INFORMATION SECURITY RISK MANAGEMENT

DOI 10.20535/2411-1031.2021.9.2.249915
УДК 004[942::(056.53+413.4)]

ВАСИЛЬ ЦУРКАН,
ДМИТРО ВОЛОШИН

ФУНКЦІЙНА МОДЕЛЬ РЕВЕРС-ІНЖИНІРИНГУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Досліджено процес реверс-інжинірингу шкідливого програмного забезпечення. Показано його орієнтованість на розкриття принципів функціонування апаратного та програмного забезпечення. Насамперед його структури, алгоритмів. Водночас ацентовано увагу на перетворенні бінарних інструкцій протягом реверс-інжинірингу на мнемоніки коду для встановлення впливу як на апаратне, так і програмне забезпечення. З огляду на це, проаналізовано відповідні методи. Зокрема, дослідження апаратних троянських програм на основі опорних векторів. При цьому встановлено застосовність результатів використання реверс-інжинірингу для навчання запропонованої моделі виявлення апаратних троянських програм. Водночас розглянуто важливість класифікування шкідливого програмного забезпечення, встановлення особливостей його впливу на комп'ютерні системи та мережі. До того ж проаналізовано проблему захисту від програм-вимагачів. Як наслідок, з'ясовано, що характерною особливістю проаналізованих досліджень є багатоаспектність і, як наслідок, неформалізованість реверс-інжинірингу шкідливого програмного забезпечення. Це призводить до різноманітності інтерпретувань функцій у межах даної діяльності. Для запобігання даному обмеженню запропоновано використання графічної нотації IDEF0. Додатковою перевагою такого вибору є її формалізованість. Завдяки цьому розроблено функційну модель реверс-інжинірингу шкідливого програмного забезпеченя. За основу її побудови взято граф IDEF0. Це дозволило формалізувати дану діяльність виокремленням функцій верхнього та нижніх рівнів (створення контрольованого середовища, вивчення поведінки шкідливого програмного забезпечення, дослідження протоколів зв'язку, аналізування коду шкідливого програмного забезпечення, створення сигнатур шкідливого програмного забезпечення). Кожну з них представлено з визначенням вхідних, вихідних даних, обмежень, ресурсів і встановленням залежностей між ними.

Ключові слова: шкідливе програмне забезпечення, функційна модель, граф IDEF0, контекстна діаграма, діаграма декомпозиції.

Постановка проблеми. Під реверс-інжинірингом розуміється діяльність, що направлена на розкриття принципів функціонування апаратного чи програмного забезпечення. Насамперед його структури, алгоритмів. Його можна використовувати у багатьох галузях [1], зокрема, й стосовно шкідливого програмного забезпечення. Серед таких завдань виокремлюється, наприклад [2]:

- вивчення мережевих протоколів зв'язку;
- пошук алгоритмів використання шкідливого програмного забезпечення, зокрема, комп'ютерних вірусів, троянів, програм-вимагачів;
- вивчення формату файлу, який використовується для зберігання даних, наприклад, баз даних електронної пошти та образів дисків.

Протягом реверс-інжинірингу шкідливого програмного забезпечення бінарні інструкції перетворюються на мнемоніки коду для встановлення алгоритму його використання і, зокрема, впливу як на апаратне, так і програмне забезпечення. Отже, розроблення функційної моделі реверс-інжинірингу шкідливого програмного забезпечення є актуальним завданням.

