Software pro evidenci zranitelností v počítačové síti

Martin Laštovička, Jakub Bartolomej Košuth, Daniel Filakovský, Antonín Dufka, Martin Husák

2020

Abstrakt

Tento dokument popisuje stejnojmenný výstup projektu "*Výzkum nástrojů pro hodnocení kybernetické situace a podporu rozhodování CSIRT týmů při ochraně kritické infrastruktury*" (VI20172020070) řešeného v programu *Bezpečnostní výzkum České republiky* v letech 2017-2020 na Masarykově univerzitě. Dokument obsahuje popis výsledku, návod k instalaci, uživatelskou dokumentaci a programátorskou dokumentaci.

Obsah

Úvod

Odhalování zranitelností v infrastruktuře organizace je důležitou součástí jejího zabezpečení před kybernetickými útoky. V tomto dokumentu je popsán výsledek č. 1 **Software pro evidenci zranitelností v počítačové síti** výzkumného projektu *Výzkum nástrojů pro hodnocení kybernetické situace a podporu rozhodování CSIRT týmů při ochraně kritické infrastruktury*. Tento výstup v podobě software slouží pro vyhledávání informací o zranitelnostech a evidenci zranitelných prvků komunikační infrastruktury. Software přistupuje k odhalování zranitelností komplexně na několika vrstvách: aktivním a pasivním monitorováním sítě, zpracováním veřejně dostupných informací o zranitelnostech, zpracováním dat o zranitelných strojích od neziskových organizací a prostřednictvím platforem pro sdílení událostí v bezpečnostní komunitě.

Pasivní monitorování sítě založené na technologii síťových toků (NetFlow, IPFIX) umožňuje analyzovat provoz sítě procházející přes měřící body a bez zásahů do sítě nebo jejích prvků určit její komponenty. Důležitou součástí této analýzy je metoda vytváření otisků (z angl. fingerprinting) zařízení, která poskytne soupis aktivních prvků v síti včetně jejich programového vybavení. Tímto způsobem je software schopný určovat operační systém zařízení, nainstalovaný antivirus a poskytované síťové služby. V případě webových služeb pak dokáže určit CMS (Content Management System) webového serveru a webový prohlížeč klienta. Všechny tyto informace jsou komplementárně doplněny údaji z periodického aktivního skenování sítě. Aktivní skeny mají obecně výrazně vyšší přesnost a úroveň detailu identifikace, ovšem jejich pokrytí (procento odhalených strojů a služeb) je silně limitováno aktuálním stavem monitorovaných zařízení a nastavením firewallů znemožňující jejich plné využití v některých typech sítí nebo určitých segmentech těchto sítí. Kombinací aktivního a pasivního přístupu tak software dosahuje uceleného pohledu na aktuální stav sítě a umožňuje tak vyhledávání zranitelných prvků.

Informace o zranitelnostech získává software z veřejně dostupných zdrojů. Tyto zdroje je možné rozdělit do dvou skupin. První skupinou jsou všeobecné databáze zranitelností obsahují informace o zranitelnostech bez ohledu na výrobce produktu nebo osobu a společnost, která zranitelnost objevila. Příklady takových databází jsou *National Vulnerability Database* (NVD)[1], *Open Sourced Vulnerability Database* (OSVDB)[2] nebo *Exploit Database*[3]. Druhou skupinou jsou databáze výrobců informačních systémů a aplikací, kteří poskytují informace o zranitelnostech vlasních produktů a v produktech třetích stran, jejichž funkcionalitu jejich produkty využívají. Obě skupiny se liší jak rozsahem poskytovaných informací, tak i úrovní detailu a aktuálností. Software všechny informace o zranitelnostech stahuje a ukládá v lokální databázi, kde je možné zranitelnosti spojit s identifikovanými prvky sítě na základě shody zranitelného software se software na daném zařízení.

Posledním významným zdrojem informací o zranitelnostech v síti jsou spolupracující organizace. Hlavním zdrojem takových informací je nezisková organizace Shadowserver,

[1] https://nvd.nist.gov/
[2] http://www.osvdb.org/
[3] https://www.exploit-db.com/

která provádí periodické skeny zranitelností v celém adresním prostoru internetu. Výstupy skenů pak zdarma nabízí dotčeným organizacím, aby mohly zranitelná zařízení zabezpečit. Druhým významným zdrojem informací je platforma pro sdílení událostí vyvinutá v projektu *VI20162019029 Sdílení a analýza bezpečnostních informací*. Pomocí této platformy zapojené organizace sdílí informace o aktuálních kyberbezpečnostních událostech a odhalených zranitelnostech. Informace z obou zdrojů software ukládá v centrální databázi pro evidenci a vyhledávání zranitelných zařízení.

Výstupem software je soupis aktivních prvků v síti s identifikací prvků zranitelných a potenciálně zranitelných. Tyto výstupy budou obecně využitelné jako samostatný systém pro evidenci zranitelností v počítačové síti a je možné k nim přistupovat programově skrze centrální databázi nebo pomocí grafického rozhraní této databáze s možností vyhledávání prvků a procházení stavu sítě. V rámci projektu výstupy vytvořeného software č. 1 slouží jako vstup pro software pro vizualizaci bezpečností situace v síti (Výsledek č. 2).

Technické parametry výsledku

Software pro evidenci zranitelností v počítačové síti propojuje různé datové zdroje poskytující informace o zranitelnostech a s jejich využitím zjišťuje, které prvky KII jsou danou zranitelností ohroženy. Správci KII nebudou muset aktivně sledovat hlášení o zranitelnostech a ručně vyhledávat zranitelné prvky infrastruktury, ale dostanou k dispozici automaticky generovaný soupis aktivních prvků v síti s vyznačením prvků zranitelných, případně potenciálně zranitelných. Výhodou tohoto přístupu je možnost detekovat zranitelnosti systémů, které nejsou přímo ve správě bezpečnostního týmu, avšak tým má možnost monitorovat síťový provoz podřízené či vlastní organizace.

Software je distribuován jako open-source pod licencí MIT, vlastníkem výsledku je Masarykova univerzita, IČO 00216224:

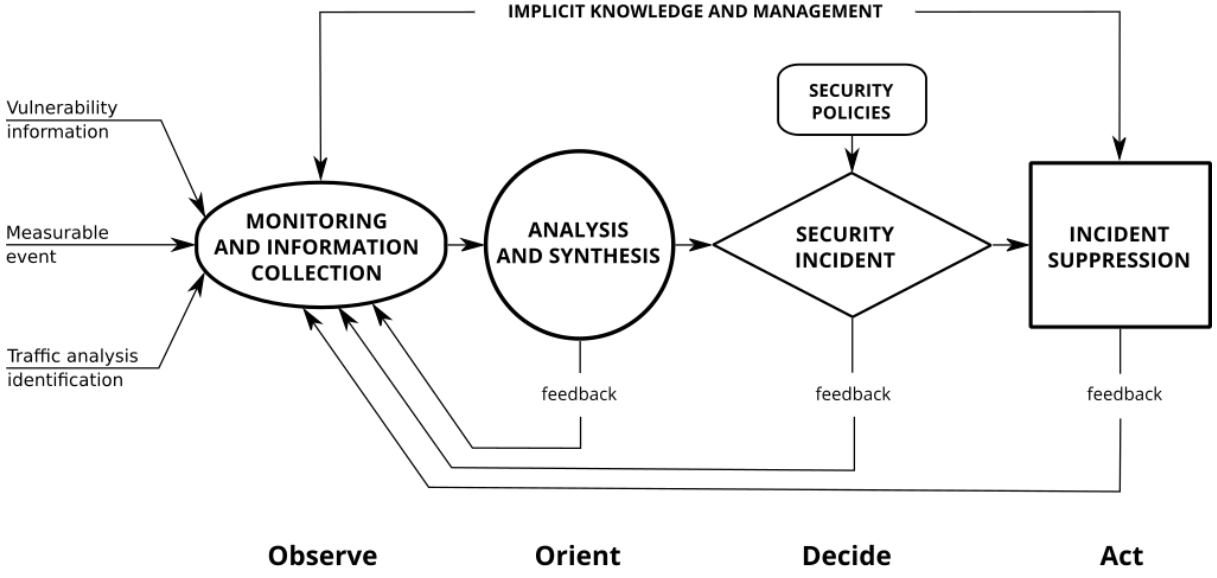
Kontaktní osoba: RNDr. Martin Laštovička
Ústav výpočetní techniky
Masarykova Univerzita
Šumavská 416/15, Brno 602 00
e-mail: lastovicka@ics.muni.cz
tel: +420 549 49 6477

Ekonomické parametry výsledku

Tržní segment představují organizace, které provozují či budují svůj CERT/CSIRT, Security Operation Centre, případně zprostředkovávají kybernetickou bezpečnost jako službu. Výsledek umožňuje uživatelům a provozovatelům sítí a služeb získat rychlý přehled o aktuálním stavu chráněné sítě, zejména pak o výskytu zranitelností v síti. Software umožňuje uživatelům rychlejší orientaci při řešení bezpečnostního incidentu a poskytuje potřebná data pro rozhodování o prioritizaci incidentů a jejich řešení. Výsledek tak přináší úsporu v čase řešení kyberbezpečnostních problémů, což snižuje nároky na lidské zdroje, případně umožňuje uživatelům zvládat více úkolů za stejný čas. Výsledek tak zkracuje dobu reakce na kyberbezpečnostní incident, čímž přispívá k obecnému zvýšení úrovně kyberbezpečnosti v organizaci. Využití výsledku je licencováno bez poplatku.

Popis výsledku

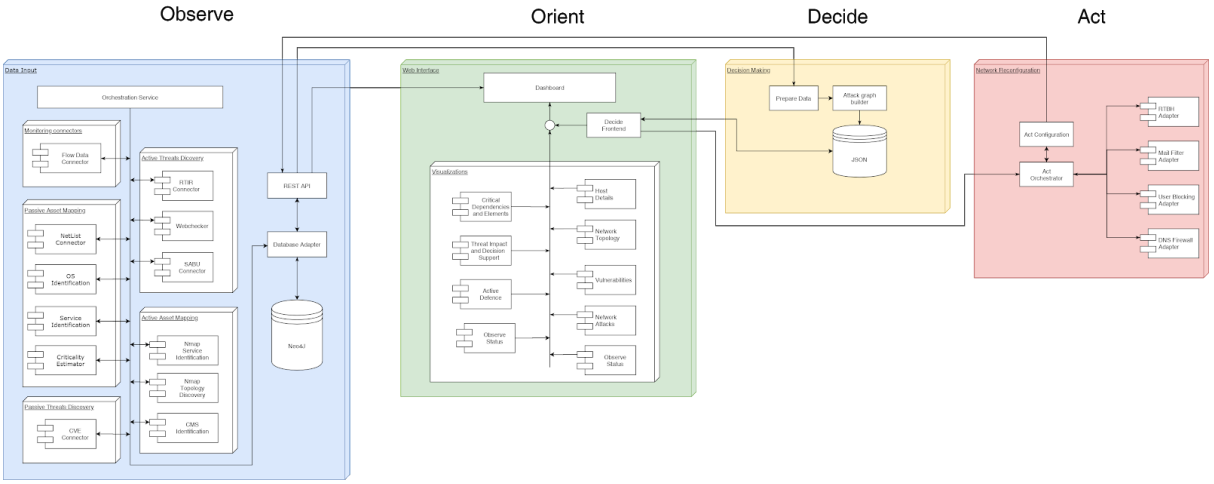
Architektura systému pro podporu rozhodování při ochraně kritických informačních infrastruktur vychází ze konceptu OODA cyklu. Původně byl navržen pro podporu rozhodování vojenských pilotů, cyklus se však dočkal uplatnění i v mnoha dalších oblastech. Variaci OODA cyklu pro podporu rozhodování při ochraně počítačových sítí vidíme na obrázku č. 1[4]. V jednotlivých fázích OODA cyklu můžeme vidět fázi sběru informací (Observe), vizualizaci (Orient), výběr obranné akce v závislosti na okolnostech (Decide) a provedení obranné akce (Act). Podstatné na OODA cyklu je to, že každá fáze poskytuje zpětnou vazbu fázi Observe, aby byly informace vždy aktuální.



Obrázek 1: OODA cyklus.

V souladu s principy rozhodování pomocí OODA a rozdělení na jednotlivé fáze je i systém rozvržen do čtyř fází, Observe, Orient, Decide a Act. Subsystémy každé fáze pak obstarávají potřebnou aplikační logiku implementující popsané fáze rozhodování. V kontextu softwarových výstupů projektu pak každý software odpovídá jedné části OODA cyklu. Na obrázku č. 2 můžeme vidět celkovou architekturu systému rozdělenou barevnými podklady do čtyř fází (obrázek je v plném rozlišení dostupný jako příloha tohoto dokumentu). Modře podbarvené jsou systémy fáze Observe, které obstarávají sběr dat a jejich ukládání do databáze. Zeleně podbarvený je systém implementující fázi Orient, zejména pro vizualizaci získaných dat. Žlutě podbarvený je systém pro podporu rozhodování ve fázi Decide, červeně pak systém ovládání aktivní obrany spadající pod fázi Act.

