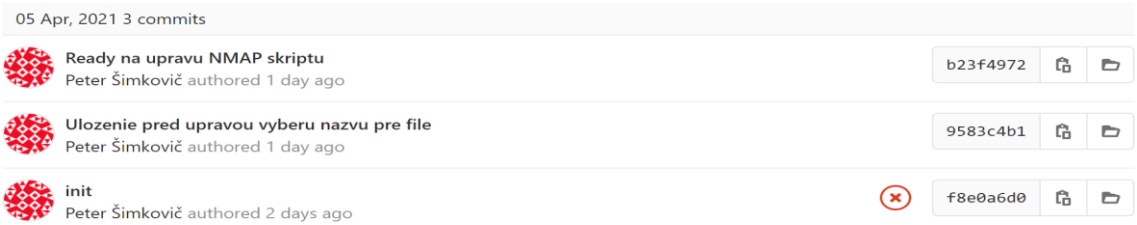
2.) git commit -m „Správa ktorá bude zaevidovaná v gitlab“
    - commit zapíše zmeny získané pomocou add do adresára gitu
3.) git push origin NazovDanejBranch
    - Tento príkaz zašle zapísané súbory z lokálneho adresára do adresára vzdialeného

Následným výstupom je to, že sa kópia všetkých súborov nachádza v git repository, z ktorej si môžu dané riešenie jednoducho sťahovať všetci oprávnený užívatelia. To či je užívateľ oprávnený na prístup k danému projektu je zabezpečené z viacerých hľadísk. Prvým z nich je to, že každý užívateľ, nehľadiac na projekt, musí mať k dispozícii privátny kľuč, ktorý odpovedá verejnému kľúču zadanému v rámci gitlab. Týmto spôsobom je možné spracovať autentifikáciu pomocou asymetrického šifrovania. Druhým krokom, ako sa dá riešenie zabezpečiť, je samotná autorizácia a nastavenie oprávnení pre každého užívateľa. Daný proces popisujem v rámci „Umiestnenia image v Docker registry“ kde nastavujem jednotlivým užívateľom role. Pokiaľ nie je užívateľ členom projektu, tak k nemú nemá prístup a dokonca ani nevie o jeho existencii. Príklad toho, ako vyzerá gitlab s daným projektom je možné vidieť na nasledujúcom obrázku.



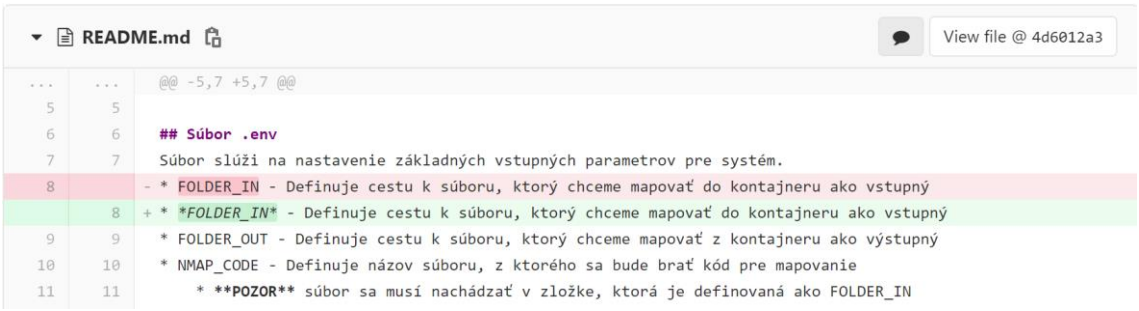

**Obrázok č. 56 Projekt v rámci gitlabu**
(Zdroj: Vlastné spracovanie)

Riešenie taktiež poskytuje logové záznamy pre každú činnosť, ktorá je s daným kódom vykonaná. Taktiež obsahuje aj históriu zmien, a tak umožňuje v prípade straty, alebo zavedenia chyby do riešenia jeho opravu a navrátenie do predošlého stavu. Zobrazenie histórie zmien je vidieť na ďalšom obrázku.