Аналіз останніх досліджень і публікацій. Дослідження апаратних троянських програм на основі реверс-інжинірингу виконано в [3]. За його основу взято метод опорних векторів. При цьому результати використання реверс-інжинірингу застосовуються для навчання запропонованої моделі виявлення апаратних троянських програм. Застосування зворотної інженерії стосовно протидії проявам ботнету запропоновано в [4]. При цьому акцентовано увагу на важливості класифікування шкідливого програмного забезпечення, встановлення особливостей його впливу на комп'ютерні системи та мережі. Тому реверс-інжиніринг обирається як один з ефективних способів запобігання їхнім проявам. Виявлення шкідливого програмного забезпечення на основі реверс-інжинірингу здійснено в [5]. Його використання направлено на встановлення алгоритмів реалізування кіберзагроз. Цим визначається результативність протидії їхнім проявам. Проблему захисту комп'ютерних систем і мереж від програм-вимагачів описано у [6]. Насамперед розкрито обмеження типових способів захисту від них. Як альтернативу розглянуто використання реверс-інжинірингу. Забезпечення безпеки програмного забезпечення як один з актуальних напрямів досліджень розкрито в [7]. При цьому основну увагу зосереджено на застосуванні методів реверс-інжинірингу шкідливого програмного забезпечення. Зокрема, стосовно виявлення і аналізування уразливостей, які ним використовуються. Дослідженню шкідливого програмного забезпечення для мобільних операційних систем приділено увагу в [8]. Це досягнуто завдяки реверс-інжинірингу Java-коду на прикладі операційної системи Android. Характеристики шкідливого програмного забезпечення встановлювалися на основі зібраного набору програм, зокрема, й зі шкідливими діями. Водночас потоки даних мобільних програмних застосунків на основі файлів маніфесту проаналізовано в [9]. Їх використання направлене перш за все на виявлення незадекларованих компонентів. Окрему увагу приділено застосуванню точок інтересів [10]. Вони орієнтовані на встановлення "маяків", за якими можливе аналізування шкідливого програмного забезпечення. Додатково вводяться метрики результативності точок довіри як оцінку впевненості. Характерною особливістю проаналізованих досліджень є багатоаспектність і, як наслідок, неформалізованість діяльності реверс-інжинірингу шкідливого програмного забезпечення. Це призводить до різноманітності інтерпретувань функцій у її межах [11].

Серед способів представлення функцій реверс інжинірингу шкідливого програмного забезпечення виокремлюються такі [11, 12]: функцій, IDEF0; робіт IDEF3, BPMN, ARIS; даних, DFD. З огляду на особливості даної діяльності [12], обрано спосіб представлення функцій у графічній нотації IDEF0 [11, 12]. Додатковою перевагою такого використання є її формалізованість [13].

Метою статті є формалізування діяльності реверс-інжинірингу шкідливого програмного забезпечення.

Виклад основного матеріалу дослідження. Діяльність реверс-інжинірингу шкідливого програмного забезпечення формалізується шляхом використання графу IDEF0 [13, 15]:

$$G = (V_{G_{IDEF0}}, E_{G_{IDEF0}}), \quad (1)$$

де $V_{G_{IDEF0}}$ – множина блоків $B_{G_{IDEF0}}$, сегментів стрілок $S_{G_{IDEF0}}$

$$V_{G_{IDEF0}} = \bigcup\{B_{G_{IDEF0}}, S_{G_{IDEF0}}),$$

$B_{G_{IDEF0}} = \{b_0\}$, b_0 – реверс-інжиніринг шкідливого програмного забезпечення як функція верхнього рівня; кожен сегмент s відображається стрілкою, $s \in S_{G_{IDEF0}}$;

$E_{G_{IDEF0}}$ – множина дуг

$$E_{G_{IDEF0}} = \bigcup\{I_{G_{IDEF0}}, C_{G_{IDEF0}}, O_{G_{IDEF0}}, M_{G_{IDEF0}}),$$

$I_{G_{IDEF0}}$ (англ. "**I**nput", укр. "Вхід"), $C_{G_{IDEF0}}$ (англ. "**C**ontrol", укр. "Керування"), $M_{G_{IDEF0}}$ (англ. "**M**echanism", укр. "Механізм") – підмножини множини

$$S_{G_{IDEF0}} \times B_{G_{IDEF0}}$$

способів з'єднань сегменту вхідної стрілки з блоком;

$O_{G_{IDEF0}}$ (англ. "**O**utput", укр. "Вихід") – підмножина множини

$$B_{G_{IDEF0}} \times S_{G_{IDEF0}}$$

способів з'єднань блоку зі сегментом вихідної стрілки;

для блоку $b_0 \in B_{G_{IDEF0}}$ існує:

- одна вхідна стрілка $s_1 \in S_{G_{IDEF0}}$, $(s_1, b_0) \in I_{G_{IDEF0}}$ (шкідливе програмне забезпечення);
- одна вихідна стрілка $s_2 \in S_G$, $(b_0, s_2) \in O_{G_{IDEF0}}$ (список сигнатур шкідливого програмного забезпечення);
- дві стрілки керування $s_3, s_4 \in S_{G_{IDEF0}}$, $(s_3, b_0), (s_4, b_0) \in C_{G_{IDEF0}}$ (можливості апаратного забезпечення, ознаки сигнатур шкідливого програмного забезпечення);
- шість стрілок механізмів $s_5, s_6, s_7, s_8, s_9, s_{10} \in S_{G_{IDEF0}}$, $(s_5, b_0), (s_6, b_0), (s_7, b_0), (s_8, b_0), (s_9, b_0), (s_{10}, b_0) \in M_{G_{IDEF0}}$ (програмне забезпечення віртуалізації, фахівець з кібербезпеки, PEiD, Process monitor, Wireshark, Ghidra);

блок $b_0 \in B_{G_{IDEF0}}$ має унікальний номер $\varphi_0(b_0) = \varphi(b_0) = 0$.

