VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza škodlivých multiplatformních PowerShell skriptů

Analysis of Crossplatform Malicious PowerShell Scripts

2020 Bc. Jan Polok

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Polok**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1801T064 Informační a komunikační bezpečnost

Téma: Analýza škodlivých multiplatformních PowerShell skriptů
Analysis of Crossplatform Malicious PowerShell Scripts

Jazyk vypracování: čeština

Zásady pro vypracování:

Windows PowerShell (dříve známý jako Microsoft Shell, MSH či pod kódovým označením Monad) je rozšiřitelný textový (řádkový) shell se skriptovacím jazykem od společnosti Microsoft. PowerShell dokáže přistupovat nejenom k souborovému systému, ale také například k registrům systému, úložišti certifikátů a dalším. V posledních letech je PowerShell aktivně využíván k tzv. „Living off the Land" útokům. Tedy útokům využívajícím možností uživatelského prostředí (např. u OS Windows lze využít nativní knihovny pro šifrování k vytvoření šifrovacího vyděračského programu). Úkolem studenta bude provést rešerši aktuálního stavu na poli bezpečnosti vzhledem k použití PowerShellu, a to zejména aktuálně dostupné verze, která je multiplatformní. Student popíše možné způsoby využití Powershellu na různých operačních systémech. V praktické části student připraví alespoň dva netriviální příklady využití Powershellu pro tvorbu škodlivého kódu.

Hlavní body zadání:
1. Rešerše aktuálního stavu na poli IT bezpečnosti vzhledem k použití PowerShell scriptu.
2. Popis PowerShell scriptu, jeho verzí a možných způsobů tvorby škodlivých kódů na různých operačních systémech.
3. Implementace minimálně dvou netriviálních PowerShell scriptů, a to pro minimálně dva operační systémy.
4. Otestování útoku a diskuze výsledků.

Seznam doporučené odborné literatury:

[1] David Kennedy, Jim O'Gorman, Devon Kearns, Mati Aharoni, Metasploit: The Penetration Tester's Guide, No Starch Press 2011, ISBN: 978-1593272883
[2] Brenton J.W. Blawat, Mastering PowerShell, Packt Publishing - ebooks Account 2015, ISBN: 978-1782173557

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Plucar, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2020 .....

Rád bych poděkoval všem, kteří mi s prací pomohli, zejména Ing. Janu Plucarovi, Ph.D. za vedení a cenné rady při tvorbě práce.

Abstrakt

Diplomová práce se zabývá bezpečností nástroje PowerShell a škodlivými skripty v něm vytvořenými. Cílem práce byla analýza možnosti využít PowerShell Core k vytvoření multiplatformního škodlivého kódu. Součástí práce je testování vybraných existujících skriptů z penetračních frameworků v poslední verzi PowerShellu Core na systémech Windows, Linux a macOS. V rámci diplomové práce bylo vytvořeno a popsáno několik škodlivých skriptů, které jsou funkční na operačních systémech Windows a Linux, vybrané z nich i na macOS. V závěru práce jsou popsány konkrétní kroky, pomocí kterých je možné snížit riziko útoku.

Klíčová slova: PowerShell, skript, multiplatformní, škodlivý, bezsouborový, útok

Abstract

The diploma thesis deals with the security of the tool PowerShell and malicious scripts created in it. The goal of the work was an analysis of the possibility of exploiting PowerShell Core by creating multiplatform malicious code. A part of the thesis covers the testing of selected existing scripts from penetration frameworks on the latest version of PowerShell Core on Windows, Linux and macOS systems. Several malicious scripts, which run on Windows and Linux operating systems, and even some on macOS, were created and described in the scope of the study. At the end of the work, specific steps are described to reduce the risk of attack.

Keywords: PowerShell, script, multiplatform, malicious, fileless, attack

Obsah

Seznam použitých zkratek a symbolů

AES	–	Advanced Encryption Standard
AMSI	–	Antimalware Scan Interface
API	–	Application Programming Interface
AV	–	Antivirus
C&C	–	Command and Control
CA	–	Certifikační Autorita
CMS	–	Cryptographic Message Syntax
COM	–	Component Object Model
DLL	–	Dynamic-Link Library
EID	–	Event Identifier
FTP	–	File Transfer Protocol
GPO	–	Group Policy
HTML	–	Hypertext Markup Language
HTTP(S)	–	Hypertext Transfer Protocol (Secure)
IP	–	Internet Protocol
JSON	–	JavaScript Object Notation
MMC	–	Microsoft Management Console
NAT	–	Network Address Translation
NTFS	–	New Technology File System
OS	–	Operační systém
PE	–	Portable Executable
PKI	–	Public Key Infrastructure
POSH	–	PowerShell
REST	–	Representational state transfer
RPC	–	Remote Procedure Call
RSA	–	Rivest, Shamir, Adleman
SSH	–	Secure Shell
SSL	–	Secure Sockets Layer
TCP	–	Transmission Control Protocol
UAC	–	User Account Control
VM	–	Virtual Machine
WDAV	–	Windows Defender Antivirus
WinRM	–	Windows Remote Management
WMI	–	Windows Management Instrumentation
WS-Man	–	Web Services for Management
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Informační technologie se staly nedílnou součástí každodenního života. Odvětví jako jsou např. energetika, zdravotní péče, státní i firemní administrativa či bankovní sektor každý den spoléhají na informační systémy a digitální komunikace. Pro práci s počítačovými systémy bylo vytvořeno mnoho nástrojů, které cílí na zjednodušení a zrychlení rutinních operací. Mnoho technologií může trpět nedostatečným zabezpečením, které bývá přehlíženo na úkor jednoduchosti používání. Jeden z nástrojů pro efektivnější používání a správu systémů je často zneužíván pro potřeby kyberkriminality.

V roce 2006 představil Microsoft nový shell a jazyk s názvem PowerShell pro operační systém Windows. Ten se stal velmi oblíbeným nástrojem pro automatizaci a administraci. PowerShell se rozšířil mezi odbornou veřejnost jako nástroj s rozsáhlým využitím a pokročilými funkcemi, které umožňují ovládat převážnou část systému. Popularitu zajistila integrace do všech operačních systémů Windows od verze 7/Server 2008 R2. Jazyk PowerShell je postaven na platformě .NET, která umožňuje nativně přistupovat k pokročilým systémovým funkcím a zároveň dovoluje jednoduchou implementaci skriptů pomocí využití existujících dostupných předinstalovaných knihoven.

Díky své nativní dostupnosti v systémech Windows a možnosti převzít kontrolu nad systémem je PowerShell čím dál více zneužíván pro tvorbu škodlivých skriptů. Útoky pomocí škodlivých skriptů se také označují jako *fileless* neboli bezsouborové, protože tyto skripty jsou spustitelné z paměti počítače a nevyžadují uložení na disk. Skript je často spouštěn na pozadí a bez vědomí oběti, což usnadňuje jeho úspěšné vykonání bez povšimnutí uživatele. V roce 2018 se PowerShell stal dostupným i na operačních systémech Linux a macOS, což usnadňuje správu více druhů systémů pomocí jednoho nástroje, ale zároveň otevírá dveře možným bezpečnostním hrozbám.

Záměrem této práce je otestovat funkcionalitu, kterou využívají existující škodlivé PowerShell skripty na poslední dostupné verzi PowerShellu a analyzovat jejich funkčnost na různých operačních systémech. Praktickým cílem je implementovat vlastní multiplatformní škodlivé PowerShell skripty, které budou funkční na alespoň dvou OS s následným rozborem jejich funkčnosti.

První kapitola práce se zabývá aktuálním přehledem na poli IT bezpečnosti vzhledem k využití PowerShellu a růstu počtu bezsouborových a Living off the Land útoků (viz kapitola 2). Následující kapitoly rozebírají technickou stránku PowerShellu a často používané techniky ve škodlivých skriptech, stejně tak jako používané metody pro detekci škodlivého kódu. V následující kapitole je popsán průběh a výsledky testování existujících škodlivých skriptů na poslední multiplatformní verzi PowerShellu. Další kapitola popisuje testování a logiku skriptů, které byly vytvořeny v rámci této práce. V poslední části práce se nachází shrnutí práce a výsledků testování s následnou diskuzí nad funkčností a technikami tvorby multiplatformních skriptů a možnostmi prevence útoků zneužívající škodlivé PowerShell skripty.

2 State of the Art

V případě napadení počítače bude nejprve bezpečnostní expert s největší pravděpodobností hledat software, který na daném počítači nemá co dělat. Útočníci proto hledali způsob jak počítač napadnout, aniž by museli k oběti cokoli instalovat. Tzv. *Living off the Land* útok využívá legitimního softwaru, který se již v systému nachází. Bezsouborové útoky často zneužívají legitimní software a nezanechávají za sebou stopy v podobě souboru na disku, proto jsou při následné analýze napadeného systému těžko odhalitelné. Oblíbeným nástrojem pro takové útoky představuje právě PowerShell. [1, 2]

První příklady bezsouborových útoků se objevily na konci roku 1989, kdy byly detekovány destruktivní viry s názvem *Frodo* či *The Dark Avenger*, které dokázaly poškozovat a infikovat ostatní soubory na disku. [3] Během následujících 20 let se objevilo několik dalších bezsouborových virů (např. *Code Red*, *SQL Slammer Worm* či *Lurk Trojan*), avšak ke většímu rozšíření bezsouborových a Living off the Land útoků nedošlo, dokud útočníci neobjevili velký potenciál nástroje PowerShell. [4]

V roce 2014 se objevil pozoruhodný bezsouborový perzistentní útok *PoweLiks*, který využil možnosti uložení se do registrů systému a tím ztížil sebeodhalení. Antivirové nástroje často skenují pouze soubory na disku, ale ne obsah registrů či paměti. V registrech je také možné nastavit, aby byl skript automaticky spuštěn po startu systému. [5, 6]

Nárůst zneužití PowerShellu pro škodlivé účely nastal podle společnosti Symantec v roce 2016. Z analýzy zachycených vzorků vyplývá, že se PowerShell skripty nejčastěji používaly jako *downloadery* pro další malware či *backdoory* pro přístup do systému. [7] Příkladem může být časté zneužití PowerShellu pro instalaci payloadu balíku *MetaSploit*. Metasploit je framework pro penetrační testování, který obsahuje mimo jiné reverzní shell *Meterpreter*. Meterpreter se používá pro vzdálenou správu kompromitovaného stroje a je schopný vytvořit interaktivní PowerShell sezení (*interactive session*). [8]

PowerShell dokáže nativně využít nástroj WinRM (Windows Remote Management), který se používá pro vzdálenou správu systému a navíc je nativně dostupný v OS Windows. To umožňuje útočníkovi za pomoci PowerShellu vzdáleně ovládat napadené počítače. Útočníci nejčastěji zneužívají WinRM k připojení kompromitovaného stroje ke kontrolnímu serveru, který dále ovládá připojené stroje. [9]

Podle společnosti Symantec došlo za rok 2018 oproti roku 2017 k 1000% nárůstu výskytů Living off the Land útoků využívající PowerShell. Ze všech zkoumaných skriptů bylo 0,9% škodlivých, což je přibližně jeden skript ze 111. [10] Podle společnosti McAfee byl v prvním čtvrtletí 2019 zjištěn 460% nárůst v detekovaných škodlivých PowerShell skriptech oproti poslednímu čtvrtletí 2018. [11]

Velmi častým vektorem útoku šíření škodlivých PowerShell skriptů je emailová komunikace. Podle společnosti Symantec v roce 2018 tvořily Office soubory celkem 48% všech škodlivých emailových příloh. Office soubory často obsahují škodlivé Visual Basic makro. Pomocí tohoto

makra je možné stáhnout a spustit škodlivý PowerShell skript. PowerShell skripty se mohou šířit pomocí podvržených odkazů, zranitelností v doplňcích prohlížeče (např. *Flash Player*) a *drive-by* stahovaní. [10]

Poslední známý trend v roce 2019 reprezentoval trojský kůň s označením *Trojan.BrowserAssistant.PowerShell*, který za pomoci PowerShellu injektoval JavaScript kód do internetových prohlížečů. Jeho úkolem bylo krást bankovní přihlašovací údaje. Společnost MalwareBytes dokázala detekovat více než sto tisíc napadení tímto trojským koňem, což je velmi vysoké číslo s ohledem na fakt, že v roce 2018 tento malware vůbec detekován nebyl. Velký vliv na šíření tohoto malwaru bylo právě Office makro *W97M.Downloader*. [12] Podle posledních dostupných dat je aktuálně PowerShell intenzivně využíván pro instalaci *kryptominerů*, a to zejména kryptoměny Monero pomocí skriptu s názvem *WindowsUpdate.ps1*. Žádný z detekovaných útoků není primárně cílen na operační systémy Linux a macOS. Všechny útoky zneužívající PowerShell jsou cílené na OS Windows. [11]

PowerShell obsahuje několik opatření, které mají zajistit vyšší bezpečnost. Nejznámější opatření je politika spouštění skriptů (*ExecutionPolicy*), která omezuje běh skriptů. Problém je v tom, že PowerShell obsahuje možnosti, jak se dá tato politika obejít či vypnout. Což ve výsledku umožňuje útočníkovi najít si způsob, jak skript na stroji oběti nakonec spustit. [9]

Další podporovaný způsob kontroly nad aktivitou PowerShellu je logování. Při každém spuštění i vypnutí PowerShell enginu je zaznamenán do systémového logu záznam o události. Ve vyšších verzích PowerShellu je možné zapnout funkci rozšířeného logování, kdy je do záznamu zanesena i informace o daném skriptu či jednotlivé příkazy, které byly vykonány. Rozšířené logování je jeden z možných způsobů jak lze zjistit, co daný škodlivý skript provedl i když za sebou na disku nezanechal žádný soubor. Rozšířené logování však často nebývá zapnuté z důvodu větší systémové zátěže, možné nadměrné velikosti logu či neznalosti administrátorů. [6, 13]

Oblíbenou metodou, jak uniknout detekci škodlivého skriptu je jeho obfuskace. PowerShell skripty je možno obfuskovat několika možnými způsoby, které jsou popsány dále v této práci. Zatímco neobfuskované škodlivé skripty jsou za použití nástroje Virus Total dobře detekovatelné na základě jejich typických příkazů, hashí či záznamu ve virové databázi, obfuskované skripty jsou bez deobfuskace velmi špatně detekovatelné. [4, 9] Podle studie deobfuskace PowerShell skriptů je pomocí VirusTotal správně detekováno (true positive) až 100% poskytnutých neobfuskovaných škodlivých skriptů. Při použití obfuskace stejných skriptů klesla detekovatelnost na 0-8%. Po použití metod pro deobfuskaci a následné analýze se detekovatelnost zvýšila až na 97%. [14] Podle společnosti Symantec používalo v roce 2016 pouze 8% detekovaných škodlivých skriptů nějaký způsob obfuskace. [7]

Pro zvýšení bezpečnosti systémů je doporučeno zavést několik opatření, které mají snížit riziko útoku zneužívající PowerShell. Nejčastější metodou je whitelistování skriptů na základě hashe. Metoda zakáže spustit jakékoli skripty, které nejsou povoleny nezávisle na nastavení politiky spouštění skriptů. Dále je vhodné zakázat spouštění Office maker a napadnutelné pluginy prohlížečů, které jsou zneužívány pro spuštění PowerShellu a doručení škodlivého skriptu. Pokud

není potřeba vzdálené správy daného systému, je vhodné nástroj WinRM vypnout. Pokud je WinRM potřeba nechat zapnutý, je doporučeno provést WinRM hardening - tvrzení vzdáleného přístupu. To může obsahovat např. zakázání výchozích portů a přechod na jiné porty, zakázání ukládání přihlašovacích dat a ověřených strojů nebo odstranění podporovaných protokolů až na Kerberos a Negotiate. V neposlední řadě je vhodné zapnout rozšířené logování a zaznamenané události pravidelně kontrolovat. [13]

2.1 Shrnutí

Bezsouborové a Living off the Land útoky jsou za posledních několik let na strmém vzestupu, a to z několika důvodů. [4]

- **Jednoduchost** Díky velkému množství dostupných skriptů a penetračních frameworků jsou útoky do velké míry předpřipravené.
- **Spolehlivost** V roce 2017 bylo u 77% úspěšných neautorizovaných proniknutí do systému využito některého bezsouborového útoku.
- **Praktičnost** Living off the Land útoky využívají legitimní software, který se v daném systému už nachází, není třeba složitě instalovat vlastní škodlivý program.
- **Nenápadnost** Bezsouborové útoky jsou obtížné pro detekci antivirovými programy.
- **Neurčitost** Bezsouborové útoky bývají těžko identifikovatelné a rekonstruovatelné, tím pádem je těžší zajistit nová bezpečnostní opatření a prevenci.

I když žádný dosud detekovaný škodlivý PowerShell skript nebyl cílen na více operačních systémů, existuje možnost, že se v budoucnu objeví multiplatformní PowerShell útok. Z tohoto důvodu je problematika bezsouborových a Living off the Land útoků velmi aktuální a je třeba se jí více zabývat. Tato práce se zabývá možností zneužití nástroje PowerShell pro škodlivé účely na různých operačních systémech.

3 Metodika

Kapitola představuje technické detaily PowerShellu a vybrané metody, které se využívají k tvorbě škodlivých skriptů. Cílem kapitoly je seznámení se s PowerShellem, metodami obfuskace, logováním, vzdálenou správou a Antimalware Scan Interface.

3.1 PowerShell

PowerShell je prostředí příkazového řádku založené na úlohách a skriptovací jazyk založený na platformě .NET. PowerShell byl vyvinut pro automatizaci úloh a snadnější správu operačních systémů a procesů. Příkazy prostředí PowerShell umožňují vytváření skriptů pro správu počítačů z příkazového řádku. [15]

První zmínky o shellu příští generace od Microsoftu s názvem Monad Shell (MSH) se objevily v roce 2002, kdy byl publikován tzv. *Monad Manifesto*. Manifest popisoval problém chybějícího nástroje pro automatizaci a administraci systémů Windows a řešení v podobě připravovaného projektu Monad. Windows poskytoval uživatelsky jednoduché grafické prostředí pro administraci, stejně tak jako podporoval široké spektrum programovacích jazyků pro pokročilé systémové programátory. Cílem projektu Monad bylo vytvořit jednoduché, účinné a rozšiřovatelné skriptovací prostředí pro správce systémů. [16]

Monad Shell byl nakonec přejmenován na Windows PowerShell, jehož první verze 1.0 byla vydána v roce 2006. [17] Vývoj Windows PowerShellu pokračoval hlavními verzemi 2.0, 3.0, 4.0, 5.0 a 5.1. Všechny verze Windows PowerShellu byly implementovány v .NET Frameworku pouze pro operační systémy Windows. Windows PowerShell je nativně integrován v operačních systémech Windows od verze 7 a Server 2008 R2. [18]

V roce 2018 byla vydána nová verze PowerShellu, která nese označení PowerShell Core 6.0. PowerShell Core je oproti Windows PowerShell multiplatformní a je podporovaný operačními systémy Windows, macOS a Linux. Toho je dosaženo změnou platformy z .NET Frameworku na .NET Core, která je multiplatformní. [19] PowerShell Core nese stejně jako .NET Core MIT licenci (open source) a zdrojové kódy jsou volně dostupné na GitHubu. [20, 21] Poslední verze PowerShellu Core je verze 7.0, která je založena na .NET Core 3.1. [22] Veškeré multiplatformní skripty v této práci byly testovány a spouštěny na POSH Core verze 7.0, pokud není uvedeno jinak. Windows PowerShell je od vydání Core verze také nazýván jako Desktop PowerShell pro jednoznačné rozdělení verzí (či edicí).

