Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na identifikáciu súborov

Bakalárska práca

2020
Matej Fedor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na identifikáciu súborov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	FMFI.KI - Katedra informatiky
Školiteľ:	RNDr. Jaroslav Janáček, PhD.

Bratislava, 2020
Matej Fedor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Matej Fedor
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Systém na identifikáciu súborov
File Identification System

Anotácia: Práca sa venuje problematike zbierania a udržiavania informácií o pôvodných súboroch, ktoré sú súčasťou operačného systému Windows a vybraných softvérových balíkov, a problematike využitia týchto informácií na identifikáciu podozrivých súborov na testovanom systéme. Výsledkom práce bude softvérové dielo, ktoré umožní kontrolovať stav testovaného systému s cieľom rozlíšiť, ktorým súborom nie je potrebné venovať pozornosť, nakoľko sú pôvodné, ktoré sú naopak podozrivé, a ktorých povaha je neznáma. Toto dielo uľahčí prácu analytikom pri hľadaní potenciálneho škodlivého kódu v systéme.

Cieľ: Vytvoriť ucelený súbor softvérových nástrojov na:
- zber informácií o pôvodných súboroch a ich uloženie,
- overenie stavu súborov na testovanom zariadení,
- prístup k databáze informácií a spracovanie výsledkov testov cez webové rozhranie,
- podporu prevádzky potrebnej infraštruktúry.

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 28.10.2019

Dátum schválenia: 28.10.2019 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Rád by som sa poďakoval svojmu školiteľovi, RNDr. Jaroslavovi Janáčkovi, PhD., za vynikajúcu spoluprácu a odbornú pomoc pri tejto práci. Vaše užitočné rady a diskusie s Vami mi mnohokrát pomohli lepšie sa orientovať v danom probléme, Vaša dôvera v našu spoluprácu mi zasa umožnila pokojnejšie preklenúť náročné obdobia, kedy som sa nevenoval tejto práci intenzívne. Vďaka patrí taktiež mojej rodine za vytvorenie vhodného pracovného prostredia pre dokončenie tejto práce. Špeciálne by som sa rád poďakoval svojmu bratovi Tomášovi za cennú pomoc pri realizácii návrhu webového rozhrania.

Abstrakt

Kľúčové slová: analýza, nástroj, identifikácia, bezpečnosť

Neoddeliteľnou súčasťou forenznej analýzy v oblasti informačnej bezpečnosti sú softvérové nástroje navrhnuté pre zefektívnenie vykonávaných analýz. Doména, v ktorej sa takéto nástroje uplatňujú špeciálne často, je identifikácia a kontrola integrity súborov v skúmaných zariadeniach. Na nástroje použité v tejto oblasti sú však kladené špecifické požiadavky odvetvia, ktoré je treba brať do úvahy pri ich vývoji. V snahe naplniť tieto požiadavky čelí softvérový vývojár výzvam, ktoré súvisia najmä s výkonnosťou a používateľskou prívetivosťou jeho aplikácie na rôznych softvérových platformách. V neposlednom rade je nutné dbať na minimalizáciu možných bezpečnostných zraniteľností výsledného produktu.

Abstract

Keywords: analysis, tool, identification, security

Software tools designed to improve the efficiency of performed analyses are inseparable part of current forensic analysis related to the field of information security. Especially frequent domain of use of those tools is identification and integrity check of files contained in observed devices. However, there are specific requirements of the industry that need to be considered during the development. In order to fulfill those requirements, software developers are challenged by terms of application efficiency and user-friendliness in multi-platform environment. Last but not the least, it is necessary to minimize the risk of potential security vulnerabilities hidden in resulting product.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Integrácia informačných technológií v určitej oblasti so sebou takmer vždy nesie okrem jasných prínosov aj nové bezpečnostné výzvy a potenciálne zraniteľnosti. Rozmach týchto technológií, ktorého svedkami sme v poslednom storočí, dramaticky navýšil možný dopad, ktorý sú tieto zraniteľnosti schopné nadobudnúť v prípade, že systémy nie sú dostatočne chránené. Tejto situácie sú si vedomí nielen zodpovední bezpečnostní špecialisti, ale aj osoby s neetickými a nezákonnými úmyslami, ktorých zámerom je čokoľvek od sebaobohacovania po poškodzovanie cudzej veci. Kriminálna činnosť za pomoci zneužitia existujúcej IT infraštruktúry je stále populárnejšia cesta, ako tieto nekalé úmysly naplniť. V snahe objasniť prípady takýchto zločinov čelia forenzni analytici po celom svete rastúcim požiadavkám na rýchlosť svojej práce. Schopnosť včas zadržať potrebné dôkazy je pritom kritická pre efektívny priebeh kriminálneho vyšetrovania ako aj pre zamedzenie kapacitného preťaženia špecializovaných vyšetrovacích oddelení.

Úloha, pred ktorou analytici stoja je však neľahká. V závislosti od formy kriminálnej činnosti môže byť doslova každý súbor obsiahnutý v zadržanom zariadení predmetom podozrenia a budúcim dôkazom v prípade. Hľadané môžu byť údaje o kriminálnej činnosti páchateľa rôzne schované v súborovom systéme zariadenia, taktiež však systémové defekty zariadenia, ktoré je skúmané za účelom odhalenia formy, akou sa ho útočník zmocnil. Identifikácia relevantných súborov si často vyžaduje detailnú analýzu súborového systému zariadenia obsahujúceho stovky tisíc súborov [13]. S tendenciou rastu kapacity pamäťových médií a množstva aplikačnej funkcionality sa tieto analýzy stávajú čoraz zdĺhavejšími a vyčerpávajúcejšími. Ich úplná algoritmizácia zas tvrdo naráža na úskalia schopnosti strojovo rozlíšiť povahu skúmaných súborov.

Preto sa v súčasnosti pre zjednodušenie tejto úlohy používajú riešenia čiastočne automatizujúce proces analýzy. Populárnym riešením a štandardom v tejto problematike je prístup, s použitím ktorého sa skúmané súbory rozdelia do viacerých kategórií. Súborom jednotlivých kategórií na referenčnom zariadení, o ktorom prepodkladáme, že nie je iným spôsobom bezpečnostne kompromitované, sú priradené jednoznačné identifikátory na základe ich obsahu. Tieto identifikátory sú spolu s inými užitočnými metadátami uložené v referenčnej databáze. S postupným rozširovaním sa referenčná databáza takýchto *valídnych* súborov stáva pre forenzného analytika užitočným nástrojom. Ne-

skôr, skúmajúc zadržané zariadenie, môže analytik časť súborového systému porovnať s referenčnou databázou a ubezpečiť sa o jeho *integrite*. Vo všeobecnosti sa touto technikou v mnohých prípadoch nedarí identifikovať podozrivé súbory v zariadení. S bohatou referenčnou databázou však môžu byť identifikované súbory, ktoré si nevyžadujú pozornosť a čas analytika. To značne zmenšuje priestor prehľadávania, ktoré analytici vykonávajú manuálne.

Typický benefit bohatej referenčnej databázy, a síce redukcia počtu súborov, ktoré je nutné analyzovať manuálne, je základnou ideou tohto prístupu. Pre niektoré kategórie skúmaných súborov však analitykovi kvalitná referenčná databáza dokáže zaistiť taktiež identifikáciu niektorých podozrivých súborov. Príkladom takej kategórie sú systémové súbory operačného systému a rôzne programové súčasti softvérových balíkov. Je totiž len veľmi málo dôvodov, prečo by v nedotknutom systéme mala existovať programová súčasť rozoznávaná referenčnou databázou podľa svojej polohy a názvu v súborovom systéme, avšak s odlišným obsahom v porovnaní s referenčnou verziou. Takýto nález môže byť indikátorom, že bola súčasť modifikovaná cudzím subjektom alebo *malvérom* a určite si vyžaduje bližšiu pozornosť analytika. Modifikovaný súbor môže v systéme vytvoriť bezpečnostnú zraniteľnosť, umožniť útočníkovi získať kontrolu nad zariadením, dokonca vykonávať procesy na pozadí bez toho, aby si používateľ systému niečo všimol. Jedným zo známych bezpečnostných útokov, ktoré sa týmto spôsobom prejavujú, je *DLL Hijacking* [6]. Pri menej sofistikovaných verziách útoku môže forenzný analytik takéto modifikácie systému okamžite odhaliť porovnaním s referenčnou databázou.

Systémové súbory sú kritické pre bezpečné fungovanie zariadenia a tvoria značnú časť všetkých jeho súborov. Preto je tvorba referenčnej databázy systémových súborov a súčastí softvérových balíkov úplným základom v problematike automatizácie forenznej analýzy v oblasti IT bezpečnosti. Nástroj na určenie povahy systémových súborov je však účinný len do takej miery, do akej dokáže analytikovi poskytnúť detailné informácie o nájdených súboroch. Popri informácii o povahe súborov je dôležitý aj prehľadný prístup k informáciám o softvérových balíkoch, do ktorých patria, operačnom systéme, na ktorom sa pôvodne vyskytujú, prípadne aký typ programovej súčasti tvoria. Všetky tieto informácie môžu urýchliť mapovanie súborového systému zadržaného zariadenia. Dôležitá je taktiež požiadavka na jednoduchú distribúciu nástroja a jeho funkčnosť na širokom spektre platforiem, neobmedzujúc tak pracovné prostredie a zvyklosti forenzného analytika.

Táto práca sa venuje účinnému riešeniu načrtnutej problematiky. Softvérový nástroj, ktorého tvorbu popisuje, si kladie za cieľ pomôcť forenzným analytikom uľahčiť ich prácu a riešiť ich výhrady voči existujúcim riešeniam, zároveň však ponúknuť prostredie dostatočne intuitívne, aby poslúžil komukoľvek so záujmom o podobný produkt. V nasledujúcej kapitole popisujeme existujúce riešenia na trhu a prečo aj v dnešnej dobe vidíme význam pre vznik tohto nástroja.

Kapitola 1

Súčasný stav problematiky

V tejto kapitole sa zaoberáme súčasným stavom a problémami v problematike načrtnutej v úvode. Uvádzame adekvátny prehľad technológií, ktoré sú v súčasnosti na trhu k dispozícii a charakterizujeme koncept prístupu, ktorý bol pri ich tvorbe zvolený pre riešenie problému identifikácie a kontroly integrity údajov. Pokúsime sa v nasledujúcom texte čitateľovi objasniť, ktoré aspekty súčasných riešení koncepčne nevyhovujú náročnejším požiadavkám praxe, naopak, ktoré prezieravé nápady hodné inšpirácie už súčasný trh ponúka. Naprieč kapitolou a najmä v jej závere dbáme, aby z uvedeného pre čitateľa vyplývali motivácie tejto práce, priestor na zlepšenie súčasných riešení, a teda aj praktický význam a užitočnosť tejto práce, hoci k precíznemu popisu dôležitých požiadavok sa ešte dostaneme v nasledujúcej kapitole.

1.1 Voľne dostupné riešenia

Predmetom záujmu tohoto prehľadu sú voľne dostupné riešenia. Na takéto zúženie existuje niekoľko dôvodov. Požiadavkou na túto prácu samotnú je jej voľná dostupnosť pre verejnosť, tvoríme ju ako alternatívu k iným voľne dostupným riešeniam, a preto veríme, že v prípade jej úspechu bude najčastejšie porovnávaná práve s týmito riešeniami. Taktiež, získať dostatočne podrobné charakteristiky komerčného softvéru, ktoré by sme mohli uviesť v tomto prehľade a porovnať sa s nimi, môže byť pre účely tejto práce značne komplikované a napokon aj nepodstatné, pretože si touto prácou nekladieme za cieľ im konkurovať.

1.2 Aplikácie pre kontrolu integrity súborov

Na trhu voľne dostupného softvéru v súčasnosti nájdeme hneď niekoľko nástrojov pre kontrolu integrity súborov. Mnohé z nich ponúkajú omnoho viac funkcionality, ktorá však nesúvisí s problematikou tejto práce. To je spôsobené najmä tým, že tieto nástroje

sú koncipované ako *Intrusion Detection System (IDS)*. V nasledujúcom popíšeme črty typických zástupcov tejto skupiny nástrojov, ktoré sú podstatné v danom kontexte.

1.2.1 Nástroje pre monitorovanie súborového systému

Do tejto skupiny môžeme zaradiť aplikácie, ako sú *Open Source Tripwire* [20], *AFICK (Another File Integrity ChecKer)* [17], *SAMHAIN file integrity* [7] alebo komplexnejší a rozsiahlejší *OSSEC (Open Source HIDS Security)* [19]. Všetky tieto nástroje vznikli za účelom monitorovania stavu súborov zariadenia alebo množiny zariadení, ktoré administrátor považuje za kritické pre správnu funkciu svojho systému. Na efektívne zaznamenanie obsahu definovaných kritických súborov je použitá známa technika, pri použití ktorej sa z obsahov zaujímavých súborov vypočíta *hash* niektorým zo známych *hashovacích algoritmov*. Primárnou úlohou nástroja je notifikovať zodpovedajúceho administrátora o zmenách, ktoré nastali v pozorovaných súboroch a umožniť mu tak rýchlo reagovať na potenciálne konanie neoprávneného subjektu, prípadne škodlivého softvéru.

Zatiaľ čo niektoré nástroje (*Open Source Tripwire, AFICK*) boli určené výhradne pre použitie na jednom zariadení a kontrolu súborov vykonávajú periodickými skenmi súborového systému, väčšina z nich ponúka aj možnosť využitia centralizovaného *server-agent* režimu monitorovania a okamžitých notifikácií o aktuálnom dianí. Takáto možnosť je síce v kontexte účelu týchto nástrojov mimoriadne užitočná, no spolu s funkcionalitou pre vytváranie záznamov o zmenách súborov (*SAMHAIN, OSSEC*), monitorovaním registrov (*OSSEC*) a mnohými ďalšími je jasné, že ich ťažisko funkcionality rieši odlišnú problematiku ako tú popísanú v úvode.

Hlavným problémom koncepcie týchto nástrojov je predpoklad, že povaha a pôvod kritických monitorovaných súborov je administrátorom plne známa. Nástroje sa tak sústredia jedine na skúmanie a monitorovanie obsahu týchto súborov a niekoľkých iných atribútov pridelených súborovým systémom. Nezahŕňajú žiadnu funkcionalitu, ktorá by pomohla bližšie identifikovať neznáme súbory a uchovať takúto informáciu. Pre forenzných analytikov je pritom možnosť určiť pôvod, typ a povahu skúmaných súborov kľúčová rovnako, ako informácia o ich zmenách a správnosti ich obsahu. Hoci všetky z nich majú spoločnú črtu použitia istej formy lokálnej databázy pre uchovávanie metadát o súboroch, tieto nástroje nemožno použiť v spojení s centralizovanou databázou známych valídnych súborov.

Ďalšími komplikáciami pri použití pre popísané účely sú napríklad podpora jedine zastaraných hashovacích algoritmov (*AFICK*), nedostatočná podpora pre rôzne platformy (*SAMHAIN*), či prílišné zameranie sa na riešenie odlišného problému a z toho vyplývajúca komplexnosť a množstvo závislostí na softvéri tretích strán, ktorá neumožňuje forenzným analytikom jednoduché skúmanie zariadenia (*SAMHAIN, OSSEC*).

Zatiaľ čo niektoré popísané nedostatky dokumentujú nevhodnosť aplikácií tejto kategórie pre naše použitie, iné ponúkajú cenné podnety, ktoré sa v našej práci snažíme adresovať.