Відповідно до (1) діяльність реверс-інжинірингу шкідливого програмного забезпечення представляється контекстною діаграмою з визначеними метою і поглядом (див. рис. 2.1)

$$D_{context} = \{G_{IDEF0},\ \varphi_0(b_0),\ \varphi(b_0)\}, \quad (2)$$

де $D_{context}$ – контекстна діаграма;

$\varphi_0(b_0)$ – номер вузла;

$\varphi(b_0)$ – номер блоку.

З огляду на рис. 2.1, на вході функції верхнього рівня – реверс-інжинірингу шкідливого програмного забезпечення може бути виконуваний файл. Наприклад [14], PE-файл (англ. "Portable Executable", укр. "Переносний виконуваний") – формат виконуваних файлів, об'єктного коду і динамічних бібліотек, що використовується в 32- і 64-розрядних версіях операційної системи Microsoft Windows. Формат PE є структурою даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'яті. Виконуваний код включає посилання для зв'язування бібліотек, що динамічно завантажуються, таблиці експорту та імпорту API-функцій, дані для керування ресурсами і дані локальної пам'яті потоку. В операційних системах сімейства Microsoft Windows формат PE використовується для EXE, DLL, SYS (драйверів пристроїв) та інших типів виконуваних файлів. Ще одним типом виконуваних файлів є ELF (англ. "Executable and Linkable Format", укр. "Формат файлів, що виконуються і зв'язуються") [15]. Ним визначається структура бінарних файлів, бібліотек, і файлів ядра. Специфікація формату дозволяє операційній системі коректно інтерпретувати машинні команди, що містяться у файлі. ELF-файл зазвичай є вихідним файлом компілятора або лінкера і має двійковий формат. Крім того, шкідливі дії програмного забезпечення можуть приховуватися у інших типах файлу. Насамперед тих, що широко використовуються при роботі за комп'ютером, наприклад, *.doc, *.xlsx, *.txt, *.pdf.

Діяльність реверс-інжинірингу визначається з огляду на можливості апаратного забезпечення і ознаки сигнатур шкідливого програмного забезпечення. Тоді як механізмами є фахівець з кібербезпеки, програмне забезпечення віртуалізації, інструментальне програмне забезпечення. Наприклад, PEiD – для знаходження загальної інформації. При обробленні шкідливого програмного забезпечення відслідковуються процеси за допомогою Process monitor, взаємодія з операційною системою та мережею – Wireshark. До того ж для роботи з кодом – Ghidra.