PowerShell je oblíbený nástroj systémových správců pro zjednodušení administrace, skriptování a automatizaci opakujících se systémových procesů. V minulosti spočíval problém v platformě heterogenním prostředí, kdy bylo nutné vytvářet samostatné skripty pro jednotlivé shelly. PowerShell poskytuje jedno interaktivní a skriptovací prostředí, které dokáže komunikovat s nástroji příkazového řádku, WMI (Windows Management Instrumentation), COM objekty (Component Object Model) a .NET knihovnami. Hlavním cílem PowerShellu Core je možnost vytvořit jeden skript, který bude fungovat na více systémech. PowerShell je také vhodný multiplatformní

nástroj pro zpracování strukturovaných dat (např. JSON či XML), práci s REST API a objektově orientovanými modely. [23, 24]

Alternativou PowerShellu může být např. Python, který je také nezávislý na platformě a dá se použít k systémové administraci. Oproti Pythonu má PowerShell výhody v tom, že je integrovaný v systému Windows (v edici Desktop) a obsahuje mnoho předpřipravených nástrojů pro automatizaci a správu systémů. I když Python dokáže interagovat se systémem, je vhodnější pro víceúčelové použití, např. zpracování statistik a dat, machine learning nebo implementaci serverových aplikací. [25]

3.1.1 Konstrukční prvky PowerShellu

PowerShell je textový řádkový shell, jehož jazyk je založen na několika základních stavebních kamenech a principech. Stejně jako v ostatních skriptovacích jazycích je možné pro tok skriptu použít cykly, podmínky a přepínače. Samozřejmostí je spouštění a využití jiných nástrojů v systému. Následující podkapitola představí základní prvky a principy PowerShellu.

Rutina Rutina (anglicky *Cmdlet*) je příkaz, který slouží k jednomu účelu. Jedná se o instanci .NET třídy implementovanou v jakémkoliv podporovaném jazyku. PowerShell po instalaci obsahuje množství rutin, není však problém naimplementovat si rutinu vlastní. Rutiny jsou konvenčně pojmenované ve formátu *Sloveso-Podstatné jméno* pro jednoduché porozumění účelu rutin, např. *Copy-Item* slouží ke kopírování souborů. Rutiny mohou přijímat vstupní parametry a argumenty, stejně tak mohou produkovat datový výstup. [26]

Funkce Funkce (*anglicky Function*) a rutiny jsou velmi podobné. Stejně jako rutina, funkce je příkaz, který slouží k vykonání jednoho účelu. Liší se však v implementaci - funkce je blok znovupoužitelného kódu napsaný v PowerShell jazyku, zatímco rutina je instance třídy, která se nachází např. v DLL knihovně. [27]

PowerShell tedy dokáže použít jakoukoli .NET třídu, vytvořit její objekt a volat její metody. Před použitím však musí být příslušná knihovna s třídou načtena do paměti. [18]

Alias Použití aliasu umožňuje přizpůsobit název volání rutin, funkcí a skriptů. Pro rutinu *Copy-Item* je např. přednastavený alias *copy*. Alias slouží ke zjednodušení psaní (zejména dlouhých) často se opakujících příkazů. [26, 28]

Data Pro operaci s daty používá PowerShell několik typů kontejnerů - proměnné, pole a hashe. Do nich mohou být uloženy textové řetězce (*stringy*), číselné hodnoty a objekty. Pro deklaraci kontejnerů se používá před názvem znak dolaru (*$*). [27]

UNIXové shelly (např. Bash) manipulují se všemi daty jako s textovým řetězcem. PowerShell zachází s daty jako s objekty. Tento objektový přístup umožňuje jednodušší předávání dat v jednom objektu namísto složitějšího rozboru textového řetězce. Např. rutina *Get-Timezone* vrací

objekt, který obsahuje jednotlivé metody a data týkající se místního časového pásma. Tento objekt je možné uložit do jakéhokoli kontejneru a dále s ním pracovat. Ukázka uložení objektu do proměnné prezentuje první řádek výpisu kódu č. 1.

PowerShell také definuje tzv. speciální či automatické proměnné, což jsou proměnné, ve kterých se udržují informace o PowerShellu a konstanty (např. *$TRUE* je konstanta pro logickou jedničku).

```
$TimeZone = Get-TimeZone
$TimeZone | Get-Member | Where-Object MemberType -eq "Property"
```

Výpis 1: Ukázka manipulace s daty a řetězení v PowerShellu

Roura Roura (anglicky *pipe*) slouží k řetězení příkazů (angl. *piping*). Roura se používá, když je potřeba přesměrovat výstup jednoho příkazu do další části kódu. Pro řetězení se používá symbol svislé čáry (|). Ukázku řetězení zobrazuje druhý řádek výpisu kódu č. 1. Do rutiny *Get-Member* je předán objekt uložený v proměnné *TimeZone*, která slouží ke zjištění metod a dat daného objektu. Výstupem rutiny *Get-Member* je opět objekt, který je předán rutině *Where-Object*. [27]

V případě, že přes rouru bude předáváno více objektů, existuje speciální proměnná *$_*, která slouží pro identifikaci aktuálního objektu v rouře. Oproti UNIXovým textovým rourám je zde objektová roura.

Operátor Operátory jsou prvky jazyka a slouží pro provádění matematických, logických, binárních, porovnávacích a ostatních operací s daty. Příklad porovnávacího operátoru je možné vidět ve výpisu kódu č. 1, kde operátorem je *-eq*, který slouží pro ověření shodnosti dvou proměnných. V tomto případě se porovnává obsah řetězce *MemberType* s řetězcem "Property". [28]

Úkoly Úkol (anglicky *Job*) slouží k nezávislému vykonávání části kódu na pozadí. Úkoly se používají k paralelizaci skriptů. [28]

Výjimky a chyby Chyby ve vykonávání příkazu se v PowerShell dělí do dvou kategorií - *Terminating* a *NonTerminating*. Terminating chyba je fatální a skončí ukončením příkazu, popř. skriptu. NonTerminating chyba nemusí způsobit zastavení průběhu příkazu. Ošetření NonTerminating chyb probíhá v POSH pomocí ošetřování výjimek, tedy pomocí *try-catch* bloku. Terminating chyby je možné ošetřit pomocí výrazu *trap*. Při použití trap může být povoleno pokračování či ukončení vykonávání příkazů či skriptu. [26, 28]

Rozšiřitelnost a znovupoužitelnost Rozšiřitelnost PowerShellu je zaručena implementováním vlastních rutin a .NET kódu. Pro znovupoužitelnost kódu jsou podporovány dva způsoby rozšiřitelnosti.

- **PSSnapin** je .NET knihovna, která se musí před použitím instalovat a registrovat v systému. Jedná se o starší způsob, který umožňuje použít zkompilované DLL knihovny.
- **Modul** je preferovaný způsob rozšiřování funkcionality. Modul je balík, ve kterém se můžou nacházet rutiny, funkce a skripty. Modul může být napsaný v PowerShellu i některém z .NET jazyků a může být distribuován textově i binárně. Oproti PSSnapinu se nemusí jednat pouze o DLL knihovnu. Moduly je možné načítat dynamicky až v případě, kdy proběhne volání kódu z daného modulu. [26] Moduly se typicky přidávají do systému při instalaci nového softwaru, aby bylo možné daný software spravovat pomocí PowerShellu.

3.1.2 Podporované platformy

Podle oficiální dokumentace patří mezi podporované OS Windows, Linux a macOS. PowerShell je dostupný pro architektury x86, x86-64 a ARM, přičemž ARM architektura není oficiálně podporována. Minimální požadované verze systémů a Linuxové distribuce dokumentuje tabulka č. 1. PowerShell je neoficiálně dostupný na Linuxových distribucích Kali a Raspbian, na kterých je udržován komunitou. [20, 21, 22]

Veškeré multiplatformní skripty v práci byly testovány a spouštěny na operačních systémech Windows 10 Enterprise Evaluation 1809 (dále jen Windows), Ubuntu 18.04.3 LTS (dále jen Ubuntu) a macOS 10.14.3 (dále jen macOS).

Tabulka 1: Podporované operační systémy

Operační systém	Minimální verze/distribuce
Windows	Windows 7 SP1 Windows Server 2008 R2 SP1
Linux	Ubuntu 16.04 Debian 9 CentOS 7 Oracle Linux 7 Red Hat Enterprise Linux 7 openSUSE 15 Fedora 29 Alpine Linux 3.8
macOS	macOS 10.13

3.1.3 Porovnání PowerShell verzí

PowerShell Core 6 neměl za cíl plně nahradit Windows PowerShell. Jednalo se o první verzi, která byla multiplatformní a měla sloužit pro základní rozšíření PowerShellu na ostatní platformy. POSH Core 6 postrádá množství nástrojů pro správu součástí Windows, které se nenachází v jiných operačních systémech (např. Hyper-V). Platforma .NET Core plně nepodporuje technologie jako WCF služby či Entity Framework, které jsou podporovány v .NET Frameworku.

Tabulka 2: Počet rutin, funkcí a modulů ve Windows PowerShellu 5.1 na Windows 10

Typ	Desktop 5.1
Rutina	625
Funkce	905
Modul	79

Tabulka 3: Počet rutin, funkcí a modulů v PowerShellu Core 6 na různých OS

Typ	Core 6 (Windows)	Core 6 (Ubuntu)	Core 6 (macOS)
Rutina	291	223	223
Funkce	144	107	107
Modul	14	10	10

Z tohoto důvodu nemusí být práce se zmíněnými technologiemi v POSH Core 6 možná. [29] Změnu přináší PowerShell 7, který má časem za úkol nahradit stávající Desktop a Core 6 verze. [18, 30] PowerShell Core není aktuálně nativně dostupný v žádném operačním systému, je třeba jej samostatně doinstalovat.

Množina rutin, funkcí a dostupných modulů je v POSH Core 6 i 7 oproti desktopové edici omezená, viz tabulky č. 2, 3 a 4. Použitý skript pro zjištění těchto počtů zobrazuje výpis č. 2. Počty rutin, funkcí a dostupných modulů byly spočítány po čistých instalacích systémů bez zavedení či stažení jakýchkoli modulů či PSSnapinů. PowerShell má na Ubuntu a macOS v obou verzích stejnou množinu funkcí, rutin a předinstalovaných modulů, v OS Windows disponuje většími možnostmi a to zejména ve verzi 7. Veškeré rutiny, funkce a moduly, které jsou dostupné na OS Ubuntu a macOS jsou dostupné i na Windows. Nezávisle na tom, že je POSH Core multiplatformní, na OS Windows má rozvinutější funkcionalitu díky implementaci od Microsoftu a cíli nahradit Desktopovou edici. Microsoft plánuje Core verzi rozšiřovat a doplňovat funkcionalitu, která nyní není dostupná. [15]

3.1.4 Rozšíření PowerShell

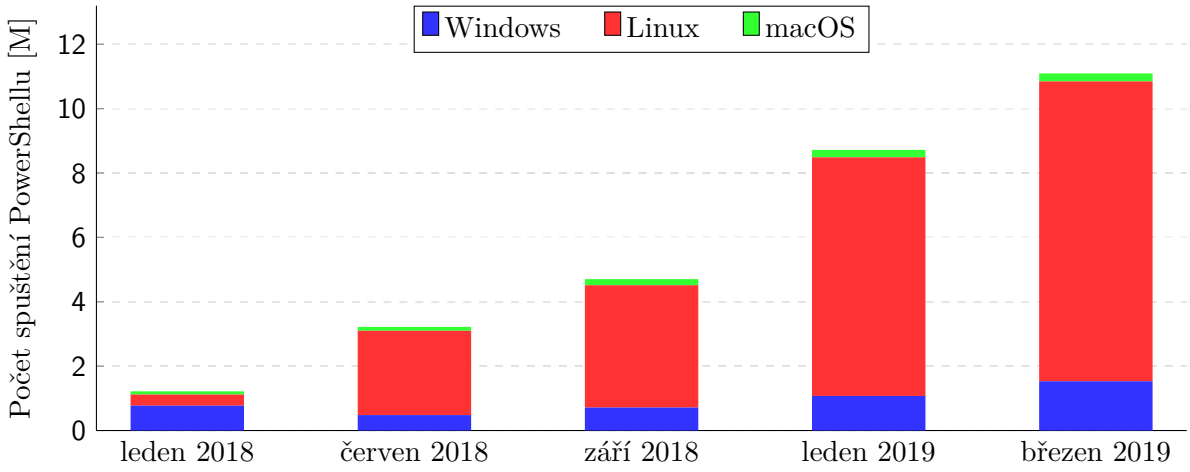
Obrázek č. 1 prezentuje zveřejněná data počtu spuštění PowerShellu 6 na různých OS. Reálný počet spuštění PowerShellu se může lišit, pokud na daném stroji byla zakázána telemetrie. PowerShell Core si našel oblibu zejména mezi linuxovými uživateli. Rozšíření na operačních

Tabulka 4: Počet rutin, funkcí a modulů v PowerShellu Core 7 na různých OS

Typ	Core 7 (Windows)	Core 7 (Ubuntu)	Core 7 (macOS)
Rutina	553	228	228
Funkce	938	41	41
Modul	58	10	10

```
$CMDLETS=(Get-Command * | Where CommandType -eq "Cmdlet" | Measure-Object |
    Select-Object Count).Count
Write-Output "Cmdlets: $CMDLETS"
$FUNCTIONS=(Get-Command * | Where CommandType -eq "Function" | Measure-Object |
     Select-Object Count).Count
Write-Output "Functions: $FUNCTIONS"
$MODULES=(Get-Module -ListAvailable | Measure-Object | Select-Object Count).
    Count
Write-Output "Modules: $MODULES"
```

Výpis 2: Skript pro zjištění počtu rutin a funkcí v dané verzi PowerShellu



Obrázek 1: Počet spuštění PowerShellu Core 6 na různých OS

systémech Windows nebylo velké z důvodu omezené funkcionality oproti Desktopové edici. Proto si Microsoft od přechodu na Core 7 slibuje větší počet uživatelů na Windows. PowerShell však nedokázal vzbudit velký zájem u uživatelů macOS. [30]

3.1.5 PowerShell skripty

Skript je posloupnost příkazů uložených v souboru. Skript je možné manuálně i automaticky opětovně spouštět. Výchozí přípona skriptů pro PowerShell je *.ps1*. Skript typicky obsahuje následující části, přičemž nemusí obsahovat všechny. [26]

- **Aplikační logika** Řešení konkrétního problému.
- **Odchytávání chyb** Ošetření případně vzniklých chyb.
- **Validace vstupu** Ošetření vstupních dat.
- **Logování** Ukládání informací o průběhu skriptu.
- **Konstrukce PowerShell jazyka** Konstrukce a syntaxe jazyka tvořící skript jako celek.

Jelikož POSH Core a Windows POSH nepoužívají shodnou množinu rutin a funkcí, skripty nejsou zcela kompatibilní. Pokud skript používá POSH konstrukce, které se vyskytují v obou verzích, měly by skripty být navzájem kompatibilní. Nicméně je nutné samostatně otestovat každý skript pro kompletní ověření kompatibility. [31]

Z důvodu sdílení stejné přípony pro skripty v Core i Desktop verzi, může být na systémech Windows nutné jasně deklarovat, pro kterou verzi PowerShellu je skript určen. Pro tento účel je možné použít příkaz *Requires* na začátku skriptu, který zajistí, že bude skript spouštěn pouze na dané verzi PowerShellu. Zajištění běhu na PowerShell Core 7 je možné pomocí příkazu ve výpisu kódu č. 3. Skript se dá spustit v definovaném procesu PowerShellu. Zatímco Desktop edice používá pro pojmenování procesu název *powershell.exe*, proces Code edice nese na všech platformách název *pwsh*, respektive *pwsh.exe*.

```
#Requires -PSEdition Core
#Requires -Version 7
```

Výpis 3: Zajištění běhu skriptu na PowerShell Core 7

3.2 Politika spouštění skriptů

Politika spouštění skriptů *ExecutionPolicy* je bezpečnostní opatření PowerShellu pro ošetření načítání konfigurace a povolení běhu skriptů. Cílem politiky je zabránit spuštění skriptů určitým uživatelům či předejít nechtěnému spuštění skriptu. Politika může nabývat různých hodnot: [7, 9, 28]

- **Restrited** Nejpřísnější nastavení politiky. Zakazuje spuštění všech skriptů, povoluje vykonání jednotlivých příkazů.
- **AllSigned** Povoluje spuštění pouze digitálně podepsaných skriptů od důvěryhodného vydavatele. Před spuštěním nedůvěryhodného skriptu skriptu je uživatel dotázán o potvrzení spuštění skriptu.
- **RemoteSigned** Povoluje spuštění všech skriptů, které byly vytvořeny na místním počítači či v místní doméně. Skripty a soubory z internetu musí být důvěryhodné, nebo musí být odemknuty např. pomocí rutiny *Unblock-File*.
- **Unrestricted** Všechny skripty jsou povoleny spouštět. Uživatel bude dotázán na spuštění nedůvěryhodného skriptu.
- **Bypass** Politika spouštění je kompletně vypnuta. Je možné spustit jakýkoli skript bez upozornění uživatele.
- **Undefined** V tomto nastavení politiky je zděděno výchozí nastavení Windows - *Restricted* pro Windows klienty, nebo *RemoteSigned* pro Windows Server.

Tabulka 5: Výchozí politika spouštění skriptů na různých OS

Rámec	Core (Windows)	Core (Ubuntu)	Core (macOS)	Desktop 5.1
LocalMachine	RemoteSigned	Unrestricted	Unrestricted	RemoteSigned
CurrentUser	Undefined	Unrestricted	Unrestricted	Undefined
Process	Undefined	Unrestricted	Unrestricted	Undefined
MachinePolicy	Undefined	Unrestricted	Unrestricted	Undefined
UserPolicy	Undefined	Unrestricted	Unrestricted	Undefined

Politiky spouštění je možné nastavit pro různé rámce systému. Pro místní stroj (*LocalMachine*), aktuálního uživatele (*CurrentUser*) a samostatný proces (*Process*). Politiku spouštění skriptů může měnit politika místního stroje (*MachinePolicy*) či politika pro daného uživatele (*UserPolicy*).

Výchozí politiky spouštění na různých operačních systémech zobrazuje tabulka č. 5. Politika *RemoteSigned* je výchozí politika v obou verzích na testovaném OS Windows. Na jiných platformách je tato politika nastavena na hodnotu *Unrestricted*, protože není podporována. Tohle bezpečnostní opatření je tedy možné zavést pouze na systémech Windows.

Proces PowerShellu je možné spustit s argumentem, který může změnit politiku spouštění skriptů v procesu, viz výpis č. 4. Proces pwsh je možné spustit s argumentem *-ExecutionPolicy Bypass*, který nastaví pro daný proces *Bypass* politiku. Skript je dodán buď s parametrem *-File* a cestou k souboru, nebo je dodán jako příkaz pomocí argumentu *-Command*, kdy je uvnitř nového procesu přečten pomocí rutiny *Get-Content* a vykonán pomocí *Invoke-Expression*. Po vykonání skriptu se uložené nastavení politiky nemění. [32].

```
pwsh.exe -ExecutionPolicy Bypass -File .\script.ps1
pwsh.exe -ExecutionPolicy Bypass -Command "& {Get-Content .\script.ps1 |
    Invoke-Expression}"
```

Výpis 4: Spouštění PowerShellu s arguemtem -ExecutionPolicy

3.3 Profily

Profil je PowerShell skript, který proběhne automaticky po spuštění procesu PowerShellu. Profil slouží k automatickému načtení specifických modulů a nastavení, nebo např. pro změnu vzhledu interaktivního prostředí pro daného uživatele či hostitelskou aplikaci PowerShellu. Profily se hodí v případě oddělení nastavení a omezení/přidání funkčnosti pro různé uživatele a aplikace, což může zvýšit bezpečnost systému.