1.2.2 AIDE -Advanced Intrusion Detection Environment

Nástroje pre monitorovanie súborového systému nie sú príliš vhodné pre potreby forenzných analytikov. Nástroj *AIDE* [8] fakticky taktiež spadá do tejto kategórie, no jeho použitie v oblasti problematiky tejto práce dáva omnoho väčší zmysel. Vyznačuje sa podporou širokej množiny hashovacích algoritmov, z ktorých mnohé spĺňajú aj súčasné požiadavky na bezpečnosť. Okrem identifikácie obsahu súborov sú počas skenovania systému získané prakticky všetky dostupné metadáta o súbore poskytnuté súborovým systémom. Veľmi intuitívnym prvkom je použitie regulárnych výrazov pre špecifikáciu množiny súborov, s ktorými sa počas skenovania bude pracovať. Nástroj pracuje v súčinnosti s viacerými možnými formami lokálnych a centralizovaných databáz, ktoré nesú záznamy o všetkých doposiaľ známych súboroch. Taktiež je to kompaktná staticky linkovaná aplikácia s dôrazom na minimum závislostí, ktorá má fungovať jednoducho a intuitívne na všetkých platformách podliehajúcich štandardu *POSIX*.

Nástroj ako celok svojou stavbou a filozofiou dobre vyhovuje základným požiadavkám forenzných analytikov. Ako si popíšeme v nasledujúcej sekcii, v kombinácii s kvalitnou a bohatou bázou valídnych známych súborov môže byť účinným nástrojom na kontrolu integrity a identifikáciu súborov v systéme. Medzi jeho hlavné nedostatky pre riešenie problému tejto práce však patrí jeho nekompatibilita s platformami spoločnosti *Microsoft*.

Okrem jeho prenositeľnosti na rôzne platformy vidíme priestor pre zlepšenie aj v oblasti množiny atribútov, ktoré si o jednotlivých súboroch možno uchovať. Potrebám analytika by lepšie vyhovela aj informácia o pôvode, tvorcovi a type súboru, pokiaľ je taká k dispozícii, taktiež však záznam o platforme, na ktorej bol súbor nájdený. Niektoré z týchto atribútov však na platformách spadajúcich pod *POSIX* nie sú tak prirodzené ako pre platformy spoločnosti *Microsoft*, tento aspekt teda nehodnotíme nutne ako nedostatok.

Nástroj *AIDE* sa svojím prístupom približuje požiadavkám, ktoré v praxi vznikli. V našej práci zdieľame niektoré prístupy k problematike s jeho tvorcami, kladieme si však za cieľ vytvoriť komplexnejší a univerzálnejší nástroj.

1.2.3 Nástroj md5deep/hashdeep

Nástroj *md5deep* je súbor minimalistických aplikácií, ktoré slúžia na rekurzívne prehľadávanie súborov a adresárov v súborovom systéme a prideľovanie identifikátorov na základe obsahu týchto súborov[3]. Každá z aplikácií prideľuje súborom identifikátory s

použitím inej *hashovacej funkcie*. Dvojice súborov a im prislúchajúcich identifikátorov sú následne uložené do súboru, ktorý slúži referenčný súbor pre prípadné kontroly systému v budúcnosti. V prípade porovnávania nájdených súborov so známymi súbormi sú aplikácie schopné zobraziť súbory, ktoré v referenčnom súbore boli alebo neboli nájdené. Ako referenčné súbory však môžu byť použité aj obsahy niektorých známych referenčných databáz. Jednu takúto databázu popisujeme v sekcii 1.3. Aplikácia *hashdeep* je novšia a momentálne jediná udržiavaná implementácia predchádzajúceho súboru aplikácií[2]. V porovnaní so svojím predchodcom umožňuje paralelne prideľovať súborom identifikátory s použitím viacerých *hashovacích funkcií* súčasne. Naviac poskytuje používateľovi všestrannejšiu možnosť vyhodnotiť nájdené výsledky. Nástroj *hashdeep* je označovaný ako aplikácia pre vykonávanie bezpečnostných auditov systému, je teda jedným z kandidátov, ktorý bol primárne navrhovaný pre oblasť, ktorou sa zaoberá aj táto práca[3].

Minimalistický dizajn týchto aplikácií umožňuje ich podporu pravdepodobne všetkými v súčasnosti používanými platformami. Tá istá minimalistickosť však môže v niektorých prípadoch spôsobiť komplikácie pri ich použití. Fakt, že aplikácie používajú pre prístup k referenčným dátam lokálnu databázu vo forme referenčného súboru komplikuje najmä škálovateľnosť a distribuovateľnosť celého riešenia. Proces údržby referenčných súborov zamestnancov organizácie môže byť pri ich častej aktualizácií únavný. Taktiež množstvo dát uložených v tejto lokálnej forme databázy môže mať výrazný dopad na rýchlosť ich vyhľadávania, čo limituje možnú škálovateľnosť riešenia. Na záver, množina atribútov, ktoré sú v referenčnom súbore obsiahnuté, a síce absolútna cesta k súboru a jeho identifikátor, nie sú postačujúce pre rýchlu identifikáciu skúmaných súborov. Pre takúto možnosť musí organizácia využiť niektorú z existujúcich udržiavaných referenčných databáz.

V jednoduchších scenároch môžu byť popisované aplikácie užitočnými pomocníkmi. Ich prevádzka však môže byť najmä pre väčšie organizácie s potrebou vytvorenia vlastnej kustomizovanej a kvalitnej referenčnej databázy nepraktická.

1.3 Existujúce zdroje informácií o súboroch

Ubehla už dlhšia doba, odkedy sa vyskytol problém s potrebou analyzovať pri kriminálnych vyšetrovaniach a bezpečnostných inšpekciách veľké množstvo používateľských dát, súčastí operačného systému a iných softvérových balíkov. Problému sa medzičasom zhostila organizácia *National Institute of Standards and Technology (NIST)* [12], vyhlásený technologický gigant a vládny projekt (U.S. Department of Commerce) zodpovedný za výskum v mnohých oblastiach. Jedným z oddelení tejto organizácie je *National Software Reference Library (NSRL)* [5], ktoré sa dlhodobo venuje podpore

a presadzovaniu efektívneho použitia modernej výpočtovej techniky v kriminálnych vyšetrovaniach súvisiacich s informačnými technológiami.

Stabilným produktom tohoto oddelenia je aj voľne dostupný *Reference Data Set (RDS)*. Rozsiahla kolekcia známych súborov, ktoré tvoria súčasti najrôznejších aplikácií, obsahuje informácie od digitálnych odtlačkov vytvorených viacerými hashovacími algoritmami až po kompletné vytrasovanie objektu k jeho vzniku. Keďže trasovateľnosť jednotlivých súborov, a teda kompletná informácia o ich pôvode je podmienka pre ich zaradenie do RDS, táto zbierka viac než *20 miliónov* súborov a dokopy *10 GB* metadát je výborným zdrojom informácií o známych a valídnych súboroch rôznych aplikácií. Špecificky vytvorená pre pomoc eliminovať z procesu analýzy známe súbory a urýchliť tak proces vyšetrovania, ba čo viac, *RDS* je neustále rozširovaný o nový softvér a aktualizovaný na štvrťročnej báze.

Pre úplnosť doplníme, že k dispozícii sú aj referenčné databázy iných oddelení *NISTu*, ktoré obsahujú digitálne odtlačky mnohých odlišných kategórií súborov, od tých spojených s kriminálnou aktivitou, ohrozovaním informačnej bezpečnosti až po databázy materiálov spojených s násilím na deťoch a mladistvých, prístup ku ktorým je extrémne obmedzený, trasovateľnosť súborov je takmer nemožná a ďalej sa nimi v tejto práci nebudeme zaoberať.

Aktuálnu verziu *RDS* je možné voľne stiahnuť a validovať voči nej súbory s použitím lokálnych prostriedkov. Pre tento účel vytvoril *NIST* dvojicu terminálových aplikácií. Zatiaľ čo serverová aplikácia *nsrlsrv* umožní načítať aktuálny stiahnutý *RDS* a počúvať požiadavky klientov o informácie vzťahujúce sa na rôzne súbory na základe mena súboru alebo jeho digitálneho odtlačku, *nsrllookup* umožňuje dopytovať sa po týchto informáciách. Napokon ešte samotné *NSRL* hostuje webovú službu, ktorá poskytuje klientom informácie z *RDS*.

RDS je obrovský príspevok komunite forenzných analytikov a cenný zdroj dát, žiaľ, nesie so sebou aj určité nedostatky. Digitálne odtlačky súborov sú počítané hneď tromi *hashovacími algoritmami* (*SHA-1, MD5, CRC32*), z ktorých ani jeden nemôžeme v dnešnej dobe považovať v praktickom použití za dostatočne odolný. *NSRL* na tomto probléme pracuje a používateľ si môže stiahnuť taktiež konverznú databázu s priamymi priradeniami odtlačkov vytvorených pomocou *SHA-1* k odtlačkom v *SHA-256*. Zatiaľ čo problém je pre používateľa vyriešený, pre systémového administrátora znamená toto riešenie značnú redundanciu dát.

Zároveň je prirodzené zaujímať sa o úplnosť RDS. Tejto téme sa v našej práci venujeme neskôr, načrtneme však, že potreby každého oddelenia pre forenznú analýzu sú odlišné, typický charakter softvérových balíkov, s ktorým toto oddelenie prichádza do kontaktu a preveruje ho, špecifický, a preto je prirodzené, že *RDS* nemôže vyhovieť všetkým používateľom. Preto budeme pracovať aj na návrhu vlastnej bázy metadát, ktorý umožní svojou univerzálnou štruktúrou vytvárať kustomizované bázy aj iným

entitám so špecifickými potrebami.

1.4 Problémy voľne dostupných riešení

Zatiaľ čo na trhu s komerčnými softvérmi pre forenzných analytikov nie je núdza o kvalitné (pokojne aj tajné) nástroje využívané špecializovanými vládnymi zložkami po celom svete, domnievame sa, že užitočný nástroj tohoto druhu medzi voľne dostupným softvérom chýba. Že o takýto produkt existuje záujem, potvrdzuje aj zadávateľ tejto práce, ktorý v súčasnej situácii nenašiel riešenie, ktoré by vyhovelo jeho požiadavkám.

Medzi najvýznamnejšie nedostatky musíme zaradiť množstvo informácie, ktoré sa o súboroch darí súčasným nástrojom spracovať. Vychádzajúc z odlišnej filozofie, tieto nástroje síce zvyšujú podstatne bezpečnosť prevádzky informačných systémov, neumožňujú však účinnú identifikáciu rôznorodej množiny softvérových súčastí, ktorá je pre proces forenznej analýzy vykonávanej na neznámom zariadení kľúčová. Takáto funkcionalita schopná vytvoriť používateľovi bázu metadát bohatú na informácie taktiež otvára dvere rozšíriteľnosti o zaujímavú novú funkcionalitu, ako napríklad detekcia zraniteľných verzií softvéru, ktorá môže byť zaujímavá nielen pre odvetvie kriminálneho vyšetrovania.

Problémom sa mnohokrát stáva taktiež architektúra nástrojov, množstvo závislostí na softvéri tretích strán a v neposlednom rade prenositeľnosť medzi často používanými softvérovými platformami. Výsledkom sú nástroje, ktoré nie je jednoduché rozšíriť o novú funkcionalitu, ktoré nie je možné jednoducho aplikovať na skúmané zariadenia v najrôznejších konfiguráciách a taktiež, hovoriac o niektorých konkrétnych riešeniach, nie je možné naškálovať do rozsahu desiatok miliónov známych súborov a stoviek tisíc skúmaných súborov.

Medzi súčasnými riešeniami sme taktiež nenašli balík, ktorý by poskytoval podporu pre problém tvorby a automatizovaného udržiavania vlastnej referenčnej bázy metadát. Táto problematika pritom nadobúda veľký význam v snahe budovať alternatívny zdroj informácií založený na špecifickom prostredí a potrebách rôznych organizácií.

Veríme, že kúsky nápadov a prístupov, ktoré sme popísali v predchádzajúcom prehľade, je len treba prehodnotiť a pozorne spojiť do funkčného celku, ktorý bude tvoriť nástroj mocný, všestranný, jednoduchý a efektívny. To, aké nápady budeme spájať a aký celok chceme touto prácou vytvoriť, popisujeme v nasledujúcej kapitole.

Kapitola 2

Ciele práce a návrh riešenia

V predošlých častiach sme stručne popísali problematiku tejto práce a zhodnotili niektoré existujúce riešenia na trhu. Prichádza čas stanoviť problém, akým sa táto práca zaoberá a predstaviť čitateľovi podrobný návrh, ktorým sa ho pokúšame riešiť. V tejto kapitole popíšeme všeobecné dizajnové ale aj detailné funkčné požiadavky na vznikajúce softvérové dielo. Vysvetlíme motiváciu týchto požiadavok a objasníme, prečo si myslíme, že práve táto špecifikácia rieši niektoré nedokonalosti súčasných riešení na trhu. Na konci kapitoly by mal byť čitateľ oboznámený so súčasťami plánovaného softvérového diela a pripravený venovať sa otázkam implementácie a optimalizácie riešenia.

2.1 Všeobecné stanovenie problému

Problém, ktorým sa táto práca zaoberá, je vytvorenie účinného nástroja pre potreby forenzných analytikov ale aj širokej odbornej verejnosti. Úlohou nástroja je poskytnúť prostriedky pre pohodlnú, efektívnu a flexibilnú identifikáciu a kontrolu integrity najmä systémových súborov a programových súčastí softvérových balíkov. Nástroj má taktiež uľahčiť a automatizovať proces tvorby a udržiavania kustomizovanej referenčnej databázy, ktorá je nutná pre úlohu identifikácie a kontroly integrity. Výstupné údaje nástroja majú byť k dispozícii v prehľadnej forme s možnosťou intuitívnej orientácie a ich následného vyhodnotenia. Použitie nástroja má byť všestranné, reflektujúce typické prístupy, ktoré prax vyžaduje. Zároveň je kladený dôraz na jeho výkonnosť a škálovateľnosť, čo ho umožňuje použiť aj v reálnych náročných podmienkach.

Pri tvorbe nástroja sme brali do úvahy konkrétne požiadavky cieľového používateľa. Tieto požiadavky nám prostredníctvom nášho školiteľa poskytol pôvodný zadávateľ problému *CSIRT.SK (Computer Security Incident Response Team Slovakia)*[9]. Zúženie stanoveného problému, ktoré z týchto požiadavok vyplýva, je zameranie vznikajúceho nástroja čiste na kontrolu a identifikáciu najmä systémových súborov platformy *Microsoft Windows*. Stav a záujmy zadávateľskej organizácie sa však v priebehu

vzniku tejto práce dramaticky zmenili. S plánovaným využitím finálneho softvérového diela na strane zadávateľa už nemôžeme počítať. Pôvodné požiadavky však považujeme za veľmi cenný vstup, ktorý nám pomohol lepšie chápať problémy predpokladaného používateľa. Vzhľadom k pokročilej fáze vývoja sme ich aj v novej situácii ponechali bez zmeny. Nasledujúce sekcie sú venované práve návrhu výsledného nástroja vyplývajúceho z požiadavok potenciálnej skupiny používateľov.

