

WrappEm: Uma Ferramenta para Injeção de Cargas Auxiliares por Manipulação em Nível de Byte no Windows

Augusto Goulart¹, Edson Fumagalli¹

¹Instituto Federal de Educação, Ciência e Tecnologia Farroupilha
CEP 97555-000 – Alegrete – RS – Brazil

jose.38001@aluno.iffar.edu.br

Abstract. *This work explores similar methods of subverting the Windows Image Loader for adversarial payload execution. These methods leverage byte-level manipulation techniques to modify the Import Directory and, when necessary, the Import Section of an executable file. This forces Windows to load an additional library into the virtual address space of the host process. The resulting artifacts from these methods – implemented through a terminal tool (WrappEm) – demonstrate that they can serve as alternatives to traditional export forwarding techniques, which rely on structurally empty and easily detectable libraries.*

Resumo. *Este trabalho explora métodos congêneres de subversão do Image Loader do Windows para a execução de cargas adversariais. Tais métodos utilizam técnicas de manipulação em nível de byte para modificar o Import Directory e, quando necessário, a Import Section de um arquivo executável. Isso força o Windows a carregar uma biblioteca adicional no espaço de endereçamento virtual do processo hospedeiro. Os artefatos resultantes desses métodos – implementados através de uma ferramenta de terminal (WrappEm) – demonstra que eles podem servir como alternativas às técnicas tradicionais de encaminhamento de exportação, as quais dependem de bibliotecas estruturalmente vazias e passíveis de detecção.*

1. Introdução

A injeção de carga auxiliar é fundamental para malwares direcionados aos binários do formato Portable Executable (PE) do Windows [Ling et al. 2023, Demetrio et al. 2021, Aryal et al. 2024, Nisi et al. 2021, Zhang et al. 2020]. Atores maliciosos frequentemente abusam de mecanismos nativos do sistema operacional (SO) para injetar essas cargas, que dependem do PE hospedeiro para serem executadas [Hauser and Cheptsov 2020], tornando-as um vetor comum para subverter o acesso ao espaço de endereçamento virtual de um processo. No Windows, cada processo é atribuído um espaço virtual na memória, que é traduzido em endereços de memória física pelo SO. Isso permite que cada processo tenha um espaço de memória isolado para execução. Esse espaço inclui todo o código, dados, pilha, heap e algumas partes das bibliotecas de vínculo dinâmico (DLLs) importadas por ele. Essas bibliotecas são buscadas da memória não-volátil conforme a hierarquia da ordem de precedência, definida sequencialmente desta forma: (a) pasta raiz do PE; (b) pasta do sistema (C:\Windows\System32); (c) pasta legado (C:\Windows\System); (d) pasta do SO (C:\Windows); (e) pasta atual; e, (f) pastas nos registros PATH (sistema e usuário) [Yosifovich et al. 2017]. Essa hierarquia permite

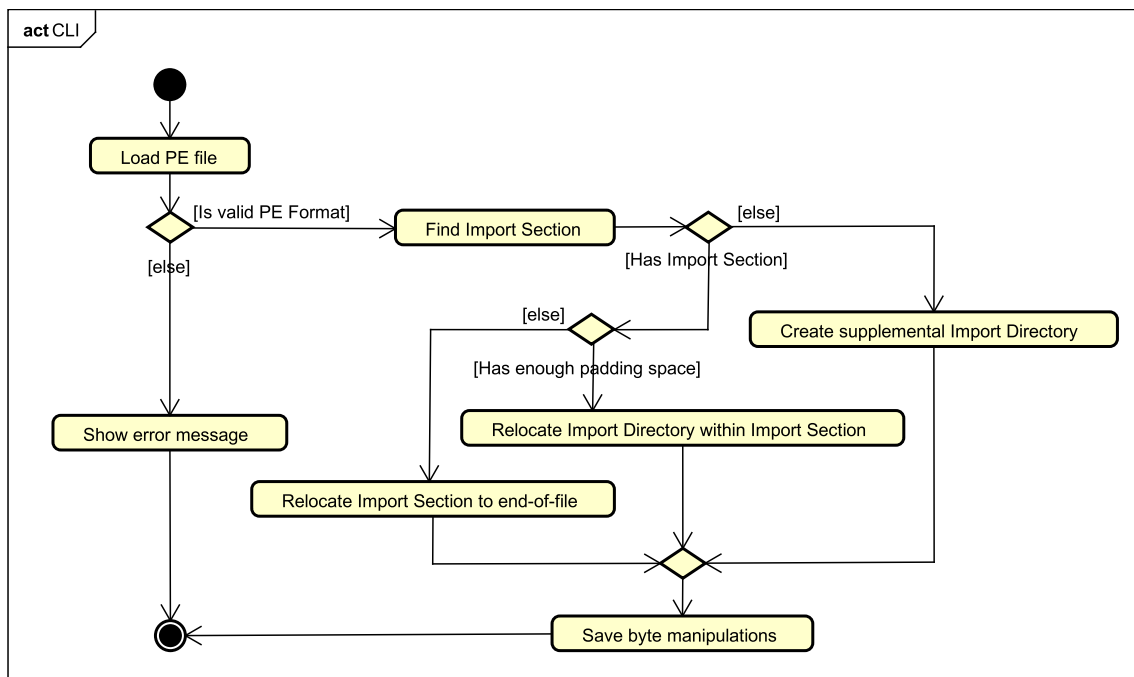


Figura 1. Diagrama de atividades da interface de linha de comando implementada como a ferramenta WrappEm.

que DLLs colocadas na pasta raiz do PE sejam importadas no lugar das bibliotecas do sistema – exceto quando essas já estão carregadas na memória volátil.