Celkem se v PowerShellu nachází čtyři profily, které mají různý rozsah působnosti. Cesty k jednotlivým profilům lze nalézt v automatické proměnné *$PROFILE*. Profily se spouští v následujícím pořadí. [15, 28]

- **AllUsersAllHosts** Profil pro všechny uživatele a všechny hostitelské aplikace.
- **AllUsersCurrentHosts** Profil pro všechny uživatele a specifické hostitelské aplikace.
- **CurrentUserAllHosts** Profil pro specifického uživatele a všechny jeho hostitelské aplikace.
- **CurrentUserCurrentHost** Profil pro specifického uživatele a specifickou hostitelskou aplikaci.

Systém profilů je funkční na všech operačních systémech. Problém však nastal na OS Ubuntu, kde se výchozí umístění skriptů pro všechny uživatele nachází v POSH podadresáři ve složce "\snap", která je nastavevená pouze pro čtení. Snapy nedovolují upravovat své soubory. Řešením může být instalace PowerShellu přes balíčkovací systém *apt* místo přichycování balíčku pomocí *snapd*.

Proces PowerShellu je možné spustit s argumentem *-NoProfile*, který zabrání spuštění jakéhokoli profilu v daném procesu. Toho často zneužívají útočníci, kteří tímto způsobem mohou dostat přístup k PowerShellu bez jakéhokoli neznámého nastavení či omezujícího rozšíření. [9]

Pokud má útočník přístup a právo pro zápis k profilovým skriptům, může do nich přidat škodlivý kód. Vykonané příkazy profilu se totiž nezobrazí při spuštění interaktivního prostředí (výjimky jsou příkazy pro vypsání textu na obrazovku či chybová hlášení). Útočník tak může ovládnout každé spuštění PowerShellu s profilem, což se ideálně hodí např. pro keylogger.

3.4 Módy jazyka PowerShell

Mód jazyka PowerShell je nastavení omezení konstrukčních prvků, které je možné v aktuálním sezení použít. Jedná se o omezení konstrukčních prvků, které může zadat sám uživatel k vykonání. Existují celkem čtyři módy jazyka PowerShell. [15]

- **FullLanguage** Jazyk bez omezení. Všechny dostupné konstrukční prvky PowerShellu je možné použít.
- **ConstrainedLanguage** Mód umožňuje použití všech dostupných konstrukčních prvků, ale omezuje typy, které lze použít. Povolené typy jsou základní datové typy (*int*, *string* atd.), vybrané třídy a typy objektů, které je možné vytvořit a dále s nimi pracovat.
- **RestrictedLanguage** Povoluje běh jednotlivých zadaných příkazů, ale není možné spouštět skripty ani bloky příkazů. V módu je také omezená dostupnost modulů, proměnných a operátorů. Přiřazování hodnot k proměnným není povoleno, stejně jako volání metod objektů.
- **NoLanguage** Mód zakazuje použití veškerých konstrukčních prvků PowerShellu. Mód se používá při vytváření instancí PowerShellu s předdefinovaným skriptem a parametry, jakýkoli uživatelský vstup není zpracován.

Aktuálně používaný mód jazyka je možné najít v automatické proměnné *$ExecutionContext.SessionState.LanguageMode*. Výchozí nastavení módu je na všech systémech nastaveno na hodnotu *FullLanguage*. Mód jazyka je možné automaticky nastavovat například v profilu.

Omezení módu jazyka PowerShell je vhodné zejména pro uživatele, kteří PowerShell nevyužívají. V případě, že útočník dokáže kompromitovat stroj oběti, který má omezený mód PowerShell jazyka, nebude útočník schopen provést příkazy či spustit svůj škodlivý skript, který používá omezené konstrukční prvky.

3.5 Nejčastěji používané techniky ve škodlivých skriptech

Škodlivé skripty používají vybrané možnosti PowerShellu, které umožňují ztížit detekci skriptu antivirovými nástroji. PowerShell obsahuje množství nástrojů, které umožňují různé metody obfuskace. Obfuskace slouží ke ztížení analýzy a snížení čitelnosti kódu. Projekt *Invoke-Obfuscation* je volně dostupný na GitHubu a jedná se o obfuskátor PowerShell skriptů a příkazů, který dokáže obfuskovat kód různými metodami. [33, 34] Následující podkapitola popisuje princip nejčastěji se vyskytujících technik pro obfuskaci a snížení detekce vykonání skriptu. Pro nejvyšší pravděpodobnost uniknutí detekce se používá kombinace všech níže popsaných metod, včetně obcházení politiky spouštění skriptů a profilů, viz kapitoly 3.2 a 3.3.

3.5.1 Obfuskace zakódováním skriptu

Jednou z nejpoužívanějších metod obfuskace v PowerShellu je zakódování celého skriptu do kódování Base64. Zakódovaný skript je možné předat v parametru spouštěného procesu pomocí argumentu *-EncodedCommand*. Po spuštění PowerShellu s tímto parametrem se daný skript nejprve dekóduje a pak vykoná. Zakódování umožňuje útočníkovi skrýt celý obsah skriptu do jednoho řetězce, který není pro člověka čitelný. Pokud je detekováno spuštění PowerShellu se zakódovaným skriptem, existuje poměrně vysoká šance, že se jedná o škodlivý skript. [34]

Způsob, kterým je možné v POSH zakódovat skript prezentuje výpis č. 5. Nejprve se do proměnné *$c* načte obsah skriptu k zakódování, který se následně zakóduje a uloží do stejné proměnné, která se přesměruje do textového souboru *EncodedScript.txt*. Následný proces spuštění zakódovaného řetězce prezentuje výpis č. 6. Zde může nastat problém v maximální délce podporovaného řádku v konzoli, což např. na OS Windows je 8191 znaků. Rozsáhlejší skripty v Base64 zakódované podobě nepůjdou spustit, protože budou moc dlouhé. [6, 7]

```
$c = Get-Content .\script.ps1
$c = [System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes(
    $c))
$c > .\EncodedScript.txt
```

Výpis 5: Způsob zakódování skriptu do Base64

```
pwsh -EncodedScript <Base64_String>
```

Výpis 6: Spuštění Base64 zakódovaného PowerShell skriptu

3.5.2 Obfuskace zakódováním příkazů ve skriptu

Base64 zakódovaný skript je možné opět dekódovat a přečíst celý obsah skriptu. Proto útočníci často zakódují i jednotlivé příkazy, které musí být postupně ve skriptu dekódovány. Obvykle jsou ve skriptech zakódovány IP adresy, URL odkazy obsahující další malware, či často používané příkazy ve škodlivých skriptech (viz podkapitola 3.6). Zakódování těchto částí skriptu může útočníkům poskytnout protekci před skenováním skriptů pro určitá klíčová slova či příkazy. Aby byly zakódované části skriptu úspěšně dekódovány, musí v ideálním případě proběhnout několik úrovní dekódování skriptu, což málokterý antivirový nástroj dělá. [33, 34]

Ukázku dekódování a spuštění zakódovaného Base64 příkazu ve skriptu zobrazuje výpis kódu č. 7. Do proměnné *$c* je uložen Base64 zakódovaný příkaz, který je následně dekódován a uložen do stejné proměnné. Dekódovaný příkaz je následně vykonán pomocí příkazu *Invoke-Expression*.

```
$c = <Base64_String>
$c = [System.Text.Encoding]::UTF8.GetString(([System.Convert]::FromBase64String
    ($c)|?{$_}))
Invoke-Expression $c
```

Výpis 7: Příklad použití zakódovaných příkazů ve skriptu

3.5.3 Obfuskace manipulací se stringy

Obfuskaci bez použití zakódování lze provést manipulací se stringy. Pokud je ve skriptu použit podezřelý řetězec (např. IP adresa), tak bude skript s vysokou šancí detekován jako škodlivý. Pokud však budou ve skriptu uložené stringy, které se budou jevit jako náhodné či neznámé řetězce, nebudou tyto stringy s největší pravděpodobností vyhodnoceny jako nebezpečné. Útočníci tohoto faktu často zneužívají, když do skriptu uloží náhodné řetězce, které následně upraví na smysluplné příkazy pomocí operací se stringy. Mezi tyto operace nejčastěji patří záměna znaků, konkatanace, trimování a použití podřetězců. [7, 34]

Příklad obfuskace využitím manipulace s textovými řetězci představuje výpis č. 8. V ukázce jsou do proměnných *$c1* a *$c2* uloženy stringy, které na první pohled vypadají jako nesmyslné shluky znaků. Pomocí funkcí *Substring*, *Replace*, *Trim* a následné konkatanace lze tímto způsobem z původních hodnot vytvořit smysluplný příkaz. V tomto případě bude výsledek uložen v proměnné *$c1*, která bude obsahovat příkaz *Start-Process*.

```
$c1 = '[Sv.qqrt'
$c2 = 'Vbocexd'
$c1 = $c1.Substring(1).Replace('v','t').Replace('.','a').Replace('q','')
$c2 = $c2.Replace('V','P').Replace('b','r').Replace('x','ss').Trim('d')
$c1 = $c1 + '-' + $c2
```

Výpis 8: Příklad obfuskace pomocí manipulace se stringy

3.5.4 Obfuskace použitím únikových sekvencí a randomizace velikosti písmen

PowerShell nerozlišuje rozdíl mezi velkými a malými písmeny (tzv. *case insensitive*). Proto je možné kombinovat velkou i malou abecedu pro dosažení stejného výsledku příkazu. Skenery skriptů můžou rozlišovat rozdíl a skenovat soubor pro daný řetězec s jasně definovanou velikostí písmen. PowerShell obsahuje únikové sekvence pro řetězce, jako např. tabulátor či nový řádek. Tyto sekvence jsou uvozeny znakem gravis (*grave accent*, `), rozlišují rozdíl mezi velkými a malými písmeny a jsou interpretovány pouze v dvojitých uvozovkách. Pomocí únikových sekvencí může být změněn obsah řetězce, podobně jako za použití metod pro stringy. Pokud je zjištěna nedefinovaná úniková sekvence, je s ní zacházeno jako s klasickým písmenem bez uvozovacího znaku, což útočníci často používají jako další možný způsob obfuskace. [14, 35]

Praktickou ukázku demonstruje výpis č. 9. Příkaz *Invoke-Expression* je zapsán pomocí náhodně zvolených velkých a malých písmen a přesto stále vykoná příkaz *Get-Date*, který je uložený v proměnné *$cmd*.

```
$cmd = "`g`E`T`-`d`A`T`E"
InVoKe-eXpReSSion $cmd
```

Výpis 9: Příklad obfuskace pomocí únikových sekvencí a randomizace velikosti písmen

3.5.5 Obfuskace zašifrováním příkazů

Jednotlivé příkazy i části skriptu je možné zašifrovat, aby byly nečitelné pro uživatele. Oproti kódování je v tomto případě vyžadován klíč pro dešifrování. Klíč se musí nacházet ve skriptu, nebo musí být stažen ze sítě. Nejjednodušší šifrovací metodou může představovat Vernamova šifra, mezi složitější algoritmy patří např. AES či RSA.

Příklad zašifrovaného příkazu ve skriptu představuje výpis č. 10. V proměnné *$c* se nachází zašifrovaný příkaz a v proměnné *$k* se nachází klíč k dešifrování. V případě, že byl příkaz zašifrován Vernamovou šifrou, bude příkaz dešifrován pomocí exkluzivní disjunkce a následně spuštěn. Dešifrování probíhá po jednotlivých znacích do proměnné *$cleartext*.

```
Clear-Variable cleartext
$c = $c.ToCharArray()
$k = $k.ToCharArray()
for($i=0; $i -lt $c.Length; $i++)
{
    $cleartext += [char]([byte][char]$c[$i] -bxor $k[$i % $k.Length])
}
Invoke-Expression $cleartext
```

Výpis 10: Příklad obfuskace pomocí šifrování příkazů ve skriptu Vernamovou šifrou

3.5.6 Skrytí okna

Pro snížení šance, že si oběť všimne vykonávání škodlivého skriptu je možné sputit PowerShell na pozadí bez viditelného okna. K tomu slouží při spuštění procesu PowerShellu argument příkazového řádku *-WindowStyle* a hodnota *Hidden*.

3.5.7 Neinteraktivní shell

PowerShell je možné spustit bez interaktivního příkazového řádku, aby nebylo možné do vykonávání skriptu uživatelsky zasáhnout. Ke způsobu spuštění bez interaktivního shellu slouží argument *-NonInteractive*.

3.5.8 Ovládání rozhraní AMSI

Přes PowerShell je možné ovládat rozhraní AMSI (Antimalware Scan Interface, viz podkapitola 3.9) a vypnout ochranu jakéhokoli antivirového programu. PowerShell obsahuje rutinu *Set-MpPreference*, přes kterou je možné nativně ovládat a vypnout Windows Defender. [23]

Tato rutina se využívá zřídka, protože vyžaduje elevované oprávnění. Jakékoli změny v nastavení Defenderu budou zaneseny do systémového logu (např. vypnutí Defenderu je dohledatelné pod EID 5001), což může zvýšit odhalitelnost škodlivého skriptu. [36] Windows Defender se nevyskytuje v operačních systémech Linux a macOS, a proto se rutina *Set-MpPreference* nevyskytuje v Core verzi PowerShellu na těchto OS.

3.5.9 Ostatní metody

Mezi další techniky, které využívají škodlivé skripty se řadí používání datových kontejnerů, jejichž název se mění pokaždé, co se skript vykoná. Názvy bývají generovány na základě iterace spuštění skriptu či náhodné veličiny. Škodlivé skripty mohou používat aliasy a zkrácené varianty argumentů a příkazů kvůli zmenšení velikosti skriptu a skenování pro určité řetězce ve skriptu. Příklad může být zkrácení argumentu *-ExecutionPolicy* na *-ep* či příkazu *Invoke-Expression* na *IEX*. V neposlední řadě útočníci píšou více příkazů na jeden řádek. Jednotlivé příkazy na stejném řádku je možné oddělit pomocí středníku. PowerShell interpretuje nový řádek jako konec

příkazu. Tím pádem není nutné explicitně ukončovat každý příkaz na samostatném řádku, jako v některých jiných jazycích (např. C). [7, 34]

3.5.10 Výskyt argumentů příkazového řádku ve škodlivých skriptech

Podle zveřejněných dat společnosti Symantec z roku 2016 je *NoProfile* nejčastější argument příkazového řádku, se kterými byl PowerShell spouštěn pro exekuci škodlivého skriptu. Procentuální výskyt všech argumentů prezentuje tabulka č. 6. Z tabulky je očividné, že škodlivé skripty kombinují více argumentů pro snížení detekce. Argument *NoLogo* nezobrazí přivítací obrazovku po spuštění PowerShellu a argument *InputFormat* definuje formát vstupních dat. PowerShell umožňuje vstupní data zpracovat jako text, nebo XML. [7]

Tabulka 6: Výskyt argumentů příkazového řádku v detekovaných škodlivých skriptech

Argument příkazového řádku	Výskyt ve všech detekovaných škodlivých skriptech
NoProfile	33,77%
WindowStyle	23,76%
ExecutionPolicy	23,43%
Command	22,45%
NoLogo	18,98%
InputFormat	16,59%
EncodedCommand	6,58%
NonInteractive	3,82%
File	2,61%

3.6 Nejčastěji používané příkazy ve škodlivých skriptech

Nalezené škodlivé skripty sdílely jeden nebo více z příkazů zobrazených ve výpisu č. 11. Tyto příkazy jsou typické pro tvorbu škodlivých skriptů. Zakázání zmíněných příkazů však není řešení, protože bývají používány pro legitimní účely. [7]

Metoda *DownloadFile* objektu typu *WebClient* slouží ke stažení souboru ze sítě. Touto metodou bývá stažen další malware do počítače, zejména zkompilované a spustitelné soubory.

Metoda *DownloadString* stejného typu objektu dovoluje stáhnout ze sítě textový obsah souboru. Tato metoda bývá zneužívána pro stažení obsahu dalšího skriptu.

Rutina *Invoke-WebRequest* slouží k zaslání HTTP/HTTPS požadavku na daný server a poskytne kolekci všech získaných odkazů, obrázků a jiných HTML elementů.

Rutina *Start-Process* se používá ke spuštění nového procesu. Pomocí této rutiny je možné spustit jiný malware či skript, který byl stažen z internetu.

Rutina *Invoke-Expression* se používá pro exekuci dynamicky vytvořeného příkazu. Často je volána po deobfuskaci a dekódování textového řetězce.

```
(New-Object System.Net.Webclient).DownloadFile()
(New-Object System.Net.Webclient).DownloadString()
Invoke-WebRequest
Start-Process
Invoke-Expression
```

Výpis 11: Nejčastěji používané příkazy ve škodlivých skriptech

3.7 Logování aktivity

I když na daném stroji proběhne vykonání škodlivého skriptu, který za sebou nezanechal žádné stopy v podobě souboru na disku, existují metody, jak dokázat jeho spuštění. K popsanému účelu se využívá logování aktivity PowerShellu. Záznamy v logu se používají při forenzní analýze napadeného stroje. Ve výchozím nastavení je logování nastaveno pouze na základní informace o spuštění PowerShellu bez jakýchkoli dalších podrobností o vykonávání příkazů či skriptů. V PowerShellu je také možné nastavit několik úrovní logování (*Log Level*) - poskytnuté množství detailů a záznamů v logu. Log level může nabývat několika hodnot, a to od nejnižší po nejvyšší jsou *Unexpected*, *Monitorable*, *High*, *Medium*, *Verbose* a *VerboseEX*. [6, 15, 23]

PowerShell umožňuje pokročilé možnosti logování. Mezi ně patří například logování načítání modulů (*Module Logging*) a logování bloků spouštěných příkazů (*Script Block Logging*). Script Block Logging dokáže logovat veškeré uživatelské i systémové příkazy, které byly v PowerShellu spuštěny. [15] Pokročilé možnosti logování jsou vhodné pro zpětnou rekonstrukci aktivity PowerShellu, protože logy mohou obsahovat kompletní znění škodlivého skriptu, který byl vykonán a nikam se neuložil. Nevýhodou pokročilých možností logování může být velikost logu, zejména v prostředí, kde se často používá PowerShell. Výhodou PowerShellu je automatická možnost přepisu logu do textového souboru (*Transcription*). Přepis je použitelný pro uložení zaznamenané aktivity jinam, než do systémového logu. [15, 37]

Legitimní PowerShell skripty můžou obsahovat důvěrné informace, které by neměly být nikde zaznamenány. Může se například jednat o hesla či jiné přihlašovací údaje. Pokud je na stroji zapnuto logování všech vykonaných příkazů a útočník dokáže ukrást log obsahující tyto údaje, veškeré důvěrné informace jsou ihned kompromitovány. Pro tento případ PowerShell podporuje tzv. *Protected Event Logging*, neboli logování šifrovaných dat. Poskytovatel události může zašifrovat důvěrné informace, aby nebyly z logu čitelné a zároveň byly zaznamenány všechny události. Pro účel šifrování je v PowerShellu použit standard CMS (Cyptographic Message Syntax), který definuje kryptograficky chráněné zprávy. CMS pro šifrování zprávy definuje infrastrukturu veřejného klíče (PKI). Zašifrované zprávy by měly být čitelné pouze na jiném stroji, který obsahuje privátní klíč. [15, 37]

I když PowerShell umožňuje použití všech výše popsaných pokročilých metod logování na všech podporovaných systémech, každý OS loguje aktivity jiným způsobem. Z tohoto důvodu budou zvlášť popsány způsoby logování PowerShellu pro jednotlivé OS.