2.2 Základné požiadavky a východiská

Prvotnou požiadavkou, od ktorej sa odvíja množstvo dizajnových rozhodnutí pri budovaní softvérového diela, je nutnosť jeho *prenositeľnosti* a úplného fungovania na čo možno najširšej množine dostupných platforiem a ich verzií. Vzhľadom k primárnemu zameraniu nástroja na identifikáciu a kontrolu integrity systémových a aplikačných súborov platformy *Microsoft Windows* je podpora jej rôznych verzií samozrejmou požiadavkou. Menovite je nutné zabezpečiť podporu pre platformy *Windows 7, Windows 8, Windows 8.1 a Windows 10*. Takáto miera podpory pre platformy spoločnosti *Microsoft* už dostatočne vyhovuje predpokladom o typických systémoch, s ktorými v súčasnej dobe prídu do kontaktu nielen pracovníci zadávateľa ale aj mnohých iných organizácií. Veľmi dôležitou pre dané odvetvie je však taktiež možnosť používať budúci nástroj aj na platformách založených na báze *Unixu*. Takáto možnosť môže byť pre analytika užitočná najmä vtedy, ak sa zaoberá analýzou zariadenia potenciálne napadnutého nejakou formou malvéru. Počas snahy odhaliť príčiny problémov a správanie malvéru na operačnom systéme *Windows* sa môže malvéru úspešne dariť maskovať svoju činnosť a brániť sofistikovaným spôsobom tomu, aby sa k nemu analytik dopátral a zdokumentoval ho. Súborový systém potenciálne napadnutého zariadenia analyzovaný nad *distribúciou Linuxu* sa však s veľkou pravdepodobnosťou bude prejavovať omnoho stabilnejšie, keďže škodlivý softvér na viacerých platformách často nebude fungovať správne. Niektorí analytici taktiež preferujú prístup k rôznym atribútom súborového systému pomocou platformy na báze *Unixu* z rôznych subjektívnych dôvodov, napríklad je pre nich tento prístup jednoduchší a časovo úspornejší. Samozrejme, že je v takom prípade žiaduce, aby bol vznikajúci nástroj podporovaný čo najširšou množinou rôznych distribúcií Linuxu nekomplikujúc tak prácu používateľom s odlišnými preferenciami. Zo základu s multiplatformovou podporou možno ale ťažiť aj v budúcnosti. Umožňuje nám to flexibilnejšie adresovať nové požiadavky používateľov a s malým úsilím dotvoriť požadovanú funkcionalitu bez ohľadu na to, či jej myšlienka je v súlade s pôvodnými úvahami o využití nástroja alebo nie. Podpora pre takto široké spektrum platforiem je zároveň medzi súčasnými riešeniami na trhu značne problematická. Požiadavky zadávateľa sú pritom dôkazom toho, že táto charakteristika má svoje opodstatnenie.

Ďalšou kľúčovou požiadavkou s veľkým dopadom na konečný dizajn mnohých súčastí nástroja je jeho *jednoduchá distribuovateľnosť*. Predpokladané spôsoby používania nástroja tejto kategórie sa totiž môžu líšiť pre každého používateľa. Jeden takýto spôsob môže vyzerať napríklad tak, že si používateľ vytvorí *live* spustiteľné médium distribúcie Linuxu s nainštalovaným nástrojom na svojom pamäťovom zariadení a ku každému skúmanému zariadeniu pristupuje prostredníctvom tohto vopred vytvoreného prostredia. Iný používateľ však môže vyžadovať možnosť spustiť tento istý nástroj priamo na skúmanom zariadení jednoducho a s minimálnou potrebnou konfiguráciou. Takto môže byť nástroj využitý na desiatkach rôznych zariadení denne, a to pokojne aj priamo v teréne mimo pracoviska. Pri vývoji je nutné uvažovať tiež potrebu distribuovať nástroj medzi veľa pracovníkov. Aj tento proces je treba udržať nenáročný. Splnenie takýchto požiadavok kladie nároky najmä na množstvo závislostí budúceho nástroja na knižniciach a softvéroch tretích strán, ktoré sa štandardne nevyskytujú na bežných používateľských zariadeniach a je nutné ich doinštalovať. Minimum takýchto závislostí zjednodušuje následné úvahy o distribúcii a prevádzke nástroja, neskôr v práci ale vysvetľujeme, ktorým závislostiam sme sa nevyhli a ako sme daný problém riešili. Množstvo nástrojov spomínaných v sekcii 1.2 nekladie na tieto vlastnosti dostatočný dôraz, a aj preto je ich použitie v popisovaných podmienkach výrazne menej praktické až nerealizovateľné.

Poslednou globálnou požiadavkou, ktorá je kritická pre reálne nasadenie budúceho softvérového diela, je jeho *škálovateľnosť*. Ako sme spomínali v úvode, úplná automatizácia riešenia problému identifikácie a kontroly integrity súborového systému je nesmierne náročná úloha. Spôsob, akým k tomuto problému pristupuje súčasná prax, nesie v určitých ohľadoch známky použitia *hrubej sily*. Len s kvalitnou databázou bohatou na množstvo referenčných záznamov môžno dosiahnuť priaznivé výsledky v miere úspešne identifikovaných súborov. S rastúcou kapacitou pamäťových médií a súčasným zväčšujúcim sa počtom súborov v súborových systémoch zariadení sa dá predpokladať aj rastúca potrebná veľkosť referenčných databáz, s ktorými musí nástroj ako celok fungovať efektívne. To kladie značné nároky na jeho dizajn, ktorý nesmie byť obmedzujúcim faktorom pre objem metadát, s ktorými pracuje.

Globálnou architektonickou črtou, ktorá vyplýva z uvedeného, je celková orientácia nástroja na centralizovanú referenčnú databázu. Riešenia, ktoré uvažujú formu lokálnej databázy, so sebou totiž nesú celý rad obmedzení a komplikácií. Lokálna databáza sa zle udržiava a obohacuje o nové záznamy, neprakticky sa využíva na pracovisku, rýchlosť vyhľadávania v databáze je neoptimálna a škálovateľnosť riešenia na úroveň desiatok miliónov referenčných záznamov nerealistická. Pritom na trhu môžeme vidieť hneď niekoľko nástrojov, ktoré sa na koncept lokálnej databázy spoliehajú. Ako sme však vysvetlili v predošlej kapitole, niektoré tieto nástroje sa otázkou komplexnej identifikácie súborov ani nezaoberajú a svoje zameranie obmedzujú na kontrolu in-

tegrity lokálneho systému. Riešenie s komplexným vybavením pre komunikáciu s centralizovanou referenčnou databázou rieši všetky spomenuté výhrady. Databázu stačí udržiavať na jedinom mieste, prístup k nej vyžaduje len internetové pripojenie a pri vhodnej konfigurácii databázového servera možno dosiahnuť omnoho lepšiu škálovateľnosť a rýchlosť vyhľadávania údajov v porovnaní lokálnym riešením. Zároveň tento prístup ponúka hodnotnú alternatívu k jednoduchším nástrojom, ktoré v súčasnosti na trhu nachádzame. Samozrejme, samotná centralizovaná databáza ako riešenie problému škálovateľnosti nie je postačujúca. Touto otázkou sa ešte podrobne zaoberáme v nasledujúcej kapitole.

2.3 Aplikácie a ich súčasti

V tejto sekcii popisujeme koncepčný návrh všetkých súčastí softvérového diela, ktoré si kladieme za cieľ vytvoriť. Vysvetľujeme ich význam a navrhujeme čitateľovi ich prínos. K popisu nástroja pristupujeme z perspektívy používateľa, pre ktorého sú kľúčové práve jeho vlastnosti a možnosti použitia v porovnaní s jeho konkrétnou implementáciou. Súčasti vznikajúceho nástroja riešia, v snahe adresovať praktické problémy, nasledujúce oblasti. Identifikácia a kontrola integrity systémových súborov, návrh a automatizované budovanie referenčnej databázy, prezentácia a vyhodnotenie výstupnej informácie.

Centrálnou súčasťou vznikajúceho nástroja je terminálová aplikácia, ktorú nazývame *File-Identification-System*. Táto aplikácia vytvorí funkčné jadro celého softvérového diela a predpokladáme, že používatelia s ňou budú prichádzať do kontaktu najčastejšie. Ako napovedá jej názov, *File-Identification-System* implementuje, spoliehajúc sa na ostatné podporné aplikačné súčasti, funkcionalitu spojenú s analýzou a identifikáciou obsahu súborového systému. Poskytne možnosť načítať zvolené súbory v súborovom systéme, vypočítať jednoznačné identifikátory na základe obsahu týchto súborov a riadiť komunikáciu s databázovým systémom za účelom ich identifikácie s použitím získaných metadát. Súbory, ktoré sú predmetom záujmu v danej analýze, môžno určiť definovaním vhodného koreňového adresára pre prehľadávanie súborového systému. Takéto kritérium však môže byť stále príliš všeobecné, preto možno analyzované súbory ďalej ľubovoľne filtrovať podľa ich názvu, a to pomocou regulárnych výrazov. Táto metóda filtrovania síce od používateľa vyžaduje znalosť príslušného štandardu, ktorý formu regulárnych výrazov upravuje, avšak s touto znalosťou u cieľového používateľa počítame. V prípade, že by bol tento predpoklad mylný, aplikácia má byť pripravená navrhnúť mu vhodnú predvolenú konfiguráciu. Použitím regulárnych výrazov však môžeme ponúknuť používateľovi silný vyjadrovací prostriedok a umožniť mu tak zaoberať sa naozaj len súbormi, ktoré sú pre jeho analýzu podstatné. Používateľ

bude mať pritom hneď niekoľko možností, akým spôsobom môže proces analýzy prebehnúť. Problém, ktorému mu chceme pomôcť čeliť, sú situácie, kedy kvalitné internetové pripojenie pre komunikáciu so vzdialeným databázovým systémom nie je k dispozícii. Takáto situácia pritom môže nastať kvôli infraštruktúrnym zlyhaniam, môže byť ale aj špecifikom niektorých pracovísk, v ktorých sa analyzované zariadenia nachádzajú. *File-Identification-System* umožní používateľovi rozdeliť proces analýzy na dve časti. V prípade, že komunikácia s databázovým serverom nie je v danej chvíli uskotočniteľná, aplikáciu možno spustiť v *offline* režime. Na pracovisku bez internetového pripojenia možno zo súborového systému skúmaného zariadenia extrahovať všetky identifikátory súborov potrebné na určenie ich povahy a uložiť ich v súbore s definovaným formátom pre neskoršiu analýzu. Počas priaznivejších podmienok v oblasti pripojenia k databázovému serveru môže používateľ vykonať druhú časť analýzy a vyhodnotiť povahu nájdených súborov. Samotné výstupné údaje budú taktiež v jednotnom dobre spracovateľnom formáte, čo umožní ich vizualizáciu inými nástrojmi. Aplikácia však tiež umožní extrahovať zo súborového systému všetky dostupné relevantné informácie o určených súboroch a vytvoriť pomocou týchto informácií novú referenčnú databázu presne podľa potrieb organizácie. V tomto ohľade je tak zamýšľaná aplikácia kľúčovým dielikom pre splnenie nášho cieľa umožniť oragizáciám pohodlné a jednoduché riešenie tvorby a udržiavania kvalitnej referenčnej databázy, v prípade, že existujúce databázy nezachytávajú dostatok potrebnej informácie.

Touto poznámkou sa dostávame k ďalšej súčasti vznikajúceho nástroja, a síce k jeho referenčnej databáze. Čitateľ je už v tejto chvíli iste oboznámený s tým, ako kriticky vplýva architektúra referenčnej databázy na užitočnosť celého nástroja. V tejto oblasti rozhodne nechceme robiť žiadne kompromisy, preto sme sa rozhodli, že návrh referenčnej databázy vypracujeme sami s ohľadom na konkrétne požiadavky zadávateľa tejto práce. Dôraz bude kladený najmä na množinu užitočných atribútov, ktoré možno o každom súbore zachovať, nízku mieru redundancie dát a optimalizáciu vyhľadávania v databáze. Medzi atribúty bude patriť názov a *absolútna cesta* súboru v súborovom systéme spolu s jeho identifikátorom. Bližšiu identifikáciu súboru umožnia používateľom informácie o spoločnosti a produkte, ktorého je konkrétny súbor súčasťou, taktiež však konkrétne informácie o tom, na akých platformách sa súbor bežne vyskytuje a akú plní úlohu. Čitateľ môže namietať, že množina uchovaných atribútov možno vo výsledku nebude tak obsiahla v porovnaní s *RDS* projektu *NSRL*[5] spomenutého v predošlej kapitole. Musíme si ale uvedomiť, že metadáta obsiahnuté v *RDS* sú zbierané ručne špeciálne vyhradeným oddelením. Naše riešenie je zasa v kontraste s predošlým plne automatizované a nevyžaduje si pri extrakcii metadát ďalšiu intervenciu používateľa. Len všestranne úsporný návrh databázy nám umožní splniť náročné požiadavky na škálovateľnosť a výkonnosť budúceho nástroja.

Samotné budovanie referenčnej databázy však nie je jednorázový akt ale konti-

nuálny, potenciálne nikdy nekončiaci proces. Referenčná databáza musí byť pravidelne obohacovaná o nové aktuálne informácie, inak veľmi rýchlo stratí svoje kvality. K budovaniu referenčnej databázy možno použiť dobre chránené referenčné zariadenia bežiace na platformách, ktorých súčasti sú pre danú organizáciu relevantné. V prípade systémových súborov operačného systému alebo programových súčastí softvérových balíkov môže každá ich aktualizácia priniesť so sebou zmenu súvisiacich súborov, ktorú je nutné zaznamenať. Rodina operačných systémov *Microsoft Windows* je notoricky známa svojím svojvoľným prístupom k aktualizácii systému, ktorá sa vykonáva automaticky, z perspektívy používateľa nedeterministickým spôsobom s ohľadom na aktuálny stav a vyťaženie systému. Automatizácia tohto procesu si vyžaduje kontrolu nad procesom aktualizácie operačného systému aj iných softvérových balíkov. Softvérové balíky, ktorých automatizovaná aktualizácia je problematická, musia byť, žiaľ, aktualizované pravidelne systémovým administrátorom. Súčasťou nástroja, ktorá bude asistovať pri kontrolovanej aktualizácii operačného systému *Microsoft Windows* rôznych verzií je konzolová aplikácia *SysUpdate.exe*. Jej úlohou bude interagovať s operačným systémom a podnietiť aktualizáciu systému bez ohľadu na mechanizmy vstavaných algoritmov systému. Toto riešenie nám umožní vykonávať priebežné dopĺňanie referenčnej databázy o aktuálne dáta predvídateľným a pravidelným spôsobom. Výhodou riešenia je taktiež krátka odozva medzi zaznamenaním nového doposiaľ neznámeho systémové súboru a jeho uložením vo funkčnej referenčnej databáze. Takúto odozvu nedokážu databázy ako *RDS*, aktualizované na štvrťročnej báze, ponúknuť.