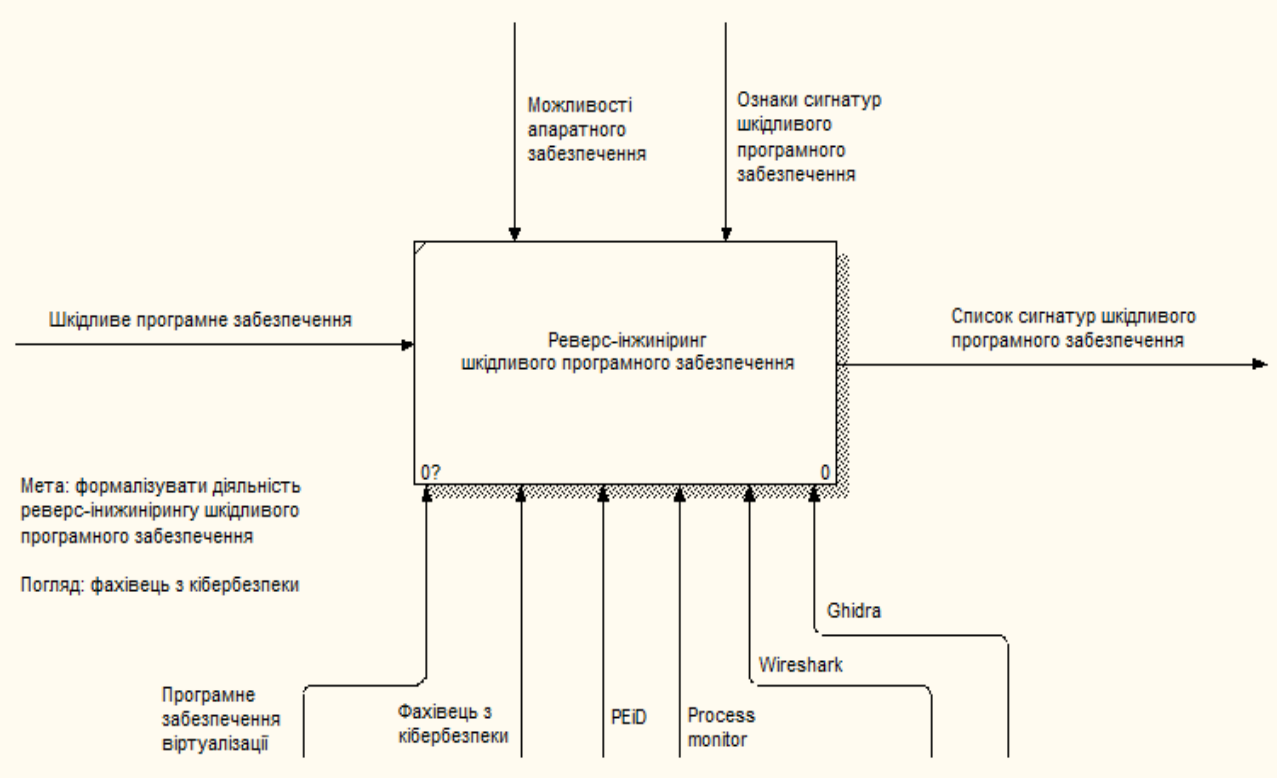

Рисунок 2.1 – Контекстна діаграма діяльності реверс-інжинірингу шкідливого програмного забезпечення

Декомпозиція контекстної діаграми діяльності реверс-інжинірингу шкідливого програмного забезпечення здійснюється виокремленням функцій нижніх рівнів [11, 12]. Відповідно до (1) враховується виконання таких умов:

$B_{G_{IDEF0}} = \{ b_0, \{b_1, b_2, b_3, b_4, b_5\}\}$, b_1 – створення контрольованого середовища, b_2 – вивчення поведінки шкідливого програмного забезпечення, b_3 – дослідження протоколів зв'язку, b_4 – аналізування коду шкідливого програмного забезпечення, b_5 – створення сигнатур шкідливого програмного забезпечення;

для блоку $b_1 \in B_{G_{IDEF0}}$ існує:

- одна вхідна стрілка $s_1 \in S_{G_{IDEF0}}$, $(s_1, b_1) \in I_{G_{IDEF0}}$ (шкідливе програмне забезпечення);
- одна вихідна стрілка $s_{11} \in S_G$, $(b_1, s_{11}) \in O_{G_{IDEF0}}$ (можливості віртуальної машини);
- одна стрілка управління $s_3 \in S_{G_{IDEF0}}$, $(s_3, b_1) \in C_{G_{IDEF0}}$ (можливості апаратного забезпечення);
- дві стрілки механізмів $s_5, s_6 \in S_{G_{IDEF0}}$, $(s_5, b_1), (s_6, b_1) \in M_{G_{IDEF0}}$ (програмне забезпечення віртуалізації, фахівець з кібербезпеки);

блок $b_1 \in B_{G_{IDEF0}}$ має унікальний номер вузла та блоку $\varphi_0(b_1) = \varphi(b_1) = 1$;

для блоку $b_2 \in B_{G_{IDEF0}}$ існує:

- одна вхідна стрілка $s_1 \in S_{G_{IDEF0}}$, $(s_1, b_2) \in I_{G_{IDEF0}}$ (шкідливе програмне забезпечення);
- одна вихідна стрілка $s_{12} \in S_G$, $(b_2, s_{12}) \in O_{G_{IDEF0}}$ (список запущених процесів в операційній системі);

– одна стрілка управління $s_{11} \in S_{G_{IDEF0}}$, $(s_{11}, b_2) \in C_{G_{IDEF0}}$ (можливості віртуальної машини);
– дві стрілки механізмів $s_5, s_6 \in S_{G_{IDEF0}}$, $(s_5, b_2), (s_6, b_2) \in M_{G_{IDEF0}}$ (програмне забезпечення віртуалізації, фахівець з кібербезпеки);

блок $b_2 \in B_{G_{IDEF0}}$ має унікальний номер $\varphi_0(b_2) = \varphi(b_2) = 2$;

для блоку $b_3 \in B_{G_{IDEF0}}$ існує:

