УДК 004.056.57

Д.Г. Буханов, Д.В. Сулохин

Выявление вредоносного программного обеспечения на основе классификации графов потоков исходных кодов

Произведен анализ существующих подходов к выявлению вредоносного программного обеспечения, определены их недостатки. Предложен подход классификации программного обеспечения в компьютерных системах обработки информации. Предложенный подход предполагает три шага: построение графа потока выполнения исходного кода приложения, векторизацию полученного графа, классификацию графа на основе использования искусственной нейронной сети адаптивно-резонансной теории. Векторизация графа происходит на основе модели bags-of-words. Полученные результаты свидетельствуют о возможности использования предложенного подхода в задачах определения вредоносного программного обеспечения.

Ключевые слова: шаблон, выявление вредоносного программного обеспечения, граф потока выполнения, нейронная сеть, классификация данных.

doi: 10.21293/1818-0442-2018-21-3-30-34

Обнаружение вредоносного программного обеспечения (ПО) является актуальной проблемой в области обеспечения информационной безопасности. На текущий момент известно два класса методов обнаружения вредоносного ПО: статический анализ и динамический анализ [1]. Для статического анализа производят поиск сигнатуры вредоносной программы в машинном коде исполняемого файла [2]. При динамическом анализе прибегают к использованию виртуальных машин для эмуляции работы процесса в виртуальном окружении и последующего анализа его поведения в системе [3].

Как статический, так и динамический анализ исполняемых файлов имеют свои недостатки. Для защиты от статического анализа создатели вредоносного ПО прибегают к методам обфускации [4] и самомодификации исходного кода [5], его упаковки и шифрования [6]. В случае динамического анализа существенным недостатком является то, что вредоносная программа может нанести вред только после прямого воздействия пользователя на систему, а до этого времени выдает себя за безопасное ПО.

Существует подход к анализу поведения процесса с помощью выделения графа потока выполнения (Control-Flow Graph, CFG) его исполняемого кода. В работе [7] описан алгоритм построения CFG для приложения, представленного в виде ассемблерного кода, путем анализа инструкций, перенаправляющих поток по другому адресу. По построенному графу можно провести сигнатурный анализ. Под сигатурой CFG понимается остовное дерево (дерево, полученное из графа путем удаления циклических ребер), вершинами которого являются вершины графа.

В работе [7] в качестве базы сигнатур используют конечный автомат, состоящий из конечного набора таких деревьев, на вход которому подается граф потока выполнения анализируемого ПО. Следует отметить, что вредоносная программа, заражая легитимное ПО, внедряет свой код в тело программы, таким образом, граф потока выполнения вредоносной программы является подграфом CFG оригина-

льной программы. Для более глубокого анализа, на вход в автомат подают не только CFG анализируемой программы, но и его подграфы.

Альтернативный подход для статического анализа графа потока выполнения описан в работе [8], где в качестве функции сравнения вершин графов выступает функция

$$\text{Match}(B_i, B_j) = \frac{LCB(B_i, B_j)}{\text{length}(B_j)} \times 100\%,$$

где B_i , B_j – вершины сравниваемых графов; $\text{length}(B_j)$ – общее количество символов в ассемблерном коде вершины B_j ; $LCB(B_i, B_j)$ – максимальное количество символов в блоке B_i , которые совпадают с символами в блоке B_j . Процент соответствия вершины одного графа вершине другого графа сохраняется в матрице A размера $m \times n$, как результат сравнения кода программы P_a и P_b . Итоговая оценка соответствия рассчитывается по формуле

$$\text{score} = \frac{\sum_{i=j, j=0}^n a_{ij}}{L},$$

где a_{ij} – диагональные элементы матрицы A ; L – общее количество вершин.

Недостатками такого подхода являются:

- большой объем базы данных сигнатур образцов вредоносного ПО;
- сложность процесса построения графа из-за наличия в исходном коде программы инструкций, аргументы которых требуют предварительного вычисления;
- большие временные затраты на поиск сигнатуры в базе сигнатур.

Решением недостатка оптимизации базы данных (БД) сигнатур вредоносного ПО является разработка системы классификации на основе искусственных нейронных сетей. Такой подход позволит не только оптимизировать размер БД, но и использовать более эффективный способ сопоставления сигнатуры вредоносного ПО с образцами в БД в сравнении с определением изоморфизма графов.