3.7.1 Logování na OS Windows

Windows ukládá logy jako sled událostí (tzv. *eventů*). Jednotlivé události jsou ukládány jako UTF-16 zakódované XML soubory s příponou *.evtx*. Pro jednoduché procházení událostí existuje zobrazovač událostí (*Event Viewer*), což je snap-in pro MMC (Microsoft Management Console) s názvem *eventvwr.msc*. Eventy jsou třízeny do několika logických kategorií, které dále podporují adresářovou strukturu pro odlišení a třízení logovaného obsahu. Každá událost má své EID (jednoznačný identifikátor události). Pomocí daného EID je možné dohledat všechny události v logu. Události se dále dělí na tři základní typy, a to oznámení (*Information*), varování (*Warning*) a chyba (*Error*). Každá událost obsahuje položky pro detailnější popis události. Mezi tyto položky patří datum, čas, uživatel, název stroje, EID, poskytovatel a typ události. [6, 9, 15]

Desktop a Core edice PowerShellu logují události do stejné kategorie (*Application and Services Logs*), ale do různého souboru. Zatímco Desktop edice loguje události do souboru "Windows PowerShell.evtx", Core edice loguje události do "PowerShellCore.evtx". EID událostí těchto logů jsou rozdílná. Často se vyskytující EID v souvislosti s PowerShell Core prezentuje tabulka č. 7.

Windows vyžaduje, aby byl poskytovatel událostí (v tomto případě *PowerShellCore*) registrován pro ukládání událostí do logu. K tomu slouží předinstalovaný skript *RegisterManifest.ps1*, který se nachází v domovském adresáři PowerShellu a bývá spuštěn automaticky při instalaci (je možné jej při instalaci nespouštět). Opětovné spuštění s argumentem *-Unregister* dovoluje logování vypnout.

Pokročilé možnosti logování je možné zapnout přes místní skupinové politiky (GPO) či registry. Pro instalaci POSH Core záznamů do GPO je nutné spustit skript *InstallPSCorePolicyDefinitions.ps1* z domovského adresáře PowerShellu, záznamy se neinstalují automaticky (oproti GPO záznamů Desktopové edice). [15]

Tabulka 7: Často se vykytující EID v logu PowerShell Core

EID	Význam
40961	Spouštění POSH konzole
40962	POSH konzole je připravená pro vstup
4100	Neautorizovaný přístup
4103	Provádění příkazu v rouře
4104	Spouštění příkazu, vytvoření scriptbloku

3.7.2 Logování na OS Linux

Linux loguje veškeré zaznamenané události jako řádky textu do souborů v adresáři "/var/log". V tomto adresáři se nachází podadresáře a soubory pro jednotlivé programy či části systému (např. soubor "/var/log/kern" obsahuje log pro kernel). Pro globální log celého systému se používá v závislosti na použité distribuci soubor "/var/log/syslog", nebo "/var/log/messages". Ubuntu - testovací linuxová distribuce pro tuto práci používá soubor "/var/log/syslog". Power-

Shell ve výchozím nastavení loguje aktivity do globálního logu - syslogu. V případě extenzivního logování PowerShellu je vhodné přesměrovat záznamy ze syslogu do samostatného souboru, který bude obsahovat pouze POSH logy. Přesměrování logování je možné pomocí změny nastavení *rsyslog*, což je systémový logovací nástroj používaný v podporovaných linuxových distribucích. Zobrazení logů je možné vypsáním obsahu těchto textových souborů nebo použitím jakéhokoli podporovaného programu pro čtení logů (v Ubuntu se nachází předinstalovaný zobrazovač *gnome-logs*). [37].

PowerShell zapisuje záznamy do syslogu ve formátu zobrazeném ve výpisu č. 12. Formát záznamu definuje hodnotu EVENTID, která má stejný význam jako EID ve Windows událostech. V OS Linux se však nepoužívá číselné označení jako ve Windows. Místo číselné hodnoty je zapsána textová reprezentace ekvivalentního identifikátoru události (např. místo 40961 je zapsáno *Perftrack_ConsoleStartupStart*). [15, 37]

Pro zapnutí pokročilých funkcí logování je nutné v domovském adresáři PowerShellu vytvořit konfigurační soubor *powershell.config.json*, což je JSON soubor definující nastavení pokročilých funkcí. Formát tohoto souboru je možné nalézt v oficiální dokumentaci od Microsoftu. [15]

```
TIMESTAMP MACHINENAME powershell[PID]: (COMMITID:TID:CID) [EVENTID:TASK.OPCODE.
    LEVEL] MESSAGE
```

Výpis 12: Formát záznamu PowerShellu v syslogu

3.7.3 Logování na macOS

Logy se na macOS stejně jako na Linuxu ukládají jako řádky textu. Logy je v tomto OS možné najít v adresářích "/var/log", "/Library/Logs" a "~/Library/Logs". Globální syslog se nachází v umístění "/var/log/system.log". Čtení logů je možné pomocí aplikace *Console* pro zobrazování logů a reportů, vypsáním textového souboru, nebo z terminálu pomocí programu *log*. Ve výchozím nastavení PowerShellu se na systému macOS loguje pouze do paměti a ne do souboru na disku. Pomocí terminálu a příkazu *log config* lze určit, jaké části systému (v tomto případě *com.microsoft.powershell*) mají být logovány perzistentně do syslogu. Pokud PowerShell neloguje perzistentně, je možné aktivitu sledovat pouze v reálném čase pomocí programů *Console* nebo *log*. [15]

Formát záznamu v logu je shodný s formátem zobrazeným ve výpisu č. 12. Zapnutí a konfigurace pokročilých logovacích funkcí probíhá na macOS stejně jako v Linuxu pomocí JSON souboru *powershell.config.json* v domovském adresáři PowerShellu.

3.8 Vzdálený přístup

PowerShell podporuje vzdálenou správu. Vzdálená správa je vhodná pro systémové administrátory pro vykonávání stejných příkazů či skriptů najednou na jednom či více systémech. Pokud je stroj kompromitován a jsou vyzrazeny přihlašovací údaje, může útočník zneužít vzdáleného přístupu a převzít kontrolu nad strojem oběti.

PowerShell umožňuje ovládat jiné stroje využitím protokolů jako SSH (Secure Shell) nebo Telnet. Za použití PowerShellu je možné vytvořit tzv. PowerShell koncový bod (*PowerShell endpoint*). PowerShell endpoint je množina konfigurací na daném systému, která definuje možnosti přihlášení a oprávnění pro vzdálený přístup. Koncový bod může sloužit jako klient i server. PowerShell Core endpoint je možné realizovat přes protokoly WMI, WS-Man (Web Services-Management) nebo SSH. Většina vzdálených přístupů je realizována pomocí protokolů WS-Man a SSH. [15, 37]

Mezi koncovými body PowerShellu je možné vytvořit interaktivní sezení, nebo posílat jednotlivé příkazy a skripty. K tomu slouží rutiny *Invoke-Command* pro vzdálené spuštění příkazu či skriptu, nebo rutina *Enter-PSSession*, která vytvoří nové interaktivní sezení.

Pro koncový bod je možné omezit práva pro snížení šance zneužití vzdáleného přístupu útočníkem. V daném koncovém bodu je možné povolit některé rutiny či funkce a nastavit omezená přístupová práva k určitým souborům či složkám. Takový omezený koncový bod se nazývá *Constrained endpoint*. [15]

3.8.1 PowerShell přes protokol WS-Man

WS-Man je otevřený standard pro protokol, který se používá pro správu zařízení, serverů, aplikací a webových služeb. WinRM je implementace WS-Man od Microsoftu vyskytující se nativně na OS Windows. PowerShell endpointy dokáží spolu komunikovat přes tento protokol. [15]

WinRM dokáže komunikovat šifrovaně i nešifrovaně. Výchozí port pro nešifrovanou komunikaci je 5985 a pro šifrovanou komunikaci 5986. Jelikož se nejedná o známé a ověřené čísla portů, je nutné tyto porty povolit pro TCP protokol na firewallu systému.

Na Desktopové edici PowerShellu se jedná o výchozí možnost vzdálené správy. Vzdálenou správu je možné v systémech Windows zapnout pomocí rutiny *Enable-PSRemoting*, která se vyskytuje i v edici Core. Rutina vytvoří nový endpoint, povolí port ve Windows firewallu a spustí WinRM server. Ve verzi Core je pro povolení WinRM vzdálené správy nejprve nutné povolit plugin pro protokol WinRM (*pwrshplugin.dll*). Povolení pluginu je možné pomocí skriptu *Install-PowerShellRemoting.ps1*, který se nachází v instalačním adresáři PowerShellu. [15]

WinRM nepodporuje nešifrovanou komunikaci, pokud se systémy nenachází ve stejné doméně. V doméně probíhá autentizace a autorizace pomocí protokolů Kerberos a Negotiate. Pro komunikaci systémů, které se nachází v pracovní skupině (*Workgroup*) je nutné použít šifrování pomocí SSL (Secure Sockets Layer). Nevýhoda SSL v pracovní skupině spočívá v obtížné distribuci certifikátů.

PowerShell přes protokol WS-Man je vhodný pro domény, kde se nachází pouze systémy Windows z důvodu nativní podpory WinRM. WinRM také pracuje s protokoly Kerberos a Negotiate, které jsou na systémech Windows výchozí zabezpečovací a autentizační protokoly. Na systémech Linux a macOS se nativně nenachází žádná implementace WS-Man. PowerShell na těchto systémech proto postrádá rutinu *Enable-PSRemoting*. V případě domény s více typy operačních systémů může nastat problém v různých implementacích WS-Man a podporou protokolů Kerberos a Negotiate. Pro doménu s různými OS je proto vhodné použít protokol SSH místo WS-Man. [28, 38]

3.8.2 PowerShell přes protokol SSH

SSH je protokol pro zabezpečenou komunikaci přes nezabezpečené médium. Přes SSH protokol může být bezpečně přenášena jakákoli síťová služba včetně vzdálené správy PowerShellu. SSH pro svou funkčnost používá port 22. Nejznámější volně dostupná implementace SSH se nachází v balíku *OpenSSH*, který je dostupný pro operační systémy Windows a Linux. Na macOS lze použít předinstalovanou implementaci SSH. [15]

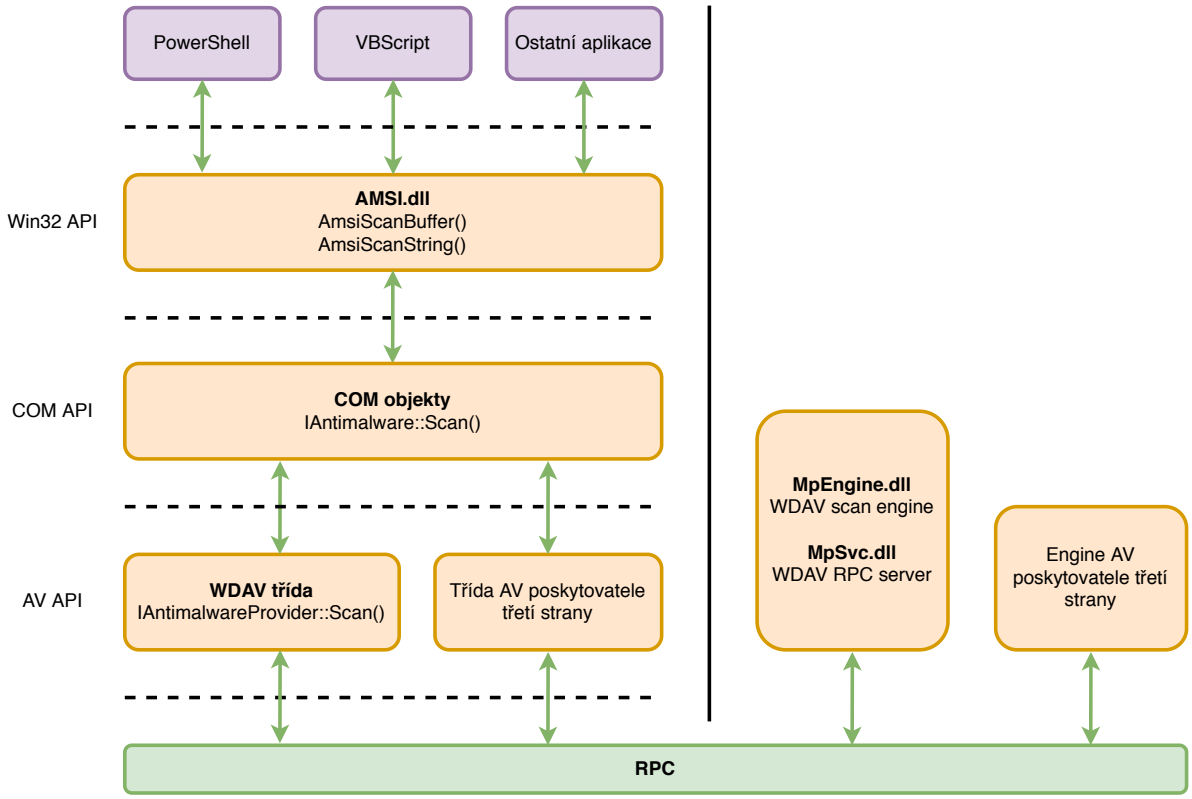
Všechny varianty podporovaných systémů se pro vzdálenou správu PowerShellu konfigurují podobným způsobem, což ulehčuje administrátorskou práci při nastavování vzdáleného přístupu. Nejprve je nutné nainstalovat balík OpenSSH, nebo alespoň jeho části *openssh-server* či *openssh-client*. Pro systémy, které nemají iniciovat spojení a má být k nim být dovoleno pouze se připojit, je vhodné nainstalovat pouze *openssh-client*. Dále je nutné povolit port 22 ve firewallu. Poslední krok je nastavit SSH jako transportní protokol pro vzdálenou správu PowerShellu. PowerShell se nastaví jako tzv. *subsystém* pro SSH v konfiguračním souboru *sshd_config*, který se nachází v konfiguračním adresáři dané SSH implementace. Subsystém vyžaduje název vzdáleného příkazu, což je v tomto případě *powershell* a cestu ke spustitelnému souboru (*pwsh*, případně *pwsh.exe*) s možnými argumenty spouštění. Po uložení konfigurace je nutné restartovat službu *sshd*. [37, 39]

Jelikož je PowerShell Core multiplatformní, SSH je preferovaný protokol ke vzdálenému přístupu k PowerShellu na různých operačních systémech. SSH podporuje autentizaci pomocí hesla i certifikátu. Při využití SSH také nezáleží, zda jsou Windows systémy v doméně nebo v pracovní skupině. Možnost autentizace heslem eliminuje nutnost složité distribuce certifikátů.

3.9 Antimalware Scan Interface

AMSI je rozhraní pro systémy Windows, přes které je možné ovládat antimalwarová řešení na daném stroji. AMSI slouží k vylepšení ochrany uživatelů, dat a aplikací. AMSI bylo vytvořeno pro tvůrce antivirových programů ke standardizaci a zjednodušení skenování podezřelého obsahu. Aplikace může přes AMSI API vyžádat skenování souboru nástrojem, který podporuje dané API. Operační systém se má chovat jako prostřední článek mezi aplikacemi a daným antimalwarovým

Obrázek 2: Architektura AMSI



programem. Pomocí AMSI je možné skenovat soubory, části paměti i datové proudy (streamy). [35, 36]

Windows má v sobě integrovaný Windows Defender Antivirus (WDAV) - základní antivirus OS Windows, který využívá AMSI ke svému chodu. Mezi další programy a mechanismy v OS Windows, které využívají AMSI patří PowerShell (od verze 5), UAC (User Access Control), Windows Script Host, JavaScript a Visual Basic skripty včetně Office maker. Mezi nejznámější antimalwarové řešení třetích stran, které podporují AMSI, patří např. ESET, AVG či BitDefender. AMSI je navrženo tím způsobem, aby se jako skener mohl registrovat jakýkoli antimalwarový software (engine) a jakákoli aplikace jej mohla za použití rozhraní využít. [40, 41]

Architekturu AMSI zobrazuje obrázek č. 2 (převzato z [40]). Při spuštění AMSI podporovaných procesů je do paměti procesu načtena knihovna *AMSI.dll*, která obsahuje metody sloužící ke skenování obsahu (např. *AmsiScanBuffer* nebo *AmsiScanString*). AMSI.dll operuje na Win32 API úrovni, což je API pro operační systémy Windows. Z knihovny je možné přes COM API zavolat metody tříd registrovaných poskytovatelů antimalwarového řešení (tzv. *Provider*). Všechny tyto třídy musí implementovat rozhraní *IAntimalwareProvider*. Pomocí RPC (Remote Procedure Call) je možné požadovaný obsah dále skenovat na odděleném místě. [40, 42]

AMSI se používá pro detekci škodlivých PowerShell skriptů. Pokud je spuštěn škodlivý PowerShell skript, který je zakódován nebo obfuskován, v paměti musí být před vykonáváním

jednotlivých příkazů dekódován a deobfuskován na čistý kód (plain code), který je srozumitelný pro PowerShell. Pomocí AMSI je možné skenovat průběh vykonávání skriptu v paměti. To je výhodné zejména v případě, kdy samotný skript před spuštěním nebyl detekován jako škodlivý z důvodu nedostatečného dekódování či neznámé signatury. [35, 40]

AMSI je možné pro konkrétní proces vypnout. K obejití AMSI je možné použít několik různých metod. PowerShell dokáže AMSI vypnout pomocí rutiny *Set-MpPreference* (viz podkapitola 3.5.8), podvrhnutím knihovny AMSI.dll při načítání do paměti, nebo modifikací dat v .NET Assembly. Vypnutí AMSI má za následek nemožnost skenování provádění škodlivého skriptu pomocí antivirového (AV) software využívající tohle rozhraní. [35]

4 Testování existujících skriptů

Kapitola se zabývá testováním již vytvořených skriptů na multiplatformní verzi PowerShellu. Cílem kapitoly je ověřit, zda existující škodlivé skripty určené pro Desktopovou edici PowerShellu jsou funkční i v Core edici na různých operačních systémech.

K testování zneužitelnosti PowerShellu existují různé frameworky (rámce), které jsou vytvořeny jako pomůcky a nástroje pro penetrační testery. Zmíněné frameworky obsahují škodlivé skripty i skripty určené k testování neoprávněného vniknutí do systému.

Testování probíhalo v elevovaném procesu PowerShellu 7 (jako *root* na Ubuntu a macOS, *Administrator* na Windows) s vypnutými antimalwarovými řešeními (AV, firewall atd.) a nastavenými konfigruacemi, které jsou nezbytné pro běh některých PowerShell rutin a funkcí. Všechny skripty, které byly vybrány k testování implementují metody, které by měly být dostupné na více operačních systémech (např. pořizování snímku obrazovky, logování stisknutých kláves) a nejsou pevně vázány na funkcionalitu OS Windows (např. manipulace s GPO, NTFS souborovým systémem či PE hlavičkami). Všechny testované skripty byly pro ověření jejich funkčnosti nejprve spuštěny ve Windows PowerShellu.

4.1 PowerSploit

PowerSploit je kolekce skriptů a modulů, která je využívána k testování zranitelností PowerShellu. Pomocí PowerSploitu jsou penetrační testeři schopni mimo jiné vykonat neautorizovaný kód, testovat perzistenci skriptů či krást data. PowerSploit je určen pro Windows Powershell od verze 2.0. [43, 44] V rámci této práce bylo otestováno několik vybraných funkcí z různých modulů PowerSploitu.

4.1.1 Modul Recon

Využívá se v průzkumné fázi penetračního testování. V modulu se nachází funkce, které skenují porty, IP adresy, Windows domény a přítomnost souborů přes HTTP dotazy. Výsledky testování zobrazuje tabulka č. 8.

Tabulka 8: Dostupnost funkcí z modulu Recon na různých OS

Funkce	Windows	Ubuntu	macOS
Invoke-Portscan	ANO	ANO	NE
Get-HttpStatus	ANO	ANO	ANO
Invoke-ReverseDnsLookup	ANO	ANO	ANO
PowerView	NE	NE	NE

4.1.2 Modul Exfiltration

Obsahuje funkce, které zneužívají PowerShell k odcizení dat. Mezi testované skripty patří funkce, které zachytávají stisknuté klávesy či zvuk mikrofonu, pořizují snímky obrazovky a pořizují výpisy paměti procesů (*memory dump*). Výsledky testování prezentuje tabulka č. 9.