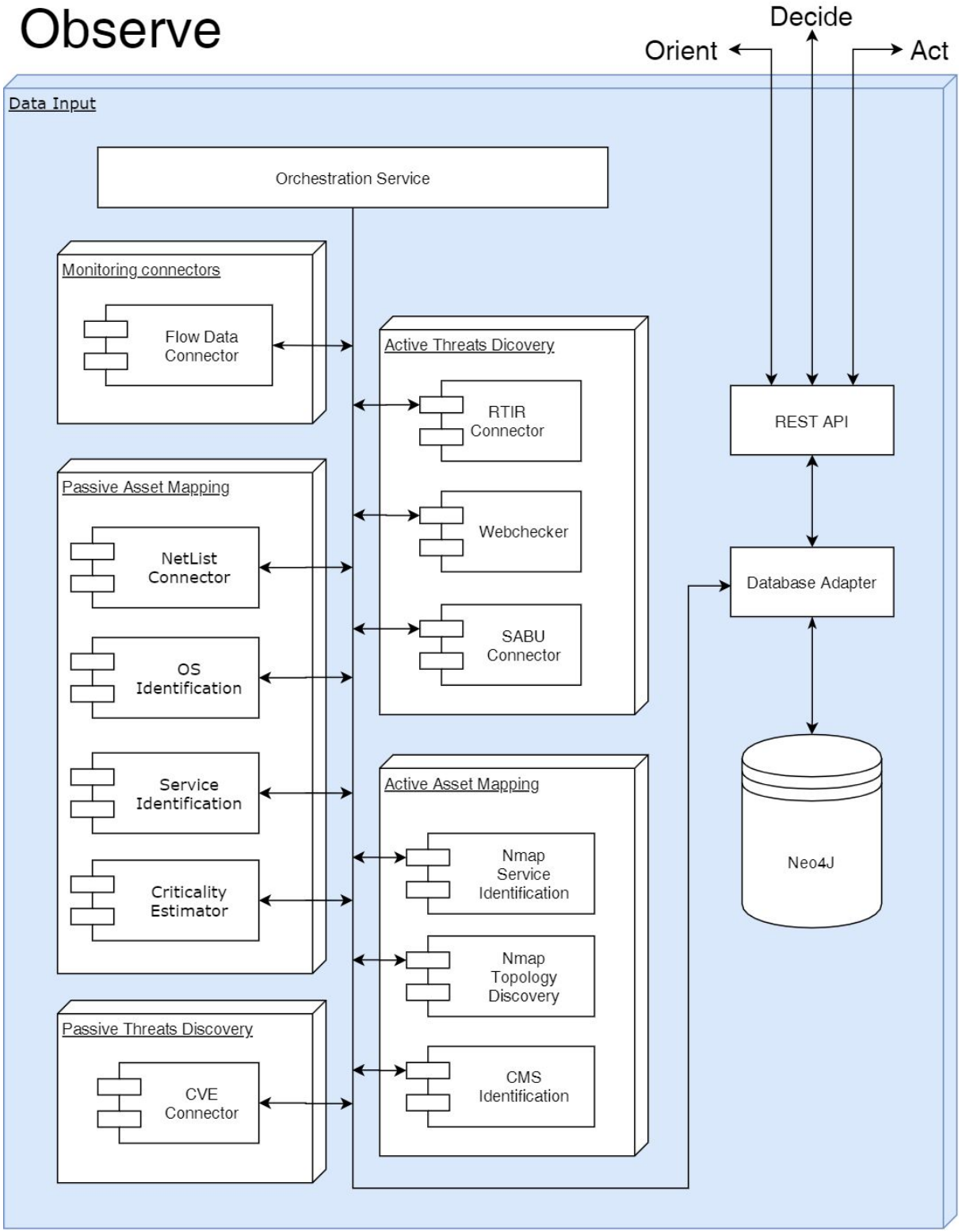
[4] KOTT, Alexander; WANG, Cliff; ERBACHER, Robert F. (ed.). *Cyber defense and situational awareness*. Springer, 2015.



Obrázek 2: Architektura systému.

Výsledek č. 1 **Software pro evidenci zranitelností v počítačové síti** popisovaný v tomto dokumentu odpovídá fázi Observe OODA cyklu. Software je navržen modulárně, aby byly jednotlivé komponenty co nejvíce nezávislé, což zvyšuje robustnost systému při výpadku části služeb a umožňuje upravovat funkcionalitu softwaru podle požadavků organizace, ve které je nasazen. Systém umožňuje jak odpojení komponent, které organizace nevyužije, tak snadné rozšíření o nové organizačně specifické komponenty. Detailní pohled na architekturu software poskytuje obrázek č. 3.

Komponenty software se dělí do dvou základních kategorií - podpůrné technologie a datové konektory. Podpůrné technologie jsou základem software sloužící pro synchronizaci a sjednocení komunikace konektorů a pro perzistentní uložení jejich výsledků do databáze. Pro zapojení software do kontextu ostatních výstupů projektu, případně jiných externích systémů, pak poskytuje REST API nad centrální databází. Datové konektory jsou samostatné jednotky.



Obrázek 3: Detail architektury Výsledku č. 1

Podpůrné technologie

Podpůrné technologie jsou základním stavebním prvkem software. Mezi podpůrné technologie software patří následující komponenty. Orchestrační služba zajišťuje správu výpočetního clusteru a pravidelné spouštění všech datových konektorů. Centrální databáze zajišťuje perzistentní uložení výstupů konektorů. Nad databází je vystaveno REST API zajišťující externí komunikaci. Databázový adaptér pak zajišťuje programový přístup k databázi a snižuje závislost na konkrétní technologické implementaci samotné databáze. Všechny komponenty podpůrných technologií jsou detailně popsány v této kapitole.

Orchestrační služba

Orchestrační služba je odpovědná za korektní spouštění jednotlivých komponent v rámci software pro evidenci zranitelností, které jsou označovány jako task (úlohy). Každá úloha má v systému přesně definovanou roli, tzn. kdy a s jakými parametry se má spouštět. Záznamy o běhu každé úlohy jsou zaznamenány do monitorovacího systému a je tak možné sledovat, zda úloha ukončila činnost bez chyb, délku běhu úlohy, její výstup a případně výpis chyb. Každá úloha navíc disponuje vlastním loggerem zapisujícím průběh spuštění komponenty do souboru */var/log/crusoe/<nazev-komponenty>.log* pro snazší dohledání případných chyb. Orchestrační služba také zajišťuje uložení výsledků jednotlivých úloh do databáze. Samotná orchestrační služba neobsahuje žádnou logiku vykonávanou v rámci jednotlivých komponent, používá jejich veřejné rozhraní a zprostředkovává jejich vzájemnou komunikaci, proto je závislá na všech komponentách CRUSOE.

Kromě základního řízení běhu komponent software může orchestrační služba sloužit také pro správu výpočetního clusteru při nasazení v distribuovaném prostředí. Toho je docíleno pomocí běžné architektury master-worker, tedy jeden jeden hlavní stroj koordinující činnost a jeden nebo více výpočetních uzlů, které provádí zadané úlohy. Při nasazení pouze na jeden stroj plní tento stroj obě role. Pro některé komponenty software je ovšem distribuované nasazení výhodné a umožňuje jim získat vyšší viditelnost v síti, např. komponenta pro aktivní skenování sítě tak může využívat více monitorovacích bodů v síti a její výsledky jsou tak méně zkresleny firewally mezi jednotlivými segmenty sítě.

Seznam úloh

Seznam úloh (tasks) je určen konfiguračním souborem *celery_config.py* a je možné jej modifikovat a přizpůsobovat dle vlastních požadavků, případně je možné přidat nové úlohy. V základní konfiguraci obsahuje seznam tasků všechny komponenty software včetně jejich konfigurace. Tyto komponenty jsou detailně popsány v následující kapitolách této zprávy.

Použité technologie

Orchestrační služba je do značné míry závislá na dalších technologiích, které se v rámci ní provádějí. Konkrétně se jedná o:

- Celery[5] – samostatný plánovač, který provádí spouštění jednotlivých tasků podle určitých pravidel, a to pomocí určené periody nebo přesně definovaného času.

[5] http://www.celeryproject.org/

- Flower[6] - monitorovací systém určený přímo pro Celery. Uchovává historii všech tasků od posledního restartu Orchestrační služby. Obsahuje informace jako název úkolu, čas spuštění, doba trvání, výsledek, který komponenta vrátila nebo chybová hlášení s celou cestou, kde chyba vznikla.
- Redis[7] - plní funkci databáze a uchovává informace o aktuálním stavu a výstupech zpracovávaných úloh.

REST API

Komponenta REST API slouží pro usnadnění komunikace s ostatními softwary vyvinutými v rámci projektu a umožňuje propojení software s libovolným systémem podporující komunikaci přes REST. Základní koncové body API (endpoints) jsou navrženy, aby poskytovaly sadu CRUD (Create, Read, Update, Delete), tedy vytvoření, čtení, aktualizace a mazání objektů v centrální databázi. Tyto objekty odpovídají datovému modelu projektu, který byl detailně popsán jako publikační výstup projektu[8]. Implementace REST API je založená na open source webovém aplikačním frameworku Django, který zajišťuje vystavení koncových bodů API a zpracování požadavků

Databázový adaptér

Tato komponenta je realizací návrhového vzoru adaptér, který usnadňuje propojení různých komponent systému. Díky databázovému adaptéru nejsou datové konektory software závislé na konkrétní technologii databáze, díky čemuž je možné jednoduše nahradit databázový systém dle potřeb organizace provozující tento software. Samotný adaptér pak obsahuje programová volání pro jednotlivé konektory, která převádí na dotazy v jazyce daného databázového systému.

Centrální databáze

Databáze slouží jako centrální úložiště dat pro celý systém podpory rozhodování, z architektonického hlediska však spadá pod fázi Observe, která obstarává sběr dat. Systémy implementující ostatní fáze OODA cyklu přistupují k databázi jako ke zdroji dat, nástroje fáze Observe proto zodpovídají za aktuálnost a konzistenci údajů v databázi.

Pro implementaci systému byla vybrána databáze Neo4j, která spadá do kategorie takzvaných grafových databází. Narozdíl od klasických relačních databází, grafové databáze umožňují jednodušeji modelovat entity a vztahy v počítačových sítích s ohledem na rozšiřitelnost. Díky grafové databázi je možné snadno přidávat nové typy entit a vztahů mezi nimi, což zjednodušuje jak experimentální vývoj, tak i nasazení v praxi, kdy je možné průběžně reagovat na možnost přísunu nových typů vstupních dat. Za běhu je tak možné přidávat do databáze nové typy uzlů (entit) a hran (relací) bez nutnosti zásahů do stávajícího datového modelu či uložených dat, jak je tomu běžně u relačních databází.

---

[6] https://flower.readthedocs.io/en/latest/
[7] https://redis.io/
[8] KOMÁRKOVÁ, Jana, Martin HUSÁK, Martin LAŠTOVIČKA a Daniel TOVARŇÁK. CRUSOE: Data Model for Cyber Situation Awareness. In Proceedings of the 13th International Conference on Availability, Reliability and Security. Hamburg: ACM, 2018. s. "36:1"-"36:10", 10 s. ISBN 978-1-4503-6448-5. doi:10.1145/3230833.3232798.

Datové konektory

Datové konektory jsou realizací dílčích částí funkcionality software. Každý konektor je zodpovědný za získání vstupních dat, jejich zpracování a následné uložení výsledků své činnosti. Díky návrhu architektury a podpůrným technologiím jsou konektory naprosto nezávislé na implementaci a činnosti ostatních konektorů. Je tak čistě na provozovateli software, které konektory se rozhodne využívat, případně zda doplní svůj vlastní konektor.

Konektor kolektoru Flowmon

Komponenta umožňuje přístup na kolektor od společnosti Flowmon[9] a následné získávání síťových toků ve formě IPFIX pomocí SSH nebo Flowmon REST API. Získané síťové toky jsou později předány dalším komponentám, jako např. OS-parser-component nebo Service component. Funkcionalitu přístupu k datům pomocí SSH je možné využít i pro libovolný jiný IPFIX kolektor, podmínkou je pouze nainstalovaná sada open-source nástrojů *nfdump*[10] na daném kolektoru.

Komponenta aktivního monitorování sítě

Komponenta aktivního monitorování sítě je zodpovědná za aktivní skenování sítě, čímž zjišťuje aktuální topologii sítě, aktivní prvky sítě a vystavené síťové služby. Pro samotné skenování sítě je využíván open-source nástroj nmap[11].

Nmap scanner je součást, která slouží pro mapování běžících síťových služeb a identifikace jejich software. Mapování služeb probíhá za pomoci skenu 100 nejčastěji používaných portů a v případě nalezení otevřeného portu je tato informace dále zpracována. Kromě tohoto základního skenu komponenta využívá pokročilých schopností Nmap identifikovat software. Na otevřené porty strojů jsou zaslány další požadavky specifické pro různé typy software a na základě odpovědí jsou zaslány více specifické požadavky dokud nedojde k identifikaci konkrétního software. Výsledná identifikace je převedena do formátu CPE (Common Platform Enumeration), který je využíván pro jednoznačnou identifikaci a je využíván rovněž v komponentách pro odhalování zranitelností.

Nmap topology scanner je součást provádějící mapování topologie sítě pomocí aktivního skenování sítě metodou traceroute. Konektor tedy získává informace o aktuálních aktivních spojeních mezi jednotlivými zařízeními v síti na úrovni L3 (síťová vrstva) ISO/OSI modelu. Takto detekovaná spojení lze využít například k identifikaci stroje jako aktivního prvku v síti nebo jako koncové zařízení. Konektor provádí TCP traceroute na portech 80 a 443, které jsou obvykle povolené na firewallech a sken tak pokryje větší množství sítě, než kdyby využíval tradiční traceroute pomocí ICMP protokolu. ICMP traceroute je využíván jako záložní varianta, pokud je cílový stroj nedosažitelný přes zmíněné TCP porty.

Traceroute skenování sítě odhalí aktuální routovací cestu mezi skenujícím zařízením a všemi koncovými zařízeními v síti. To je ovšem nedostatečné pro zmapování síťové

[9] https://www.flowmon.com/cs
[10] https://github.com/phaag/nfdump
[11] https://nmap.org/

topologie, jelikož takovýto postup zákonitě vyprodukuje pouze stromovou strukturu neodpovídající reálným sítím obsahujícím redundantní cesty a tedy i cykly ve výsledném grafu. Proto je potřeba takový sken spouštět opakovaně z vícera monitorovacích bodů umístěných v různých (logických i fyzických) lokalitách v síti. Tím je dosažena detekce většího množství linek v síti a výsledná mapa tak blíže odpovídá reálné topologii.

