Revisión de algoritmos de detección de *malware* ofuscado basados en *machine learning*

Erly Galia Villarroel Enríquez
20182063@aloe.ulima.edu.pe
https://orcid.org/0000-0001-8566-0494
Universidad de Lima

Recibido: 6 de agosto del 2022 / Aceptado: 9 de octubre del 2022
doi: https://doi.org./10.26439/ciis2022.6076

RESUMEN. Los desarrolladores de *malware* cada vez evolucionan más sus técnicas para lograr atacar efectivamente el sistema. Una de estas técnicas es la ofuscación de código, la cual dificulta la detección de *malware* en los mecanismos tradicionales que utilizan los antivirus actuales. Ante ello, se propone la revisión de artículos relacionados con la detección de *malware* ofuscado con *machine learning* para elegir las mejores técnicas de análisis de este tipo de *malware* y emplear los mejores algoritmos para una futura experimentación con ellos.

PALABRAS CLAVE: *malware*, ofuscación, detección, *machine learning*

REVIEW OF OBFUSCATED MALWARE DETECTION ALGORITHMS BASED ON MACHINE LEARNING

ABSTRACT. Malware developers develop their techniques to attack computer systems effectively. One of these techniques is code obfuscation which makes it difficult to detect malware in the traditional mechanisms used by current antiviruses. This article reviews research related to the detection of obfuscated malware with machine learning to choose the best analysis techniques for this type of malware and use the best algorithms for future experimentation with them.

KEYWORDS: malware, obfuscation, detection, machine learning

1. INTRODUCCIÓN

El reporte de FortiGuard Labs (Fortinet, 2022) para América Latina y el Caribe señala que esta región recibió el 10 % del total de intentos de ciberataques que se dieron en el mundo en el 2021, con más de 289 000 millones de intentos de ciberataques, un aumento del 600 % con respecto al 2020. Según datos recabados por FortiGuard Labs, el laboratorio de inteligencia de amenazas de Fortinet, Perú es el tercer país que más intentos de ataques recibió (11 500 millones), después de México y Brasil.

Duo et al. (2022) sostienen que los ciberataques son problemas de seguridad que se presentan en sistemas cibernéticos y que pueden perjudicar el rendimiento del sistema o causar mayores daños en él. Estos ataques pueden ser de denegación de servicio, *phishing* y *malware*, entre otros.

Según diversos autores (Saravia et al., 2019, como se citó en Ashik et al., 2021), el *malware* o *software* malicioso es un programa informático dañino inyectado en programas legítimos para perpetrar acciones ilícitas. Debido al rápido crecimiento de internet y a la aparición de diversos dispositivos conectados a través de redes, el panorama de ataques de *malware* ha aumentado afectando a la privacidad de los usuarios.

Con respecto a las causas que originan las infecciones de *software* malicioso, Ashik et al. (2021) señalan principalmente a las descargas de *software* gratuito. Estas incluyen *freeware* (*software* gratuito) de juegos, navegadores web, antivirus gratuitos, etcétera.

Existen tres formas comunes en la que un *malware* puede entrar a un dispositivo. La primera es el ataque de descarga: cuando el atacante aloja *malware* en un servidor web para infectar a los dispositivos que visiten esa página web. La segunda forma es un ataque de actualización: cuando una aplicación benigna se actualiza tomando características de *malware*. Finalmente, está el ataque de reempaquetado, donde el desarrollador de *malware* empaqueta una aplicación benigna inyectando código malicioso en dicha aplicación (Felt et al., 2014, como se citó en Surendran & Thomas, 2022).

Los ataques de *malware* han ocasionado grandes pérdidas financieras tanto a organizaciones como a individuos (Ashik et al., 2021), así como amenazas de filtración de información personal y robo de información sensible (Lee et al., 2022).

Según Liu et al. (2020), la detección de *malware* es afectada por técnicas como carga dinámica de código, ofuscación de código, entre otras. La ofuscación de código es una técnica que dificulta la detección de comportamiento malicioso en análisis estáticos. Además, facilita al desarrollador de *malware* la creación de múltiples variables de *malware* que pueden causar detecciones erróneas en la mayoría de sistemas existentes (Arp et al., 2014, como se citó en Ouk & Pak, 2022).

Los atacantes usualmente emplean la ofuscación de *malware* para evadir la detección de los programas antivirus. Estos están basados en la coincidencia de patrones de *malware*, pero apenas detectan los nuevos *malware* (Mimura et al., 2016, como se citó en Mimura & Ito, 2022).