Tabulka 9: Dostupnost funkcí z modulu Exfiltration na různých OS

Funkce	Windows	Ubuntu	macOS
Get-KeyStrokes	NE	NE	NE
Get-MicrophoneAudio	NE	NE	NE
Get-TimedScreenshot	NE	NE	NE
Out-Minidump	NE	NE	NE

4.1.3 Modul ScriptModification

Slouží k modifikaci, kódování a šifrování skriptů a knihoven. Výsledky testování prezentuje tabulka č. 10.

Tabulka 10: Dostupnost funkcí z modulu ScriptModification na různých OS

Funkce	Windows	Ubuntu	macOS
Out-EncodedCommand	ANO	ANO	ANO
Out-CompressedDll	ANO	ANO	ANO
Out-EncryptedScript	NE	NE	NE
Remove-Comments	ANO	ANO	ANO

4.1.4 Modul Mayhem

Slouží pro dokázání škodlivosti PowerShellu. Modul obsahuje funkci *Set-MasterBootRecord*, která slouží k přepsání hlavního spouštěcího záznamu (Master Boot Record) a funkci *Set-CriticalProcess*, která má zaručit pád systému po ukončení procesu PowerShellu. Výsledky testování prezentuje tabulka č. 11.

Tabulka 11: Dostupnost funkcí z modulu Mayhem na různých OS

Funkce	Windows	Ubuntu	macOS
Set-MasterBootRecord	NE	NE	NE
Set-CriticalProcess	NE	NE	NE

4.2 PowerShell Empire

Jedná se o post-exploitační framework obsahující mimo jiné škodlivé PowerShell skripty. Empire je soubor skriptů čerpající z ostatních frameworků jako jsou PowerSploit, Posh-SecMod a UnmanagedPowerShell.

Framework dokáže využít agenta, který slouží k udržení spojení se strojem oběti, aby mohl útočník interaktivně ovládat stroj oběti. Agent je v závislosti na OS implementovaný pro Windows v PowerShellu (kompatibilní s verzí 2.0) a pro Linux a macOS v Pythonu (2.6/2.7). Agent po spuštění zůstává běžet na pozadí a naváže vzdálené připojení k ovládacímu serveru útočníka. [45, 46]

4.2.1 Modul Collection

Obsahuje funkce pro různé využití. Nachází se zde např. skripty pro pořízení snímku obrazovky, zachytávání provozu sítě či pro krádež uložených dat a přihlašovacích údajů z prohlížečů Chrome, Firefox a Internet Explorer. Výsledky testování se nachází v tabulce č. 12.

Tabulka 12: Dostupnost funkcí z modulu Collection na různých OS

Funkce	Windows	Ubuntu	macOS
Get-FoxDump	NE	NE	NE
Get-ChromeDump	NE	NE	NE
Get-BrowserData	ANO	NE	NE
Invoke-NetRipper	NE	NE	NE
Get-ScreenShot	ANO	NE	NE

4.2.2 Modul Exfil

Obsahuje skripty pro generování síťového provozu a odesílání kusů uživatelských dat na vzdálený server. Výsledky testování prezentuje tabulka č. 13

Tabulka 13: Dostupnost funkcí z modulu Exfil na různých OS

Funkce	Windows	Ubuntu	macOS
Invoke-EgressCheck	ANO	ANO	ANO
Invoke-PostExfil	ANO	ANO	ANO
Invoke-ExfilDataToGitHub	ANO	ANO	ANO

4.2.3 Modul Fun

Slouží k pobavení a nastolení chaosu na stroji oběti. Příkladem může být funkce s názvem *Invoke-Thunderstruck*, která po vykonání opakovaně zvyšuje hlasitost a spustí na pozadí video písně Thunderstruck od AC/DC. Výsledky testování prezentuje tabulka č. 14.

Tabulka 14: Dostupnost funkcí z modulu Fun na různých OS

Funkce	Windows	Ubuntu	macOS
Invoke-Thunderstruck	ANO	NE	NE
Invoke-VoiceTroll	NE	NE	NE
Set-WallPaper	NE	NE	NE

4.3 Nishang

Kolekce post-exploitačních PowerShell skriptů. Narozdíl od frameworku Empire zde není agent pro udržování spojení. Skripty v Nishangu pro stejnou funkcionalitu (např. keylogger) nejsou shodné se skripty v ostatních frameworcích, jsou implementovány jinými autory. Framework také obsahuje skript s názvem *Invoke-AmsiBypass*, který slouží k obejití AMSI. Na PowerShell Core však není funkční. [47, 48]

4.3.1 Modul Gather

Obsahuje skripty pro získání informací o systému. Mezi ně patří např. skript pro zjištění běhu OS ve virtuálním stroji (VM), keylogger či skripty pro krádež dat a hesel. Výsledky testování prezentuje tabulka č. 15.

Tabulka 15: Dostupnost funkcí z modulu Gather na různých OS

Funkce	Windows	Ubuntu	macOS
Keylogger	ANO	NE	NE
Check-VM	ANO	NE	NE
Get-Wlan-Keys	ANO	NE	NE
Invoke-CredentialsPhish	ANO	NE	NE

4.3.2 Modul Shells

Obsahuje skripty pro vytvoření reverzních shellů přes různé protokoly. Reverzní shell se používá ke zpětnému připojení ke stroji útočníka, odkud může útočník ovládat stroj oběti. Výsledky testování zobrazuje tabulka č. 16.

Tabulka 16: Dostupnost funkcí z modulu Shells na různých OS

Funkce	Windows	Ubuntu	macOS
Invoke-PowerShellTcp	ANO	ANO	ANO
Invoke-PowerShellUdp	ANO	ANO	ANO
Invoke-PoshRatHttp	ANO	NE	NE

4.3.3 Modul Client

Obsahuje funkce, které dokáží infikovat určité soubory škodlivým PowerShell skriptem. Mezi často používané soubory pro infikaci patří např. Office dokumenty, které nesou škodlivý obsah a makro, které tento obsah spustí. Výsledky testování prezentuje tabulka č. 17.

Tabulka 17: Dostupnost funkcí z modulu Client na různých OS

Funkce	Windows	Ubuntu	macOS
Out-Word	NE	NE	NE
Out-Excel	NE	NE	NE

4.4 Poznámky k testování

Testované frameworky jsou primárně určeny pro Windows PowerShell verze 2.0. Na testovacím stroji se však nacházela verze 5.1, která způsobila určité problémy v kompatibilitě na Desktopové edici při ověřování funkčnosti skriptů. Častý problém byl v řádku, který zobrazuje výpis č. 13. Pro funkčnost ve verzi 5.1 musel být nahrazen kódem, který prezentuje výpis č. 14. Problém se týkal zejména frameworku PowerSploit.

Verze 2.0 již není podporována a je považována za méně bezpečnou z důvodu chybějící podpory pro rozšířené logování a integrace s AMSI. Windows PowerShell 5.1 je výchozí verze na použitém testovacím systému, a proto byly skripty nejprve testovány na této verzi.

```
$GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddress')
```

Výpis 13: Nekompatibilní často se vyskytující řádek ve skriptech PowerSploitu

```
$GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddress', [reflection.
    bindingflags] "Public,Static", $null, [System.Reflection.CallingConventions
    ]::Any, @((New-Object System.Runtime.InteropServices.HandleRef).GetType(),
    [string]), $null);
```

Výpis 14: Patch pro kompatibilitu některých PowerSploit skriptů na Windows PowerShell 5.1

4.5 Závěr testování

V rámci práce bylo vybráno a otestováno celkem 34 funkcí z 10 modulů tří frameworků pro penetrační testování PowerShellu. Veškeré testované skripty jsou funkční ve Windows PowerShellu. V Core edici bylo na systému Windows funkčních celkem 56% ze všech testovaných skriptů, na Ubuntu 32% a na macOS pouze 29% testovaných skriptů. Mnoho existujících skriptů tedy není funkčních v Core edici na více operačních systémech.

Testované skripty a funkce často nebyly funkční na operačních systémech Linux a macOS z důvodu využití Windows API a registrů, které se na daných OS nenachází. Skripty, které fungovaly na všech operačních systémech využívaly pouze .NET Core knihoven. Častý problém, který způsoboval nefunkčnost existujících skriptů v Core edici na OS Windows, je nesprávné volání či chybějící reference na tzv. *unsafe* metody Windows API. Tyto metody jsou nízkoúrovňové a využívají ukazatele (pointery) pro manipulaci s daty v paměti. S největší pravděpodobností se jedná o částečnou nekompatibilitu implementace .NET Frameworku a .NET Core. Skripty, které byly funkční v Desktopové edici nebyly ve stejném znění funkční v Core edici. Příčinou také mohly být některé chybějící nativní funkce či rutiny (v Core edici chybí např. rutina *Get-WmiObject*).

Mnoho skriptů z různých frameworků nebylo vůbec testováno, protože mají za úkol modifikovat určité části systému Windows, které se nenachází v jiných systémech. Příkladem může být skript *Invoke-Mimikatz*, který slouží pro bezsouborové spuštění softwaru Mimikatz. Mimikatz dokáže na systémech Windows obcházet bezpečnostní opatření (např. vygenerování *Kerberos Golden Ticket*) či krást hesla uložené v paměti.

5 Vlastní implementace škodlivých multiplatformních skriptů

V rámci práce bylo vytvořeno celkem šest multiplatformních skriptů, které se dají rozdělit do tří kategorií. Mezí ně patří keylogger, ransomware a botnet. Kapitola popisuje princip fungování vytvořených skriptů. Cílem kapitoly je objasnit, jakým způsobem bylo dosaženo funkčnosti skriptů na více operačních systémech.

5.1 Keylogger

Keylogger slouží k zachytávání stisknutých kláves na klávesnici. Jedná se o sledovací software, který využívají útočníci pro krádež hesel či citlivých údajů, které oběť zadává pomocí klávesnice do počítače. [49] Způsob zachytávání kláves je specifický pro každý OS, a proto žádný z testovaných existujících keylogger skriptů nebyl funkční na systémech Linux a macOS.

Vytvořený keylogger práce je realizován v jednom skriptu s názvem *KeyLogger.ps1* a je funkční na systémech Windows a Linux. Základní diagram aktivit skriptu prezentuje obrázek č. 3. Vytvořený skript obsahuje dvě funkce, které jsou závislé na typu operačního systému. Jedná se o funkce *LinuxFunc* a *WindowsFunc*. V těchto funkcích se nachází další příkazy a funkce, které realizují zachytávání kláves na daném systému pomocí operací se soubory a knihovnami, které se nachází pouze v daném typu OS. Skript pak obsahuje společnou multiplatformní část, kterou využívají obě funkce. Ve společné části se nachází funkce *Write-This*, která se používá pro implementaci různých výstupů zachycených kláves. Výstup je možný realizovat buď do hostitelské aplikace, textového souboru či odeslání na vzdálený FTP server. Výstup do hostitelské aplikace probíhá po znacích, zatímco zápis znaků do souboru a odesílání pomocí FTP je řešeno pomocí bufferu.

Ve výše uvedeném diagramu aktivit se nenachází symbol pro ukončení skriptu - koncový uzel. Skript po spuštění a úspěšném nalezení všech potřebných závislostí nikdy sám neskončí, aby bylo zachyceno co nejvíce stisknutých kláves. Běh tohoto skriptu se ukončí při vyvolání terminující chyby, nenalezení požadovaných knihoven, souborů či cest, nebo přerušení běhu procesu pomocí zaslání příslušného signálu k ukončení či klávesové zkratky CTRL+C.

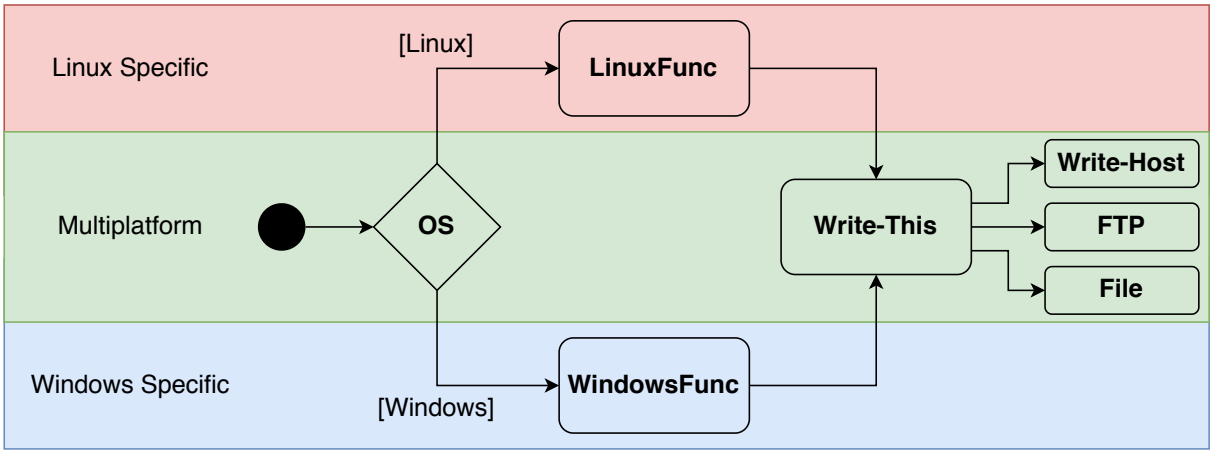
V následujících podkapitolách jsou popsány způsoby, pomocí kterých se v tomto skriptu zachytávají klávesy na jednotlivých systémech.

5.1.1 Linux keylogger

Keylogger na systémech Linux je řešen pomocí zachytávání vstupních struktur klávesnice a jejich následného rozboru na text.

Obecný princip čtení dat ze vstupního zařízení Klávesnice při stisku klávesy vyšle počítači skenovací kód klávesy (*scancode*). Tento kód je dlouhý jeden až tři byty. Kernel operačního systému tento skenovací kód přeloží na klávesový kód (*keycode*), který má pevnou velikost dva

Obrázek 3: Základní tok aktivity skriptu keyloggeru



byty. Tento klávesový kód je potom v závislosti na rozložení klávesnice přeložen na určitý symbol (*keysym*). O překlad na symbol se v grafickém prostředí stará X Window System (či jiný server vykreslující uživatelské rozhraní) a v konzoli je překlad řešen pomocí balíčku *kbd*. [50]

V adresáři "/dev/input" je možné najít soubory reprezentující jednotlivá připojená vstupní zařízení jako jsou například myš, klávesnice či webkamera. Soubory v tomto adresáři mají název *eventX*, kde X je celé číslo. Nalezení korespondujícího čísla souboru *eventX* ke klávesnici je možné například pomocí vypsání všech vstupních zařízení ze souboru "/proc/bus/input/devices", "/dev/input/by-id" nebo "/dev/input/by-path". [51]

Soubory reprezentující vstupní zařízení je možné číst. Při vstupu ze zařízení se v tomto souboru objeví nová binární data, která se dají reprezentovat pomocí struktur *input_event* zobrazené ve výpisu č. 15. Struktura obsahuje celkem čtyři položky, a to časové razítko, typ, kód a hodnotu. Struktura *timeval* pro časové razítko má rozdílnou velikost na 32 a 64-bitovém systému. Z tohoto důvodu je nutné při čtení vstupních struktur rozlišovat používanou architekturu. Celá struktura *input_event* má tedy velikost 16 bytů na 32-bitovém systému a 24 bytů na 64-bitovém systému. [52] V PowerShellu je možné vstupní strukturu reprezentovat objektem, jehož vytvoření zobrazuje výpis č. 16.

Struktury je možné logicky dělit podle typu, který je zjistitelný z položky *type*. Položky *type*, *code* a *value* jsou číselné hodnoty, jejichž význam je definován v hlavičkovém souboru *input-event-codes.h* ve zdrojovém kódu Linuxového kernelu. [53] Například typ 1 je označen jako *EV_KEY*, který značí změnu stavu dané klávesy. V takové struktuře je v položce *code* uložen klávesový kód (*keycode*) a v položce *value* informace o stisku (hodnota 1), držení (hodnota 2) či uvolnění klávesy (hodnota 0). Klávesový kód však značí pouze danou klávesu (např. hodnota 30 je definována jako *KEY_A*), ale nenese žádnou informaci o symbolu, který má tato klávesa reprezentovat.

```c
struct input_event {
    struct timeval time;
    unsigned short type;
    unsigned short code;
    unsigned int value;
};
```

Výpis 15: Struktura vstupní události v jazyce C

```powershell
$Evt = New-Object Object
$Evt | Add-Member Time_sec ([ulong])
$Evt | Add-Member Time_usec ([ulong])
$Evt | Add-Member Type ([ushort])
$Evt | Add-Member Code ([ushort])
$Evt | Add-Member Value ([uint])
```

Výpis 16: Vytvoření objektu korespondující se strukturou vstupní události v jazyce PowerShell

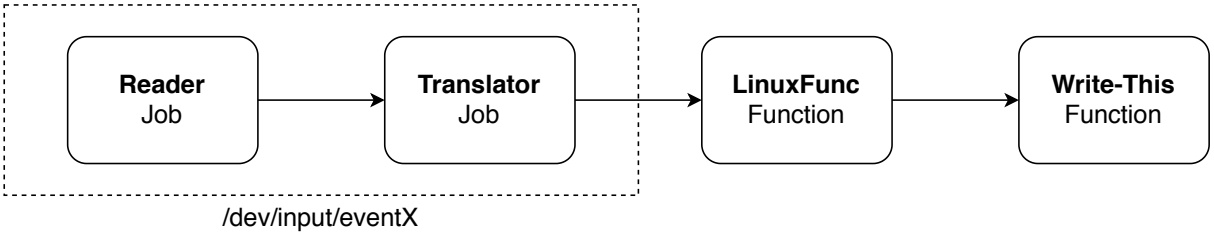
Realizace v PowerShellu Na systémech Linux je nutné skript keyloggeru spustit s oprávněním *root*, nebo pod uživatelem patřícím do skupiny *input*, aby byl povolen přístup ke čtení souborů *eventX*. PowerShell umožňuje příkaz *#Requires -RunAsAdministrator* na začátku skriptu, který zajistí běh jako administrátor, nicméně tento příkaz je dostupný pouze na systémech Windows. Pro zajištění běhu na systémech Linux je nutné v průběhu skriptu ověřit, zda má uživatel přístup k těmto souborům. Toho se dá dosáhnout například příkazem *whoami* nebo *id* pro zjištění uživatele či skupin, do kterých uživatel patří.

Po spuštění funkce *LinuxFunc* je z konfiguračních souborů "/etc/default/keyboard", případně "/etc/vconsole.conf" načteno rozložení klávesnice. Dále se do paměti načte slovník dvojic ve formátu [*keycode*, *definice*], např. [30, KEY_A] ze zdrojových kódů nacházející se v podadresářích složky "/usr/src". Poté se detekují připojené klávesnice a odpovídající soubory *eventX*.

Běh keyloggeru je na systémech Linux řešen pomocí jobů. Pro každý soubor reprezentující klávesnici jsou spuštěny dva joby - *Reader* a *Translator*. Reader má za úkol v nekonečné smyčce číst binární data ze souboru *eventX* po 16 či 24 bytech v závislosti na 32 či 64 bitovém systému. Z jobu Reader je počet bytů následně přečten jako jeden objekt v jobu Translator. Translator má za úkol provádět rozbor binárních dat do objektu, jehož definici zobrazuje výpis č. 16. Výstupní objekt je pak přečten ve funkci *LinuxFunc*, která slouží jako agregátor těchto objektů z různých klávesnic.

Funkce *LinuxFunc* poté překládá data z přečtených objektů na určité znaky. Z důvodu možnosti překládání virtuálních kláves na určité znaky pomocí různých překladačů (X Window System, kbd...) není možné unifikovat přesný překlad na všech instalacích systému Linux. Z tohoto důvodu obsahuje skript funkci *Translate*, která v závislosti na načteném rozložení klávesnice přeloží virtuální klávesu na definovaný znak. Skript keyloggeru obsahuje možnost překladu virtuálních kláves na určité znaky z anglické a české klávesnice. Přeložený symbol je pak předán do multiplatformní funkce *Write-This*. Popsaný tok dat reprezentuje obrázek č. 4

Obrázek 4: Tok dat v linuxové části keyloggeru



5.1.2 Windows keylogger

Keylogger na systémech Windows je řešen pomocí zavěšení funkce na frontu zpráv obsahující obsahující informace o nízkoúrovňových klávesnicových vstupních událostech.

