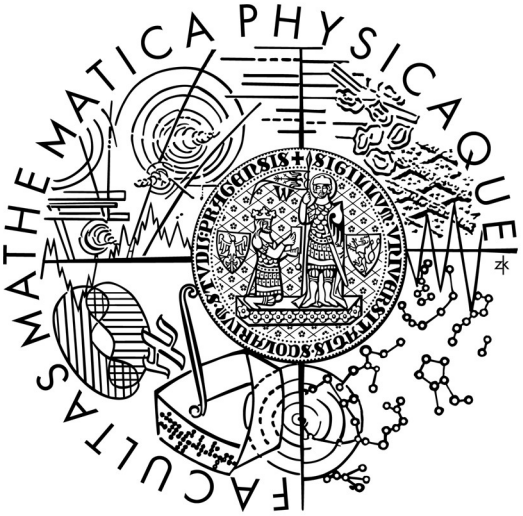
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



David Matoušek

Metody sledování chování procesů na Windows

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Jakub Yaghob, Ph.D.

Studijní program: Informatika, softwarové systémy

Děkuji vedoucímu své diplomové práce Jakubovi Yaghobovi za rady, kontrolu a cenné připomínky, které mi pomohly při realizaci této práce. Dále bych chtěl poděkovat Martinu Drábovi za konzultace problémů, kterých se tato práce dotýká. Nakonec bych rád poděkoval autorům použité literatury a všem výzkumníkům, kteří se dělí o své poznatky v oblasti, se kterou souvisí má práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 1. listopadu 2009 David Matoušek

Obsah

Název práce: Metody sledování chování procesů na Windows

Autor: David Matoušek

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Jakub Yaghob, Ph.D.

E-mail vedoucího: Jakub.Yaghob@mff.cuni.cz

Abstrakt: Tato práce rekapituluje vybrané metody pro sledování chování procesů na operačních systémech Windows. Zvláštní pozornost je věnována použití virtualizace jako pomocného prostředku k řešení problému. Cílem práce bylo zhodnotit možnosti jednotlivých metod a navrhnout systém pro sledování chování procesů za pomoci virtualizace. Navržený systém je postaven na virtualizačním software VirtualBox od Sun Microsystems a operačním systému Windows XP Service Pack 3. Běh virtualizovaného systému Windows je upraven tak, že v přesně definovaných stavech jsou na základě aktuálního stavu paměti vytvářeny elementární události sledovacího systému. Tyto události lze dále analyzovat a pomocí nich modelovat objekty existující v systému a jejich interakce. Zjištěné interakce mezi sledovanými procesy a objekty v systému determinují chování těchto procesů.

Klíčová slova: sledování, chování, proces, Windows

Title: Windows process behavior checking

Author: David Matoušek

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Yaghob, Ph.D.

Supervisor's e-mail address: Jakub.Yaghob@mff.cuni.cz

Abstract: The thesis summarizes selected methods of process behavior monitoring on Windows operating systems. The thesis is focused especially on using virtualization as a support tool for process behavior checking. The main goal of the thesis was evaluation of process behavior checking methods and designing a virtualization based monitoring system. The designed system is based on VirtualBox virtualization software by Sun Microsystems and operating system Windows XP Service Pack 3. The execution of the virtualized Windows system is modified so that in precisely defined states the monitoring system creates elementary events based on the current state of the system memory. These events can then be analyzed in order to simulate the operating system objects and their interactions. The interactions between the monitored processes and other objects in the system then determine the behavior of these processes.

Keywords: behavior, monitoring, process, Windows

1 Úvod

Osobní počítače a Internetová konektivita se v posledních patnácti letech dostali z oblasti luxusu do oblasti běžně dostupného zboží a služeb. V roce 2008 přesáhl počet uživatelů Internetu neuvěřitelných 1,5 miliardy, což je asi pětinásobek počtu uživatelů z konce roku 2000. V roce 1996 neměl Internet ani 60 milionů uživatelů. A vše nasvědčuje tomu, že tento růstový trend bude nadále pokračovat.

Lidé nejčastěji využívají služeb emailu a systému webových stránek. Stále častěji pak jsou využívány služby pro instantní komunikaci, sociální sítě a služby pro sdílení dat. S narůstajícím počtem uživatelů počítačů a Internetu a s rozvojem technologií nutně klesá průměrná znalost běžného uživatele o těchto technologiích. Lidé jsou naučeni používat několik málo aplikací, pomocí kterých využívají služeb poskytovaných počítači a Internetem. Většinou nemají povědomí o technických záležitostech v pozadí a tedy ani o otázkách bezpečnosti užívání těchto technologií. I proto je stále větší poptávka po aplikacích, které běžným uživatelům bezpečnost zaručí.

Každý den se do oběhu dostane řada nových škodlivých programů, pro které se vžilo pojmenování *malware* (ze spojení anglických slov malicious a software). Je potřeba, aby tvůrci bezpečnostních aplikací reagovali na nové hrozby co nejrychleji. Jejich každodenní práce je analýza nových vzorků malware a implementace protiopatření, která pak posílají svým zákazníkům. Některé produkty jsou takto aktualizovány i několikrát denně.

1.1 Zaměření a cíle práce

Analýza neznámého programu vyžaduje expertní znalosti o platformě a operačním systému, na kterém analyzovaný program běží. Bylo vyvinuto mnoho nástrojů i rozsáhlých aplikací, které pomáhají při provádění takové analýzy. Jednou z hlavních oblastí, která je zkoumána při analýze neznámeho programu je jeho chování. Co to vlastně je chování programu a jaké aspekty chování jsou důležité z hlediska bezpečnosti? Tyto otázky jsou podrobněji rozebrány v dalších částech této práce.

Tato práce se zabývá právě metodami používanými v aplikacích, které se snaží pomoci při analýze neznámých programů běžících na operačních systémech Windows s jádrem NT verze 5 a vyšším (odpovídá verzím Windows 2000, XP, 2003, Vista, 2008 a Seven) tím, že provádějí tzv. dynamickou analýzu. Veškerý text by měl být chápán v souvislosti s těmito operačními systémy. Dynamická analýza znamená skutečné nebo simulované spuštění jedné nebo několika málo instancí testovaného vzorku a sledování akcí, které vzorek při svém běhu vykonal. Další možností je analýza statická, při níž se vzorek nespouští a pouze se analyzuje jeho binární reprezentace. Statickou analýzu tato práce nepokrývá.

Práce shrnuje a srovnává vybrané metody, běžně i méně používané při implementaci systémů dynamické analýzy. Speciálně se věnuje otázce využití virtualizace, jako prostředku pro vybudování systému pro sledování chování neznámého programu. Dále je v této práci prezentován funkční základ sledovacího systému, který je implementován s pomocí virtualizačního software VirtualBox od firmy Sun Microsystems. Odtud také pochází jeho název PABOV – Process Analyzer Based on VirtualBox. Tento systém je v prezentované

verzi omezen pro analýzu programů běžících na 32-bitové verzi operačního systému Windows XP SP3.

Cílem práce je podat souhrn metod používaných pro sledování chování programů, zhodnotit jejich možnosti a porovnat výstupy existujících volně dostupných sledovacích systémů s výstupy navrženého systému PABOV.

1.2 Krátké shrnutí obsahu práce

V kapitole 2 se krátce věnujeme úvodu do problematiky chování procesů na OS Windows. V této kapitole je podán základní náhled na část architektury Windows, která je podstatná pro pochopení dalších částí práce. Je zde vysvětleno, které aspekty chování jsou pro analýzu chování z hlediska bezpečnosti důležité a které naopak lze opominout.

Kapitola 3 je pak přehledem metod použitelných pro implementaci sledovacího systému bez použití virtualizace. Na to navazuje přehled dostupných virtualizačních a jim příbuzných metod popsaný v kapitole 4 opět s ohledem na možnosti využití pro implementaci sledovacího systému.

Existující volně dostupné projekty na téma sledování a analýza chování programů jsou představeny v kapitole 5. U každého projektu je rozebrán jeho návrh a použitá metoda.

Následuje seznámení s virtualizačním softwarem VirtualBox a popis možností jeho využití k implementaci sledovacího systému. Zde je prezentován i základ systému PABOV, konkrétně popis modifikací zdrojových kódů VirtualBoxu pro potřeby PABOVu a jak je řešeno napojení vlastního zdrojového kódu PABOVu na modifikované zdrojové kódy VirtualBoxu.

Kapitola 7 popisuje návrh sledovacího systému PABOV, funkce jednotlivých částí a komunikace mezi nimi. Dále jsou zde diskutovány použité struktury, objekty systému a postupy, které pomáhají k zjišťování vztahů mezi objekty a chování sledovaných procesů a vytváření uživatelsky příjemného výstupu celého systému. Kapitola končí popisem postupu sestavení a instalace systému.

Závěrečné kapitoly se věnují testům jak navrženého systému, tak i existujících dostupných systémů popsaných v kapitole 5. Na příkladech reálného malware jsou srovnány výstupy testovaných systémů. Hlavní důraz je kladen na to, zda by z daných výstupů bylo možné odhalit škodlivost nebezpečných vzorků a pokud ano, pak zda by bylo možné takovou detekci provést automaticky bez nadměrného zesložitění sledovacího systému a bez přílišné chybovosti.

Úplný závěr práce pak hodnotí dosažené výsledky v této práci a diskutuje možná rozšíření.

Součástí práce je i přiložené médium, které obsahuje elektronickou verzi této práce, zdrojové kódy PABOVu a další software a pomocné soubory potřebné k jeho běhu. Dále jsou na médiu testované vzorky a výstupy sledovacích systémů, které v práci porovnáváme. Struktura souborů na přiloženém médiu je popsána v příloze A této práce. Struktura zdrojových kódů pak je popsána v příloze B.

Práce končí seznamem použité literatury a referencemi.

2 Vybrané základy fungování systému, procesů a jejich chování

Jádro operačního systému je složeno z mnoha komponent, které jsou navzájem provázány. Není třeba detailně znát všechny tyto komponenty a jejich funkce, abychom mohli porozumět problematice sledování procesů a jejich chování. V této části je shrnutí nejdůležitějších pojmů, které mají vztah k dalšímu textu.

2.1 Object Manager

Nejdůležitější částí vzhledem k naší problematice je Object Manager. Jedná se o komponentu systému, která se stará o objekty v systému – o jejich vznik, zánik, přístup k nim apod. Pro Object Manager je každý zdroj v systému objekt. Ať už se jedná o soubor, proces, vlákno, registrový klíč nebo třeba semafor, vše má vlastnost a základní atributy objektu. Na každý objekt se lze odkazovat pomocí jeho adresy v paměti, kde se nalézá jeho objektová hlavička. Pokud však proces chce vytvořit nebo manipulovat nějakým objektem, nepracuje přímo s pamětí, ale získá od Object Manageru tzv. *handle*, což je index specifický pro kontext volajícího procesu, pomocí něhož dokáže Object Manager cílový objekt najít. Takto jsou objekty a jejich vnitřní struktura odstíněny od kódu procesů.

2.2 Procesy a aspekty jejich chování

Při spuštění programu v operačním systému dojde k vytvoření procesu – to je objekt v jádru systému, který má přidělen vlastní adresový prostor, může přistupovat ke zdrojům (jako jsou soubory, zařízení, registry, synchronizační primitiva a další objekty) a službám systému (systémová volání) a sdružuje vlákna, které vykonávají binární kód programu.

Oddělení adresových prostorů procesů je základním nástrojem bezpečnosti operačního systému. Opomineme-li možnost přístupu procesu ke zdrojům a službám systému, může proces ovlivňovat pouze vlastní paměť a tak žádná jeho činnost nemůže narušit běh samotného systému nebo ostatních procesů. Chyba nebo záměrné poškození v procesu se tak nepromítne do běhu ostatních částí a procesů systému. K tomu, aby mohlo dojít k nějaké interakci mezi procesy nebo k interakci se zdroji systému musí vlákno procesu volat nějaké z definovaných systémových volání. Tato volání jsou podrobena ověření přístupových práv a jejich parametry jsou ověřovány.

Dalším prostředkem k realizaci bezpečnosti v systému je podpora hardwaru pro exekuci kódu aplikačního software v jiném režimu (módu) než kódu jádra systému. Veškerý kód programu v procesu je spouštěn v uživatelském módu, který je omezen tak, že některé instrukce procesoru jsou zakázány a pokus o jejich vykonání vede k chybě. Kód procesu tak například nemůže manipulovat s mapováním paměti a ovlivnit tak paměť jiného procesu. Naopak kód jádra je vykonáván v módu jádra, ve kterém jsou k dispozici všechny instrukce. Obsluha systémových volání přepíná mezi uživatelským módem a módem jádra. Kód systémového volání je pak vykonáván v módu jádra a je proto možné, že manipuluje i s jiným procesem

nebo zdroji systému.

Uživatelé potřebují přístup ke svým datům z různých aplikací a přílišné omezování v oblasti práv programů vede k tomu, že práce uživatele je ztěžována technickými záležitostmi, které ho nezajímají. Proto se často setkáváme s nastavením systému, kde libovolný program spuštěný uživatelem není omezen svými právy. A tak je běžné, že malware je spouštěn s právy, které mu umožňují přístup a manipulaci se zdroji v systému tak, že bez problému dosáhne svého cíle – např. krádeže dat nebo přístupových kódů, instalace do systému, infikování důvěryhodných procesů atp.

Z hlediska bezpečnosti a chování procesu jsou operace prováděné procesem v rámci jeho vlastního adresového prostoru v uživatelském módu poměrně nezajímavé. S výjimkou sdíleného mapování paměti nebo mapování souboru nemůže kód procesu v uživatelském módu nijak ohrozit bezpečnost systému nebo uživatelská data, ovlivnit chování systému nebo jiných procesů a pro práci se zmíněným mapováním paměti musí také nejdříve využít volání systémových služeb. Chování procesu zajímavé z hlediska bezpečnosti je tedy určeno voláním systémových služeb. Chce-li proces otevřít soubor a číst z něj data, získat možnost k ukončení jiného procesu, poslat data přes síťové rozhraní nebo zapsat hodnotu do registru, musí provést volání k tomu určené systémové služby.

V aplikacích identifikace vzorků malware se chování procesu, které nevyužívá systémová volání, typicky nezkoumá. Způsoby uchovávání dat, šifrování dat, struktura protokolu komunikace vzorku s Internetovým serverem apod. jsou v první fázi analýzy nedůležité a u většiny vzorků se neprovádějí z důvodu nedostatku zdrojů. Nových vzorků k analýze se denně objeví velmi mnoho a jen velmi málo z nich si zaslouží větší pozornost. V dalším textu se budeme téměř výhradně zabývat analýzou chování, při němž dochází k interakci se systémem a jeho objekty.

2.3 Knihovny, import a export funkcí, volání

Proces, který chce interagovat se systémem, musí používat systémová volání. Z mnoha důvodů je vhodné, aby operační systém měl relativně málo systémových volání s jasně definovanými rolemi tak, aby implementace jádra byla co nejjednodušší. Naopak logika složitějších operací by měla být umístěna v uživatelském režimu. Operační systém nabízí řadu knihoven, které implementují obrovskou škálu funkcí, které běžné aplikace mohou chtít použít. Tím knihovny operačního systému odstiňují programátory uživatelských aplikací od systémových volání, která se navíc nedoporučuje používat přímo z důvodu možných odlišností v různých verzích operačního systému.

Binární reprezentace programu v sobě uchovává odkazy na funkce a knihovny dostupné v systému. Při zavedení programu do paměti, tedy při vytvoření procesu, se do jeho adresového prostoru namapují kopie potřebných knihoven a do připravených tabulek se uloží konkrétní adresy požadovaných funkcí. Tabulka, která popisuje funkce, které program během svého běhu chce použít se nazývá IAT (Import Address Table). Obsahuje seznam knihoven, ze kterých chce tzv. importovat některé funkce a dále seznam těchto funkcí. Při spouštění procesu se pak načte daná knihovna a zjistí se adresy vyžadovaných funkcí. Tyto adresy se zapíší do připravených částí IAT. Program, který využívá importovanou funkci, pak ve svém kódu má pouze nepřímé volání s odkazem na tu část IAT, která obsahuje adresu importované funkce. Po namapování knihovny do adresového prostoru procesu a vyplnění adresy importované funkce v IAT bude program již bez problémů fungovat tak, jak bylo zamýšleno.

K tomu, aby se v knihovně našla požadovaná funkce, musí knihovna nějakým způsobem

poskytovat seznam funkcí, které nabízí (exportuje). K tomu slouží EAT (Export Address Table). Tato tabulka obsahuje seznam všech funkcí, které knihovna exportuje, a k nim příslušné adresy do kódu knihovny.

Odstínění se týká všech běžných operací, jako jsou například práce se soubory, práce s registry, instalace zařízení, práce se sítí atd. Navíc řada systémových funkcí obsahuje další části, které se v různých situacích opakují, nebo mají svůj jasně definovaný vlastní význam. Proto je mezi systémovými knihovnami hierarchie – tj. i systémové knihovny importují funkce z dalších systémových knihoven. Například celá sada systémových volání je zapouzdřena v systémové knihovně `ntdll.dll`. A pokud chce program například otevřít soubor, volá nejprve funkci **CreateFileA** (popř. **CreateFileW**) ze systémové knihovny `kernel32.dll`, která dále importuje a volá funkci **NtCreateFile** z `ntdll.dll` zapouzdřující systémové volání pro tvorbu a otevírání souborů a zařízení, která teprve provede požadované systémové volání. Volání se obslouží v jádře a jeho výsledky se navrátí zpět volající funkci do `ntdll.dll`, odkud jsou dále předány zpět do funkce z knihovny `kernel32.dll`, odkud nakonec putují zpět do programu, který chtěl otevřít soubor.

I jádro operačního systému obsahuje různé součásti, které poskytují nějaké funkce ostatním částem a zároveň využívají funkcí částí jiných. Object Manager se stará o objekty v systému, Memory Manager spravuje paměť, Configuration Manager implementuje operace s registry apod. Samotné volání systémové služby tak může vyvolat další řadu funkcí v různých částech jádra podobně jak je tomu u knihoven v uživatelském režimu.

2.4 Ovladače (drivers) a mód jádra

Jak již bylo řečeno, kód vykonávaný v uživatelském módu nemůže sám o sobě ovlivnit nic kromě paměti procesu, ve kterém je spouštěn. Naopak kód vykonávaný v módu jádra není ničím omezen. Takový kód ani nemusí využívat funkcí jádra, které jsou poskytovány různými částmi systému. Může obcházet veškerá zabezpečení definovaná systémem práv a manipulovat libovolnými prostředky, které má systém k dispozici. Z tohoto důvodu je nutné, aby veškerý kód spuštěný v módu jádra systému byl důvěryhodný.

Ve skutečnosti jsou k dispozici čtyři úrovně vykonávání instrukcí zvané Ring 0 (režim jádra), Ring 1, Ring 2 a Ring 3 (uživatelský režim). Jádro operačního systému běží výhradně v neomezeném režimu jádra, kdežto uživatelské aplikace v nejvíce omezeném uživatelském režimu. Ring 1 a 2 jsou jakési mezistupně. V úrovni Ring 2 lze vykonávat instrukce Ring 3 a navíc lze provádět některé operace, které v Ring 3 nejsou dovoleny. Ring 2 je určen privilegovaným uživatelským procesům. Ring 1 opět rozšiřuje možnosti Ring 2 a je určen pro kód s přístupem k hardware. Ring 0 pak nemá omezení. V používaných operačních systémech se však většinou používají pouze úrovně Ring 0 pro jádro a Ring 3 pro všechny uživatelské aplikace.

Některé aplikace vyžadují přístup k jádru systému a systém jim to umožňuje prostřednictvím ovladačů. Uživatelské programy tak mohou do systému nainstalovat ovladač, jehož kód bude vykonán v režimu jádra. Bohužel se koncept podepisování ovladačů, který s sebou nese poměrně snadné dohledání autora libovolného ovladače, nedostal do 32-bitových verzí Windows a tudíž není možné jednoduše zajistit, aby každý ovladač v systému byl důvěryhodný. Navíc byly objeveny postupy jak vykonat kód v režimu jádra i bez ovladače.

Vzhledem k možnostem, které má každý kód v módu jádra se považuje z hlediska bezpečnosti za fatální, pokud se v tomto módu začne vykonávat škodlivý kód. Podobně je tomu i v případě problému sledování chování procesů. Mnohé techniky umožňují sledovat chování

procesů, dokud je jejich kód vykonáván v uživatelském režimu. Jakmile ale proces zavede do systému ovladač nebo jiným způsobem dojde k exekuci kódu v režimu jádra, tak naprostá většina sledovacích metod selhává v tom smyslu, že kód v režimu jádra nedokáže spolehlivě sledovat. Naštěstí však proces opět musí využít systémových volání, která lze vysledovat, k tomu, aby načetl do systému ovladač nebo se jinak propracoval k režimu jádra. A vzhledem k tomu, že jen velmi málo legitimních programů vyžaduje práci s ovladačem a prakticky žádný legitimní program se nesnaží použít nestandardních způsobů k vykonání kódu v režimu jádra, mohou sledovací systémy s velkou pravděpodobností konstatovat nalezení malware, pokud zjistí u sledovaného procesu takové chování. V existujících sledovacích systémech je zcela běžné omezení na sledování chování v rámci uživatelského režimu popř. konstatování, že výstupy při sledování chování kódu běžícím v režimu jádra nemusí být úplné.

2.5 Sledování vzorku, interakce a falešné výstupy

Častým modelem při tvorbě sledovacího systému je existence speciálně odděleného a udržovaného systému, ve kterém probíhá simulace nebo skutečný běh testovaného programu. Tento systém je po použití restaurován do svého původního stavu tak, aby analýza jednoho vzorku neovlivnila analýzy vzorků dalších. Typicky se pak z tohoto systému nějakým způsobem dostává informace o dění v systému při běhu testovaného programu ven, kde jsou informace sbírány další částí sledovacího systému a poté případně dále zpracovávány.

Některé metody pro sledování chování implicitně umožňují omezení sledování na konkrétní testovaný program, u jiných metod však nutně dojde ke sledování chování celého systému. Zdálo by se, že první případ je snadnější a výhodnější pro implementaci sledovacího systému, protože není třeba filtrovat výstup sledování, jelikož získaná data se týkají pouze programu, který nás zajíma. Pokud by byl sledovací systém implementován takto jednoduše, pak by výsledky sledování pro řadu programů nebyly příliš přínosné. Je totiž běžné, že se malware snaží v systému usídlit tak, že svůj výkonný kód nějakým způsobem vloží do důvěryhodných procesů v systému a v jejich kontextu je pak tento kód vykonán. Pokud by se sledovací systém omezil pouze na jediný proces sledovaného programu, veškeré chování vykonané v kontextu jiného procesu by mu uniklo. Pro další sledování by bylo potřeba implementovat logiku sledovacího systému, která rozpozná takovou akci a začne sledovat i infikovaný proces. Problém zde je, že možností jak přesunout exekuci do jiného procesu je velmi mnoho. Na druhou stranu pokud nám jde pouze o prosté posouzení, zda testovaný vzorek vykazuje chování malware, pak nám sledování jednoho procesu postačí s tím, že pokusy o infikování jiných procesů se považují za znak chování malware, neboť jsou neobvyklé u legitimních programů.

U metod, které nutně sledují celý systém, je situace přesně opačná. Ve změti zachycených událostí při sledování je třeba vybrat ty, které mají relevanci ke sledovanému vzorku. Je tedy potřeba umět odfiltrovat události, které v systému běžně nastávají, od těch, které byly vyvolány činností sledovaného programu. Některé sledovací systémy tuto filtraci vůbec neřeší a buďto opomíjejí možnost výskytu událostí z ostatních částí systému nebo spoléhají na to, že většina událostí během sledování bude vyvolána sledovaným programem a tedy že i nefiltrovaný výstup bude z větší části relevantní. Při použití těchto metod by však bylo lepší mít dobré informace o objektech v systému a vědět, které ze sledovaných operací je mohou ovlivňovat. Pokud bychom měli přesnou představu o interakcích mezi objekty v systému, mohli bychom na výstupu systému dostat jen události související s objekty, které mohly být ovlivněny testovaným programem. Výstup by ani pak nebyl prostý nerelevantních událostí, ale jejich počet by byl výrazně menší a jejich výskyt by byl objektivně zdůvodnitelný.

3 Metody používané v sledovacích systémech bez použití virtualizace

V této kapitole rozebereme některé metody, které se dají použít pro implementaci sledovacích systémů nebo jejich částí a které nejsou založeny na virtualizaci. Nepoužití virtualizace v praxi znamená, že systém, v němž je spuštěn sledovaný program, je nainstalován přímo na reálném hardware a přímo používá reálná zařízení jako jsou např. pevné disky. U nevirtualizovaných systému vzniká problém, že testovaný vzorek může systém poškodit, což je třeba řešit, pokud chceme systém použít i pro jiné vzorky.

Podobný problém se vyskytuje i v úplně jiných oblastech použití výpočetní techniky – např. v Internetových kavárnách nebo hernách je pro provozovatele důležité udržet v provozu stanice poskytované klientům a přitom klienty co nejméně omezovat. Proto vznikla mnohá komerční i nekomerční řešení, z nichž nejznámějším je zřejmě Deep Freeze od společnosti Faronics Corporation. Jedná se o software, který restauruje počítač do uloženého stavu při každém jeho restartu. Uživatelé si tak mohou ukládat na disk a média svá data, spouštět libovolné programy, ale po restartu počítače veškeré změny a data zmizí a stanice je tak připravena pro dalšího klienta. Alespoň to slibuje propagační materiál tohoto produktu. Existují i hardwarová řešení, která při restaurování systému použijí vlastní a od reálného systému oddělenou kopii jeho původního stavu.

Následující metody jsou popsány v kontextu záměru implementace sledovacího systému. Seznam metod rozhodně není kompletní, ale měl by pokrýt naprostou většinu běžně používaných nebo teoreticky uvažovaných metod. U popisů většiny metod se snažíme pouze o krátký popis jejich principu bez zacházení do zbytečných, často velmi technických, detailů.

Řada metod funguje na principu přepisování kódu nebo dat struktur, které jsou primárně určené pouze pro čtení. V takovém případě jejich implementace musí řešit problémy spojené s ochranou paměti. Detaily těchto problémů zde nejsou rozebírány, čtenář se o nich může dozvědět více například v [2].

3.1 Hákování v uživatelském módu (user mode hooking)

Hákováním obecně se rozumí technika, při které běžná cesta vykonávání kódu (execution path) je změněna. V části 2.3 jsme uvedli, jak probíhá volání funkce, která vede na volání systémové služby. V některých úsecích lze vykonávání kódu přesměrovat a změnit tak cestu vykonávání kódu. Přesměrování se provádí za účelem exekuce vlastního kódu, který například reportuje, že nastala nějaká událost (proces se pokouší otevřít soubor). Přesměrování je však možné pouze v rámci adresového prostoru cílového procesu, proto implementace libovolné techniky hákování v uživatelském módu nutně s sebou obnáší problém umístění vlastního kódu do cílového procesu. Výjimku tvoří metoda, která využívá systémovou podporu pro ladění. U této metody se samotná systémová implementace podpory ladění stará o posílání informací o ladících událostech do procesu, který ladí cílový proces.

Všechny zde popsané metody umožňují přerušení vykonávání kódu na začátku hákované

funkce. V takovém případě po přesměrování vykonávání kódu na kód sledovací části, může sledovací část provést analýzu vstupních parametrů volání a kdykoliv zavolat původní kód pro vykonání samotné operace, kterou program vyžadoval. Poté se vykonávání kódu dostane zpět do sledovací části, kde se může provést další zkoumání například výsledku a výstupních parametrů hákované funkce a následuje návrat k volajícímu hákované funkce. Některé metody podporují i hákování uprostřed kódu funkce. V takovém případě se část původního kódu funkce vykoná před přesměrováním a kód zbytku funkce se vykoná až po návratu ze sledovací části, která se již dále ke slovu nedostane.

Všechny zde popsané metody jsou jednoduše detekovatelné sledovaným procesem a při jejich použití se lze vyhnout sledování. Navíc žádná ze své podstaty neumožňuje sledování dění v jádře.

3.1.1 Vkládané hákování (inline hooking)

Podstata této metody je v přepsání paměti, v níž se nachází kód funkce, kterou chceme sledovat. Kód se přepisuje nejčastěji instrukcí nepodmíněného skoku (JMP), která okamžitě přesměruje vykonávání kódu na novou adresu bez modifikace obsahu zásobníku nebo dalších registrů. Místo instrukce skoku lze použít instrukci volání funkce (CALL), která však navíc na zásobník přidá návratovou adresu, což je adresa instrukce následující za instrukcí volání funkce.

Další možnost je o něco komplikovanější. Namísto instrukce JMP nebo CALL se použije neplatná instrukce, která vyvolá výjimku operačního systému. Výjimku lze odchytit nainstalováním obslužné rutiny strukturované výjimky (SEH – Structured Exception Handler), provést vlastní sledovací kód, vyvolat původní kód a dále pokračovat v chodu programu. Tato metoda je pomalejší a náročnější na implementaci. V praxi není příliš používaná i proto, že k instalaci SEH může dojít i v kódu, který obaluje funkci, kterou chceme hákovat. V takovém případě by ale naše obslužná rutina nebyla zavolána. V některých případech je však možné tuto techniku použít. Kromě strukturované obsluhy výjimek nabízí Windows od verze XP i tzv. vektorovou obsluhu výjimek (Vectored Exception Handling). Z pohledu implementace sledování však má VEH prakticky stejné parametry jako SEH.

Do této kategorie hákování spadá i přepis ladící instrukcí (breakpoint). Podobně jako v případě neplatné instrukce se i zde při vykonání ladící instrukce dostane k řízení operační systém. Ten zkontroluje, zda je proces laděn jiným procesem. Pokud ne, vyvolá výjimku stejně jako u předchozí techniky. Pokud však cílový proces laděn je, systém informuje ladící proces o této události a předá mu řízení. Ladící proces má pak možnost vykonat libovolnou operaci a dále s cílovým procesem manipulovat. Lze tedy cílový proces navést do stavu, kdy se opět vykoná původní kód, jako kdyby proces nebyl laděn a hákován. Pokud je program laděn, pak všechny výjimky operačního systému jsou nejprve předány jeho ladícímu procesu a jsou doručeny pouze v případě, že to ladící proces povolí. Proto hákování pomocí ladění není omezené jen na použití ladící instrukce, ale může k němu být použitá libovolná neplatná instrukce nebo instrukce, která vyvolá výjimku.

Všechny uvedené metody mají společný problém. Přepisem instrukce tak, aby řízení dostal sledovací systém, dojde k poškození kódu. Typicky je ale po vykonání sledovacího kódu požadováno vykonání originálního kódu tak, aby se program choval, jako kdyby k žádnému hákování nedošlo. Proto je nutné před přepisem instrukcí tyto instrukce uschovat. Ke kopii těchto instrukcí je potřeba připojit instrukce, které zajistí návrat do zbytku těla funkce, které zůstalo zachováno.

Nejčastěji výsledná cesta vykonávání kódu při zavolání hákované funkce vypadá následovně:

1. Provede se skok na adresu hákované funkce a provedou se všechny instrukce až do instrukce, která vykonávání přesměrovává. Nejčastěji je přesměrovávací instrukce hned první instrukcí hákované funkce, ale není to podmínkou.
2. Následuje vykonání sledovací funkce, která může provádět libovolné úkony, typicky se jedná o analýzu vstupních parametrů hákované funkce.
3. Sledovací funkce volá úsek kódu s kopií přepsaných instrukcí.
4. Ty následuje skok na zbytek těla hákované funkce.
5. Po skončení původního kódu se řízení dostane zpět do sledovací funkce, která může provést analýzu výsledku a výstupních parametrů.
6. Řízení se navrátí zpět volajícímu hákované funkce.