Такие подходы зачастую оперируют множеством числовых параметров. Для анализа графа требуется получить его численное представление в виде вектора чисел, т.е. нужно выделить множество вещественных или целочисленных параметров, которые будут характеризовать граф потока выполнения.

В работе [9] рассматривают метод классификации графов с помощью алгоритма graph2vec. Для векторизации графа применяется подход, в котором считается количество уникальных вершин графа и из полученных результатов формируется конечный вектор. Причем процедура сравнения двух вершин включает в себя сравнение целочисленных векторов, полученных из вершины путем подсчета количества вхождений ассемблерных инструкций архитектуры x86 в заголовков вершины.

Построение графа потока выполнения

Граф потока выполнения представляет собой модель поведения процесса в системе. Вершинами графа являются непрерывные участки ассемблерного кода, в которых отсутствуют операции перенаправления потока выполнения. В архитектуре процессоров x86 можно выделить следующие инструкции, перенаправляющие поток выполнения на другой участок памяти:

- инструкции условного и безусловного перехода (jmp, jcc, и т.п.);
- инструкция call;
- инструкция get;
- инструкция loop.

Каждая из них при построении CFG добавляет новую вершину в граф. Исключением являются инструкции условного перехода (jmp, jcc, jnz, и т.п.), которые перенаправляют поток сразу в две вершины. Адрес начальной инструкции первой вершины – это адрес перехода в инструкции условного перехода, а начальным адресом второй является значение суммы адреса инструкции условного перехода и 0x1. Так, например:

```
0x0004FF8    jnz 0x0004FF0.
```

Этот ассемблерный код добавляет в граф потока две новые вершины:

- с начальным адресом 0x0004FF0;
- с начальным адресом 0x0004FF9.

Слева на рис. 1 изображен пример ассемблерного кода, который состоит из семи инструкций x86-64 архитектуры, а также граф потока выполнения, который получается на основе преобразования этих демонстрационных инструкций.

Рис. 1. Пример построения графа потока выполнения

Первая инструкция от точки входа в приложение попадает в начальную вершину графа. Следующая инструкция безусловного перехода добавляет новую пустую вершину в граф, которую можно не включать в него, так как это не имеет практического смысла. Следующая инструкция вызывает функцию func (для упрощения примера ассемблерный код функции не приводится), которая также добавляет в граф новую вершину. Выполнение функции func завершается командой get, которая создаст новую пустую вершину. После идет условный переход по относительному адресу +0x2. Эта инструкция добавляет в граф две вершины с соответствующим ассемблерным кодом.

Графовое представление вредоносного ПО, как в случае анализа сигнатур, меняет свою форму, если используются методы обфускации, шифрования, упаковки исходного кода программы [10]. Для доступа к инструкциям исходного кода ПО используются различные подходы с применением распаковщиков и песочниц [11–13].

Ассемблерный код необходимо привести к нормальной форме, проведя деобфускацию кода. В работе [6] описан процесс нормализации ассемблерного кода, который состоит из следующих шагов:

- удалить незначимые инструкции por;
- удалить пары push eax; pop eax, если регистр eax не был использован в промежутке между этими инструкциями;
- удалить пары inc eax; dec eax, если регистр eax не был использован в промежутке между этими инструкциями;
- эквивалентные инструкции xor eax, eax; mov eax, 0 удалить из исходного кода ПО.

Одной из проблем построения графа потока выполнения является тот факт, что адреса переходов и вызовов функций могут быть как абсолютными (jmp 0x0000f4f5), так и вычисляемыми (jmp eax). Если адреса являются вычисляемыми и возможно вычислить абсолютный адрес перехода, нужно его заменить. Иначе эта инструкция игнорируется и процесс построения продолжается без её участия.

Классификация программного обеспечения при помощи методов интеллектуального анализа данных

Методы интеллектуального анализа данных успешно применяются в средствах обнаружения атак и вторжений, функционирующих в сетевой инфраструктуре. В работе [14] для анализа состояния компьютерной сети использовали классификатор, в основе которого лежит нейронная сеть на базе адаптивно резонансной теории (АРТ) [15]. АРТ сети типа АРТ-2 подходят для задач кластеризации, хранения и идентификации образов, представленных в форме непрерывных значений сигналов.