Obecný princip použití háků ve Windows Hák (anglicky *Hook*) je bod v mechanismu předávání zpráv v systémech Windows. Posloupnost háků se nazývá hákový řetěz (anglicky *Hook chain*), což je implementace fronty zpráv, která se využívá v OS Windows. Windows obsahuje několik hákových řetězů, každý pro jednotlivý typ háku. Typy háků jsou definované v dokumentaci systému Windows. Mezi jeden z typů háků patří typ s označením *WH_KEYBOARD_LL*. V řetězu tohoto typu háku se posílají zprávy o nízkoúrovňových klávesnicových vstupních událostech.

Aplikace (v tomto případě PowerShell skript) může registrovat svou funkci jako hák do hákového řetězu. Registrace funkce jako háku do řetězu se také nazývá zavěšení. Takový hák slouží pro odchytávání a monitorování obsahu systémové zprávy ještě před tím, než se zpráva dostane k cílovému procesu. Zachycenou zprávu je možné přečíst a poté zahodit, či poslat dalšímu háku v řetězu. [54]

Háky se používají pro debugování aplikací či simulování vstupních událostí. Používání háků zvyšuje režii systému, který musí obsloužit všechny háky v daném řetězu, než se zpráva dostane k cílovému procesu. Pokud je zavěšená funkce výpočetně náročná, může způsobit značné zpoždění v čase doručení zprávy. Z tohoto důvodu je doporučeno používat háky pouze v nutných případech po co nejkratší dobu.

Keylogger může zavěšeného háku zneužít tím, že si zprávu z klávesnice přečte a poté ji přepošle dalšímu háku v řetězu. Oficiální syntaxe funkce v jayzce C++, kterou je možné zavěsit, zobrazuje výpis č. 17. Syntaxe této funkce v PowerShellu představuje výpis č. 18.

V případě zavěšení do řetězu háku typu *WH_KEYBOARD_LL* se při volání zavěšené funkce v proměnné *nCode* nachází informace, zda má být zpráva přeposlána bez úprav (záporná hodnota), nebo je možné do ní zasahovat (hodnota 0). V proměnné *wParam* se nachází informace o tom, zda je klávesa stisknuta (hodnota 0x0100) či uvolněna (hodnota 0x0101). V proměnné *lParam* se nachází ukazatel na strukturu typu *KBDLLHOOKSTRUCT*. Struktura nese obdobné

informace jako struktura *input_event* na systémech Linux. Ve struktuře se nachází mimo jiné informace o kódu virtuální klávesy, skenovacím kódu či časovém razítku.

Stejně jako na systémech Linux je skenovací kód z klávesnice přeložen na klávesový kód, který se v systémech Windows nazývá kód virtuální klávesy (*Virtual-Key Code*). Kód virtuální klávesy je možné za použití Windows API přeložit na určitý Unicode znak.

```
LRESULT HookProc(
  int nCode,
  WPARAM wParam,
  LPARAM lParam
)
{
   //Procedure Handling
   return CallNextHookEx(NULL, nCode,
        wParam, lParam);
}
```

Výpis 17: Syntaxe funkce v jazyce C++, kterou je možné zavěsit

```
function HookProc {
 param (
  [Parameter(Mandatory = $true)]
  [int] $nCode,
  [Parameter(Mandatory = $true)]
  [System.IntPtr] $wParam,
  [Parameter(Mandatory = $true)]
  [System.IntPtr] $lParam
 )
    #Procedure Handling
    return CallNextHookEx([System.
        IntPtr]::Zero, $nCode, $wParam
        , $lParam)
}
```

Výpis 18: Syntaxe funkce v PowerShellu, kterou je možné zavěsit

Realizace v PowerShellu Na systémech Windows je skript funkční když jej spustí jakýkoli uživatel, není třeba administrátorského oprávnění.

Pomocí rutiny *Add-Type* je ve funkci *WindowsFunc* vytvořena nová .NET Core třída (úplný název třídy je *API.WinAPI*), která obsahuje hlavičky potřebných externích statických funkcí z Windows API. K úspěšnému běhu využívá funkce *WindowsFunc* externí knihovny *user32.dll* a *kernel32.dll*. Pomocí funkcí zmíněné třídy je do řetězu háku typu *WH_KEYBOARD_LL* zavěšena funkce *HookCallback*, která opět za použití Windows API přeloží kód virtuální klávesy na Unicode znak.

Funkce *HookCallback* není spouštěna z funkce *WindowsFunc*, ale je spouštěna v samostatném vlákně. To může způsobit kolizi v jejich souběhu při volání multiplatformní funkce *Write-This*, zejména při zapisování znaku do bufferu. Z tohoto důvodu je z metody *HookCallBack* nejprve volána metoda *LockWrite*, která zajišťuje, že metodu *Write-This* může ve stejnou chvíli volat pouze jedno vlákno. Souběh je řešen pomocí proměnné *$Semaphore*, která se chová jako semafor a může nabývat pouze hodnot *$true*, nebo *$false*. V případě hodnoty *$false* se náhodně čeká 1 až 40 milisekund, dokud semafor opět nebude obsahovat hodnotu *$true*. V případě hodnoty

$true je semafor nastaven na hodnotu *$false*, provede se funkce *Write-This* a po jejím provedení se semafor opět nastaví na hodnotu *$true*.

5.2 Ransomware

Ransomware je vyděračský software, který slouží pro šifrování dat nebo znemožnění práce se systémem. Útočník od oběti vyžaduje výkupné výměnou za klíč pro odšifrování či odblokování systému. [49]

Ransomware vytvořený v rámci práce je funkční na systémech Windows, Linux i macOS. Implementace skriptů v PowerShellu využívá pouze knihoven, které se nacházejí v rámci .NET Core. Vytvořené skripty jsou plně multiplatformní, žádná jejich část není nijak závislá na specifických součástech určitého OS. Ransomware slouží k šifrování uživatelských dat pomocí hybridního šifrování - kombinace symetrického a asymetrického šifrování.

5.2.1 Způsoby šifrování využívané v ransomwaru

Data je možné zašifrovat různými způsoby a algoritmy. Použití pouze jednoho typu šifrování však při tvorbě ransomwaru nemusí být zcela účinné. Tato podkapitola představí možné způsoby šifrování, které se zneužívají v ransomwaru.

Symetrické šifrování Je vhodné pro šifrování souborů, protože je rychlé a výpočetně nenáročné oproti asymetrickému šifrování. Nevýhoda spočívá v problému uložení klíče. Klíč musí být buď odeslán útočníkovi, nebo musí být uložen na systému oběti. V případě, že daný systém není připojen k internetu, není možné klíč útočníkovi odeslat. Při nalezení uloženého klíče na systému je oběť schopná své soubory odšifrovat, což je pro útočníka nežádoucí. Klíč není možné odeslat i v případě nedostupnosti útočníkova serveru. Použití pouze symetrického šifrování tedy není vhodné z důvodu distribuce klíče. [55]

Asymetrické šifrování ze strany oběti Je mnohem pomalejší a výpočetně náročnější oproti symetrickému šifrování. Při asymetrickém šifrování ze strany oběti vytvoří ransomware na systému oběti dvojici klíčů. Pomocí veřejného klíče jsou šifrovány soubory oběti. Problém opět nastává v distribuci privátního klíče pro dešifrování souborů, podobně jako u symetrického šifrování. [55]

Asymetrické šifrování ze strany serveru Je vytvořen veřejný a privátní klíč na serveru útočníka. Útočníkův veřejný klíč je zapsán v ransomwaru a při dosažení cílového stroje jsou tímto klíčem šifrovány soubory oběti. Systém oběti nemusí být připojen k internetu a nikam se na něj neukládá klíč pro dešifrování (útočníkův privátní klíč), protože není součástí ransomwaru. Problém tohoto způsobu je dešifrování dat. V případě napadení více strojů jsou všechny jejich soubory zašifrovány pomocí stejného veřejného klíče od útočníka. Pro dešifrování dat je nutné,

aby útočník poslal oběti svůj privátní klíč. Stejným privátním klíčem je možné dešifrovat soubory všech ostatních obětí. Problém tohoto způsobu šifrování není distribuce privátního klíče, ale použití stejného veřejného klíče pro šifrování dat na více strojích. [55]

Hybridní šifrování Kombinuje výhody všech výše popsaných metod šifrování a zároveň eliminuje jejich nevýhody. V hybridním šifrování se pro šifrování dat využívá symetrického šifrování. Pro zašifrování klíče symetrické šifry se používá asymetrické šifrování ze strany oběti. Pro zašifrování privátního klíče oběti se využívá asymetrické šifrování ze strany serveru.

Hybridní šifrování má výhodu, že pomocí veřejného klíče útočníka je zašifrován pouze privátní klíč oběti, který může zůstat uložený na systému oběti. I když je pro oběť dostupný její zašifrovaný privátní klíč, bez rozšifrování privátním klíčem útočníka jej nemůže k ničemu použít. Až oběť zjistí, že má zašifrovaná data, musí poslat útočníkovi svůj zašifrovaný privátní klíč. Útočník jej dešifruje pomocí svého privátního klíče a dešifrovaný privátní klíč oběti pošle zpět. Oběť po obdržení svého dešifrovaného privátního klíče jej může použít k dešifrování klíče symetrické šifry, pomocí kterého si dešifruje svá data. [55]

5.2.2 Realizace v PowerShellu

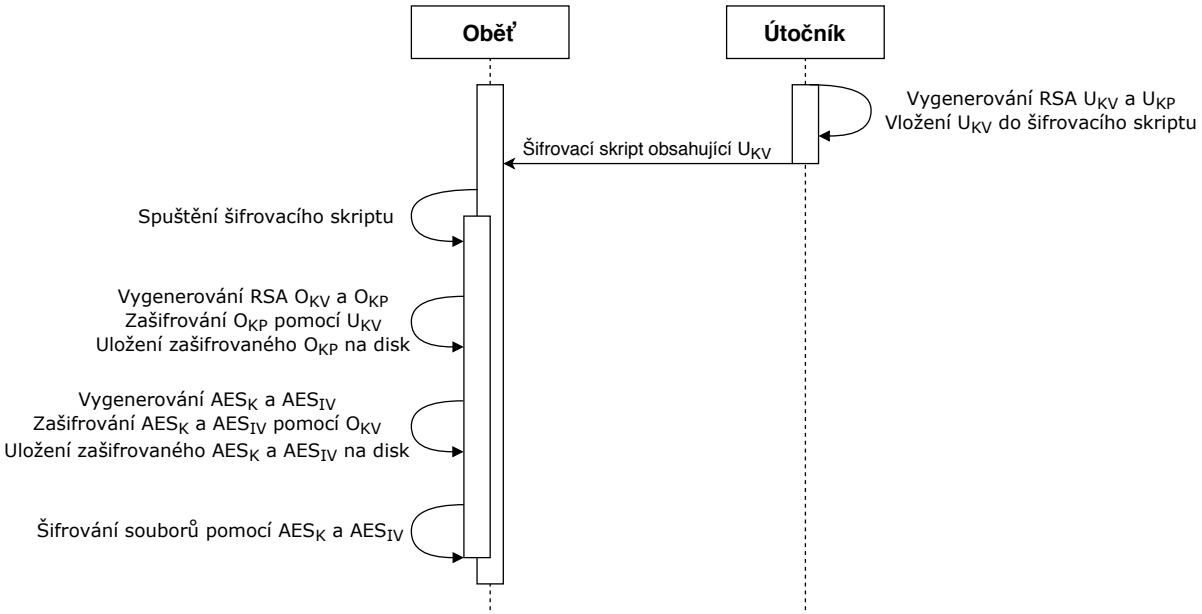
Vytvořené ransomware skripty využívají výše popsané hybridní šifrování. Použitá symetrická šifra je AES s velikostí klíče 256 bitů, velikostí bloku 128 bitů a provozním režimem řetězení šifrových bloků (*Cipher Block Chaining*). Použité asymetrické šifry jsou RSA s velikostí klíče 4096 bitů.

Procesy hybridního šifrování a dešifrování pomocí vytvořených PowerShell skriptů představují sekvenční diagramy v obrázcích č. 5 a 6. Veřejný a privátní klíč útočníka jsou označeny jako U_{KV} a U_{KP}, veřejný a privátní klíč oběti jsou označeny jako O_{KV} a O_{KP}. Označení AES_K a AES_{IV} se používá pro klíč a inicializační vektor šifry AES.

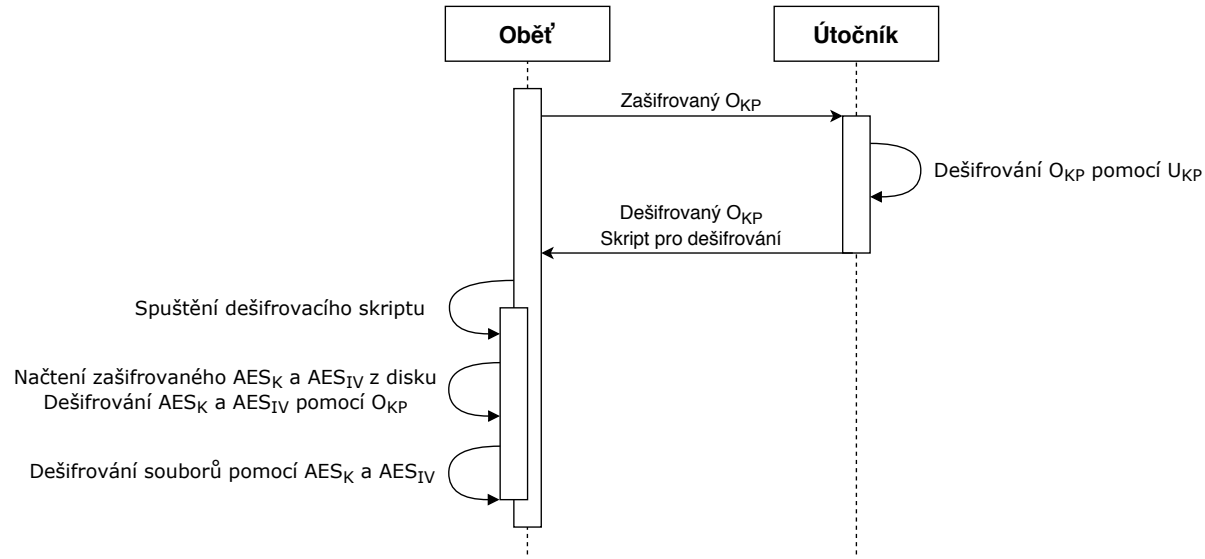
V příloze této práce se nachází vygenerované testovací klíče U_{KV} a U_{KP} v souboru "KEYS.txt" a tři skripty.

- **Ransomware_AttackerScript.ps1** Útočníkův skript. Pomocí tohoto skriptu je útočník schopen vygenerovat nové klíče U_{KV} a U_{KP} a dešifrovat zašifrovaný klíč O_{KP}.
- **Ransomware_Encrypt.ps1** Realizuje hybridní šifrování a pro svou funkčnost musí obsahovat klíč U_{KV}. Spuštění tohoto skriptu na stroji oběti způsobí vygenerování O_{KV}, O_{KP}, AES_K a AES_{IV}. V rámci skritpu proběhne šifrování O_{KP}, AES_K, AES_{IV} a šifrování souborů.
- **Ransomware_Decrypt.ps1** Realizuje hybridní dešifrování a potřebuje k běhu dešifrovaný klíč O_{KP}. V rámci skriptu proběhne dešifrování AES_K a AES_{IV}, pomocí kterých je následně provedeno dešifrování souborů.

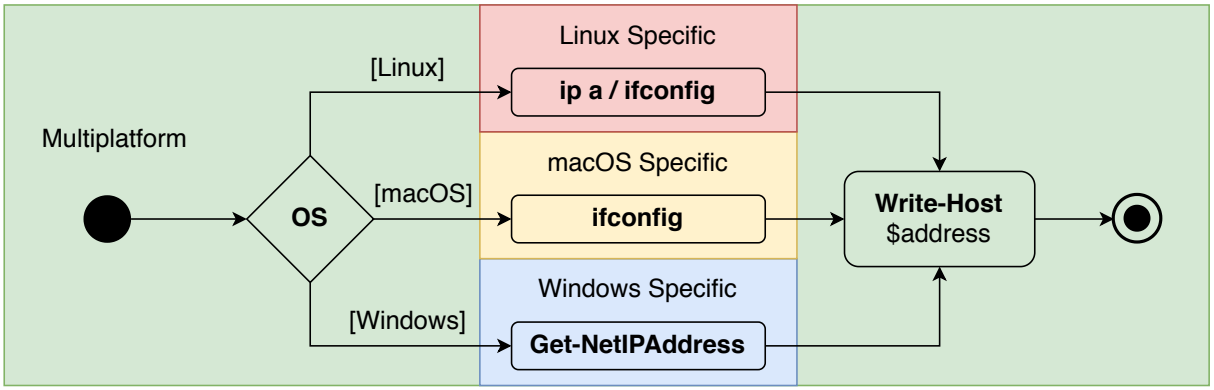
Obrázek 5: Sekvenční diagram procesu hybridního šifrování



Obrázek 6: Sekvenční diagram procesu hybridního dešifrování



Obrázek 7: Aktivitní diagram způsobu zjištění IP adres v PowerShellu na různých OS



5.3 Botnet

Botnet je síť softwarových agentů, kteří provádí zadané úlohy na základě příkazu vlastníka sítě. Botnet je možné využít pro legální účely (distribuované výpočty), ale také pro účely nelegální (*Distributed Denial of Service*, rozesílání spamu, těžba kryptoměn atd.). Vlastník sítě se nazývá *operátor botnetu* či *botmaster*. Napadený stroj oběti, na kterém na pozadí běží softwarový agent, se nazývá *zombie*, *slave*, nebo *bot* (zkratka slova robot). Hlavní stroj, ke kterému se boti připojují, se nazývá kontrolní či C&C server (z anglického Command and Control). [49]

Vytvořený botnet je funkční na systémech Windows, Linux i macOS. Jedná se o multiplatformní skripty, které z převážné části využívají knihoven .NET Core, které jsou dostupné na všech systémech. V určité části realizace je však žádoucí vypsat přiřazené IP adresy k danému stroji. K tomuto účelu neexistuje v .NET Core ani PowerShellu žádná rutina ani příkaz, které by byly na všech OS funkční. Z tohoto důvodu je nutné použít příkazy či programy, které jsou specifické pro daný systém, z nich zjistit IP adresy a následně se vrátit do multiplatformní části skriptu. Zmíněný způsob prezentuje aktivitní diagram č. 7. Na systémech Windows je pro zjištění IP adres možné použít rutinu *Get-NetIPAddress*. Na systémech Linux se používá program *ip* s parametrem *a*, na starších distribucích je možné použít program *ifconfig*. Na systémech macOS se pro zjištění IP adres používá pouze program *ifconfig*.

5.3.1 Způsob připojení botů ke kontrolnímu serveru

Připojení botů a C&C serveru je realizováno jako reverzní shell. C&C server útočníka poslouchá na daném portu (výchozí port 443) a přijímá nová spojení, která jsou iniciována boty. Připojení bota ke kontrolnímu serveru má výhodu, že bot může mít pouze privátní IP adresu. Připojení C&C serveru k botovi by mohlo bránit použití NAT (Network Address Translation) v síti bota.

V případě neúspěšného připojení ke kontrolnímu serveru se boti snaží každých 60 sekund opět připojit. V případě výpadku C&C serveru budou po jeho opětovném zprovoznění do 60 sekund všichni boti opět připojeni (za předpokladu, že se nezmění IP adresa či port serveru).