Tuto metodu hákování lze použít i pro hákování uprostřed nebo i na konci hákované funkce. V takovém případě odpadá krok 5.

Může se stát, že knihovna obsahující funkci, kterou chceme hákovat, není v procesu ještě zavedená a k jejímu zavedení dojde až později. To je především případ nových procesů, které obsahují pouze knihovnu `ntdll.dll` a hlavní modul programu. Stát se to může i při plném běhu programu v případě, že si program sám načítá externí knihovnu za běhu. Pro metodu vkládaného hákování to ale není složitý problém. Stačí zahákovat některou z funkcí, které se podílejí na načtení knihovny – některé z nich jsou vždy dostupné, neboť jsou obsaženy v `ntdll.dll`, a pak jen reagovat na načtení knihovny, která nás zajímá.

Vkládané hákování s instrukcí skoku je technika velmi používaná a dá se říci, že i podporovaná výrobcem systému, neboť existuje knihovna Detours přímo od Microsoftu. Hákování ladící instrukcí je také velmi běžná technika, kterou implementují nástroje pro ladění aplikací.

Výhodou této metody je snadná implementovatelnost. K tomu napomáhá i existence řady knihoven, které nabízejí jednoduchá rozhraní pro vkládané hákování. Další podstatnou výhodou je velké množství funkcí, které lze takto hákovat. Knihovny v uživatelském režimu obsahují mnohem více funkcí než je služeb operačního systému. Řada funkcí knihoven ani služby operačního systému nevolá nebo je z jejich volání velmi komplikované zjistit skutečnou podstatu chování procesu. Hákování na úrovni funkcí uživatelského režimu proto velmi usnadňuje analýzu chování procesu.

Nevýhodou pak je možnost sledovaného programu zbavit se nainstalovaných háků jednoduše tak, že obnoví původní instrukce v hákovaných funkcích. Další možností, jak se vyhnout sledování, je přímé volání funkcí nižší úrovně. Pokud by například byla hákována funkce pro otevření souboru z `kernel32.dll`, pak je možné místo ní volat přímo funkci z `ntdll.dll` nebo i přímo vykonat instrukce pro volání systémové služby, které má otvírání souborů na starost. Je sice možné hákovat i funkce `ntdll.dll` a často se to také dělá, nicméně těmito metodami nelze hákovat přímé volání systémové služby.

3.1.2 Hákování Import Address Table (IAT hooking)

Tato metoda podporuje hákování pouze takových funkcí, které sledovaný program importuje. Jak již bylo zmíněno v úvodu, pokud program importuje nějakou knihovní funkci, je při zavedení programu také načtena knihovna do adresového prostoru procesu programu. K importovaným funkcím se zjistí konkrétní adresy v adresovém prostoru a ty se uloží do IAT. Tyto adresy v IAT lze změnit u funkcí, které chceme hákovat, a přesměrovat tak volání zvolených funkcí na sledovací kód.

Na rozdíl od vkládaného hákování se nenarušuje kód hákované funkce a je tedy jednodušší zavolat originální kód. Je však možné získat řízení pouze na začátku funkce, nikoliv uprostřed. To však většinou v běžných aplikacích nevadí.

Největším problémem této metody je její instalace do nových procesů. U již běžících procesů, které už načetly všechny knihovny obsahující funkce, které chceme hákovat, lze jednoduše přepsat hodnoty v IAT a tím dosáhnout kýžených přesměrování. Pokud však daná knihovna v procesu ještě není, nelze hákování IAT v daném okamžiku uplatnit. Takový problém vzniká především u nových procesů jak již bylo popsáno v předchozí kapitole. Implementace IAT hákování je proto nejčastěji podepřena vkládaným hákováním některé z funkcí pro načítání knihovny do procesu nebo se do procesu vloží speciální inicializační kód, který se postará o načtení všech knihoven, jejichž funkce chceme hákovat. Další možností je tzv. hybridní hákování – viz kapitolu 3.2.6.

Sledování implementované IAT hákováním lze obejít stejnými metodami jako vkládané hákování, jak bylo popsáno v kapitole 3.1.1. Navíc však lze IAT hákování obejít zcela jednoduchým trikem. Analýzou knihovny lze programově zjistit původní adresu funkce, která je hákována, a volat ji pak přímo. Při takovém volání se vyhneme IAT, čímž se zároveň vyhneme sledování. Adresu funkce lze zjistit i bez použití knihovních funkcí a tím pádem tomuto postupu nelze zabránit jen za použití hákování IAT.

Nevýhody této metody jsme právě uvedli. Výhodou hákování IAT oproti vkládanému hákování je ještě jednodušší implementace, při níž odpadá starost s narušením původního kódu hákovaných funkcí.

3.1.3 Hákování Export Address Table (EAT hooking)

Tato metoda je velmi podobná hákování IAT, ale její obcházení je trochu komplikovanější. Namísto změn adres v IAT u modulu, který hákovanou funkci importuje, se změní adresa v EAT modulu, který funkci exportuje. Když se při načítání knihovny do adresového prostoru běžným způsobem hledá adresa importované funkce, jako výsledek hledání se označí právě adresa v EAT. Pokud je tato přesměrována na sledovací funkci, bude sledovací funkce volána namísto funkce původní. A to jak při načítání knihovny při inicializaci procesu, tak i při případném ručním načítání za běhu procesu.

EAT je třeba modifikovat ihned při načtení knihovny do adresového prostoru procesu ještě předtím, než se provede algoritmus importu. Vzniká tak podobný problém jako u hákování IAT. Opět se nejčastěji řeší buďto speciálním inicializačním kódem pro načtení všech potřebných knihoven nebo hybridním hákováním.

Vlastnosti této metody jsou stejné jako u hákování IAT, pouze nebude fungovat uvedený jednoduchý trik na obcházení hákování, jelikož běžným algoritmem pro získání adresy funkce v knihovně získáme právě adresu sledovací funkce. Existuje však složitější způsob vyhledání originální adresy funkce, kterým pak lze hákování EAT obejít. Postačí namísto analýzy knihovny načtené do adresového prostoru procesu analyzovat binarní obraz knihovny na disku. Taková analýza je složitější než analýza již zavedené knihovny v paměti, ale je možné ji spolehlivě implementovat a obejít tak i hákování EAT.

3.1.4 Hákování zpráv oken (windows hooks)

Grafická rozhraní většiny uživatelských aplikací s grafickým rozhraním na Windows jsou ovládány přes tzv. okenní zprávy (windows messages). Při vytváření a destrukci okna, kreslení částí okna (tlačítek, vstupů, seznamů apod.), uživatelském vstupu do okna (pohyb

nebo klikání myší, vstup z klávesnice), změn rozměrů a vlastností okna a dalších operací s grafickým rozhraním se cílové aplikaci posílají právě okenní zprávy, na které aplikace reaguje příslušnou akcí. Přímo v systému je v tomto rozhraní podporováno hákování zpracovávání těchto zpráv, které je možno využít například pro sledování dění v rámci grafického rozhraní aplikace. Tato metoda je omezena pouze na grafické rozhraní a nelze pomocí ní sledovat jiné aspekty chování aplikací.

Implementace pro sledování dění v cizích aplikacích vyžaduje implementaci dynamické knihovny, která je namapována do adresového prostoru cílového procesu. K instalaci háku pak slouží funkce **SetWindowsHookEx**.

3.2 Hákování v jádře (kernel mode hooking)

Již jsme zmínili, že jádro operačního systému obsahuje různé části. Ty implementují a nabízejí řadu funkcí stejným způsobem jako knihovny uživatelského módu. Proto je možné využít většiny technik, které jsme uvedli v kapitole 3.1 i v režimu jádra. Vkládané hákování, hákování IAT i EAT budou fungovat. Pouze jejich vlastnosti jsou v režimu jádra odlišné.

Nejdůležitějším rozdílem je podpora ze strany výrobce systému. Zatímco hákování v uživatelském režimu se dá považovat za podporované, v režimu jádra se podobné techniky zakazují. Na 64-bitových platformách jsou zákazy dokonce vynuceny systémovou komponentou PatchGuard, která periodicky kontroluje integritu jádra a případné modifikace v něm řeší okamžitým pádem systému. Tím se Microsoft snaží odradit tvůrce software od nedokumentovaného modifikování obsahu jádra včetně používání většiny technik popsaných v této kapitole.

Již popsanou nevýhodou metod hákování v uživatelském režimu je jejich detekovatelnost a možnost obcházení háků. Lze si tak představit program, který se bude úmyslně chovat jinak, pokud zjistí, že je hákován. Tato nevýhoda téměř odpadá u většiny metod popsaných v této kapitole neboť detekce hákování v jádře je bez použití ovladače poměrně složitá a zpravidla lze technikám detekce i efektivně zabránit. Většinu technik v této kapitola navíc nemůže kód v uživatelském módu obejít, případně se tomu opět dá zabránit. Pokud však sledovaný software použije ovladač nebo jiný způsob pro vykonání kódu v režimu jádra, pak se stává detekce i obcházení opět snadnými.

Další rozdíl je v tom, že pomocí vkládaného hákování, hákování IAT nebo EAT v jádře lze sledovat dění v jádře. Ne každá technika hákování v jádře toto umožňuje. Zajímavé také je, že všechny další metody popsané v této kapitole lze nahradit vkládaným hákováním. Například techniky modifikace různých ukazatelů na funkce se snadno nahradí vkládaným hákováním začátku původní funkce, na kterou odkazoval ukazatel.

Následuje popis technik hákování v jádře nebo z jádra. Software implementující takové hákování nutně používá ovladač, jehož kód je vykonáván v režimu jádra a má tedy přístup k paměti a kódu jádra.

3.2.1 Hákování System Service Descriptor Table (SSDT hooking)

Do této metody patří různé modifikace struktur týkající se dvou tabulek služeb jádra. Systémové služby lze rozdělit do dvou kategorií – hlavní služby poskytované hlavním modulem jádra (nejčastěji `ntoskrnl.exe` nebo `ntkrnlpa.exe`) a služby grafického rozhraní (GDI – Graphics Device Interface). Rozhodně důležitější částí pro sledování chování programu je první tabulka s hlavními systémovými službami. Volání těchto služeb je

zapouzdřeno knihovnou `ntdll.dll` v uživatelském režimu. Sledování práce programu s různými grafickými objekty a okny není až na výjimky z hlediska bezpečnosti příliš zajímavé. Volání služeb grafického rozhraní zapouzdřují systémové knihovny `user32.dll` a `gdi32.dll`.

SSDT popisuje tabulky aktuálně využívané vláknem. Systém udržuje dvě verze SSDT. První verze je pro vlákna, která nevyužívají služeb grafického rozhraní, druhá verze pro vlákna, která grafické rozhraní využívají. První verze SSDT obsahuje pouze popis pro hlavní služby – ukazatel na tabulku služeb, počet služeb a velikost jejich parametrů. Druhá verze, tzv. SSDT Shadow, obsahuje popis jak pro hlavní služby, tak pro služby grafického rozhraní.

Pro pochopení možností při hákování SSDT je potřeba lépe porozumět principu aplikačního volání systémových služeb. K volání systémové služby slouží speciální instrukce. U starších procesorů a verzí Windows se používala instrukce přerušení (INT 2Eh), u novějších se pak z důvodu zrychlení volání používá speciální instrukce SYSENTER nebo SYSCALL – dle architektury. Nastavení hodnot registrů procesoru pak určuje číslo volané služby, číslo tabulky služeb, ke které se číslo služby vztahuje, a umístění argumentů služby. Volající vlákno je pak vykonáním instrukce pro volání služby přepnuto do režimu jádra.

Ke každému vláknu si jádro udržuje řadu informací. Jednou z nich je i příznak využívání služeb grafického rozhraní. Na 32-bitových verzích navíc je součástí této informace i ukazatel na jednu z dostupných SSDT, kterou vlákno právě využívá. Vlákna, která využívají služeb grafického rozhraní, mají oproti ostatním vláknům navíc frontu zpráv, která slouží právě při práci okny a ostatními grafickými objekty. Pokud vlákno nemá nastaven příznak využívání služeb grafického rozhraní, jeho ukazatel na SSDT ukazuje na první SSDT. V opačném případě odkazuje na SSDT Shadow, která popisuje i tabulku služeb grafického rozhraní. Pokud vlákno nemá příznak využívání služeb grafického rozhraní nastaven, pak se mu tento příznak nastaví při prvním volání libovolné služby grafického rozhraní a při té příležitosti se mu změní i ukazatel na SSDT. Binární reprezentace programu v sobě může nést informaci o tom, že program bude využívat služeb grafického rozhraní. V takovém případě je hlavnímu vláknu programu nastaven příznak využívání služeb grafického rozhraní ihned při jeho vytvoření.

Při vykonání instrukce pro volání systémové služby se vezme ukazatel na SSDT, dle čísla tabulky služeb se vyhledá příslušná tabulka a na základě čísla funkce služby se v ní dohledá ukazatel na funkci, která službu implementuje. Cílová funkce se zavolá a po jejím dokončení se výsledek vrací zpět volajícímu s přepnutím do uživatelského módu.

Nejpoužívanější technika hákování SSDT mění ukazatele v první nebo druhé tabulce služeb. Tím přesměrovává volání vybraných služeb na vlastní kód, který typicky provede analýzu vstupních parametrů volání. Poté se volá originální kód pro vykonání skutečné funkce služby. Po jeho dokončení může sledovací kód provést analýzu výsledku a výstupních parametrů a nakonec vrátí řízení zpět systému, který vrací řízení volajícímu služby.

Další varianty hákování SSDT vedou na tentýž efekt, jen se použije jiné technické provedení. SSDT obsahuje ukazatele na tabulky služeb. Jedna z možností je vytvořit kopii tabulky služeb a změnit ukazatel v SSDT. Modifikace tak lze provádět ve vlastní kopii, na kterou je ukazatel v SSDT přesměrován. Jinak je princip stejný. Tato metoda má stejné vlastnosti jako prvně zmíněná metoda. Další variantou je změna ukazatele na SSDT ve struktuře jádra obsahující informace o vláknu. U této varianty je potřeba změnit informaci u všech vláken procesů, které chceme sledovat. Ukazatel na SSDT se změní tak, aby odkazoval na kopii SSDT, která obsahuje ukazatel na kopii tabulky služeb, ve které jsou modifikované záznamy adres funkcí, které chceme hákovat. Výhoda této metody je v možnosti zvolit si vlákna, kterých se

hákování týká. Nevýhoda je pak v tom, že je třeba dávat speciální pozor na procesy, jejichž programy nejsou označeny jako programy využívající grafické rozhraní a přesto při svém běhu služby grafického rozhraní používají. V takovém případě totiž systém ukazatel na SSDT ve struktuře vlákna přepíše a tím přestanou být naše modifikace aktivní.

Hákování SSDT se týká volání služeb, které jádro poskytuje aplikacím uživatelského režimu. Tyto služby mohou být používány i v jádře, ale v takovém případě se často volají přímo bez použití SSDT. Z toho důvodu tato metoda neumožňuje kvalitní sledování dění v jádře.

3.2.2 Hákování Model Specific Register (MSR hooking)

MSR je kolekce registrů procesoru, která umožňuje nastavení nejrůznějších vlastností, které jsou specifické pro daný model procesoru. V módu jádra lze MSR číst i zapisovat pomocí speciálních instrukcí (RDMSR, WRMSR).

Na Windows XP a vyšších se pro přístup k službám jádra používají instrukce SYSENTER nebo SYSCALL. SYSENTER je jméno instrukce u procesorů Intel, SYSCALL náleží procesorům AMD. Jejich funkce je totožná, drobné odlišnosti jsou pouze v nastavení registrů při jejich používání. Následující popis je platný pro SYSENTER.

Tři registry MSR ovlivňují zpracování instrukce SYSENTER – SYSENTER_CS_MSR, SYSENTER_ESP_MSR a SYSENTER_EIP_MSR. Poslední jmenovaný udává adresu obsluhy systémových volání, která, jak již bylo popsáno v předchozím textu, na základě čísla tabulky systémových služeb a čísla služby zavolá funkci, která službu implementuje. Změnou SYSENTER_EIP_MSR lze získat řízení po zavolání instrukce SYSENTER. Náš kód pak může implementovat volání systémové služby nebo volání sledovací funkce pro služby, které chceme hákovat. Výsledný efekt a vlastnosti jsou pak stejné jako v případě první popsané metody hákování SSDT. Tato metoda není příliš používána, jelikož její technické provedení je náročnější než hákování SSDT a nejsou zde žádné výhody pro sledovací systémy.

Hákování MSR se opět týká služeb jádra pro uživatelské aplikace. Stejně jako u hákování SSDT tedy není možno touto metodou sledovat dění v jádře. Navíc je možno použít alternativní přístup k systémovým službám přes přerušení a tím se vyhnout sledování založeném na hákování MSR.

3.2.3 Hákování Interrupt Descriptor Table (IDT hooking)

Tabulka vektorů přerušení (IDT) obsahuje informace o obslužných rutinách pro jednotlivá přerušení procesoru. Každý procesor počítače má vlastní kopii IDT, kterou používá. Některá přerušení mohou být zajímavá z hlediska implementace sledovacích systémů nebo jejich částí.

U starších verzí Windows (2000) a starších procesorů se namísto instrukcí pro volání systémové služby SYSENTER a SYSCALL používala instrukce přerušení 2Eh. Každé přerušení má svůj záznam v IDT, který udává pozici obslužné rutiny. Je možné změnit tento záznam a vlastním kódem obsluhovat zvolená přerušení – například 2Eh. Tím získáme podobné možnosti jako u hákování MSR. Opět se nejedná o příliš používanou metodu pro zachytávání volání systémových služeb ze stejných důvodu jako v případě hákování MSR. Pokud navíc procesor podporuje instrukci SYSENTER nebo SYSCALL, tak lze hákování přerušení 2Eh obejít. Při hákování IDT je vždy potřeba instalovat háky na všechny procesory.

Kromě přerušení 2Eh jsou zajímavé i další přerušení, které však mohou mít různé indexy na různých počítačích. Přerušení generují například zařízení USB, zařízení PS/2, disky, síťové

karty apod. Operace těchto zařízení mohou být předmětem sledování, ale jedná se o velmi nízkoúrovňový přístup k problému, pro jehož použití se těžko hledají argumenty, neboť je technicky náročné a velmi obtížně se operace zařízení spárují s procesy, jejichž chování k operaci vedlo. Navíc v případě nutnosti volání původní obsluhy přerušení nezíská nová obsluha řízení po dokončení původní obsluhy, protože ta přerušení dokončí a vykonávání kódu pokračuje tam, kde bylo před přerušením.

3.2.4 Hákování I/O Request Packet (IRP hooking)

Systém obsahuje řadu ovladačů. Ty pro komunikaci s okolím mohou použít tzv. zařízení (devices). Zařízení mohou reprezentovat skutečný hardware počítače, nebo to mohou být čistě virtuální zařízení, která jsou vytvořena ovladači právě z důvodů komunikace s okolím. Na zařízení spravované jedním ovladačem lze navázat jiné zařízení, které spravuje nějaký jiný ovladač, a tím docílit toho, že požadavky na jedno zařízení budou spravovány více ovladači, jejichž zařízení tvoří řetězec. Požadavkům v tomto kontextu se říká IRP.

Každý ovladač je reprezentován objektem, jehož součástí je i seznam funkcí, které ovladač na zařízení obsluhuje. Vznikne-li požadavek na zařízení, je předán ovladači, který jej obsluhuje. Ovladač požadavek zpracuje a ten pak přechází na další zařízení v řetězci. Existence řetězců zařízení umožňuje redukovat složitost ovladačů. Také to umožňuje implementovat novou funkcionalitu do jádra bez nutnosti měnit zavedené ovladače.

Ovladač, který již zpracoval požadavek, nezíská řízení zpět, ledaže by v požadavku nastavil tzv. dokončovací funkci (IRP completion routine). Dokončovací funkce bude zavolána, když ovladač zařízení následující v řetězci dokončil zpracování požadavku. Ovladač posledního zařízení v řetězci dokončovací funkci nastavit nemůže.

U každého ovladače je pak možno hákovat libovolnou funkci, kterou obsluhuje zařízení, jednoduše tak, že ukazatel na původní obsluhu v tabulce funkcí objektu ovladače nahradí ukazatelem na sledovací funkci. Výsledky požadavku se pak dají získat pomocí implementace vlastní dokončovací funkce. Je zde však jeden problém. Pokud je volána původní funkce, může ta nastavením dokončovací funkce přepsat nastavenou dokončovací funkci sledovací funkce. Vzhledem k tomu, že po zavolání původní se již sledovací funkce nedostane ke slovu, nebude dokončovací funkce nastavená sledovací funkcí nikdy zavolána. Analýzu výstupu tak je možno provádět jen pokud původní funkce ovladače nepoužívá dokončovací funkci.

Hákování IRP lze použít pro sledování přístupů k disku, k síťovým a dalším zařízením. Jedná se tedy o možnosti podobné těm, které nabízí hákování IDT s tím podstatným rozdílem, že k hákování dochází na vyšší úrovni v kontextu vlákna, které požadavek zpracovává.

Podobnou funkcionalitu lze implementovat i pomocí filtrování – viz kapitolu 3.3, což je, na rozdíl od hákování IRP, podporovaný způsob od výrobce systému. Pro implementaci sledovacího systému tak hákování IRP nemá smysl používat, kromě velmi specifických případů, které podporované metody filtrování nepokryjí. Navíc nelze obecně zaručit možnost analýzy výstupu zpracování požadavků.

Touto metodou se přirozeně sledují i požadavky, které mají původ v jádře.

3.2.5 Hákování objektů jádra (Kernel object hooking)

Každý objekt v jádře má přiřazen svůj typ, který je opět reprezentován objektem. Typové objekty obsahují informace společné pro všechny objekty daného typu. Mezi ně patří

i ukazatele na funkce, které jádro volá v různých specifických situacích. Jedná se o následující funkce:

- DumpProcedure – Nepoužívá se.
- OpenProcedure – Systém volá tuto funkci, když se otevírá objekt daného typu.
- CloseProcedure – Systém volá tuto funkci, když se zavírá handle objektu daného typu.
- DeleteProcedure – Systém volá tuto funkci, když již nikdo objekt daného typu nepoužívá a tento může zaniknout.
- ParseProcedure – Systém volá tuto funkci, když je potřeba najít konkrétní objekt daného typu na základě jeho jména.
- SecurityProcedure – Systém volá tuto funkci, když se nastavuje nebo zjišťuje nastavení bezpečnosti objektu daného typu.
- QueryNameProcedure – Systém volá tuto funkci, když je k objektu daného typu potřeba zjistit jeho jméno.
- OkayToCloseProcedure – Systém volá tuto funkci, aby zjistil, zda je povoleno objektu daného typu zaniknout.

Není však povinností typového objektu všechny tyto funkce implementovat. Opět lze ukazatele na tyto funkce přesměrovat na funkce sledovacího systému a zachytit tak například všechny pokusy o otevření objektu apod. Analýza výstupních hodnot zde není problém, neboť při zavolání původní funkce se řízení vrací do sledovací funkce.

I tato metoda přirozeně zachytí operace nezávisle na tom, zda mají původ v uživatelském režimu nebo v jádře.

3.2.6 Hybridní hákování (hybrid hooking)

Hybridním hákováním se myslí využití síly jádra k implementaci hákování v uživatelském režimu – ať už vkládaného hákování, hákování IAT nebo EAT. V popisu těchto metod jsme zmínili problém, který nastává, pokud chceme hákovat funkci z knihovny, která ještě není do procesu načtená. Elegantně lze toto vyřešit v jádře, které přímo nabízí možnost zaregistrovat funkci (voláním **PsSetLoadImageNotifyRoutine** nebo **PsSetLoadImageNotifyRoutineEx**), která bude zavolána, když nějaký proces načte knihovnu nebo jiný modul do svého adresového prostoru.

Kvůli jednoduchému řešení zmiňovaného problému je hybridní hákování často používáno u aplikací, které implementují hákování v uživatelském režimu.

3.3 Filtrování

Uvedli jsme, že hákování v jádře jsou nedokumentované a nepodporované metody. Jejich možnosti jsou však obrovské. Lze je nějak nahradit pomocí dokumentovaných metod? Odpověď pro některé operace v systému přináší filtrování. Základní princip filtrování je podobný jako u hákování – v definovaných okamžicích při běhu systému se vykonávání kódu přesměruje na filtrovací funkci. Ta může analyzovat stav operace a případně i ovlivnit další pokračování operace. U některých operací je možno získat řízení i po dokončení a analyzovat výsledný stav. Realizace filtrování je však naprosto odlišná od implementace hákování.

Díky zabudované podpoře filtrování v částech systému, které jej poskytují, je filtrovacím

funkcím často poskytováno poměrně velké množství informací o operaci. U hákování jsou k dispozici vždy pouze argumenty hákované funkce. Další informace je potřeba dohledat a to nemusí být vždy snadné.

Již bylo zmíněno, že v systému existují zařízení spravovaná ovladači. Tato zařízení mohou být spolu provázána v řetězcích a takto tvořená hierarchie je základem některých filtrovacích metod. Ovladač může vytvořit vlastní zařízení, které naváže s již existujícím (pomocí funkce **IoAttachDevice**) a tím docílí, že požadavky zaslané na navázané zařízení budou poslány ke zpracování i jemu. To je základ všech metod filtrování na zařízeních. Při implementaci filtrování čistě za pomocí **IoAttachDevice** se však programátor musí potýkat s řadou problému, které se stále opakují a jsou častým zdrojem chyb. Z tohoto důvodu Microsoft pro určité oblasti systému implementoval dodatečné mechanismy, které zjednodušují psaní filtrovacích ovladačů. Z metod popsaných dále v této kapitole se to týká filtrování souborového systému a části filtrování síťové komunikace. Zpravidla se pro řešení problému používá zjednodušená varianta, pokud je dostupná.

3.3.1 Filtrování souborového systému

V systému je zaveden filtrovací ovladač pojmenovaný Filter Manager, který implementuje filtrování pro souborové systémy. Filter Manager zavádí koncept tzv. minifilter ovladačů, který zjednodušuje filtrování souborových systémů. Zjednodušení je poskytováno přes speciální funkce, které Filter Manager nabízí. Ovladač, který chce implementovat filtrování, se zaregistruje jako minifilter u Filter Manageru a vybere zařízení, která chce filtrovat. Registrace se provádí voláním **FltRegisterFilter**.

Filter Manager umožňuje definovat pozici ovladače mezi všemi registrovanými minifilter ovladači. Při obdržení požadavku na souborový systém Filter Manager postupně informuje registrované ovladače, dle jejich registrované pozice, o události volitelně před jejím vykonání nebo i po jejím vykonání. Ovladač, který právě zpracovává informaci o události, má možnost ovlivňovat další zpracování události například zamítnutím požadavku a vrácením chyby volajícímu nebo pozměněním parametrů volání.

Filtrování přes Filter Manager je jednoduché na implementaci díky široké podpoře funkcí, které Filter Manager nabízí, a také díky velkému množství informací, které jsou jím poskytovány u každé události. Toto filtrování přirozeně zachycuje požadavky mající původ v uživatelském režimu i v jádře. Filtrování lze obejít pouze z režimu jádra při použití přímých požadavků na nižší vrstvy souborového systému. Metoda přirozeně podporuje analýzu jak vstupních parametrů operací, tak i jejich výsledků.

3.3.2 Filtrování registrových operací

O registry systému se stará komponenta s názvem Configuration Manager. Na rozdíl od file systému, registr není navržen jako zařízení spravované ovladačem a nelze tedy uplatnit obecný koncept filtrování. Configuration Manager ale implementuje filtrování operací s registry, které se používá velmi podobně jako filtrování nabízené Filter Managerem.

Ovladač, který chce filtrovat operace s registry, se zaregistruje u Configuration Manageru (k tomu slouží funkce **CmRegisterCallback** nebo **CmRegisterCallbackEx**). Opět lze při registraci definovat pozici vůči ostatním ovladačům. Configuration Manager pak při libovolné operaci s registry postupně informuje registrované ovladače a to jak před vykonáním samotné operace, tak i po jejím vykonání. I zde má ovladač možnost ovlivnit to, zda požadavek bude dále zpracováván nebo zda a jak bude předčasně dokončen.

Vlastnosti této metody jsou podobné jako vlastnosti filtrování souborového systému přes Filter Manager. Implementace je snadná, podporuje analýzu na vstupu i na výstupu, zachycuje i dění v jádře, je plně podporovaná a dokumentovaná.

3.3.3 Filtrování síťového provozu

Návrh síťové komunikace v systému obsahuje celou řadu komponent, které jsou provázané v hierarchii. Aplikace přistupující k síti má na výběr z několika různých knihoven v uživatelském režimu. Pro vysokou úroveň abstrakce použije například knihovnu WinInet. Nižší úroveň poskytuje knihovna Windows Sockets, tu využívají knihovny vyšší vrstvy, ale aplikace ji může použít i přímo. Další vrstvou v hierarchii je vrstva obsluhující služby Windows Sockets v jádře. Ta dále používá vrstvu Transport Driver Interface (TDI), která je již reprezentována zařízením a ovladačem a ve které se vytvářejí síťové požadavky (IRP). Vrstva TDI používá rozhraní poskytované další vrstvou s názvem Network Driver Interface Specification (NDIS), která odstiňuje specifika prostředí jádra Windows a nabízí jednotná rozhraní pro transportní vrstvu a pro ovladače síťových zařízení. Poslední vrstvou je síťový hardware a jeho ovladače.

V této hierarchii je možné filtrovat síťový provoz na více místech. Nejčastější jsou filtrovací ovladače na vrstvě TDI a NDIS (TDI Filter Driver a NDIS Filter Driver). Základním principem filtrování na vrstvě TDI je obecná metoda filtrování na zařízení popsaná v úvodu kapitoly 3.3. Pro implementaci filtrování na vrstvě NDIS je k dispozici několik typů ovladačů vlastní sada funkcí, speciální názvosloví atd.

Další možnost filtrování nabízí vrstva Windows Sockets v uživatelském režimu. Ta se sama skládá z několika dalších vrstev, z nichž jednu tvoří vrstva poskytovatelů transportních služeb (Transport Service Providers). Do této vrstvy lze zapojit vlastní knihovnu, která poskytuje potřebné funkce, přes rozhraní poskytovale služeb (Service Provider Interface – SPI). Po registraci poskytovatele bude knihovna Windows Sockets používat pro realizaci základních funkcí, jako jsou například **connect** nebo **accept**, právě funkce implementované novým poskytovatelem. Podobně lze implementovat vlastního poskytovatele funkcí související s jmenným prostorem v síti (namespace service provider).

Výše uvedený návrh se lehce pozměnil s příchodem Windows Vista, kde byla odbourána vrstva TDI. Nově naopak přibyla v jádře komponenta Winsock Kernel, která umožňuje kódu v jádře přistupovat k síti podobně, jako to umožňují knihovny Windows Sockets v uživatelském režimu. Dále v uživatelském režimu přibyla vrstva Windows Socket Switch, která leží mezi vrstvou Windows Sockets v uživatelském režimu a vrstvou obsluhující služby Windows Sockets v jádře. Nejvetší změnou je pak zavedení Windows Filtering Platform, což je komponenta, která prochází mnoha vrstvami celé síťové architektury a umožňuje jednoduché a efektivní filtrování síťového provozu na různých vrstvách.