To, čo je ale podstatné pre používateľa navrhovaného nástroja na konci dňa, je výstupná informácia, ktorú o skúmaných súboroch vďaka popísanej funkcionalite získa. Tá môže byť vzhľadom k počtu analyzovaných súborov vyčerpávajúco rozsiahla. Dobrá orientácia vo výstupných dátach a možnosť ich jednoduchého vyhodnotenia sú kľúčové charakteristiky, od ktorých závisí využiteľnosť celého nástroja. V tejto oblasti môžeme uvažovať o terminálovej aplikácii, ktorá bude slúžiť na prezentáciu výsledkov používateľovi, prípadne vhodnom uložení výstupu do súboru. Pri dátach, ktoré pozostávajú z dlhých reťazcov znakov, ako sú napríklad absolútne cesty k súborom v súborovom systéme, a ktorých počet môže potenciálne narásť až na úroveň stoviek tisíc, nám ale obe tieto riešenia prídu dosť neprehľadné. Napokon sme sa rozhodli prezentovať používateľovi výstupné dáta prostredníctvom webovej aplikácie. Táto aplikácia nám umožní implementovať omnoho prehľadnejšie a prívetivejšie prostredie, v ktorom bude mať používateľ možnosť intuitívne filtrovať výstup na základe rôznych kritérií a taktiež zhodnotiť jednoducho jeho globálne charakteristiky. Webová aplikácia ale nebude slúžiť len pre účel prezentácie výstupných dát. Bude tvoriť domovské *online* prostredie celého nástroja, ktoré bude môcť používateľ využiť aj inými spôsobmi. Ten sa z času na čas môže napríklad ocitnúť v nepríjemnej situácii, kedy by potreboval využiť poznatky uschované v referenčnej databáze, no *File-Identification-System* momentálne nemá k

dispozícii. Webová aplikácia má v takomto prípade veľký potenciál pomôcť používateľovi riešiť jeho komplikácie. Preto bude okrem prezentácie výsledkov webová aplikácia taktiež schopná identifikovať používateľom zvolený načítaný súbor pomocou referenčnej databázy na základe identifikátora obsahu, ktorý sama vypočíta. Rovnako bude schopná identifikovať aj zoznam súborov, ktorých identifikátory si mohol používateľ predpočítať v *offline* režime konzolovej aplikácie.

Popísanou funkcionalitou chceme používateľovi dopriať maximálnu flexibilitu pri použití budúceho nástroja pre identifikáciu a kontrolu integrity súborov. Množstvo alternatívnych spôsobov, akým možno už v tomto návrhu pristúpiť k dátam v referenčnej databáze, však nie je výhodné len pre používateľa, ktorý z nich môže ťažiť. V prípade výskytu nových doposiaľ nespomenutých používateľských požiadavok ťaží z týchto alternatív aj vývojár, ktorý má na výber hneď niekoľko stabilných platforiem, pomocou ktorých môže vyhovieť požiadavkám a doimplementovať optimálne a intuitívne riešenia. Práve implementácia a návrhu, ktorý sme popísali v tomto texte a s ňou spojené výzvy, sú predmetom nasledujúcej kapitoly.

Kapitola 3

Metodika a priebeh práce

V predošlých kapitolách sme čitateľovi bližšie predstavili bežnú pracovnú prax niektorých forenzných analytikov a problémy, ktoré v nej vznikajú. Taktiež sme koncepčne navrhli nástroj, ktorý by mohol byť nápomocný pri riešení týchto problémov. Bez korektnej implemntácie nie je však táto práca ničím viac, než nápadom na pracovnom stole jej oponenta. V tejto kapitole popíšeme priebeh implementácie vznikajúceho nástroja. Pozastavíme sa pri významných úvahách a dizajnových rozhodnutiach, ktoré určili konečné schopnosti nástroja. Budeme taktiež diskutovať o rôznych alternatívach, z ktorých sme si pri týchto rozhodnutiach vyberali.

3.1 Základné vývojárske princípy

Berúc do úvahy konkrétne požiadavky popísané v predošlej kapitole, musíme si v záujme ich splnenia definovať niektoré vývojárske zásady, ktorých sa budeme držať počas celého procesu vzniku navrhovaného nástroja. Kľúčovou požiadavkou smerodatnou pre ďalší vývoj sa ukázala byť prenositeľnosť nástroja medzi širokým spektrom rôznych platforiem a operačných systémov. Samotná multiplatformová kompatibilita softvéru nám však nemusí priniesť dostatočne presvedčivé praktické výsledky. Jednoduchá použiteľnosť nástroja vyžadujúca čo najminimalistickejšie nutné zmeny systému zo strany používateľa nás núti brať pri návrhu do úvahy aj možné líšiace sa konkrétne inštalácie každého individuálneho zariadenia, na ktorom má byť nástroj použitý. Podpora pre staršie platformy nám nedovoľuje očakávať, že cieľové zariadenia všeobecne disponujú najnovšími verziami požadovaných knižníc a softvérových balíkov. Posledné, čo potrebuje používateľ riešiť v snahe vykonávať svoju prácu, sú problémy so spätnou kompatibilitou nášho nástroja, prípadne nedostupnosťou niektorých jeho nutných softvérových závislostí pre jeho konkrétne zariadenie. Pre obmedzenie týchto nepríjemných používateľských skúseností sme pri vývoji kládli dôraz na vhodný výber technológií, ktoré poskytujú okrem vyspelosti aj stabilitu a rozumnú spätnú kompatibilitu. Snažili

sme sa minimalizovať množstvo závislostí na softvéri tretích strán na nutné minimum a vyhnúť sa zbytočnému používaniu neštandardných, ikeď možno pre programátora pohodlnejších, knižníc, ktoré nie sú všeobecne súčasťou bežnej inštalácie operačného systému a rušili by kompaktnosť budúcej aplikácie.

S týmito myšlienkami na zreteli sme pristúpili k ich realizácii. Prvotná je voľba programovacieho jazyka, v ktorom bude implementovaná funkcionalita konzolových aplikácií. Tieto tvoria funkčné jadro celého nástroja, preto od tejto voľby značne závisí jeho celková výkonnosť. V prostredí súčasných softvérových požiadavok je myšlienka platformovej nezávislosti vlastná mnohým programovacím jazykom a ich zodpovedajúcu implementáciu interpretera alebo kompilátora nájdeme dostupnú prakticky pre všetky využívané operačné systémy a ich verzie. Toto kritérium nás teda neobmedzuje a kritické bude až počas samotnej implementácie. Pretože však navrhovaný nástroj dimenzujeme na prácu s potenciálne veľkými objemami dát, pre používateľa je určite dôležitá jeho celková výkonnosť a časová efektivita. Na zabezpečenie týchto vlastností musíme dbať od úplného začiatku vývoja. Hlavnou motiváciou pre použitie programovacieho jazyka *C++* je práve rýchlosť vykonávania natívneho kódu, ktorý vznikne jeho kompiláciou. Hoci rýchlosť vykonávania považujeme za kľúčovú, dôležité boli pre nás aj iné vlastnosti jazyka, ako sú napríklad udržiavateľnosť výsledného kódu, možnosť modularizácie kódu a množstvo podpory zo strany štandardnej knižnice, od ktorej závisí najmä rýchlosť implementácie riešenia, taktiež však potenciálna chybovosť počas vývoja a čitateľnosť výsledného kódu. V týchto parametroch sa nám programovací jazyk *C++* javí ako vynikajúci kompromis v porovnaní s jazykom *C*, ktorý pri vhodnej implementácii dokáže poskytnúť ešte vyššiu rýchlosť vykonávania kódu, avšak na úkor ostatných významných vlastností. Použitím jazyka *C++* taktiež nestrácame žiadnu z možností programovania na nízkej úrovni. Tú pritom využijeme pri množstve systémových volaní špecifických pre konkrétne operačné systémy, ktorých použitie si povaha nášho nástroja vyžaduje. S použitím niektorých iných programovacích jazykov, ako je napríklad *Java*, by sme boli nútení použiť pre túto funkcionalitu špecializované *frameworky*[1]. Významné benefity so sebou nesie použitie jazyka *C++* aj v oblasti prirodzeného vykonávania skompilovaného a *staticky linkovaného* binárneho kódu priamo na cieľovej platforme, a to bez nutnosti inštalovať akýkoľvek ďalší softvér pre vytvorenie prostredia, v ktorom sa bude aplikácia vykonávať. Táto téma už súvisí priamo s motiváciou pre vývojárske princípy definované v úvode tejto sekcie a bližšie ju vysvetľujeme v sekcii 3.3.

Po výbere vhodného programovacieho jazyka sme realizovali vývojárske princípy z úvodu sekcie najmä tým, že sme striktne obmedzili programátorske knižnice, na ktorých budú naše *C++* aplikácie závisieť. Funkcionalitu sme implementovali s použitím *C++ STL (Standard Template Library)*. Táto rozsiahla štandardná knižnica je s malými odlišnosťami korektne implementovaná väčšinou súčasných *C++* kompilátorov.

Riešenie teda nie je závislé na použitej platforme operačného systému, na výber máme taktiež niekoľko alternatív kvalitných kompilátorov. Štandardná knižnica je však priebežne vyvíjaná a upravovaná príslušnými štandardmi. Použitie funkcionality knižnice definovanej v najnovšom štandarde by mohlo spôsobiť problémy pri kompilácii nástroja na starších platformách. Usúdili sme, že kompilátory konformné so štandardom *C++11* sú v súčasnosti dostatočne dostupné na to, aby sme mohli využiť výhody knižnice definovanej týmto štandardom pre uľahčenie vývoja, zníženie náchylnosti na chyby a zvýšenie čitateľnosti výsledného kódu. Funkcionalitu definovanú štandardmi *C++14, C++17* a *C++20* sme z dôvodu spätnej kompatibility pre staršie platformy zámerne nepoužili. Pri použití systémových knižníc operačného systému sme sa obmedzili na využitie funkcionality definovanej všeobecne uznávaným štandardom *POSIX*. Spektrum platforiem podrobujúcich sa tomuto štandardu je dostatočne široké na to, aby splnilo naše náročné požiadavky na prenositeľnosť systému. Závislosti na iných externých knižniciach sme minimalizovali na nutné minimum. Pri implementácii niektorých súčastí konzolových aplikácií sme sa však takýmto závislostiam nevyhli. To, z akých súčastí sa skladajú jednotlivé konzolové aplikácie a ako sme v prípade ich tvorby postupovali, popisujeme v nasledujúcej sekcii.

3.2 Implementácia konzolových aplikácií

Počas celého vývoja vznikajúceho nástroja sme preferovali modulárny dizajn aplikácií. Tento prístup implementovaný od samotného začiatku vývoja so sebou nesie niekoľko významných výhod. Jednotlivé nesúvisiace súčasti kódu sú od seba už prvotným návrhom funkčne oddelené, čo nám efektívne zabráni v tom, aby sme medzi týmito súčasťami vytvárali v procese tvorby nezmyselné závislosti, ktoré nám skomplikujú ďalší vývoj aplikácie. Z dlhodobého hľadiska pomôže tento prístup jednoduchému pridávaniu a testovaniu novej funkcionality bez toho, aby bolo nutné meniť ostatné súčasti aplikácie. Práve naopak, modularizácia návrhu nám umožnila v neskorších fázach vývoja vytvárať nad existujúcimi súčasťami nové vrstvy abstrakcie kódu, s ktorými sme pôvodne nepočítali, a ktoré bez akéhokoľvek vplyvu neskorších úprav základných súčastí implementujú funkcionalitu spojenú s ich paralelným vykonávaním, prípadne behom aplikácie v odlišných používateľských režimoch. V nasledujúcich častiach sa venujeme elementárnym súčastiam aplikácie *File-Identification-System*. Implementácia kompaktnejšej aplikácie *SysUpdate.exe* kopíruje prístupy a úvahy, ktoré popisujeme v ďalšom texte a v tejto práci sa jej nebudeme podrobnejšie venovať.

3.2.1 Vstupný komponent

Úlohou vstupného komponentu aplikácie *File-Identification-System* je implementácia metód načítania a filtrovania vstupných dát pred ich ďalším spracovaním. Tieto metódy načítania sú menovite načítanie dát zo vstupného súboru vytvoreného v *offline* režime aplikácie (trieda *Input::InputFile*) a načítanie dát rekurzívnou iteráciou cez objekty v súborovom systéme s používateľom definovaným koreňom prehľadávania (trieda *Input::InputScanner*). Zatiaľ čo realizácia prvého zo spomínaných je po určení vhodného formátu súboru pomerne priamočiara, prehľadávanie súborového systému si vyžadovalo úvahy o tom, ako túto funkcionalitu vytvoriť. Programátorsky najjednoduchším riešením by bolo využitie niektorej z dostupných knižníc, ktoré implementujú rôzne metódy práce so súborovým systémom, okrem iného aj rekurzívny iterátor jeho objektov. Potenciálnymi kandidátmi by bola súčasť štandardnej knižnice *C++ STL std::filesystem* alebo neštandardná *boost::filesystem*. Funkcionalita *C++ STL* bola obohatená o prácu so súborovým systémom až s príchodom štandardu *C++17*. Nutná prítomnosť kompilátora, ktorý takto nový štandard korektne implementuje, môže byť najmä v starších systémoch problematická, čo môže spôsobovať zbytočné komplikácie v procese kompilácie nástroja. Použitie knižnice *boost::filesystem* by podľa nášho uváženia nebolo zlým rozhodnutím. Rodina *C++* knižníc *boost* si zakladá na svojej implementácii pre drvivú väčšinu existujúcich platforiem a systémov. Hoci toto riešenie by nám poskytlo pohodlie pri jednoduchom použití existujúceho rekurzívneho iterátora súborového systému, nevýhodný nám prišiel fakt, že by sme takýmto spôsobom zaviedli do nástroja závislosť na relatívne rozsiahlej knižnici tretej strany pre použitie jedinej jej metódy. Alternatívou k tomuto riešeniu by pritom mohla byť kompaktná knižnica *dirent.h*, ktorej prítomnosť v systéme je garantovaná a upravená štandardom *POSIX*. Tá umožňuje pristúpiť a iterovať cez objekty zvoleného adresára v súborovom systéme, pričom štandard garantuje možnosť obdržať názvy týchto objektov. Existuje taktiež obdobná implementácia *dirent.h* pre platformy *Microsoft Windows*, ktorú by sme v prípade potreby mohli použiť ako súčasť nášho nástroja. Práve kompaktnosť tejto knižnice nám v kontexte naších skutočných potrieb imponovala natoľko, že sme si ju vybrali pre tvorbu komponentu. Potrebný rekurzívny iterátor objektov súborového systému sme s pomocou tejto knižnice implementovali vlastnoručne, vďaka čomu sme sa vyhli zbytočnej závislosti na rozsiahlejšej knižnici, ktorej funkcionalitu naša aplikácia nevyužije.

3.2.2 Komponent pre určenie identifikátora

Úlohou tohto komponentu je priradiť každému súboru jednoznačný identifikátor na základe jeho obsahu. Takýto problém možno elegantne riešiť pomocou *hashovacej funkcie* s vhodnými vlastnosťami, ktorá obsahu súboru priradí digitálny odtlačok alebo aj *hash*. Vo všeobecnosti možno tieto vlastnosti popísať tromi požiadavkami. Prvou

požiadavkou je, aby bolo pomocou funkcie možné výpočtovo nenáročne a rýchlo určiť na základe digitálnej informácie jej digitálny odtlačok fixnej dĺžky. Druhou požiadavkou je, aby bolo prakticky výpočtovo nemožné odvodiť z tohto digitálneho odtlačku akúkoľvek informáciu o pôvodnom digitálnom vstupe funkcie. Treťou požiadavkou je, aby bolo prakticky výpočtovo nemožné určiť pre zadaný vstup funkcie odlišný vstup, ktorému funkcia priradí rovnaký digitálny odtlačok. Odlišné vstupy, ktorým je *hashovacou funkciou* priradený rovnaký digitálny odtlačok budeme nazývať *kolízie*. Keďže vstupom *hashovacej funkcie* je nekonečné množstvo odlišných reťazcov a obor hodnôt funkcie je konečná množina reťazcov fixnej dĺžky, kolízie medzi možnými vstupmi funkcie budú vždy existovať. V skutočnosti platí, že sa k takejto kolízii *hashovacej funkcie* možno dopočítať v konečnom čase, upresníme však, že pre praktické použitie je postačujúca funkcia, pre ktorú dopočítanie požadovanej kolízie v únosnom časovom intervale, napríklad počas dĺžky ľudského života, nie je reálne. Dôležité pre naše použitie *hashovacej funkcie* je taktiež pripomenúť, že identifikátory, ktoré sú ňou súborom priradené, nemôžu byť z dôvodu výskytu kolízií jednoznačné. Pre praktické použitie je veľmi kritická nízka miera výskytu kolízií pre bežné vstupy funkcie. Toto kritérium kladie isté nároky na rovnomerné rozdelenie nekonečného množstva vstupov *hashovacej funkcie* medzi prvky jej konečného oboru hodnôt. Hoci svojím spôsobom vyplýva z predošlých požiadavok, považujeme za užitočné ho čitateľovi pripomenúť, keďže táto miera má veľký praktický vplyv na počet kolízii vyskytujúcich sa v budúcej referenčnej databáze.