Veškerá komunikace mezi botem a C&C serverem je zabezpečená pomocí SSL. Po úspěšném připojení bota ke kontrolnímu serveru proběhne autentizace certifikátu serveru, bot svůj certifikát nemá.

5.3.2 Realizace v PowerShellu

Botnet je v PowerShellu realizován pomocí dvou skriptů. Skripty jsou funkční, když jsou spuštěny jakýmkoli uživatelem, není třeba elevovaná práva.

- **Botnet_Master.ps1** Skript je určen pro operátora (*mastera*) botnetu. Skript se chová jako kontrolní server, ke kterému se připojují jednotliví boti. Pomocí skriptu je možné hromadně ovládat vzdálené stroje a zasílat jim příkazy k vykonání.
- **Botnet_Slave.ps1** Skript je spuštěn na stroji oběti (*slave*). Po spuštění se skript chová jako agent, který se připojí na kontrolní server a dále vyčkává na instrukce a příkazy od operátora, které po přijetí vykoná.

Ze skriptu *Botnet_Master.ps1* je možné ovládat připojené boty dvěma různými způsoby - synchronně a asynchronně. Synchronní způsob ovládání odešle botům daný příkaz a čeká na odpověd v textové reprezentaci. Synchronní ovládání je vhodné, pokud útočník vyžaduje interaktivitu botů. Asynchronní ovládání slouží pouze k odeslání příkazů bez jakéhokoli čekání na odpověď. Asynchronní ovládání je vhodné pro příkazy, jejichž vykonávání trvá delší dobu, probíhají v nekonečné smyčce, či nemají žádný textový výstup. Základní implementaci synchronního a asynchronního vykonávání příkazů na straně bota představují výpisy č. 19 a 20.

```
$bytes = $sslStream.Read($buf, 0,
    $buf.Length)
$cmd = [System.Text.Encoding]::UTF8.
    GetString($buf)
$ret = Invoke-Expression $cmd |
    Out-String
$ret = [System.Text.Encoding]::UTF8.
    GetBytes($ret)
$sslStream.Write($ret)
```

Výpis 19: Bot - synchronní vykonávání příkazů

```
$bytes = $sslStream.Read($buf, 0,
    $buf.Length)
$cmd = [System.Text.Encoding]::UTF8.
    GetString($buf)
Invoke-Expression $cmd | Out-Null
```

Výpis 20: Bot - asynchronní vykonávání příkazů

Po spuštění skriptu *Botnet_Master.ps1* je vytvořena nová instance PowerShellu, která na pozadí naslouchá na daném portu a přijímá nová spojení od botů. Skript v popředí funguje jako interaktivní shell, ve kterém je možné zobrazit aktuálně připojené boty, zasílat jim příkazy a modifikovat množinu botů, kteří mají příkazy obdržet.

Po spuštění skriptu *Botnet_Slave.ps1* je vytvořená nová instance PowerShellu, která na pozadí funguje jako softwarový agent. Jelikož je agent realizován v nové instanci, která běží na pozadí, nemusí být na první pohled očividné, že je agent spuštěn. Po spuštění agenta proběhne návrat zpět do interaktivního prostředí příkazové řádky. Skript je proto vhodné ukrýt ve stroji oběti například v profilu PowerShellu.

5.3.3 Poznámky k realizaci SSL

Pro potřeby práce byla vytvořena nová certifikační autorita (CA), která byla podepsána sama sebou (*self-signed*). Pro realizaci SSL byl pomocí vytvořené CA podepsána žádost a vytvořen certifikát pro útočníkův C&C server. Certifikát pro kontrolní server byl vystaven pro subjekt *myserver.vsb.cz*. Veškeré požadavky, konverze, certifikáty a privátní klíče byly generovány pomocí nástroje *OpenSSL*. Certifikáty a privátní klíče CA i kontrolního serveru se nacházejí ve složce "CA+Certs".

Certifikát i privátní klíč pro kontrolní server je uložen v souboru "myserver.pfx", který se musí nacházet ve stejném adresáři jako skript *Botnet_Master.ps1*. Formát PFX je binární formát, který umožňuje uložení certifikátu i privátního klíče do jednoho souboru. Při spuštění skriptu *Botnet_Master.ps1* je certifikát i privátní klíč načten z tohoto souboru. Při importu PFX souboru je nutné uvést heslo souboru. Heslo je uloženo v čitelné podobě v samotném skriptu (*hardcoded password*), což při odcizení skriptu může způsobit přečtení hesla a následné odcizení privátního klíče ze souboru "myserver.pfx". Alternativou může být zadávání hesla po spuštění skriptu, aby nebylo uloženo v čitelné podobě ve skriptu.

Certifikát i privátní klíč je také možné jednorázově importovat a následně načítat ze systémového úložiště certifikátů. K tomu slouží rutiny **-Certificate*, které jsou dostupné pouze na OS Windows. Systémy Windows, Linux i macOS ukládají certifikáty každý do jiného specifického úložiště a není možné jednoduchým společným způsobem zajistit funkčnost na všech systémech. Pro jednoduchost používání a zajištění co největší možné kompatibility napříč operačními systémy a jejich distribucemi se do skriptu importuje samotný PFX soubor, který se nenachází v systémovém úložišti certifikátů.

Jelikož byl certifikát kontrolního serveru podepsán pomocí nové CA, která je podepsána sama sebou, nebude se certifikát této CA nacházet na stroji oběti a certifikát serveru nebude možné ověřit. Z tohoto důvodu se certifikát CA nachází v samotném skriptu *Botnet_Slave.ps1* v Base64 kódování. Na stroji oběti proto není třeba nijak přistupovat k systémovému úložišti certifikátů. Pomocí certifikátu CA proběhne ověření certifikátu C&C serveru. Certifikát C&C serveru je vyhodnocen jako důvěryhodný, pokud jeho vydavatel je shodný se subjektem certifikátu CA.

5.4 Testování vlastních skriptů

Podkapitola popisuje průběh testování vytvořených skriptů.

Veškeré testování probíhalo na *čistých instalacích* systémů Windows, Ubuntu a macOS bez jakýchkoli dodatečně nainstalovaných programů či AV řešení, veškerá konfigurace byla ponechána ve výchozím stavu (byl pouze doinstalován PowerShell Core). Infekce byla simulována pomocí manuálního cíleného překopírování skriptů na stroj oběti přes síť pomocí programu *scp*. V reálném scénáři by se škodlivé skripty na stroj oběti mohly dostat například pomocí metody *drive-by download*, podvržené přílohy v emailu či ukrytí v makrech Office souborů.

Systémy Linux a macOS ve výchozím stavu po instalaci neobsahují žádné AV řešení, a proto nebyl soubor nijak skenován pro přítomnost škodlivého kódu. Systémy Windows obsahují integrovaný WDAV, který po překopírování souborů na stroj oběti tyto soubory skenuje. Skenování skriptů neodhalilo přítomnost škodlivého kódu. Na všechny OS tedy proběhlo překopírování souborů bez jakýchkoli problémů a bez detekování škodlivého kódu.

Po infekci byly skripty manuálně spouštěny z různých uživatelských účtů. Skripty byly spouštěny bez elevovaných práv i s elevovanými právy. Skoro všechny skripty byly funkční i bez elevovaných práv, až na skript keyloggeru, který na systémech Linux vyžaduje spuštění s elevovanými právy pro přístup k souborům "/dev/input/eventX".

Skripty byly spouštěny pomocí výchozí aplikace PowerShellu - *pwsh*, respektive *pwsh.exe* na systému Windows. Na systému Windows bylo nutné proces spustit s parametrem *-ep Bypass* pro obejití politiky spouštění skriptů, na systémech Linux a macOS tento parametr nebyl potřeba. Všechny skripty byly bez detekce spuštěny a vykonaly svou činnost.

Různá AV řešení třetích stran by však mohly detekovat škodlivost kódu na všech platformách již při infekci. Pro skenování pomocí různých AV řešení byly vytvořené skripty nahrány na webovou stránku Virus Total (*https://www.virustotal.com/*), která umožňuje mimo jiné skenování souborů pomocí různých AV. Ani jeden ze skriptů nebyl detekován jakýmkoli z 59 dostupných AV řešení jako škodlivý či podezřelý. Podle všech dostupných AV řešení jsou tyto skripty bezpečné.

6 Výsledky a shrnutí

V této kapitole se nachází rekapitulace výsledků testování, shrnutí práce a diskuze nad možnými technikami prevence.

6.1 Výsledky testování existujících skriptů

V rámci diplomové práce bylo otestováno celkem 34 skriptů z 10 modulů tří různých frameworků pro penetrační testování PowerShellu. Cílem testování skriptů bylo zjistit, zda existující potenciálně škodlivé skripty určené pro Windows PowerShell jsou funkční i v Core edici verze 7. Všechny skripty byly nejprve otestovány ve Windows Powershell, aby byla ověřena jejich funkčnost. Pro testování byly vybrány skripty, které realizují aktivity, které by měly být dostupné na více systémech (např. zachytávání stisknutých kláves). Detaily k testování existujících skriptů se nachází v kapitole č. 4.

V Core edici bylo na systému Windows funkčních celkem 56% ze všech testovaných skriptů, na Ubuntu 32% a na macOS pouze 29% testovaných skriptů. Na systémech Linux a macOS byla nefunkčnost způsobena z velké míry tím, že testované skripty využívaly specifické nástroje a API pro Windows. Necelá polovina skriptů nebyla funkční ani na systému Windows z důvodu chybějících rutin či neúplné kompatibility .NET Frameworku a .NET Core. Skripty, které byly funkční na všech systémech využívaly pouze konstrukčních prvků PowerShellu a .NET Core knihoven, které jsou dostupné na všech systémech. Velká část skriptů určených pro Windows PowerShell tedy není funkční v PowerShellu Core.

6.2 Vlastní implementace multiplatformních skriptů

V rámci diplomové práce bylo vytvořeno celkem šest skriptů, které se dají rozdělit do tří kategorií.

Keylogger Je funkční na OS Windows a Linux. Pro systémy Linux je keylogger řešen pomocí čtení binárních dat ze souboru */dev/input/eventX* a následného rozboru těchto dat na virtuální klávesy, které jsou přeloženy na konkrétní znak. Pro systém Windows je keylogger řešen pomocí Windows API a zavěšení funkce do řetězu háku *WH_KEYBOARD_LL*, která přeloží vstupní parametry funkce na konkrétní znak. Výstup znaků je realizován pomocí multiplatformní funkce *Write-This*, která umožňuje výpis do hostitelské aplikace, uložení do souboru nebo odeslání na FTP server.

Existující implementace keyloggeru z frameworku Powersploit také používá háky, nicméně tato implementace není v Core edici na OS Windows funkční. Implementace z frameworku Nishang nepoužívá háky, ale je v edici Core na OS Windows funkční. Oproti existujícím skriptům je v rámci diplomové vytvořený skript v edici Core na systémech Windows funkční s použitím háků a jako jediný PowerShell skript je také funkční na systémech Linux.

Ransomware Je funkční na OS Windows, Linux i macOS. Implementace ransomwaru je řešena pomocí tří skriptů a hybridního šifrování. Útočníkův skript dokáže generovat a načítat RSA klíče útočníka. Pomocí privátního klíče útočníka může tento skript dešifrovat zašifrovaný privátní klíč oběti. Šifrovací skript slouží ke generování RSA klíčů oběti, šifrování souborů pomocí AES, šifrování klíče AES pomocí veřejného klíče oběti a šifrování privátního klíče oběti pomocí veřejného klíče útočníka. Skript pro dešifrování realizuje dešifrování souborů pomocí dešifrovaného AES klíče, který byl dešifrován privátním klíčem oběti.

Oproti existujícím PowerShell skriptům realizující ransomware je vytvořená implementace z této diplomové práce multiplatformní. Ostatní ransomware skripty využívají například registry systému Windows pro uložení klíče či Windows API funkce, které znemožňují běh těchto skriptů na jiných OS. Použití hybridního šifrování zaručuje, že na stroji oběti není nikde uložen klíč pro dešifrování souborů v čitelné podobě.

Botnet Je funkční na OS Windows, Linux i macOS. Botnet je realizován pomocí dvou skriptů. Jeden skript slouží jako C&C server a druhý slouží jako agent na stroji oběti. Připojení je realizováno jako reverzní shell a tudíž může bot mít k dispozici pouze privátní IP adresu. Komunikace mezi kontrolním serverem a botem je šifrována pomocí SSL. Operátor je pomocí skriptů schopen vzdáleně provádět PowerShell příkazy na připojených botech. Provádění příkazů je realizováno dvěma způsoby - synchronně a asynchronně. Při použití asynchronního provádění příkazů operátor nedostává od botů žádnou zpětnou vazbu, probíhá pouze zaslání příkazu k provedení. Při použití synchronního provádění příkazů dostává operátor od botů zpět textovou reprezentaci výsledku provedeného příkazu.

Ani jeden z vytvořených skriptů nebyl detekován jako podezřelý či škodlivý jakýmkoli AV řešením, které využívá Virus Total. Veškeré skripty byly bez jakýchkoli obtíží umístěny a spuštěny na stroj oběti. Kapitola č. 5.4 podrobněji popisuje průběh testování vytvořených skriptů.

6.3 Způsoby tvorby škodlivých multiplatformních skriptů

Při tvorbě skriptů diplomové práce byly ověřeny a použity tři různé techniky, kterými je možné realizovat multiplatformní škodlivé skripty.

Plně kompatibilní skript Využívá pouze .NET Core knihovny a konstrukční prvky PowerShellu, které se nachází ve všech systémech. Jedná se o ideální implementaci skriptu, kdy není nutné ošetřovat tok skriptu pro použití programů či částí OS, které jsou specifické pro daný systém. Skript vytvořený touto technikou má společné všechny použité příkazy a implementaci logiky při běhu na různých OS.

Příkladem použití této techniky tvorby skriptů jsou vytvořené skripty pro ransomware (viz kapitola 5.2).

Skript s využitím specifických nástrojů Využívá .NET Core knihovny a konstrukční prvky PowerShellu, které se nachází ve všech systémech. Pro specifickou funkcionalitu, kterou není možné realizovat v PowerShellu ani .NET Core, jsou použity nástroje daného systému. Výhoda této techniky spočívá ve společné multiplatformní implementaci logiky skriptu při běhu na různých OS. Nevýhodou je nutnost použití různých specifických nástrojů či příkazů, které řeší stejný problém v různých OS a následné ošetření jejich výstupů pro získání stejného formátu výsledku.

Příkladem použití této techniky tvorby skriptů jsou vytvořené skripty pro botnet (viz kapitola 5.3).

Skript realizující specifické funkce pro daný OS Obsahuje několik funkcí, kde každá z nich je implementována specifickým způsobem pro daný OS. Oproti výše uvedeným technikám spolu specifické funkce pro jednotlivé systémy nesdílí logiku, ani použité specifické příkazy či nástroje. Nevýhoda této techniky spočívá v několikanásobné implementaci stejného řešení určitého problému, pokaždé jiným specifickým způsobem. Specifické funkce však mohou volat plně kompatibilní multiplatformní funkce a tím pádem do určité míry zmenšit opakovanost kódu. Po spuštění skriptu je vždy zavolána pouze jedna ze specifických funkcí v závislosti na OS, na kterém je skript spuštěn.

Příkladem použití této techniky tvorby skriptů je vytvořený skript pro keylogger (viz kapitola 5.1).

Ideální technika tvorby multiplatformních skriptů je *plně kompatibilní skript*. Pokud je nutné použít několik specifických nástrojů či příkazů pro daný systém, vhodnou technikou je vytvořit *skript s využitím specifických nástrojů*. Součástí skriptu je zachovaná hlavní logika a převážná část kódu v multiplatformní části skriptu. Nejhorší možnou variantou je *skript realizující specifické funkce pro daný OS* z důvodu implementace několika specifických funkcí, které řeší stejný problém, ale nesdílí stejnou logiku, příkazy či nástroje.

6.4 Prevence a snížení rizika napadení

V podkapitole se nachází obecná doporučení, která můžou pomoci snížit riziko útoku pomocí PowerShell skriptu. [13, 56]

Nastavení restriktivní politiky spouštění skriptů Pomocí politiky spouštění skriptů je možné omezit spouštění skriptů. I když je politika lehce obejitelná a na systémech Linux a macOS není vůbec podporována, jedná se alespoň o základní bezpečnostní mechanismus před nechtěným spuštěním potenciálně škodlivých skriptů na OS Windows. Detaily o nastavení politik je možno nalézt v kapitole č. 3.2.

Nastavení práv profilů pouze pro čtení Do profilu PowerShellu je možné ukrýt škodlivý skript. Jelikož se profil spouští po spuštění procesu PowerShellu, spustí se s ním také škodlivý

skript. Po modifikaci profilu je vhodné omezit jeho práva pouze ke čtení, aby jej nebylo možné upravit a vložit do něj škodlivý skript. O detailech profilů pojednává kapitola č. 3.3.

Nastavení omezeného módu jazyka Uživatelům, kteří PowerShell nepoužívají nebo jej využívají minimálně je vhodné omezit mód jazyka PowerShell. Omezení módu jazyka může útočníkovi ztížit kompromitaci stroje, protože nebude mít dostupné veškeré konstrukční prvky PowerShellu. Módy jazyka PowerShell popisuje kapitola č. 3.4.

Zapnutí rozšířeného logování V případě vykonání škodlivého skrpitu je možné zaznamenat jednotlivé příkazy, které daný skript provedl. Rekonstrukce aktivity skriptu může pomoci odhalit, jakým způsobem byl PowerShell spuštěn a co přesně skript vykonal. Pomocí rozšířeného logování je možné zjistit změny systému a případnou perzistenci škodlivého kódu, pokud se skript uložil do systému oběti pro pozdější opětovné spuštění. Rozšířené možnosti logování a způsoby logování na různých systémech popisuje kapitola č. 3.7.

Omezení práv uživatele využívající vzdálený přístup V případě nutnosti vzdáleného přístupu pomocí PowerShellu je vhodné omezit práva vzdáleně připojeného uživatele na nezbytné minimum, tedy vytvořit tzv. *Constrained endpoint*. V případě kompromitace přihlašovacích údajů bude mít útočník omezená práva na daném systému. Způsoby vzdáleného přístupu popisuje kapitola č. 3.8.

Zabezpečení vzdáleného přístupu Vzdálený přístup PowerShellu je možné realizovat přes několik technologií, nejčastěji však SSH a WinRM. Pokud však na daném systému není vyžadován vzdálený přístup, je vhodné jej vůbec nezapínat. Dále je vhodné nakonfigurovat cílový systém pouze jako klient či server v závislosti na používaném směru připojení. Ve Windows doménách je vhodné používat pouze protokoly Kerberos a Negotiate. Pro zamezení odcizení přihlašovacích údajů by se neměly nikam ukládat přihlašovací údaje a používat předvyplněná hesla. Dále je možné nakonfigurovat nestandardní porty pro přenosové technologie, tedy nepoužívat výchozí porty 80 a 443 pro WinRM a 22 pro SSH.

Zakázání Windows PowerShell verze 2 Opatření je aplikovatelné pouze pro systémy Windows. I když je verze 5.1 poslední dostupná verze Windows PowerShellu, z důvodu zpětné kompatibility se na systému stále nachází Windows PowerShell verze 2.0, která již není podporována. Při spuštění skriptu je možné definovat, na které verzi Windows PowerShellu se má skript vykonat. Používání verze 2.0 je vnímáno jako bezpečnostní riziko, protože oproti verzi 5.1 neumožňuje rozšířené možnosti logování a nepoužívá AMSI pro skenování přítomnosti škodlivých řetězců a skriptů. Použití AMSI je popsáno v kapitole 3.9. Windows Powershell verze 2.0 je možné zakázat a vynutit tak spouštění příkazů a skriptů na Windows PowerShell 5.1.