Komponenta se skládá ze tří základních částí – samotného procesu skenování, parsování a modulu zodpovědného za údržbu konzistence dat v databázi, jako je například uzavírání neaktualizovaných spojení.

Modul Scanner

Modul Scanner byl navržen tak, aby byl snadno použitelný ve velkých sítích, jejichž kompletní sken zabere netriviální množství času. Proto je základem modulu možnost konfigurace paralelního skenování z více strojů a dělení celého skenovaného prostoru na menší celky, které je možné zpracovat samostatně a následně spojit výsledky do celkového pohledu.

Modul Parser

Úkolem modulu Parser je načíst výsledky provedeného skenu a dle specifikace datového modelu je uložit do grafové databáze. Jelikož jsou skeny prováděny opakovaně z více různých míst, musí tento modul zajistit jak přidávání nově detekovaných spojení a služeb, tak aktualizaci spojení detekovaných dříve.

Modul Cleaner

Modul Cleaner řeší problematiku skenování v diskrétních časových intervalech. Fakt, že při skenu nebylo detekováno dříve odhalené síťové spojení nebo služba má totiž dvě základní vysvětlení. Prvním je neaktivita cílového zařízení, na které probíhá sken, a tím pádem chybějící detekce posledního hopu traceroute, resp. služby. Druhým vysvětlením je pak skutečný zánik daného spojení nebo ukončení poskytování síťové služby.

Komponenta Pasivního Monitorování Sítě

Cílem této komponenty je provést analýzu záznamů síťového provozu na bázi technologie IPFIX, za účelem detekce a identifikace služeb spuštěných na zařízeních v síti. Výstupem komponenty je zobrazení přiřazující IP adresám identifikátory detekovaných služeb. Komponenta obsahuje subkomponenty, z nichž každá slouží k detekci jiného typu služeb a může v rámci své funkcionality kombinovat více metod identifikace. Subkomponenty jsou spouštěny samostatně a jejich výstupy jsou sloučeny do výsledného výstupu komponenty. Funkcionalita komponenty je nezávislá na konkrétním prostředí. Interakce s okolím probíhají skrze dvojici souborů - vstupního a výstupního. Cesty k těmto souborům jsou předány komponentě a ta již samostatně provede svou činnost. Vstupní soubor by měl obsahovat záznamy IPFIX, na nichž se má provádět analýza, a do výstupního souboru bude po dokončení zpracování zapsán výstup komponenty ve formátu JSON. Komponenta obsahuje tři moduly: run, core, rules. Jejich návrh a funkcionalita jsou rozepsány níže.

Subkomponenta pro identifikaci antivirového software provádí identifikace na základě komunikace se specifickými doménami. Z jednotlivých IPFIX záznamů jsou pozorovány hodnoty (doménová jména), které jsou porovnávány s předem definovaným seznamem pravidel. Při nalezení shody hodnoty s nějakým pravidlem je odpovídající typ antivirového software zařazen v pomocné datové struktuře k IP adrese, která vyprodukovala daný síťový tok. Po zpracování všech vstupních IPFIX záznamů je pomocná struktura transformována do výstupu subkomponenty tak, že pro každou IP adresu je zvolen ten nejpravděpodobnější antivirus (na základě četnosti).

Subkomponenta pro identifikaci webového prohlížeče obsahuje dvě identifikační metody. První metodou je komunikace se specifickými doménami, kdy jsou z jednotlivých IPFIX záznamů pozorovány hodnoty (doménová jména) a porovnávány s předem definovaným seznamem pravidel. Druhou metodou je identifikace z položky *HTTP User-agent*, která obsahuje textový řetězec identifikující webový prohlížeč.

Poslední součástí je *subkomponenta pro identifikaci služeb v síti* využívající metod strojového učení. Ve fázi učení je je nejprve natrénována nad síťovým provozem obsahujícím NBAR2 značky. Pomocí toho je vytvořen model umožňující klasifikovat síťovou službu z parametrů komunikace jako jsou například charakteristiky objemu, velikosti paketů, časové rozložení nebo nastavení TCP/IP hlaviček. Metoda je tak schopná z běžné komunikace určit, jaká služba tuto komunikaci vyvolala a identifikovat poskytovatele této služby.

Modul run

Tento modul obsahuje vstupní bod komponenty - metodu run. Jejím zavoláním s dvojicí (vstupní soubor, výstupní soubor) dochází k následující sekvenci operací:

1. načtení vstupních dat ze vstupního souboru,
2. inicializace jednotlivých subkomponent,
3. spuštění subkomponent nad vstupními daty,
4. výpis výsledků do výstupního souboru.

Modul core

Modul core je jádrem komponenty, které využívají jednotlivé subkomponenty pro reprezentaci svých výsledků a umožňuje jim snadnou interoperabilitu. Hlavní součástí tohoto modulu je třída Result, která společně s hierarchií tříd odpovídajícím jednotlivým použitým podčástem CPE řetězce (vendor:product:version) poskytuje rozhraní pro jednoduchou manipulaci s nálezy. Struktura tohoto modulu je znázorněna na třídním diagramu pod odstavcem.

Modul rules

Pro subkomponenty, respektive jejich metody detekce, které nevyužívají algoritmicky sofistikovanějších technik, lze použít modul rules. Modul rules implementuje třídu Rules, která dokáže zpracovávat síťové toky na základě předem definovaných pravidel. Tato pravidla lze definovat pomocí konfiguračního souboru ve formátu JSON.

Konfigurační soubor pravidel má strukturu slovníku mezi názvy pravidel a samotnou definicí pravidel. Definice pravidla je opět slovník, který obsahuje “informační klíče” a “pravidlové klíče”. Informační klíče slouží ke specifikaci informací, pokud dojde k nalezení shody. Pravidlové klíče jsou nepovinné a definují, jaké podmínky musí splnit daný tok, aby nastala shoda s daným pravidlem. Položky, které slouží k nalezení shody pomocí řetězce jsou reprezentovány regulárním výrazem, případně i jejich seznamem. Položky, které hledají shodu pomocí číselné hodnoty, mohou být reprezentovány ve formátu “první..poslední”, případně i se specifikováním velikosti přírůstku “první,druhý..poslední”.

Komponenta Fingerprintingu Operačních Systémů

Problematika fingerprintingu operačních systémů je aktuálně natolik výzkumně zajímavá, že její implementace byla oddělena od obecné komponenty pro pasivní monitoring počítačové sítě. Komponenta pro identifikaci operačních systémů si klade za cíl detekovat a identifikovat operační systém zařízení nacházejících se v síti na základě dat obsažených v síťových tocích technologie IPFIX. Funkcionalita je založená na třech metodách popsaných ve článku *Passive OS Fingerprinting Methods in the Jungle of Wireless Networks*[12]. Konkrétně se jedná o specifické domény, User-Agent webového prohlížeče, a výchozí hodnoty TCP/IP zasílaných paketů. Výstupy jednotlivých metod jsou zkombinovány a nález s nejvyšší četností pro danou IP adresu je zvolen jako výsledný.

Metoda využívající TCP/IP parametry byla navíc rozšířena o použití algoritmů strojového učení. Byl použit model rozhodovacích stromů, který se dle článku *Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use?*[13] jeví jako vhodný pro řešení tohoto problému.

Modul Specifické domény

Na základě pozorování automatického chování operačních systémů (kontrola připojení k internetu, kontrola aktualizací, odesílání telemetrických dat) bylo vytvořeno mapování mezi některými specifickými doménovými jmény a operačními systémy, které tyto domény běžně kontaktují během provádění svých rutin. Jedná se například o adresy aktualizačních serverů, služeb pro kontrolu konektivity, a podobné.

Modul HTTP User-Agent

User-Agent je volitelná hlavička HTTP protokolu, do které webové prohlížeče vkládají informace o klientském software. Tato hlavička často obsahuje i název operačního systému, případně i jeho verzi, který pak lze parsovat a využít k identifikaci běžícího OS.

---

[12] Martin Laštovička, Tomáš Jirsík, Pavel Čeleda, Stanislav Špaček a Daniel Filakovský. Passive OS Fingerprinting Methods in the Jungle of Wireless Networks. In NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. Taipei, Taiwan. 2018. ISBN 978-1-5386-3416-5.

[13] Martin Laštovička, Antonín Dufka a Jana Komárková. Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use? In IEEE. Proceedings of the 14th International Wireless Communications and Mobile Computing Conference. Limassol, Cyprus. 2018. ISBN 978-1-5386-2069-4.

Modul TCP/IP parametry

Identifikace na základě TCP/IP parametrů je založená na závislosti některých výchozích hodnot TCP/IP paketů na implementaci konkrétního operačního systému, který je vyprodukoval. V komponentě byly využity následující tři parametry:

- IP TTL – hodnota Time to Live IP pole zaokrouhlena na nejbližší vyšší mocninu dvojky,
- TCP SYN Size – velikost paketu iniciujícího TCP spojení v bajtech,
- TCP Win Size – hodnota TCP Window pole v TCP hlavičce.

Komponenta detekce CMS

Účelem komponenty detekce CMS je získávání informací o webových serverech a systémech pro správu obsahu (z anglického CMS - Content Management System) webových aplikací. Komponenta se primárně zaměřuje právě na identifikaci CMS systémů, které jsou uživatelsky přívětivé a jednoduché na správu, a tudíž velmi rozšířené. Kvůli jejich rozmanité funkcionalitě a množství pluginů jsou ovšem náchylné k výskytu zranitelností. Pomocí údajů z komponenty mohou správci získat lepší obraz o webových aplikacích a jejich zranitelnostech. Díky tomu je možné snadněji odhalit slabá místa v zabezpečení infrastruktury.

Komponenta Webchecker

Komponenta slouží pro identifikaci aktivních webových stránek organizace a pro kontrolu platnosti certifikátů, které tyto weby předkládají uživatelům. Pro odhalení aktivních webů komponenta využívá pasivního monitorování provozu, kdy v zachycených datech vyhledává uživatelské přístupy na HTTP(S) stránky. Pro každou takto objevenou stránku komponenta provede validaci dat a poskytne informaci o hostování stránky na příslušném stroji s doplňující informací o tom, zda je stránka přístupná přes HTTP nebo HTTPS. Pro všechny detekované HTTPS stránky navíc komponenta provede aktivní ověření platnostnosti certifikátu a případné detekované problémy jsou rovněž výstupem komponenty, čímž správcům poskytne nejen globální mapu stránek organizace, ale pomůže také identifikovat např. neudržované stránky s expirovaným certifikátem, které mohou značit další bezpečnostní rizika.

Komponenta získávání informací o zranitelnostech

Hlavní funkcionalita komponenty spočívá v získávání informací o zranitelných službách, službách vystavených z interní sítě do světa a strojích nakažených botnetem. Tyto informace komponenta získává z údajů poskytovaných organizací ShadowServer[14]. ShadowServer je nezisková organizace, která provádí periodické skeny zranitelností v celém adresním prostoru internetu. Výstupy skenů pak zdarma nabízí dotčeným organizacím, aby mohly zranitelná zařízení zabezpečit. Pro získání přístupu k datům je potřeba se zaregistrovat a projít procesem ověření, zda mohou být informace sdíleny.

[14] https://www.shadowserver.org/wiki/

Aktuálně ShadowServer sleduje 42 typů různých zranitelností, mezi které patří zejména detekce chybně konfigurovaných služeb, které jsou zneužitelné pro reflexi a amplifikaci DDoS útoků. Dále sleduje vybrané zranitelnosti aplikací, především webových služeb. Poslední kategorií zranitelností je detekce vzdáleného přístupu. Ten sám o sobě nepředstavuje riziko, ovšem v mnoha případech bývá vzdálený přístup na zařízení zapnutý z výroby bez vědomí administrátora (např. síťové tiskárny, kamery, aj.) nebo využívá zastaralou technologii (např. telnet protokol). Z těchto důvodů je vhodné v přehledu stavu organizace uvádět i tyto informace. Speciálním případem detekcí společnosti ShadowServer je odhalování strojů nakažených malware. ShadowServer spolupracuje s bezpečnostními složkami v Evropě a v případě odhalení řídícího centra botnetu je toto centrum zlikvidováno a na jeho místo je umístěn detektor. Nakažená zařízení se tak snaží připojit k tomuto detektoru a jejich aktivita je zachycena.

Komponenta automaticky stahuje výše popsaná data relevantní pro danou organizaci a ukládá je v centrální databázi pro další zpracování.

CVE Konektor

CVE konektor slouží k získávání informací o zranitelnostech ve formě CVE[15]. Skládá se ze dvou částí. První z nich zabezpečuje získání dat z americké národní databáze NVD[16] a druhá získává informace od jednotlivých výrobců software. Všechny získané popisy zranitelností dále klasifikuje dle v projektu vytvořené taxonomie.

Modul NVD

Modul NVD získává data z tzv. Data Feeds, které NVD zveřejňuje na své webové stránce[17] (v lidsky čitelné podobě). Samotná komponenta data stahuje z adresy https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-YYYY.json.zip, kde řetězec YYYY zastupuje konkrétní číslo roku, např. 2018. Stažené soubory jsou následně uloženy a dekomprimovány.

Modul Vendor CVE

CVE konektor získává informace o zranitelnostech také od osmi výrobců software - Adobe, Android, Apple, Cisco, Lenovo, Microsoft, Oracle a RedHat. Takto získané informace slouží k doplnění dat z NVD databáze, které z této databáze není možné získat. K již vytvořenému uzli CVE v grafové databázi je tak možné přidat následující údaje:

- popis zranitelnosti od výrobce, který je mnohdy mnohem podrobnější než v NVD,
- informace o dostupnosti záplaty resp. aktualizace software řešící danou zranitelnost,
- datum zveřejnění zranitelnosti výrobcem, pokud se liší od data zveřejnění v NVD.

Jelikož jsou některé zranitelnosti zveřejněny v NVD až po jejich zveřejnění výrobcem, získávají se zranitelnosti od výrobců 14 dní zpětně, aby bylo možné data od výrobců namapovat na zranitelnosti z NVD.