**Obrázok č. 57 História zmien v rámci gitového projektu**
(Zdroj: Vlastné spracovanie)

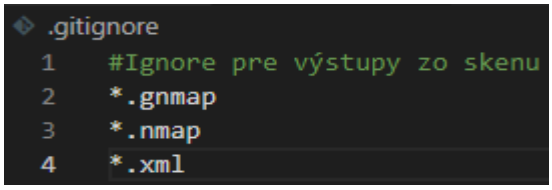
Príklad zobrazenia samotných zmien a stavu pred a po zmene je možné vidieť na nasledujúcom obrázku.




**Obrázok č. 58 Zobrazenie zmien v rámci jednotlivých súborov**
(Zdroj: Vlastné spracovanie)

Takéto riešenie je teda zabezpečené nie len z pohľadu neoprávneného prístupu, ale taktiež aj z pohľadu nežiadúcich zmien. Taktiež dochádza k pravidelnému zálohovaniu repository, čo opäť prispieva k bezpečnosti dát a informácii.

V rámci gitu je ešte nutné nastaviť to, aby nedochádzalo k zasielaniu výstupných súborov do repository. Tým by mohlo dôjsť k neoprávnenému nakladaniu s dátami získanými u klienta. K tomu slúži súbor .gitignore, v ktorom stačí zadať to, čo chceme aby git ignoroval a nezasielal do vzdialenej repository.




**Obrázok č. 59 Súbor .gitignore zaisťujúci to, aby sa nezdielali výstupy zo skenu**
(Zdroj: Vlastné spracovanie)

## 3.2 Tvorba skenovacieho skriptu

Z analýzy je zrejmé, že pre účely tohto systému bude najlepšou voľbou nástroj Nmap. Nástroj samotný poskytuje veľké množstvo nástrojov, ktoré uľahčujú a zefektívňujú proces skenovania.

V rámci časti, kde je popisovaný algoritmus a zmysel run.sh súboru, je taktiež vysvetlené aj to, akým štýlom bude do systému prenesený a spustený skript. Vďaka tomu zostáva už len navrhnúť a nastaviť nmap tak, aby čo najlepšie odpovedal podmienkam bežnej spoločnosti. Základná štruktúra volania skenovania vyzerá nasledujúco:

```
nmap -sV --unprivileged -iL ${IN_FOLDER}/${IPs} -oA ${OUT_FOLDER}/${OUTPUT} -A
```

- -sV – jedná sa o detekciu verzie služieb, ktoré bežia na otvorených portoch.
    - Informácia o verzii umožňuje identifikovať, to ak systém používa neaktualizované služby, ktoré tak môžu byť potencionálnym rizikom pre aktívum.
- --unprivileged – používa sa v prípade, že užívateľ nemá dostatočné oprávnenia.
    - V aktuálnej situácii je to použité z toho dôvodu, že v prípade že dochádza k skenovaniu z dockeru, tak samotné skenovanie prechádza cez ďalšieho hosta. Následne, keď dochádza k skenovaniu celého subnetu, tak docker rozpoznáva spustených hostov cez TTL, a keďže je medzi skenovaným subnetom, a dockerom ďalšie zariadenie, tak docker eviduje každé volanie ako úspešné, pretože prichádza odpoveď od prostredníka s TTL a nerieši obsah odpovede. Z toho dôvodu eviduje všetky zariadenia v sieti ako online a následne na tom skenovanie spadne.
- -iL – „input from list“ oznamuje nmapu, že zoznam ip adries je umiestnený v súbore
- ${IN_FOLDER}/${IPs} – označuje samotný súbor s ip adresami a jeho umiestnenie. V tomto prípade sú cesta a súbor reprezentované premennými, ktoré sú braté z .env súboru. Takéto riešenie je efektívne hlavne v tom prípade, že máme pripravené skenovanie viacerých sieti a nechceme prepisovať jednotlivé ip adresy, ale stačí len v .env zmeniť zdrojový súbor a skenovanie prebehne s novým súborom.