Las investigaciones actuales se centran en mejorar la detección de *malware* empleando algoritmos de *machine learning*. Un ejemplo de ello es la investigación de Mahindru & Sangal (2020), quienes extrajeron características de las aplicaciones que recolectaron (benignas o con *malware*) para luego entrenarlas usando distintos algoritmos de *machine learning* y compararlas para determinar el mejor algoritmo para detectar *malware* en dispositivos móviles Android.

Esta búsqueda de mejora en detección incluye al *malware* ofuscado. Según Wu et al. (2021), un desafío abierto es explicar el comportamiento malicioso de un *malware* ofuscado a través del análisis de sus características. Los modelos actuales de *machine learning* solo usan una pequeña porción de estas características para explicar la clasificación de un programa ofuscado como *malware*.

Para Sun et al. (2021), un nuevo desafío en el área de seguridad es el desarrollo de una solución que cuenta con precisión y escalabilidad para detección de *malware* y su clasificación en familias.

Finalmente, Surendran y Thomas (2022) sugieren que se debe reducir la tasa de falsos positivos (cuando una aplicación es clasificada erróneamente como *malware*).

2. DESCRIPCIÓN GENERAL DEL TRABAJO

Los desafíos en este campo, como se explicó en la sección de introducción, resaltan la necesidad de una mayor investigación. El presente documento tiene como objetivo analizar las técnicas de preprocesamiento utilizadas para desocultar *malware* ofuscado, además de algoritmos de *machine learning*, para encontrar el mejor algoritmo que pueda detectar *malware* ofuscado en menor tiempo y con la mayor precisión posible.

3. CONTRIBUCIONES

Las contribuciones del presente trabajo son las siguientes:

- Realizar una investigación actualizada sobre la detección de *malware* ofuscado.
- Dar una visión sobre las técnicas de preprocesamiento para desocultar *malware* ofuscado.

- Comparar los modelos de *machine learning* usando métricas objetivas como precisión y exactitud.

REFERENCIAS

Ashik, M., Jyothish, A., Anandaram, S., Vinod, P., Mercaldo, F., Martinelli, F., & Santone, A. (2021). Detection of malicious software by analyzing distinct artifacts using machine learning and deep learning algorithms. *Electronics, 10*(14), 1694. https://doi.org/10.3390/electronics10141694

Duo, W., Zhou, M., & Abusorrah, A. (2022). A survey of cyber attacks on cyber physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*, *9*(5), 784-800. https://doi.org/10.1109/JAS.2022.105548

Fortinet. (2022, 8 de febrero). *América Latina sufrió más de 289 mil millones de intentos de ciberataques en 2021* [Comunicado de prensa]. https://www.fortinet.com/lat/corporate/about-us/newsroom/press-releases/2022/fortiguard-labs-reporte-ciberataques-america-latina-2021

International Business Machines. (2022). *X-Force Threat Intelligence Index 2022*.IBM Security X-Force. https://www.ibm.com/security/data-breach/threat-intelligence/

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., & Liu, H. (2020). A review of android malware detection approaches based on machine learning. *IEEE Access*, *8*, 124579-124607. https://doi.org/10.1109/ACCESS.2020.3006143

Lee, D., Jeon, G., Lee, S., & Cho, H. (2022). Deobfuscating mobile malware for identifying concealed behaviors. *Computers, Materials and Continua*, *72*(3), 5909-5923. http://dx.doi.org/10.32604/cmc.2022.026395

Mimura, M., & Ito, R. (2022). Applying NLP techniques to malware detection in a practical environment. *International Journal of Information Security*, *21*(2), 279-291. https://doi.org/10.1007/s10207-021-00553-8

Ouk, P. C., & Pak, W. (2022). High performance classification of android malware using ensemble machine learning. *Computers, Materials and Continua, 72*(1), 381-398. http://dx.doi.org/10.32604/cmc.2022.024540

Sun, B., Takahashi, T., Ban, T., & Inoue, D. (2021). Detecting Android malware and classifying its families in large-scale datasets. *ACM Transactions on Management Information Systems (TMIS), 13*(2), 1-21. https://doi.org/10.1145/3464323

Surendran, R., & Thomas, T. (2022). Detection of malware applications from centrality measures of syscall graph. *Concurrency and Computation: Practice and Experience*, *34*(10), e6835. https://doi.org/10.1002/cpe.6835

Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., & Lyu, M. R. (2021). Why an Android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(2), 1-29. https://doi.org/10.1145/3423096

Revisión de algoritmos de detección de *malware* ofuscado basados en *machine learning*

Erly Galia Villarroel Enríquez
20182063@aloe.ulima.edu.pe

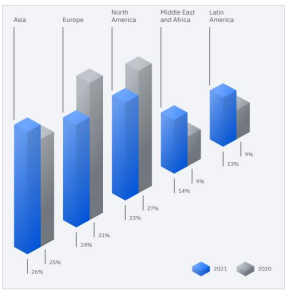
Resumen. Los desarrolladores de *malware* cada vez evolucionan más sus técnicas para lograr atacar efectivamente el sistema, una de estas técnicas es la ofuscación de código que dificulta la detección de *malware* en los mecanismos tradicionales que utilizan los antivirus actuales. Ante ello se propone la revisión de artículos relacionados con la detección de *malware* ofuscado con *machine learning* para elegir las mejores técnicas de análisis de este tipo de *malware* y emplear los mejores algoritmos para una futura experimentación con los mismos.