3.4 Další podpora ze strany operačního systému

V této části uvedeme dvě další metody, které jsou oficiálně podporovány a dokumentovány.

3.4.1 Notifikační funkce

Notifikační funkce slouží ke sledování potenciálně zajímavých událostí v systému. Všechny jsou dostupné v režimu jádra a některé z nich jsou pomocí systémových služeb poskytovány i aplikacím v uživatelském režimu. Systém nabízí funkce, pomocí kterých si aplikace

nebo ovladač registrují vlastní funkci, kterou pak systém v přesně definovaném okamžiku zavolá.

Notifikace dostupné v uživatelském režimu umožňují zaregistrovat změny například v adresáři souborového systému nebo klíči registru. Bohužel není pomocí nich možné zjistit původce změny a proto jsou pro účely implementace sledovacího systému nedostatečné.

Zajímavé z hlediska sledování chování procesů jsou tedy pouze některé notifikace v jádře. Konkrétně se jedná o možnosti sledování vzniku nových procesů, vzniku nových vláken a načítání knihoven a modulů do paměti procesu nebo do systému. Ovladač pro tyto notifikace registruje svoje funkce pomocí volání **PsSetCreateProcessNotifyRoutine**, **PsSetCreateProcessNotifyRoutineEx**, **PsSetCreateThreadNotifyRoutine** nebo **PsSetLoad ImageNotifyRoutine**. Notifikační funkce registrované těmito funkcemi jsou vykonávány systémem v kontextu vlákna, které danou operaci zapříčinilo, a jsou jim dispozici všechny potřebné informace o provedené operaci.

Notifikační funkce pro sledování vzniku procesů a vláken se týkají prakticky pouze operací jejichž původ je v uživatelském režimu. Sledování načítání knihoven a modulů se však týká i operací vykonaných v jádře.

3.4.2 Podpora ze strany Object Manageru

Od verzí Windows Vista Service Pack 1 a Server 2008 implementuje Object Manager mechanismus, který se dá přirovnat k filtrování registrů poskytovaným Configuration Managerem. Ovladač může pomocí funkce **ObRegisterCallbacks** zaregistrovat funkci, která bude volána v případě nějaké operace nad objekty v systému. Při registraci je možno zvolit pozici vůči ostatním registrovaným funkcím. Registrovat lze dva druhy funkcí. První jsou volány před samotnou událostí a je v nich možné analyzovat vstupní parametry a případně je změnit. Druhý typ funkcí je volán po provedení operace, což znamená možnost provést analýzu výsledků operace.

Bohužel v aktuálně dostupné verzi jsou podporovány pouze operace nad dvěma typy objektů – procesy a vlákny. Navíc u těchto objektů se dají zachytit pouze operace otevření objektu a duplikace handle od otevřeného objektu. Nejedná se tedy o nijak silný nástroj pro sledování. Metoda přirozeně umožňuje sledovat dané operace i pokud jejich původ je v jádře.

4 Metody používané v sledovacích systémech s použitím virtualizace a podobných metod

V této kapitole rozebereme metody při nichž se analýza vzorku provádí ve speciální instanci systému, která běží ve virtualizovaném prostředí. Takový systém nemá přímý přístup k zařízením a hardwaru počítače, ale tato zařízení jsou mu k dispozici prostřednictvím virtualizačního software. Virtualizační software má za úkol zcela odstínit virtualizovaný systém tak, aby nedocházelo k neočekávaným změnám v hlavním systému. Vyspělá virtualizační řešení, která jsou dnes běžně k dispozici, umožňují navíc efektivní správu virtualizovaných systémů tak, že lze jednoduše a automatizovaně uložit stav systému před analýzou nějakého vzorku, poté spustit analýzu ve virtualizovaném systému a po jejím skončení obnovit systém do předchozího stavu. Dnes již existují desítky virtualizačních řešení, většinou však fungují na stejných principech. Popisované produkty byly voleny s ohledem na možnost využití pro implementaci sledovacího systému pro vzorky běžící na systémech Windows, jak již bylo řečeno v úvodu práce. Opět se snažíme o pouhé přiblížení principů popisovaných metod, ne o obsažení všech jejich detailů.

Při použití virtualizace je samozřejmě možné použít všechny metody, které jsme uvedli v minulé kapitole. Místo reálného systému použijeme systém virtualizovaný a v něm pak můžeme aplikovat libovolnou metodu bez virtualizace. Díky virtualizaci jsou zde však ještě další možnosti, z nichž některé vybrané jsou popsány v této kapitole.

Speciální metodě nazvané oddělení (sandboxing) se věnuje kapitola 4.4. Tato metoda nevyužívá virtualizaci, ale její vlastnosti jsou z hlediska možností odstínění analyzovaného vzorku a sledování jeho chování virtualizaci blízké.

4.1 Emulace – Bochs, QEMU

Emulace je druh virtualizace, při které nedochází k přímému vykonávání instrukcí procesorem. Bochs je emulátor procesoru Intel x86, běžných zařízení pro vstup a výstup a BIOSu. Podporuje sadu instrukcí 386, 486, Pentium I až 4, x86-64, MMX, SSE a 3DNow!. Pomocí tohoto emulátoru lze spouštět řadu operačních systému včetně variací UNIXu, Linuxu, DOSu, Windows verze 95, 98, NT4, 2000, XP a Vista.

Výhoda emulace na rozdíl od virtualizace s přímým vykonáváním instrukcí je v tom, že hostující systém může být na libovolné architektuře. Bochs tak snadno umožňuje emulovat architekturu x86 na libovolném procesoru – aktuálně podporovány jsou procesory x86, PPC, Alpha, Sun a MIPS – bez speciální podpory ve zdrojových kódech pro jednotlivé hostující procesory. Kdekoliv lze přeložit zdrojový kód Bochsu napsaný v C++, tam jím lze emulovat architekturu x86. Nevýhodou při srovnání s virtualizací s přímým vykonáváním instrukcí je pak ve výkonu, kterého emulovaný stroj dosahuje. Protože Bochs musí zpracovat každou jednotlivou instrukci sám softwarově, je logické, že jeho výkon bude rapidně nižší než je výkon virtualizačních technologií, které částečně nechávají instrukce virtualizovaného stroje vykonávat přímo na hostujícím procesoru.

Bochs zprostředkovává emulovaný stroj v grafickém náhledu, jehož hlavní část zobrazuje grafický výstup systému. Uživatelský vstup nad tímto náhledem (myš, klávesnice) je přeposílán do emulovaného stroje. Operační paměť emulovaného stroje udržuje Bochs v adresovém prostoru svého procesu, tj. v operační paměti hostujícího stroje. Pevný disk je emulován jako soubor v hostujícím systému. Data uložená do operační paměti nebo na disk mají stejnou volatilitu jako u normálního počítače – operační paměť data neuchová mezi dvěma běhy, pevný disk ano. Bochs zvládne i emulaci síťového hardware, ale tam již je implementace závislá na hostujícím systému. Emulace síťové karty využívá síťovou kartu hostitele. Pakety poslané na emulovanou síť jsou tak poslány do skutečné sítě.

4.1.1 Analogie vkládaného hákování

Plná kontrola nad emulovaným strojem je silným nástrojem pro možnosti sledování. Bochs má v každém okamžiku plný přístup k operační paměti emulovaného stroje. Navíc sám implementuje každou instrukci procesoru. Úpravou kódu Bochse bychom tak snadno mohli dosáhnout možnosti přerušit vykonávání instrukcí emulovaného stroje v libovolném okamžiku, analyzovat jeho paměť a pokračovat dále ve vykonávání instrukcí. Lze si představit snadnou definic podmínek pro přerušení jako například rovnost registru EIP (adresa vykonávané instrukce – instruction pointer) s adresou vybrané funkce v jádře. Tím bychom získali možnosti podobné technikám vkládaného hákování ať už v jádře nebo uživatelském režimu. Je zde však několik zásadních odlišností.

Na rozdíl od klasického hákování, při přerušení a analýze emulovaného systému není úplně snadné vykonat nějaký kód mimo původní cestu vykonávání kódu. Nejprve bychom museli zvolit nějaké vlákno, které nám funkci zavolá. Zde jednoznačně nejjednodušší varianta je využití vlákna, které má vykonat instrukci, jež způsobila přerušení. To také odpovídá klasickému hákování. Podle typu funkce, kterou bychom chtěli uměle zavolat, a podle aktuálního kontextu přerušeného kódu bychom pak museli uložit stav registrů (například na zásobník), jejichž hodnoty by volaná funkce mohla změnit a dále na zásobník a do registrů vložit argumenty pro volanou funkci.

Abychom z umělého volání funkce něco měli, potřebovali bychom alokovat paměť pro výsledek takové funkce, který bychom si pak mohli přečíst. Pro funkce, jejichž výsledek je relativně malý, bychom opět mohli použít zásobník přerušeného vlákna. U větších bychom pak potřebovali vlastní paměť v emulovaném prostředí. Jednotlivé volání již existujících funkcí nám navíc neposkytuje komfort, který bychom u takové analýzy potřebovali. Proto bychom raději měli možnost vložit do emulovaného systému vlastní kód a ten pak volat výše popsanou metodou. Zde už se bez vlastního kusu paměti v emulovaném prostředí neobejdeme, ledaže bychom chtěli vymýšlet obskurní interpretace funkcí bez využití interní paměti emulovaného stroje.

Alokace paměti v emulovaném systému je opět otázkou volání funkcí, takže i tuto část lze zvládnout, i když se tím již celá situace opět komplikuje. Máme tedy představu, která nám umožňuje čistě emulačními prostředky zastavovat vykonávání kódu a provádět analýzu jak zvenčí tak zevnitř emulovaného systému. Některé dílčí úkony by zde však mohli být zjednodušeny, pokud bychom do systému nezasahovali pouze z emulátoru. Máme rozumnou možnost implementovat vlastní komponentu, například ovladač, kterou nainstalujeme do emulovaného systému a která se nám postará o alokace paměti a implementuje i podporu pro jinak kostrbaté volání funkcí při analýze.

Dokonce lze velmi snadno zařídit jistou možnost komunikace mezi komponentou, jež nainstalujeme do emulovaného systému a kódem, kterým modifikujeme emulátor. Emulátor tím, že sám zpracovává všechny instrukce, přímo vybízí k možnosti implementace

vlastních speciálních instrukcí, které jinak nemají pro emulovaný procesor žádný smysl.

Je pouze na nás, jak moc a zda vůbec budeme zasahovat do paměti emulovaného stroje, zda budeme dovnitř instalovat vlastní komponentu a případně kombinovat emulaci s dalšími možnými technikami. Při rozhodování může hrát roli to, že různé zásahy do běhu stroje mohou napomoci detekci sledovacího systému. Pokud je emulátor dobře implementován, emulovaný systém nemůže poznat, zda běží na reálném hardware či nikoliv. To znamená, že ani sledované vzorky tuto šanci nemají. Jakmile ale začneme implementovat nové instrukce, instalovat nové komponenty apod., utváří se možnosti pro detekci našeho sledovacího systému.

4.1.2 QEMU

QEMU je otevřený emulátor procesoru s podporou mnoha hostujících i virtualizovaných systémů a procesorových architektur. Vlastnosti emulátoru QEMU jsou podobné jako u emulátoru Bochs. Navíc ale ke QEMU existuje akcelerátor zvaný KQEMU, který umožňuje zrychlit běh QEMU tím, že se instrukce virtualizovaného stoje částečně spouštějí na skutečném procesoru. S použitím KQEMU tedy QEMU již není čistý emulátor, ale spíše se řadí mezi virtualizační software, který popisujeme v následující kapitole.

4.2 Virtualizace – VMware, Virtual PC, VirtualBox

V této části se seznámíme s řešeními na bázi virtualizace, které alespoň částečně provozují instrukce virtualizovaného systému na hostujícím procesoru z důvodů navýšení výkonu oproti čisté emulaci. Z řady řešení se budeme věnovat pouze třem produktům, které výrazně ovlivnili vývoj na poli virtualizace v systémech Windows.

4.2.1 Krátký historický přehled

První produkt s podporou virtualizace systému Windows pochází od společnosti VMware Inc. Společnost byla založena v roce 1998 a již v roce 1999 přestavila svůj první produkt VMware Workstation. V roce 2001 pak nabídla i serverová řešení VMware GSX Server a VMware ESX Server. Produkty VMware nabízely v prvé řadě možnost virtualizovat různé systémy na hostujícím systému Windows na architektuře IA-32. Později byla implementována i verze pro varianty Linuxů a Macintosh. V roce 2004 byla přidána podpora pro 64-bitovou architekturu. Postupem času přicházela na trh konkurence, ale vývoj produktů řady VMware byl dlouhodobě o krok napřed. V roce 2006 společnost VMware ukončuje řadu VMware GSX Server a nahrazuje ji produktem VMware Server, který dává zdarma k použití.

Druhým velkým hráčem na trhu virtualizace pro Windows se chtěla stát společnost Microsoft Corporation se svými produktem Virtual PC a Virtual Server. Tyto produkty původně vyvíjela společnost Connectix Corporation. První verze s podporou Windows byla dostupná již v roce 2001. Na počátku roku 2003 koupil tyto produkty Microsoft. Ve Virtual PC oficiálně podporoval pouze virtualizaci Windows. Ve Virtual Serveru pak nabízel podporu i pro UNIXové systémy. Původně komerční řešení v konkurenci produktů VMware neobstála. Microsoft se tak rozhodl uvolnit oba produkty zdarma. Dnešní zaměření Microsoftu v oblasti technologie Virtual PC je integrace do operačního systému Windows 7, který má poskytovat možnost běhu různých verzí Windows najednou.

V roce 2006 vyvinula společnost Innotek komerční virtualizační software VirtualBox. K dispozici byla i verze dostupná zdarma pro osobní a akademické využití. V roce 2007 byla

uvolněna první verze VirtualBox OSE, která byla dostupná zcela zdarma a to včetně zdrojových kódů. Společnost Innotek byla později převzata společností Sun Microsystems, která ve vývoji VirtualBoxu pokračovala. Dnes je VirtualBox nabízen ve standardní verzi, která je zdarma pro osobní a akademické užití a ve verzi OSE, jejíž role se nezměnila. Část VirtualBoxu je založena na zdrojových kódech QEMU.

4.2.2 Možnosti přímého vykonávání vzhledem k režimu procesoru

Princip fungování VMware, Virtual PC a VirtualBoxu je stejný. Odlišnosti jsou především v množství funkcí a úrovni komfortu, které tyto produkty nabízejí. Výkonnostní rozdíly jsou pak otázkou různých optimalizací, některé z nich zmíníme.

Virtualizovaný stroj při svém běhu střídá několik režimů (módů), ve kterých pracuje. Již jsme zmínili snahu pouštět instrukce virtuálního systému přímo na hostujícím procesoru. Tento režim však sám o sobě nestačí. Architektura x86 podporuje několik režimů procesoru, které stroj během svého běhu vystřídá. Jen v některých režimech je možný přímé vykonávání instrukcí reálným procesorem. Toto omezení plyne z návrhu architektury x86, která nebyla navrhována se záměrem snadné virtualizace. Omezme se v dalším na architekturu IA-32.

Po startu stroje začíná procesor v tzv. reálném módu (Real Mode). Operační systém během svého zavádění přepne procesor do chráněného režimu (Protected Mode). Rozdíl mezi módy je například v práci s pamětí a práci s ostatním hardware. Reálný mód je velmi jednoduchý, na rozdíl od chráněného režimu nenabízí žádnou podporu pro multitasking, ochranu paměti ani rozdílné úrovně vykonávání instrukcí (Ring 0 až 3).

Instrukce reálného módu nemůže virtualizační software nechat vykonávat přímo na hostujícím procesoru. Zde virtualizační software používá emulaci stejně jako Bochs. U chráněného módu je potřeba rozlišit instrukce, které se mají vykonat v uživatelském režimu, od těch, které se mají vykonat v režimu jádra virtualizovaného stroje.

Většinu instrukcí pro uživatelský režim lze pouštět přímo. Virtualizační software vytvoří pro virtuální stroj jeho vlastní proces (nazveme ho *hostující proces*), v jehož kontextu se tyto instrukce vykonají. Jelikož jsou instrukce vykonávány ve vlastním procesu, nemohou poškodit systém. Určitě však nastane problém s pamětí, když systém ve virtualizovaném stroji spravuje více procesů, které mají být paměťově nezávislé. K tomuto a dalším problémům se dostaneme později.

Pro instrukce určené pro vykonání v režimu jádra tento přístup není možný, neboť hostující proces stroje nemá možnost tyto instrukce vykonat. Virtualizační software proto implementuje ovladač, který mu dá možnost vykonávat instrukce v nižších úrovních procesoru (Ring 0 nebo 1) hostujícího systému. Je však třeba dát pozor, aby nedošlo k poškození jádra hostujícího systému. Proto jen část instrukcí je možno pouštět opravdu přímo byť v úrovni Ring 1.

Další běžně používaný režim procesoru je virtuální reálný režim (Virtual Real Mode nebo také Virtual 8086, VM86). Tento mód se používá výhradně pro spouštění starých 16-bitových aplikací vytvořených pro běh na operačním systému DOS. Systémy Windows ve verzích před Vista na 32-bitových architekturách podporují běh těchto aplikací právě pomocí virtuálního reálného režimu uvnitř virtuálního stroje NTVDM (NT Virtual DOS Machine). Instrukce určené pro vykonávání v režimu Virtual 8086 je možné pouštět přímo na hostujícím procesoru.

Asi poslední režim procesoru, kterým je třeba se zabývat je tzv. SMM (System Management Mode). Pokud je procesor v tomto režimu, celý operační systém a zařízení jsou pozastaveny

a stroj je řízen speciálním kódem v privilegovaném režimu. Tento režim se používá v krajních stavech systému jako je například vypínání hardware počítače, zavádění hibernace, obsluha chybových stavů zaviněných hardware. Tento režim je potřeba zcela emulovat stejně jako reálný mód.

V dalším textu se budeme zajímat pouze o chráněný režim procesoru. Reálný mód a SMM plně emulujeme a tudíž nevznikají problémy a taktéž Virtual 8086 není zajímavý, protože zde si můžeme dovolit pouštět všechny instrukce přímo na hostujícím procesoru a prakticky provádíme to, co systémový program NTVDM.

4.2.3 Analýza a přepis instrukcí pro práci s pamětí

Uvedli jsme, že některé instrukce lze pouštět přímo na hostujícím procesoru. Každý úsek kódu, který chceme spustit, musíme dopředu analyzovat. Můžeme tak zjistit, zda je úsek kódu zcela bezpečný, tzn. lze jej beze změny pustit na hostujícím procesoru, nebo zda je bezpečný ve formátu svých instrukcí, ale je potřeba provést nějakou akci v parametrech instrukcí popř. se postarat o načtení paměti, kterou chce kód využít, a nebo zda je zcela nebezpečný a je třeba jeho instrukce změnit, aby nedošlo k poškození hostujícího prostředí.

Virtualizovaný stroj má od počátku přidělenou nějakou pevnou velikost operační paměti. Z pohledu virtualizovaného stroje se bude jednat o fyzickou paměť, ale reálná implementace na hostujícím stroji alokuje paměť virtuální. Paměť hostujícího procesu bude obsahovat paměť, kterou využívají instrukce uživatelského režimu virtuálního stroje. Máme minimálně dvě možnosti, jak zachovat správný přístup těchto instrukcí k paměti.

První možností je při změně kontextu adresového prostoru v rámci virtuálního stroje nahradit paměť v hostujícím procesu tak, aby si adresy virtuální paměti hostujícího i virtuálního stroje odpovídaly. Instrukce, která pak přistoupí k paměti na nějaké virtuální adrese ve virtuálním stroji, se tak může přímo a beze změny vykonat v hostujícím procesu. Tato možnost tedy přemapovává obsah paměti hostujícího procesu tak, jak dochází ke střídání kontextů adresových prostorů ve virtuálním systému.

Druhou možností je modifikace instrukcí pro práci s pamětí. Adresový prostor hostujícího procesu bude naplněn daty, které instrukce virtualizovaného systému potřebují, ale jejich adresy nebudou odpovídat adresám ve virtualizovaném systému. Abychom zajistili, že instrukce dosáhnou těchto dat i v tomto provedení, změníme parametry (adresy) v těchto instrukcích.

Oba přístupy jsou možné a mají vlastní problémy, přístupy lze navzájem i kombinovat. U obou je však nutné, aby virtualizační software implementoval vlastní mapování paměti, kterou alokoval pro operační paměť virtuálního stroje.

V případě instrukcí, které mají být vykonány v režimu jádra virtualizovaného systému, lze použít pouze druhý přístup, neboť hostující systém obsazuje důležité adresy v jádře vlastním kódem a daty. Úrovně procesoru Ring 1 a 2 nejsou běžně používány v operačním systému a adresový rozsah dostupný těmto úrovním bývá nevyužit. Ve virtualizačním software se tak pro zvýšení výkonu používá následující trik. Instrukce určené k vykonání na úrovni Ring 0, 1 a 2 se vykonávají pomocí ovladače v úrovni Ring 1. Tím se zabrání poškození jádra hostujícího systému, ale zároveň není potřeba tolika změn a tím se šetří čas.

4.2.4 Nebezpečné instrukce

U řady instrukcí nelze povolit jejich vykonávání na hostujícím procesoru a nepostačí ani změna parametrů. Týká se to řady privilegovaných instrukcí, ale i několika instrukcí

povolených v uživatelském režimu. Takové instrukce je potřeba nahradit v kódu tak, aby virtualizační software měl možnost zcela implementovat jejich dopady. To lze řešit minimálně dvěma způsoby. První možností je zavedení speciálních instrukcí, které jsou neplatné na dané architektuře a pokus o jejich vykonání na hostujícím procesoru způsobí výjimku, kterou lze zachytit a ošetřit. Jedná se o techniku podobnou jedné z možností pro vkládané hákování popsané v kapitole 3.1.1. Druhou možností je náhrada problémových instrukcí instrukcemi volání funkcí, které emulují chování tak, jak je potřeba.

Jako příklad takové instrukce uveďme instrukci CLI, která slouží pro zákaz přerušení na procesoru. Rozhodně nelze tuto privilegovanou instrukci vykonat na hostujícím počítači, neboť by to zásadním způsobem narušilo běh hostujícího systému. Vzhledem k tomu, že virtualizační software se v každém případě musí starat o záležitosti kolem přerušení virtualizovaného systému je poměrně snadné tuto instrukci emulovat. Jednoduše si pro daný virtualizovaný procesor poznamená, že nemá být přerušován, čímž se ovlivní emulace dějů starajících se o přerušení. Pokud by však instrukci CLI obsahoval kód určený pro uživatelský režim virtualizovaného stroje, nebyl by to problém a přepis a emulace by nebyla potřeba z toho důvodu, že v uživatelském režimu není tato instrukce povolena a její vykonání by vedlo na výjimku, se kterou si virtualizační software poradí stejně jako s každou jinou výjimkou v hostujícím procesu.

Pro příklad instrukce dostupné uživatelskému režimu, kterou je třeba emulovat, uveďme instrukci SIDT, která zkopíruje tabulku GDTR procesoru do určené paměti. Emulace je potřeba z toho důvodu, že tabulka GDTR může být ve virtualizovaném systému na jiném místě než u hostujícího systému. Pokud by se instrukce SIDT nechala vykonat přímo hostujícím procesorem, zkopírovaná data by se lišila od skutečných hodnot virtualizovaného systému. Takové rozdíly jsou silně nežádoucí.

4.2.5 Hardwarově asistovaná virtualizace

Jak již možná vyplynulo z předchozího textu, virtualizace na architektuře x86 není triviální problém. Na tuto skutečnost v nedávné době zareagovali i výrobci procesorů – Intel a AMD. Od roku 2006 tak poskytují vybrané procesory pro platformu x86 hardwarově asistovanou virtualizaci. Technologie Intelu (Intel VT) a AMD (AMD-V) nejsou kompatibilní, ale v principu se jedná o tutéž myšlenku – rozšíření podporovaných režimů procesoru o tzv. *režim hypervisor* (hypervisor je pojem obecně používaný v terminologii virtualizace pro komponentu, která spravuje virtualizaci), jinak také zvaný *root mód*, a rozšíření instrukční sady procesoru o instrukce, pomocí kterých hypervisor spravuje virtualizační prostředí.

Režim hypervisor si lze představit jako úroveň Ring -1, tj. ještě nižší úroveň vykonávání kódu než je režim jádra. V režimu hypervisor je možné konfigurovat některé instrukce procesoru tak, že jejich vykonávání v běžném režimu přepne procesor do režimu hypervisor a předá řízení k tomu určenému kódu. Tím lze předejít nutnosti modifikací instrukcí virtualizovaného stroje určených pro režim jádra. Lze je vykonat přímo na hostujícím procesoru a část virtualizačního software implementující kód pro režim hypervisor se postará o to, aby zpracování těchto instrukcí nepoškodilo hostující systém. Následkem toho lze kód jádra virtualizovaného stroje vykonávat mnohem rychleji než metodami popsanými v předchozích částech.

Dnes již všechny moderní virtualizační aplikace, včetně VMware, Virtual PC a VirtualBoxu, podporují hardwarově asistovanou virtualizaci. Tato nová technologie ale rozhodně neřeší všechny problémy, se kterými si virtualizační software musí poradit. Jedná se čistě o snahu o zjednodušení implementace virtualizace procesorů x86 a z toho plynoucí zvýšení výkonu virtualizovaných strojů.

4.2.6 Možnosti detekce virtuálního prostředí

Pokud by virtualizační software byl dokonalý, neměl by kód běžící ve virtualizovaném systému možnost poznat, že běží ve virtualizovaném prostředí. Ukázalo se však, že za běžného provozu drobné odlišnosti od nevirtualizovaného prostředí virtualizovaným systémům nevadí. Dílčí optimalizace virtualizačního software za účelem nárůstu výkonu virtualizovaného stroje takové odlišnosti přináší. Další odlišnosti se objevili v historii vývoje virtualizačních software jako chyby v implementaci. Dnešní verze například produktů VMware nabízejí možnost konfigurace různých optimalizací a tím i redukci počtu odlišností, ale za cenu možného snížení výkonu.

Odlišnosti v chování virtualizovaného a klasického stroje dávají kódu běžícím ve virtualizovaném prostředí možnost detekce virtualizace. Jako příklad uveďme existenci vlastních instrukcí virtualizačního software. V minulých částech jsme popsali metodu, při níž se některé instrukce ve virtualizovaném kódu nahrazují speciálními instrukcemi, které zavádí virtualizační software. Vlastní instrukce se používají i pro implementaci zajímavých funkcí virtualizačního software jako je například možnost plynule přecházet mezi obrazovkou hostujícího a virtualizovaného stroje nebo možnost sdílet schránku (clipboard). Pokud však virtualizovaný kód obsahuje tyto instrukce a pokus o jejich vykonání nezpůsobí výjimku, jak by se stalo na nevirtualizovaném stroji, pak to potvrzuje přítomnost virtualizace. Na to může implementace virtualizačního software odpovědět náhradou všech svých vyhrazených existujících ve virtualizovaném kódu na instrukce, které se zpracují jako na nevirtualizovaném stroji. Problém pak nastává, pokud se metoda analýzy pro daný kód nepoužívá například z důvodů optimalizace.

Byla sepsána řada prací na toto téma a místy se objevují nové metody detekce. Dobrá shrnutí přináší například [19] a [20].

4.2.7 Možnosti sledování

Jaké jsou tedy naše možnosti pro implementaci sledovacího software za pomoci virtualizace? Existující virtualizační software nabízí různé druhy přístupů programů k virtuálním strojům, ale tyto prostředky jsou velmi omezené, neboť jsou především určeny pro automatickou správu virtuálních strojů, nikoliv ke sledování dění uvnitř.

Se znalostí principů virtualizačních technik bezpochyby lze začít implementovat vlastní virtualizační software. S takovým přístupem si lze snadno představit, že získáme podobné možnosti jako jsme uvedli u emulace – viz kapitolu 4.1.1. Je však zřejmé, že tato úloha by byla nesmírně složitá. Většinu času z celého projektu by zabrala právě implementace dostatečně silné virtualizace a relativně malou částí projektu by byl samotný sledovací systém.

Jelikož je ale VirtualBox OSE distribuován se zdrojovými kódy pod licencí GNU General Public License (GPL), je možné jeho implementaci využít. Namísto implementace vlastního řešení na míru, můžeme analyzovat kód VirtualBoxu OSE a modifikovat jej tak, aby vyhovoval pro implementaci sledovacího systému. Zdarma tak získáme hotový virtualizační stroj s podporou nejnovějších technologií a po základním seznámení s architekturou VirtualBoxu se můžeme soustředit na vlastní cíle. Správnými modifikacemi kódu VirtualBoxu bychom měli získat stejné možnosti jako při implementaci vlastního virtualizačního software.

4.3 Paravirtualizace

Virtualizace, jak jsme ji popsali v předchozí části, se často nazývá plná nebo úplná (full virtualization). Dochází při ní k plnému oddělení hostujícího a virtualizovaného systému. Virtualizovaný systém pracuje s virtuálním hardware, zatímco hostující systém pracuje s reálným hardware (s výjimkou případu, kdy se snažíme ve virtualizovaném prostředí virtualizovat další stroj, ale tento extrémní případ vynechme). Virtualizovanému stoji tak můžeme prostřednictvím virtualizačního software poskytnout zcela jiný hardware, než kterým disponuje hostitelský stroj. Tuto výhodu kompenzuje nevýhoda redukce výkonu virtualizovaného stroje, která je u plné virtualizace znatelná.

Paravirtualizace přistupuje k problému odlišně. Její základní podmínkou je, že hostující i virtualizovaný stroj mají k dispozici podobný (ale ne totožný) hardware. U paravirtualizace se tedy hostující a virtualizovaný hardware příliš neliší a to umožňuje, aby virtualizovaný stroj v hojné míře využíval reálný hardware přímo. Tím lze získat na výkonu virtualizovaného stroje ve srovnání s úplnou virtualizací. Na druhou stranu není oddělení virtualizovaného stroje úplné a tudíž virtualizovaný kód může poznat, že je virtualizován.

Pro implementaci paravirtualizace je potřeba podpora jak od hostujícího systému, tak od virtualizovaného. Z toho plyne i poměrně úzká specializace využití paravirtualizace. Tato virtualizační technologie se používá pro běh UNIXových systémů (s otevřeným zdrojovým kódem, který lze upravit tak, aby podporovaly paravirtualizci) za účelem zisku maximálního výkonu při zachování výhod virtualizace.

Jako příklad jedné z možných technik používaných u paravirtualizace uveďme odstranění obtížně virtualizovaných instrukcí ze systému. Tím se lze vyhnout potřebě emulace a související analýzy, která způsobuje znatelné zpomalení. K tomu je ovšem potřeba změna kódu systému a také tím dáváme virtualizovanému kódu možnost detekce virtualizace.

Firma VMware Inc. přišla s vlastní otevřenou koncepcí paravirtualizace, která odstraňuje nevýhodu nutnosti úprav kódu systému a tím by se stala paravirtualizace použitelnou i pro uzavřené systémy jako jsou například Windows. Myšlenka spočívá ve vytvoření virtualizačního rozhraní, se kterým výrobce operačního systému počítá (je tedy zabudované do systému) a jehož prostřednictvím je možné dosáhnout v systému stavu, který paravirtualizace potřebuje. Zásah do kódu systému se tak děje při jeho výrobě a virtualizační software implementující paravirtualizaci pak pouze využije hotového rozhraní. Systém tak může podporovat paravirtualizaci a může být stále uzavřený a jeho kód neměnitelný. Tento koncept byl implementován do jádra Linuxu, systémy Windows však podporu nepřidaly.

