СИГНАТУРНИЙ АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО КОДУ В СИСТЕМАХ ЗАХИСТУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ МЕРЕЖ

Каспрівський Дмитро
Здобувач вищої освіти
Львівський державний університет безпеки життєдіяльності

Полотай Орест
к.т.н., доцент, науковий керівник
Львівський державний університет безпеки життєдіяльності

Постійне зростання кількості та різноманітності шкідливого програмного забезпечення створює суттєві ризики для безпеки інформаційно-комунікаційних систем і мереж та зумовлює необхідність своєчасного виявлення шкідливих програм і їх варіантів [1]. Одним із традиційних підходів до виявлення відомих загроз є сигнатурний аналіз, що передбачає пошук у файлах характерних ознак шкідливого програмного коду, зокрема певних послідовностей байтів, рядків або інших особливостей, за якими може бути ідентифікований конкретний зразок чи сімейство шкідливого програмного забезпечення [2]. Водночас ефективність сигнатурного підходу обмежується можливістю модифікації програмного коду, використання пакування, шифрування, поліморфізму та інших методів обфускації, які дають змогу змінювати зовнішні характеристики шкідливого файлу без суттєвої зміни його функціональності [2; 3]. Тому дослідження особливостей сигнатурного аналізу та його можливостей щодо виявлення модифікованого й обфускованого шкідливого програмного коду є актуальним завданням для підвищення ефективності систем захисту інформаційно-комунікаційних мереж.

Виявлення шкідливого програмного забезпечення є важливою складовою сучасного захисту інформації. Оскільки кіберзагрози постійно змінюються та вдосконалюються, системи безпеки мають не лише розпізнавати вже відомі шкідливі програми, а й вміти реагувати на нові та ще невідомі їхні версії. Саме тому сучасні антивірусні програми використовують одразу декілька методів перевірки, що дає змогу забезпечити більш надійний захист.

Одним із найпоширеніших методів є сигнатурний аналіз. Його суть полягає у порівнянні файлу, який перевіряється, з інформацією про вже відомі шкідливі програми. Такий підхід добре працює під час виявлення поширених і раніше досліджених загроз та зазвичай забезпечує невелику кількість помилкових спрацювань. Водночас сигнатурний аналіз має певні обмеження. Наприклад, він не завжди може виявити нову шкідливу програму або її модифіковану версію, яка раніше не була внесена до антивірусної бази.

Основою цього підходу є сигнатура шкідливої програми – своєрідний цифровий «відбиток» файлу, який формується на основі його вмісту та

характерної послідовності байтів. Завдяки такій сигнатурі антивірус може визначити, чи відповідає досліджуваний файл конкретному відомому зразку шкідливого програмного забезпечення [4].

Сукупність таких сигнатур зберігається у спеціальній вірусній базі. Під час перевірки пристрою антивірус порівнює файли з інформацією, що міститься в цій базі. Якщо виявлено відповідність, файл може бути визначений як шкідливий. Основною проблемою такого методу є те, що навіть незначна зміна коду або структури шкідливої програми може призвести до зміни її сигнатури. У результаті антивірус може не розпізнати модифікований файл до того моменту, поки відповідна нова сигнатура не буде додана до вірусної бази [5].

В таблиці 1 показано порівняння сигнатурного аналізу з іншими видами аналізів шкідливого програмного забезпечення.

Таблиця 1

Порівняння методів виявлення шкідливого програмного забезпечення

Метод виявлення	Принцип роботи	Переваги	Недоліки
Сигнатурний	Порівняння з базами сигнатур	Висока точність якщо загрози відомі	Нових загроз не виявляє
Евристичний	Аналіз структур коду, які є підозрілими	Виявлення нових модифікацій структур коду	Можливі помилкові виявлення
Поведінковий	Моніторинг дій програмного забезпечення	Виявлення zero-day атак	Високе навантаження

На практиці сигнатурний аналіз полягає в тому, що програмний код, який перевіряється, порівнюється з відомими ознаками шкідливих програм. Такими ознаками можуть бути хеш файлу, характерні текстові фрагменти, певні послідовності байтів, звернення до системних API, підключені бібліотеки та інші особливості структури виконуваного файлу.

Варто зазначити, що використання лише хешу добре підходить для пошуку конкретного, незміненого файлу. Якщо навіть трохи змінити його вміст, хеш також зміниться, і система вже не зможе знайти початковий збіг. Тому для виявлення змінених версій шкідливих програм використовують сигнатури, які враховують не одну характеристику, а характерні фрагменти коду або сукупність різних ознак.