---

[15] https://cve.mitre.org/
[16] https://nvd.nist.gov/
[17] https://nvd.nist.gov/vuln/data-feeds

CVE klasifikátor

Modul na základě vlastností daného CVE (konkrétně dle CVSS, CPE a popisu) zjistí, jaký dopad může mít zneužití zranitelnosti na systém. Klasifikátor jako výsledek vrací dopad dle následující taxonomie:

- arbitrary code execution as root/administrator/system,
- gain root/system/administrator privileges on system,
- privilege escalation on system,
- gain user privileges on system,
- arbitrary code execution as user of application,
- gain privileges on application,
- system integrity/availability/confidentiality loss,
- application integrity/availability/confidentiality loss,
- communication integrity/availability/confidentiality loss.

Tyto kategorie nejsou výlučné a jedna zranitelnost může mít více dopadů na systém. Kategorie byly vytvořeny tak, aby odpovídaly kategoriím NVD, ale byla přidána větší granularita pro získání lepšího obrazu reálných dopadů. Každá kategorie odpovídá zisku určitých privilegií útočníkem, nebo schopnosti útočníka spouštět v systému příkazy nebo systém poškodit. Klasifikace zranitelností a detailní popis kategorií byl publikován v článku *Community Based Platform for Vulnerability Categorization*[18].

SABU konektor

SABU konektor slouží k příjmu informací z platformy SABU[19]. SABU je platforma pro sdílení bezpečnostních událostí provozovaná sdružením CESNET, do které jsou zapojeny zejména akademické sítě v ČR, ale i další partneři. Partneři mezi sebou sdílí nejen informace o bezpečnostních událostech, ale i informace o výskytu zranitelností, získané například skenováním sítí jedním z partnerů nebo získáváním dat z třetích stran. Technicky je SABU konektor přijímacím konektorem systému Warden[20], centrálního uzlu platformy SABU. Konektor přijímá zprávy o výskytu zranitelností v chráněné síti, následně zprávy zpracuje a data uloží do grafové databáze systému CRUSOE.

RTIR Konektor

Hlavní funkcionalitou RTIR konektoru je získávání údajů o bezpečnostních incidentech. RTIR[21] je open-source tiketovací systém široce využívaný bezpečnostní komunitou pro správu incidentů. Konektor k tomuto systému přistupuje pomocí REST API, čímž získá data o incidentech často v podobě nestrukturovaného textu. Z těchto primárních dat následně extrahuje zájmové informace jako jsou zařízení zasažená incidentem, časové značky,

---

[18] Jana Komárková, Lukáš Sadlek a Martin Laštovička. Community Based Platform for Vulnerability Categorization. In NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. Taipei, Taiwan. 2018. ISBN 978-1-5386-3416-5.

[19] https://sabu.cesnet.cz/cs/start

[20] https://warden.cesnet.cz/cs/index

[21] https://bestpractical.com/rtir

klasifikace incidentu nebo osoby a organizace zapojené do incidentu. Takto zpracovaná data následné ukládá do centrální databáze.

Komponenta vyhledávání kritických uzlů

Komponenta rovněž označovaná jako *Criticality estimator* je zodpovědná za vyhledávání nejkritičtějších uzlů v síti na základě jejich centrality. Pojmem "nejkritičtější uzel" jsou označované uzly, které jsou v grafu důležité, protože mají velký vliv na na tok informací v grafu. Takové uzly typicky slouží jako propojení různých částí grafu a v případě jejich odstranění dochází k porušení souvislosti grafu nebo k výraznému omezení toku informací. Konektor získává informace pomocí algoritmů vyhledávajících nejkratší cesty v grafu a s počtem incidentních hran jednotlivých uzlů.

Komponenta pracuje primárně nad topologií sítě organizace, která je v databázi udržována pomocí komponenty aktivního monitorování sítě. Díky informacím vypočítaných touto komponentou dokážeme identifikovat důležité uzly v síti, získat detailní informace o následcích potencionálního výpadku uzlu a připravit plán obnovy.

Algoritmus pro zjištění vlivu uzlů na tok informací v grafu

Algoritmus je postavený na implementaci *algo.betweenness*[22] dostupné v knihovně grafových algoritmů databáze Neo4j. Princip algoritmu spočívá ve výpočtu nejkratší cesty mezi každými dvěma uzly grafu pomocí prohledávání do šířky. Následně je každý uzel ohodnocený číslem, které odpovídá počtu nejkratších cest procházejících tímto uzlem. Uzly, kterými prochází více nejkratších cest tak mají vyšší skóre a jsou považovány za důležitější z pohledu jejich vlivu na tok informací v grafu.

Algoritmus pro zjištění počtu incidentních hran uzlu (popularita uzlu)

Algoritmus je dostupný v knihovně grafových algoritmů databáze Neo4j pod názvem *algo.degree*[23]. Iterativní proces algoritmu spočívá v projití všech uzlů definovaných pomocí Cypher dotazu a jednoduchým spočítáním všech vstupních a výstupních hran daného typu. Typ hran pro výpočet je rovněž nutné definovat v Cypher dotazu. Pro každý uzel je výstupem číslo udávající počet hran tohoto typu.

Netlist konektor

Hlavní úlohou Netlist connectoru je mapování IP adres v grafové databázi na příslušný segment sítě. Při mapování IP adres navíc určuje nejmenší segment z těch, kam lze adresu přiřadit, čímž umožní dohledat nejvíce specifický kontakt pro danou adresu. Poslední úlohou konektoru je dodání informace o doménovém jménu pro každou IP adresu (reverzní DNS záznam), která spadá do sítě organizace. Netlist konektor vychází ze správy aktiv dané organizace a bude tak doménově specifický pro každou organizaci. Netlist konektor je proto spíše ukázková komponenta ilustrující nasazení software v konkrétní organizaci, kde jsou informace o aktivech exportovány do strojově čitelného CSV souboru.

[22] https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/betweenness-centrality/
[23] https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/degree-centrality/

Návod k instalaci

Pro instalaci Softwaru pro evidenci zranitelností v počítačové síti je možné zvolit dva postupy. Prvním z nich je manuální instalace jednotlivých nástrojů na základě přiložených návodů. Druhou možností je využití předpřipraveného Ansible předpisu, který automatizovaně nasadí celý software.

Manuální instalace

Tuto možnost je vhodné zvolit v případě instalace konkrétního nástroje pro evidenci zranitelností v počítačové síti. Pro kompletní instalaci je doporučeno využít připravený Ansible balíček.

Každý nástroj obsahuje kromě zdrojového kódu i README soubor, který shrnuje jeho případy využití, obsahuje informace o závislostech potřebných pro úspěšný běh nástroje jakož i návod pro manuální instalací nástroje.

Automatizovaná instalace

Prerekvizity

Před zahájením samotné instalace je třeba nainstalovat podpůrné systémy, které zaručí bezproblémovou instalaci Softwaru pro evidenci zranitelností v počítačové síti. Níže je možné najít konkrétní verze podpůrných systémů, u kterých je garantováno, že instalace proběhne bez problémů.

- Ansible 2.9.10
- Vagrant 2.2.9
- VirtualBox 6.0.24 r139119

Je vhodné poznamenat, že jiné verze výše zmíněných podpůrných systémů nutně nemusí být problémové a je velká šance, že ani nebudou.

Příprava

Instalační balíček je dostupný ve složce **ansible**. Před samotným spuštěním je nutné doplnit konfigurační údaje k jednotlivým nástrojům v konfiguračním souboru **ansible/group_vars/all/vars**. Konkrétně je požadováno následující:

- flowmon_key_path - lokální cesta k SSH klíči který má přístup na FlowMon kolektor.
- flowmon_ssh_passphrase - heslo k výše uvedenému SSH klíči.
- warden_ca_path, warden_cert_path, warden_key_path - lokální cesty k souborům potřebných pro korektní běh Wardenu. Tyto soubory je možné získat na https://warden.cesnet.cz/cs/participation
- neo4j_password - Vámi zvolené heslo pro grafovou databázi.
- flower_password - Vámi zvolené heslo pro grafické rozhraní orchestrační služby.
- rtir_user, rtir_password - Přihlašovací údaje do tiketovacího systému RTIR.
- shadowserver_user, shadowserver_password - Přihlašovací údaje do služby Shadowserver.

- shodan_api_key - api klíč s přístupem na Shodan api.

Instalace

Pro nainstalování celého balíčku je dostačující zadat příkaz **vagrant up** ze složky **ansible**.

Kontrola instalace

Instalací si lze ověřit dostupností následujících služeb ve vytvořeném prostředí:

- Grafová databáze Neo4j - dostupná na portu **TCP/7474**
- Orchestrační služba - dostupná na portu **TCP/5555**
    - V rozhraní orchestrační služby je možné vidět právě probíhající i již dokončené úkoly.

Uživatelská dokumentace

Software pro evidenci zranitelností v počítačové síti je z velké části navržen, aby běžel samostatně bez interakce s uživatelem. Jednotlivé komponenty autonomně sbírají a zpracovávají data a výsledky své činnosti ukládají do centrální databáze. Pro interakci s uživatelem a prezentaci dat je primárně určen Výsledek č. 2 - Webová aplikace pro vizualizaci bezpečnostní situace v počítačové síti. Nicméně Výsledek č. 1 lze provozovat samostatně a získat jeho veškerou funkcionalitu.

Správa aplikace

Pro monitorování běhu Software a jeho dílčích komponent slouží Orchestrační služba. Její grafické rozhraní v podobě systému Flower je dostupné na serveru s Výsledkem č. 1 na portu TCP/5555. Flower uživateli poskytuje přehled o stavu jednotlivých částí výpočetního clusteru, statistiky úspěšně a neúspěšně dokončených úloh a průměrné zatížení výpočetních uzlů. Ukázka dlouhodobě nasazeného výpočetního uzlu s úlohami Výsledku č.1 můžeme vidět na Obrázku č. 4.

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@crusoe	Online	6	123088	1198	120686	1200	2.44, 2.1, 1.81

Obrázek 4: Přehled běžících a dokončených úloh výpočetního uzlu

Webové rozhraní také poskytuje detailní přehled všech vykonaných úloh včetně jejich vstupních parametrů a aktuálního stavu úlohy. Pro dokončené úlohy jsou následně doplněny informace o výstupu úlohy a doby potřebné pro její zpracování. Obrázky č. 5 a 6 ukazují příklady úspěšně dokončených úloh Komponenty fingerprintingu operačních systémů a Komponenty Webchecker.

Worker	Name	State	args	Result	Started	Runtime
celery@crusoe	crusoe.OS_parse	SUCCESS	(('Processed scan flag: out, number of flows: 793594', '/data/flow/out_202011181140.json'),)	'New: 0, unchanged: 10501, changed: 1666, inactive: 17099 sessions Measurement: python = 26.317745447158813 neo = 62.96205520629883'	2020-11-18 10:45:54.259	89.280

Obrázek 5: Úspěšně dokončená úloha Komponenty fingerprintingu operačních systémů

Worker	Name	State	args	Result	Started	Runtime
celery@crusoe	crusoe.detect_domains	SUCCESS	(('Processed scan flag: in, number of flows: 882', '/data/flow/in_202011181145.json'),)	'658 domains detected. Measurement: python = 3.030447006225586 neo = 37.21648836135864'	2020-11-18 10:50:07.891	40.248

Obrázek 6: Úspěšně dokončená úloha Komponenty Webchecker

Ukázku neúspěšné úlohy můžeme vidět na Obrázku č. 7. Rozhraní pro takové úlohy poskytuje kompletní přehled chyby, která vedla k selhání úlohy, a *stack trace* pro usnadnění dohledání chyby. V tomto ukázkovém případě došlo k neošetřené výjimce, kdy vstupní IPFIX data obsahovala nevalidní IP adresu, pravděpodobně následkem chyby exportéru.

crusoe.service 166ef72c-4c5b-48b1-ad9b-fe0b44aceaf8

Basic task options

Name	crusoe.service
UUID	166ef72c-4c5b-48b1-ad9b-fe0b44aceaf8
State	FAILURE
args	(('Processed scan flag: out, number of flows: 757041', '/data/flow/out_202011181505.json'),)
kwargs	{}
Result	None

Advanced task options

Received	2020-11-18 15:10:45.045426
Started	2020-11-18 15:10:45.046492
Failed	2020-11-18 15:11:43.604172
Retries	0
Worker	celery@crusoe
Exception	ValueError("'141.60.202.1.23' does not appear to be an IPv4 or IPv6 address")
Timestamp	2020-11-18 15:11:43.604172
Traceback	Traceback (most recent call last): File "/usr/local/lib/python3.7/dist-packages/celery/app/trace.py", line 412, in trace_task

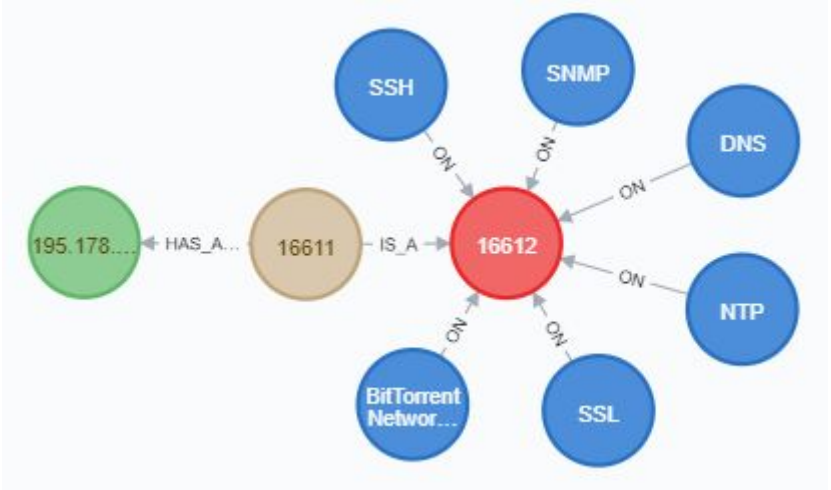
Obrázek 7: Neúspěšně dokončená úloha Komponenty pasivního monitorování sítě

Přístup k výstupům software

Všechny výstupy Software jsou ukládány do centrální grafové databáze, jejíž webové rozhraní je v základním nastavení dostupné na portu TCP/7474. Toto rozhraní umožňuje uživateli procházet uložená data a zadávat příkazy v dotazovacím jazyce Cypher[24] s jejichž pomocí lze provádět pokročilou analýzu dat a získat komplexní pohled na data.