– одна вхідна стрілка $s_1 \in S_{G_{IDEF0}}$, $(s_1, b_3) \in I_{G_{IDEF0}}$ (шкідливе програмне забезпечення);
– одна вихідна стрілка $s_{13} \in S_G$, $(b_3, s_{13}) \in O_{G_{IDEF0}}$ (список запущених процесів у мережі);
– одна стрілка управління $s_{11} \in S_{G_{IDEF0}}$, $(s_{11}, b_3) \in C_{G_{IDEF0}}$ (можливості віртуальної машини);
– дві стрілки механізмів $s_5, s_6 \in S_{G_{IDEF0}}$, $(s_5, b_3), (s_6, b_3) \in M_{G_{IDEF0}}$ (програмне забезпечення віртуалізації, фахівець з кібербезпеки);

блок $b_3 \in B_{G_{IDEF0}}$ має унікальний номер $\varphi_0(b_3) = \varphi(b_3) = 3$;

для блоку $b_4 \in B_{G_{IDEF0}}$ існує:

– три вхідних стрілки $s_1, s_{12}, s_{13} \in S_{G_{IDEF0}}$, $(s_1, b_4), (s_{12}, b_4), (s_{13}, b_4) \in I_{G_{IDEF0}}$ (шкідливе програмне забезпечення, список запущених процесів в операційні системі, список запущених процесів у мережі);
– одна вихідна стрілка $s_{14} \in S_G$, $(b_4, s_{14}) \in O_{G_{IDEF0}}$ (список виділених сигнатур);
– дві стрілки управління $s_4, s_{11} \in S_{G_{IDEF0}}$, $(s_4, b_4), (s_{11}, b_4) \in C_{G_{IDEF0}}$ (ознаки сигнатур шкідливого програмного забезпечення, можливості віртуальної машини);
– шість стрілок механізмів $s_5, s_6, s_7, s_8, s_9, s_{10} \in S_{G_{IDEF0}}$, $(s_5, b_4), (s_6, b_4), (s_7, b_4), (s_8, b_4), (s_9, b_4), (s_{10}, b_4) \in M_{G_{IDEF0}}$ (програмне забезпечення віртуалізації, фахівець з кібербезпеки, PEiD, Process monitor, Wireshark, Ghidra);

блок $b_4 \in B_{G_{IDEF0}}$ має унікальний номер $\varphi_0(b_4) = \varphi(b_4) = 4$;

для блоку $b_5 \in B_{G_{IDEF0}}$ існує:

– одна вхідна стрілка $s_{14} \in S_{G_{IDEF0}}$, $(s_{14}, b_5) \in I_{G_{IDEF0}}$ (список виділених сигнатур);
– одна вихідна стрілка $s_2 \in S_{G_{IDEF0}}$, $(b_5, s_2) \in O_{G_{IDEF0}}$ (список сигнатур шкідливого програмного забезпечення);
– одна стрілка управління $s_4 \in S_{G_{IDEF0}}$, $(s_4, b_5) \in C_{G_{IDEF0}}$ (ознаки сигнатур шкідливого програмного забезпечення);
– одна стрілка механізмів $s_6 \in S_{G_{IDEF0}}$, $(s_6, b_5) \in M_{G_{IDEF0}}$ (фахівець з кібербезпеки);

блок $b_5 \in B_{G_{IDEF0}}$ має унікальний номер $\varphi_0(b_5) = \varphi(b_5) = 5$;

Завдяки цьому отримаємо декомпозицію контекстної діаграми діяльності реверс-інжинірингу шкідливого програмного забезпечення функціями нижніх рівнів (див. рис. 2.2). Завдяки їх виокремленню та з урахуванням (1), (2) утворюється діаграма декомпозиції такого виду

$$D_{noncontext} = \{G_{IDEF0},\ \varphi_n(b_m),\ \varphi(b_m)\},$$

де $D_{noncontext}$ – діаграма декомпозиції;

$\varphi_n(b_m)$ – номер вузла;

$\varphi(b_m)$ – номер блоку.