Таким чином, під час створення сигнатур для систем захисту інформаційно-комунікаційних мереж важливо правильно вибрати ознаки шкідливого програмного забезпечення. Вони повинні залишатися достатньо стабільними навіть після внесення змін до коду, але водночас бути достатньо специфічними, щоб система не реагувала на звичайні, безпечні файли.

У дослідженні як тестовий зразок використано PowerShell-код зворотної мережевої оболонки (reverse shell), сформований за допомогою генератора Reverse Shell Generator. Reverse Shell Generator Такий код є характерним

прикладом підозрілої PowerShell-активності, оскільки поєднує встановлення мережевого TCP-з'єднання, передавання даних через мережевий потік та виконання команд, отриманих від віддаленої сторони. Подібна конструкція reverse shell також наведена в рекомендаціях Australian Cyber Security Centre як приклад PowerShell-коду, здатного встановлювати з'єднання із заданою IP-адресою та портом, отримувати команди й передавати результати їх виконання назад через мережевий потік [5].

```
powershell -nop -c "$client = New-Object
System.Net.Sockets.TCPClient('10.10.10.10',9001);$stream = $client.GetStream();
[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -
ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0,
$i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' +
(pwd).Path + '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Le
ngth);$stream.Flush()};$client.Close()"
```

Рисунок 1 – досліджуваний код [6]

У досліджуваному зразку можна виділити декілька характерних ознак, потенційно придатних для сигнатурного виявлення. До них належать використання класу System.Net.Sockets.TCPClient для створення TCP-з'єднання, виклик методу GetStream() для роботи з мережевим потоком, формування масиву байтів для приймання даних, використання System.Text.ASCIIEncoding для перетворення отриманих даних у текст та конструкція Invoke-Expression (iex) для виконання отриманого рядка як команди. Остання ознака є особливо суттєвою з погляду аналізу безпеки, оскільки Invoke-Expression безпосередньо виконує переданий йому рядок як PowerShell-команду; Microsoft окремо застерігає від використання цього механізму з неперевіреними даними через можливість виконання довільних команд [7].

Аналіз структури досліджуваного PowerShell-зразка дає змогу виділити низку ознак, які можуть бути використані для формування сигнатури. До найбільш характерних належать звернення до класу System.Net.Sockets.TCPClient, використання методу GetStream() для організації мережевого обміну, операції Read() та Write() з мережевим потоком, а також застосування Invoke-Expression (iex) для виконання отриманих команд. Сукупність цих ознак є більш інформативною для сигнатурного аналізу, ніж окремий рядок або хеш файлу, оскільки відображає функціональні особливості досліджуваного коду. Використання PowerShell для виконання команд розглядається в MITRE ATT&CK як окрема техніка T1059.001, причому серед способів її виявлення зазначаються підозрілі ланцюжки виконання PowerShell, зокрема поєднання команд із мережевими з'єднаннями [8].

В таблиці 2 показано ознаки досліджуваного PowerShell-коду.

Таблиця 2

Характерні ознаки досліджуваного PowerShell-коду

№	Елемент коду	Функціональне призначення	Значення для сигнатурного аналізу
1	powershell	Запуск командного інтерпретатора PowerShell	Дає змогу визначити використання PowerShell як середовища виконання
2	System.Net.Sockets.TCPClient	Створення TCP-клієнта для встановлення мережевого з'єднання	Висока інформативність у поєднанні з іншими мережевими ознаками
3	10.10.10.10, 9001	IP-адреса та порт віддаленого вузла	Можуть використовуватися як конкретні IOC, але легко змінюються
4	$client.GetStream()	Отримання мережевого потоку для обміну даними	Характерна ознака реалізації мережевої взаємодії
5	$stream.Read()	Отримання даних із мережевого потоку	У поєднанні з Write() може вказувати на двосторонній обмін
6	$stream.Write()	Передавання даних через мережевий потік	Додаткова ознака реалізації віддаленого керування
7	System.Text.ASCIIEncoding	Перетворення отриманих байтових даних у текст	Допоміжна ознака, сама по собі малоспецифічна
8	iex / Invoke-Expression	Виконання отриманого рядка як команди PowerShell	Висока інформативність для виявлення підозрілої поведінки
9	[byte[]]$bytes	Створення буфера для приймання даних	Допоміжна структурна ознака
10	while (...) + Read()	Циклічне отримання команд	У поєднанні з мережевими операціями характеризує механізм reverse shell
11	pwd	Отримання поточного робочого каталогу	Допоміжна ознака формування відповіді віддаленому вузлу

На основі визначених характерних ознак досліджуваного зразка для практичної реалізації сигнатурного виявлення можуть бути сформовані правила YARA різного рівня складності (таблиця 3). Перший варіант може ґрунтуватися на пошуку однієї характерної ознаки, наприклад рядка System.Net.Sockets.TCPClient. Такий підхід є простим у реалізації, проте його недоліком є можливість хибних спрацювань, оскільки окрема ознака не обов’язково є достатньою для однозначної ідентифікації шкідливого коду.