Typickým kandidátom v tejto oblasti je už tradične *hashovacia funkcia MD5*. Rokmi používania overené praktické skúsenosti potvrdzujú priaznivé vlastnosti tejto funkcie. Proces výpočtu digitálneho odtlačku nie je výpočtovo náročný a miera výskytu kolízií vzniknutých digitálnych odtlačkov vyhovovala jej praktickému použitiu aj v oblasti informačnej bezpečnosti. Avšak s pribúdajúcimi výsledkami výskumov, v ktorých sa podarilo v časovom rozpätí jednotiek hodín dopočítať kolízie k požadovaným vstupom funkcie, sa súčasná prax začína od jej použitia odvracať k robustnejším riešeniam[11].

My sme sa v snahe poskytnúť používateľovi moderné riešenie s dôrazom na bezpečnosť rozhodli následovať tento trend. Predvolená *hashovacia funkcia*, ktorú sme použili v komponente pre určenie identifikátora je omnoho robustnejšia *SHA-256* (trieda *HashAlgorithm::SHA2*). Určenie identifikátora touto funkciou je podstatne výpočtovo náročnejšie v porovnaní s *MD5*, je to však výzva, ktorej sme sa v záujme ponúknutia vysokej miery informačnej bezpečnosti nevyhli a budeme jej čeliť inými optimalizáciami nášho nástroja. Zároveň si uvedomujeme premenlivosť situácie na poli informačnej bezpečnosti a rovnako tak odlišné požiadavky rôznych používateľov. Preto sme pre realizáciu tohto komponentu použili kryptografickú knižnicu *cryptopp*, ktorá implementuje ešte množstvo iných alternatív pre vhodnú *hashovaciu funkciu* a naviac možno predpokladať, že bude v budúcnosti stále ponúkať aktuálne riešenia vyhovujúce

praxi. Knižnica je taktiež k dispozícii na všetkých platformách, pre ktoré náš nástroj navrhujeme. Použitie tejto knižnice je síce nutnou závislosťou nášho nástroja, no v súčinnosti s tým, že sme funkcionalitu pre určenie identifikátora obsahu súboru koncipovali ako samostatný komponent, nám umožní promptne reagovať na meniace sa požiadavky našich používateľov.

3.2.3 Výstupný komponent

Úlohou tretieho veľkého komponentu aplikácie je získanie a formátovanie výstupných dát, a to hneď tromi metódami. Najjednoduchšou z nich je výstup aplikácie v *offline* režime (trieda *Output::OutputOffline*), ktorý používateľovi poskytne zoznam súborov nájdených vstupným komponentom a im zodpovedajúcich jednoznačných identifikátorov pre ich neskoršiu analýzu. Ďalšie dve metódy výstupu už pri svojej činnosti aktívne komunikujú s databázovým serverom. Prvá z nich využíva referenčnú databázu pre identifikáciu, prípadne kontrolu integrity objektov súborového systému na základe ich absolútnej cesty a názvu a ich jednoznačného identifikátora (trieda *Output::OutputValidateDB*). Používateľ je informovaný o najdených valídnych súboroch, zároveň je však upozornený na neznáme súbory, súbory so známym obsahom, ktorých názov alebo pozícia v súborovom systéme je odlišná od pôvodne zaznamenanej a podozrivé súbory, ktorých názov a pozícia je známa, no ich obsah je evidentne odlišný od toho zaznamenaného v referenčnej databáze. Získané informácie sú priebežne ukladané do výstupného súboru vo formáte *CSV* pre ich jednoduché ďalšie čítanie a vizualizáciu. Druhá metóda výstupu, ktorá pre svoju činnosť využíva komunikáciu s databázovým serverom, implementuje funkcionalitu pre extrakciu maximálneho dostupného množstva atribútov, ktoré o daných súboroch v súborovom systéme existujú (trieda *OutputUpdateDB*). Získané informácie sú potom vložené do referenčnej databázy. Táto metóda výstupu teda slúži na vybudovanie referenčnej databázy a v súčasnosti je implementovaná výhradne pre platformy spoločnosti *Microsoft Windows*. Toto obmedzenie súvisí najmä s pôvodnými požiadavkami zadávateľa tejto práce, ktorého záujmom bolo udržiavať referenčnú databázu systémových súborov a programových súčastí softvérových balíkov rôznych verzií operačných systémov *Windows*. Vďaka vhodnému modulárnemu dizajnu aplikácie však v budúcnosti nič nebráni vytvoreniu obdobnej funkcionality aj pre platformy na báze *Unixu*.

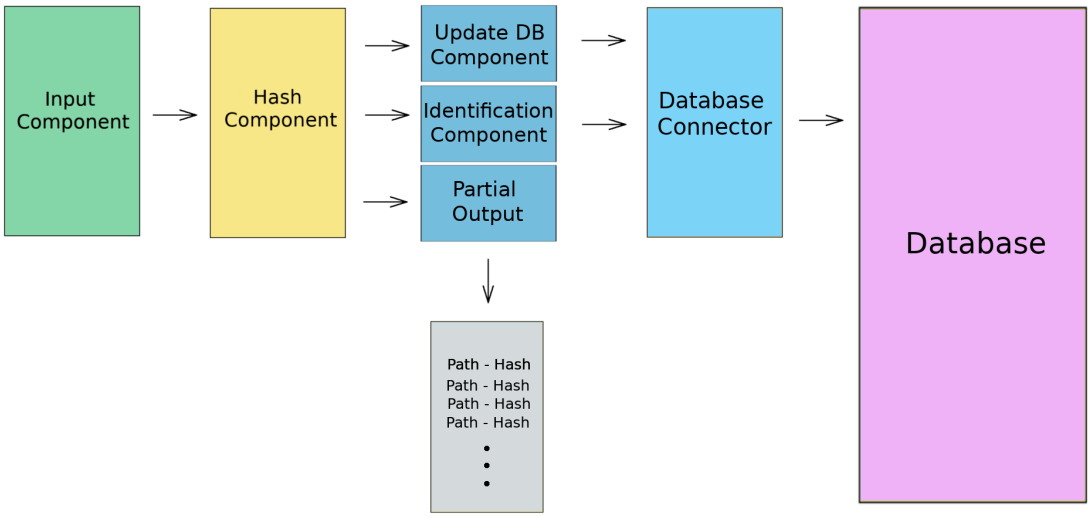
Dôležitým rozhodnutím pri tvorbe výstupného komponentu bol výber ďalšej závislosti našej aplikácie, knižnice, prostredníctvom ktorej sa bude komponent pripájať k databázovému serveru. Od knižnice sme požadovali podporu predpripravených požiadavok (*prepared statements*), ktoré umožňujú podstatné zefektívnenie vykonávania veľkého množstva rovnakých požiadavok s odlišnými parametrami. Významná bola pre nás taktiež podpora aplikácie pre čo najširšiu množinu databázových serverov a

ich verzií. K dispozícii sme mali 3 možné alternatívy. Prvé dve z nich boli od spoločnosti *Oracle*, a síce *MySQL Connector/C++*[4] a *MySQL C API 5.5*. Programátorské rozhranie *MySQL Connector/C++* je vyspelé objektovo orientované programátorské rozhranie pre komunikáciu výhradne s *MySQL* servermi. V súčasnosti je štandardnou voľbou vývojárov. Aplikácia používajúca toto rozhranie dokáže komunikovať s databázovými servermi verzie *MySQL 5.7* a vyššie. Jeho nevýhodou je jeho nekompatibilita so staršími verziami databázových serverov *MySQL*. S tými dokázal pracovať starší produkt *MySQL C API 5.5*, ten však už nie je udržiavaný a jeho použitie sa neodporúča. V čase ukončovania tejto práce už *MySQL C API 5.5* nebol oficiálne k dispozícii. Riešenie, pri použití ktorého by sme pre podporu starších verzií databázových serverov potrebovali dve odlišné implementácie komponentov našej aplikácie, bolo pre nás neprijateľné. Naviac, novší produkt *MySQL Connector/C++* nijako nepodporuje komunikáciu so systémom *MariaDB*, ktorý sa stal štandardom v oblasti voľne dostupných databázových riešení. Preto sme sa napokon radi rozhodli pre tretiu alternatívu, ktorou je *MariaDBConnector/C*[18]. Táto knižnica vychádza priamo z pôvodného *MySQL C API 5.5*, nikdy však neprestala byť kontinuálne vyvíjaná. Nevýhodou tejto knižnice môže byť v porovnaní s novšími riešeniami zastaraný dizajn jej programátorského rozhrania, ktorý neponúka možnosť dostatočne prehľadného kódu, a ktorého sa projekt *MySQL* už v minulosti zbavil. Výhodou je však možnosť jednotnej implementácie komunikácie so všetkými verziami databázových serverov *MariaDB* a taktiež databázovými servermi verzie *MySQL 5.5* a vyššie. Vďaka takto všestrannej funkcionalite, ktorú nám *MariaDBConnector/C* ponúka, sme mohli už v prvotnej verzii našej aplikácie jedným modulom podporovať dva z najčastejšie používaných databázových serverov a dať tak na výber naším potenciálnym používateľom.

3.2.4 Paralelné spracovanie informácie

V doterajších sekciách sme sa zaoberali najmä takými aspektami implementácie aplikácie, ktoré zaručia jej kompaktnosť a kompatibilitu s okolitým systémovým prostredím jej používateľov, prípadne jej jednoduché udržiavanie a ďalší vývoj. Výpočtová náročnosť priradzovania identifikátorov súborom spolu s predpokladaným veľkým objemom spracovávaných dát nás ale počas vývoja prinútila zamyslieť sa nad otázkami výkonnosti a škálovateľnosti softvérového riešenia, ktoré tvoríme. Ako vysvetľujeme v ďalšom texte, architektúra riešenia, ktorú sme do tohto momentu v práci popisovali (obrázok 3.1), nesie so sebou hlavne v oblasti výkonnosti ešte značný priestor pre zlepšenie.

Aplikácia počas svojho behu vykonáva dva význačné druhy výpočtových operácií. Zatiaľ čo iterácia objektov súborového systému a najmä čítanie ich obsahu je *náročné na vstupno-výstupné operácie* a vyžaduje si relatívne málo procesorového času, samotný proces *hashovania* je *náročný na operácie procesora*, no nevyužíva pritom vstupno-



Obr. 3.1: Sekvenčný návrh architektúry *File-Identification-System*

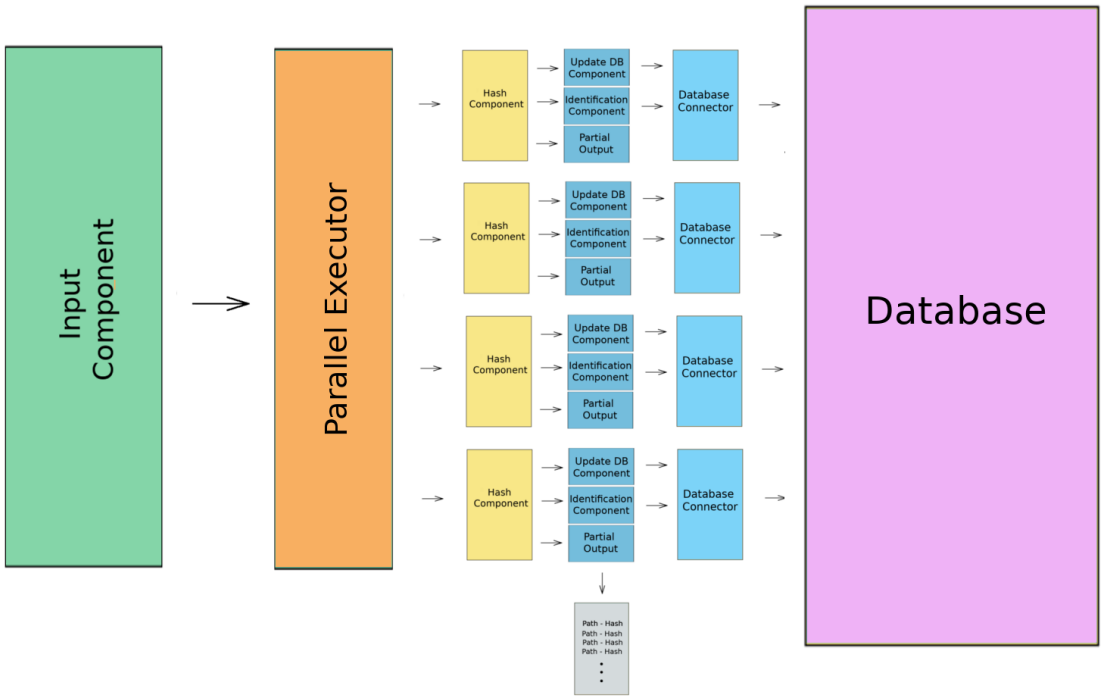
výstupné zariadenia. Možno triviálne nahliadnuť, že pri doterajšej architektúre počas vykonávania vstupno-výstupných operácií zostáva procesor zariadenia nevyužitý čakať na ich skončenie, počas vykonávania operácií náročných na procesor zostávajú naopak nevyužité vstupno-výstupné zariadenia. Konkurentný prístup k referenčnej databáze, ktorý dokáže ponúknuť databázový server, naša aplikácia taktiež nevyužíva. Celkové neoptimálne využitie prostriedkov procesora počas behu aplikácie je ilustrované na obrázku 4.1. Zvýšenie efektivity vykonávania aplikácie by mohla priniesť vhodná *paralelizácia* jej behu (trieda *ParallelExecutor*).