Z výše uvedeného plyne, že paravirtualizace není vhodná pro implementaci sledovacího systému pro Windows.

4.4 Oddělení na úrovni uživatelského režimu (user-level sandboxing)

Poslední metodou, kterou zde zmíníme, je tzv. *oddělení* (sandboxing). S virtualizací má společnou pouze myšlenku izolace. Na rozdíl od virtualizace ale nevytváříme celý nový systém, nýbrž izolujeme pouze chování jednotlivých procesů tak, aby jejich chování nemohlo narušit běh ostatních komponent a aplikací v systému.

Takovou izolaci lze zajistit například sofistikovaným použitím hákovacích technik v jádře systému, ale praxe u řady bezpečnostních produktů, které se o takové oddělení snažily, ukázala, že složitost úlohy je extrémní a výsledné produkty jsou nespolehlivé v tom smyslu, že oddělení není dostatečné a je možné okolní systém infiltrovat. Teoreticky je však implementace možná.

Jiný přístup nabízí oddělení založené pouze na technikách dostupných v uživatelském režimu, pro které existují minimálně dva koncepty – Vx32 ([22]) a Chromium ([23]).

4.4.1 Vx32

Základní myšlenkou Vx32 je kontrolovaný běh izolovaného kódu v rámci speciálního (hostujícího) procesu. Kontrola běhu je zajištěna pomocí technik analýzy a emulace. Hlavním přínosem metody, kterou Vx32 prezentuje, je nárůst výkonu, kterého se dosahuje využitím segmentace poskytované 32-bitovými procesory x86. Segmentace umožňuje oddělit přístup izolovaného kódu od zbytku paměti hostujícího procesu. Odpadá tak nutnost neustálých změn v parametrech instrukcí, které pracují s pamětí. Přístup k datům je tedy izolován hardwarově procesorem.

Emulace je však stále třeba například pro instrukce volání systémových služeb. Právě zde je nutno ošetřit všechny možné interakce a tím implementovat samotné oddělení. Koncept Vx32 byl implementován pro UNIXové systémy. Nic ale nebrání implementaci téže metody na Windows. Metoda je obecně použitelná i na jiných architekturách než je IA-32, nicméně výkonnostní výhody plynoucí z použití segmentace jsou dostupné pouze na procesorech, které podobnou vlastnost mají. Vlastnosti metody vzhledem k implementaci sledovacího systému jsou téměř stejné jako u emulace. Podstatnou nevýhodou je složitost izolace, kterou je potřeba bezchybně implementovat, aby izolovaný kód nemohl poškodit hostující systém.

4.4.2 Chromium

Realizace izolace v metodě prezentované projektem Chromium je zcela odlišná od všech předchozích postupů. Chromium poněkud neobvykle využívá hákování v uživatelském režimu. Izolovaný kód běží ve svém procesu, který může libovolně využívat bez omezení. Tento proces však je omezen systémovým nastavením práv tak, že nemá prakticky žádná oprávnění a nemůže tedy napáchat v systému škody.

V takovém stavu by ale nemohl žádný program fungovat, takže je potřeba nějakým způsobem kompenzovat zavedená omezení. To se děje právě pomocí hákování v uživatelském režimu. Funkce, které zajišťují běžný chod programu a které jsou omezené nastavím práv procesu, jsou hákovány. Nový kód má prostředky pro komunikaci s externím procesem, který na požádání realizuje v kontrolované části systému požadavky izolovaného procesu. Na rozdíl od přímého využití metod hákování v uživatelském režimu ke sledování chování procesu nemůže proces hákování obejít tak, aby vykonal nějakou operaci aniž by o ní byl sledovací systém informován. Jakékoliv obcházení hákovaných funkcí povede k vykonání originálního kódu, který není možno provést z důvodu omezení plynoucí z nastavení práv procesu. Pokud chce izolovaný proces něco vykonat, musí použít hákování. Každá realizace požadavku v externím procesu může samozřejmě sloužit k implementaci sledování.

Výhoda této metody je, že izolace provádí sám systém. Nevýhodou může být poměrně složitá implementace všech funkcí, které kompenzují omezení tak, aby aplikace spuštěné v izolovaném prostředí mohly normálně fungovat. Tento koncept byl implementován pro operační systém Windows, kde omezení procesu je možné jednoduše realizovat.

5 Volně dostupné sledovací systémy

V této kapitole představíme již fungující sledovací systémy, které navíc jsou volně dostupné k použití prostřednictvím webových serverů. Takových systémů není mnoho a každý systém je navržen trochu jinak a používá jinou metodu nebo kombinaci metod ke sledování. Každý systém krátce popíšeme a pokud jsou informace dostupné, zmíníme jeho základní návrh, metody použité pro sledování a vlastnosti systému. U každého systému ukážeme jeho výstup na dvou demonstračních vzorcích. Analýze reálných vzorků malware se věnuje až kapitola 8 této práce.

Jako demonstrační vzorky si vypůjčíme dva programy od společnosti Diamond Computer Systems Pty. Ltd., konkrétně její volně použitelné konzolové programy CmdLine a GetIP.

Popis CmdLine je k dispozici na stránce http://tds.diamondcs.com.au/consoletools/cmdline.php, jedná se o program, který vypíše seznam všech běžících procesů v systému a k nim vydá plnou cestu a seznam argumentů, se kterými byly spuštěny. Informace o argumentech procesu se získávají z jeho paměti. CmdLine tedy musí nejprve získat seznam běžících procesů a následně každý proces otevřít a přečíst si tu část jeho paměti, kde se požadované informace nalézají. V ideálním případě by se z výstupu sledovacích systémů mělo dát zjistit také to, že vzorek vypsal informaci na standardní výstup. Úplné výstupy jednotlivých systémů jsou na přiloženém médiu v adresářích `/output/sample_cmdline/<SystemName>/`, kde `<SystemName>` je jméno sledovacího systému.

Popis GetIP je k dispozici na stránce http://tds.diamondcs.com.au/consoletools/getip.php, jedná se o program, který se připojí na server a stáhne obsah stránky, která udává IP adresu klienta. IP adresu program vypíše a skončí. V ideálním případě by se z výstupu sledovacích systémů mělo dát zjistit, že se program připojil na server a stáhl obsah webové stránky, případně také, že vypsal informaci na standardní výstup. Úplné výstupy jednotlivých systémů jsou na přiloženém médiu v adresářích `/output/sample_getip/<SystemName>/`, kde `<SystemName>` je jméno sledovacího systému.

Výstupy systémů jsou často rozsáhlé, proto nejsou součástí textu práce, ale jsou uloženy pouze na přiloženém médiu k této práci.

5.1 CWSandbox

Webová rozhraní: http://research.sunbelt-software.com/Submit.aspx, http://www.cwsandbox.org/?page=submit

O systému

CWSandbox je dílem výzkumníků z university Mannheim. Cílem tohoto projektu je pomoci svými výstupy lidským pracovníkům rychleji analyzovat neznámý vzorek. Autoři si stanovili tři požadavky, které by systém měl splňovat. V prvé řadě systém nepotřebuje lidskou intervenci vyjma nahrání vzorku do systému. Je potřeba plná samostatnost systému při analýze. Dalším bodem je úplnost. Tím je myšleno, že systém by měl zachytit každý aspekt chování vzorků, který může být důležitý. Posledním požadavkem je korektnost,

neboli požadavek, aby ve výstupu systému figurovaly pouze události způsobené vzorkem.

Systém je postaven na metodě hákování v uživatelském režimu. Na reálném systému se vytvoří proces testovaného vzorku, do kterého je sledovacím systémem nahrána sledovací knihovna, která implementuje hákování vybraných funkcí. Autoři si jsou vědomi nedostatků použité metody, konkrétně možnosti obejít hákování v uživatelském režimu. Argumentují však, že výskyt technik obcházení hákování je u malware řídký a systém tedy bude pracovat spolehlivě pro většinu vzorků.

Komerčním partnerem projektu je společnost Sunbelt Software, Inc. Ta provozuje svoji vlastní kopii a nabízí systém k použití komerčním subjektům. Pro účely této práce bylo použito právě rozhraní společnosti Sunbelt Software, Inc. z toho důvodu, že původní rozhraní provozované univerzitou Mannheim nefungovalo dostatečně rychle.

Komentář k demonstračním výstupům

Výstup pro demonstrační vzorek CmdLine obsahuje zajímavé informace v hlavní části Process Details. Zde se v části Process section dozvíme, že CmdLine získal seznam ostatních procesů a postupně je otvíral. Dále je v části Virtual Memory Section seznam operací čtení paměti. Další dostupné informace nejsou zajímavé a jedná se spíše o nadbytečná data, která by výstup nemusel obsahovat. Operace testovacího vzorku jsou pouze vyjmenované, nejsou chronologicky setříděné, nicméně z výstupu lze částečně vyčíst skutečné chování CmdLine.

Výstup pro demonstrační vzorek GetIP obsahuje informace o třech procesech. První je proces vzorku, další dva k vzorku žádný vztah nemají a výstup by je obsahovat neměl. Není zřejmé, proč systém do výstupu zahrnul tyto procesy. Navíc u těchto procesů výstup obsahuje pouze informaci o jakési chybě. V hlavní části Network Activity je zaznamenán přístup vzorku k síti, tj. stažení webové stránky ze serveru. K dispozici jsou i hlavičky HTTP požadavku, ale příchozí data uvedena nejsou. Část Technical Details pak obsahuje další informace o běhu procesu, nic podstatného ale nepřidává. U tohoto vzorku výstup obsahuje všechna data potřebná ke zjištění jeho chování.

5.2 Norman SandBox

Webové rozhraní: http://www.norman.com/security_center/security_tools/submit_file/en-us

O systému

Z dílny společnosti Norman ASA pochází sledovací systém Norman SandBox. Technologie tohoto systému je použita v komerčních produktech společnosti Norman. Motivací autorů jsou nedostatky běžných antivirových technologií, které k ochraně počítače potřebují poznat útočící malware. Nový malware je zařazen do databází antivirových společností v řádu několika hodin až dní. Klienti pak prostřednictvím aktualizací lokálních databází svých antivirových produktů získávají ochranu s poměrně velkým zpožděním. Efektivní rozpoznání malware i bez záznamu v databázi antiviru by pomohlo ochránit klientské stanice v době před aktualizací databáze.

Dle informací na webu výrobce je Norman SandBox postaven na metodě emulace. Norman SandBox částečně emuluje počítač, lokální síť a systém Windows. Systém pracuje v několika krocích. Nejprve je vzorek v emulovaném stroji spuštěn. Následuje fáze pozorování chování vzorku. Dále jsou nasbíraná data analyzována a nakonec jsou výsledky analýzy podrobeny

vyhodnocení rizik. Technologie rozpoznává řadu podezřelých jevů v chování vzorku a na základě toho určuje, zda se jedná o malware.

Komentář k demonstračním výstupům

Výstup systému Norman SandBox je ve srovnání s ostatními velmi stručný. U CmdLine obsahuje jedinou relevantní informaci o tom, že testovaný vzorek si vyžádal oprávnění SeDebugPrivilege, které je nutné pro přístup k procesům cizích uživatelů – například k systémovým procesům. Jinou relevantní informaci výstup neuvádí. Je pravděpodobné, že pouhé čtení paměti cizích procesů není považováno za nebezpečnou operaci, neboť nemůže ovlivnit chod ostatních procesů v systému, a tudíž není ve výstupu zmíněno.

Pro GetIP výstup obsahuje informaci o připojení na server a vyžádání obsahu stránky. Podrobnější informace nejsou k dispozici, ale u tohoto příkladu je informace ve výstupu zcela dostačující.

5.3 Anubis

Webové rozhraní: http://anubis.iseclab.org/

O systému

Anubis je nástroj pro analýzu chování vzorků spustitelných na systémech Windows se speciálním zaměřením na analýzu malware. Za přispění organizace Secure Business Austria jej vytvořila výzkumná laboratoř International Secure Systems Lab spadající pod technickou univerzitu ve Vídni. Uživatel systému Anubis by měl být schopen na základě vygenerovaného výstupu analýzy rozpoznat záměr testovaného vzorku a jeho konkrétní kroky.

Anubis je postaven na projektu QEMU. Základní metodou systému Anubis je emulace. Kód QEMU byl modifikován tak, aby vyhovoval požadavkům projektu jako je například snadné sledování dění v emulovaném systému. Anubis virtualizuje Windows XP, jejíž čistou instalaci pokaždé načte z uloženého obrazu. Anubis sleduje vybrané funkce v systému pomocí hlídání adresy vykonávané instrukce procesoru. Sledovací komponenta dostává informace o volání vybraných funkcí a má možnost přistupovat k jejím parametrům.

Pro lepší možnosti analýzy Anubis také používá techniky vkládání kódu do emulovaného stroje a volání funkcí v kontextu procesu sledovaného vzorku tak, jak jsme popsali v kapitole 4.1.1.

V nedávné době bylo do systému Anubis zakomponováno seskupování (clustering) výstupů, které byly systémem vygenerovány. Testované vzorky a jejich výstupy byly rozřazeny do kategorií podle společných skupin prvků odhalených v jejich chování. To má pomoci odhalit rodiny malware, tj. různé vzorky malware, které se navzájem svým chováním příliš neliší – například různé vývojové verze téhož malware nebo více instancí téhož malware při použití polymorfismu nebo metamorfismu ze snahy vyhnout se detekci antivirů. Velkým přínosem seskupování výsledků je v tom, že pokud je v systému již znám nebezpečný vzorek, který vykazuje podobné chování jako nový neznámý vzorek, můžeme nový vzorek označit jako nebezpečný a automatizovat například jeho zařazení do databáze antiviru. Tím lze výrazně zkrátit dobu potřebnou pro doručení informace o novém vzorku k zákazníkovi antivirového produktu.

Komentář k demonstračním výstupům

Oba výstupy začínají shrnutím vybraných aspektů chování testovaných vzorků, ale v konkrétních případech demonstračních vzorků GetIP a CmdLine jsou daná shrnutí velmi zavádějící. CmdLine je označen za program, který vykonal velmi rizikovou operaci, když změnil soubory, které nejsou dočasné. Totéž stojí ve výstupu pro GetIP. Dále nízkým rizikem byly označeny akce CmdLine i GetIP, které údajně čtou a modifikují registrové hodnoty a navíc sledují registrové klíče. U GetIP navíc středním rizikem byla ohodnocena akce vzorku, která údajně změnila bezpečnostní nastavení Internet Exploreru, čímž by mohla být narušena bezpečnost surfování.

Z dalšího obsahu výstupu je však pravděpodobné, že aspekty chování, na jejichž základě Anubis identifikoval výše zmíněná rizika, jsou způsobeny běžnými operacemi systémových komponent. Nejsou tudíž relevantní a systém by z nich neměl usuzovat na chování testovaných vzorků.

Výstup u obou programů uvádí jejich výstup na konzoli, což v případě demonstračních programů zcela odhaluje podstatu jejich chování. Tato funkce je rozhodně přínosná pro snadné rozpoznání bezpečných programů. Pokud by však CmdLine neměl konzolový výstup, výsledek systému Anubis by nijak nepomohl k získání informací o jeho skutečném chování – pravděpodobně informace o otevírání a čtení paměti procesů chybí ze stejných důvodů jako jsme uvedli u systému Norman SandBox.

Pro GetIP výstup obsahuje informaci o DNS dotazu na adresu webového serveru a dále informaci o HTTP požadavku, který byl poslán na server. Výstup je podpořen záznamem síťového provozu, ve kterém je však potřeba rozeznat běžný provoz ostatních komponent systému a provoz způsobený vzorkem. Pro tento demonstrační příklad jsou informace dostačující.

Výstupy pro oba programy, zvláště pak GetIP, obsahují poměrně velké množství dílčích informací, které bez dalšího kontextu nevypovídají o chování vzorků.

5.4 Joebox

Webové rozhraní: http://www.joebox.org/submit.php

O systému

Joebox je sledovací systém kompatibilní se systémy Windows XP a Windows Vista vytvořený organizací Joe Security. Cílem projektu je vytvářet srozumitelné výstupy popisující chování testovaných vzorků na vysoké úrovni abstrakce, které budou moci být automaticky zpracovávány dalšími aplikacemi.

Systém obsahuje dva reálné počítače – analytický a kontrolní. Na kontrolní počítač dorazí testovaný vzorek. Ten je předán na serverovou aplikaci, která je nainstalována na analytickém počítači. Na základě zadání server může spustit připravené skripty AutoIt, pomocí kterých lze ovládat celý analytický systém včetně ovládaní grafických rozhraní aplikací. Dále se server stará o zavedení testovaného vzorku a o instalaci sledovací části uživatelského režimu do jeho procesu. Dále na analytickém stroji v režimu jádra běží dva ovladače. První zajišťuje sledování v jádře systému a druhý se stará o to, aby komponenty sledovacího systému nebyly vidět z uživatelského režimu. Celý systém je navíc podpořen síťovým sledovačem (network sniffer) tshark, který je nainstalován na kontrolním počítači. Všechny poznatky jsou

přes server na analytickém stroji doručovány po síti zpět na kontrolní počítač, kde se vyhodnocují a vytváří se výstupy systému.

Uživatelská komponenta sledovacího systému pracuje na bázi hákování EAT vybraných funkcí. Sledovací komponenta v jádře analytického systému je založena na hákování SSDT. Velký důraz byl při implementaci kladen na znesnadnění detekce přítomnosti sledovacího systému.

Komentář k demonstračním výstupům

Výstupy v porovnání s ostatními systémy obsahují záplavu informací, z nichž velká část je naprosto zbytečná, pokud bereme v potaz pouze získání informací o chování vzorků. Naštěstí však výstup obsahuje i velmi užitečné informace a navíc je úroveň detailu většiny informací ve výstupu nadprůměrná při srovnání s ostatními systémy.

Část Chronological sections poměrně dobře dokumentuje důležité aspekty chování CmdLine v čase. Vidíme jak standardní inicializaci procesu, tak jeho vlastní chování. Nejsou zde sice obsaženy operace otevření procesu, ale čtení paměti přítomno je. Otevření procesu je odvoditelné z části Chronological functions.

U GetIP obsah částí Chronological sections a Chronological functions příliš mnoho neřekne, jelikož se uživatel ztratí v záplavě operací, které vykonává systémová knihovna, kterou GetIP používá pro přístup k Internetu. Výstup ale obsahuje části DNS a HTTP, které dokumentují dotaz na adresu webového serveru a vlastní dotaz na obsah stránky serveru. Výstup je podpořen externím záznamem síťového provozu, který však obsahuje i provoz způsobenými ostatními komponentami systému.

5.5 ThreatExpert

Webové rozhraní: http://www.threatexpert.com/submit.aspx

O systému

Z dílny společnosti ThreatExpert Ltd. pochází systém nazvaný ThreatExpert. Jedná se o automatický analyzátor projevů chování vzorků, vytvořený za účelem automatického ohodnocování nových hrozeb a jejich registrování do databází antivirových produktů.

Detaily implementace systému nejsou zveřejněny. Z popisu na stránkách projektu však plyne použití virtualizovaného prostředí a hákování, ověřili jsme, že se jedná o hákování v uživatelském módu. Pravděpodobně tak systém tvoří virtuální stroj, který obsahuje serverovou aplikaci, do níž se zvenku zašle vzorek na otestování. Ten se spustí ve virtuálním stroji a pomocí hákování vybraných funkcí se sleduje jeho chování, které je zpětně reportováno ven z virtualizovaného systému, kde se z těchto dat vytvoří výstup systému. Poté je virtuální stroj zrestaurován a připraven na další vzorek.

Komentář k demonstračním výstupům

Výstupy jsou velmi stručné. Pro CmdLine je výstup v podstatě prázdný a neobsahuje nic, z čeho by šlo usuzovat na jeho chování. Důvod je opět pravděpodobně ten, že CmdLine neprovádí nebezpečné změny v systému. U GetIP výstup obsahuje prakticky pouze informaci o stažení obsahu stránky webového serveru, což přesně vystihuje chování GetIP.

5.6 BitBlaze

Webové rozhraní: https://aerie.cs.berkeley.edu/submitsample.php

O systému

BitBlaze je dílem pracovníků a studentů Computer Science Division univerzity Berkeley. Jde o systém, který implementuje jak dynamickou tak statickou analýzu vzorků. Systém se skládá z mnoha komponent, které se podílejí na tvorbě a zpracování výstupu. Pro dynamickou analýzu používá upravený kód emulátoru QEMU. Modifikace umožňuje sledovat vybrané části emulovaného stroje, zatímco testovaný vzorek běží v systému, a také umožňuje do systému přidávat zásuvné moduly (plugin), které implementují další funkcionalitu nebo spojují systém s dalšími aplikacemi, které mohou pomoci s částí analýzy. Mezi přednosti systému patří možnost sledovat i dění v jádře emulovaného systému.

Mezi nástroje napojené v systému patří například Panorama, který sleduje, zda v systému nedochází k úniku senzitivních informací jako jsou například uživatelská hesla, data, vstupy apod. Dalším nástrojem je Renovo, který se snaží odhalit skutečný kód zašifrovaných vzorků tak, že sleduje přístup testovaného vzorku k emulované paměti. HookFinder se pak snaží detekovat hákování implementované testovaným vzorkem. Malware často aplikuje hákování k dosažení svých cílů jako je například skrytí vlastních objektů a projevů chování. Ještě zmíníme komponentu BitScope, která se snaží odhalit vztah mezi vstupy a výstupy testovaného vzorku. Taková znalost může sloužit pro detekci vzorků, které záměrně implementují kód, který znesnadňuje analýzu – jako jsou například tzv. časové bomby, kdy škodlivý kód se vykoná pouze v případě, že systémový čas odpovídá nějaké formuli.

Komentář k demonstračním výstupům

Výstupy systému BitBlaze jsou odlišné od výstupů ostatních systému z toho důvodu, že BitBlaze provádí i statickou analýzu vzorků. Výstup obsahuje část informace, kterou vzorky vypsaly na konzoli.

U výstupu pro CmdLine je chování vzorku možné odhadnout pouze na základě grafu z výstupu statické analýzy. Data z dynamické analýzy nenapovídají skutečnému chování vzorku. Navíc je CmdLine v části Hooks/Taint Info označen za program, který vykazuje znaky chování rootkitu. Z daných informací není zřejmé, co je tím myšleno. Výstup je podpořen výstupem síťového sledovače (sniffer), který však obsahuje veškerou komunikaci, která se na síti udála a tedy i komunikaci běžnou pro samotný systém Windows zapojený v síti.

U GetIP je situace podobná, ale vzorek nevykazuje chování rootkitu a na jeho chování se dá usoudit pouze analýzou výstupu síťového sledovače, který obsahuje komunikaci GetIP s webovým serverem.

6 Sledovací systém založený na VirtualBoxu

V této kapitole je vysvětlen výběr virtualizačního software VirtualBox pro ukázkovou implementaci, která je součástí práce. Dále popíšeme, jak je potřeba pozměnit zdrojové kódy VirtualBoxu, abychom docílili možnosti sledování dění ve virtualizovaném stroji.

6.1 Volba základního virtualizačního software

Od počátku bylo jasné, že pokud se práce má dostat až do fáze implementace sledovacího systému, nelze virtualizační software vyvíjet od základů. Proto první fáze práce byla věnována průzkumu existujících produktů, na jejichž bázi by se mohl sledovací systém s pokud možno minimálním úsilím postavit.

Prvním kandidátem byl emulátor QEMU. Orientace projektu je však spíše na UNIXové systémy, i když podpora Windows také existuje. Od počátku zkoumání QEMU však nastávaly problémy s kompilací celého produktu, což vyústilo v rozhodnutí QEMU opustit a případně se k němu opět vrátit později, pokud by situace u jiných produktů nebyla příznivější.

Dalším na řadě byl emulátor Bochs, zde se i přes drobné obtíže podařilo produkt zkompilovat. Bochs bez problémů fungoval a bylo v něm možné provádět i základní změny transformace v sledovací systém. Potíž však byla v prakticky nepoužitelném výkonu, který Bochs na běžném osobním počítači vykazoval. Pouhá instalace emulovaného systému zabrala celý den a samotný běh systému nebyl o nic rychlejší, prakticky zcela nepoužitelný. Na základě těchto zjištění byl přehodnocen návrh sledovacího systému. Základní myšlenka nového návrhu, který počítal s výkonem Bochse, byla neemulovat celý systém, ale pouze testovaný vzorek. Pro běh jediného (nebo několika málo) procesů by byl výkon Bochse dostačující. Návrh počítal s tím, že všechny funkce emulovanému procesu dodá sledovací systém z hostujícího operačního systému. Další úpravy v kódu Bochse nasvědčovaly tomu, že tento nový návrh je realizovatelný. V této fázi byla práce s Bochsem přerušena s tím, že použitelnost je lepší než u QEMU, nicméně by bylo dobré se zkusit poohlédnout ještě po dalších možnostech dříve, než padne konečné rozhodnutí.

Na poli virtualizace není příliš mnoho produktů, jejichž zdrojový kód by byl dostupný. U produktů řady VMware nebo Virtual PC a Virtual Server zdrojové kódy dostupné nejsou a jejich využití tedy nebylo možné. Poslední možností byl VirtualBox. I s jeho kompilací byly z počátku problémy, ale stejně jako u případu Bochse se je podařilo úspěšně překonat. Zdrojové kódy VirtualBoxu se na rozdíl od Bochse vyznačují dobrou úpravou kódu a lepšími komentáři kódu. Část VirtualBoxu je ale založena na zdrojových kódech QEMU, kde kvalita a úprava kódu a především komentářů je nesrovnatelně horší než u zbytku systému. Bohužel právě části založené na QEMU se ukázali být pro implementaci sledovacího systému jako stěžejní. Zkoumání kódu VirtualBoxu potvrdilo, že implementace sledovacího systému je možná, kód je kvalitnější než u Bochse a především výkon systému je takový, že by se dal virtualizovat i celý systém, přesto však výkon nebyl nikterak oslnivý a zdálo se, že sledovací systém bude relativně pomalý, stále těžko prakticky použitelný. Změny v kódu VirtualBoxu za účelem přerušování výkonu v definovaných stavech procesoru totiž bylo možné vykonat pouze s vypnutím řady optimalizací, což vedlo právě ke snížení výkonu.

Zde je třeba zmínit, že výzkumu úprav zdrojových kódů VirtualBoxu pro účely implementace vlastních funkcí se věnoval i kolega Martin Dráb. Výsledkem jeho práce byla modifikace VirtualBoxu, která umožňovala napojení vlastního kódu na kód VirtualBoxu tak, že nový kód dostal řízení v předem definovaných stavech virtuálního procesoru. Ovšem nejpodstatnějším výsledkem jeho práce byla optimalizace výkonu. S Martinovými modifikacemi mohl sledovací systém vykazovat dostačující výkon.

Martin Dráb svolil, aby výsledek jeho práce byl použit pro účely této práce jako základ sledovacího systému. Tím se rozhodlo o použití VirtualBoxu jako základu pro sledovací systém, neboť předchozí pokusy s kódem QEMU a Bochse nedosahovaly srovnatelných výsledků jako Martinova modifikace VirtualBoxu. Díky Martinově práci tedy prezentovaný ukázkový sledovací systém má prakticky použitelný výkon. Pokud by se tato modifikace nepoužila, pravděpodobně by celý systém rovněž vznikl na základě kódu VirtualBoxu, výkon systému by však byl výrazně nižší.

6.2 Napojení vlastního kódu na VirtualBox

Ukázkový systém je postaven na VirtualBoxu OSE verze 1.6.2. Tato verze VirtualBoxu je již poměrně stará, ale pro sledovací systém se ukázala jako plně postačující. Změny v kódu VirtualBoxu jsou dvojího druhu. V první řadě se jedná o změny, které umožňují napojení vlastního kódu prostřednictvím knihoven, které jsou načteny do procesu virtuálního stroje VirtualBoxu, kterým je `VirtualBox.exe`. Druhým typem změn jsou změny, které umožňují v definovaných stavech přerušit chod virtualizátoru a předat řízení sledovacímu systému. Všechny modifikace jsou jednotně označeny komentářem, který začíná řetězcem `CHANGE-PABOV`, u delších úseků je konec označen řetězcem `\ CHANGE-PABOV`. Proto lze snadno dohledat všechny změny oproti originálnímu kódu. Navíc jména funkcí, konstant a globálních proměnných, které patří PABOVu, jsou důsledně prefixovány řetězci `pa_`, `PA_` nebo `Pa`.

6.2.1 Změny umožňující kompilaci a volání vlastních funkcí z kódu VirtualBoxu

Změn prvního typu je hned několik:

- Každý zdrojový soubor, který se odvolává na nějakou funkci knihovny PABOVu musí použít (include) hlavičkový soubor `include/VBox/pabov.h` tím, v sobě obsahuje řádek `#include <VBox/pabov.h>`. Tento hlavičkový soubor obsahuje informace o všech funkcích PABOVu, které kód VirtualBoxu může volat.
- Změny v kódu VirtualBoxu jsou uvozeneny preprocesorovou podmínkou na existenci definice makra `PA_PABOV_ENABLED`. To je možné definovat v souboru `pabov/src/common/debug.h`. Pokud je definováno, zdrojový kód VirtualBoxu je kompilován včetne změn pro PABOV, v opačném případě se kompiluje pouze původní kód.
- V souboru `src/VBox/Main/USBControllerImpl.cpp` bylo potřeba určitou pasáž v kódu zakomentovat, aby se kód vůbec zkompiloval.
- Před použitím funkcí knihoven PABOVu v kódu VirtualBoxu je potřeba zavolat inicializační funkci **pa_init**. Při skončení práce s knihovnami voláme funkci **pa_finit**. Obě funkce implementuje soubor `src/VBox/VMM/pabov.cpp`.

- V souboru `src/VBox/VMM/VM.cpp` jsou implementovány funkce nejvyšší vrstvy virtuálního stroje. Funkce **VMR3Create** vytváří objekt virtuálního stroje. V případě úspešného vytvoření stoje se volá inicializační funkce **pa_init**. Funkce **VMR3Destroy** ukončuje objekt virtuálního stroje. Zde voláme funkci **pa_finit**.
- Kód souboru `src/VBox/VMM/VM.cpp` se přeloží do knihovny VirtualBoxu `VBoxVMM.dll`. Kód souboru `src/VBox/VMM/pabov.cpp` tedy také musí být přeložen do tohoto modulu, proto je potřeba změny kompilačních skriptech. Soubor `src/VBox/VMM/Makefile.kmk` je tedy doplněn o kompilaci `src/VBox/VMM/pabov.cpp`.
- Dalším modulem VirtualBoxu, u kterého potřebujeme, aby mohl volat funkce knihoven PABOVu, je `VBoxREM.dll`. Tento modul implementuje rekompilátor instrukcí virtualizovaného stroje založený na QEMU. Rekompilátor se používá na základě analýzy kódu virtualizovaného stroje. Po rekompilaci problémových pasáží lze kód spouštět přímo na hostujícím procesoru, jak bylo popsáno v kapitole o metodách virtualizačních systémů. Inicializace, kterou provádí funkce **pa_init**, má efekt pouze pro modul, ve kterém je obsažen kód této funkce. Proto je potřeba volat **pa_init** i z modulu `VBoxREM.dll`. Funkci **pa_init** voláme z inicializační funkce rekompilátoru **REMR3Init** v souboru `src/recompiler/VBoxRecompiler.c`. Funkci **pa_finit** pak voláme v témže souboru z funkce **REMR3Term**, která ukončuje práci rekompilátoru.
- Podobně jako předchozím případě, i zde je potřeba upravit kompilační skript, aby se kód `src/VBox/VMM/pabov.cpp` dostal do modulu `VBoxREM.dll`. Příslušnou úpravu obsahuje soubor `src/recompiler/Makefile.kmk`.
- Navíc kód `src/VBox/VMM/pabov.cpp` používá hlavičkových souborů z Windows Platform SDK, proto je nutné provést změnu také v kompilačním skriptu `kBuild/tools/VCC80X86.kmk` tak, aby potřebné hlavičkové soubory byly dostupné.
- Později při vývoji docházelo k pádu procesu VirtualBoxu. K vyřešení problému postačilo v kódu knihovny Qt, kterou VirtualBox používá k vykreslování grafického rozhraní, v souboru `qt-3.3.x-p8/src/kernel/qpixmap_win.cpp` přidat ověření použitelnosti ukazatele na referencovanou paměť.