Pro ilustraci možností ruční analýzy dat využijeme prvních pět případů užití specifikovaných v technické zprávě *Návrh architektury systému pro ochranu KII založeném na OODA* vydané v prvním roce řešení projektu. Tyto případy užití odpovídají výstupům Výsledku č. 1 a je možné je přímo zodpovědět správně formulovanými dotazy:

1. Identifikace služeb mapovaných na konkrétní stroje, např. *server A poskytuje službu "mail" a "web"*
    - Cypher dotaz: match (i:IP)-[]-(n:Node)-[]-(h:Host)-[]-(s:NetworkService) where i.address = "256.10.49.121" return i,n,h,s



Obrázek 8: Identifikace služeb běžící na stroji

2. Identifikace strojů
    - určení OS na úrovni Windows/Mac OS X/Linux, ideálně i s verzí

[24] https://neo4j.com/developer/cypher/

- Cypher dotaz: match (i:IP)-[]-(n:Node)-[]-(h:Host)-[]-(s:SoftwareVersion) where i.address = "256.10.49.122" return i,n,h,s

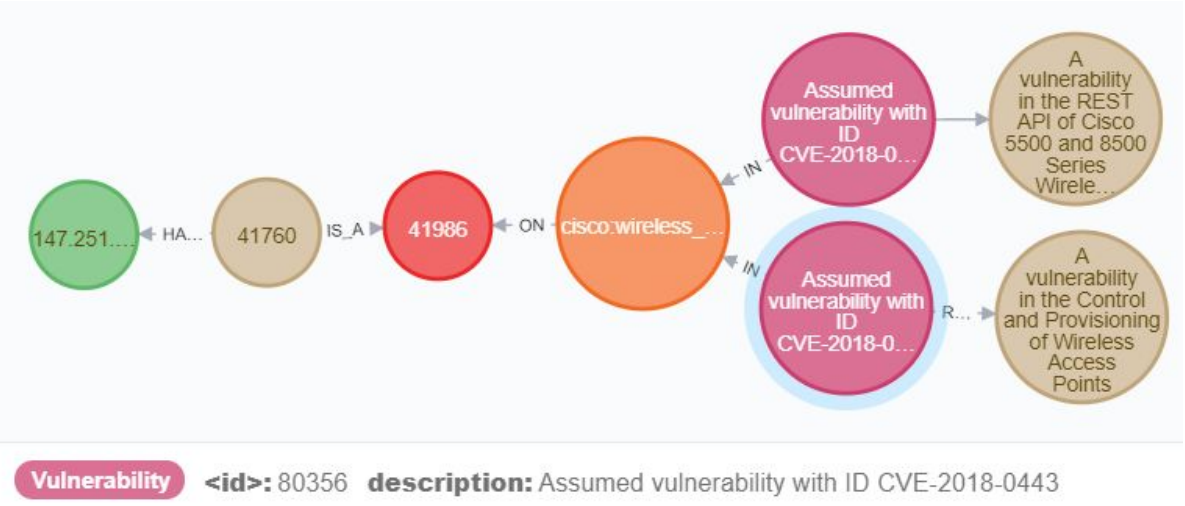
Obrázek 9: Identifikace operačního systému zařízení

3. Seznam zranitelností strojů
   - Detekci zranitelností lze rozdělit do dvou skupin podle způsobu jejich odhalení. První skupinou jsou detekce odhalené aktivním skenem zaměřeným přímo na danou zranitelnost. Mezi ně patří detekce od společnosti ShadowServer a z Komponenty Webchecker.
   - Globální přehled výskytu nově detekovaných zranitelností můžeme získat např. tímto Cypher dotazem: MATCH (i:IP)-[]-(t:SecurityEvent) where t.detection_time > datetime("2021-01-01T00:00:00Z") RETURN t.type, COUNT(t) order by count(t) desc limit 5

"t.type"	"COUNT(t)"
"cert"	17732
"portmapper"	1477
["Vulnerable.Config","Test"]	1440
"rdp"	160
"mdns"	104

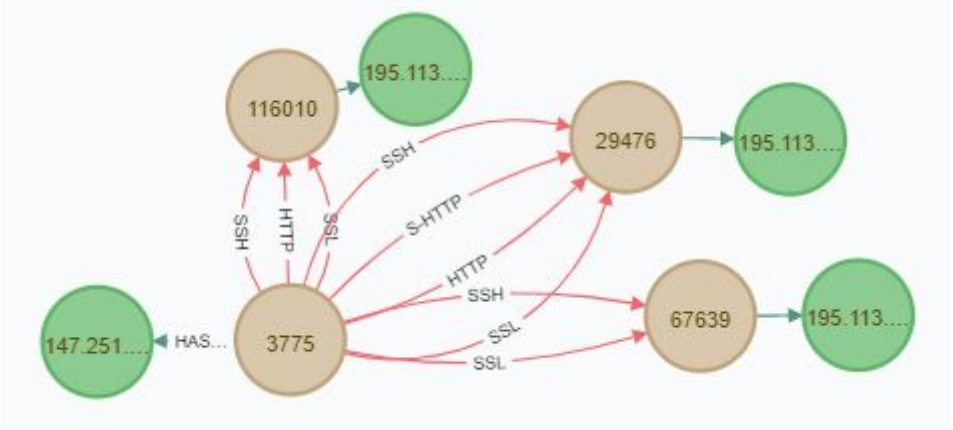
Tabulka 1: Přehled nejčastějších zranitelností v síti

   - Do druhé skupiny zranitelností patří ty, jež byly odhaleny komponentou CVE konektor, která spojuje známé zranitelnosti se zařízeními na základě shody zranitelného software se software detekovaným na zařízení komponentami pasivního a aktivního monitorování sítě.
   - Zjištění zranitelností konkrétního zařízení můžeme provést dotazem: MATCH (c:CVE)-[]-(v:Vulnerability)-[]-(s:SoftwareVersion)-[]-(h:Host)-[]-(n:Node)-[]-(i:IP) where i.address = "256.10.49.123" return c,v,s,h,n,i



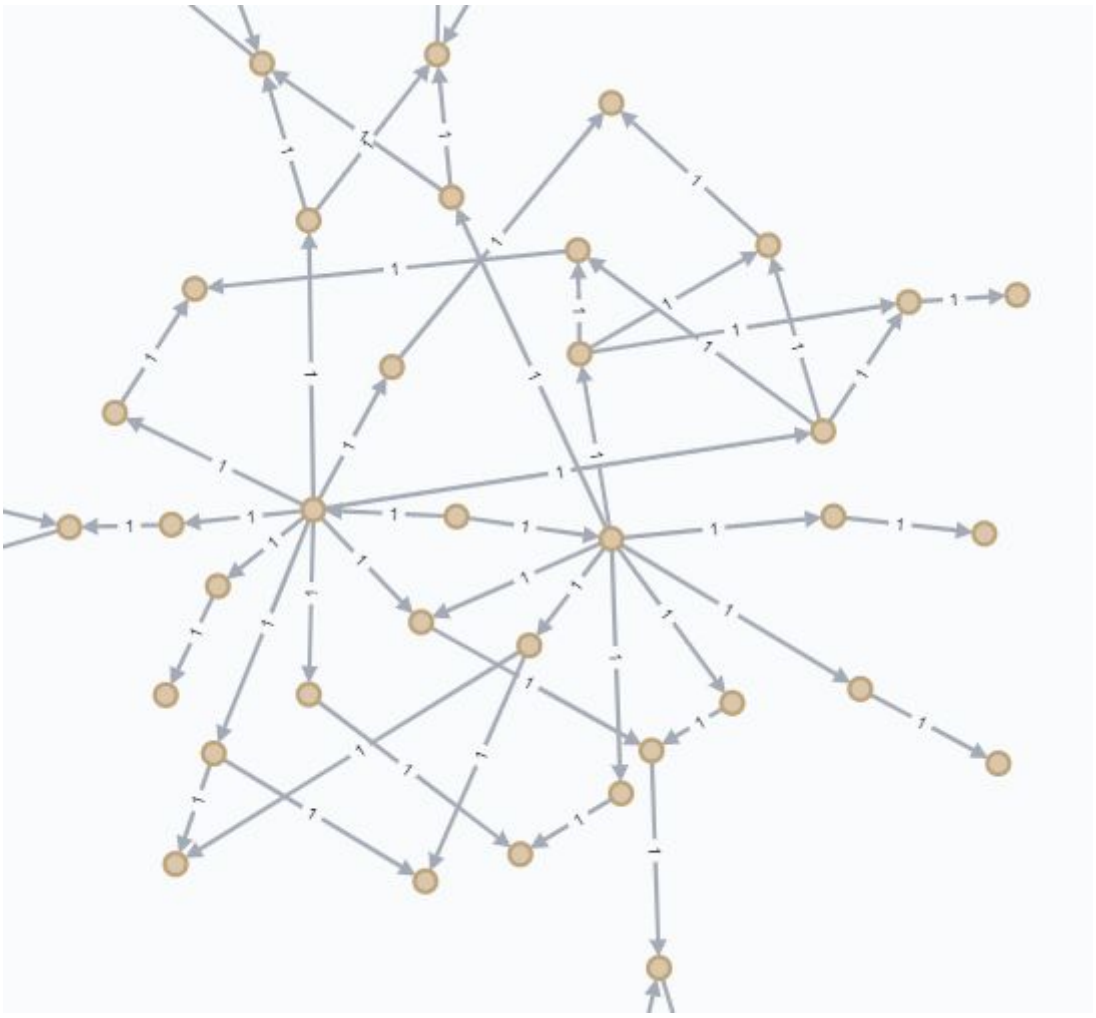
Obrázek 10: Identifikace zranitelností zařízení

4. Závislosti strojů
    - Závislosti strojů jsou identifikovány na základě poskytování a využívání služeb detekovaných pasivním monitorováním sítě. V tomto případě je důležitá orientace hran značící, který stroj je závislý na kterém. Přehled závislostí konkrétního stroje získáme Cypher dotazem: match (i:IP)-[]-(n1:Node)-[dep:IS_DEPENDENT_ON]->(n2:Node)-[]-(i2:IP) where i.address = "256.10.49.124" return i,n1,n2,i2



Obrázek 11: Identifikace závislostí zařízení

5. Topologie sítě
    - Zobrazení kompletní topologie velké sítě je vysoce nepraktické. Proto je lepší zvolit pohled pouze na síťové prvky významné pro správný chod sítě. Takové prvky jsou z celkové topologie identifikovány Komponentou vyhledávání kritických uzlů, která každému uzlu sítě dopočítává metriky jejich významnosti. Hodnoty těchto metrik závisí na velikosti a uspořádání sítě, a proto se budou přesné hodnoty pro Cypher dotaz lišit pro každou síť.
    - Zobrazení nejdůležitějších prvků sítě tak provedeme dotazem: MATCH (n1:Node)-[r:IS_CONNECTED_TO]->(n2:Node) where n1.topology_betweenness > 500 and n2.topology_betweenness > 500 and r.hops =1 RETURN n1,r, n2 LIMIT 100



Obrázek 12: Topologie nejvýznamnějších síťových prvků

Programátorská dokumentace

Zdrojové kódy software jsou doplněny dokumentací jednotlivých metod, která detailně pokrývá jejich použití a parametry. V tomto dokumentu je popsána struktura jednotlivých komponent, detaily jejich implementace a použití. Struktura popisu je shodná s popisem výsledku, tedy odpovídá komponentám dle navržené architektury software.

Podpůrné technologie

Orchestrační služba

Orchestrační služba je uspořádána do šesti souborů – konkrétně dvou Python souborů obsahujících logiku Orchestrační služby, dvou konfiguračních souborů specifikujících jeho chování a dvou pomocných souborů obsahujících dokumentaci a funkční závislosti.

```
filler-orchestration-service
        ├── celery_config.py
        ├── celery_logger.py
        ├── config
        │   └── conf.ini
        ├── crusoe.py
        ├── README.md
        └── requirements.txt
```

V rámci crusoe.py a celery_config.py je pro každou komponentu vytvořena řada úkolů, které mají předdefinované základní parametry, s jakými se daná komponenta používá.

Orchestrator je implementován v následujících souborech:

- crusoe.py - obsahuje hlavní logiku orchestratoru. Jeho úkolem je volání funkcí a zpracování jejich výsledků.
- celery_logger.py - jde o hlavní logger pro orchestrační službu a pro komponenty v ní spouštěny. Logger lze nastavit pomocí parametrů s požadovanými vlastnostmi pro každý komponent zvlášť. Tento přístup umožňuje centralizované úpravy loggeru. Jednotlivé komponenty nemusí implementovat svůj vlastní logger, jelikož orchestrační služba již tuto funkcionalitu nabízí. Na pozadí je logger implementován pomocí systému structlog.
- celery_config.py - hlavní konfigurační soubor orchestrační služby, který definuje s jakou periodou, případně v který konkrétní čas, mají být spouštěny dané tasky. Obsahuje také globální nastavení orchestrační služby.
- config.ini - jedná se o konfigurační soubor určený pro jednotlivé komponenty, které v něm mají svou vlastní specifickou sekci. Obsahuje informace, jako například autentizace, cesta k souborům a další potřebné parametry pro danou komponentu.
- requirements.txt - obsahuje seznam požadovaných Python balíků využívaných v orchestrační službě.