- -oA – umožňuje vytvoriť súbory s výstupom zo skenovania. V prípade oA sa jedná o tri súbory, kde každí je vhodní pre iné použitie.
- ${OUT_FOLDER}/${OUTPUT} – obdobne ako IN_FOLDER, reprezentuje názov a cestu k uloženiu súboru s jednotlivými výstupmi. Taktiež sa jedná o premenné, ktoré je možné zmeniť v rámci .env, a tak zaistiť že budú výstupy pre užívateľa ľahko rozpoznateľné a prehľadné.
- -A – invokuje takzvané agresívne skenovanie. Vďaka tomu bude výstup skenovania obsahovať viaceré informácie o jednotlivých zariadeniach a ich službach, ako sú na príklad informácie o OS, verziách, tracerouts a podobne.
    - Nevýhodou takéhoto skenovania je to, že je ľahšie detekovateľné v rámci siete. To však v mojom prípade nevadí, keďže sa jedná o schválené skenovanie.

Výstup z takéhoto skenovania je naozaj obsiahly a obsahuje veľké množstvo informácii. Na obrázku nižšie bude možné vidieť len pár riadkov z výstupu skeneru.

```
80/tcp    open  http              Microsoft IIS httpd 8.5
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/8.5
|_http-title: Ensembles
135/tcp   open  msrpc             Microsoft Windows RPC
139/tcp   open  netbios-ssn       Microsoft Windows netbios-ssn
443/tcp   open  ssl/https?
| ssl-cert: Subject: commonName=[redacted]
| Not valid before: 2015-08-18T08:44:42
|_Not valid after:  2016-02-17T08:44:42
|_ssl-date: 2021-05-02T14:58:07+00:00; 0s from scanner time.
445/tcp   open  microsoft-ds      Windows Server 2012 R2 Datacenter 9600 microsoft-ds
1717/tcp  open  fj-hdnet?
1972/tcp  open  intersys-cache    InterSystems Cache database
2000/tcp  open  tcpwrapped
3389/tcp  open  ssl/ms-wbt-server?
| ssl-cert: Subject: commonName=[redacted]
| Not valid before: 2021-01-08T10:59:07
|_Not valid after:  2021-07-10T10:59:07
5060/tcp  open  tcpwrapped
8443/tcp  open  ssl/http          Jetty 9.4.0.v20161208
```

**Obrázok č. 60 Zobrazenie výstupu zo skenovania bez využitia skriptov**
(Zdroj: Vlastné spracovanie)

Na takomto výstupe je možné pozorovať jednotlivé otvorené porty, verzie služieb ktoré na nich bežia, a taktiež aj OS. Pre skúsenú osobu by to boli dostatočné informácie pre spracovanie solídnej analýza a k návrhu opatrení a úprav. Tento nástroj však nie je určený

len pre takéto osoby, ale taktiež aj pre ľudí, ktorí nie sú v rámci danej problematiky dostatočne znalý. Pre to je nutné nmap volanie rozšíriť o skripty, ktoré overujú dané informácie o skenovaní s databázou zraniteľností, a aktívne vypisujú jednotlivé zraniteľnosti na jednotlivých zariadeniach. Inštaláciu a prípravu pre takéto skripty bolo možné vidieť v časti „Nastavenie image“, kde sťahujem a aktualizujem skripty nmap-vulners a vulscan z github repozitára. Oba skripty aktívne overujú jednotlivé služby a ich verzie voči online a offline databázam so zraniteľnosťami. V prípade že dôjde k identifikácii takejto zraniteľnosti, tak dôjde aj k vypísaniu CVE kódu zraniteľnosti, na základe ktorého je možné si zraniteľnosť vyhľadať, zistiť ako funguje a primárne zistiť to, ako je možné chybu napraviť.

Volanie nmap so skriptami je nasledujúce:

```
nmap --script nmap-vulners/vulners.nse,vulscan/vulscan.nse --script-args vulscandb=scipvuldb.csv -sV --unprivileged -iL ${IN_FOLDER}/${IPs} -oA ${OUT_FOLDER}/${OUTPUT} -A
```

Takéto volanie už obsahuje aj vyššie popisované skripty. V rámci argumentov došlo k pridaniu názvu zdrojovej databázy so známymi zraniteľnosťami, v tomto prípade scipvuldb.