Po aplikaci těchto změn se při startu virtuálního stroje volá inicializační funkci **pa_init**, která načte do procesu `VirtualBox.exe` knihovny implementující sledovací systém PABOV a z nich získá adresy funkcí, které se volají z dalšího kódu VirtualBoxu.

6.2.2 Změny umožňující sledování dění virtualizovaného stroje

VirtualBox se snaží co nejvíce kódu virtualizovaného stroje vykonat přímo na hostujícím procesoru. Pokud se tak děje, mluví se v rámci VirtualBoxu o tzv. *čistém módu* (raw mode). Pokud analýza kódu odhalí kód, který nelze vykonat přímo, VirtualBox použije rekompilátor a běží v tzv. *módu REM*. Pro implementaci sledovacího systému postačuje, pokud budeme umět dostat řízení, kdykoliv hodnota registru EIP virtualizovaného procesoru splní nějakou podmínku. To se ukázalo jako poměrně snadné v případě, že VirtualBox je v módu REM. V opačném případě se například díky hardwarově asistované virtualizaci nedá snadno vykonávání kódu přerušit.

V módu REM VirtualBox analyzuje kód a vytváří tzv. *překladové bloky* (Translation Blocks). Tyto bloky jsou tvořeny instrukcemi, které již lze vykonat přímo na hostujícím procesoru,

jelikož jsou všechny problematické instrukce nahrazeny. V kódu začíná překladový blok libovolnou instrukcí a končí vždy u instrukce, která může provést skok. Zkonstruované překladové bloky nejsou po použití zahazovány. Z důvodu zvýšení výkonu se vyplatí uchovat jednou vytvořené překladové bloky a v případě, že se procesor dostane na stejnou adresu znovu, použít již hotový. Bloky jsou indexovány hašovací tabulkou, takže jejich vyhledání je rychlejší než případná zcela nová konstrukce. Překladové bloky jsou pak volány jako funkce. Navíc v některých případech jsou existující bloky přímo spojeny instrukcí skoku, takže jedno volání překladového bloku může vykonat i více těchto bloků. Občas je potřeba již hotové překladové bloky zničit. Děje se tak například v případě, že kód paměti, kterou překladový blok pokrývá, byl změněn nebo v případě, že hašovací tabulka je přeplněná.

Realizace možnosti přerušit chod virtualizace na základě podmínky na hodnotu EIP procesoru je možná například tak, že upravíme kód VirtualBoxu, aby definované adresy přerušení nikdy nebyly uprostřed překladového bloku. To znamená, že při analýze označíme adresy, na kterých chceme získat řízení tak, jakoby zde byla instrukce skoku, kterou překladový blok končí. Pak můžeme upravit vykonávání překladových bloků tak, že kdykoliv se má vykonat překladový blok, který začíná instrukcí na adrese, kterou chceme přerušit, předáme řízení PABOVu. Navíc je potřeba zamezit spojování bloků v případě, že se nějaký blok má spojit s blokem, jehož adresa má být přerušena.

Při zkoumání vlivu různých optimalizací na výkon VirtualBoxu bylo rozhodnuto, že sledovací systém bude postaven jen za pomocí přerušení v kódu jádra virtualizovaného procesoru. Tím lze lehce zvednout výkon sledovacího systému. Pokud by se kdykoliv ukázalo, že toto rozhodnutí je příliš omezující, lze implementovat další úpravy tak, aby bylo možné přerušovat i výkon uživatelského kódu.

Na základě výše uvedených zkoumání byly provedeny následující změny vedoucí k možnosti spolehlivě přerušit chod virtualizace na základě podmínky na hodnotu EIP virtualizovaného procesoru:

- Kód v souboru `src/VBox/VMM/EM.cpp` implementuje tzv. *manažera vykonávání kódu* (Execution Monitor/Manager). Tato komponenta se stará o běh virtuálního stroje a o nastavování stroje do různých běhových módů. Zde jsme ve funkci **EMR3Init**, která inicializuje manažera vykonávání kódu nad daným objektem virtuálního stroje zcela zakázali pro daný virtuální stroj přepnutí do čistého módu.
- Ve funkci **cpu_exec** v souboru `src/recompiler/cpu-exec.c` je implementována smyčka, ve které dochází k výkonu překladových bloků. Na základě stavu procesoru se zde také zjišťuje, zda je možno přejít z čistého módu do REM. Tento přechod je zakázán pro adresy kódu, který je hákován.
- Dále je ve funkci **cpu_exec** pro každý překladový blok, který se má vykonat, zjištěno, zda je pro něj definováno přesměrování. Pokud ano, je zakázáno jeho spojování s dalšími bloky a předá se řízení PABOVu.
- Funkce **cpu_exec** jsou v souboru `src/recompiler/cpu-exec.c` dvě, změněna je pouze první verze, která se používá ve VirtualBoxu.
- Kód v souboru `src/recompiler/target-i386/translate.c` implementuje tvorbu překladových bloků. Ve funkci **gen_intermediate_code_internal** je provedena změna, která zajistí, že pokud překladový blok pokrývá adresu, která je pro sledovací systém zajímavá, pak instrukce na této adrese bude první instrukcí, kterou překladový blok pokrýva. Nestane se tedy, aby instrukce, na jejíž adrese chceme vykonávání kódu přerušit, byla uprostřed překladového bloku.

7 PABOV

V této kapitole se seznámíme s ukázkovou implementací základu sledovacího systému nazvaného Process Analyzer Based on VirtualBox (PABOV). V prezentované verzi není PABOV dokončen tak jako ostatní popisované systémy – nemá propojen vstup ani výstup s webovým rozhraním, tvoří pouze výstup ve formátu HTML a konečná analýza nasbíraných dat používá pouze několik vybraných událostí. Skutečně se tedy jedná o základ sledovacího systému, nikoliv o hotový systém. Přesto i v této fázi vývoje má PABOV co nabídnout ve srovnání s konkurenčními projekty. Na této závěr kapitoly je uveden postup pro sestavení a instalaci systému.

Příloha B této práce pak popisuje strukturu implementace PABOVu v jeho zdrojových kódech.

7.1 Základní vlastnosti systému

Základní myšlenka PABOVu je sledovaní celého virtualizovaného systému na úrovni jeho jádra tak, abychom v každém okamžiku měli k dispozici informace o všech důležitých objektech v systému. To je poměrně radikální přístup k problému, se kterým se u podobných systémů příliš nesetkáme. Základem většiny sledovacích systémů je rozprostření nějakého typu háků, jejichž kód pak při získání řízení analyzuje situaci, která nastala a snaží se zjistit co nejvíce informací na základě aktuálního stavu paměti. Hákování navíc bývají často v uživatelském režimu, což znemožňuje zisk informací o objektech jádra. PABOV také implementuje jistý typ hákování, viz kapitolu 4.1.1, ale velké množství háků je nainstalováno jen z důvodu podchycení práce s objekty jádra tak, abychom zajistili korektnost informací, které si PABOV uchovává o stavu virtualizovaného systému. Tím, že PABOV zná stav sledovaného systému tak detailně, má možnost odhalovat vztahy mezi objekty v systému, které jinak nejsou zřejmé. Na druhou stranu tento přístup s sebou nese řadu problémů, z nichž dva nejpodstatnější jsou veliká složitost systému a také úzká specializace na konkrétní verzi systému.

Sledování výhradně v jádře má několik důsledků. Jak již bylo vysvětleno v úvodu práce, všechny interakce sledovaného procesu musí probíhat přes jádro. Sledování jádra tak pro zjištění důležitých aspektů chování plně postačuje. Na druhou stranu u řady operací je míra abstrakce v jádře velmi nízká, takže i triviální volání knihovní funkce v uživatelském režimu může mít za následek velmi složité interakce v jádře. I tento fakt z části způsobuje další problém, kterým je velká velikost výstupu sledovacího systému založeného pouze na sledování jádra. Namísto koncentrace na jediný proces, se do systému dostávají události od všech jeho částí. Systém je tak zahlcen informacemi o událostech, které vůbec nejsou relevantní z hlediska požadovaného výstupu. Toto však není pouze negativní vlastnost. Možnost sledovat události, jejichž původ je v jádře systému je jedinou možností, jak alespoň částečně zjistit chování malware, který nějakým způsobem docílí vykonávání svého kódu v režimu jádra. Pro vzorky pracující čistě v uživatelském režimu se navíc eliminuje možnost vyhnout se sledování.

Další komplikací, která doprovází sledování v jádře, je nedostupnost dokumentace řady funkcí a míst v systému, která jsou pro sledování zajímavá. Systémy sledující běh

v uživatelském režimu se mohou opřít o rozsáhlou dokumentaci většiny funkcí a jejich parametrů. V případě PABOVu bylo často nutné provést analýzu implementace různých částí jádra. Takový přístup je samozřejmě komplikovanější a náchylnější k chybám.

Při návrhu byl ohled brán i na možnost detekce sledovacího systému. Možnosti detekce virtuálního prostředí jsme již zmínili dříve. Implementace VirtualBoxu je však poměrně kvalitní a ze známých a používaných technik pro detekci virtualizace jich naprostá většina přítomnost VirtualBoxu neodhalí. K lepšímu skrytí napomůže i správné nastavení a úpravy ve virtualizovaném systému. Například lze zcela vypnout všechny procesy, které běžně ve virtualizovaném systému VirtualBox provozuje za účelem poskytnutí řady funkcí, které zpříjemňují uživatelům práci s virtuálním stojem. Tyto funkce však nejsou ve virtualizovaném stroji potřeba při jeho využití pro sledování chování vybraných procesů.

7.2 Důležité aspekty návrhu

V této části rozebereme návrh systému PABOV. Dále uvedeme všechny komponenty PABOVu a jejich vzájemné propojení. Jedna instance systému PABOV pracuje pouze s jedním virtuálním strojem VirtualBoxu. Tento stroj je uchováván ve stavu tzv. *živého snímku* (live snapshot), což znamená, že na stroj byl nainstalován operační systém, ten byl nakonfigurován do stavu, který vyhovuje pro účely sledovacího systému, a v plném provozu byl jeho stav uložen na disk hostujícího počítače. Tento uložený stav se pak při spuštění sledovacího systému načte a systém je tak plně funkční a připraven pro sledování během několika sekund. Použitím snímku šetříme čas, který by jinak byl potřeba k nastartování virtualizovaného stroje a operačního systému. PABOV aktuálně pracuje a je omezen na virtuální stroj s instalací 32-bitového systému Windows XP Service Pack 3.

7.2.1 Průběh testování vzorku krok po kroku

V následujících odstavcích popíšeme základní průběh testování vzorku, načež navážeme dalšími kapitolami, které rozeberou některé mechanismy detailněji.

Na počátku je VirtualBox vypnutý, virtuální stroj je uložen ve formě snímku, pro který navíc existuje soubor PABOVu, který popisuje stav operačního systému tohoto stroje. Dále je na disku hostujícího počítače umístěna složka se soubory, které se mají analyzovat. *Uživatelská aplikace PABOVu* (`pabov.exe`) je spuštěna s parametry, které specifikují cestu k dané složce a čas, po který má testování probíhat. Uživatelská aplikace načte konfigurační soubor, ve kterém jsou udány informace o virtuálním stroji – například jméno snímku stroje, IP adresa jeho síťového rozhraní atd.

Prostřednictvím aplikace VirtualBox Command Line Management Interface (`VBoxManage.exe`) uživatelská aplikace spustí VirtualBox a načte virtualizovaný stroj ze snímku. Změny v kódu VirtualBoxu způsobí, že proces VirtualBox načte knihovny PABOVu, které implementují sledovací funkce. Tyto knihovny provedou inicializaci, při které si načtou stav operačního systému virtualizovaného stroje, který je právě spouštěn, z externího souboru (*databáze stavu*). VirtualBox spustí virtualizovaný stroj.

Virtualizovaný stroj obsahuje jedinou malou komponentu PABOVu. Knihovna `pa_ind.dll` je nainstalována ve virtualizovaném systému tak, že Windows Explorer načte tuto knihovnu při každém svém spuštění do vlastního procesu. Tato knihovna implementuje TCP server, který čeká na požadavky na síťovém rozhraní virtuálního stroje. Na tento server se připojí klient implementovaný jednou z knihoven PABOVu z procesu VirtualBoxu. Klient dále

lokálně namapuje sdílený disk virtuálního systému a zkopíruje přes něj složku s testovaným vzorkem do virtuálního systému. Pak pošle serveru ve virtuálním stroji informace o tom, který ze zkopírovaných souborů má spustit, a jaké má případně použít parametry. Běžným systémovým voláním knihovna PABOVu uvnitř Windows Exploreru pak spustí testovaný vzorek. Tím, že rodičovský proces testovaného vzorku je právě Windows Explorer, vypadá celá situace podobně jako v případě, kdy uživatel běžného systému spustí nějakou aplikaci.

O výsledku celé operace kopírování a spouštění je informována uživatelská aplikace PABOVu, která případné chyby nahlásí prostřednictvím chybového výstupu konzole uživateli. Pokud vše proběhlo bez problémů sledovací systém běží a uživatelská čeká aplikace než uplyne čas určený pro analýzu.

Knihovny PABOVu v procesu VirtualBoxu zachytávají předem definované stavy virtuálního stroje, ve kterých tvoří záznamy o dění v systému. Po vypršení času pro analýzu uživatelská aplikace prostřednictvím `VBoxManage.exe` ukončí virtualizovaný stroj. Při tom knihovny PABOVu zpracují nasbírané záznamy a vytvoří výstup systému. Systém je opět připraven pro další vzorek.

7.2.2 Realizace sledování

Jak již bylo zmíněno, pozměněné chování VirtualBoxu nám umožňuje získat řízení, když se virtualizovaný procesor chystá vykonat instrukci na určené adrese. Míst ve virtualizovaném systému, které chceme sledovat, je celá řada. Zavedení podmínky na získání řízení v jednom místě nazýváme v PABOVu hák. V prezentované verzi PABOV hákuje přes 130 míst. Tato místa jsou v kódu uchovávána jako relativní adresy instrukcí, na kterých má dojít k přerušení. Tím mimo jiné dochází k již zmiňovanému omezení na konkrétní verzi operačního systému.

Ke každé adrese háku je v PABOVu přidána i informace o počtu parametrů a umístění těchto parametrů v paměti ve stavu, ve kterém stroj bude při předání řízení PABOVu. Na rozdíl od nejběžnějšího hákování funkcí, kdy jsou háky instalovány tak, že řízení je přesměrováno před voláním hákované funkce, je řada háků PABOVu nainstalována doprostřed funkcí. S tím souvisí i to, že parametry se často nacházejí v lokálních proměnných a dočasně alokovaných úsecích paměti, kdežto v běžném případě jsou k dispozici na zásobníku aktuálního vlákna. Tyto skutečnosti opět posilují vazbu mezi implementací PABOVu a konkrétní verzí operačního systému. Na rozdíl od tradičního přístupu, nelze v případě PABOVu dynamicky rozpoznat adresy v neznámém jádře ani lokace důležitých parametrů. Důvodem takového návrhu PABOVu je snaha o minimalizaci počtu míst, které se hákují. Tato místa jsou pečlivě vybrána tak, aby v nich bylo možné získat co nejvíce informací důležitých pro daný stav.

7.2.3 Obsluha háků – stav virtuálního systému

Některé systémové události nejsou přímo důležité z hlediska sledování chování vzorku, ale jsou nainstalovány z toho důvodu, že jejich obsluha pomáhá získat informace o aktuálním stavu systému. Někdy je hák užitečný jak pro zisk informací o systému, tak pro sledování chování.

PABOV si při sledování dění ve virtuálním systému uchovává několik hlavních vlastních struktur, které odrážejí stav systému. Tyto struktury jsou realizovány jako hašovací tabulky s řetězením, seskupováním (jedna část řetězce neobsahuje jeden prvek, ale celou skupinu volitelné velikosti) a volitelnou propagací při vyhledávání (nalezený prvek se propaguje vpřed v řetězci).

Nejdůležitější struktura je tabulka živých objektů jádra virtualizovaného systému. Object Manager virtuálního jádra spravuje objekty, jak bylo nastíněno v úvodu práce. PABOV chce mít informaci o každém takovém objektu u sebe. Objekty jsou indexovány jejich virtuální adresou, která je platná v paměti virtualizovaného stroje. Jelikož však v průběhu času může objekt zaniknout a nový objekt vzniknou tak, že jeho virtuální adresa je totožná s adresou předchozího, udržuje si PABOV navíc vlastní unikátní identifikátor těchto objektů. Dále si PABOV u objektu pamatuje jeho typ a případně i jméno, pokud objekt nějaké má. U některých objektů se ještě uchovávají další specifické informace. Operace nad objekty jádra se na nízké úrovni provádějí pomocí ukazatele na objekt, tj. pomocí jeho virtuální adresy. Obsluha háku, který zachycuje takovou operaci, pak má velmi rychlý přístup k základním údajům o objektu.

Na vysoké úrovní v jádře a na úrovni kódu v uživatelském režimu se s objekty pracuje pomocí handle. Každý proces v systému a navíc systém samotný ma vlastní tabulku platných handle, pomocí které kód v jádře získá virtuální adresu objektu. Jelikož některé háky PABOVu jsou instalovány na vysoké úrovni v jádře, má i PABOV vlastní strukturu, která páruje handle a objekty. V této struktuře je indexem dvojice hodnota handle a systémový identifikátor procesu, v rámci kterého má být hodnota platná.

Třetí strukturou je tabulka procesů a vláken virtuálního systému. Tato struktura je indexována systémovým identifikátorem procesu nebo vlákna. U procesů je ve struktuře udržován identifikátor rodiče procesu, u vlákna jde o identifikátor procesu, do kterého vlákno patří. Dále struktura obsahuje ukazatel na objekt procesu nebo vlákna. U procesů pak navíc je udržován spojový seznam modulů, které jsou načteny do daného procesu. Prvek tohoto seznamu obsahuje jméno modulu, virtuální adresu modulu v paměťovém prostoru procesu a velikost modulu v paměti. U vláken je udržován spojový seznam tzv. *čekacích stavů* (wait state). Řada operací v systému vyžaduje synchronizaci, ta je v systému realizována čekáním na jeden nebo více synchronizačních objektů. Volající vlákno je v takovém případě pozastaveno do doby, než je objekt, na který se čeká, aktivní a poté vlákno pokračuje ve své činnosti. Čekací stav vlákna popisuje objekty, na které vlákno čeká, typ, důvod čekání a některé další informace.

Čtvrtou strukturou je tabulka nedokončených operací. Některé operace v systému nelze dokončit ihned. Prvním možným řešením je pozastavení činnosti volajícího vlákna do doby, než je požadavek vyřízen. U některých operací existuje ještě druhá možnost – asynchronní vyřízení požadavku, což znamená, že požadavek je systémem zaregistrován, ale vlákno může pokračovat v činnosti. Po dokončení požadavku systém dá nějakým způsobem najevo, že operaci dokončil. V obou případech PABOV uchovává v tabulce nedokončených operací informaci o těchto operacích při jejich zadání a z tabulky je odstraňuje při jejich dokončení. U nedokončené operace si PABOV udržuje ukazatel na požadavek (IRP), pokud je tento k dispozici, a tímto ukazatelem také indexuje ve struktuře. Pokud k dispozici není, pak se jako klíč bere hodnota bázového registru procesoru (EBP na IA-32), který poměrně jednoznačně určuje vlákno, které operaci provádí a zároveň určuje jeho pozici na zásobníku, jelikož při volání funkcí je bázový registr upravován, aby udržoval kontext proměnných a parametrů funkce v rámci aktuálního volání. Výjimečně se namísto hodnoty bázového registru vezme pouze identifikátor vlákna. Dále se u nedokončené operace uchovává informace o typu operace, kontrolním kódu operace, souborovém objektu, pro který je operace prováděna, vstupní data a výstupní data operace a některé další detaily.

Jelikož virtualizovaný systém nestartuje od začátku, ale je načítán ze snímku, který byl pořízen uprostřed jeho běhu, je nutné nějakým způsobem naplnit výše zmíněné struktury informacemi o systému ve stavu snímku. K tomu se používá externí soubor (databáze stavu), který je vytvořen společně se snímkem. V tomto souboru jsou uloženy informace tak,

aby jejich načtení odpovídalo stavu, ve kterém je virtualizovaný systém po načtení ze snímku.

7.2.4 Obsluha háků – události pro analýzu chování

U řady událostí v systému je potřeba více než jeden hák k tomu, abychom řízení dostali při každé možné kombinaci, která může při zpracovávání dané události nastat. Takové háky pak mají společnou obsluhu. Ostatní háky mají obsluhy vlastní.

Zaměřme se nyní na obsluhy háků, které se přímo účastní sledování chování vzorku. Úkolem těchto obslužných funkcí vzhledem k analýze chování vzorku je sestavit jeden nebo více tzv. *objektů událostí* a přidat je do seznamu těchto objektů. Objekt události je jednotná struktura navržená tak, aby se s její pomocí dala popsat libovolná akce, která v systému nastala a která může být relevantní vzhledem k analýze chování. Při běhu vzorku ve virtualizovaném systému se objekty událostí pouze vytvářejí a zařazují do seznamu. Až po skončení této fáze a ukončení běhu virtuálního stroje se tato data dále analyzují a vytváří se výstup systému.

Objekt události je struktura, která obsahuje řadu údajů, z nichž většina nemusí být vyplněná. Mezi povinné položky patří systémové identifikátory vlákna a jeho procesu, které událost vyvolaly (tzn. v jejich kontextu se volala obsluha háku) a unikátní identifikátory objektů vlákna a procesu v rámci PABOVu. Dalšími a posledními povinnými položkami jsou typ události a logický a fyzický čas objektu události – od kdy a do kdy událost trvala. Jako logický čas je bráno chronologické pořadí události.

Obslužná funkce háku se vždy snaží do objektu události poznamenat co nejvíce informací. Informace může obsluha získat ze tří zdrojů. Za prvé jsou to parametry háků, které jsou obsluze přímo předány. Za druhé to jsou struktury, které jsme zmínili v předchozí kapitole. Poslední možností je dodatečné čtení paměti virtualizovaného stroje. Pokud je tak například hákována funkce jádra na vysoké úrovní, kde se ještě pracuje s handle nějakého objektu, pak je využita struktura, která páruje handle a objekty, a do objektu události se zkopírují z tabulky objektů některé informace jako například jméno a identifikátor objektu.

7.3 Zpracování objektů událostí a struktura výstupu

Po vypršení času určeného na zkoumání vzorku je virtuálnímu stroji předán pokyn na ukončení. Přitom dochází ke spuštění kódu PABOVu, který má za úkol vytvořit výstup na základě událostí nasbíraných při běhu virtuálního stroje. V této fázi se vytváří nové struktury, které pomáhají vypozorovat nejrůznější souvislosti dějů ve virtualizovaném systému a generovat výstup.

7.3.1 Důsledky chování vzorku

PABOV má k dispozici kompletní seznam všech událostí, které ve virtualizovaném systému nastaly od jeho startu. Ve finální analýze tak figurují i data, která byla nasbírána před nahráním a spuštěním samotného vzorku. Toho se využívá například ke konstrukci seznamu procesů, jejich vláken a modulů. Pokud bychom tato data neměli, nemohli bychom v další analýze pracovat s objekty, které v systému existovali před spuštěním vzorku. Pokud by pak například testovaný vzorek infikoval již existující proces v systému, nevěděli bychom o tomto procesu nic. Takto máme v každém okamžiku k dispozici struktury, které umožňuji podat přesné informace o jednotlivých událostech.

Při probírání seznamu událostí se naplňuje také struktura všech objektů v systému. Ty se

většinou rozlišují a dohledávají pomocí unikátních identifikátorů. Výjimku tvoří pojmenované objekty systému, pro které jádro systému vytváří více instancí na základě nějakého kontextu. Nejjednodušším příkladem takových objektů jsou soubory na disku. V jádře má každý separátní přístup k souboru svůj objekt, ale v logice naší analýzy je potřeba sjednotit tyto objekty pod jeden. Naopak například u procesů je jejich systémové označení pomocí identifikátoru nejednoznačné tím způsobem, že pokud proces zanikne, může být v budoucnosti spuštěn jiný proces, který dostane stejné označení. U procesů je tak vždy použit pouze unikátní identifikátor a nehledí se na systémový identifikátor nebo jméno hlavního modulu.

Každý ze zmiňovaných objektů může být ovlivněn chováním testovaného vzorku. Abychom koncového uživatele uchránili před množstvím zbytečných údajů ve výstupu, zaměřuje se PABOV při tvorbě výstupu často pouze na události, které ovlivňují nějaké objekty. Ostatní události mohou pomoci při analýze, ale jejich výčet není uživateli předložen. Například u přístupu k registru PABOV pokládá za důležité informovat uživatele o všech klíčích a hodnotách, které mohli být testovaným vzorkem vytvořeny nebo změněny. Naopak pouhé čtení hodnot z registru se považuje za operaci pro koncového uživatele nezajímavou. Tímto výběrem zajímavých událostí se výrazně zlepšuje čitelnost výstupu.

Při analýze seznamu událostí je jako první objekt, který je označen za ovlivněný, brán proces vzorku. Na základě jeho chování se pak z tohoto procesu šíří označení i na další objekty. Téměř výhradně se označení šíří pouze přes události, které mohou vést k změně chování dalších procesů. Je-li například zaznamenáno vytvoření souboru na disku nebo jeho změna způsobená označeným procesem, je i daný soubor označen za ovlivněný. Pokud kdykoliv později nějaký neoznačený proces čte tento označený soubor, má se za to, že změna, kterou provedl označený proces může vést ke změně chování neoznačeného procesu, který je tedy na základě čtení souboru označen. V jiných případech je proces ovlivněn přímo, například pokud označený proces vytvoří nový proces, nebo změní paměť neoznačenému procesu, je tento automaticky označen.

Výstup PABOVu pak obsahuje pouze procesy, které jsou označeny, a pouze akce těchto procesů, které nastaly po jejich označení.

7.3.2 Aktuální pokrytí na výstupu

V prezentované verzi je finální analýza PABOVu poměrně chudá a pokrývá pouze základní operace. Jádro PABOVu, zvláště pak hákování, pokrývá o mnoho více jevů v systému. Je pouze otázkou dalšího rozvoje, aby i finální analýza a výstup pokrývaly další oblasti.

Aktuální verze plně pokrývá vytváření a ukončování procesů a jejich vláken, načítání modulů do procesů, vytváření a mazání souborů a registrových klíčů, přejmenovávání souborů a registrových klíčů, čtení a zápisy do souborů, nastavování, čtení a mazání registrových hodnot, alokace a uvolňování paměti, změny ochrany paměti, kopírování paměti, některé operace se sokety (asociace s adresou, připojení, poslouchání, přijímání připojení, odpojení, odeslání a přijetí dat).

Řada jevů je zachycena jako události, ale jejich zpracování není ve finální fázi implementováno. U nich je potřeba pouze dopsat kód, který je zpracuje a předá na výstup. U jiných jevů je naopak pouze implementováno hákování a k jejich pokrytí by bylo potřeba vytvořit události a následně i kód, který je zpracuje pro výstup. Ojediněle se pak může vyskytnout i potřeba zachytávat chování, které není pokryté hákováním. Tam by pak samozřejmě bylo nutno implementovat kompletně všechny tři vrstvy, aby došlo k jejich pokrytí.

PABOV také zatím téměř neimplementuje rozpoznání a filtraci běžných jevů v systému. Při ovlivnění systémového procesu se tak snadno může stát, že výstup je zamořen irelevantními daty.

7.3.3 Slabiny konkurenčních systémů

Před samotným návrhem tvorby výstupu bylo hojně pracováno s již existujícími sledovacími systémy za účelem rozpoznání silných a slabých míst v jejich výstupech. Ve výstupech PABOVu se tak snažíme zachovat silné stránky a eliminovat některá slabá místa.

Jedna z identifikovaných slabin všech konkurenčních systémů byla nedostupnost detailních informací o operacích. Dozvěděli jsme se tak například, že testovaný vzorek vytvořil spustitelný skript na disku, ale nezískali jsme informace o jeho obsahu. Výstup PABOVu se snaží uživateli všechna tato data poskytnout a to jak v originální binární formě, tak i ve formě textového náhledu.

Další velkou slabinou téměř všech systémů byla absence chronologického seznamu událostí. Často se tak setkáváme pouze se seznamy událostí zařazené do kontextu procesu, který je vykonal, ale v rámci dvou procesů, nebo často i v rámci událostí z různých kategorií, nelze určit, ke které události došlo dříve. Struktura výstupu PABOVu požadavek na chronologické uspořádání událostí respektuje. V první části výstupu jsou události řazeny dle jejich příslušnosti k procesu, který je vykonal. Zde jsou zaznamenány i detaily těchto událostí včetně odkazů na relevantní data. V druhé části je pak dán chronologický seznam všech ve výstupu obsažených událostí. Obě části jsou propojeny odkazy tak, že je snadné pro položku chronologického seznamu dohledat relevantní detaily a data a naopak pro položku v kontextu procesu je snadné dohledat její chronologické zařazení. Navíc i v první části se objevují logické časy událostí, takže i pouze v rámci této části je snadné rozhodnout, která ze dvou událostí nastala dříve.

7.3.4 Struktura výstupu

Výstupem PABOVu je HTML dokument, který je podpořen soubory s daty zaznamenaných událostí. Dokument začíná sekcí BASIC INFORMATION, kde jsou k dispozici základní údaje o analýze jako např. čas začátku a konce analýzy, jméno analyzovaného vzorku, argumenty pro proces vzorku. V této sekci je také uveden obsah adresáře, který byl zkopírován do virtuálního stroje.

PABOV se soustředí na odhalení chování typického pro malware. Proto na výstup dává téměř výhradně události, které nějakým způsobem ovlivňují systém a jeho objekty. Druhou částí výstupu nazvanou PROCESS MONITORING DEPENDENCY je strom označených procesů. Kořenem stromu je proces vzorku. Další procesy pak přibývají tak, jak byly označeny při finální analýze. Pokud například označený proces vytvořil soubor a jiný neoznačený proces z něj četl, bude tento proces označen a ve stromě bude synem procesu, který soubor vytvořil. Strom zároveň obsahuje i odkazy na další části výstupu.

Pro každý označený proces následuje ve výstupu jedna sekce, která se mu věnuje. Sekce začíná souhrnem informací o procesu. Je zde uvedena například plná cesta k hlavnímu modulu procesu, systémový identifikátor procesu a jeho rodiče, čas začátku sledování procesu (tj. čas události, na základě které se stal označeným), čas startu procesu a případně také čas ukončení procesu. Velmi důležitou informací je důvod sledování. Proces testovaného vzorku je sledován proto, že se jedná o hlavní cíl sledování. Ostatní procesy pak mají udaný důvod svého sledování. Vždy je tedy jasné, proč se proces stal součástí výstupu.