Níže uvádíme seznam předpřipravených úloh v souboru crusoe.py, které je možné naplánovat ke spuštění pomocí konfiguračního souboru:

- rtir_connector - úloha pro spuštění komponenty RTIR konektor
- scan_init - pomocná úloha pro spuštění Komponenty aktivního monitorování sítě
- topology_scan - pomocná úloha pro spuštění Komponenty aktivního monitorování sítě
- vertical_scan - pomocná úloha pro spuštění Komponenty aktivního monitorování sítě
- topology_scan_save - pomocná úloha pro spuštění Komponenty aktivního monitorování sítě
- vertical_scan_save - pomocná úloha pro spuštění Komponenty aktivního monitorování sítě
- attack_graphs - pomocná úloha pro Software č. 3
- shadowserver - pomocná úloha pro spuštění Komponenty získávání informací o zranitelnostech
- shodan - pomocná úloha pro spuštění Komponenty získávání informací o zranitelnostech
- cleaner - úloha pro periodické odmazávání historických dat z databáze
- sabu - úloha pro spuštění komponenty SABU konektor
- act_overseer -pomocná úloha pro Software č. 4
- netlist - úloha pro spuštění komponenty Netlist konektor
- nvd_CVEs - pomocná úloha pro spuštění Komponenty CVE konektor
- vendor_CVEs - pomocná úloha pro spuštění Komponenty CVE konektor
- OS_parse - úloha pro spuštění Komponenty fingerprintingu operačních systémů
- service - úloha pro spuštění Komponenty pasivního monitorování sítě
- check_certs - úloha pro spuštění Komponenty Webchecker
- detect_domains - pomocná úloha pro spuštění komponenty Netlist konektor
- compute_criticality - úloha pro spuštění Komponenty vyhledávání kritických uzlů
- flowmon - úloha pro spuštění komponenty Konektoru kolektoru Flowmon
- flowmon_chain - pomocná úloha pro spuštění komponenty pasivního monitorování sítě
- cms_scan - úloha pro spuštění Komponenty detekce CMS

Použití

V první řadě je třeba se ujistit, že je dostupný broker, jako například redis-server, v opačném případě by spuštění orchestrační služby skončilo chybou. To je možné otestovat příkazem:

```
# redis-cli ping
```

V případě, že chceme využít i monitorovací systém Flower, použijeme ke spuštění příkaz:

```
# celery flower --broker = redis: // localhost: 6379/0
```

Samotný Orchestrator (Celery) se spouští pomocí:

```
# celery worker -A /path/to/crusoe -l = INFO -B
```

REST API

REST API je strukturováno dle použitého Django frameworku a plně přebírá jeho adresářovou strukturu. Jednotlivé soubory a metody rovněž následují běžnou implementaci tohoto frameworku.

```
neo4j-rest
        ├── django
          └── manage.py
          └── db.sqlite3
          └── static
            └── “default Django file structure”
          └── crusoe_django
            └── admin.py
            └── apps.py
            └── conf.ini
            └── models.py
            └── settings.py
            └── tests.py
            └── urls.py
            └── views.py
            └── wsgi.py
            └── middleware
              └── whitelist.py
        ├── README.md
```

Koncové body API (endpoints) jsou definovány v souboru views.py, jejich kompletní výpis a dokumentace je k dostání v příloze tohoto dokumentu.

Použití
Server poskytující REST API je možné spustit příkazem

```
# python3 django/manage.py runserver
```

Databázový adaptér

Databázový adaptér je členěn podle jednotlivých komponent architektury software, kterým poskytuje rozhraní pro manipulaci s centrální databází. Komponenta se skládá z 15 modulů zajišťujících základní funkcionalitu a jednoho modulu podpůrného. Podpůrný modul *AbstractClient* je počátečním bodem komponenty. Jeho hlavní funkcí je vytvoření spojení s databází, které se následně využívá v rámci specifických modulů popsaných níže. Kromě toho obsahuje i sadu dotazů v nativním dotazovacím jazyce Neo4j databáze Cypher Query Language (dále jen Cypher), které slouží k vytvoření omezení zajišťujících integritu dat v databázi. Díky tomu databáze sama zabrání pokusu o vložení chybných dat. Ostatní moduly komponenty pak *AbstractClient* doplňují o funkcionalitu specifickou pro konektory, jimž vytváří rozhraní pro přístup do databáze. Výsledná struktura komponenty vypadá následovně:

```
neo4j-client
        ├── neo4jclient
          └── AbsClient.py
          └── AttackGraphClient.py
          └── CMSClient.py
          └── Cleaner.py
```

```
    └── CriticalityClient.py
    └── CveConnectorClient.py
    └── MissionAndComponentClient.py
    └── NETlistClient.py
    └── NmapClient.py
    └── OSClient.py
    └── RESTClient.py
    └── RTIRClient.py
    └── SabuConnectorClient.py
    └── ServicesClient.py
    └── VulnerabilityCompClient.py
    └── WebCheckerClient.py
├── README.md
├── test/
```

Pro ověření funkčnosti Neo4j-client komponenty byla s využitím frameworku pytest[25] napsána také sada testů. Tyto testy pomocí vkládání tzv. mock dat do testovací instance databáze ověřují, že jednotlivé funkce modulů vkládají data správně a perzistentně do databáze. Testy pro tuto komponentu se nacházejí ve složce test a je možné je volat z kořenového adresáře komponenty.

Použití

Komponenta Neo4j-client je primárně využívána v rámci konektorů fáze Observe. Pro její použití v konektoru je třeba:

1. Importovat požadovaný modul:
   ```
   # from neo4jclient import <ClientYouWantToUse>
   ```
2. Druhým krokem je inicializace spojení s databází pomocí následujícího příkazu:
   ```
   # client =
   <ClientYouWantToUse>.<ClientYouWantToUse>(password="Pass")
   ```
3. Posledním krokem je pak samotné volání požadované metody. To je možné pomocí následujícího příkazu:
   ```
   # client.<MethodYouWantToUse>
   ```

Centrální databáze

Pro implementaci systému byla vybrána databáze Neo4j, která spadá do kategorie takzvaných grafových databází. Pro software není potřeba databázi žádným způsobem upravovat, pouze správně nastavit firewall a listenery, aby byla databáze přístupná pro komponenty software.

Použití

Po instalaci Neo4j databáze (např. pomocí dodaných Ansible skriptů) je dostupné grafické rozhraní databáze na adrese:

```
# http://localhost:7474
```

[25] https://docs.pytest.org/en/latest/

Datové konektory

Konektor kolektoru Flowmon

Komponentu tvoří jeden soubor flowmon_connector.py a poskytuje dvě metody, *download_ssh()* a *download_rest()*. Obě metody splňují stejnou funkcionalitu. Po připojení na kolektor jsou získaná síťová data ukládána do předem definovaného souboru ve formátu JSON.

```
flowmon-connector
        ├── flowmon_m/
        │   ├── flowmon_connector.py
        ├── README.md
        ├── setup.py
        ├── requirements.txt
```

Použití

Při stahování méně než 10 000 záznamů z kolektoru se doporučuje použít REST API, což je zajištěno metodou:

```
>>> flowmon_connector.download_rest()
```

Funkce zprostředkovává komunikaci s Flowmon REST API, odkud jsou získávána požadovaná data. Prostřednictvím argumentů, které byly metodě předány, je umožněna autentizace, nastavení časového okna získaných síťových dat, nastavení domény, na kterých je přístupný Flowmon REST API a také filtr dat. V případě stahování většího objemu dat, tedy stahování více než 10 000 záznamů, je nutné použít službu SSH. K tomuto účelu slouží metoda:

```
>>> flowmon_connector.download_ssh()
```

Služba SSH není primárně určena k získávání síťových toků, a proto je potřeba, kromě výše zmíněných parametrů (filtr, autentizace, časové okno) v metodě REST API definovat i další parametry. Konkrétně jde o cestu k aplikaci nfdump a její parametrům, které se používají na analýzu dat přímo na kolektoru a na konvertování do čitelné formy. Specifikovaná je struktura souborů, ve kterých se síťové toky nacházejí, a sond, ze kterých se mají data zpracovat.

Komponenta Aktivního Monitorování Sítě

Komponenta se skládá ze tří základních částí – samotného procesu skenování, parsování a modulu zodpovědného za údržbu konzistence dat v databázi, jako je například uzavírání neaktualizovaných spojení. Adresář projektu má tuto strukturu:

```
nmap_topology_scanner
        ├── nmap_topology_scanner/
        │   ├── scanner.py
        ├── README.md
        ├── setup.py
```

Scanner obsahuje funkci *scan()* obstarávající skenovací proces, přičemž jako argument bere slovník s konfiguračními údaji. Ten mimo jiné udává i velikost a počet subnetů, které se mají v jednom volání tohoto procesu skenovat, nebo přihlašovací údaje ke strojům, které mají být použity ke skenování. Po ukončení skenu vybraných subnetů jsou výsledky zapsány do XML souboru (ve formátu nástroje nmap, který je pro samotné skenování využíván) na stroji, který skenování prováděl. Modul skener tento soubor stáhne na stroj, který modul spustil, a předá jej k dalšímu zpracování.

Výsledky provedeného skenu jsou uloženy do grafové databáze. Jelikož jsou skeny prováděny opakovaně z více různých míst, musí tento modul zajistit jak přidávání nově detekovaných spojení, tak aktualizaci spojení detekovaných dříve. Komponenta proto obsahuje funkcionalitu sloužící k uzavírání neaktualizovaných spojení.

Použití
Scanner lze spustit následovně:

```
>>> from nmap_topology_scanner import scan
>>> scan(subnets, 'nmap args', logger_instance)
```

Parametr *subnets* je v tomto případě seznam obsahující adresní rozsahy, která se mají skenovat.

Komponenta Pasivního Monitorování Sítě

Komponenta se skládá z následujících položek:

```
services-component
    ├── src
    │   └── core.py
    │   └── rules.py
    │   └── run.py
    │   └── utils.py
    │   └── data
    │       └── av.json
    │       └── si_nbar.json
    │   └── services
    │       └── antivirus.py
    │       └── browser.py
    │       └── service_identifier.py
    ├── README.md
    ├── setup.py
```

Adresář *src/data* obsahuje další datové soubory požadované komponentou nebo subkomponentami. *run.py* provede jednotlivé dílčí komponenty detekční služby, zkombinuje jejich výsledky a vytvoří nový soubor s výsledky. *core.py* obsahuje implementaci objektu *Result*, který obsahuje *Hierarchii* pro každé sledované zařízení (IP). Objekt *Hierarchie* podčást identifikace CPE (prodejce: produkt: verze). *Hierarchie* obsahuje strom, jehož úrovně se skládají z objektů *Vendor*, *Product* a *Version*. Tato struktura je navržena pro

výsledky jednotlivých detekčních metod a další zpracování. Každá dílčí součást by měla tyto třídy použít ke zjednodušení interakce s ostatními částmi komponenty. Třída *Result* obsahuje mapování mezi softwarem detekovaným na dané IP a přidruženým objektem *Hierarchie*. *rules.py* je navržen pro zjednodušení identifikace používání služeb zpracování záznamů síťových toků na základě pravidel. Poskytuje objekt pravidel, který je inicializován sadou předkonfigurovaných pravidel, která jsou poté slouží k porovnávání s toky.

Použití

Vstupním bodem komponenty je metoda run, která je exportována přímo na nejvyšší úroveň. Nejprve je potřeba importovat komponentu a poté ji lze přímo spustit:

```
>>> import services_component
>>> services_component.run(vstupni_soubor, vystupni_soubor)
```

Komponenta Fingerprintingu Operačních Systémů

Komponenta je složena ze tří modulů pro základní funkcionalitu a tří modulů, které implementují jednotlivé identifikační metody. Implementace se nachází ve složkách *osrest* a *osrest/method*.

```
OS-parser-component
    ├── osrest
    │   └── method
    │       └── domain.py
    │       └── tcpml.py
    │       └── useragent.py
    │   └── data
    │       └── os.json
    │       └── num2os.json
    │       └── config.ini
    │       └── fingers_map.csv
    │       └── train.csv
    │   └── OS_parser.py
    │   └── run.py
    ├── test
    │   └── test_os.py
    ├── README.md
    ├── setup.py
```

Modul *run* je vstupním bodem komponenty. Obsahuje metodu *parse*, která přijímá dva argumenty - cestu k souboru se síťovými toky ke zpracování a cestu k souboru pro výstup. Metoda *parse* načte síťové toky ze souboru, předá je modulu *OS_parser* ke zpracování a následně zapíše výsledky do výstupního souboru.

Modul *OS_parser* je hlavní částí komponenty. Obsahuje volné funkce, které dokáží inicializovat jednotlivé detekční metody, spustit je nad síťovými toky, sloučit jejich výsledky, a připravit je k výstupu.

Jednotlivé detekční metody jsou navrženy jako samostatné objekty, které v konstruktoru přebírají potřebné individuální nastavení a poté již dokáží fungovat zcela samostatně pomocí jednotného rozhraní. Toto rozhraní vyžaduje existenci jediné metody *run*, která přebírá načtené síťové toky jako parametr a na výstupu vrací svůj výsledek ve tvaru slovníku mezi IP adresami a mapováním názvů OS na čísla, která reprezentují míru důvěry v daný OS.

Komponenta v databázi vytváří uzel typu OS, který nese informaci o verzi konkrétního operačního systému. Tento uzel je propojen s uzlem typu IP hranou HAS_OS. Ta nese atributy postihující proměnlivost prostředí:

- *start_time* – časová známka detekce operačního systému pro danou IP,
- *end_time* – časová známka poslední detekce OS u dané IP. V případě, kdy nedochází ke změně OS na IP adrese, je tento timestamp pouze inkrementován s každou detekcí. V případě dynamického přidělování IP adres pak spolu se start_time jasně vymezuje časové období, ve kterém adresu používalo zařízení s daným operačním systémem.