Більш надійним підходом є формування комбінованої сигнатури, яка передбачає одночасну наявність декількох характерних ознак. Для

досліджуваного зразка до такого набору можуть бути включені System.Net.Sockets.TCPClient, GetStream, Invoke-Expression, stream.Read та stream.Write. Умовою спрацювання правила може бути наявність декількох із зазначених ознак, що дозволяє зменшити залежність результату від окремого елемента програмного коду.

Окремий варіант може передбачати використання набору ознак, пов'язаних не з конкретною IP-адресою або портом, а з функціональною структурою коду. Такий підхід є доцільнішим для протидії простій модифікації зразка, оскільки IP-адреса та номер порту можуть бути змінені без зміни основного механізму роботи reverse shell. Таким чином, під час побудови сигнатур доцільно надавати перевагу сукупності характерних структурних і функціональних ознак, а не окремим змінним параметрам.

Практична реалізація зазначених правил YARA дозволила б порівняти їхню здатність виявляти початковий та модифікований варіанти програмного коду. Зокрема, можна дослідити, які ознаки зберігаються після зміни структури або обфускації коду та наскільки це впливає на результат сигнатурного виявлення. У межах цієї роботи наведені правила розглядаються як запропонований підхід до побудови сигнатур, тоді як їх програмна реалізація та експериментальна перевірка можуть бути предметом подальших досліджень.

Таблиця 3

Характерні ознаки досліджуваного PowerShell-коду

Варіант правила	Основний принцип	Приклади ознак	Очікувана особливість
YARA-1	Одна характерна ознака	TCPClient	Простота, але можливі хибні спрацювання
YARA-2	Комбінація декількох ознак	TCPClient + GetStream + Invoke-Expression	Вища специфічність
YARA-3	Сукупність функціональних ознак	мережевий обмін + читання/запис + виконання команд	Потенційно більша стійкість до простої модифікації

Таким чином, сигнатурний аналіз залишається важливим компонентом систем захисту інформаційно-комунікаційних систем і мереж, оскільки дає змогу оперативно виявляти відомі зразки та характерні ознаки шкідливого програмного забезпечення. На прикладі PowerShell reverse shell визначено набір структурних і функціональних ознак, які можуть використовуватися для формування сигнатур, зокрема мережевих компонентів, операцій читання та запису даних і механізмів виконання отриманих команд. Показано доцільність використання комбінованих сигнатур YARA, які враховують сукупність ознак, а не окремий елемент програмного коду. Застосування такого підходу в системах захисту ІКС та мереж може підвищити оперативність виявлення відомих загроз і забезпечити додатковий рівень контролю мережевої активності. Водночас

модифікація та обфускація програмного коду залишаються чинниками, що обмежують ефективність суто сигнатурного підходу, що визначає доцільність його поєднання з іншими методами аналізу.

Список літератури

1. ENISA. ENISA Threat Landscape 2025. European Union Agency for Cybersecurity, 2025. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025

2. A. S. et al. Malware classification and composition analysis: A survey of recent developments. Journal of Information Security and Applications. 2021. Vol. 59. 102828. DOI: 10.1016/j.jisa.2021.102828

3. Brezinski K., Ferens K. Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits. Security and Communication Networks. 2023. Vol. 2023. Article 8227751. DOI: 10.1155/2023/8227751

4. Гавриш, Б. М., Тимченко, О. В., Борзов, Ю. О., & Кобевко, А. Т. (2022). Класифікація шкідливого програмного забезпечення та основні методи захисту. Комп'ютерні технології друкарства, 2(48), 142–154.

5. Iqbal, S., & Zulkernine, M. (2018). SpyDroid: A framework for employing multiple real-time malware detectors on Android. 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), 1–8. https://doi.org/10.1109/MALWARE.2018.8659365

6. Reverse Shell Generator. URL: https://www.revshells.com/

7. Microsoft. Invoke-Expression (Microsoft.PowerShell.Utility). Microsoft Learn. URL: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-7.5

8. Australian Cyber Security Centre (ACSC). Advisory 2020-008: Copy-paste compromises – tactics, techniques and procedures used to target multiple Australian networks. Australian Signals Directorate, 2020. URL: https://www.cyber.gov.au/about-us/advisories/advisory-2020-008-copy-paste-compromises