U každého procesu je pak uveden seznam modulů procesu a dále následují kategorie událostí, které proces vykonal. Pokud by nějaká kategorie byla prázdná, není z důvodu přehlednosti do výstupu vůbec přidána.

Poslední část výstupu je chronologický seznam událostí. Zde je uveden čas události a systémový identifikátor procesu a vlákna, který operaci vykonal, ale především je zde krátký a lidsky srozumitelný popis události. Pokud by uživatel požadoval více informací o události, má možnost se pomocí odkazu dostat k detailům, které jsou součástí záznamu události v kategoriích událostí u daného procesu.

7.3.5 Demonstrační výstup CmdLine a GetIP

Zde okomentujeme výstup PABOVu pro demonstrační programy CmdLine a GetIP, podobně jako jsme na těchto příkladech v kapitole 5 zhodnotili výstup konkurenčních sledovacích systémů. Jedná se tedy o výstup ze zkoumání vzorků, které nejsou škodlivé. Pro tyto vzorky byla doba běhu vzorku nastavena na 30 sekund a vzorky byly spouštěny bez dalších parametrů. Plný výstup je opět k dispozici na přiloženém médiu této práce.

CmdLine

Výstup k CmdLine začíná souhrnem základních informací o analýze. Z uvedených údajů plyne, že čas analýzy byl 44 sekund. Vzhledem k 30 sekundovému běhu vzorku to znamená, že inicializace virtuálního stroje a zpracování nasbíraných dat celkově trvalo 14 sekund.

V části PROCESS MONITORING DEPENDENCY je uveden pouze testovaný vzorek, což znamená, že nedošlo k ovlivnění chování jiných procesů v systému.

Následuje sekce vztahující se k procesu vzorku `cmdline.exe`. Zde vidíme, že proces běžel pouhé 3 sekundy. PABOV ale neimplementuje možnost předčasného ukončení analýzy z toho důvodu, že sledovaný proces mohl ovlivnit některé objekty v systému. V případě delšího běhu by tedy mohlo dojít k tomu, že se nějaký proces v systému stane označený na základě přístupu k označeným datům. V případě CmdLine se tak však nestalo. Pro proces vzorku je dále uveden seznam jeho modulů. Jedinou kategorií událostí, která je ve výstupu uvedena je kategorie VIRTUAL MEMORY ACTIVITY, která zaznamenává chování procesu vzhledem k virtuální paměti. V případě CmdLine je seznam událostí poměrně rozsáhlý a týká se výhradně operací čtení paměti. Z výstupu je patrné, že proces vzorku postupně četl paměť z ostatních procesů v systému. K dispozici jsou jména a identifikátory zdrojových a cílových procesů, adresy v obou procesech, velikost dat a odkaz na data v binárním formátu i textovém náhledu. V případě čtení paměti se jako zdrojový proces bere proces, ze kterého jsou data načtena, a jako cílový proces je označen ten proces, do jehož paměti jsou data načtena. Pokud by výstup neobsahoval odkazy na konkrétní data mohli bychom jenom odhadovat skutečný záměr vzorku. Jelikož ale data k dispozici máme, můžeme po krátkém zkoumání dat dospět k závěru, že CmdLine se u každého procesu v systému snaží zjistit jméno jeho hlavního modulu a argumenty, se kterými byl spouštěn.

Poslední částí výstupu je chronologický seznam událostí, který již nové informace nepřináší.

Z výstupu je zcela zřejmá skutečná činnost vzorku CmdLine.

GetIP

Pro vzorek GetIP má výstup podobnou strukturu. Analýza trvala 43 sekund a jediným označeným procesem byl proces vzorku, který běžel pouhé 3 sekundy.

V tomto případě však výstup obsahuje čtyři kategorie událostí pro proces vzorku. V části věnované procesům a vláknům (PROCESSES AND THREADS ACTIVITY) se dozvíme, že proces vzorku si vytvořil další čtyři vlákna. Následuje část FILES ACTIVITY, která udává aktivity vzorku vzhledem k souborům. Vzorek vytvořil v adresáři dočasných souborů Internet Exploreru soubor `getmyip[1].htm`, dále vytvořil ve svém adresáři soubor `getip.exe.tmp`, který následně smazal. Do obou těchto souborů byla zapsána tato data:

```
Your IP: 80.250.20.18.
```

Následuje kategorie přístupů k registru, kde je uvedeno velké množství registrových hodnot. Jedná se však o nerelevantní informace, jelikož tyto události má nejspíše na svědomí některá ze standardních systémových knihoven, kterou vzorek použil.

Poslední uvedenou kategorií je kategorie přístupu k síti. První tabulka udává seznam událostí, která mají co do činění s navazováním spojení. Zde jediná relevantní informace udává připojení procesu vzorku k IP adrese 67.225.206.92 na port 80/TCP. Druhá tabulka obsahuje záznam přenosů dat. Máme k dispozici celý HTTP GET požadavek na server www.diamondcs.com.au a také doručenou odpověď, která obsahuje právě data, která byla zapsána do souborů na disku.

Chronologický seznam na konci výstupu pak všechny nasbírané informace přehledně dává do časové souslednosti. Po načtení všech knihoven do procesu a jejich inicializaci, při které došlo k zmiňovaným zásahům do registru a vytvoření nových vláken v procesu, se vzorek připojil k Internetovému serveru, na který odeslal požadavek. Po obdržení odpovědi vzorek pracoval se soubory na disku, jak bylo uvedeno výše, a skončil.

Také z tohoto výstupu je činnost sledovaného vzorku naprosto zřejmá. Jediná nepříjemná věc ve výstupu je obsažení informací o nerelevantních událostech, které měla na svědomí standardní systémová knihovna. Při zkoumání časové souslednosti událostí však uživatel tyto informace dokáže identifikovat a odfiltrovat.

7.4 Pomocné prvky

Kromě stěžejních komponent systému PABOV, obsahuje implementace i některé prvky, které jsou nahraditelné nebo nejsou zcela nezbytné pro chod systému, nicméně napomáhají při jeho vývoji nebo provozu.

V aktuální verzi se jedná o tyto prvky: testovací programy, automatický generátor adres, speciální verze uživatelské aplikace a skripty, které se používají při tvorbě snímku virtuálního systému a k tomu přidružené databáze stavu, a instalátor knihovny PABOVu načítané do Windows Exploreru ve virtualizovaném systému.

7.4.1 Testovací programy

Testovací programy nejsou v aktuální verzi příliš používané, nicméně do budoucna se s nimi počítá. V PABOVu jsou dva typy testovacích programů. Jednou skupinou jsou programy, které přímo testují nějakou část kódu PABOVu – například implementaci datové struktury. Druhým typem jsou programy, které prověřují analytické schopnosti systému. Tyto programy se nechají analyzovat v PABOVu a výstup systému by pak měl odpovídat implementaci těchto programů.

7.4.2 Automatický generátor adres

Automatický generátor adres (`hlp_imgofs.exe`) se týká háků PABOVu. Již jsme zmínili, že aktuální verze PABOVu je úzce spjata se systémem Windows XP Service Pack 3 a že nelze jednoduše automaticky zjišťovat adresy háků ani lokace jejich parametrů v neznámých verzích systému. Nicméně Windows XP Service Pack 3 je k dispozici také v několika verzích, jejichž binární obrazy jádra se liší. Odlišnosti jsou však minimální, ale takové, že adresy háků pro jednu verzi nejsou platné pro verzi jinou. Automatický generátor adres je nástroj, který na základě hlavičkového souboru se seznamem adres háků v určitém formátu a k tomu příslušného jádra operačního systému dokáže k jiné verzi jádra vytvořit hlavičkový soubor s adresami platnými pro druhé jádro. To vše za předpokladu, že se jádra příliš neliší, což v praxi znamená, že důležité části kódu jsou totožné, až na posunutí relativní k modulu jádra. Pro každou adresu ze zdrojového obrazu jádra se hledá adresa v druhém jádře taková, že délky několika sousedních instrukcí v okolí dané adresy jsou v obou jádrech shodné a zároveň v nich tvoří jedinečné posloupnosti. U některých adres tak stačí analyzovat desítky okolních instrukcí, jinde se jedná o stovky. Celý proces při aktuálním počtu háků PABOVu trvá několik desítek sekund. PABOV tak není omezen na jednu konkrétní verzi systému Windows XP Service Pack 3, ale měl by být kompatibilní s většinou verzí tohoto označení.

7.4.3 Nástroje pro tvorbu generátoru databáze stavu

Při instalaci systému je potřeba vytvořit snímek a k němu příslušnou databázi stavu. K tomu je vytvořena speciální uživatelská aplikace (`pabov_db.exe`), která pouze nastartuje VirtualBox a virtuální stroj a informuje knihovny PABOVu v procesu VirtualBoxu o tom, že tento běh stroje má při svém ukončení uložit stav do externího souboru. Uživatel, který instaluje virtuální stroj pro potřeby PABOVu pak ručně restartuje virtuální systém, čímž se smaže obsah stavových struktur, který je udržován v knihovnách PABOVu. Zatímco systém startuje, PABOV sleduje dění a naplňuje své struktury. Uživatel nechá systém plně nastartovat a pak kdykoliv spustí skript, který pomocí `VBoxManage.exe` vytvoří snímek virtuálního stroje a ukončí jej, na což zareagují knihovny v procesu VirtualBoxu a uloží obsah struktur do určeného souboru databáze stavu.

7.4.4 Instalátor knihovny do Windows Exploreru ve virtualizovaném stroji

Jedná se o jednoduchý program, kterým uživatel připravující virtualizovaný systém pro potřeby PABOVu do něj nainstaluje pomocnou knihovnu PABOVu tak, že Windows Explorer spuštěný ve virtualizovaném systému načte a spustí kód této knihovny při každém svém startu. Jak již bylo zmíněno, tato knihovna pak vytvoří TCP server, na který se připojuje klientská knihovna PABOVu z hostujícího systému, a na základě požadavků spouští vzorky pro testování.

7.5 Sestavení, instalace a spuštění systému

Zprovoznit systém PABOV není jednoduché. Problémy mohou nastat už při snaze zkompilovat VirtualBox a ani poté není situace snadná. V této části popíšeme všechny kroky potřebné pro sestavení a instalaci PABOVu nad VirtualBoxem verze OSE 1.6.2.

7.5.1 Sestavení VirtualBoxu

Nejprve je potřeba stáhnout zdrojové kódy VirtualBoxu OSE 1.6.2. Ty se dají pořídit ze stránky http://www.virtualbox.org/wiki/Download_Old_Builds_1_6 jako VirtualBox 1.6.2 OSE tarball. Dále je třeba tyto zdrojové kódy sestavit. K tomu napomáhají instrukce na stránkách http://www.virtualbox.org/wiki/Build_instructions a http://www.virtualbox.org/wiki/Windows_build_instructions. Na druhé jmenované stránce je seznam prerekvizit. Opravdu je potřeba přesně dodržet stanovené verze, jinak se sestavování stane velmi problematické. I při dodržení instrukcí jsou některé části obtížně sestavitelné, nicméně na stránkách produktů označené jako prerekvizity jsou často další instrukce, pomocí nichž lze celý problém zvládnout.

Po úspěšném sestavení VirtualBoxu by mělo být možné výsledné spustitelné soubory normálně používat. Jelikož však nedojde k instalaci produktu, nastane problém s ovladači produktu, které je třeba dle návodů nainstalovat do systému. Jedná se především o ovladač pro virtuální sítě a o ovladač, který zajišťuje základní funkcionalitu VirtualBoxu. Pokud by problémy s instalací ovladačů přetrvávali, je nejjednodušším řešením stáhnout binární distribuci VirtualBoxu 1.6.2 a tu nainstalovat a poté její soubory přepsat soubory sestavenými ze zdrojových kódů.

V této fázi by mělo fungovat grafické rozhraní VirtualBoxu a mělo by být možné vytvořit virtuální stroj a nainstalovat do něj operační systém.

7.5.2 Sestavení PABOVu

Do složky s VirtualBoxem zkopírujte soubory PABOVu. Tím dojde k přepsání některých souborů VirtualBoxu, čímž se aplikují změny uvedené v kapitole 6.2. Nyní můžete zkusit sestavit VirtualBox se změnami. Nahrání nových souborů do složky, odkud chcete spouštět VirtualBox, ponechte na později – až po instalaci a konfiguraci operačního systému do virtuálního stroje.

Nyní příkazem `make` z distribuce MinGW ([30], vyzkoušena byla verze GNU Make 3.80 společně s GCC 3.4.5 mingw-vista special r3) spuštěným v adresáři `pabov/src` sestavíte soubory PABOVu. V adresáři `pabov/bin` se tak objeví soubory včetně `pabov_host.exe`, `pa_log.dll` a `pa_ind.dll`. Ujistěte se, že PABOV je sestaven se správnou cestou logovacího souboru – viz definici makra `PA_LOG_FILE` na řádku 144 v souboru `pabov/src/log/log.h`.

7.5.3 Instalace a konfigurace operačního systému

V prezentované verzi PABOV přímo podporuje 32-bitové operační systémy Windows XP Service Pack 3 verze jádra 5.1.2600.5512 a 5.1.2600.5657 anglické lokalizace. Pokud máte jinou verzi Windows XP Service Pack 3, můžete ji zkusit nainstalovat a poté použít automatický generátor adres a tak přidat podporu pro novou verzi. Není však zaručeno, že generátor uspěje.

Běžným způsobem nainstalujte operační systém do virtuálního stroje. Virtuální stroj určený pro PABOV musí být nutně vybaven virtuální síťovou kartou. Je nutné ověřit, že hostující a virtuální systém jsou navzájem dostupné přes lokální síť. V případě problémů je možné zkusit na virtuálním stroji vypnout firewall operačního systému. Konfigurace jednotlivých komponent operačního systému a instalace software třetích stran je čistě na potřebách administrátora PABOVu, kromě nastavení uvedených v této kapitole. Je vhodné nainstalovat balík VirtualBox Guest additions. Nebylo ověřováno, zda systém PABOV funguje

i bez instalace tohoto balíku.

Vytvořte ve virtuálním systému složku vyhrazenou pro PABOV – např. `C:\PABOV`. Nyní do virtuálního systému zkopírujte soubory `pabov_host.exe`, `pa_log.dll` a `pa_ind.dll` z `pabov/bin`. Vytvořte složky, do které se budou kopírovat testované vzorky – např. `C:\PABOV\Incoming`. Spusťte `pabov_host.exe` tak, že jako první argument uvedete cestu k této složce, jako druhý argument uvedete název pro sdílení této složky v rámci sdílení souborů Windows a jako třetí argument uvedete plnou cestu ke knihovně `pa_ind.dll`. Příkaz tak může vypadat například takto:

```
pabov_host.exe C:\PABOV\incoming PABOV C:\PABOV\pa_ind.dll.
```

Po úspěšné instalaci můžete všechny tři soubory smazat. Dále zapněte automatické přihlašování uživatele. Jak na to popisuje dokument na stránce http://support.microsoft.com/kb/315231. Nyní systém restartujte a uložte jeho živý snímek.

7.5.4 Nastavení PABOVu

Soubory `pabov.conf`, `pa_log.dll`, `pa_main.dll`, `pa_mgr.dll`, `pabov.exe` a `pabov_db.exe` zkopírujte do složky VirtualBoxu. Upravte konfigurační soubor PABOVu `pabov.conf` dle následujících pokynů.

Každý řádek konfiguračního souboru, který je prázdný nebo začíná znakem '#' je ignorován, ostatní řádky jsou platné a definují nastavení PABOVu. Poslední řádek souboru musí být prázdný. Každý platný řádek má formát

```
proměnná=hodnota.
```

Následuje vysvětlení významu proměnných:

- `timeout_server_startup` – Maximální čas potřebný pro spuštění virtuálního stroje v milisekundách. Hodnota by měla být nastavena minimálně na 15000.
- `timeout_sync` – Maximální čas potřebný pro synchronizaci s knihovnou PABOVu v procesu VirtualBoxu. Hodnota by měla být nastavena minimálně na 5000.
- `timeout_reply` – Maximální čas pro odpověď na požadavek zaslaný knihovně PABOVu v procesu VirtualBoxu. Hodnota by měla být nastavena minimálně na 60000.
- `time_reboot_max` – Maximálního hodnota čekání na restart virtuálního stroje. Viz pozdější popis parametrů `pabov.exe`.
- `time_terminate_max` – Maximálního délka běhu virtuálního stroje. Viz pozdější popis parametrů `pabov.exe`.
- `vbox_manage` – Jméno spustitelného souboru VirtualBox Command Line Management Interface. Mělo by být `VBoxManage.exe` pro VirtualBox verze 1.6.2.
- `vbox_machine` – Jméno virtuálního stroje. Zadejte jméno, které jste udali při vytváření virtuálního stroje ve VirtualBoxu. Pozor, jsou rozlišována velká a malá písmena.
- `vbox_machine_state` – Jméno souboru s databází stavu. Nedefinujte žádné jméno tak, že zakomentujete řádek s nastavením této proměnné.
- `vbox_username` – Jméno uživatele, který má zapisovací oprávnění ke složce, do které se budou kopírovat testované vzorky.

- `vbox_password` – Heslo uživatele, který má zapisovací oprávnění ke složce, do které se budou kopírovat testované vzorky.
- `vbox_share_name` – Jméno sdílení složky, do které se budou kopírovat testované vzorky, v rámci sdílení souborů Windows. Uveďte jméno, které jste vložili jako druhý parametr při spouštění `pabov_host.exe`.
- `vbox_share_drive` – Jméno volného disku, na který se bude mapovat sdílená složka – např. `J:`.
- `vbox_ip` – IP adresa virtuálního stroje.
- `vbox_upload_folder` – Cesta ke složce, kam se budou kopírovat testované vzorky.

7.5.5 Pořízení databáze stavu

Nyní vytvoříme databázi stavu pomocí `pabov_db.exe`. Nejprve je ale potřeba zkopírovat soubory VirtualBoxu sestaveného ze změněných zdrojových kódů do složky, odkud spouštíte VirtualBox. Spusťte `pabov_db.exe` s argumentem cesty k souboru, kam se uloží databáze stavu, např.:

```
pabov_db.exe c:\sandbox.pbv.
```

Dojde ke spuštění virtuálního stroje. Virtuálním systém restartuje pomocí Windows Exploreru (nabídka Start). Nyní počkejte, než systém nastartuje. To může trvat delší dobu. Počkejte, než je systém plně aktivní a v klidu. Pro jistotu počkejte dalších několik minut potom, co systém zobrazí plochu a bude se jevit jako plně připraven.

Na hostujícím systému vytvořte a spusťte skript (můžete jej pojmenovat např. `pabov_db_snap_poweroff.bat`), který obsahuje následující řádky, kde za `sandbox` dosaďte jméno virtuálního stroje a za `Ready` dosaďte libovolný název snímku:

```
VBoxManage.exe snapshot sandbox take Ready
VBoxManage.exe controlvm sandbox poweroff.
```

Nový snímek virtuálního stroje byl vytvořen a virtuální stroj byl vypnut. Virtuální stroj by měl nyní obsahovat dva snímky. První (starší) snímek můžete smazat. Na určené cestě se nyní nalézá soubor s databází stavu virtuálního stroje. Nyní můžete změnit proměnnou `vbox_machine_state` v konfiguračním souboru `pabov.conf`. Její hodnotu nastavte na jméno souboru databáze stavu.

7.5.6 Testování vzorku

V této fázi již je PABOV připraven k analýze. Vytvořte na hostujícím systému složku a umístěte do ní testovací vzorek spolu se všemi soubory, které může potřebovat. Spusťte `pabov.exe` a jako první argument předejte plnou cestu k složce se vzorkem. Druhý argument je čas v sekundách, za jak dlouho se má virtuální systém restartovat. V prezentované verzi není argument plně podporován a měl by být vždy nastaven na nulu. O restart se stará knihovna ve Windows Exploreru ve virtuálním systému. Pokud nechcete systém restartovat, zadejte 0. Třetí argument je délka běhu virtuálního systému od spuštění testovacího vzorku. Čtvrtý argument udává jméno hlavního spustitelného souboru testovacího vzorku v rámci složky udané v prvním argumentu. Pokud chcete vzorek spustit s argumenty, připojte je na konec příkazu. Příkaz může vypadat například takto:

```
pabov.exe C:\Test\Malware 0 360 video.exe.
```

Tento příkaz nastartuje virtuální stroj a zkopíruje do jeho sdílené složky obsah složky `C:\Test\Malware` a následně nechá ve virtuálním stroji spustit program `video.exe`, který se v této složce nalézá. Od spuštění `video.exe` bude virtuální stroj ukončen za 6 minut a není vyžadováno restartování.

Při ukončování stroje se vygeneruje výstup do adresáře `output`, který existuje nebo se vytvoří v adresáři VirtualBoxu. Aktuální verze PABOVu generuje výstup pouze jako HTML dokument. Jméno souboru výstupu pro konkrétní analýzu má tvar `YYYY-MM-DD_HH-MM-SS_<name>_<hash>.html`, kde první část označuje datum a čas spuštění analýzy, část `<name>` je jméno hlavního spustitelného souboru vzorku a `<hash>` je MD5 obsahu tohoto souboru.

8 Srovnání sledovacích systémů

V kapitole 5 jsme představili několik příbuzných, ale již hotových projektů, v kapitolách 6 a 7 jsme za podpory VirtualBoxu navrhli sledovací systém, který je poměrně odlišný od ostatních uvedených systémů. Tato kapitola se věnuje testování všech uvedených sledovacích systémů na reálných vzorcích malware. Ty byly nasbírány v průběhu psaní práce, převážně z nevyžádané pošty a ze serveru http://www.offensivecomputing.net/. Celkem analyzujeme a komentujeme pět vzorků malware. Komentáře k výstupům systémů pro neškodné vzorky jsou v kapitole 5, pro systém PABOV pak v kapitole 7.3.5.

Každý vzorek malware nejprve krátce popíšeme na základě informací dostupných na Internetu. K jejich vyhledání byla použita jména vzorků z výstupů antivirových skenů ze serveru http://www.virustotal.com/. Poté jej postoupíme testovaným systémům. Plné výstupy systémů k testovaným vzorkům jsou přiloženy na médiu, které je součástí této práce. V této kapitole výsledky pouze zrekapitulujeme, přičemž se budeme soustředit na dva základní aspekty. V prvé řadě nás bude zajímat, zda lidský analytik může z výstupu poznat, že se jedná o malware. Poté nás bude zajímat, zda by na základě výstupu bylo snadné automatizovaně poznat, že se jedná o vzorek malware. Účelem testování vzorků malware je poznat sílu sledovacích systémů – to znamená jejich schopnost odhalit škodlivé chování. V případě, že systém zahrnuje skenování antivirovým software, nepovažujeme výstup antiviru za součást výstupu sledovacího systému, protože takové části výstupu jsou často tvořeny analytikem antivirové společnosti. Je vhodné zmínit, že pojmenování vzorků není jednoznačné. Každá společnost, která se zabývá analýzou malware má vlastní pojmenování, často naprosto odlišné od konkurence.

Na závěr kapitoly zhodnotíme všechny testované systémy.

U systému PABOV byly vzorky spouštěny bez parametrů a doba běhu byla nastavena na pět minut. U ostatních systémů, pokud bylo možno nastavit, bylo testování vzorku zadáno také bez parametrů.

Při čtení této kapitoly je vhodné zároveň nahlížet do výstupů přiložených na médiu.

8.1 Vzorek malware #1 Virut.AV

Soubor vzorku na médiu: `/input/Virut.AV.e_x_e`

Adresáře s výstupy na médiu: `/output/Virut.AV/<SystemName>`

Virut.AV se kopíruje do systémového adresáře pod jménem cizího známého procesu. Pro tuto kopii vytváří záznam v registrovém klíči `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`, aby byla spuštěna vždy po startu systému. Může se pokoušet spojit s IRC serverem a zapojit se tak do tzv. botnetu. Může měnit nastavení registru Windows Firewallu v klíči `HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List`, aby mohl nerušeně přistupovat k síti. Virut také někdy mění obsah systémového souboru `HOSTS`, čímž se často malware snaží znemožnit antivirovým produktům aktualizaci jejich databází.

CWSandbox

Výstup obsahuje informace o čtyřech procesech. Prvním je proces vzorku. Druhým je systémový proces `svchost.exe`, který byl do výstupu zahrnut z důvodu nazvaného `DCOMService`. Co to ale znamená není zřejmé. Navíc k tomuto procesu výstup obsahuje pouze seznam v něm načtených modulů. Dalšími procesy zahrnuty do výstupu jsou vzorkem spuštěný skript `xxsmnl.bat` a `C:\WINDOWS\system32\iexplore.exe`.

V informacích o aktivitách procesu vzorku je uvedeno, že vzorek vytvořil skript `xxsmnl.bat`, jehož obsah ale není k dispozici, dále zkopíroval sám sebe do souboru `C:\WINDOWS\system32\iexplore.exe` a oba tyto soubory smazal. Jelikož ale nejsou události seřazené chronologicky, není jisté, který z možných scénářů skutečně nastal. Vzorek vytvořil dva nové procesy `xxsmnl.bat` a `iexplore.exe`. U vzorku je také zaznamenána neobvyklá aktivita práce v rámci jeho vlastního adresového prostoru. Informace však nejsou dostatečně detailní, aby bylo možné s určitostí zjistit, o co se jedná. Je pravděpodobné, že se jedná o hákování funkcí v uživatelském režimu, to je typické pro malware určený ke krádežím přihlašovacích údajů k webovým službám.

Z informací o procesu skriptu `xxsmnl.bat` vyplývá, že pravděpodobně jediný jeho úkol bylo odstranění původního souboru vzorku z disku.

Proces `C:\WINDOWS\system32\iexplore.exe` vytvořil registrovou hodnotu `Microsoft Internet Explorer` v klíči `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` tak, aby docházelo k jeho automatickému spuštění při každém startu systému. I u tohoto procesu je zaznamenána neobvyklá práce v rámci vlastní paměti, stejně jako u původního procesu vzorku. Dále se tento proces připojil na IRC server v Internetu. Výstup obsahuje záznam odeslaných a přijatých dat.

Z dostupných informací je zřejmé, že se jedná o malware. Instalace vlastního souboru do systémového adresáře, zajištění automatického spuštění tohoto souboru a vymazání původního souboru vzorku jsou dostatečnými důkazy. Dalším typickým projevem malware je skryté připojení k IRC serveru. Všechny tyto aktivity jsou ve výstupu jasně zdokumentovány a na jejich základě by nebyl problém automaticky označit vzorek jako malware. Přítomnost malware potvrzuje i použití jména prohlížeče Internet Exploreru a jména jeho hlavního modulu `iexplore.exe`.

Norman SandBox

Výstup je velmi krátký. Velmi snadno se z něj vyčte, že vzorek zkopíruje svůj modul do souboru `C:\WINDOWS\SYSTEM32\winamp.exe`, který také zaregistruje jako `Winamp Agent` v registrovém klíči `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`, čímž se snaží zajistit si automatické spouštění při každém startu systému, což je navíc explicitně ve výstupu řečeno. Tento program je také vzorkem ihned spuštěn. Vzorek vytváří skript `kslil.bat`, jeho účel však z výstupu není jasný, neboť konkrétní data o obsahu vytvořených souborů nejsou k dispozici. Výstup dále uvádí, že kód vzorku obsahuje pasáže zaměřené proti ručním nebo automatickým analýzám.

Z výstupu je patrné, že se jedná o malware. Instalace sama sebe je pro malware typická, navíc použití jména známého software Winamp to jen potvrzuje. Tyto závěry by se daly vyvodit snadno automaticky nicméně jistota odhadu by byla nižší vzhledem k tomu, že výstup obsahuje pouze dvě indície.

Anubis

Výstup je velmi rozsáhlý a začíná souhrnem, kde jsou zhodnoceny rizika. Vysoké riziko u tohoto vzorku představuje fakt, že se jedná o IRC bota; že skenuje počítače v síti; a že mění a ničí soubory, které nejsou dočasné. Dále je zaznamenáno, že se vzorek nainstaluje do systému tak, aby byl znovu spuštěn, a že spouští další procesy v systému.

Dále se uvádí, že vzorek infikoval systémový proces `winlogon.exe` a dále spustil ještě jednu instanci tohoto programu a systémový proces `cmd.exe`. `winlogon.exe` dále spustil dvě instance `ntvdm.exe`. Výstup obsahuje informace o všech těchto spuštěných procesech.

Následuje dlouhá a nezajímavá pasáž se seznamem informací o síťové komunikaci. Dále se uvádí seznam přístupů vzorku k systémovému registru, seznam je dlouhý a v záplavě dat je těžké najít něco užitečného. Zdá se, že uvedené přístupy k registru mají na svědomí systémové knihovny, které vzorek používá.

V další části výstupu je informace o vytvořeném skriptu `uscxwdg.bat`. Jeho obsah uveden není. Navíc je zde informace, že vzorek vytvořil a modifikoval systémový soubor `winlogon.exe` v systémovém adresáři.

Následuje rozbor aktivit procesů `winlogon.exe`, `cmd.exe` a `ntvdm.exe`, kde se zdá být důležitá jak existující infikovaná, tak nově spuštěná instance `winlogon.exe`. U první instance je zaznamenána změna nastavení Windows Firewallu a také změna souboru `HOSTS`. U nové instance pak je záznam o vytvoření dvou nových souborů v systémovém adresáři `iica.exe` a `izns.exe`. Tyto programy byly dále spuštěny společně se dvěma instancemi `ntvdm.exe`. Dále je uvedeno, že tento proces `winlogon.exe` kontaktoval webový server `nadsamcabrab12.com`, z něhož se pokusil stáhnout soubory `e.exe` a `rs3.exe`, což se ovšem nepodařilo. Výpis pokračuje informacemi o síťovém provozu, kde je zaznamenán pokus o skenování portu 135 na 386 IP adresách a také připojení na IRC server. Následuje dlouhý seznam informací o každém kontaktu, který však nic nového nepřináší.

Z dalšího se dozvíme, že procesy `ntvdm.exe` byly spuštěny pro vytvořený program `izns.exe`. Systémový proces `ntvdm.exe` je virtuální stroj určený pro podporu 16-bitových aplikací na 32-bitovém systému Windows. Je tedy pravděpodobné, že poté, co se vzorek pokusil stáhnout obsah dalších programů z webového serveru, zkusil stažené soubory spustit, ale jelikož se stahování nezdařilo, soubory neobsahovaly data určená ke spuštění, následkem čehož systém usoudil, že by se mohlo jednat o 16-bitové aplikace a spustil virtuální stroj k jejich emulaci.

Výstup je doplněn záznamem síťového provozu.

Výstup obsahuje dostatek informací dosvědčující, že vzorek je malware. Systém navíc sám v souhrnu uvedl domněnky, že se o malware jedná. Automatické rozpoznání by tedy vzhledem k dostatku důkazů bylo snadné.

Joebox

Výstup je prakticky prázdný. Je možné, že vzorek rozpoznal sledovací systém a ukončil se nebo spadl. Nejpravděpodobnější scénář je však pád systému, jelikož výstup na řádku `Errors` uvádí chybu `No Ping` a Joebox implementuje jednoduchý protokol na ověřování stavu analytického stroje. Pozdější opakované testování nepřineslo nový výsledek.

ThreatExpert

Výstup obsahuje shrnutí, které je však pravděpodobně výtahem z databáze antiviru. Tomu nasvědčuje fakt, že detaily dostupné dále ve výstupu analýzy chování neobsahují dostatek informací k odvození závěrů v shrnutí.