Nie všetky komponenty aplikácie však možno pohodlne paralelizovať. Paralelizácia vstupného komponentu, ktorý rekurzívne iteruje zvolený podstrom súborového systému, nemôže vo všeobecnosti fungovať efektívne bez toho, aby sme o jeho štruktúre získali od používateľa dodatočnú informáciu. Jedine s takouto informáciou o hĺbke a počte jednotlivých podstromov iterovaného súborového priestoru by sme mohli urobiť vedomé rozdelenie skúmaného podstromu medzi paralelné inštancie triedy *Input::InputScanner*. Toto implementačne náročné riešenie by pritom podľa našich úvah neprinieslo žiadne viditeľné zlepšenie výkonnosti, operácie vykonávané touto triedou sú výpočtovo jednoduché. Preto vstupný komponent našej aplikácie nebude vykonávaný paralelne a bude dodávať vstupné dáta rovnomerne inštanciám ostatných komponentov na ďalšie spracovanie. Paralelizácia komponentu pre určenie určenie jednoznačného identifikátora a výstupného komponentu je už omnoho prirodzenejšia. Ani jeden z týchto komponentov síce nemôže správne pracovať, ak ponúka ako jediná inštancia svoju funkcionalitu viacerým paralelným vláknam súčasne (tieto komponenty nemožno popísať ako *thread-safe*), ich paralelizácia v prípade, že každé vlákno disponuje svojou vlastnou inštanciou komponentu je však v tomto prípade možná. V zahraničnej literatúre bývajú takéto

komponenty popísané ako *viacnásobne prístupné (reentrant)*). Viacnásobná prístupnosť komponentu pre určenie identifikátora je po pohľade na jeho konkrétnu implementáciu zrejmá. Paralelné vykonávanie tohto komponentu umožní našej aplikácii vyťažiť, v závislosti na voľbe používateľa, niekoľko alebo aj všetky fyzické jadrá procesora. Zároveň sa často môže stať, že zatiaľ čo niektoré inštancie práve vykonávajú operáciu náročnú na procesorový čas, iné inštancie práve využívajú vstupno-výstupné zariadenia na čítanie súboru. Pre dosiahnutie viacnásobnej prístupnosti výstupného komponentu už ale musíme riešiť niekoľko problémov. Samotná komunikácia s databázovým systémom nie je problém. Počas tohto procesu má naša aplikácia v oblasti konkurentného čítania podporu na strane databázového servera, čo umožňuje rýchlejší prístup požadovaným dátam. Komplikovaná sa ale ukazuje byť fáza, kedy sú získané dáta ukladané do výstupného súboru. Ak sa chceme vyhnúť tomu, aby každé vlákno muselo zapisovať do osobitného súboru, musí sa aplikácia v bode zápisu *synchronizovať*. Avšak aj v prípade, že viacero inštancií zapisuje do súboru pomocou svojho otvoreného prúdu pre zapisovanie synchronizovane, na úrovni operačného systému neexistuje žiadny mechanizmus, ktorý by synchronizoval tieto zápisy pomocou viacerých otvorených prúdov, dáta sú tak aj naďalej vpísané konkurentne. Obsah výstupného súboru by teda aj napriek synchronizácii na úrovni aplikácie ostával nedeterministický. Riešením problému je napokon až to, že všetky inštancie, ktoré potrebujú pre svoju činnosť zapisovať do výstupného súboru, budú zdieľať spoločný otvorený prúd pre zápis do súboru (reprezentovaný triedou *Output::OutputOffline*) a vo svojích zápisoch sa budú synchronizovať.

Návrh architektúry, ktorý vzišiel z týchto úvah, uvádzame v na obrázku 3.2. Otázkou pri jeho realizácii ale ostávalo, ako implementovať rovnomernú dodávku dát zo vstupného komponentu zvyšku paralelne vykonávaných komponentov. Prvou alternatívou by mohla byť komunikácia jednotlivých vlákien so vstupným komponentom pomocou sieťových *socketov* s použitím modelu *request-reply*. Bližší prieskum ale ukázal ťažkopádnosť tohto riešenia, ktoré by bolo nemožné realizovať nezávisle na použitej platforme bez použitia externej knižnice pre sieťovú komunikáciu, ako je *0MQ*[14]. Omnoho elegantnejšie a kompaktnejšie sa ukázalo byť použitie dátovej štruktúry *fronta* zo štandardnej knižnice *STL*. Túto frontu sme doplnili o funkcionalitu súvisiacu so synchronizovaným vkladaním a vyberaním prvkov a použili ju ako priebežne dopĺňaný zásobník vstupných dát pre ich paralelné spracovanie.

Následné experimenty potvrdili vhodný prístup k paralelizácii aplikácie. Počas experimentov sme testovali beh aplikácie s rôznym počtom paralelných vlákien vykonávania. Aplikáciu sme použili v režime, ktorý získaval vstupnú informáciu priamo iteráciou objektov v súborovom systéme a overoval povahu súborov pomocou referenčnej databázy obsahujúcej približne *45 000* záznamov. Počet overovaných súborov bol približne *200 000* a ich veľkosť sa pohybovala v rozmedzí od jednotiek *bajtov* po jednotky *megabajtov*. Viditeľne účinnejšie vyťaženie procesora možno pozorovať aj na obrázku 4.2.



Obr. 3.2: Paralelný návrh architektúry *File-Identification-System*

Čas (s) (1 vlákno)	Čas (s) (4 vlákna)	Čas (s) (8 vlákien)	Čas (s) (12 vlákien)
404	199	193	173
398	201	198	179

Tabuľka 3.1: Časy potrebné pre vykonanie úkonu s rôznym počtom paralelných vlákien

Poznatkom získaným z experimentov je najmä optimálny predvolený počet paralelných vlákien vykonávania, pri ktorom aplikácia dosiahne maximálnu výkonnosť. Vzhľadom k špecifickým vlastnostiam behu v jednotlivých fázach, kedy sú vyťažené ako procesor, tak aj vstupno-výstupné zariadenia, optimálna hodnota počtu paralelných vlákien sa ustáľuje na dvojnásobku počtu fyzických jadier procesora. V závislosti od veľkosti skúmaných súborov sa ale môže optimálna hodnota meniť. Ako možno pozorovať v v tabuľke 3.1, pre súbory, ktorých veľkosť nepresiahla jednotky *megabajtov* je optimálna hodnota počtu paralelných vlákien *12*. Vo všeobecnosti platí, že hodnota na úrovni počtu fyzických jadier procesora nedokáže kvôli významnému podielu vstupno-výstupných operácií počas behu aplikácie dostatočne vyťažiť procesor zariadenia. V optimálnej konfigurácii je aplikácia podľa našich experimentov schopná dosiahnuť až *56%* skrátenie času, ktorý potrebuje na overenie fixných súborov v súborovom systéme v porovnaní s behom v sekvenčnom režime. Toto vylepšenie považujeme za vynikajúci výsledok, ktorý je brilantnou odpoveďou na obavy z praktickej použiteľnosti našej aplikácie pre veľké dáta.

3.3 Kompilácia a inštalácia konzolových aplikácií

Pri kompilácii konzolových aplikácií pre platformu *Microsoft Windows* musíme počítať s tým, že niektorí používatelia potrebujú aplikáciu použiť denne pokojne aj na desiatkach odlišných skúmaných zariadení. Je teda nutné zabezpečiť jednoduchú distribúciu a inštaláciu aplikácie. Každý úkon, ktorý je nutné pri inštalácii vykonať, zbytočne plytvá časom používateľa. Ideálnym prípadom by bol stav, kedy by používateľ jednoducho skopíroval spustiteľný binárny súbor aplikácie z prenosného pamäťového zariadenia do súborového systému skúmaného zariadenia a bez akejkoľvek inštalácie doplňujúcich balíkov mohol aplikáciu použiť. Takýto stav možno docieliť *statickým linkovaním* aplikácie a jej závislostí. Všetky externé závislosti aplikácie potrebné pre jej beh sú v takomto procese linkovania pripojené k aplikačnému súborovému systému. Cenou za aplikáciu schopnú bežať samostatne bez predpokladov kladených na inštaláciu operačného systému je veľkosť, do akej môže binárny súbor aplikácie narásť v prípade, že sa pri svojej činnosti spolieha na množstvo závislostí. Aj to je dôvod, prečo sme sa počet externých knižníc, na ktorých naše aplikácie závisia, snažili od počiatku minimalizovať. Pre používateľov pracujúcich s distribúciou Linuxu by použitie statického linkovania závislostí zatiaľ nemalo význam, keďže na skúmanie zariadení s touto platformou sme náš nástroj nezamerali a samotný priebeh jeho kompilácie je na tejto platforme omnoho používateľsky jednoduchší. Úplné statické linkovanie aplikácie so závislosťami sa nám však pre komplikácie a obmedzenosť času nepodarilo. Súbor pre *dynamické linkovanie* knižnice *MariaDBConnector/C* počas behu aplikácie teda musíme distribuovať spolu so spustiteľným aplikačným súborom. Túto nedokonalosť budeme musieť vyriešiť v budúcnosti.

3.4 Implementácia návrhu databázy

Ako sme uviedli v predchádzajúcej kapitole, v súčasťou riešenia pre používateľov vznikajúceho nástroja je aj návrh referenčnej databázy, ktorý by mal čo najlepšie reflektovať požiadavky daného odvetvia. Tie možno rozdeliť do dvoch kategórií. Používateľa zaujíma najmä množstvo informácie o jednotlivých súboroch, ktoré referenčná databáza uschováva a taktiež rýchlosť prístupu k týmto informáciám. Systémového administrátora zasa zaujíma aj to, ako efektívne táto databáza využíva diskový priestor databázového servera pod jeho správou, teda aké náklady so sebou nesie poskytovanie referenčnej databázy a jej ďalšie rozširovanie. Toto sú oblasti, na ktoré sme sa v našom návrhu primárne zamerali.

Adresujúc používateľské požiadavky, medzi atribúty referenčnej databázy sme zaradili každý relevantný atribút, ktorý o súbore dokážeme pomocou súborového systému pri budovaní databázy získať. Patrí medzi ne okrem absolútnej cesty, názvu súboru a

jeho identifikátora taktiež údaj o čase jeho vytvorenia a poslednej modifikácie, záznam o tom, kedy bol súbor po prvýkrát registrovaný v referenčnej databáze, informácie o spoločnosti a produkte, ktorého je súbor súčasťou, taktiež však akú úlohu súbor vrámci produktu plní a na akých operačných systémoch bol doposiaľ tento súbor pozorovaný. Pri rozdeľovaní týchto atribútov do *relácií* sme zasa dbali na minimalizáciu redundancie dát. Výslednú relačnú schému uvádzame na obrázku 4.3. Jediným kompromisom návrhu v oblasti redundancie uložených dát je relácia *product*. Alternatívne riešenie by bolo vynechať túto reláciu z návrhu a odkazovať sa v hlavnej relácii *file_info* priamo na samotné relácie obsahujúce údaje o spoločnostiach a ich produktoch. Týmto návrhom by sme ale stratili záznam o vzťahu jednotlivých spoločností, ich produktov a verzií. V prípade záujmu o takúto informáciu by táto musela byť zrekonštruovaná z údajov o každom súbore v databáze. Náročnosť tejto rekonstrukcie by závisela od potenciálne veľkého počtu súborov zaznamenaných v referenčnej databáze. Náš prístup síce zanáša do návrhu istú formu redundancie dát, je to však len redundancia na úrovni primárnych kľúčov relácií. Vďaka relácii *product* však vieme z databázy ľahko získať v prípade potreby aj informáciu o spoločnostiach a ich produktoch, ktorých súbory v sú v nej evidované.

Pre realizáciu návrhu sme si vybrali databázový systém *MariaDB*, ktorý je štandardnou voľbou medzi voľne dostupnými databázovými riešeniami. Návrh je implementovaný formou spustiteľného skriptu, ktorý umožňuje používateľovi jednoducho vytvoriť popisovanú relačnú schému na svojom databázovom serveri. Súčasťou implementácie sú aj *SQL pohľady a procedúry*, ktoré rekonštruujú požadované informácie o súboroch obsiahnuté naprieč reláciami a riešia tiež otázku vkladania nových záznamov do databázy. Týmto prístupom sme nielen vytvorili používateľsky jednoducho nasaditeľné riešenie, vzniknuté rozhranie zároveň umožňuje aplikáciám, ktoré s databázovým serverom komunikujú, abstrahovať od konkrétneho návrhu relačnej schémy, pokiaľ, samozrejme, ostáva nemenná množina atribútov uchovaných v databáze. To dovolí nám aj používateľom doladzovať návrh databázy jednoducho aj po nasadení nástroja.

Neskoršie experimenty však odhalili problémy s rýchlosťou vyhľadávania v referenčnej databáze. Počas experimentu sme používali popisovanú referenčnú databázu obsahujúcu približne *45 000* záznamov o súboroch. Pre vyhľadávanie v databáze sme použili aplikáciu *File-Identification-System*, ktorá v získavala informácie o zozname približne *4 500* súborov, ktorý bol vopred vytvorený v *offline* režime aplikácie. Názvy súborov aj im prislúchajúce identifikátory boli teda predpočítané a rýchlosť získania metadát o všetkých súboroch závisela výhradne od efektivity komunikácie s databázovým serverom a vyhľadávania v databáze. Výsledky všetkých meraní uvádzame v tabuľke 3.2. Neuspokojivé výsledky prvých meraní, kedy dĺžka vyhľadávania pohybovala na úrovni jednotiek minút, sme sa rozhodli adresovať vytvorením indexov pre atribúty *absolute_path* a *file_digest*, podľa ktorých v databáze informácie o súboroch vyhľadá-

1 vlákno (s)	8 vlákien (s)	1 vlákno, index (s)	8 vlákien, index (s)
562 s	286 s	6 s	3 s
573 s	301 s	6 s	3 s

Tabuľka 3.2: Časy potrebné pre vykonanie úkonu s rôznym indexovaním databázy

vame. Indexy, ktoré sme pre tieto dva atribúty vytvorili používajú dátovú štruktúru *binárny strom*, ktorá umožní výrazné zrýchlenie vyhľadávania na základe hodnôt týchto atribútov, a síce na úroveň rádovo logaritmickú v závislosti od počtu záznamov v relácii. Pridané dátové štruktúry si však vyžadujú dátový priestor na disku servera navyše, preto je nutné vytvárať indexy s rozvahou. Atribúty obsahujúce primárne kľúče relácií majú svoje indexy pre efektívne vyhľadávanie vytvorené automaticky. Akonáhle je identifikovaný konkrétny relevantný záznam v relácii *file_info*, spájanie relácií na základe referencií cudzími kľúčmi, a teda rekonštrukcia finálneho záznamu z ostatných referencovaných relácií už prebieha efektívne rádovo v logaritmickom čase od počtu záznamov. Žiadne ďalšie indexy sme preto pre naše účely nepotrebovali vytvárať. Z výsledkov tejto nenáročnej úpravy sme boli sami prekvapení. Čas vyhľadávania informácií o fixnom počte súborov sa nám podarilo zredukovať o približne *98%*. Takto navrhnutá referenčná databáza môže byť škálovaná na rádovo vyššie počty referenčných záznamov, ako *45 000*.

Popísané indexy sú novou súčasťou implementácie návrhu databázy. Prítomnosť indexov však spomaľuje vkladanie do databázy, keďže pre zachovanie ich funkcie musia byť indexy po vložení každého záznamu aktualizované a reorganizované. Preto sme skript vytvárajúci potrebné indexy oddelili od základného skriptu implementujúceho návrh databázy. Používateľ má tak možnosť vytvoriť indexy atribútov podľa svojho uváženia, napríklad, až po ukončení procesu vladania veľkého množstva dát.

3.5 Funkcionalita webovej aplikácie

Primárnym účelom webovej aplikácie je prehľadným a prívetivým spôsobom zobraziť výstupné dáta aplikácie *File-Identification-System*. Používateľ má k dispozícii možnosť filtrovať identifikované súbory podľa podporovaných kategórií popísaných v sekii 3.2.3. Podľa nášho názoru má ale využitie webovej aplikácie omnoho väčší potenciál. Preto obsahuje webová aplikácia funkcionalitu po vzore konzolovej aplikácie *File-Identification-System*. Okrem vizualizácie výstupných dát teda dokáže overiť povahu používateľom načítaného súboru aj zoznamu súborov a im prisláchajúcih identifkátorov, ktorý vznikol v *offline* režime konzolovej aplikácie. Výsledný vzhľad aplikácie uvádzame na obrázkoch 4.4 a 4.5.

Kapitola 4

Ďalšie bezpečnostné rozšírenia

V doterajších kapitolách sme popísali vznik nástroja pre identifikáciu a kontrolu integrity súborov. Ukončenie prác na základných mechanizmoch softvérového diela však považujeme len za začiatok vývoja nástroja v oblasti informačnej bezpečnosti. Nároky odvetvia sa totiž neustále vyvíjajú a nútia softvérových vývojárov reagovať na aktuálny dopyt analytikov po komplexnejších riešeniach pre identifikáciu nových, doposiaľ neznámych bezpečnostných hrozieb. Aj toto softvérové dielo je len funkčným jadrom, ktorého úlohou je poslúžiť ako stabilný základ pre ďalšie bezpečnostné rozšírenia jeho funkcionality. Témou tejto kapitoly je práve takéto rozšírenie reagujúce na známu bezpečnostnú hrozbu, ktorá umožňuje zabrániť odhaleniu škodlivého obsahu v súborovom systéme nástrojom pre ich identifikáciu.