Выходной слой такой сети представляет собой набор нейронов, каждый из которых представляет собой один класс объектов. Таким образом, подавая на вход качественно похожие графовые модели, будут активизироваться соответствующие им нейроны

выходного слоя. В свою очередь, если же входной сигнал не активирует хотя бы один нейрон выходного слоя, то это значит, что на вход АРТ сети подается новый образ вредоносного ПО, для классификации которого в выходной слой этой нейронной сети добавляется новый нейрон. Такой подход позволяет уменьшить размер памяти, требующийся для хранения графов известного вредоносного ПО. Особенностью данного классификатора является то, что его можно использовать в системах реального времени, так как алгоритм обучения не затрагивает прошлую память и позволяет дообучать сеть в процессе функционирования.

Рис. 2. Граф потока выполнения приложения, включающего в себя сетевые операции и операции доступа к файлам в системе. Содержимое одной вершины

Приложение, обладающее сравнительно простой логикой, в итоге представляет собой, достаточно большую сущность в виде графа. Для ускорения процесса построения графа возможно применение некоторых ограничений на количество вершин. В данном примере вершины графа содержат в себе наборы машинных инструкций, некоторые из которых изображены на рисунке справа.

В качестве способа векторизации полученного графа была использована модель bags-of-words. Используя преобразования графа в вектор, были получены следующие данные:

[11, 14, 23, 8, 52, 52, 52, 44, 44, 21, 21, 44, 23].

Для его классификации была использована сеть АРТ-2 показаны на рис. 3.

Рис. 3. Модель используемой нейронной сети типа АРТ-2

Основная идея заключается в том, что распознавание образов, представленных векторами $I' = \{I_1, \dots, I_n\}$, является результатом частичного или

Результаты исследования

В рамках описанного выше алгоритма для построения графа потока выполнения была разработана программа, на вход которой подается дизассемблированный код исполняемого файла с набором инструкций x86-64, а результатом работы является граф потока выполнения этого исполняемого файла.

На рис. 2 изображен граф потока выполнения для простого приложения, совершающего операции сетевого подключения к удаленной машине и чтения системных файлов. При построении графа количество вершин, входящих в него, напрямую зависит от размера исходного исполняемого файла.

полного соответствия состояния весов одного из обученных распознающих нейронов входному нормализованному вектору, т.е. вхождения в резонанс сенсорного ($F_{ВХ}$) и распознающего ($F_{ВЫХ}$) слоев сети. Возникший резонанс оценивается управляющими нейронами (R).

Если оценка резонанса управляющими нейронами R достаточна, т.е. превышает заранее определенный порог, то считается, что соответствие между вектором входных данных и образом из памяти сети (Y) установлено.

На рис. 4 изображен график зависимости времени распознавания нового вектора от количества разных ранее обученных векторов. По оси абсцисс изображено количество различных запомненных векторов, по оси ординат – время распознавания поданного образа.

Рис. 4. График зависимости времени распознавания нового вектора от количества разных обученных векторов

Для эксперимента были сгенерированы 2 различных класса состояний графа потока выполнения,

каждый из которых состоял из 6 тыс. образов. Каждый последующий образ генерировался путем добавления дополнительных инструкций. Класс вредоносного программного обеспечения базировался на том, что не выполнял закрытие сетевого сокета и освобождение файлового дескриптора.

В ходе эксперимента были получены следующие результаты: программное обеспечение, которое не являлось вредоносным, классифицировалось с точностью 0,97, а вредоносное программное обеспечение – с точностью 0,95. Небольшое расхождение при классификации можно пояснить наличием случайных факторов добавления новой инструкции, которые при большом количестве смещали восприятие памяти АРТ (У).

Из графика и полученных данных классификации видно, что использование в качестве классификатора АРТ-2 показывает результаты, на основе которых можно сделать вывод о возможности использования предложенного подхода в системах реального времени.

Заключение

В работе был предложен подход классификации вредоносного ПО с помощью построения графа потока выполнения ассемблерного кода. Предложен способ векторизации графа и дальнейшего его анализа. Эффективность системы обнаружения вредоносного ПО при таком подходе к векторизации графового представления зависит от функции эквивалентности векторных представлений, в работе применялся подход попарного сравнения элементов, который не дает положительных результатов при анализе сильно обфусцированных исходных кодов. Для повышения качества распознавания полиморфного вредоносного ПО было предложено использовать другой подход к сравнению векторных представлений (использование искусственных нейронных сетей для классификации численных векторов). В качестве классификатора была использована сеть АРТ-2. В отличие от большинства других искусственных сетей она не требует переобучения при добавлении нового класса распознаваемых образов. Проведенные эксперименты показывают возможность использования АРТ-2 в качестве классификатора вредоносного ПО. Такой подход позволяет создавать системы обнаружения вредоносного кода, работающие в режиме реального времени.