Popsaný uzel OS přesně neodpovídá datovému modelu, a proto bude v následujícím roce vyvíjen způsob mapování výstupů fingerprintingu na jasně definovaná CPE a tím pádem odpovídající entitě *Software Resource* datového modelu.

Použití

Vstupním bodem komponenty je metoda parse modulu *run*. Nejprve je potřeba importovat komponentu a poté zavolat metodu *parse()* s cestami ke vstupnímu a výstupnímu souboru:

```
>>> from osrest import run
>>> run.parse(flow_path, output_path)
```

Komponenta detekce CMS

Celá logika komponenty je obsažena v jednom skriptu *scanner.py* a pomocném textovém souboru ve formátu JSON. Tento soubor obsahuje informace o CPE (Common Platform Enumeration) a jejich verzích tak, aby byla komponenta kompatibilní s datovým modelem a dalšími komponentami. Strukturu komponenty lze znázornit následujícím diagramem:

```
CMS-component
    ├── cmsscan
    │   ├── data
    │   │   └── cms.json
    │   ├── scanner
    │   │   └── scanner.py
    ├── README.md
    ├── setup.py
```

run() je základní metoda pro spuštění celého modulu. Její úlohou je načtení informací o CPE z konfiguračního souboru a následná identifikace CMS na zařízeních v síti. Toho dosahuje spuštěním nástroje WhatWeb[26], který síť aktivně skenuje a vrátí výsledky. Metoda *parse()* se stará o zpracování výstupů skenu a mapování jednotlivých názvů a verzí CMS do

[26] https://github.com/urbanadventurer/WhatWeb

standardního formátu CPE. Výstupem metody jsou strukturované informace o jednotlivých CMS v síti ve formátu CPE.

Soubor cms.json obsahuje údaje o službách, které komponenta detekuje. V souboru se nachází textový název služby a její verze, které následně slouží pro mapování vstupních dat na data výstupní.

Požadavky
Pro úspěšné spuštění komponenty je potřeba mít nainstalovaný nástroj WhatWeb. Ten je volně k dispozici na adrese https://github.com/urbanadventurer/WhatWeb.

Použití
Komponentu je možné spustit pomocí metody:

```
>>> cmsscan.run(whatweb_path, hosts, extra_params="-q",
out_path, cpe_path="", logger=structlog.get_logger(),
is_file=True)
```

Metoda přijímá na vstupu následující parametry:

- whatweb_path - cesta k nástroji WhatWeb.
- hosts - seznam webů, které má komponenta oskenovat.
- extra_params - volitelné parametry nástroje WhatWeb.
- out_path - cesta k souboru, do kterého bude zapsaný výstup komponenty.
- cpe_path - cesta k souboru s údaji o službách, které komponenta detekuje. Při nevyplnění parametru se použije defaultní soubor *cms.json*.
- logger - instance loggeru pro zaznamenávání stavu komponenty a jejího běhu.
- is_file - v případě, že je seznam webů, které se mají skenovat, načítaný ze souboru, je tento parametr nastavený na *True* a parametr *hosts* obsahuje cestu k tomuto souboru.

Komponenta Webchecker

Struktura komponenty je velmi jednoduchá. Obsahuje jeden hlavní skript s veškerou funkcionalitou. Struktura vypadá následovně:

```
webchecker-component
    ├── src
    │   └── core.py
    ├── README.md
    ├── setup.py
```

Core modul se skládá z jediné třídy *Webchecker* a obsahuje následující metody:

- *__init__* - Konstruktor třídy. Po vytvoření nového objektu se tato rezervovaná metoda postará o inicializaci logování a načtení konfigurace.
- *run_detect()* - Metoda pro spuštění detekce aktivních webů. Jediným parametrem je cesta k souboru obsahujícímu aktuální síťové toky dodané podpůrnými nástroji fáze Observe. Metoda načte aktuální data a postupně buduje seznam všech webů, na

které přistoupil alespoň jeden uživatel. Pro zajištění konzistence dat metoda navíc kontroluje formát IP adres a pomocí DNS překladu i korektnost domény. Výstupem metody je seznam všech navštívených webů s časem poslední aktivity.
- *get_ips()* - pomocná metoda pro validaci správnosti vazby mezi IP adresou a doménovým jménem.
- *run_certs()* - metoda pro spuštění kontroly certifikátů. Slouží jako wrapper metody *check_cert*, které postupně předá všechny weby ke kontrole a vrací výsledky kontroly certifkátů.
- *check_cert()* - metoda pro samotnou kontrolu platnosti certifikátu. Jako vstup dostane doménové jméno, která má zkontrolovat, a následně spustí aktivní kontrolu. To provádí pomocí aktivního dotazu na daný server

Výstupem části komponenty kontrolující certifikáty je informace o problémech s nimi. Konkrétně je výstupem JSON struktura, která obsahuje následující informace.
- *time* - údaj o času detekce.
- *data* - seznam obsahující položky s jednotlivými problémy. Každá položka obsahuje:
    - *type* - zpřesnění typu položky, vždy obsahuje hodnotu “cert”. Výstup komponenty je v rámci fáze Observe dále zpracovávaný a tato informace slouží k odlišení detekcí problémů s certifikáty od jiných detekovaných problémů v síti.
    - *hostname* - hostname stroje, u kterého byl problém detekovaný.
    - *description* - detailní informace o problému. Obsahuje vždy jednu z následujících možností:
        - "Expired certificate."
        - "Self-signed certificate."
        - "Certificate revoked."
        - "Certificate hostname mismatch."
        - "Certificate will expire on {date}."
        - "Unspecified certificate error."
    - *confirmed* - informace o tom, jestli byl problém potvrzený manuální kontrolou. Jde o další z údajů, který je předpřipravený pro automatizaci práce s výstupem v rámci fáze Observe. Jeho hodnota je iniciálně vždy *False*, protože manuální kontrola může proběhnout až po automatické detekci.

Druhým výstupem komponenty je informace o aktivních webech. Stejně jako při kontrole validity certifikátů je výstupem JSON struktura. V tomto případě má následující formát:
- *time* - údaj o času detekce.
- *data* - seznam položek s následujícími informacemi:
    - *ip* - adresa webu, na který byly detekovány přístupy uživatelů dle informací z pasivního monitorování síťového provozu.
    - *hostname* - hostname, korespondující s výše uvedenou IP adresou.

Použití

Vzhledem k tomu, že komponenta vykonává dvě rozdílné úlohy, je i její spuštění rozděleno do dvou samostatných celků. V obou případech je ovšem pro korektní běh komponenty nutné vytvořit instanci třídy *Webchecker* následujícím způsobem:

```
>>> from webchecker_component import Webchecker
>>> wc = Webchecker(config, logger_object)
```

Konstruktor třídy bere na vstupu dva parametry:

- *config* - slovník, jehož úlohou je bližší specifikace IP rozsahů, se kterými bude komponenta pracovat. Slovník může mít jednu z následujících struktur:
    - *prázdný slovník* - případ, kdy nechceme nijak omezovat rozsah zpracovávaných IP rozsahů.
    - *slovník obsahující následující položky:*
        - *"ignore"* - seznam IP rozsahů, které budou ignorované při kontrole certifikátů.
        - *"target_network"* - seznam IP rozsahů, u kterých bude komponenta detekovat aktivní weby.
- *logger_object* - instance loggeru, která bude použitá pro aktuální běh komponenty. V případě, že zůstane tento parametr nevyplněný, bude použita instance standardního *structlog*.

Po vytvoření instance třídy má uživatel následující možnosti spuštění:

1. Kontrola certifikátů
   ```
   >>> wc.run_certs(hostnames)
   ```

Kde parametr *hostnames* je seznam párů (IP, doménové jméno) nebo jen seznam doménových jmen. Kontrola bude provedena jen nad položkami, které jsou definované v seznamu *hostnames*.

2. Detekce aktivních webů
   ```
   >>> wc.run_detect(flows)
   ```

Kde parametr *flows* je cesta k souboru, který obsahuje síťové toky, nad kterými má samotná detekce probíhat.

Komponenta Získávání Informací o Zranitelnostech

Komponenta se skládá ze dvou částí. První je subkomponenta shadowserver, složený ze čtyř souborů napsaných v jazyce Python, které se nacházejí ve složce *shadowserver_module*. Druhá část obsahuje 2 soubory ve složce *shodan_module*. Kromě toho jsou k dispozici testy v podsložce tests:

```
vulnerability_component/
├── shadowserver_module/
│   └── Download.py
│   └── Remove.py
│   └── Neo4j.py
│   └── Shadowserver.py
├── shodan_module/
│   └── Shodan.py
│   └── Shodan_config.json
├── tests/
│   └── data/
│       └── example-masaryk_university-ip.csv
```

```
    └── rooibos-masaryk_university-isp.csv
    └── threat-masaryk_university-ip.csv
  └── testCSV/
    └── 2018-10-22-botnet_drone-masaryk_university-ip.csv
    └── 2018-10-23-scan_redis-masaryk_university-ip.csv
    └── 2020-10-22-scan_error_csv1-masaryk_university-ip.csv
    └── 2020-10-22-scan_valid_csv-masaryk_university-ip.csv
    └── 2020-10-22-valid_csvs-masaryk_university-ip.csv
    └── invalid_csv1.csv
    └── invalid_csv2.csv
  └── neo4j_test.py
  └── remove_test.py
├── README.md
├── setup.py
```

- Download.py – Tento modul subkomponenty ShadowServer se stará o stahování dat ze serverů organizace ShadowServer. Modul nejprve naváže spojení s externím serverem, provede autentizaci, a následně ze serveru stáhne údaje za poslední den.
- Remove.py – Úlohou tohoto modulu je odstranit chybné nebo neaktuální údaje z modulu Download. Konkrétně tento modul metody zajišťuje:
    - zpracování pouze souborů mladších 12 hodin,
    - odstranění všech stažených souborů, které neobsahují korektní údaje,
    - odstranění všech stažených souborů s již zpracovanými údaji.
- Neo4j.py – Hlavní funkcionalitou tohoto modulu je zpracování stažených souborů a uložení získaných údajů do Neo4j databáze. Modul nejprve získá jména všech souborů z modulu Download. Následně jednotlivé soubory rozparsuje a získá tak v nich obsažené informace. Ty převede do formátu JSON a uloží do databáze.
- Shadowserver.py – Řídící modul, který má na starosti spouštění jednotlivých metod. Po jeho skončení vypíše statistiky o běhu a nalezených zranitelnostech jako výstup do Orchestrační služby.
- Shodan.py - Úkolem tohoto modulu je získat informace o potenciálních problémech ze služby Shodan.
- Shodan_config.json - Konfigurační soubor obsahující informace o problémech, které mají byt hledané v rozhraní služby Shodan.
- tests/ – V této složce se nachází testovací metody pro komponentu.

Údaje ze serverů ShadowServer jsou ve formě csv souborů, které pro každý typ detekce zranitelnosti obsahuje kontextově specifické údaje. Kromě toho mají všechny detekce společné údaje, kterými jsou:

- timestamp – čas detekce,
- ip – IP adresa stroje, na kterém byl problém detekovaný,
- protocol – protokol, který je svázán s detekovaným problémem,
- port – port, který je svázán s detekovaným problémem.

Samotný název detekce je pak dostupný v názvu CSV souboru, který má fixní formát sestávající z časové známky, typu detekce a jména skenované organizace, např. *2018-12-17-scan_ssl_poodle-masaryk_university-ip.csv*.

Před uložením do databáze je potřeba data přeuspořádat do potřebného formátu. Ten má následující strukturu:

- údaje o detekčním systému.
- seznam detekcí za poslední období:
    - IP – IP adresa stroje, kde byl problém detekován,
    - timestamp – čas detekce,
    - vulnerability – název detekovaného problému,
    - description – detailní informace ohledně detekce.

Komponenta v databázi vytváří následující typy uzlů:

- IP - obsahuje údaj o IP adrese,
- DetectionSystem - obsahuje údaj o zdroji detekcí, v tomto případě se v něm nachází název organizace Shadowserver,
- SecurityEvent - obsahuje údaje o bezpečnostním incidentu. Konkrétně se jedná o čas detekce, typ problému, popis problému a údaj o tom, či existence tohoto problému byla potvrzena.

Komponenta také vytváří dva typy hran:

- SOURCE_OF - hrana mezi IP adresou a bezpečnostním incidentem, která indikuje, že stroj s danou IP adresou je zdrojem daného bezpečnostního incidentu,
- RAISED_BY - hrana mezi detekčním systémem a bezpečnostním incidentem, která indikuje, že daný bezpečnostní incident byl detekován daným detekčním systémem.

Použití

Komponentu je možné spustit po spuštění metody:

```
>>> from vulnerability_component import Shadowserver
>>> Shadowserver.process_vulnerabilities("neo4jPass",
    "ShadowLogin", "ShadowPass")
```

která bere na vstup následující parametry:

- neo4jPass - heslo k instanci Neo4j databáze,
- ShadowLogin - přihlašovací jméno k serveru Shadowserver,
- ShadowPass - přihlašovací heslo k serveru Shadowserver.

CVE Konektor

```
cve-connector/
  ├──cve_connector/
    ├──nvd_cve/
      ├── categorization/
        ├── cia_loss.py
        ├── classifier.py
        ├── code_execution.py
        ├── gain_privileges.py
        ├── helpers.py
      ├── test/
        ├── test-data/
```

```
            ├── classifier_data.json
            ├── nvdcve-1.0-2017.json
            ├── test-data1.json
            ├── test-data2.json
          ├── classifier_tests.py
          ├── neo4j_test_client.py
          ├── nvd_test.py
        ├── cve_parser.py
        ├── toneo4j.py
        ├── utility.py
    ├──vendor_cve/
        ├── implementation/
          ├── data/conf.ini
          ├── parsers/
          ├── storing_to_db/neo4j_storing.py
          ├── utilities/
          ├── vendors_storage_structures/
          ├── vulnerability_metrics/
          ├── main.py
  ├── README.md
  ├── setup.py
```

Komponenta se skládá ze dvou dílčích subkomponent: nvd_cve a vendor_cve.