Systém rozpoznal tvorbu skriptů `ejkm.bat` a `mgjtrl.bat` a souboru `lssas.exe` v systémovém adresáři, obsah skriptů není udán, obsah souboru `lssas.exe` je odvoditelný pouze z výstupu antivirového skenování. Výstup ještě obsahuje informaci o instalaci `lssas.exe` do registrového klíče `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`, čímž vzorek zajišťuje automatické spuštění tohoto programu.

Vzhledem k možnostem rozpoznání malware je výstup prakticky totožný s výstupem systému Norman SandBox.

BitBlaze

Výstup obsahuje pouze informaci o vytvoření skriptu `aqfjlohr.bat`, jehož obsah je následující:

```
@echo off
:deleteagain
del /A:H /F Virut.AV.e_x_e.exe
del /F Virut.AV.e_x_e.exe
if exist Virut.AV.e_x_e.exe goto deleteagain
del aqfjlohr.bat
```

Výstup je podpořen záznamem síťového provozu, který však neobsahuje žádná důležitá data.

Na základě těchto informací není možno rozpoznat, že se jedná o malware, i když je zřejmé, že se vzorek pokouší sám sebe smazat, což je neobvyklé.

PABOV

Z výstupu se dozvíme, že sledovaný vzorek vytvořil dva nové procesy – `cmd.exe` a `logon.exe`. Výstup obsahuje informace o všech těchto třech procesech.

Jako první jsou k dispozici informace o procesu vzorku. V části věnované procesům a vláknům dostáváme informaci o vytvoření zmíněných procesů a jejich hlavních vláken. Zajímavější je další část o aktivitách se soubory v systému, která zmiňuje vytvoření souboru `logon.exe` v systémovém adresáři a dále vytvoření skriptu `rgrhoc.bat`. U obou souborů jsou k dispozici jejich data, proto lze vyčíst, že jediná funkce skriptu je smazání sebe sama a souboru vzorku. Následuje seznam aktivit v registru, kde jsou však pravděpodobně pouze nerelevantní informace o událostech, které mají na svědomí standardní knihovny použité vzorkem. V poslední části u procesu vzorku je záznam aktivit souvisejících s virtuální pamětí. Při bližším zkoumání lze vypozorovat, že se jedná o aktivity spojené s tvorbou procesu, kde procesu `cmd.exe` je předložen argument `/c` a cesta ke skriptu `rgrhoc.bat`. Je tedy zřejmé, že tento proces byl spuštěn za účelem vykonání skriptu a tedy smazání souboru vzorku.

Informace k procesu `cmd.exe` pouze potvrzují předchozí nález. Tento proces smazal soubor vzorku a poté i skript.

U procesu `logon.exe` je zaznamenáno několik zajímavých událostí. V části věnované registrům je udáno nastavení registrové hodnoty `Windows Logon Application` v klíči `\REGISTRY\MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`.
Zapsaná hodnota je plná cesta k souboru `logon.exe` v systémovém adresáři. Tím si vzorek

zajišťuje opětovné spouštění po startu systému. V další části je dán záznam síťové aktivity, kde lze vyčíst opakované spojení se serverem adresy 83.68.16.6 na port 5190 a serverem adresy 67.43.226.242 na port 8080. Při bližším prozkoumání dat je zřejmé, že se jedná o IRC komunikaci, která je běžná u některých typů malware. Z daných dat je tak možno odhalit části botnetu.

Na konci výstupu jsou události chronologicky uspořádány.

Instalace nového programu do systémového adresáře a zápis do registru zajišťující automatické spouštění tohoto programu při startu systému stejně jako smazání vlastního souboru vzorku jsou typické znaky malware, které by šlo automaticky snadno rozpoznat. Připojení k IRC serveru a následná komunikace taktéž nasvědčují přítomnosti malware. Pro manuální analýzu je dalším vodítkem název instalovaného programu `logon.exe`, který má evokovat příslušnost k systému, stejně jako název registrové hodnoty `Windows Logon Application`.

Shrnutí

Systémy Anubis, CWSandbox a PABOV podaly detailní informace o chování vzorku. Výstupy systémů Anubis a CWSandbox však byly zahlceny řadou nepodstatných informací. Systém Anubis sám rozpoznal řadu jevů, které označil za velmi podezřelé, a uvedl je v shrnutí na začátku výstupu.

Výstupy systémů Norman SandBox a ThreatExpert jsou dostatečné pro rozpoznání malware, ale řada aspektů chování vzorku nebyla zaznamenána. Není však vyloučeno, že se proces vzorku chová při různých svých instancích odlišně na základě nějakého náhodného faktoru.

Nejhorší výsledky vydaly systémy BitBlaze a Joebox. Výstup BitBlaze byl takřka prázdný a neobsahoval dostatek informací k identifikaci malware. Joebox dokonce skončil s chybou a nevydal žádné informace.

8.2 Vzorek malware #2 OnLineGames.kw

Soubor vzorku na médiu: `/input/OnLineGames.kw.e_x_e`

Adresáře s výstupy na médiu: `/output/NSAnti.TZK/<SystemName>`

Tento malware byl vytvořen pro krádeže informací uživatelů hrající online hry. Je o něm známo, že do adresáře Windows vytváří soubor `cmdbcs.exe`, který následně zapíše do registrové hodnoty v klíči `HKLM\SoftWare\Microsoft\Windows\CurrentVersion\Run`, což mu zaručuje automatické spuštění při každém startu systému. Také vytváří knihovnu `cmdbcs.dll` v systémovém adresáři.

CWSandbox

Stejně jako v případě testování demonstračních vzorků zde systém vydal výstup poněkud znepřehledněný informacemi o procesech, které nemají žádnou relevanci k vzorku. Navíc k těmto procesům jsou opět uvedeny pouze nic neříkající chyby a části výstupu jsou nesmyslné. Relevantní informace výstupu se týkají procesu vzorku, procesu `cmdbcs.exe`, který vzorek spustil a procesu Windows Exploreru, který byl vzorkem infikován.

Vzorek vytvořil svoji kopii v adresáři Windows pod jménem `cmdbcs.exe`. Spuštění této

kopie ale v seznamu aktivit vzorku chybí. Dále se pokoušel vyhledat grafická okna tříd `AVP.ALERTDIALOG` a `avp.product_Notxfication`, což může být snaha o detekci produktů z dílny Kaspersky Lab.

V informacích o nově spuštěném procesu `cmdbcs.exe` je zaznamenáno následující. Proces vytvořil soubor `cmdbcs.dll` v systémovém adresáři a smazal soubor původního vzorku. Dále instaloval sám sebe do registrového klíče `HKEY_LOCAL_MACHINE\SoftWare\Microsoft\Windows\CurrentVersion\RUN`, aby si zajistil pravidelné spouštění se startem systému. Vyhledal a ukončil proces původního vzorku (zde je třeba připomenout chybějící chronologii zaznamenaných událostí). Dále otevřel proces Windows Exploreru, alokoval v něm virtuální paměť, zapsal do ní data a vytvořil v něm nové vlákno.

Data k aktivitám infikovaného Windows Exploreru jsou nesmyslná.

Výstup je z části nesmyslný, to poukazuje na chyby v sledovacím systému. Přesto však z výstupu je zřejmé, že testovaný vzorek je malware. To je patrné především z instalace vlastního souboru do systémového adresáře, zajištění automatického spouštění a infekce procesu Windows Exploreru. Na základě dostupných informací je toto hodnocení snadné učinit i automaticky.

Norman SandBox

Výstup udává vytvoření souborů `cmdbcs.exe` a `cmdbcs.dll` a instalaci do registrového klíče. Dále je ve zaznamenáno vyhledání procesu Windows Exploreru a infekce tohoto procesu vlastním kódem. Program `cmdbcs.exe` je vzorkem spuštěn. Zajímavou informací ve výstupu je vyhledání oken AVP, což je pravděpodobně snaha o detekci antivirového produktu z dílny Kaspersky Lab.

Instalace vlastních souborů do systémových adresářů a následná registrace jsou typické pro malware. Podezření potvrzuje infekce procesu Windows Exploreru. Automatické označení vzorku za malware by tedy bylo na základě výstupu systému jednoduché.

Anubis

Výstup obsahuje shrnutí, které za nejrizikovější akci vzorku označuje změnu a zničení souborů, které nejsou dočasné. Při představování systému Anubis jsme však viděli, že tato informace se může objevit i u neškodného vzorku. Shrnutí dále uvádí, že vzorek vytváří další procesy a že se instaluje do systému tak, aby si zajistil automatické spouštění.

Následují informace o aktivitách v rámci procesů systému. Vzorek vytvořil proces `cmdbcs.exe`, který poté infikoval proces Windows Exploreru. Z tohoto důvodu jsou sledovány všechny tři procesy – proces vzorku, `cmdbcs.exe` a proces Windows Exploreru. Dále výstup uvádí dlouhý a nezajímavý seznam operací vzorku s registry následovaný shrnutím aktivit se soubory, kde se uvádí vznik souboru `cmdbcs.exe` v adresáři Windows.

Další částí jsou informace o aktivitách procesu `cmdbcs.exe`. Ten odstraňuje proces a soubor původního vzorku, vytvoří soubor `cmdbcs.dll` v systémovém adresáři a infikuje Windows Explorer. Výstup pokračuje výpisem aktivit Windows Exploreru, kde je jediná užitečná informace, že Windows Explorer načetl knihovnu `cmdbcs.dll`. Zbytek informací není relevantní.

Z výstupu je zřejmé napadení Windows Exploreru a instalace sama sebe do systému. Takové aktivity jsou pro malware typické, proto lze usoudit, že vzorek je malware. Systém Anubis má všechny tyto informace k dispozici a na jejich základě lze vzorek za malware automaticky

označit.

Joebox

Výstupní shrnutí udává, že během analýzy vznikly dva nové procesy a jeden existující byl infikován. Následuje statistika počtu volání jednotlivých systémových služeb, která příliš mnoho nevypovídá. Dále se dočteme, že testovaný vzorek spustil proces `cmdbcs.exe`, který následně spustil proces `explorer.exe`. Spuštěním se zde ale pravděpodobně rozumí chování, které zapříčinilo sledování dalšího procesu.

Výstup dále udává seznam operací se soubory, sekcemi, registry a dalšími objekty systému. Asi jediný zajímavý je údaj o vytvoření souboru `cmdbcs.exe` v adresáři Windows. Následuje část výstupu, kde jsou operace seřazeny chronologicky. Z této části je nejlépe poznat chování vzorku. Je zde vidět vytvoření souboru `cmdbcs.exe` i jeho následné spuštění.

Výstup pokračuje informacemi o aktivitách spuštěného procesu `cmdbcs.exe`. Tyto informace mají stejnou strukturu jako u původního vzorku. Lze vyčíst vytvoření knihovny `cmdbcs.dll` v systémovém adresáři a smazání souboru původního vzorku. Dále je uvedeno zajištění spouštění `cmdbcs.exe` vytvořením záznamu v registrovém klíči. Následuje otevření původního procesu vzorku a jeho ukončení a také otevření procesu Windows Exploreru. Zajímavou informací je infekce Windows Exploreru. Následuje opět chronologický přehled těchto událostí.

Výstup uzavírají informace o aktivitách infikovaného procesu Windows Exploreru. Důležitá je informace o načtení knihovny `cmdbcs.dll`. Druhou zajímavou informací je opakované načítání souboru `C:\Program Files\RealVNC\VNC4\winvnc4.exe`, to je však pravděpodobně součást sledovacího systému a je tudíž nejspíše irelevantní.

Výstup velmi podrobně dokumentuje události v systému způsobené testovaným vzorkem, ze kterých se dá jednoznačně usoudit na přítomnost malware – instalace vlastních souborů do složek systému, vytvoření záznamu v registru pro automatické spuštění a především útok na proces Windows Exploreru. Automatické označení vzorku by v tomto případě nebyl problém.

ThreatExpert

Výstup opět obsahuje shrnutí, které je pravděpodobně dílem lidského pracovníka, neboť ve výstupu chybí důkazy uvedených tvrzení.

Systém rozpoznal vytvoření souborů `cmdbcs.exe` a `cmdbcs.dll` v adresáři Windows resp. v systémovém adresáři. Dále je uvedeno vytvoření procesu `cmdbcs.exe` a také načtení knihovny `cmdbcs.dll` do procesu Windows Exploreru. Nakonec je k dispozici informace o registraci `cmdbcs.exe` do klíče pro automatické spouštění.

Na základě těchto informací je zřejmé, že se jedná o malware. Nálezy uvedené ve výstupu jsou dostatečné a snadno rozeznatelné i pro automatické označení.

BitBlaze

Výstup je prakticky prázdný. Systém však nevydal žádnou chybu. I výstup statické analýzy je velmi krátký a nic zajímavého o vzorku nevypovídá.

PABOV

Výstup uvádí ovlivnění tří procesů. Proces vzorku spustil nový proces `cmdbcs.exe`, který dále ovlivnil proces Windows Exploreru.

Zajímavé události u procesu vzorku se omezují na vytvoření souboru `cmdbcs.exe` v adresáři Windows. Máme k dispozici úplná zapsaná data, proto je snadné odhalit, že soubor `cmdbcs.exe` je přesnou kopií souboru původního vzorku. Další události jako například aktivity v oblasti virtuální paměti mají přímou souvislost se spuštěním procesu `cmdbcs.exe`. Analýzou těchto událostí zjistíme, že nový proces byl spuštěn s argumenty, mezi nimiž je cesta k původnímu souboru vzorku a systémový identifikátor procesu vzorku.

U procesu `cmdbcs.exe` je zaznamenáno ukončení procesu vzorku, vytvoření souboru `cmdbcs.dll` v systémovém adresáři, vytvoření registrové hodnoty `cmdbcs` v klíči `\REGISTRY\MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`, alokace a změny v paměti Windows Exploreru a také vytvoření nového vlákna ve Windows Exploreru. Opět máme k dispozici všechna data při vytváření souboru `cmdbcs.dll`, takže můžeme odhalit zajímavý trik, který je velmi pravděpodobně namířen proti antivirové detekci tohoto souboru. Ten je totiž konstruován ve více fázích tak, že zprvu chybí standardní hlavička spustitelného souboru a teprve nakonec je doplněna. Pokud tak antivirový systém nekontroluje celý soubor při každém zápisu, což by mohlo být časově náročné, ale kontroluje například pouze zapsaná data, nemusí při vytváření souboru `cmdbcs.dll` detekovat infekci i přesto, že soubor v databázi má. Nakolik je tento trik účinný proti dnešním antivirům však můžeme jen spekulovat. Z dat zapsaných do procesu Windows Exploreru to vypadá, že záměrem procesu `cmdbcs.exe` je načíst knihovnu `cmdbcs.dll` do procesu Windows Exploreru za asistence nově vytvořeného vlákna.

To se potvrzuje ze zaznamenaných událostí k procesu Windows Exploreru. Mezi jeho moduly nalezneme na posledním místě právě `cmdbcs.dll`. Další zaznamenané události příliš mnoho neříkají, případně nejsou relevantní.

Výstup opět končí seznamem chronologicky uspořádaných událostí, ve kterých je celé dění přehledně zrekapitulováno.

Z výše uvedeného jasně plyne škodlivost vzorku. Jsou zde opět typické znaky jako smazání původního souboru vzorku, skrytá instalace do adresáře Windows a instalace knihovny do systémového adresáře, zajištění automatického spouštění zápisem do registru a především infekce procesu Windows Exploreru novým vláknem a kódem, který načetl knihovnu `cmdbcs.dll` do procesu Windows Exploreru. Všechny tyto projevy chování jsou automaticky snadno detekovatelné a tudíž by nebyl problém s automatickým označením tohoto vzorku za malware.

Shrnutí

Škodlivost projevů daného vzorku lze nejlépe vyčíst z výstupů systémů Joebox a PABOV. Zarušení nerelevantními daty je v případě PABOVu malé a v případě Joeboxu jsou tato data většinou umístěna ve vlastních sekcích, tudíž příliš neruší.

Dostatek informací k odhalení malware byl poskytnut i systémy CWSandbox, Norman SandBox, Anubis a ThreatExpert. U systémů CWSandbox a Anubis však výstup obsahoval velmi mnoho nerelevantních dat a v případě CWSandboxu dokonce chybná data. Naopak výstupy systémů Norman SandBox a ThreatExpert byly poměrně skoupé. Tyto systémy poskytly pouze nejdůležitější údaje bez dalších detailů.

Výstup systému BitBlaze byl prakticky prázdný a tudíž nepoužitelný.

8.3 Vzorek malware #3 Swizzor.dvu

Soubor vzorku na médiu: `/input/Swizzor.dvu.e_x_e`

Adresáře s výstupy na médiu: `/output/Swizzor.dvu/<SystemName>`

Tento vzorek malware může přidávat do webového prohlížeče v infikovaném systému různé lišty, odkazy, ikony a mění chování prohlížeče tak, aby se čas od času zobrazovala reklama. Po spuštění vytvoří novou instanci webového prohlížeče, který infikuje vlastním kódem, pomocí něhož si z webového serveru stáhne data jako např. reklamu nebo knihovny implementující ActiveX objekty, které poté instaluje do prohlížeče.

CWSandbox

K chování procesu vzorku výstup zachycuje vytvoření instance Internet Exploreru a jeho následné nakažení vlastním kódem. Vzorek vytvořil v procesu Internet Exploreru vlastní vlákno. Dále je zaznamenána podobná neobvyklá aktivita nad vlastní pamětí jako v případě vzorku Virut.AV.

Následuje seznam aktivit vytvořeného procesu Internet Exploreru. V tomto seznamu jsou obsaženy jak aktivity nového vlákna, tak standardní aktivity prohlížeče. Je zaznamenán přístup na server `ayb.lop.com` a stažení jedné webové stránky. Dále je opět zaznamenána neobvyklá aktivita nad vlastní pamětí. Zajímavé je, že v seznamu modulů tohoto procesu se objevuje i soubor původního vzorku. To není vůbec obvyklé. Další informace k dispozici nejsou.

Vzhledem k možnostem klasifikace vzorku výstup obsahuje vytvoření nového procesu prohlížeče, jeho nakažení a následné stažení stránky z webového serveru. Není tedy zcela zřejmé, že se jedná o malware. Spuštění prohlížeče je časté i u legitimních aplikací. Na druhou stranu načtení souboru rodičovského procesu do nového procesu prohlížeče je podezřelé. Nicméně by bylo možné vzorek označit jako podezřelý a postoupit jej manuální analýze.

Norman SandBox

Výstup neobsahuje žádné informace, které by vedli k odhalení škodlivosti vzorku. Dle výstupu Normanu vzorek vykazuje jen běžnou práci s registry.

Anubis

Výpis opět v počátečním shrnutí uvádí čtyři jevy, které jsme viděli i u ostatních vzorků. Jejich relevance je diskutabilní. Zvláště vysoké riziko u změny a zničení souborů, které nejsou dočasné, zřejmě prakticky znamená vytvoření libovolného souboru na disku mimo adresář dočasných souborů.

Dále je uvedeno, že vzorek spustil proces Internet Exploreru. V informacích popisující chování procesu vzorku není příliš zajímavých údajů kromě části o vytvoření procesu Internet Exploreru a nového vlákna v něm – jedno vlákno je vždy vytvořeno při tvorbě nového procesu, zde však vidíme vytvoření ještě jednoho dalšího. Výstup pokračuje informacemi o spuštěné instanci Internet Exploreru. Zde je řada informací, které vystihují normální

chování tohoto programu. Lze však nalézt jeden neobvyklý jev, kterým je načtení souboru vzorku do procesu jako knihovny. Kód vzorku tak v kontextu procesu Internet Exploreru pravděpodobně ovlivňuje jeho další chování, čímž se dá vysvětlit přístup prohlížeče k webovému serveru `ayb.lop.com` a stažení jedné z jeho stránek. Další informace k dispozici nejsou.

Z výstupu není zcela zřejmé, že se jedná o malware. Spuštění procesu prohlížeče je poměrně běžná akce i u legitimních programů. Neobvyklé je načtení souboru vzorku do procesu prohlížeče. Společně s tím, že vzorek vytvořil nové vlákno v tomto procesu, jsou to jediné indície, o které se lze opřít při rozhodování o přítomnosti malware v tomto případě. Pro automatické rozpoznání malware by však samotná tato informace zřejmě nestačila, případně by jistota odhadu byla nízká.

Joebox

Ve výstup nejprve zjistíme, že došlo ke spuštění instance Internet Exploreru vzorkem. Následují seznamy práce se soubory a registry, které však neobsahují žádné podezřelé záznamy. Ani přístup k dalším objektům není nijak zajímavý, snad kromě vytvoření vláken v procesu Internet Exploreru, kterému předchází zápis dat do paměti tohoto procesu.

Následují informace o chování procesu Internet Exploreru. Tyto informace obsahují záznam o načtení souboru vzorku do procesu. Výstup je ukončen záznamy síťového provozu, kde je zdokumentován přístup na webový server `ayb.lob.com` a stažení jedné jeho stránky.

Výstup systému Joebox obsahuje informace podobné těm, které vydal systém Anubis. Při klasifikaci může hrát roli chronologický seznam akcí se záznamem o zapsání dat do paměti Internet Exploreru před vytvořením druhého vlákna. Zápis do paměti nového procesu jeho rodičem je běžný, ale právě chronologické uspořádání umožňuje odlišit zápis, který již není součástí tvorby procesu. Ještě lepší by byla naše pozice, pokud by záznam o vytvoření druhého vlákna obsahoval více detailů – například adresu a parametry pro obslužnou funkci vlákna. Tyto údaje však nemáme.

ThreatExpert

Pokud pomineme data související s antivirovým skenem, pak výstup obsahuje pouze informaci o načtení souboru vzorku do procesu Internet Exploreru. Samotná tato informace nestačí pro vyhodnocení rizik vzorku.

BitBlaze

Výstup BitBlaze obsahuje pouze záznam o několika přístupech vzorku k registru a čtení souboru hlavního modulu Internet Exploreru. Z těchto informací rozhodně nelze správně vzorek ohodnotit.

PABOV

Výstup se zabývá procesem vzorku a procesem `iexplore.exe`, který vzorek spustil.

V událostech k procesu vzorku se dočteme, že došlo ke spuštění procesu `iexplore.exe`, kromě jeho hlavního vlákna bylo vytvořeno ještě jedno další. Proces `iexplore.exe` je instancí Internet Exploreru. Proces vzorku dále vytvořil registrový klíč `\REGISTRY\USER\S-1-5-21-1004336348-1060284298-2122583443-1003\Software\bags burn film`, ve kterém pak opakovaně měnil hodnotu `Mp3 Iso`. V kategorii

zachycující události práce s virtuální pamětí se kromě událostí přímo souvisejících se spuštěním nového procesu objevují i další položky. Konkrétně poslední položka velmi pravděpodobně obsahuje kód a data pro nově vzniklé vlákno. Data obsahují plnou cestu k původnímu souboru vzorku a také odkazy na standardní knihovní funkce **LoadLibrary** a **Sleep**.

U procesu `iexplore.exe` je zajímavá přítomnost souboru vzorku mezi moduly tohoto procesu. Lze tak vyvodit závěr, že nové vlákno vytvořené procesem vzorku vykonalo kód zapsaný do procesu Internet Exploreru, který do něj načetl modul vzorku. Další události uvádí vytvoření dočasných souborů Internet Exploreru, což souvisí s komunikací s Internetovým serverem, která je zaznamenána v poslední sekci výstupu u tohoto procesu. Té předchází seznam událostí souvisejících s registry, kde opět dochází k nastavování registrové hodnoty `Mp3 Iso`. Jedná se však pravděpodobně o zašifrovaná data. Ostatní záznamy jsou nejspíše důsledkem činnosti standardních knihoven. V oblasti síťové aktivity je zaznamenáno připojení na server adresy 66.220.17.154 a port 80/TCP. Máme k dispozici jeden HTTP požadavek směřovaný na doménu `ayb.lop.com` a odpověď serveru.

Výstup napovídá, že by se mohlo jednat o malware, ale důkazů není mnoho. Samotné spuštění procesu Internet Exploreru je běžné i u neškodného software, ale vytvoření vlákna a načtení souboru vzorku jako modulu do procesu Internet Exploreru jsou snadno detekovatelné jevy příznačné pro malware. Pro úplnou automatickou detekci malware v tomto případě chybí dostatek důkazů, ale vzorek by na základě zjištěného mohl být označen minimálně jako podezřelý a podstoupen dalšímu zkoumání.

Shrnutí

U tohoto vzorku žádný výstup neobsahoval takové množství usvědčujících důkazů, aby bylo možné zcela nepochybně označit vzorek jako malware. Přesto systémy CWSandbox, Anubis, Joebox a PABOV identifikovali některé podezřelé jevy, na základě kterých by se vzorek dal označit jako podezřelý.

Výstupy systémů Norman SandBox, ThreatExpert a BitBlaze byly v tomto případě nedostatečné.

8.4 Vzorek malware #4 Wsnpoem.AG

Soubor vzorku na médiu: `/input/Wsnpoem.AG.e_x_e`

Adresáře s výstupy na médiu: `/output/Wsnpoem.AG/<SystemName>`

Činnost tohoto vzorku je o něco rozsáhlejší než u předchozích vzorků. Jeho detekce by tak měla být snazší. Malware zkopíruje vlastní modul do souboru `ntos.exe` v systémovém adresáři. Dále se vytvoří soubory `audio.dll` a `video.dll` v adresáři `wsnpoem` v systémovém adresáři. Malware si zajišťuje automatické spouštění pomocí změny registrové hodnoty `UserInit` v klíči `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` nebo vytvořením hodnoty v klíči `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`. Dále infikuje procesy běžící v systému vlastním kódem a v uživatelském režimu hákuje funkce pro posílání HTTP požadavků. To mu umožňuje krást přihlašovací informace uživatelů k různým webovým službám jako je např. eBay. Ukradené informace ukládá do výše zmíněného souboru `audio.dll` a následně se je snaží odeslat webovému serveru. Vzorek také může vytvořit backdoor na náhodném síťovém portu

systému.

CWSandbox

Výstup je rozsáhlý a obsahuje informace o 23 procesech. Informace o některých procesech jsou však prázdné. Důvod rozsáhlého výstupu je postupná infekce všech procesů v systému, na základě které sledovací systém začal sledovat všechny tyto procesy.

Z výstupu plyne, že vzorek vytvořil svoji kopii pod jménem `ntos.exe` v systémovém adresáři. Dále vytvořil soubor `audio.dll` v adresáři `wsnpoem` v systémovém adresáři. Také je zaznamenána instalace v registru zajišťující automatické spouštění `ntos.exe`. Následuje otevření systémového procesu `winlogon.exe`, zapsání dat do paměti procesu a vytvoření nového vlákna v tomto procesu. Zajímavý je záznam přístupu vzorku ke službě Protected Storage (služba určená pro uchovávání citlivých informací jako například přihlašovacích jmen a hesel) a záznamům, které zde uchovává Internet Explorer.

`winlogon.exe` pak vytvořil soubor `video.dll` v adresáři `wsnpoem`, otevřel proces `svchost.exe` a infikoval jej. Proces `svchost.exe` pak zaslal dva požadavky webovému serveru na adrese 72.232.239.117. Z tohoto procesu pak došlo k infekci dalších procesů běžících v systému.

V ostatních nakažených procesech jsou zaznamenány neobvyklé aktivity v rámci jejich paměťových prostorů. Pravděpodobně jde o instalaci háků určených k získání citlivých dat. Toto však pouze na základě výstupu nelze dokázat.

Výstup odhaluje rozsáhlou infekci šířící se ze vzorku přes systémové procesy `winlogon.exe` a `svchost.exe` do ostatních procesech v systému. Už samotná tato aktivita jasně dokládá přítomnost malware a klasifikace vzorku tedy není obtížná. Ostatní nasbírané údaje pouze potvrzují škodlivost vzorku.

Norman SandBox

Výstup neobsahuje žádné informace o chování vzorku. Je možné, že vzorek detekoval Norman SandBox a ukončil se. To je však jenom domněnka.

Anubis

Výstup na počátku shrnuje důležité události. Zde je zajímavá informace o instalaci programu do systémového adresáře a zajištění automatického spouštění vzorku.

Následují informace o chování vzorku, kde v části registrových aktivit je zaznamenána změna hodnoty `userinit` v klíči `HKLM\software\microsoft\windows nt\currentversion\winlogon` zajišťující spouštění `ntos.exe` ze systémového adresáře. Dále v seznamu vytvořených souborů je vidět právě vytvoření souboru `ntos.exe` v systémovém adresáři.

Z tohoto výstupu je zřejmé, že se jedná o malware. Nasvědčuje tomu neobvyklá metoda instalace zajišťující automatické spouštění a také zkopírování souboru vzorku do systémového adresáře pod jménem, které má vyvolat dojem legitimní aplikace. To je však jasné lidskému analytikovi, ale automatický systém by toto pozorování učinil obtížně. Pro automatickou detekci se tak snadno využije pouze informace o zajištění spouštění netradičním způsobem a instalace do systémového adresáře. V tomto případě by rozpoznání použití netradiční mohlo stačit k verdiktu, že vzorek vykazuje chování malware.

Joebox

Výstup je extrémně rozsáhlý. Příčina je zřejmá po shlédnutí sekce `Startup` na začátku výstupu. Joebox rozpoznal, že nějakým způsobem došlo k ovlivnění systémového procesu `winlogon.exe` a přes něj dále procesu `svchost.exe`. V důsledku toho mělo dojít k ovlivnění dalších patnácti procesů a výstup tedy obsahuje informace o všech těchto procesech.

Pokud se zaměříme na aktivity procesu vzorku, bylo zaznamenáno vytvoření souboru `ntos.exe` v systémovém adresáři a vytvoření adresáře `wsnpoem`. Dále je zajímavá změna hodnoty `userinit` v registrovém klíči `\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` tak, že zajišťuje spouštění právě `ntos.exe`. Poté je uvedeno otevření procesu `winlogon.exe` a vytvoření vlákna v tomto procesu. Chronologický výpis vybraných událostí obsahuje informaci o zápisu dat do paměti `winlogon.exe` před vytvořením nového vlákna. Jedná se o čtyři zápisy.

Výstup pokračuje informacemi o aktivitách napadeného procesu `winlogon.exe`. Zde je zaznamenáno vytvoření adresáře `wsnpoem` v systémovém adresáři a souborů `audio.dll` a `video.dll`. Dále je uvedena informace o otevření procesu `svchost.exe` a vytvoření nového vlákna v tomto procesu. V chronologickém výpisu pak jsou opět vidět čtyři zápisy dat do paměti procesu `svchost.exe` předcházející vytvoření nového vlákna.

Následuje část věnovaná chování procesu `svchost.exe`. Zde je však velké množství dat a je obtížné odlišit informace vztahující se k napadení od informací zachycující běžné chování tohoto procesu. Zajímavou je však informace o nastavení registrové hodnoty `EnableFirewall` na 0 v klíči `\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile`. Tato hodnota způsobí vypnutí vestavěného firewallu Windows a je nepravděpodobné, že by systémový proces sám od sebe toto provedl. Z chronologického výstupu je vidět postupné otevírání dalších procesů v běžících systému a vytváření nových vláken, kterým opět předchází zápisy do paměti cílových procesů.

Další informace ve výstupu zachycují chování ostatních napadených procesů, ale informace odrážejí jejich běžné chování. Z našeho hlediska tedy nejsou příliš zajímavé.