Литература

1. Sikorski M. Practical Malware Analysis. The Hands – On Guide to Detecting Malicious Software / M. Sikorski, A. Honig. – San Francisco: No Starch Press, 2012. – 766 p.
2. Харченко С.С. Сигнатурный анализ программного кода / С.С. Харченко, Е.М. Давыдова, С.В. Тимченко // Ползуновский вестник. – 2012. – № 3. – С. 60–64.
3. Аветисян А.И. Технологии статического и динамического анализа уязвимостей программного обеспечения / А.И. Аветисян, А.А. Белеванцев, И.И. Чукляев // Вопросы кибербезопасности. – 2014. – Т. 4, № 3. – С. 20–28.
4. Linn C. Obfuscation of executable code to improve resistance to static disassembly / C. Linn, S. Debray // In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03). – 2003. – P. 290–299. – URL: <http://www2.cs.arizona.edu/solar/papers/CCS2003.pdf> (дата обращения: 18.02.2018).
5. Cai H. Certified self-modifying code / H. Cai, Z. Shao, A. Vaynberg // Conf. on Prog. Lang. Design and Implementation (PLDI). – New York: ACM Press, 2007. – P. 66–77.
6. Cappaert J. Self-encrypting code to protect against analysis and tampering / J. Cappaert, N. Kisslerli, D. Schellekens, B. Preneel // 1st Benelux Workshop on Information and System Security. – 2006. – P. 14. – URL: <https://www.esat.kuleuven.be/cosic/publications/article-811.pdf> (дата обращения: 21.02.2018).
7. Bonfante G. Architecture of a Morphological Malware Detector / G. Bonfante, M. Kaczmarek, J. Marion // Journal in Computer Virology. – 2009. – Vol. 5, No. 3. – P. 263–270.
8. Vinod P. Static CFG Analyzer for Metamorphic Malware Code / P. Vinod // Proc. 2nd Int. Conf. Security of Information and Networks. – N.Y.: ACM SIGSAC, 2009. – P. 225–228.
9. Narayanan A. Subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs / A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan // arXiv:1606.08928. – 2016. – URL: <https://arxiv.org/pdf/1606.08928.pdf> (дата обращения: 23.02.2018).
10. Касперски К. Упаковки исполняемых файлов в LINUX/BSD и методы борьбы с ними // Системный администратор. – 2006. – № 9. – URL: https://www.evilmfingers.com/publications/research_RU/linux-unpack.pdf (дата обращения: 27.02.2018).
11. Filiol E. Advanced viral techniques: mathematical and algorithmic aspects // Journal in Computer Virology. – 2009. – Vol. 5, No. 4. – P. 263–270.
12. Filiol E. Computer Viruses: from Theory to Applications. – Verlag: Springer-Verlag, 2005. – 411 p.
13. Ször P. The Art of Computer Virus Research and Defense. Hangerstown: Phoenix BookTech, 2005. – 744 p.
14. Буханов Д.Г. Определение состояния компьютерной сети на основе использования нейронных сетей арт / Д.Г. Буханов, В.М. Поляков, А.В. Смакаев // Вестник Белгород. гос. технол. ун-та им. В.Г. Шухова. – 2017. – Т. 1, № 7. – С. 157–162.
15. Jain L. Innovations in ART Neural Networks / L. Jain, B. Lazzerini, U. Halici. – Heidelberg; New York: Physica-Verl, 2000. – 189 p.

Буханов Дмитрий Геннадьевич

Магистрант каф. программного обеспечения вычислительной техники и автоматизированных систем (ПОВТиАС) Белгородского государственного университета им. В.Г. Шухова (БГТУ им. В.Г. Шухова) Костюкова ул., д. 46, г. Белгород, Россия, 308012
Тел.: +7-904-084-93-66
Эл. почта: db.old.stray@gmail.com

Сулохин Дмитрий Васильевич

Студент 5-го курса каф. ПОВТиАС БГТУ им. В.Г. Шухова Костюкова ул., д. 46, г. Белгород, Россия, 308012
Тел.: +7-950-718-43-55
Эл. почта: profacorp@gmail.com

Bukhanov D.G., Sulokhin D.V.