Subkomponenta nvd_cve je zodpovědná za stahování datových kanálů CVE z NVD, jejich analýzu a přiřazování předpokládaného dopadu a uložení do databáze.

- Složka categorization obsahuje implementaci kategorizačního algoritmu rozděleného mezi několik modulů.
- Modul cve_parser.py obsahuje analyzátor dat CVE poskytovaných NVD.
- Modul toneo4j.py obsahuje funkce, které přidávají analyzovaná data do databáze Neo4j podle přítomných CPE v databázi.
- Modul utility.py obsahuje funkce, které stahují a odstraňují zdroj dat CVE z NVD.

Získané soubory ve formátu JSON modul rozparsuje a získaná data uloží do pomocné struktury. Z každého souboru je možné získat obecné informace o souboru, např. počet CVE a informace o každém CVE:

- CVE ID - unikátní identifikátor CVE,
- CPE - identifikátor produktu v podobě výrobce, název, verze produktu,
- CWE ID - identifikátor slabiny,
- URL webové stránky, kde byla zranitelnost zveřejněna,
- popis zranitelnosti,
- CVSSv2 a CVSSv3 - ohodnocení závažnosti zranitelnosti,
- datum a čas zveřejnění v databázi NVD,
- datum a čas poslední modifikace záznamu.

Struktura CPE řetězce a informace v CVSS metrikách jsou detailně popsány v technické zprávě Automatizované získávání informací o zranitelnostech.
Při rozhodování, které CVE do databáze přidat se rozhodujeme následovně:

1. Pokud se v databázi nachází alespoň jedna softwarová verze, které se zranitelnost týká a CVE bylo vytvořeno nebo změněno během posledních 24 hodin, je CVE do databáze přidáno. V opačném případě přidáno není. Následně je vytvořen uzel typu Vulnerability a CVE je s ním spojeno vztahem REFERS_TO. Softwarová verze je pak s uzlem Vulnerability spojena hranou IN.
2. Pokud se CVE v databázi již nachází, ale změnila se některá z jeho vlastností, je CVE aktualizováno tak, aby odpovídalo aktuálnímu stavu v NVD.

Subkomponenta vendor_cve je zodpovědná za stahování dat o CVE z webů výrobců software (podporovaní výrobci jsou Adobe, Android, Apple, Cisco, Lenovo, Microsoft, Oracle, RedHat).

- Složka data obsahuje konfigurační soubor s adresami URL pro weby výrobců.
- Složka parsers obsahuje:
    - obecný analyzátor a analyzátory pro konkrétní typy souborů (HTML, JSON a XML)
    - analyzátory pro konkrétní dodavatele - každá složka obsahuje analyzátory dat poskytnutých výrobcem. Obvykle existuje parser pro hlavní web (končící na main_page_parser.py), který odkazuje na konkrétní webové stránky pro jednotlivé chyby zabezpečení (končící na vulnerability_parser.py).
- Složka storing_to_database obsahuje modul neo4j_storing.py odpovědný za přidání CVE do databáze.
- Složka utilities obsahuje několik pomocných funkcí.
- Složka vendors_storage_structure obsahuje pro každého dodavatele třídu, která ukládá její informace.
- Složka vulnerability_metrics obsahuje funkce související s CVSSv2 a CVSSv3.
- Modul main.py obsahuje funkci vstupního bodu pro zpracování dat od dodavatelů.

Použití
Pro získání dat z NVD je potřeba importovat potřebné funkce pomocí příkazu na prvním řádku. Funkce na čtvrtém řádku pak vykonává samotnou funkcionalitu modulu a potřebuje tři parametry:

- čas - zranitelnosti, které byly vytvořeny nebo modifikovány po tomto datu jsou přidány do grafové databáze. Příklad výpočtu tohoto parametru je uveden na druhém a třetím řádku ukázkového kódu,
- heslo k Neo4j databázi,
- cesta ke složce, kde se budou ukládat soubory stažené z NVD.

```
>>> from cve_connector.nvd_cve import toneo4j
>>> from datetime import datetime, timedelta
>>> specified_time =
(datetime.now()-timedelta(days=1)).isoformat()
>>> toneo4j.move_cve_data_to_neo4j(specified_time,
"neo4jpassword",
```

```
    "/tmp/cve-data")
```

Modul Vendor CVE funguje podobně a rovněž potřebuje složku pro ukládání souborů a heslo k databázi.

```
>>> from cve_connector.vendor_cve.implementation.main import
    add_vendor_CVEs
>>> add_vendor_CVEs("/tmp/ms_downloads", "neo4jpassword")
```

SABU Konektor

Konektor pozůstává z jedného souboru napsaného v pythonu, ktorý obsahuje veškerou funkcionalitu.

```
sabu-connector/
  ├──sabu/
     ├──JsonParsing.py
  ├── README.md
  ├── setup.py
```

Skript JsonParsing.py, sestává ze čtyř metod:

- *warden_is_running* - metoda ověřuje, zda Warden běží a pokud ne, pokusí se ho spustit. V případě neúspěšného pokusu o spuštění metoda vyhodí výjimku.
- *dump_data_to_json_if_ref_key* - metoda uloží všechna získané data ze zpráv ve kterých se vyskytl klíč “Ref” obsahující hodnotu “cve id” do souboru /var/www/neo4j/import/sabu_ref_key.json.
- *dump_data_to_json_if_not_ref_key* - podobně jako u předchozí metody, s výjimkou, že ukládá data ze zpráv ve kterých se klíč “Ref” nevyskytl. Data ukládá do /var/www/neo4j/import/sabu_not_ref.json.
- *parse* - metoda rozparsuje zprávy přijaty z wardenu. Vyselektuje jen ty ve správném formátu a následně s těmito daty zavolá metody na uložení dat do jsonů popsané výše. Nakonec budou tyto json soubory zpracovány neo4j klientem a uloženy do grafové databáze.

Zprávy přijaty ze systému Warden jsou ve formě JSON. Tyto zprávy obsahují, mimo jiné, klíče zajímavé pro další zpracování:

- DetectTime - čas detekce,
- Target.IP4 - IPv4 adresa stroje, na kterém se vyskytuje zranitelnost,
- Impact - popis zranitelnosti,
- Category - typ zranitelnosti.

V případě zprávy obsahující klíč Ref:

- DetectTime - čas detekce,
- Target.IP4 - IPv4 adresa cíle útoku,
- Ref - CVE identifikační číslo .

Použití

Konektor se spouští spuštěním metody:

```
>>> from sabu import JsonParsing
>>> JsonParsing.parse(directory, passwd, regex,
    path_to_warden_filer_receiver, path_to_neo4j)
```

s těmito parametry:

- directory - adresář obsahující přijaté zprávy z wardenu,
- passwd - heslo k instanci Neo4j databáze,
- regex - regulární výraz pro vyhledání IP adresy z rozsahu MU ve zprávě z wardenu,
- path_to_warden_filer_receiver - cesta k programu warden_filer_receiver,
- path_to_neo4j - cesta k adresáři, kde budou uloženy výsledné jsony.

RTIR Konektor

Veškerá funkcionalita komponenty je zajištěna jedním modulem *rtir*. Kromě toho modul obsahuje ještě sadu testů. Celá struktura vypadá následovně:

```
RTIR-connector/
  ├── rtir_connector/
      └── rtir.py
      └── tests/
          └── rtir_test.py
      └── certs/
          └── cert_file.crt
  ├── README.md
  ├── setup.py
```

Modul rtir obsahuje metodu *parse_rt()*, která je výchozím bodem programu. Po jejím spuštění je inicializován logger, do kterého se zapisují informační hlášení. Následně se zpracují argumenty metody a zavolá se metoda *download_data()*, která naváže spojení s RTIR serverem a stáhne seznam aktuálních bezpečnostních událostí. V případě, že tento seznam obsahuje ještě nezpracované události, zavolá se metoda *parse_ticket()*, která vrátí detailní informace o dané bezpečnostní události. Posledním krokem je zpracování takto získaných dat do formátu JSON a jejich následné uložení do grafové databáze.

JSON soubor obsahuje údaje o jednotlivých bezpečnostních událostech, které se následně přidávají do databáze. Každá událost v tomto soubore obsahuje:

- kategorii bezpečnostní událostí,
- datum vytvoření v systému RTIR,
- entitu, která událost vytvořila,
- seznam IP adres, kterých se událost týká,
- e-mailové adresy na ohlašovatele události,
- stručný popis události.

V grafové databázi konektor vytváří následující typy uzlů:

- IP - IP adresa původce události, pokud je k dispozici,
- DetectionSystem - zdroj události, pokud je ohlašovatelem detekční systém,
- SecurityEvent - obsahuje údaje o bezpečnostní události.

Konektor v databázi také vytváří následující hrany:

- SOURCE_OF - hrana mezi IP adresou a bezpečnostní událostí, která indikuje, že stroj s danou IP adresou je původcem dané události,
- RAISED_BY - hrana mezi detekčním systémem a bezpečnostní událostí indikující, že právě tento systém událost detekoval.

Použití

Konektor je spuštěn voláním metody:

```
>>> rtir.parse_rt(user, password, output, uri, subnet_filter,
    last_day, logger)
```

s parametry:

- *user* je přihlašovací jméno k RTIR systému,
- *password* je přihlašovací heslo k účtu *user,*
- *output* je cesta, kam se uloží výsledný JSON soubor,
- *uri* je URL adresa REST API RTIR systému,
- *subnet_filter* je filtr který zajišťuje, aby konektor získal pouze bezpečnostní události z dané podsítě,
- *last_day* je parameter specifikující, aby konektor stáhl pouze data za poslední den,
- *logger* je instance loggeru, kterou bude konektor využívat.

Komponenta vyhledávání kritických uzlů

```
criticality-estimator/
  ├── src/
     ├── core.py
  ├── README.md
  ├── setup.py
```

Úkolem této komponenty je provádět výpočty v databázi s cílem objevování nejdůležitějších uzlů na základě jejich centrality. Výsledky jsou uloženy v rovněž v grafové databázi, proto tato komponenta neexportuje žádná data.

Komponenta se skládá z jednoho souboru:

- core.py se používá pro výpočet kritičnosti (důležitosti) uzlů v databázi.

Komponenta v současné době umožňuje výpočet:

- topologická středová mezipoloha (angl. betweenness centrality)
- topologická centralita měřená stupněm uzlu (angl. degree centrality)
- centralita měřená stupněm uzlu na základě závislostí poskytovaných služeb

Použití

Konektor je spuštěn voláním metody:

```
>>> from criticality_estimator import CriticalityEstimator
>>> ce = CriticalityEstimator(bolt="bolt://localhost:7687",
password="test", logger=logger)
>>> ce.run()
```

Netlist konektor

```
NETlist-connector/
  ├── NETlist_connector/
     ├── data/
        ├── subnets_data.txt
     ├── contacts.py
     ├── domains.py
     ├── subnetParser.py
  ├── README.md
  ├── setup.py
```

Tato komponenta je odpovědná za průběžné aktualizace názvů domén odpovídajících IP adresám. Kromě toho aktualizuje informace o podsítích i odpovědných kontaktech a organizačních jednotkách. Komponenta se skládá z následujících tří souborů:

- contacts.py je zodpovědný za přidávání kontaktů pro podsítě.
- Úkolem dopmains.py je dohledávat doménová jména k IP adresám.
- subnetParser.py vyhledá podsítě a komunikuje s instancí databáze.

Použití

Konektor je spuštěn voláním metody:

```
>>> from NETlist_connector.subnetParser import NETlist
>>> nl = NETlist("neo4_password", "path to neo4j import
directory", logger=logger)
>>> nl.update()
```

Poděkování

Práce na software byla podpořena z projektu *Výzkum nástrojů pro hodnocení kybernetické situace a podporu rozhodování CSIRT týmů při ochraně kritické infrastruktury* (VI20172020070) řešeného v programu *Bezpečnostní výzkum České republiky* v letech 2017-2020 na Masarykově univerzitě. Autory software jsou Martin Laštovička, Jakub Bartoloměj Košuth, Daniel Filakovský, Antonín Dufka a Martin Husák.

Příloha 1: Architektura systému

V přiloženém souboru *Příloha č. 1 - Architektura systému.pdf* je k dispozici diagram znázorňující architekturu systému pomocí vektorové grafiky.

Příloha 2: Koncové body REST API

Kompletní dokumentace REST API je dostupná na adrese https://app.swaggerhub.com/apis/CSIRT-MU/DASHBOARD/1.0.0. Její offline kopie je dostupná v přiloženém souboru *Příloha č. 2 - Crusoe REST API.pdf*. Přehledový seznam koncových bodů API je uveden níže:

AccessControlLayer
- GET /org_units
- GET /org_units/{name}/subnets

CompleteViews
- GET /acces_control_layer
- GET /host_layer
- GET /mission_layer
- GET /network_layer
- GET /response_layer
- GET /system_layer
- GET /threat_layer

HostLayer
- GET /services
- GET /services/{name}
- GET /software
- GET /services/{name}/ips
- GET /software/{name}/ips

MissionLayer
- GET /mission/{name}/configuration/{config_id}/hosts
- GET /mission/{name}/configurations
- GET /mission/{name}/hosts
- GET /missions
- GET /missions/hosts
- DELETE /missions/{name}
- GET /missions/{name}
- POST /missions

NetworkLayer
- GET /ip/{address}/cve
- GET /ip/{address}/events
- GET /ip/{address}/events/{date}
- GET /ip/{address}/events/latest
- GET /ip/{address}

- GET /ip/{address}/services
- GET /ip/{address}/software
- GET /ip
- GET /subnets
- GET /subnets/{subnet}
- GET /subnets/{subnet}/ips

ResponseLayer

- GET /events/after/{date}
- GET /events/{date}
- GET /events

ThreatLayer

- GET /cve/{cve_id}
- GET /cve/{cve_id}/ips
- GET /cve