## 3.3 Spracovanie výstupov a znázornenie opatrenia

Po úprave skriptu o nmap-vulners a vulscan začali byť výstupy o niečo lepšie spracovateľné pre bežného užívateľa. Obsahujú informácie o jednotlivých službách, o ich zraniteľnostiach a taktiež odkazujú na jednotlivé opatrenia. Keďže sa jedná o veľmi citlivé dáta, tak nie je možné zobraziť podrobný a reálny výstup zo skenovania. Ako príklad je uvedené to, ako je možné spracovať výstup zo skenovania.

```
22/tcp   open  ssh         OpenSSH 7.4p1 Debian 10+deb9u4 (protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:7.4p1:
|       MSF:ILITIES/UBUNTU-CVE-2019-6111/    5.8 https://vulners.com/metasploit/MSF:ILITIES/UBUNTU-CVE-2019-6111/    *EXPLOIT*
|       MSF:ILITIES/SUSE-CVE-2019-6111/ 5.8 https://vulners.com/metasploit/MSF:ILITIES/SUSE-CVE-2019-6111/  *EXPLOIT*
|       MSF:ILITIES/SUSE-CVE-2019-25017/    5.8 https://vulners.com/metasploit/MSF:ILITIES/SUSE-CVE-2019-25017/ *EXPLOIT*
|       MSF:ILITIES/REDHAT_LINUX-CVE-2019-6111/ 5.8 https://vulners.com/metasploit/MSF:ILITIES/REDHAT_LINUX-CVE-2019-6111/  *EXPLOIT*
```

**Obrázok č. 61 Znázornenie výstupu s príkladom zraniteľnosti**
(Zdroj: Vlastné spracovanie)

V rámci obrázku je možné vidieť názov chyby, jej kód, a aj to, akej služby sa priamo týka. Vďaka linku k zraniteľnosti je veľmi jednoduché sa s danou zraniteľnosťou

oboznámiť a nájsť jej popis aj s možnou nápravou. Taktiež výstup obsahuje aj úroveň zraniteľnosti, v prezentovanom prípade 5.8, čo znamená že sa jedná o zraniteľnosť kategórie „Medium“. Samotné rozpätie je v rozsahu 0-10.

Po otvorení linku so zraniteľnosťou je možné dôjsť k popisu samotného problému.

*„Problém bol zistený v OpenSSH 7.9. Kvôli implementácii scp odvodenej od 1983 rcp si server vyberá, ktoré súbory / adresáre sa odošlú klientovi. Klient scp však vykonáva iba zbežné overenie vráteného názvu objektu (zabránené sú len útoky na priechod adresárom). Škodlivý server scp (alebo útočník typu Man-in-The-Middle) môže prepísať ľubovoľné súbory v cieľovom adresári klienta scp. Ak sa vykoná rekurzívna operácia (-r), server môže manipulovať aj s podadresármi (napríklad prepísať súbor .ssh / authorized_keys).“* (12)

Príkladom riešenia takéhoto problému je jednoduchá aktualizácia, poprípade zváženie toho, či je nutné mať daný port otvorený. V rámci danej zraniteľnosti je veľmi jednoduché dohľadať aj to, akých služieb a akých verzii sa to týka, a podľa toho vyhľadať vhodnú aktualizáciu. V popisovanom prípade je daná zraniteľnosť ošetrená od verzie OpenSSH 8.0.

V reálnom prostredí by stačilo, aby osoba, ktorá skenovanie spracováva, predala správcovi danej siete, ktorý by si prevzal zodpovednosť za eliminovanie daných zraniteľností.

## 3.4 Zhrnutie

V rámci návrhovej časti došlo k vytvoreniu systému, ktorý umožňuje jednoduché skenovanie sieti a hľadanie zraniteľností. Systém je vytvorený tak, aby bol veľmi jednoducho a rýchlo implementovateľný akoukoľvek oprávnenou osobou v rámci spoločnosti. Taktiež sa bral ohľad na to, že väčšina zamestnancov nedisponuje dostatočnými znalosťami či už v oblasti linuxu, tak v oblasti bezpečnosti. Preto bol systém v čo najväčšej miere zautomatizovaný.

V rámci celého procesu tvorby systému bol bratý ohľad na bezpečnostné zásady a princípy. Pri vytváraní a úprave prostredia v rámci Dockeru došlo k zabezpečení na všetkých úrovniach. Či už na úrovni vytvorenej image, spusteného kontajneru, uloženia v rámci repozitára, tak aj zdieľania medzi užívateľmi.