4.1 Špecifické vlastnosti súborového systému NTFS

Súborový systém *NTFS (New Technology File System)* je v súčasnosti predvoleným súborovým systémom rodiny operačných systémov *Microsoft Windows*. V porovnaní so svojimi predchodcami, súborovými systémami rodiny *FAT (File Allocation Table)*, prináša množstvo novej pokročilej funkcionality. Veľkú časť týchto inovácií bolo možné realizovať vďaka úplne novému prístupu k tomu, ako sú chápané objekty v súborovom systéme. Tie sú reprezentované ako zoznam usporiadaných dvojíc atribútov a prúdov bajtov (nazývaných aj *stream*), ktoré tvoria hodnotu atribútu[15]. Každý objekt súborového systému je tvorený atribútmi rôznych typov a im zodpovedajúcimi hodnotami. Väčšina typov atribútov nie je prístupná používateľovi a slúži výhradne pre správne fungovanie súborového systému[16]. Medzi typy atribútov, ktoré sú prístupné patrí typ *$DATA*. To, čo používatelia chápu ako obsah súboru, je dátovým prúdom (v zahraničnej terminológii *data stream*) tvoriacim hodnotu atribútu typu *$DATA*. Predvolený dátový atribút je nepomenovaný a jeho hodnota je poskytnutá používateľovi, ktorý sa snaží pristúpiť k obsahu súboru. Súbor však môže obsahovať aj ďalšie pomenované

dátové atribúty s odlišnými hodnotami. Týmto hodnotám sa hovorí *alternatívne dátové prúdy* (v zahraničnej terminológii *Alternate Data Streams*). Používateľ môže súborom vytvárať nové alternatívne dátové prúdy s ľubovoľným obsahom. K alternatívnym dátovým prúdom možno pristúpiť vo formáte *názov_súboru:názov_prúdu:$DATA*[15] a aj v súčasnosti sú aplikáciami hojne využívané[16]. Dôvodom, prečo sú alternatívne dátové prúdy považované za kontroverznú funkcionalitu je to, že sú pred používateľom kompletne ukryté. Nie sú viditeľné v prieskumníkovi súborového systému a nezobrazia ich ani konzolové nástroje pre navigáciu v súborovom systéme. Obsah alternatívneho dátového prúdu nemá žiadny vplyv na to, čo z používateľskej perspektívy pozorujeme ako obsah súboru. Alternatívny dátový prúd je pritom fakticky ďalší ukrytý súbor a môže obsahovať čokoľvek, napríklad spustiteľný binárny kód. V súborovom systéme neexistujú žiadne obmedzenia pre veľkosť ich obsahu. Alternatívne dátové prúdy môžu byť vytvorené aj pre adresáre. Špecifikom alternatívnych dátových prúdov je fakt, že im súborový systém nepriradzuje časy vytvorenia a poslednej modifikácie. Tieto hodnoty majú všetky dátové prúdy spoločné a menia sa s modifikáciou ľubovoľného dátového prúdu[15].

Príkladom legitímneho využitia alternatívnych dátových prúdov sú napríklad prúdy s názvom *Zone.Identifier*[16]. Takýto alternatívny dátový prúd je vytvorený každému súboru, ktorý bol stiahnutý niektorým z populárnych webových prehliadačov. Obsahuje informáciu o prehliadači, pomocou ktorého bol stiahnutý a *bezpečnostnej zóne*, do ktorej patrí. Na základe bezpečnostnej zóny vytvára operačný systém upozornenia v prípade pokusu spustiť takýto stiahnutý súbor. Mnohé aplikácie majú vlastné využitie alternatívnych dátových prúdov a dokumentácia ich obsahu v referenčnej databáze je rovnako dôležitá, ako je dokumentácia obsahu predvolených nepomenovaných dátových prúdov.

4.2 Zneužitie alternatívnych dátových prúdov

Hlavným dôvodom, prečo sú alternatívne dátové prúdy dlhodobo považované za bezpečnostnú hrozbu, je ich ukrytie v súborovom systéme. Tento trend nie je v súčasnosti tak výrazný, v minulosti ale nástroje operačného systému *Microsoft Windows* poskytovali len chabú podporu pre prácu s alternatívnymi dátovými prúdmi[13]. Pri použití operačného systému *Windows XP* bolo síce možné pristúpiť k alternatívnemu dátovému prúdu, ktorého názov používateľ poznal, neexistoval však žiaden systémový nástroj, ktorý by mu umožnil identifikovať dátové prúdy súboru, ktoré nepozná. Naviac, štandardné programátorské rozhranie operačného systému *Win32 API* taktiež neposkytovalo žiadnu metódu prístupu. Táto skutočnosť umožňovala jednoducho ukrývať v alternatívnych dátových prúdoch obsah spojený s kriminálnou aktivitou. Analýza a

odhalenie takéhoto ukrytého škodlivého obsahu boli náročné na automatizáciu a vyžadovali si veľa času a úsilia forenzných analytikov. Častou technikou bolo tiež nahradenie obsahu známych alternatívnych dátových prúdov ukrývaným obsahom, čo ešte ďalej vylepšilo maskovanie kriminálnej aktivity. Alternatívne dátové prúdy môžu obsahovať aj spustiteľný strojový kód. Niektoré malvéry využívali túto techniku maskovania, aby ostali nepozorované anti-vírusovými softvérmi. Škodlivý softvér spustený z alternatívneho dátového prúdu bude naviac v Manažérovi úloh zobrazený pod pôvodným menom súboru v súborovom systéme, čo umožňuje pred používateľom dobre maskovať jeho činnosť ako legitímnu operáciu inej aplikácie[13].

Neskoršie verzie rodiny operačných systémov *Microsoft Windows* už v svojom programátorskom rozhraní poskytovali metódy prístupu k alternatívnym dátovým prúdom súborov a adresárov. Podpora pre prácu s alternatívnymi dátovými prúdmi na úrovni konzolových a iných systémových nástrojov sa taktiež zlepšila a v súčasnosti je prístup k nim omnoho otvorenejší a jednoduchší. Anti-vírusové softvéry sú taktiež pre prácu dátovými prúdmi vhodne pripravené a maskovať činnosť škodlivého softvéru je v súčasnosti na operačných systémoch *Microsoft Windows* omnoho zložitejšie[16]. Aj to sú dôvody, prečo alternatívne dátové prúdy už v súčasnosti nie sú tak významnou bezpečnostnou hrozbou pre samotnú inštaláciu systému. Aj naďalej sa v nich však môže nachádzať evidencia o kriminálnej činnosti, ktorá musí byť bez vhodného nástroja odhalená ručne zdĺhavou prácou forenzného analytika.

4.3 Podpora alternatívnych dátových prúdov

Z popísaných poznatkov vyplýva, že alternatívne dátové prúdy súborového systému *NTFS* možno veľmi jednoducho použiť pre ukrytie evidencie o kriminálnej činnosti pred nástrojmi pre identifikáciu súborov, ktoré na ich spracovanie nie sú pripravené. Využiteľnosť takých nástrojov je potom značne kompromitovaná. Predmetom tejto kapitoly teda nie je bezpečnostné rozšírenie pre obohatenie funkcionality nástroja, ale rozšírenie kritické pre zabezpečenie spoľahlivého plnenia jeho elementárnej funkcie. Aj napriek závažným dôsledkom, ktoré môže prehliadanie alternatívnych dátových prúdov v procese bezpečnostného auditu nadobudnúť, ale podpora pre prácu s alternatívnymi dátovými prúdmi medzi voľne dostupnými riešeniami nie je samozrejmosťou. Aplikácia *md5deep* pri iterácii objektov v súborovom systéme ignoruje obsah alternatívnych dátových prúdov[13]. Vzhľadom k roku, z ktorého uvedený zdroj pochádza sme sa rozhodli overiť, či je toto tvrdenie pravdivé aj pre jej novšiu implementáciu nazývanú *hashdeep*. Po analýze zdrojového kódu nástroja pre uľahčenie priebehu bezpečnostných auditov musíme konštatovať nezmenenú situáciu v oblasti podpory pre prácu s alternatívnymi dátovými prúdmi[2]. Pre korektnú implementáciu funkcionality je taktiež

dôležité nezabúdať pri rekurzívnom prehľadávaní súborového systému na alternatívne dátové prúdy adresárov. V minulosti je známy minimálne jeden príprad komerčného nástroja pre identifikáciu a kontrolu integrity súborov, ktorý nepodporoval prácu s týmito alternatívnymi dátovými prúdmi[13].

Hoci situácia v oblasti podpory spracovania týchto atribútov je mierne znepokojivá, jej implementácia do existujúcich softvérových aplikácií je relatívne nenáročná a v štandardnom programátorskom rozhraní operačných systémov *Microsoft Windows* nachádza prívetivú podporu[15]. Po vzore oddelenia *NRSL*[5] organizácie *NIST* sme starostlivo implementovali podporu pre spracovanie alternatívnych dátových prúdov súborov aj adresárov[13]. Riešeniu problému sme sa však venovali aj pre platformy založené na báze *Unixu*. Zistili sme, že v prípade, že je súborový systém *NTFS* k zariadeniu používajúcemu túto platformu pripojený s potrebnými opciami povoľujúcimi dátové prúdy (*"-o streams_interface=windows"*), názvy alternatívnych dátových prúdov súborov a adresárov možno získať pomocou rozšíreného atribútu *ntfs.streams.list*. Po obdržaní názvu alternatívneho dátového prúdu už k nemu možno pristúpiť v súborovom systéme vo formáte *názov_súboru:názov_prúdu*[10]. Zatiaľ čo veľkosť čítaného alternatívneho dátového prúdu nie je v tejto platforme obmedzená, veľkosť zoznamu názvov týchto prúdov pre konkrétny súbor alebo adresár je limitovaná na *64kB*[10]. V prípade malého množstva veľkých alternatívnych dátových prúdov je teda možné k týmto prúdom bez problémov pristúpiť. Problémy môže spôsobiť príliš veľký počet alternatívnych dátových prúdov priradených jednému objektu. V takom prípade nemožno pristúpiť k žiadnemu z nich. O takejto skutočnosti je však používateľ upovedomený a alternatívne dátové prúdy konkrétneho podozrivého súboru môže preskúmať odlišnými metódami neskôr. S použitím tohto prístupu môže náš nástroj len s malými obmedzeniami kopírovať podporu pre alternatívne dátové prúdy, ktorú ponúka používateľom využívajúcim platformu *Microsoft Windows*.

Veríme, že po tomto funkčnom rozšírení ponúka náš nástroj používateľom spoľahlivú základnú funkcionalitu pre identifikáciu a kontrolu integrity súborov. Po prieskume kvalít niektorých existujúcich nástrojov zameraných na túto oblasť zisťujeme, že pre takýto nástroj existuje na trhu priestor aj využitie.

Záver

Forenzná analýza v oblasti informačnej bezpečnosti je neustále sa vyvíjajúce odvetvie. Úlohou forenzných analytikov v tomto odvetví je skúmať zaistené zariadenia, či už je to za účelom získania evidencie o kriminálnej činnosti páchateľa alebo v snahe odhaliť a analyzovať bezpečnostné zraniteľnosti systémov a navrhovať riešenia, ako pred týmito zraniteľnosťami chrániť samotné systémy aj dáta ich používateľov. Rastúce množstvo dát, ktoré obsahujú typické výpočtové zariadenia súčasnosti, a stále nové sofistikovanejšie spôsoby zneužitia ich funkcionality ale kladú zvyšujúce sa nároky na prácu analytikov, ktorá musí byť rýchla, avšak aj naďalej vykonávaná prezieravo a som zmyslom pre detail. To núti odborníkov v oblasti zamýšľať sa nad novými metódami, ktoré zovšeobecňujú jednotlivé problémy forenznej analýzy, a dokážu vo viacerých jej doménach aspoň čiastočne automatizovať proces skúmania a redukovať tak intelektuálnu prácu analytika na únosnú mieru. Výsledkom je dopyt po softvérových nástrojoch, ktoré realizujú tieto metódy a ich implementácia umožňuje ich vhodné použitie v rôznorodých podmienkach nekonzistentného sveta informačných systémov.

Ukazuje sa, že voľne dostupné existujúce nástroje často neposkytujú dostatočnú funkcionalitu a všestrannosť, ktorú si oblasť forenznej analýzy vyžaduje. Táto práca sa zaoberá tvorbou softvérového nástroja, ktorý si kladie za cieľ lepšie vyhovieť týmto požiadavkám. Identifikuje nedostatky existujúcich riešení a snaží sa ich adresovať. Popisuje motivácie stojace za návrhom jednotlivých súčastí nástroja a analyzuje úvahy, ktoré boli potrebné pre ich vznik. V predošlých kapitolách sme sa venovali významným miľníkom vývoja, ktorými sme postupne vylepšovali vznikajúce softvérové dielo. Dbali sme pritom najmä na to, aby mal čitateľ so záujmom možnosť porozumieť požiadavkám praxe, taktiež však niektorým implementačným detailom, ktoré sme považovali za významné a užitočné pre komunitu softvérových vývojárov.

Výsledkom práce je softvérový nástroj, ktorého vlastnosti vyplývajú z kombinácie požiadavok odborníkov v oblasti forenznej analýzy ale aj nášho ďalšieho prieskumu danej problematiky a našich vlastných nápadov. Medzi najdôležitejšie charakteristiky nástroja patrí flexibilita, ktorú ponúka svojím používateľom pri identifikácii súborov a kontrole ich integrity. Hoci niektoré existujúce nástroje taktiež ponúkajú vhodnú prenositeľnosť a využiteľnosť pre rôzne platformy a operačné systémy, nepodarilo sa nám nájsť nástroj, ktorý by túto vlastnosť kombinoval s dostatočnou škálovateľnosťou

celkového riešenia pripraveného realizovať automatizované analýzy veľkého množstva dát jednoduchým a časovo efektívnym spôsobom. V práci vysvetľujeme detailným a porozumiteľným spôsobom, ako sme sa tieto vlastnosti pokúsili dosiahnuť pri vývoji nástroja my. Prezentujeme tiež výsledky, ktoré hodnotíme ako veľmi pozitívne. Radi však budeme konfrontovaní aj s inými názormi a prístupmi. V tejto práci taktiež diskutujeme o niektorých bezpečnostných nedostatkoch existujúcich nástrojov na trhu, ktoré sme počas obdobia štúdia tejto problematiky objavili. Ako reakciu na tieto zistenia sme nielen dbali o to, aby sme sa podobným chybám v našej práci vyhli. Venovali sme v texte celú kapitolu tomu, aby sme popísali vážny dopad, ktorý tieto nedostatky môžu nadobudnúť, a taktiež konkrétne implementačné riešenia, ktorými by sme radi posmelili čitateľa, aby podobnú funkcionalitu implementoval aj vo svojej podobnej aplikácii.