Introducción

El reporte de FortiGuard Labs (Fortinet, 2022) para América Latina y el Caribe señala que esta región recibió el 10 % del total de intentos de ciberataques que se dieron en el mundo en el 2021, con más de 289 000 millones de intentos de ciberataques. Según datos recabados por FortiGuard Labs, Perú es el tercer país que más intentos de ataques recibió (11 500 millones), después de México y Brasil. Los ataques de *malware* han ocasionado grandes pérdidas financieras tanto a organizaciones como a individuos (Ashik et al., 2021), amenazas de filtración de información personal y robo de información sensible (Lee et al., 2022). El problema descrito anteriormente ha generado la necesidad de mecanismos para detección de *malware*; sin embargo, los atacantes siempre innovan en técnicas para evitar la detección. Tal como lo señalan Liu et al. (2020), quienes mencionan que la detección de *malware* es afectada por técnicas como carga dinámica de código, ofuscación de código, entre otras. La ofuscación de código es una técnica que dificulta la detección de comportamiento malicioso en análisis estáticos. Además, facilita al desarrollador de *malware* la creación de múltiples variables de *malware* que pueden causar detecciones erróneas en la mayoría de sistemas existentes. (Arp et al., 2014, como se citó en Ouk & Pak, 2022).

Los atacantes usualmente emplean la ofuscación de *malware* para evadir la detección de los programas antivirus, estos están basados en la coincidencia de patrones de *malware* que apenas detectan los nuevos *malware* (Mimura et al., 2016, como se citó en Mimura & Ito, 2022).

Figura 1: Ataques de *malware* a nivel mundial, 2021 vs. 2020. Fuente: IBM Security X-Force



Materiales y métodos

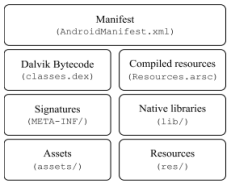
Se revisaron artículos en las bases de datos IEEE Xplore y Scopus que trataban sobre técnicas para detectar *malware* ofuscado en aplicaciones (análisis de *bytecode* y llamadas al sistema) y los principales algoritmos de *machine learning*.

Resultados

Detección basada en análisis de *bytecode*

Para el caso de aplicaciones Android, Latchar et al. (2020) sostienen que estas están empaquetadas en forma de archivos APK (Android Package), estos a su vez constan de varios archivos, dentro de los cuales está incluido un archivo ejecutable Dalvik (dex), siendo este el que contiene el *bytecode* o código de *bytes*. Este código es ejecutado por el sistema de Android Runtime (ART), este entorno de ejecución se encarga de transformar el *bytecode* a código máquina para que la aplicación pueda ser ejecutada por el sistema operativo correspondiente.

Figura 2: Estructura de los archivos APK según Chen et al. (2019)



Detección basada en análisis de llamadas al sistema

Según Surendran y Thomas (2022), las llamadas sensibles a una API generalmente contienen más información que las llamadas al sistema; sin embargo, al utilizar mecanismos que utilizan llamadas al sistema pueden darse erróneas clasificaciones cuando una aplicación benigna invoca llamadas API que son típicas de aplicaciones con *malware*. Por lo tanto, el seguimiento de las llamadas al sistema de una aplicación contiene más información relevante sobre el comportamiento malicioso que las llamadas API sensibles invocadas por la aplicación.

Figura 3: Captura de pantalla de la ejecución en Linux del comando *strace*, el archivo C a ejecutar fue renombrado como "prueba" y se obtuvieron las llamadas al sistema y la frecuencia de estas.

```
tatoo@tatoo-VirtualBox:~/Documentos$ gcc -o prueba sec.c
tatoo@tatoo-VirtualBox:~/Documentos$ strace -c ./prueba
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0,00    0,000000           0         1           read
  0,00    0,000000           0         1           write
  0,00    0,000000           0         3           close
  0,00    0,000000           0         8           mmap
  0,00    0,000000           0         3           mprotect
  0,00    0,000000           0         1           munmap
  0,00    0,000000           0         3           brk
  0,00    0,000000           0         4           pread64
  0,00    0,000000           0         1         1 access
  0,00    0,000000           0         1           execve
  0,00    0,000000           0         2         1 arch_prctl
  0,00    0,000000           0         1           set_tid_address
  0,00    0,000000           0         3           openat
  0,00    0,000000           0         3           newfstatat
  0,00    0,000000           0         1           set_robust_list
  0,00    0,000000           0         1           prlimit64
  0,00    0,000000           0         1           getrandom
  0,00    0,000000           0         1           rseq
------ ----------- ----------- --------- --------- ----------------
100,00    0,000000           0        39         2 total
```