## 3.5 Ekonomické zhodnotenie

V rámci navrhovania opatrení z hľadiska bezpečnosti, nie je možné počítať so ziskom z danej investície. Dôležitejším faktorom, ako vytvorený zisk, sú v tejto oblasti ušetrené náklady, ktoré by vznikli v prípade bezpečnostného incidentu. Preto je nutné pri návrhu opatrení brať ohľad na to, aby náklady na opatrenia neboli vyššie ako náklady z incidentu, ktorému mali zabrániť.

Pri spracovávaní tohto projektu nie je možné presne vyčísliť ušetrené náklady. Pre príklad je však možné uviesť, že výška pokuty v prípade porušenia, nezavedenia alebo nepripravenosti na nariadenie GDPR môže dosahovať až 20 000 000 eur, alebo 4% z celkového ročného obratu spoločnosti. Ďalším problémom, ktorý z nezabezpečenej siete vyplýva, môže byť ukradnutie projektov vo vývoji, poprípade hotových projektov. To znamená, že by spoločnosť mohla prísť nie len o renomé, ale taktiež aj o know-how a samotný produkt, ktorý poskytujú. Opäť sú takéto škody len ťažko vyčísliteľné, ale v prípade vývojových spoločností v danom obore, sa jedná o škody v rádoch desiatok miliónov českých korún.

Samotné náklady, ktoré z projektu plynú sú v porovnaní s vymenovanými stratami minimálne. Vďaka tomu, že spoločnosť disponuje potrebnou infraštruktúrou, ako je gitlab, docker registry a podobne, tak nie je nutné nijakým spôsobom zvyšovať náklady, či už na hardware, tak aj na software. Jediné náklady, ktoré pri realizácii projektu vznikli, sú náklady na ľudskú prácu. Z toho dôvodu, že sa jednalo o nástroj v rámci dockeru, čo aktuálne nie je v kombinácii s nmap v globálnej miere používané, tak som sa stretával s viacerými problémami, na ktoré bolo pomerne náročné nájsť riešenia. Napriek tomu trval návrh a implementácia približne 10 pracovných dní, čo znamená približne 80 hodín. Hrubé náklady na hodinu práce sú v tomto prípade 250 Kč. Čo sa týka nákladov na školenie, tak tie nie je nutné počítať, keďže je nástroj navrhnutý tak, aby ho dokázal používať akýkoľvek zamestnanec podieľajúci sa na bežnej vývojovej činnosti v spoločnosti. Vo výsledku sú teda náklady na projekt vo výške 20 000 Kč. Je možné teda prehlásiť, že náklady na nástroj sú nižšie ako prípadne náklady vzniknuté v súvislosti s bezpečnostným incidentom.

# ZÁVER

Hlavným cieľom práce bolo vytvoriť jednoduchý, komplexný a kompaktný nástroj pre audit počítačovej siete a prvkov na nej umiestnených. Pre splnenie hlavného cieľu práce bolo nutné si rozdeliť prácu na viacero dielčích cieľov.

Prvým dielčím cieľom bolo spracovanie teoretickej základne, vďaka ktorej budú práci rozumieť aj osoby, ktoré nie sú v oblasti bezpečnosti a počítačových sieti dostatočne zorientované.

Ďalším, druhým dielčím cieľom diplomovej práce bolo spracovanie analýz, na ktorých podklade bude možné čo najefektívnejšie navrhnúť a vyvinúť požadovaný skenovací nástroj. Prvou analýzou bola analýza požiadaviek spoločnosti, na základe ktorých bolo možné navrhnúť fundamentálne parametre pre funkcionalitu systému. Na danú analýzu nadväzovala analýza technických prostriedkov spoločnosti, v ktorej bolo možné na báze firemnej infraštruktúry, vybrať čo najvhodnejšie prostriedky na vývoj systému. Poslednou časťou analýzy bola analýza rizík, v rámci ktorej došlo k identifikácii a zhodnoteniu aktív spoločnosti a rizík, ktoré na dané aktíva vplývajú. Vďaka tomu bolo možné identifikovať aktíva, ktoré budú skenované, ale taktiež aj potvrdiť to, že jedným z najzávažnejších zdrojov rizík je absencia monitorovacieho systému.