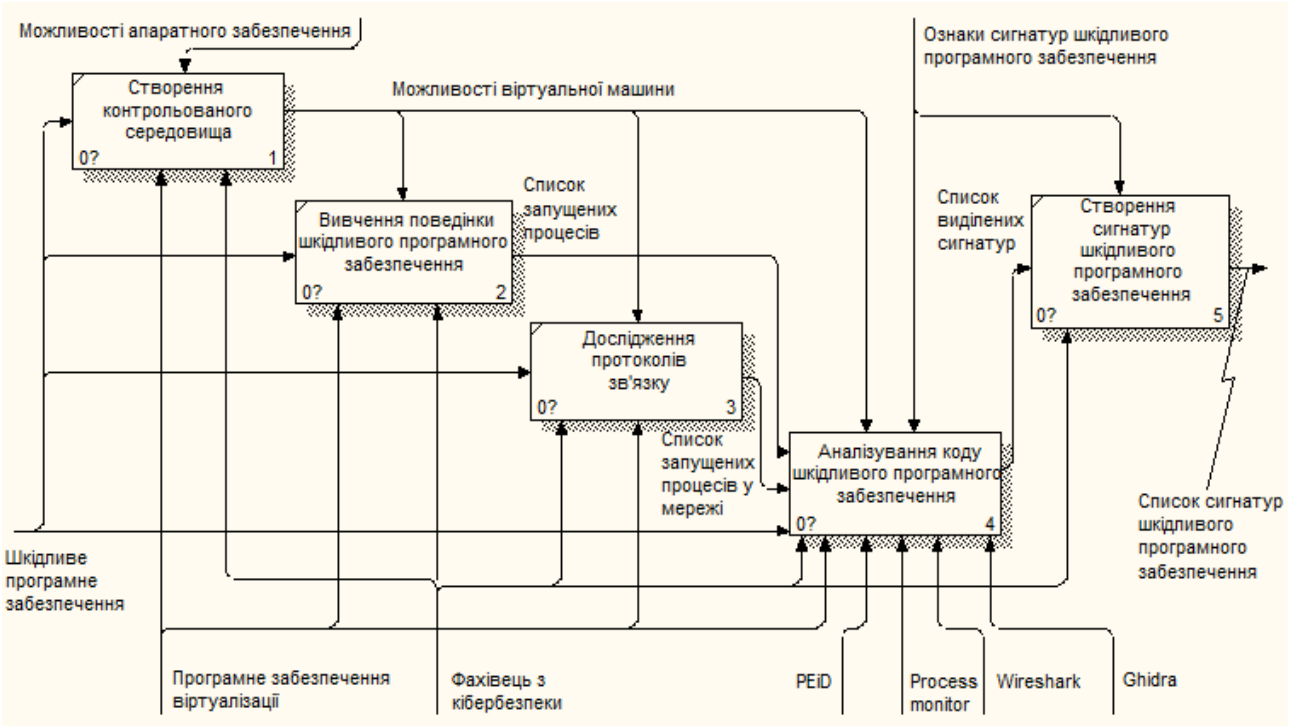

Рисунок 2.2 – Діаграма декомпозиції діяльності реверс-інжинірингу шкідливого програмного забезпечення

Висновки. Встановлено особливості реверс-інжинірингу шкідливого програмного забезпечення. Виокремлено відповідні інструментальні засоби, зокрема, дизассемблери, відлагоджувачі, програми перегляду, мережні аналізатори. Крім того, зіставлено методи реверс-інжинірингу шкідливого програмного забезпечення. Показано багатоаспектність їх застосування і, як наслідок, неформалізованість означеної діяльності. Для подолання даного обмеження запропоновано використання графічної нотації IDEF0.

Розроблено функційну модель реверс-інжинірингу шкідливого програмного забезпечення на основі графу IDEF0, використання якої дозволило формалізувати дану діяльність виокремленням функцій верхнього та нижніх рівнів (створення контрольованого середовища, вивчення поведінки шкідливого програмного забезпечення, дослідження протоколів зв'язку, аналізування коду шкідливого програмного забезпечення, створення сигнатур шкідливого програмного забезпечення). Кожну з них представлено з визначенням вхідних, вихідних даних, обмежень, ресурсів і встановленням залежностей між ними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Abigail A. Reverse Engineering Research. [Online]. Available: http://dx.doi.org/10.13140/ RG.2.2.28030.51520. Accessed on: Aug. 30, 2021.

[2] APRIORIT: How to Reverse Engineer Software (Windows) the Right Way? [Online]. Available: https://www.apriorit.com/dev-blog/364-how-to-reverse-engineer-software-windows- in-a-right-way. Accessed on: Aug. 30, 2021.

[3] G. Jain, S. Raghuwanshi, and G. Vishwakarma, "Hardware Trojan: Malware Detection Using Reverse Engineering and SVM", in *Intelligent Systems Design and Applications. ISDA 2017. Advances in Intelligent Systems and Computing*, A. Abraham, P. Muhuri, A. Muda, and N. Gandhi (eds), Vol. 736, 2018, pp. 530-539, doi: https://doi.org/10.1007/ 978-3-319-76348-4_51.

[4] B. Thakar, C. Parekh, "Reverse Engineering of Botnet (APT)", in *Information and Communication Technology for Intelligent Systems (ICTIS 2017). Vol. 2. ICTIS 2017. Smart Innovation, Systems and Technologies*, S. Satapathy, and A. Joshi (eds), Vol 84, 2018, pp. 252-262, doi: https://doi.org/10.1007/978-3-319-63645-0_28.