Za najdôležitejší prínos tohto nástroja však považujeme jeho funkcionalitu súvisiacu s možnosťou budovania a priebežného udržiavania centralizovanej referenčnej databázy súborov. Myslíme si, že práve táto súčasť prispieva k jeho všestrannosti najvýraznejšie. Je to totiž charakter záznamov uložených v referenčnej databáze, ktorý určuje konečné zameranie nástroja pre identifikáciu súborov. Pokúsili sme sa ponúknuť organizáciám možnosť nespoliehať sa na existujúce zdroje referenčných dát a vytvoriť si výkonné a jednoducho udržiavateľné riešenie presne podľa ich konkrétnych potrieb. Nástroj, ktorý sme vytvorili by tak mohol slúžiť na identifikáciu podozrivých súborov v systéme rovnako ako na identifikáciu prítomnosti verzií softvérových balíkov, ktoré sú považované za zraniteľné a nebezpečné. Tieto nedostatky by mohli byť na dotyčných zariadeniach následne odstránené. Práve túto oblasť považujeme za možný smer ďalšieho vývoja softvérového diela. Jeho súčasťou sa môžu stať výstupné komponenty schopné budovať databázu súborov s určitou konkrétnou požadovanou vlastnosťou, ktorá napokon rozšíri jeho možné uplatnenie.

Oblasť forenznej analýzy núti svojich odborníkov prichádzať stále s novými kreatívnymi spôsobmi riešenia problémov. Len čas ukáže, či sa nám im touto implementačnou prácou podarilo pomôcť.

Literatúra

[1] Guide to JNI (Java Native Interface). https://www.baeldung.com/jni. Použité 2020-05-30.

[2] jessek/hashdeep. https://github.com/jessek/hashdeep/. Použité 2020-06-15.

[3] md5deep and hashdeep - Latest version 4.4. http://md5deep.sourceforge.net/. Použité 2020-06-15.

[4] MySQL Connector/C++ 8.0 Developer Guide. https://dev.mysql.com/doc/connector-cpp/8.0/en/. Použité 2018-10-15.

[5] National Software Reference Library (NSRL). https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl. Použité 2019-10-10.

[6] Notepad++ And Unsigned DLLs. https://medium.com/@threathuntingteam/notepad-and-unsigned-dlls-a5cdcfb86749. Použité 2019-10-01.

[7] Sahmhain. https://www.la-samhna.de/samhain/index.html. Použité 2020-1-23.

[8] AIDE (Advanced Intrusion Detection Environment). https://aide.github.io/. Použité 2020-1-23.

[9] CSIRT.SK. https://www.csirt.gov.sk/csirtsk-7d6.html. Použité 2018-10-15.

[10] GETXATTR(2) Linux Programmer's Manual. https://www.man7.org/linux/man-pages/man2/getxattr.2.html. Použité 2020-02-08.

[11] MD5 Collisions The Effect on Computer Forensics. https://repo.zenk-security.com/Cryptographie%20.%20Algorithmes%20.%20Steganographie/MD5%20Collisions.pdf. Použité 2018-11-30.

[12] NIST (national Institute of Standards and Technology). https://www.nist.gov/. Použité 2019-10-10.

[13] A Win32-Based Technique for Finding and Hashing NTFS Alternate Data Streams. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50914. Použité 2020-01-04.

[14] ZeroMQ. https://zeromq.org/. Použité 2019-03-30.

[15] © Microsoft 2020. File Streams (Local File Systems). https://docs.microsoft.com/en-us/windows/win32/fileio/file-streams. Použité 2020-01-04.

[16] Adrian Denkiewicz. [CQURElabs] Alternate Data Streams. https://cqureacademy.com/blog/alternate-data-streams. Použité 2020-01-04.

[17] Eric Gerbier. AFICK (Another File Integrity Checker). http://afick.sourceforge.net/index.html. Použité 2020-1-23.

[18] MariaDB. C & C++ connectors. https://mariadb.com/kb/en/library/mariadb-connector-c/. Použité 2018-10-15.

[19] OSSEC Project Team. OSSEC: Open Source HIDS Security. https://www.ossec.net/. Použité 2020-1-23.

[20] Tripwire. Open Source Tripwire. https://github.com/Tripwire/tripwire-open-source. Použité 2020-1-23.

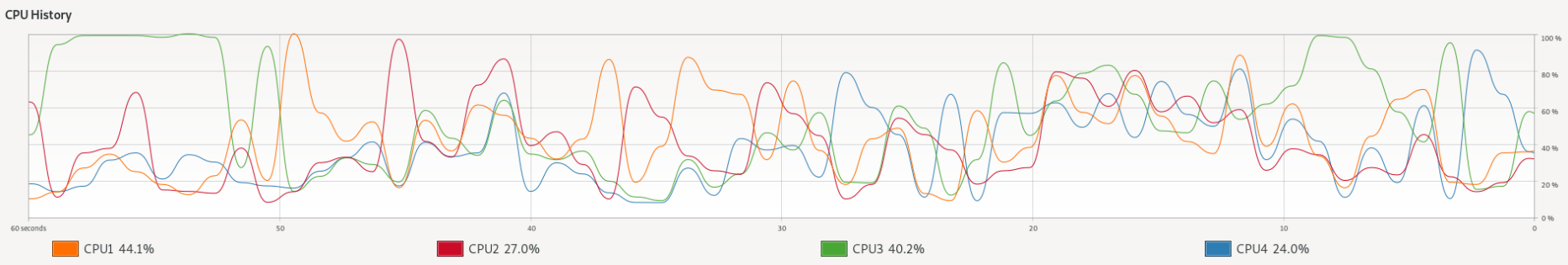
Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu. Pretože kompilácia niektorých aplikácií na operačných systémoch Windows si môže vyžadovať zbytočné prvotné úsilie spojené s inštaláciou rôznych softvérových balíkov, v adresári *build* možno nájsť skompilované a staticky nalinkované aplikácie *File-Identification-System.exe* a *Sysupdate.exe* pripravené na spustenie.

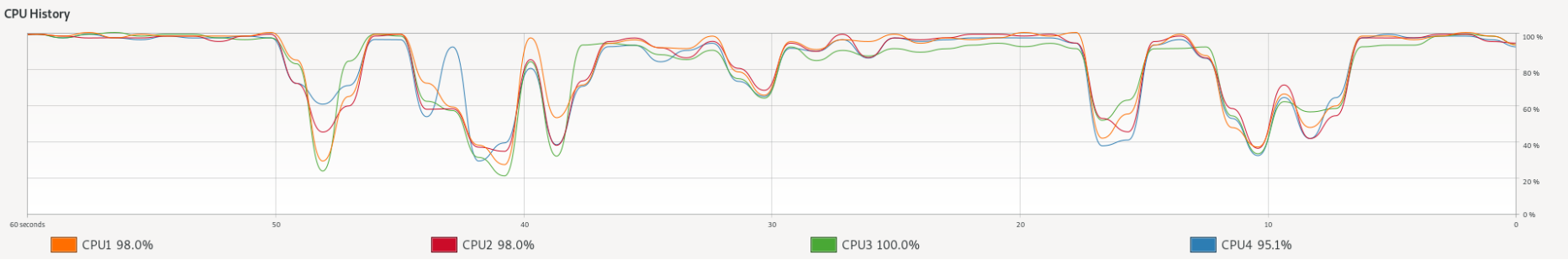
Zdrojový kód s užívateľskou príručkou je zverejnený aj na stránke `https://github.com/M-Fedor/File-Identification-System`.

Príloha B: Schémy návrhov a záznamy experimentov

V tejto prílohe uvádzame schematické ilustrácie a taktiež snímky priebehov experimentov, ktoré boli príliš rozsiahle na to, aby boli vyobrazené priamo v texte.

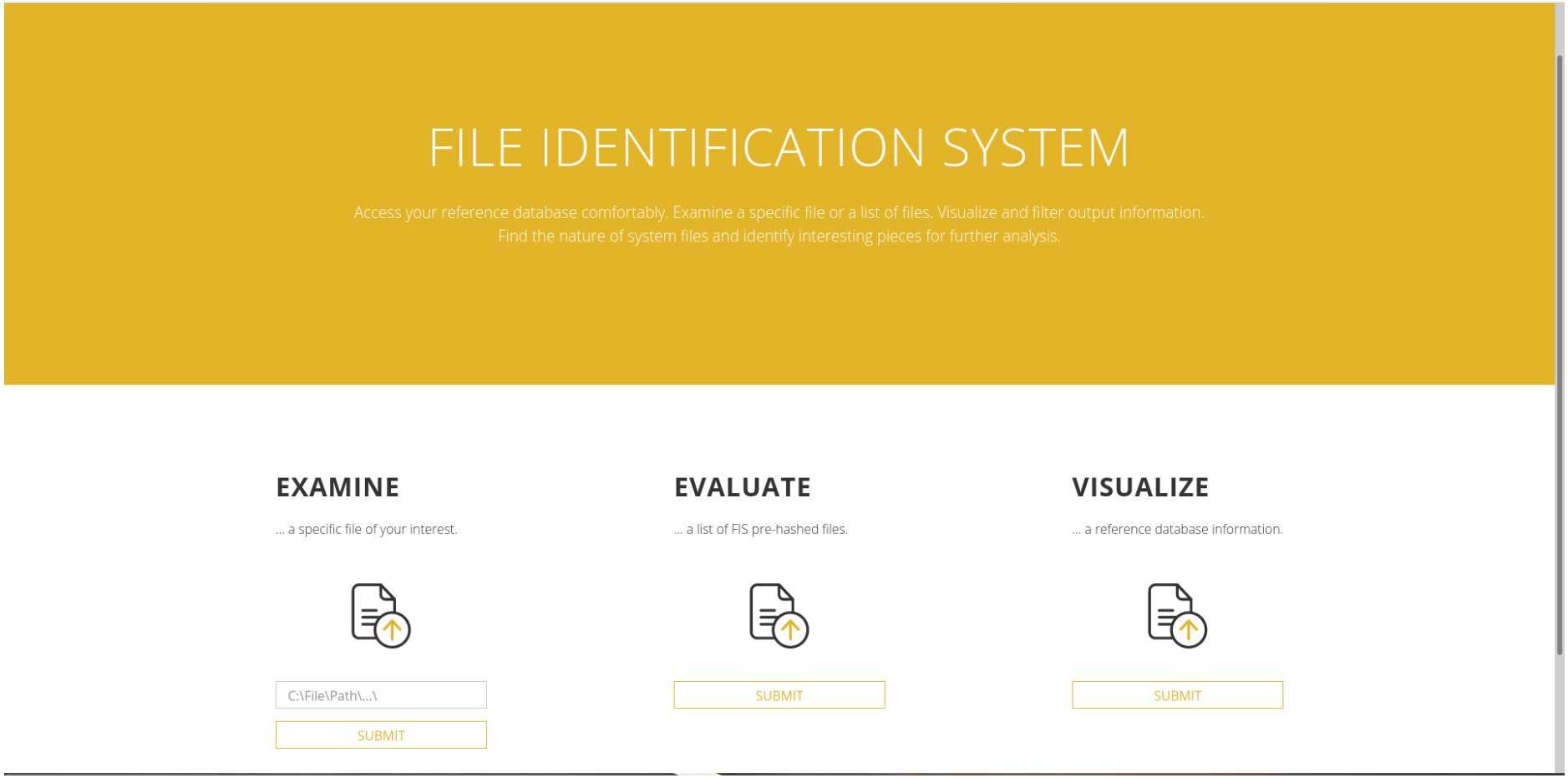


Obr. 4.1: Vyťaženie fyzických jadier procesora v čase aplikáciou *File-Identification-System* s použitím jedného vlákna vykonávania. Môžno pozorovať nedostatočne efektívnu prácu s procesorom zariadenia.

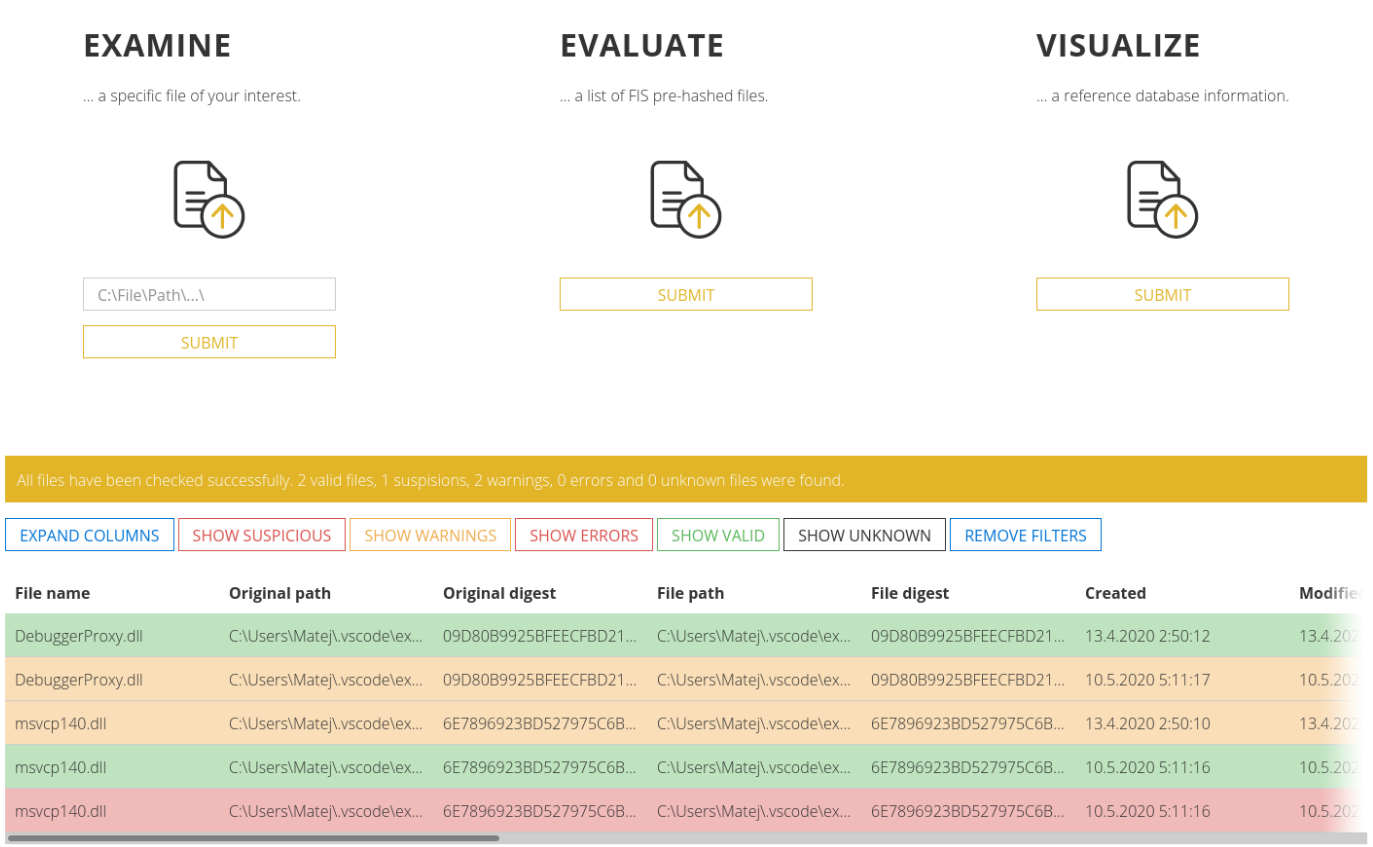


Obr. 4.2: Vyťaženie fyzických jadier procesora v čase aplikáciou *File-Identification-System* s použitím ôsmych paralelných vlákien vykonávania. Môžno pozorovať omnoho efektívnejšie využitie procesora zariadenia.

Obr. 4.3: Relačný diagram výsledného návrhu referenčnej databázy



Obr. 4.4: Používateľske rozhranie webovej aplikácie



Obr. 4.5: Používateľske rozhranie webovej aplikácie