Posledným dielčím cieľom práce sú vlastné návrhy na riešenie. V prvej časti návrhu sa venujem tvorbe prostredia pre implementáciu skenovacieho skriptu. Vďaka spracovanej analýze bolo možné zvoliť a spracovať také prostredie, ktoré v plnej miere využíva infraštruktúru spoločnosti. Následkom toho je, že vývoj systému nevytváral žiadne dodatočné nároky na hardware alebo software spoločnosti. Celkovým výstupom z danej časti práce je komplexný, ľahko prenášateľný, dostupný a jednoducho použiteľný systém. Po spracovaní systému ostával už len návrh skenovacieho skriptu. Z dôvodu, že mal byť systém čo najjednoduchší, zvolil som taký nmap skript, ktorý priamo vypíše odhalené zraniteľnosti aj s hodnotou CVE. Vďaka tomu je výstup zo skenovania schopný spracovať takmer ktokoľvek. Samotný návrh na opatrenia bude pre každú sieť jedinečný. Všeobecne sa však dá povedať, že základnými opatreniami pre akúkoľvek sieť je správne nastavenie politiky hesiel, pravidelné aktualizácie a princíp reštrikcií, kedy je zakázané všetko, čo priamo nesúvisí s výkonom povolania.

# ZOZNAM POUŽITÝCH ZDROJOV


1. ONDRÁK Viktor, Petr SEDLÁK a Vladimír MAZÁLEK. *Problematika ISMS v manažerské informatice.* Brno: CERM, Akademické nakladatelství, 2013. ISBN 978-80-7204-872-4.

2. C SYSTEM CZ: *Úvodní C SYSTEM CZ* [online]. [cit. 2020-10-10] Dostupné z: https://www.csystem.cz/

3. KUROSE, James F. a Keith W. ROSS. *Počítačové sítě.* Brno: Computer Press, 2014. ISBN 978-80-251-3825-0.

4. JORDÁN, Vilém a Viktor ONDRÁK. *Infrastruktura komunikačních systémů I: univerzální kabelážní systémy.* Brno: Akademické nakladatelství CERM, 2013. ISBN 978-80-214-4839-1.

5. SMITH, J. E. (James E.) a Ravi NAIR. *Virtual machines: versatile platforms for systems and processes.* Amsterdam: Elsevier, 2005. ISBN 1-55860-910-5.

6. What is a Container?: *A standardized unit of software. Docker* [online]. [cit. 2021-01-18]. Dostupné z: https://www.docker.com/resources/what-container

7. Nmap Security Scanner: *Introduction. Nmap* [online]. [cit. 2021-01-19]. Dostupné z: https://nmap.org/

8. What is Kali Linux?: *About Kali Linux. Kali* [online]. [cit. 2021-01-20]. Dostupné z: https://www.kali.org/docs/introduction/what-is-kali-linux/

9. SMEJKAL, Vladimír a Karel RAIS. *Řízení rizik ve firmách a jiných organizacích.* 3., rozš. a aktualiz. vyd. Praha: Grada, 2010. ISBN 978-80-247-3051-6.

10. MILLER, Philip M. *TCP/IP: the ultimate protocol guide vol. 1 ; data delivery and routing.* Boca Raton: BrownWalker Press, 2009. ISBN 978-1-59942-491-0.

11. MILLER, Philip M. *TCP/IP: the ultimate protocol guide vol. 2 ; applications, acces and data security.* Boca Raton: BrownWalker Press, 2009. ISBN 978-1-59942-493-4.

12. CVE-2019-6111. *Vulners* [online]. 31. 01. 2019 [cit. 2021-4-2]. Dostupné z: https://vulners.com/cve/CVE-2019-6111

# ZOZNAM POUŽITÝCH SKRATIEK

| | |
|---|---|
| VM | Virtual Machine |
| SW | Software |
| HW | Hardware |
| OS | Operačný Systém |
| IP | Internet Protocol |
| CVE | Common Vulnerabilities and Exposures |

# ZOZNAM POUŽITÝCH OBRÁZKOV

# ZOZNAM POUŽITÝCH TABULIEK