Tabla 1: Descripción de algunas llamadas al sistema

Nombre	Descripción
access	Verifica los accesos a un fichero
pread64	Para leer o escribir en un fichero
brk	Cambia el tamaño del segmento de datos
fork	Permite a un proceso crear un proceso hijo
accept	Acepta una conexión sobre un conector (*socket*)

Técnicas de *machine learning* utilizadas en detección de *malware* ofuscado

Tabla 2: Comparación de artículos que emplean *malware* ofuscado

Artículo	Dataset	Técnicas de ofuscación	Técnica de ML empleada	Resultados (% accuracy)
AndrODet: An adaptive Android obfuscation detector	Se utilizó el conjunto de datos de AMD y también se recopilaron aplicaciones del repositorio de software "F-Droid".	Identifier renaming String encryption Control flow obfuscation	SVM	92,02 81,41 68,32
Impact of Code Deobfuscation and Feature Interaction in Android Malware Detection	Se utilizó el conjunto de datos Drebin y AndroZoo.	Code Reduction	LightGBM	99,54
Detection of Malware Applications from Centrality Measures of Syscall Graph	Se utilizó el conjunto de datos Drebin y AMD	Dynamic code loading, renaming and String Encryption	Random Forest	99
Obfuscated VBA Macro Detection Using Machine Learning	Recolectado por los mismos autores a través de búsqueda en Google	Random obfuscation, split obfuscation, encoding obfuscation and logic obfuscation	Random Forest	90,3
Detecting obfuscated JavaScripts using Machine Learning	Utilizaron el dataset "jsDeliver", el cual ofuscaron en la página "javascriptobfuscator.com"	Minificación y compresión	Random Forest SVM	99 97

Conclusiones

A través de la revisión de varios artículos de investigación y documentos de conferencias, podemos denotar la importancia de la aplicación de técnicas de aprendizaje automático en la detección de *malware* ofuscado. Además, destacamos dos importantes técnicas de detección de *malware* ofuscado en aplicaciones Android: el análisis *bytecode* y las llamadas al sistema, para futuros trabajos, estas serán utilizadas en una experimentación utilizando los algoritmos Random Forest y MLP que fueron los que obtuvieron los mejores resultados en todos los documentos revisados.

Referencias

Amin, M., Shah, B., Sharif, A., Ali, T., Kim, K. I., & Anwar, S. (2022). *Android malware detection through generative adversarial networks. Transactions on Emerging Telecommunications Technologies, 33*(2), e3675.

Chen, Y. C., Chen, H. Y., Takahashi, T., Sun, B., & Lin, T. N. (2021). Impact of Code Deobfuscation and Feature Interaction in Android Malware Detection. *IEEE Access, 9*, 123208-123219.

International Business Machines. (2022).*X-Force Threat Intelligence Index 2022.* IBM Security X-Force. https://www.ibm.com/security/data-breach/threat-intelligence/

Mahindru, A., & Sangal, A. L. (2021). MLDroid—Framework for Android malware detection using machine learning techniques. *Neural Computing and Applications, 33*(10), 5183-5240.

Mirzaei, O., de Fuentes, J. M., Tapiador, J., & Gonzalez-Manzano, L. (2019). AndrODet: An adaptive Android obfuscation detector. *Future Generation Computer Systems, 90*, 240-261.

Sun, B., Takahashi, T., Ban, T., & Inoue, D. (2021). Detecting Android Malware and Classifying Its Families in Large-scale Datasets. *ACM Transactions on Management Information Systems (TMIS), 13*(2), 1-21.

Surendran, R., & Thomas, T. (2022). Detection of malware applications from centrality measures of syscall graph. *Concurrency and Computation: Practice and Experience, 34*(10), e6835.

Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., & Lyu, M. R. (2021). Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM), 30*(2), 1-29.

Xu, P., & Khairi, A. E. (2021). Android-COCO: Android Malware Detection with Graph Neural Network for Byte-and Native-Code. arXiv preprint arXiv:2112.10038.

Agradecimientos

Agradezco a mi asesor Juan Gutiérrez por brindarme su apoyo a lo largo de la elaboración de este documento, al profesor Pablo Rojas por brindarme pautas para realizar la exposición, y a mis amigos por animarme a exponer este póster.