Detection of malware based on the classification of source code graphs

This paper contains an analysis of existing approaches to detect the malicious software. An approach to classify the software in computer systems of information processing is offered. Proposed approach consists of three steps: construction of the flow graph of an application's source code, vectorization of the received graph and classification of the graph based on an artificial neural network of adaptive resonance theory. The graph is vectorized based on the bag-of-word model. Obtained results show an application of proposed approach in malware detection field.

Keywords: malware detection, control-flow graph, neural network, data classification.

doi: 10.21293/1818-0442-2018-21-3-30-34

References

1. Sikorski M., Honig A. *Practical Malware Analysis. The Hands – On Guide to Detecting Malicious Software*. San Francisco, No Starch Press Publ., 2012. 766 p.
2. Harchenko S.S., Davydova E.M., Timchenko S.V. Signature analysis of program code. *Polzunovskij vestnik*, 2012, no. 3, pp. 60–64 (In Russ.).
3. Avetisyan A.I., Belevancev A.A., Chuklyaev I.I. Technologies for static and dynamic analysis of software vulnerabilities. *Voprosy kiberbezopasnosti*, 2014, vol. 4, no. 3, pp. 20–28.
4. Linn C, Debray S. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, 2003, pp. 290–299. Available at: <http://www2.cs.arizona.edu/solar/papers/CCS2003.pdf> (accessed: February 18, 2018).
5. Cai H., Shao Z., Vaynberg A. Certified self-modifying code. In *Conf. on Prog. Lang. Design and Implementation (PLDI)*. New York, ACM Press Publ., 2007, pp. 66–77.
6. Cappaert J., Kisserli N., Schellekens D., and Preneel B. Self-encrypting code to protect against analysis and tampering. *1st Benelux Workshop on Information and System Security*, 2006, pp. 14. Available at: <https://www.esat.kuleuven.be/cosic/publications/article811.pdf> (accessed: February 21, 2018).
7. Bonfante G., Kaczmarek M., Marion J. Architecture of a Morphological Malware Detector. *Journal in Computer Virology*, 2009, vol. 5, no. 3, pp. 263–270.

8. Vinod P. Static CFG Analyzer for Metamorphic Malware Code. *Proc. 2nd Int. Conf. Security of Information and Networks*. New York, ACM SIGSAC Publ., 2009, pp. 225–228.

9. Narayanan A., Chandramohan M., Chen L., Liu Y., and Saminathan S. Subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv:1606.08928*, 2016, Available at: <https://arxiv.org/pdf/1606.08928.pdf> (accessed: February 23, 2018).

10. Kasperski K. Methods of prevention of packers in Linux and BSD operation systems. *System Administrator*, 2006. Available at: https://www.evilmfingers.com/publications/research_RU/linux-unpack.pdf (accessed February 27, 2018) (In Russ.).

11. Filiol E. Advanced viral techniques: mathematical and algorithmic aspects. *Journal in Computer Virology*, 2009, vol. 5, no. 4, pp. 263–270.

12. Filiol E. *Computer Viruses: from Theory to Applications*. Verlag Springer-Verlag, 2005. 411 p.

13. Ször P. *The Art of Computer Virus Research and Defense*. Hangerstown Phoenix BookTech, 2005. 744 p.

14. Buhanov D.G., Polyakov V.M., Smakaev A.V. Determination of computer network status based on ART neural networks. *Vestnik Belgorodskogo gosudarstvennogo tekhnologicheskogo universiteta im V.G.Shuhova*, 2017, vol. 1, no 7, pp. 157–162 (In Russ.).

15. Jain L., Lazzerini B., Halici U. *Innovations in ART Neural Networks*. New York, Physica-Verl Heidelberg Publ., 2000, 189 p.

Dmitry G. Buhanov

Graduate student. Software for computer facilities and automated systems, Belgorod State Technological University named after V.G. Shoukhov
46, Kostyukov st., Belgorod, Russia, 308012
Phone: +7-904-084-93-66
Email: db.old.stray@gmail.com

Dmitry V. Sulokhin

A fifth-year student. Software for computer facilities and automated systems, Belgorod State Technological University named after V.G. Shoukhov
46, Kostyukov st., Belgorod, Russia, 308012
Phone: +7-950-718-43-55
Email: profacorp@gmail.com