Povolování určitých skritpů (*Whitelisting*) Na systému je možné povolit spouštění pouze určitých skriptů, které byly předem schváleny. Skripty se nachází na tzv. *Whitelistu*. Veškeré ostatní skripty není povoleno spouštět. Tato metoda je jedna z nejúčinnějších prevencí, protože umožňuje spouštět pouze známé a ověřené skripty. Integrita skriptu může být ověřena např. pomocí jeho hashe.

Whitelisting je lepší metoda než *Blacklisting*, která povoluje běh všech skriptů, až na výjimky, které nemají být spuštěny. Stejný škodlivý skript může jednoduše změnit svoji hash pomocí jakékoli metody obfuskace. Vybrané metody obfuskace jsou popsány v kapitole č. 3.5.

Použití dalších AV řešení I když Windows ve výchozím stavu obsahuje integrovaný Windows Defender Antivirus, instalace dalších AV může poskytnout dodatečnou ochranu. V ideálním případě by další AV řešení využívalo AMSI na systémech Windows z důvodu jednodušší spolupráce s PowerShellem. Na systémech Linux i macOS může doinstalovaný AV alespoň provádět analýzu skriptů (či souborů s příponou *.ps1*) pomocí vyhledávání podezřelých textových řetězců či hledání shod vzorů ve skriptech (*pattern matching*).

Pokud uživatel vůbec nevyužívá PowerShell, může se zdát jako nejjednodušší bezpečnostní opatření zakázat spustitelný soubor *pwsh*, případně *pwsh.exe*. Tento soubor je však pouze výchozí hostitelská aplikace, která umožňuje přístup k PowerShellu pomocí příkazové řádky. Veškerá logika engine PowerShellu se nachází v souboru *System.Management.Automation.dll*. PowerShell skripty a příkazy je tedy možné spustit i bez programu *pwsh*. [56]

Efektivní odstranění PowerShellu na systémech Linux a macOS je možné realizovat pouhou odinstalací PowerShellu a .NET Core. Na systémech Windows efektivní odstranění prakticky není možné, protože .NET Framework i PowerShell jsou integrované součásti OS Windows. Možné však je zablokovat přístup k souboru *System.Management.Automation.dll* a odebrat mu všechny práva. Tímto způsobem se sice PowerShell na systému bude dál nacházet, ale nebude spustitelný, ani nijak použitelný. [57]

7 Závěr

Dilpomová práce se zabývala bezpečností a zneužitelností nástroje PowerShell pro tvorbu multiplatformních škodlivých skriptů. První kapitola se zabývala přehledem aktuálních trendů vzhledem ke zneužití PowerShellu pro bezsouborové a Living off the Land útoky. Analyzovaný 1000% nárůst ve výskytech škodlivých PowerShell skriptů v roce 2018 oproti roku 2017 a 460% nárůst výskytů v prvním čtvrtletí roku 2019 oproti roku 2018 značí aktuálnost problematiky.

V další kapitole byl popsán PowerShell z technické stránky. V této kapitole byly mimo jiné přiblíženy některé bezpečnostní mechanismy PowerShellu, které je možné použít pro snížení rizika útoků zneužívající PowerShell skripty. Dále zde byly ukázány vybrané metody obfuskace, často používané parametry a často se vyskytující příkazy ve škodlivých skriptech.

V rámci práce byly otestovány vybrané existující skripty z frameworků pro penetrační testování Windows PowerShellu na PowerShell Core verze 7 v operačních systémech Windows, Ubuntu a macOS. Z výsledků testování je zřejmé, že většina testovaných skriptů není na Ubuntu ani macOS funkční z důvodu použití Windows API, registrů, či jiných nástrojů, které jsou specifické pro systém Windows. Na systému Windows nefungovala v Core edici méně než polovina testovaných skriptů z důvodu chybějících rutin, metod či knihoven, což bylo zřejmě způsobeno neúplnou kompatibilitou mezi .NET Frameworkem a .NET Core.

Dále bylo v rámci praktické části práce vypracováno a otestováno celkem šest multiplatformních škodlivých skriptů, které se dají rozdělit do tří kategorií - keylogger, ransomware a botnet. Skript keyloggeru slouží pro zachytávání stisknutých kláves na kompromitovaném stroji oběti a je funkční na systémech Windows a Linux. Skripty pro ransomware implementují vyděračský software, který využívá hybridního šifrování na systémech Windows, Linux i macOS. Skripty pro botnet prezentují možnost zneužití PowerShellu pro realizaci multiplatformního botnetu využívající šifrovanou komunikaci pomocí SSL na operačních systémech Windows, Linux i macOS.

Pro tvorbu nových skriptů byly použity tři různé techniky tvorby multiplatformních skriptů. Ideální technikou je plně kompatibilní skript, který je funkční bez jakýchkoli úprav v plném znění na všech podporovaných operačních systémech. V případě, že je nutné použít některé specifické nástroje systému, ale je možné zachovat multiplatformní logiku skriptu, je vhodné vytvořit skript s využitím specifických nástrojů. Po volání systémového specifického nástroje je předán jeho výstup do multiplatformní části skriptu. Pokud není mezi jednotlivými systémy možné sdílet logiku skriptu ani použité příkazy či nástroje, existuje možnost implementace specifických funkcí pro daný OS, kdy na začátku skriptu je zavolána jedna ze specifických funkcí v závislosti na systému, na kterém je skript spuštěn.

V poslední kapitole se nachází doporučení pro prevenci a snížení rizika útoku zneužívající PowerShell skripty. I když nebyl doposud detekován žádný multiplatformní útok zneužívající PowerShell, je pravděpodobné, že se v budoucnu takový útok objeví.

Literatura

1. MANSFIELD-DEVINE, Steve. Fileless attacks: compromising targets without malware. *Network Security* [online]. 2017, roč. 2017, č. 4, s. 7–11 [cit. 2020-02-13]. Dostupné z: www.scopus.com.
2. BALDIN, Andy. Best practices for fighting the fileless threat. *Network Security* [online]. 2019, roč. 2019, č. 9, s. 13–15 [cit. 2020-02-13]. Dostupné z: www.scopus.com.
3. COOPER, Stephen. *Fileless malware attacks explained (with examples)* [online]. Comparitech, 2018-06-11 [cit. 2020-02-13]. Dostupné z: https://www.comparitech.com/blog/information-security/fileless-malware-attacks/.
4. *The Rise of Fileless Malware and Attack Techniques* [online]. All Answers Ltd., 2019-06-06 [cit. 2020-02-22]. Dostupné z: https://ukdiss.com/examples/fileless-malware-attack-techniques.php.
5. WUEES, Candid; ANAND, Himanshu. *Living off the land and fileless attack techniques: An ISTR Special Report* [online]. Symantec Corporation, 2017-07 [cit. 2020-02-18]. Dostupné z: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-living-off-the-land-and-fileless-attack-techniques-en.pdf.
6. KAZANCIYAN, Ryan; HASTINGS, Matt. *Investigating PowerShell Attacks* [online]. Black Hat, 2014 [cit. 2019-10-14]. Dostupné z: https://www.blackhat.com/docs/us-14/materials/us-14-Kazanciyan-Investigating-Powershell-Attacks-WP.pdf.
7. *The Increased Use of PowerShell in Attacks* [online]. Symantec Corporation, 2016. Verze 1.0 [cit. 2019-10-15]. Dostupné z: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf.
8. KENNEDY, David; O'GORMAN, Jim; KEARNS, Devon; AHARONI, Mati. *Metasploit: The Penetration Tester's Guide*. No Starch Press, 2011. ISBN 978-1593272883.
9. KLEIN, Ondřej. *Zneužití Powershell skriptu pro potřeby počítačové kriminality* [online]. Ostrava, 2018 [cit. 2019-10-14]. Dostupné z: http://hdl.handle.net/10084/128647. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava.
10. *Internet Security Threat Report* [online]. Symantec Corporation, 2019 [cit. 2019-10-15]. Dostupné z: https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf.
11. *McAfee Labs Threats Report: August 2019* [online]. McAfee Labs, 2019 [cit. 2020-02-14]. Dostupné z: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf.

12. *2020 State of Malware Report* [online]. MalwareBytes Labs, 2020 [cit. 2020-02-08]. Dostupné z: https://resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report.pdf.

13. *Securing PowerShell in the Enterprise* [online]. Australian Cyber Security Centre, 2019 [cit. 2019-10-15]. Dostupné z: https://www.cyber.gov.au/sites/default/files/2019-03/Securing_PowerShell.pdf.

14. LI, Zhenyuan; CHEN, Yan; CHEN, Qi Alfred; ZHU, Tiantian; XIONG, Chunlin; YANG, Hai. Effective and light-weight deobfuscation and semantic-aware attack detection for powershell scripts. In: *Proceedings of the ACM Conference on Computer and Communications Security* [online]. 2019, s. 1831–1847 [cit. 2020-02-13]. Dostupné z: www.scopus.com.

15. *PowerShell Scripting: PowerShell* [online]. Microsoft Docs, 2018-08-27 [cit. 2020-01-17]. Dostupné z: https://docs.microsoft.com/cs-cz/powershell/scripting/overview.

16. SNOVER, Jeffrey P. *Monad Manifesto* [online]. 2002-08-08. Verze 1.2 [cit. 2020-01-20]. Dostupné z: https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Components.PostAttachments/00/01/91/05/67/Monad%20Manifesto%20-%20Public.doc.

17. *Windows PowerShell (Monad) Has Arrived* [online]. Microsoft Developer Blogs, 2006-04-25 [cit. 2020-01-20]. Dostupné z: https://devblogs.microsoft.com/powershell/windows-powershell-monad-has-arrived/.

18. *PowerShell: Wikipedia, The Free Encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-02-02]. Dostupné z: https://en.wikipedia.org/wiki/PowerShell.

19. AIELLO, Joey. *PowerShell Core 6.0: Generally Available (GA) and Supported!* [online]. Microsoft Developer Blogs, 2018-01-10 [cit. 2020-02-02]. Dostupné z: https://devblogs.microsoft.com/powershell/powershell-core-6-0-generally-available-ga-and-supported/.

20. *PowerShell for every system!: PowerShell/PowerShell* [online]. GitHub [cit. 2020-03-05]. Dostupné z: https://github.com/PowerShell/PowerShell.

21. *Home Repository for .NET Core: dotnet/core* [online]. GitHub [cit. 2020-02-06]. Dostupné z: https://github.com/dotnet/core.

22. AIELLO, Joey. *Announcing PowerShell 7.0* [online]. Microsoft Developer Blogs, 2020-03-04 [cit. 2020-03-05]. Dostupné z: https://devblogs.microsoft.com/powershell/announcing-powershell-7-0/.

23. NEVES, David das; PETERS, Jan-Hendrik. *Learn PowerShell Core 6.0*. Packt Publishing, 2018. ISBN 978-1788838986.

24. POSEY, Brian. *PowerShell Core vs. PowerShell: What are the differences?* [online]. TechGenix, 2019-03-19 [cit. 2020-02-07]. Dostupné z: http://techgenix.com/powershell-core/.

25. MEHTA, Siddharth. *Python vs PowerShell for System and SQL Server Administration* [online]. MS SQL Tips, 2019-06-04 [cit. 2020-02-07]. Dostupné z: https://www.mssqltips.com/sqlservertip/6043/python-vs-powershell-for-system-and-sql-server-administration/.

26. VÁCHAL, Libor. *Mobilní správa Windows serveru vprostředí KIV* [online]. Plzeň, 2016 [cit. 2020-02-10]. Dostupné z: http://hdl.handle.net/11025/23691. Diplomová práce. Západočeská univerzita v Plzni.

27. BLAWAT, Brenton J.W. *Mastering Windows PowerShell Scripting*. Packt Publishing, 2015. ISBN 978-1782173557.

28. HOLMES, Lee. *Windows PowerShell CookBook: The Complete Guide to Scripting Microsoft's Command Shell*. O'Reilly Media, Inc., 2013. ISBN 978-1449320683.

29. CHAND, Mahesh. *Difference Between .NET Framework and .NET Core* [online]. C# Corner, 2020-02-04 [cit. 2020-02-07]. Dostupné z: https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/.

30. LEE, Steve. *The Next Release of PowerShell – PowerShell 7* [online]. Microsoft Developer Blogs, 2019-04-19 [cit. 2020-02-16]. Dostupné z: https://devblogs.microsoft.com/powershell/the-next-release-of-powershell-powershell-7/.

31. PIETROFORTE, Michael. *Differences between PowerShell versions* [online]. 4sysops [cit. 2020-02-10]. Dostupné z: https://4sysops.com/wiki/differences-between-powershell-versions/.

32. *Fileless Malware Execution with PowerShell Is Easier than You May Realize: Technical Brief* [online]. McAfee Labs, 2017 [cit. 2020-02-10]. Dostupné z: https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-fileless-malware-execution.pdf.

33. BOHANNON, Daniel. *PowerShell Obfuscator: danielbohannon/Invoke-Obfuscation* [online]. GitHub [cit. 2020-02-25]. Dostupné z: https://github.com/danielbohannon/Invoke-Obfuscation.

34. HENDLER, Danny; KELS, Shay; RUBIN, Amir. Detecting malicious powershell commands using deep neural networks. In: *ASIACCS 2018 - Proceedings of the 2018 ACM Asia Conference on Computer and Communications Security* [online]. 2018, s. 187–197 [cit. 2020-02-25]. Dostupné z: www.scopus.com.

35. ROUSSEAU, Amanda. *Hijacking .NET to Defend PowerShell* [online]. EndGame Inc., 2017 [cit. 2020-02-25]. Dostupné z: https://arxiv.org/abs/1709.07508.

36. MITTAL, Nikhil. *AMSI: How Windows 10 Plans To Stop Script-Based Attacks and How Well Does It* [online]. Black Hat, 2016 [cit. 2020-02-26]. Dostupné z: https://www.blackhat.com/docs/us-16/materials/us-16-Mittal-AMSI-How-Windows-10-Plans-To-Stop-Script-Based-Attacks-And-How-Well-It-Does-It.pdf.

37. PETERS, Jan-Hendrik. *Powershell Core 6.2 Cookbook*. Packt Publishing, 2019. ISBN 978-1789803303.

38. PIETROFORTE, Michael. *Enable PowerShell remoting* [online]. 4sysops, 2018 [cit. 2020-03-05]. Dostupné z: https://4sysops.com/wiki/enable-powershell-remoting/.

39. PIETROFORTE, Michael. *Enable PowerShell Core 6 remoting with SSH transport* [online]. 4sysops, 2018 [cit. 2020-03-05]. Dostupné z: https://4sysops.com/archives/enable-powershell-core-6-remoting-with-ssh-transport/.

40. *Antimalware Scan Interface (AMSI): Win32 apps* [online]. Microsoft Docs, 2019-04-19 [cit. 2020-03-01]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal.

41. LIBERMAN, Tal. *The Rise and Fall of AMSI* [online]. BlackHat, 2018 [cit. 2020-03-01]. Dostupné z: https://i.blackhat.com/briefings/asia/2018/asia-18-Tal-Liberman-Documenting-the-Undocumented-The-Rise-and-Fall-of-AMSI.pdf.

42. DONALDSON, Ashley. *Detecting AMSI Bypass* [online]. Ionize, 2020-01-28 [cit. 2020-03-01]. Dostupné z: https://ionize.com.au/detecting-amsi-bypass/.

43. *PowerSploit* [online] [cit. 2020-03-07]. Dostupné z: https://powersploit.readthedocs.io/en/latest/.

44. *PowerSploit - A PowerShell Post-Exploitation Framework: PowerShellMafia/PowerSploit* [online]. GitHub [cit. 2020-02-14]. Dostupné z: https://github.com/PowerShellMafia/PowerSploit.

45. *PowerShell Empire: Building an Empire with PowerShell* [online] [cit. 2020-03-07]. Dostupné z: https://www.powershellempire.com/.

46. *Empire is a PowerShell and Python post-exploitation agent. EmpireProject/Empire* [online]. GitHub [cit. 2020-03-07]. Dostupné z: https://github.com/EmpireProject/Empire.

47. MITTAL, Nikhil. *AMSI: How Windows 10 Plans to Stop Script-Based Attacks and How Well It Does It* [online]. Lab of a Penetration Tester, 2016-12-23 [cit. 2020-03-07]. Dostupné z: http://www.labofapenetrationtester.com/2016/09/amsi.html.

48. *Nishang - Offensive PowerShell for red team, penetration testing and offensive security. samratashok/nishang* [online]. GitHub [cit. 2020-03-07]. Dostupné z: https://github.com/samratashok/nishang.

49. KOLOUCH, Jan. *CyberCrime*. CZ.NIC, 2016. ISBN 978-8088168188.

50. *Keyboard Input* [online]. ArchWiki [cit. 2020-03-17]. Dostupné z: https://wiki.archlinux.org/index.php/Keyboard_input.

51. 153ARMSTRONG. *Exploring /dev/input* [online]. The Hacker Diary, 2017-04-21 [cit. 2020-03-15]. Dostupné z: https://thehackerdiary.wordpress.com/2017/04/21/exploring-devinput-1/.

52. PAVLÍK, Vojtěch. *Linux Input drivers v1.0* [online]. The Linux Kernel Archives, 2001 [cit. 2020-03-15]. Dostupné z: https://www.kernel.org/doc/Documentation/input/input.txt.

53. *Linux kernel source tree: torvalds/linux* [online]. GitHub [cit. 2020-03-15]. Dostupné z: https://github.com/torvalds/linux.

54. *Hooks: Win32 apps* [online]. Microsoft Docs, 2018-05-31 [cit. 2020-03-23]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/winmsg/hooks.

55. MARINHO, Tarcísio. *Ransomware encryption techniques* [online]. Medium, 2018-08-30 [cit. 2020-03-24]. Dostupné z: https://medium.com/@tarcisioma/ransomware-encryption-techniques-696531d07bb9.

56. MONOGIOUDIS, Isidoros. *PowerShell Security Best Practises* [online]. Digital Shadows, 2019-10-08 [cit. 2020-04-06]. Dostupné z: https://www.digitalshadows.com/blog-and-research/powershell-security-best-practices/.

57. *Microsoft recommended block rules (Windows 10): Windows Security* [online]. Microsoft Docs [cit. 2020-04-06]. Dostupné z: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-block-rules.

A Příloha v IS EDISON

Součástí práce je elektronická příloha v zip souboru. Zde se nachází vytvořené skripty a příslušné soubory.

- **Botnet/Botnet_Master.ps1** Skript pro operátora botnetu, C&C server.
- **Botnet/Botnet_Slave.ps1** Skript pro oběť botnetu.
- **Botnet/myserver.pfx** Soubor obsahující certifikát a zašifrovaný privátní klíč pro C&C server.
- **Botnet/CA+Certs/myCA.pem** Certifikát CA.
- **Botnet/CA+Certs/myCA.srl** Soubor obsahující sériové číslo certifikátu C&C serveru.
- **Botnet/CA+Certs/myCApriv.pem** Soubor obsahující zašifrovaný privátní klíč CA.
- **Botnet/CA+Certs/myserver.crt** Certifikát pro C&C server.
- **Botnet/CA+Certs/myserverpriv.pem** Soubor obsahující zašifrovaný privátní klíč pro C&C server.
- **Botnet/CA+Certs/PASSWORD.txt** Textový soubor obsahující heslo pro privátní klíče CA a C&C serveru.
- **KeyLogger/KeyLogger.ps1** Skript keyloggeru.
- **Ransomware/KEYS.txt** Textový soubor obsahující útočníkův veřejný a privátní klíč.
- **Ransomware/Ransomware_AttackerScript.ps1** Útočníkův skript.
- **Ransomware/Ransomware_Decrypt.ps1** Skript pro hybridní dešifrování souborů.
- **Ransomware/Ransomware_Encrypt.ps1** Skript pro hybridní šifrování souborů.