[5] S. Megira et al., "Malware Analysis and Detection Using Reverse Engineering Technique", *Journal of Physics: Conference Series 1140*, pp. 1-13, 2018, doi: https://doi.org/10.1088/ 1742-6596/1140/1/012042.

[6] S. Naveen, and T. Kumar Gireesh, "Ransomware Analysis Using Reverse Engineering". *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, and R. Kashyap (eds), Vol. 1046, 2019, pp. 185-194, doi: https://doi.org/10.1007/978-981-13-9942-8_18.

[7] Z. Chen, B. Pan, and Y. Sun, "A Survey of Software Reverse Engineering Applications". *Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science*, X. Sun, Z. Pan, and E. Bertino (eds), Vol. 11635, 2019, pp. 235-245, doi: https://doi.org/ 10.1007/978-3-030-24268-8_22.

[8] M. Kedziora, P. Gawin, M. Szczepanik, and I. Jozwiak, "Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code". *International Journal of Network Security & Its Applications*, Vol. 11, No.1, pp. 1-14, January 2019, doi: https://dx.doi.org/10.2139/ssrn.3328497.

[9] G. Sharma, M. Mabrishi, K. Hiran, and R. Doshi, "Reverse Engineering for potential Malware detection. Android APK Smali to Java", *Journal of Information Assurance & Security*, Vol. 15, Iss. 1, pp. 26–34, 2020.

[10] A. See, M. Gehring, M. Mühlhäuser, M. Fischer, and S. Karuppayah, Malware Sight-Seeing : Accelerating Reverse-Engineering via Point-of-Interest-Beacons. [Online]. Available: https://arxiv.org/abs/2109.04065. Accessed on: Aug. 30, 2021.

[11] D. Voloshin, "Functional approach to reverse engineering malware. *Інформаційні технології та безпека*, на XXI Міжнародній науково-практичній конференції, Київ, 2021, с. 230-231.

[12] В. Цуркан, "Метод функціонального аналізування систем управління інформаційною безпекою", *Кібербезпека: освіта, наука, техніка,* Том 4, № 8, с. 192-201, 2020, doi: https://doi.org/10.28925/2663-4023.2020.8.192201.

[13] International Organization for Standardization. (2012, Sept. 15). *ISO/IEC/IEEE 31320-1:2012. Information technology. Modeling Languages. Part 1: Syntax and Semantics for IDEF0*. Geneva, 2012, 120 p.

[14] PE Format. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/debug/ pe-format. Accessed on: Aug. 30, 2021.

[15] Executable and Linkable Format (ELF). [Online]. Available: http://www.skyfree.org/linux/ references/ELF_Format.pdf. Accessed on: Aug. 30, 2021.

Стаття надійшла до редакції 02.09.2021.

REFERENCE

[1] Abigail A. Reverse Engineering Research. [Online]. Available: http://dx.doi.org/10.13140/ RG.2.2.28030.51520. Accessed on: Aug. 30, 2021.

[2] APRIORIT: How to Reverse Engineer Software (Windows) the Right Way? [Online]. Available: https://www.apriorit.com/dev-blog/364-how-to-reverse-engineer-software-windows- in-a-right-way. Accessed on: Aug. 30, 2021.

[3] G. Jain, S. Raghuwanshi, and G. Vishwakarma, "Hardware Trojan: Malware Detection Using Reverse Engineering and SVM", in *Intelligent Systems Design and Applications. ISDA 2017. Advances in Intelligent Systems and Computing*, A. Abraham, P. Muhuri, A. Muda, and N. Gandhi (eds), Vol. 736, 2018, pp. 530-539, doi: https://doi.org/10.1007/ 978-3-319-76348-4_51.

[4] B. Thakar, C. Parekh, "Reverse Engineering of Botnet (APT)", in *Information and Communication Technology for Intelligent Systems (ICTIS 2017). Vol. 2. ICTIS 2017. Smart Innovation, Systems and Technologies*, S. Satapathy, and A. Joshi (eds), Vol 84, 2018, pp. 252-262, doi: https://doi.org/10.1007/978-3-319-63645-0_28.

[5] S. Megira et al., "Malware Analysis and Detection Using Reverse Engineering Technique", *Journal of Physics: Conference Series 1140*, pp. 1-13, 2018, doi: https://doi.org/10.1088/1742-6596/1140/1/012042.