Z výstupu systému Joebox je evidentní, že se jedná o malware. Instalace do systému a především vysledovaný řetězec napadení procesů běžících v systému je jasným důkazem. Dalším důkazem je vypnutí systémového firewallu. Na základě dostupných informací lze označit vzorek jako malware bez problémů i automaticky.

ThreatExpert

Výstup obsahuje informace o vytvoření souboru `ntos.exe` v systémovém adresáři a souborech `audio.dll` a `video.dll` v adresáři `wsnpoem` v systémovém adresáři. Zajímavé jsou malé velikosti obou knihoven. Dále je zachycena změna registrové hodnoty `userinit` v klíči `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon`, u které však chybí nově nastavená hodnota. Poslední zajímavou informací ve výstupu je stažení stránky `http://77.221.133.188/.c/o/cfg.bin`.

Vytvoření souboru `ntos.exe` v systémovém adresáři je podezřelé. Stejně tak vytvoření malých souborů s příponou DLL, které nemohou být díky své velikosti skutečnými dynamickými knihovnami. Výstup ale neobsahuje důležitou informaci o novém obsahu registrové hodnoty `userinit`. Pouze na základě výstupu tak není možné s určitostí tvrdit,

že se jedná o malware, ale dle zjištěných skutečností by bylo možné vzorek označit za podezřelý a podstoupit jej ruční analýze.

BitBlaze

Výstup tohoto systému neobsahuje žádné informace kromě upozornění, že vzorek možná spadl ihned po spuštění nebo jeho struktura není podporována.

PABOV

Výstup zachycuje dění v 17 procesech.

Proces vzorku vytvořil soubor `ntos.exe` v systémovém adresáři. Jedná se o kopii souboru vzorku. Zajímavé však je, že nový soubor byl vzápětí přepsán jinými daty. Vzorek smazal několik souborů cookies. Zde se však pravděpodobně jedná o důsledek činnosti nějaké systémové knihovny. Dále bylo zaznamenáno zapisování dat do souboru `audio.dll` v adresáři `wsnpoem` v systémovém adresáři. Ze zapsaných dat zde není zřejmý jejich smysl. Proces vzorku také komunikoval přes roury pojmenované `__SYSTEM__7F4523E5__` a `__SYSTEM__64AD0625__`. Vzhledem k tomu, že nebylo zaznamenáno vytvoření adresáře `wsnpoem` nebo zmiňovaných rour, dá se předpokládat, že je vytvořil jiný proces. U registrových aktivit je zajímavá pouze první událost. Jedná se o nastavení hodnoty `Userinit` v registrovém klíči `\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Winlogon`. Ostatní aktivity jsou důsledkem běžné činnosti systémových knihoven. Hodnota `Userinit` byla nastavena tak, aby došlo k automatickému spouštění `ntos.exe` ze systémového adresáře. Proces vzorku také alokoval paměť v systémovém procesu `winlogon.exe`, kam dále zapsal data a následně v tomto procesu vytvořil nové vlákno. Z tohoto důvodu byl pak proces `winlogon.exe` sledován.

V kontextu infikovaného procesu `winlogon.exe` došlo k vytvoření adresáře `wsnpoem` v systémovém adresáři a v něm dále k vytvoření souborů `audio.dll` a `video.dll`. Vznikla také roura pojmenovaná `__SYSTEM__64AD0625__`, která pravděpodobně sloužila ke komunikaci s procesem vzorku. Byly zaznamenány opakované zápisy do této roury. Proces dále infikoval další systémový proces `svchost.exe`, ve kterém alokoval paměť, zapsal do ní data a vytvořil v něm nové vlákno. Zapsaná data jsou téměř totožná jako v případě infekce `winlogon.exe`.

Nakažený proces `svchost.exe` vytvořil rouru `__SYSTEM__7F4523E5__`, která opět zřejmě slouží ke komunikaci s původním procesem vzorku. Z dalších zaznamenaných událostí je zřejmá infekce ostatních procesů v systému, která probíhá velmi podobně jako u infekce procesů `winlogon.exe` a `svchost.exe`. Zaznamenaná registrová aktivita nevybočuje nad rámec běžných funkcí tohoto procesu. U síťové aktivity je zaznamenáno započetí naslouchání na třech portech – 6409, 6739 a 5352. Není však jisté, zda se jedná o normální projev tohoto procesu nebo zda jde o důsledek infekce.

Záznamy u dalších procesů již relevantní nejsou. Jedná se výhradně o události související s běžnou činností nakažených procesů.

Z výstupu je zřejmá činnost malware. Rozsáhlá infekce všech procesů v systému je postačujícím důkazem, který je dále podpořen zajištěním automatického spuštění kopie souboru vzorku. Navíc je použitá metoda pro zajištění automatického spouštění velmi neobvyklá. U legitimního software se vyskytuje pouze v několika velmi speciálních případech. Automatická detekce by tak v tomto případě byla velmi jednoduchá a spolehlivá.

Shrnutí

Systémy CWSandbox, Joebox a PABOV poskytly ve svých výstupech dostatek informací k jasné identifikaci škodlivého chování příznačného pro malware.

ThreatExpert a Anubis vydaly poměrně málo informací, které by pro spolehlivou automatickou detekci stačit nemusely. Nicméně by se na základě výstupů těchto systémů dal vzorek označit za podezřelý.

Systémy BitBlaze a Norman SandBox nezaznamenaly žádné informace použitelné pro klasifikaci vzorku.

8.5 Vzorek malware #5 Rustock.B

Soubor vzorku na médiu: `/input/Rustock.B.e_x_e`

Adresáře s výstupy na médiu: `/output/Rustock.B/<SystemName>`

Rustock.B je předchůdcem legendárního malware Rustock.C, jehož funkční vzorek v podobě spustitelného programu se bohužel nepodařilo získat. Rustock.B ale plně postačí pro naše účely. Je zástupcem malware využívající techniky rootkitů, což znamená, že se snaží svoje aktivity skrývat. Tento malware instaluje ovladač do tzv. alternativního proudu dat (alternate data stream, ADS) `C:\Windows\System32:lzx32.sys`. Pro načtení ovladače potřebuje také vytvořit záznam v registrovém klíči `HKLM\SYSTEM\CurrentControlSet\Services`. Vytváří zde nový klíč s názvem `pe386`. Načtený ovladač pak hákuje MSR, pomocí kterého mění chod některých funkcí tak, aby skryl svojí přítomnost v systému. Dále mění funkce síťových ovladačů, aby se vyhnul případným firewallům instalovaným v systému. Může z Internetu stahovat další programy, měnit obsah stránek v prohlížečích, vytvořit backdoor na libovolném síťovém portu a rozesílat nevyžádanou poštu.

CWSandbox

Výstup je opět poněkud chybový, avšak důležité informace obsahuje. Z výstupu lze vyvodit, že proces vzorku použil komponentu Service Control Manager v procesu `services.exe` k načtení ovladače do systému a infikoval proces Windows Exploreru vlastním kódem a vytvořením nového vlákna. Informace o načtení ovladače obsahují jak jméno souboru ovladače `C:\WINDOWS\System32:lzx32.sys`, tak i jméno klíče v registru `\Registry\Machine\System\CurrentControlSet\Services\pe386`. Zajímavé proto je, že výstup neuvádí vytvoření ADS `C:\WINDOWS\System32:lzx32.sys` vzorkem. V seznamu aktivit infikovaného Windows Exploreru zaujme pouze přístup k webovému serveru na adrese 208.66.194.158.

Načtení ovladače může být příznakem malware, ale i mnoho legitimních aplikací skrytě instaluje ovladač do jádra. V tomto případě však je tělo ovladače uloženo v alternativním proudu dat, což je možná snaha o skrytí tohoto ovladače, se kterou se v případě legitimních instalací nesetkáme. Připočteme-li infekci Windows Exploreru, je zřejmé, že testovaný vzorek je malware. Na základě daného výstupu lze toto rozpoznání učinit snadno i automaticky.

Norman SandBox

Výstup je krátký a obsahuje jen pár informací. Podivný je záznam o vytvoření souboru `C:\WINDOWS\TEMP\\x12;lzx32.sys`, přičemž další výpis uvádí instalaci ovladače

C:\WINDOWS\SYSTEM32:lzx32.sys. Kromě informací související s instalací ovladače výstup nezachycuje žádné zajímavé události.

Zde se z výstupu nedá jednoznačně určit, zda se jedná o malware. Instalace ovladače v ADS je určitě podezřelá, nicméně pouze na základě jedné události je obtížné činit závěry. Vzorek by bylo možné označit jako podezřelý a postoupit jej dalšímu zkoumání.

Anubis

Výstup začíná shrnutím, ve kterém zaujme především velmi riziková aktivita posílání emailů pravděpodobně za účelem dalšího šíření vzorku. Důležité je i zaznamenání instalace do systému tak, aby docházelo k opětovnému spouštění vzorku.

Dále je ve výstupu uvedeno zdůvodnění sledování procesů services.exe a procesu Windows Exploreru. services.exe je sledován pouze z důvodu instalace služby (ovladače) do systému prostřednictvím komponenty Service Control Manager. Proces Windows Exploreru je sledován z důvodu infekce vzorkem.

Informace o aktivitách procesu vzorku obsahují záznam o vytvoření souboru ovladače C:\WINDOWS\system32:lzx32.sys a o zavedení tohoto ovladače pod názvem pe386. Následuje informace o infekci procesu Windows Exploreru – zápisu dat a vytvoření nového vlákna.

Výstup pokračuje seznamem informací o procesu services.exe. Řada informací z výstupu se vztahuje k běžnému chování tohoto procesu. Zaznamenané síťové aktivity však rozhodně běžné nejsou. Jedná se o přístupy k rozdílným legitimním serverům. Je možné, že se tak vzorek snaží získat informace o kvalitě připojení nakaženého uzlu. K dispozici jsou záznamy síťové komunikace podpořené samostatným souborem se záznamem od síťového sledovače. Byl zaznamenán i jeden pokus o přístup na SMTP server, ale žádná data nebyla přenesena. Je však zajímavé, že Anubis sledoval services.exe. Pouze na základě informací uvedených ve výstupu k tomu neměl důvod, neboť pokud vzorek pouze využil komponentu systému, která se stará o služby a ovladače a kterou implementuje tento proces, k načtení ovladače, proces samotný nebyl nijak kompromitován.

Informace o aktivitách nakaženého procesu Windows Exploreru obsahují jedinou zajímavou informaci o smazání souboru původního vzorku. Ostatní zachycené události jsou běžné pro tento proces.

Z výstupu je zřejmé, že se jedná o malware. Je v něm zachycena instalace ovladače z ADS, což je velmi neobvyklé, navíc vzorek infikoval proces Windows Exploreru, což je typické pro malware. Stahování různých webových stránek z kontextu procesu services.exe je rovněž velmi podezřelé. Sledovací systém měl k dispozici dostatek informací k automatické klasifikaci vzorku jako malware.

Systém sám navíc upozorňuje na snahu vzorku o šíření prostřednictvím emailu, avšak jediným důkazem tohoto se zdá být pokus o navázání spojení se SMTP serverem.

Joebox

Joebox rozpoznal vytvoření ADS C:\WINDOWS\system32:lzx32.sys vzorkem a vytvoření vlákna v procesu Windows Exploreru, kterému předcházely zápisy do paměti tohoto procesu. Jiné zajímavé události u procesu vzorku uvedeny nejsou.

Výstup pokračuje záznamem událostí ve vztahu k načtenému ovladači lzx32.sys. Informací

je zde však málo a neuvádí se žádné zajímavé skutečnosti.

Další částí je popis aktivit nakaženého Windows Exploreru, ale všechny záznamy odpovídají standardnímu chování tohoto programu.

Výstup uzavírá záznam síťové komunikace, která je velmi podobná jako v případě systému Anubis. I zde je k dispozici externí soubor se záznamem celé síťové komunikace. Z výstupu ale není poznat, jaký proces nebo ovladač provedl síťové aktivity.

Z dostupných informací můžeme za příznak malware pokládat pouze infekci procesu Windows Exploreru. Vytvoření ADS může být podezřelé, ale výstup neobsahuje informace o zavedení ovladače, i když obsahuje část, která se věnuje jeho aktivitám. Propojení se vzorkem ale není dostatečně zdokumentováno. Z pohledu automatického systému by zřejmě nešlo s určitostí rozhodnout, zda je daný vzorek malware, podezření na základě infekce procesu by však vzniknout mohlo.

ThreatExpert

Výstup uvádí vytvoření ADS `C:\WINDOWS\system32:lzx32.sys` a změny v registru potřebné pro načtení ovladače. Dále byly zaznamenány změny v paměti procesu `services.exe`, není však k dispozici dostatek informací, aby bylo možné poznat nějaký záměr. Ani ostatní informace nejsou příliš vypovídající. Výstup také uvádí síťové aktivity vzorku, opět jen velmi povrchně. Zajímavá je poslední část výstupu, kde systém uvádí, že odhalil schopnost vzorku sbírat emailové adresy pro účely rozesílání nevyžádané pošty. Zdali tato část nějak souvisí s analýzou chování vzorku nebo zda se jedná spíše o výstup antivirového skenu není z výstupu jasné.

Ve výstupu jsou tak relevantní pouze uvedené změny v registru a vytvoření ADS. Chybí záznam o načtení ovladače. Nelze tedy s určitostí tvrdit, že se jedná o činnost malware. Na základě dostupných informací by tak systém mohl doporučit vzorek pro ruční analýzu.

BitBlaze

Tento systém opět nebyl schopen k vzorku cokoliv vydat. Výstup je prázdný a obsahuje pouze informaci, že pravděpodobně vzorek krátce po spuštění spadl nebo že jeho formát není podporován.

PABOV

Výstup obsahuje informace o procesu vzorku a ovlivněném procesu Windows Exploreru.

Proces vzorku infikoval paměť procesu Windows Exploreru a vytvořil v něm nové vlákno. V zapsaných datech je zřejmý odkaz na původní soubor vzorku. Dále byl vytvořen alternativní proud dat `system32:lzx32.sys` v adresáři Windows. Jedná se o ovladač, jeho obsah je k dispozici.

Proces Windows Exploreru smazal soubor původního vzorku a provedl několik zápisů do registru, které ale nevybočují z běžného chování tohoto procesu. Ani další zaznamenané operace nejsou z našeho hlediska zajímavé.

Získaných informací není mnoho. Výstup zaznamenává velmi podezřelé události jako je infekce procesu Windows Exploreru a následné smazání souboru vzorku. Dále použití alternativního proudu dat je velmi neobvyklé u legitimních procesů. Vzorek lze označit za podezřelý, ale pro spolehlivou automatickou detekci by bylo potřeba ještě více důkazů.

Shrnutí

Systémy CWSandbox a Anubis rozpoznaly klíčové události, které spolehlivě usvědčují daný vzorek jako malware.

Systémy Norman SandBox, Joebox, ThreatExpert a PABOV obsahují některé indicie, na základě kterých by bylo možné označit vzorek za podezřelý. Nasbíraných důkazů však není dostatek pro spolehlivou klasifikaci vzorku.

Systém BitBlaze vydal chybový výstup.

V případě našeho systému PABOV je důvod nedostatku informací ve výstupu u tohoto vzorku jasný. Kód vzorku přenesený do Windows Exploreru používá standardní služby procesu `services.exe` k instalaci ovladače. Tyto služby jsou provozovány pomocí RPC, které aktuální verze PABOVu implementuje pouze v hákovací vrstvě, chybí tedy implementace na vrstvě událostí a ve finální analýze pro výstup. Stejně tak PABOV v této verzi nepokrývá načítání ovladačů do systému a paměťově mapované soubory, což by i při nepokrytí RPC stačilo pro odhalení důležitých aspektů chování tohoto vzorku.

8.6 Celkové hodnocení

Systém CWSandbox zcela uspěl u třech vzorků, kde by na základě jeho výstupu bylo snadné a spolehlivé automaticky klasifikovat vzorek jako malware. U jednoho vzorku bylo informací dostatek pro klasifikaci, ale oproti ostatním systémům chyběla řada detailů. Zbývající vzorek pak bylo možné označit jen jako podezřelý. Výstupy tohoto systému byly občas chybové a často obsahovaly mnoho nerelevantních dat.

Sledovací systém Norman SandBox poskytl pouze jedenkrát dostatek informací pro spolehlivou klasifikaci. U dvou vzorků by na základě výstupu tohoto systému mohly být vzorky, z důvodu malého množství informací, označeny jen jako podezřelé. U zbývajících dvou vzorků pak výstupy neobsahovaly dostatek relevantních informací. Výstupy tohoto systému jsou obecně krátké a obsahují málo informací, často jsou uvedeny jen ty nejpodstatnější.

Výstup systému Anubis byl zcela dostačující ve dvou případech. V jednom případě pak byl jeho výstup dostačující, ale některé detaily chyběly. U zbylých dvou případů by nešlo spolehlivě klasifikovat vzorek lépe než jako podezřelý. Výstupy Anubise se vyznačují velkým množstvím informací, z nichž řada bývá nerelevantní. U sledovacích systémů je však lepší větší úroveň detailů než naopak, což Anubis splňuje. Systém Anubis jako jediný obsahuje analýzu chování na vysoké úrovni. Výsledkem této analýzy je shrnutí na začátku každého výstupu, které se s rozdílnou úspěšností snaží o klasifikaci rizik nejdůležitějších projevů chování testovaných vzorků.

Joebox zcela uspěl u dvou vzorků, u kterých by automatická klasifikace jako malware byla snadná a spolehlivá. U dalších dvou by na základě jeho výstupu bylo možné označit vzorky za podezřelé. U jednoho vzorku nevydal vůbec žádné informace a skončil s chybou. Některé sekce ve výstupu tohoto systému obsahují informace, které o ničem nevypovídají. Kromě PABOVu je tento systém jediný, který dává na výstup i chronologicky setříděné události. To dává uživateli informace do souvislostí, které jsou často důležité pro rozpoznání skutečného chování vzorků.

Systém ThreatExpert se spíše opírá o informace doplněné z antivirových skenů. Pouze na základě sledování chování vzorku většinou nedokáže odhalit důležité aspekty,

které by mohly vést k spolehlivé klasifikaci. Pouze u jednoho vzorku by mohlo dojít k jednoznačnému označení vzorku za malware, ale i zde výstup neobsahoval řadu důležitých informací. Tři vzorky by mohly být na základě výstupu systému ThreatExpert označeny za podezřelé. U jednoho vzorku pak systém neodhalil nic podezřelého.

Systém BitBlaze zcela selhal ve všech případech. Často vydal pouze chybový výstup. V ostatních případech obsahovaly jeho výstupy pouze kusé informace, na základě kterých by nebylo možné vzorky ohodnotit.

Systém PABOV zcela uspěl ve třech případech a u zbývajících dvou by jeho nálezy postačovaly ke klasifikaci vzorku za podezřelé. Silnou stránkou PABOVu je existence seznamu chronologicky setříděných událostí a také dostupnost dat řady operací jako je například práce se soubory nebo s pamětí. Taková data u ostatních systémů zcela chybí nebo jsou dostupná pouze u síťového provozu. Mnohem lepších výsledků by mohlo být dosaženo, pokud by implementace byla doplněna o další, zatím ne zcela pokryté, události v systému.

9 Závěr

9.1 Shrnutí práce

V první části práce byla shrnuta řada metod, které jsou nejčastěji v praxi použité nebo teoreticky uvažované pro implementaci sledování chování procesů na operačních systémech Windows. Odděleny byly metody, které nevyužívají virtualizaci od těch, které jsou na virtualizaci založeny. V této části práce nejde příliš do hloubky, jelikož by při důkladnějším popsání všech aspektů daných metod práce narostla do nežádoucích rozměrů.

V další části jsme představili existující systémy, vysvětlili principy jejich fungování a na příkladech demonstrovali výstupy těchto systémů. Na to navázal popis konstrukce sledovacího systému PABOV založeného na virtualizačním software VirtualBox. Netriviální části návrhu i implementace byly detailně popsány.

Vrchol práce je ve srovnání všech uvedených sledovacích systémů na vzorcích reálně se vyskytujícího malware.

9.2 Splnění cílů

Cílem práce bylo shrnout možné metody implementace sledovacího systému a navrhnout sledovací systém za použití virtualizace. Teoretická část byla splněna popsáním široké škály metod, z nichž některé jsou využity popsanými existujícími systémy.

Výsledkem praktické části bylo vytvoření základu sledovacího systému PABOV. Ve srovnání s ostatními dostupnými systémy dosahuje PABOV velmi dobrých výsledků a kvalitou a obsahem výstupů se řadí mezi nejlepší dostupné systémy. Rozhodně se však nedá tvrdit, že jde o hotový systém – viz následující část.

Přestože to nebyl primární vytyčený cíl práce, vytvoření jak teoretické tak především praktické části práce mělo velmi pozitivní přínos na znalosti autora práce v oblasti fungování operačního systému Windows. Tvorba sledovacího systému PABOV umožnila nahlédnout v detailu na řadu systémových funkcí, jejichž dokumentace není veřejně přístupná a jejichž chování by jinak bylo jen obtížně zjistitelné. Skutečné pochopení některých těchto dějů bylo zásadní pro správnou implementaci jádra systému PABOV.

9.3 Další možný rozvoj

Prezentovanou verzi systému PABOV uvádíme jako základ sledovacího systému, nikoliv jako zcela dokončený projekt. Je zde obrovský potenciál pro zdokonalení systému. Největším nedostatkem aktuální verze je chybějící pokrytí mnoha projevů chování. Není zde však zásadní problém, který by znemožňoval taková rozšíření. Jde spíše o velikou časovou náročnosti implementace kódu pro zpracování desítek dalších událostí.

Sledovací systém spouští testované vzorky ve virtualizovaném systému. V aktuální verzi byl použit čistý systém. To jistě není ideální prostředí pro zkoumání projevů malware, neboť řada projevů škodlivého chování malware je aktivována pouze v případě zjištění přítomnosti nějakého konkrétního software nebo činnosti uživatele. V tomto ohledu by bylo vhodné do sledovaného systému přidat nějakou formu automatického uživatele, který by například prováděl přihlášení do online systémů bank nebo jiných systémů atraktivních pro tvůrce malware. Stejně tak by v systému mohl být nainstalován software klientů instantní komunikace jako je Skype, MSN, ICQ apod. tak, abychom mohli vypozorovat krádeže přihlašovacích údajů a jiné zajímavé jevy.

Oproti existujícím dostupným systémům PABOV není přístupný přes webové rozhraní. Kromě alokace serveru a implementace samotného rozhraní by bylo k zpřístupnění systému PABOV potřeba malých změn v kódu tak, aby bylo možné nastavit limity na použitou paměť a diskový prostor tak, aby záměrně škodlivé programy generující velké množství dat nezahltily server nebo jeho síťová spojení. Tato nutnost plyne z dostupnosti dat jednotlivých sledovaných událostí v oblasti paměti, souborů a sítě, kterou PABOV jako jediný z představených systémů implementuje. Tato část je poměrně snadná a autor věří, že se v blízké budoucnosti podaří projekt poskytnout k veřejnému použití.

Možným rozšířením systému je i zajištění podpory pro jiné verze operačního systému. Windows Vista a Windows 7 včetně jejich 64-bitových verzí jistě budou v blízké budoucnosti hrát významnou roli i na poli malware. Takové rozšíření by ale vzhledem k návrhu systému PABOV bylo velmi časově nákladné.

Za dobu psaní této práce se značně posunul i projekt VirtualBoxu. Použitá verze pro systém PABOV je sice dostačující, ale vzhledem k novému vývoji již poměrně zastaralá. Pro další rozvoj se tak nabízí i možnost podpory nejnovější verze VirtualBoxu.

Koncoví uživatelé by pak jistě uvítali implementaci detekce a filtrování událostí, které se váží k chování standardních systémových komponent nebo knihoven a jejichž výskyt ve výstupech pouze snižuje jejich čitelnost.

Po vzoru systému Anubis by také bylo vhodné implementovat detekci důležitých událostí a nějakým způsobem je na výstupu zdůraznit nebo seskupit do shrnutí.

Příloha A – Struktura přiloženého média

Zde je uvedena struktura a krátký popis souborů a adresářů na přiloženém médiu.

- **/**`DT.pdf` – text práce, tento soubor
- **`input/`** – obsahuje všechny použité testované vzorky jak neškodných programů CmdLine a GetIP, tak i vzorky malware, jejichž přípony jsou z bezpečnostních důvodů přejmenovány na nespustitelné
- **`output/`** – obsahuje výstupy analýz sledovacích systémů řazených do adresářů dle názvů jednotlivých vzorků
- **`src/`** – obsahuje zdrojové kódy systému PABOV a zdrojové soubory VirtualBoxu, které byly upraveny; struktura zdrojových kódů je blíže popsána v příloze B.

Příloha B – Struktura zdrojových kódů

Příloha obsahuje popis struktury a obsahu zdrojových souborů PABOVu – tj. souborů a adresářů pod `src/pabov/src/`. Neobsahuje popis změn ve zdrojových kódech VirtualBoxu. Ty byly popsány v kapitole 6.2. Práce využívá implementaci MD5 Message-Digest Algorithm od RSA Data Security, Inc.

- **`/`**
  - `Makefile`, `*.def` – soubory nutné pro sestavení PABOVu
  - `pabov.conf` – vzorový konfigurační soubor
  - `pabov.c` – implementace uživatelské aplikace PABOVu
  - `pabov_db.c` – implementace uživatelské aplikace PABOVu používané při generování databáze stavu
  - `pabov_host.c` – implementace instalátoru knihovny ve virtualizovaném stroji
- **`common/`** – obsahuje implementace funkcí, které jsou různě využívány v ostatních komponentách systému
  - `common.c`, `common.h` – implementace různých jednotlivých, navzájem nesouvisejících, funkcí
  - `dl_list.c`, `dl_list.h` – implementace obousměrného spojového seznamu
  - `hash_table.c`, `hash_table.h` – implementace hašovací tabulky
  - `sync.c`, `sync.h` – implementace nástrojů synchronizace
  - `compat.h`, `missing.h` – základ případné budoucí podpory pro více cílových operačních systémů, doplnění struktur a konstant chybějících v distribuci MinGW
  - `debug.h` – nastavení ladění při vývoji systému
  - `error.h` – správa chybových kódů
  - `types.h` – základní typy používané v PABOVu a konverze typů
- **`conf/`** – obsahuje implementaci práce s konfiguračním souborem
- **`helpers/`** – obsahuje implementaci automatického generátoru adres s využitím Hacker Disassembler Engine 32 C od Vyacheslava Patkova
- **`hook/`** – obsahuje implementaci různých komponent souvisejících s háky PABOVu
  - `events.c`, `events.h` – implementace veškeré funkcionality objektů událostí včetně závěrečného zpracování seznamu objektů událostí
  - `handler.c`, `handler.h` – implementace obslužných funkcí háků PABOVu
  - `hook.c`, `hook.h` – implementace vrstvy hákování PABOVu, která se nachází hned nad kódem VirtualBoxu

    - `monitor.c`, `monitor.h` – implementace instalace háků na základě databáze háků
    - `database.*` – databáze háků podporovaných verzí operačních systému
- **`io/`** – obsahuje implementaci vstupně-výstupních částí
    - `tcp.c`, `tcp.h` – implementace komponent TCP serveru a klienta
    - `output_html.c`, `output_html.h` – implementace výstupu v HTML
- **`log/`** – obsahuje implementaci knihovny pro ladící výpisy
- **`manager/`** – obsahuje implementaci komunikace mezi uživatelskou aplikací a knihovnou PABOVu v procesu VirtualBoxu a implementaci komponenty této knihovny
    - `manager.c`, `manager.h` – implementace komponenty knihovny PABOVu v procesu VirtualBoxu
    - `manager_cli.c`, `manager_cli.h` – implementace komunikačního rozhraní s knihovnou PABOVu v procesu VirtualBoxu
- **`md5/`** – obsahuje implementaci MD5 Message-Digest Algorithm od RSA Data Security, Inc.
- **`pabov/`** – obsahuje implementaci hlavní knihovny PABOVu v procesu VirtualBoxu
- **`tests/`** – obsahuje implementaci testů
    - `tests.h` – společný hlavičkový soubor testů
    - **`component/`** – obsahuje implementaci testů jednotlivých komponent systému
        - `hash_table*.*` - obsahuje testy implementace hašovací tabulky
    - **`functionality/`** – obsahuje implementace testů analytických schopností systému
- **`vbox/`** – obsahuje implementaci komponent souvisejících s virtuálním strojem
    - `inside_dll.c`, `inside_dll.h` – implementace knihovny instalované do Windows Exploreru virtuálního systému
    - `state.c`, `state.h` – implementace stavu virtuálního systému
    - `vm.c`, `vm.h` – implementace získávání informací o virtuálním systému
    - `vbox.h` – opis potřebných struktur z hlavičkových souborů VirtualBoxu

Literatura a reference

[1] Baker Art, Lozano Jerry: *The Windows 2000 Device Driver Book: A Guide for Programmers (2nd Edition)*

[2] Hoglund Greg, Butler Jamie: *Rootkits: Subverting the Windows Kernel*

[3] Nebbett Gary: *Windows NT/2000 Native API Reference*

[4] Oney Walter: *Programming the Microsoft Windows Driver Model, Second Edition*

[5] Russinovich Mark E., Solomon David A.: *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000*

[6] Microsoft Developer Network, http://msdn.microsoft.com/

[7] Intel 64 and IA-32 Architectures Software Developer's Manuals, http://www.intel.com/products/processor/manuals/

[8] Wikipedia, the free encyklopedia, http://www.wikipedia.org/

[9] rootkit.com, http://rootkit.com/

[10] Under the Hood: New Vectored Exception Handling in Windows XP, http://msdn.microsoft.com/en-us/magazine/cc301714.aspx

[11] System Call Optimization with the SYSENTER Instruction – CodeGuru.com, http://www.codeguru.com/cpp/w-p/system/devicedriverdevelopment/article.php/c8223

[12] File System Filter Drivers, http://www.microsoft.com/whdc/driver/filterdrv/default.mspx

[13] bochs: The Open Source IA-32 Emulation Project, http://bochs.sourceforge.net/

[14] VMware, http://www.vmware.com/

[15] Windows Virtual PC, http://www.microsoft.com/windows/virtual-pc/

[16] VirtualBox, http://www.virtualbox.org/

[17] Probert Dave: *Windows Kernel Internals II – Virtual Machine Architecture*, http://www.i.u-tokyo.ac.jp/edu/training/ss/msprojects/data/06-VirtualMachineArchitecture.ppt

[18] Lo Jack, VMware Inc.: *VMware and CPU Virtualization Technology*, http://download3.vmware.com/vmworld/2005/pac346.pdf

[19] Ferrie Peter, Symantec Advanced Threat Research: *Attacks on Virtual Machine Emulators*, http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

[20] Omella Alfredo Andr´es, Grupo S21sec Gesti´on S.A.: *Methods for Virtual Machine Detection*, http://www.s21sec.com/descargas/vmware-eng.pdf

[21] VMware Inc.: *Paravirtualization API*, http://www.vmware.com/vmi

[22] Vx32: Lightweight, User-level Sandboxing on x86 http://pdos.csail.mit.edu/papers/vx32:usenix08/

[23] Chromium Developer Documentation, http://dev.chromium.org/

[24] CWSandbox, http://www.cwsandbox.org/

[25] Norman SandBox, http://www.norman.com/security_center/security_tools/submit_file/en-us

[26] Anubis, http://anubis.iseclab.org/

[27] Joebox, http://www.joebox.org/

[28] ThreatExpert, http://www.threatexpert.com/

[29] BitBlaze, https://aerie.cs.berkeley.edu/

[30] MinGW – Minimalist GNU for Windows, http://mingw.org/