[6] S. Naveen, and T. Kumar Gireesh, "Ransomware Analysis Using Reverse Engineering". *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, and R. Kashyap (eds), Vol. 1046, 2019, pp. 185-194, doi: https://doi.org/10.1007/978-981-13-9942-8_18.

[7] Z. Chen, B. Pan, and Y. Sun, "A Survey of Software Reverse Engineering Applications". *Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science*, X. Sun, Z. Pan, and E. Bertino (eds), Vol. 11635, 2019, pp. 235-245, doi: https://doi.org/10.1007/978-3-030-24268-8_22.

[8] M. Kedziora, P. Gawin, M. Szczepanik, and I. Jozwiak, "Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code". *International Journal of Network Security & Its Applications*, Vol. 11, No.1, pp. 1-14, January 2019, doi: https://dx.doi.org/10.2139/ssrn.3328497.

[9] G. Sharma, M. Mabrishi, K. Hiran, and R. Doshi, "Reverse Engineering for potential Malware detection. Android APK Smali to Java", *Journal of Information Assurance & Security*, Vol. 15, Iss. 1, pp. 26–34, 2020.

[10] A. See, M. Gehring, M. Mühlhäuser, M. Fischer, and S. Karuppayah, Malware Sight-Seeing : Accelerating Reverse-Engineering via Point-of-Interest-Beacons. [Online]. Available: https://arxiv.org/abs/2109.04065. Accessed on: Aug. 30, 2021.

[11] D. Voloshin, "Functional approach to reverse engineering malware. In *Proc. XXI International Scientific and Practical Conference Information Technology and Security*, Kyiv, 2021, pp. 230-231.

[12] V. Tsurkan, "Method of information security management systems functional analysis", *Cybersecurity: Education, Science, Technique*, Vol. 4, Iss. 8, pp. 192-201, 2020, doi: https://doi.org/10.28925/2663-4023.2020.8.192201.

[13] International Organization for Standardization. (2012, Sept. 15). *ISO/IEC/IEEE 31320-1:2012. Information technology. Modeling Languages. Part 1: Syntax and Semantics for IDEF0*. Geneva, 2012, 120 p.

[14] PE Format. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/debug/ pe-format. Accessed on: Aug. 30, 2021.

[15] Executable and Linkable Format (ELF). [Online]. Available: http://www.skyfree.org/linux/references/ELF_Format.pdf. Accessed on: Aug. 30, 2021.

VASYL TSURKAN,
DMITRY VOLOSHIN

FUNCTIONAL MODEL OF REVERSE ENGINEERING MALWARE

The process of reverse engineering malicious software is studied. Its focus on revealing the principles of hardware or software operation is shown. First of all, its structure, algorithms. At the same time, attention is focused on the transformation of binary instructions during reverse engineering into code mnemonics to establish the impact on both hardware and software. With this in mind, the relevant methods are analyzed. In particular, the study of hardware Trojans based on reference vectors. The applicability of reverse engineering to train the proposed hardware Trojan detection model has been established. At the same time, the importance of classifying malware,

identifying its features, and affecting computer systems and networks is discussed. In addition, the problem of protection against extortionist programs is analyzed. As a result, it was found that the characteristic feature of the analyzed research is the multifaceted nature and, as a consequence, the informal nature of reverse engineering malicious software. This leads to a variety of interpretations of functions within the activity. To avoid this limitation, the use of graphic notation IDEF0 is proposed. An additional advantage of this choice is its formality. Due to this, a functional model of reverse engineering malicious software has been developed. It is based on the graph IDEF0. This allowed formalizing this activity by separating the functions of the upper and lower levels (creating a controlled environment, studying the behavior of malicious software, researching communication protocols, analyzing malicious code, creating malicious signatures). Present each of them with the definition of input, output data, constraints, resources, and establish relationships between them.

Keywords: malware, function model, IDEF0 graph, context diagram, decomposition diagram.

Цуркан Василь Васильович, кандидат технічних наук, доцент, доцент кафедри кібербезпеки і застосування інформаційних систем і технологій, Інститут спеціального зв'язку та захисту інформації Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна, ORCID 0000-0003-1352-042X, v.v.tsurkan@gmail.com.

Волошин Дмитро Віталійович, магістрант, Інститут спеціального зв'язку та захисту інформації Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна, ORCID 0000-0002-4600-2816, cost.filatoff@gmail.com.

Tsurkan Vasyl, candidate of technical sciences, associate professor, associate professor at the cybersecurity and application of information systems and technologies academic department, Institute of special communication and information protection of National technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.

Voloshin Dmitry, master's student, Institute of special communication and information protection of National technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.