O encaminhamento de exportação é usado como um mecanismo para sequestro de DLLs e, embora explorá-lo para fins de *proxy* seja uma técnica bem estabelecida de interceptação de funções e uma base para a execução de código externo no Windows [Fernández-Álvarez and Rodríguez 2023], este introduz: (a) artefatos simples de detecção, devido à necessidade de uma estrutura binária com padrões repetitivos; e, (b) lógica de redirecionamento redundante, devido à necessidade de cada exportação chamar sua contraparte de destino.

Para mitigar essas desvantagens e também a crescente capacidade das soluções de segurança em analisar tanto recursos [Lad and Adamuthe 2022, Rizvi et al. 2022, Zhang et al. 2020] quanto comportamentos [Nguyen et al. 2024] adversariais tradicionais, torna-se imperativo explorar métodos alternativos de manipulação. Portanto, este trabalho documenta a concepção e validação de uma ferramenta em código aberto – nomeada WrappEm – para manipulação direta de arquivos PE, visando contribuir para uma compreensão mais profunda das vulnerabilidades do sistema Windows.

2. Metodologia

O formato de arquivos executáveis no Windows é definido pela especificação PE [Microsoft 2010]. Essa especificação fornece informações sobre como o SO carrega dependências externas no espaço de endereçamento virtual de um processo. A parte principal desse método é a análise (i.e. *parsing*) do Import Directory, realizada pelo Image Loader [Cheng et al. 2021]. Essa estrutura consiste em um vetor, terminado por um valor nulo, de Import Directory Descriptor; cada elemento contém informações sobre a DLL

que está sendo importada pelo arquivo PE [Yosifovich et al. 2017]. Ao manipular essa estrutura, é possível adicionar uma importação a mais para ser carregada pelo próprio SO, concedendo a essa nova DLL acesso ao espaço de endereçamento virtual do PE alvo. O uso indevido de espaços vazios entre seções é uma técnica conhecida de injeção de carga no Windows [Aryal et al. 2024]. No entanto, em vez de código, é possível que a Import Section possua espaço de preenchimento (i.e. *padding*) para acomodar o Import Directory original mais um elemento adicional. Em suma, esse método faz a realocação do diretório de importação dentro da seção.

O próximo método explorado faz a realocação da seção de importação no fim do arquivo. Este usa uma técnica de manipulação baseada em bytes conhecida como Section Injection [Demetrio et al. 2021] para anexar a Import Section como a última seção do arquivo PE. Isso elimina as exigências de preenchimento (i.e. *padding*) do método anterior, permitindo que ele seja utilizado após a realocação da seção. Duas restrições técnicas devem ser observadas sobre esse método: (a) o posicionamento do novo Section Header deve considerar o alinhamento da seção, caso contrário, corre-se o risco da Section Table invadir os limites da primeira seção (i.e. escrita fora dos limites); e, (b) todos os endereços relativos da seção devem ser incrementados pelo deslocamento virtual da realocação. Outro método explorado, porém com pouca particularidade, é a criação suplementar do Import Directory quando o PE não possui DLLs importadas.

Para validar esses métodos, e a própria ferramenta WrappEm, foram selecionadas para manipulação, com base em uma classificação das vinte principais DLLs importadas por malwares e alvos benignos [Yousuf et al. 2023], as bibliotecas: `dinput8.dll`, `version.dll`, `gdi32.dll`. Uma DLL *proxy* foi implementada para cada amostra alvo selecionada. Paralelamente, os métodos de manipulação direta de PE foram implementados utilizando C++ (padrão ISO/IEC 14882:2017), devido à sua capacidade de acesso à memória em baixo nível. O código está disponível publicamente online¹ sob a licença Microsoft Reciprocal License (Ms-RL) e pode ser compilado com MinGW, LLVM-Clang ou MSVC sem dependências externas (i.e. não são necessários cabeçalhos próprios do Windows). A figura 1 mostra o fluxo de trabalho da ferramenta ao carregar o arquivo alvo para manipulação.

3. Resultados preliminares

Tabela 1. Largura da imagem conforme o tamanho do arquivo PE

Faixa de tamanho do arquivo (kilobytes)	Largura da imagem (bytes)
$tam < 10$	32
$10 \leq tam < 30$	64
$30 \leq tam < 60$	128
$60 \leq tam < 100$	256
$100 \leq tam < 200$	384
$200 \leq tam < 500$	512
$500 \leq tam < 1000$	768
$tam \geq 1000$	1024

¹<https://github.com/oAGoulart/wrappem>

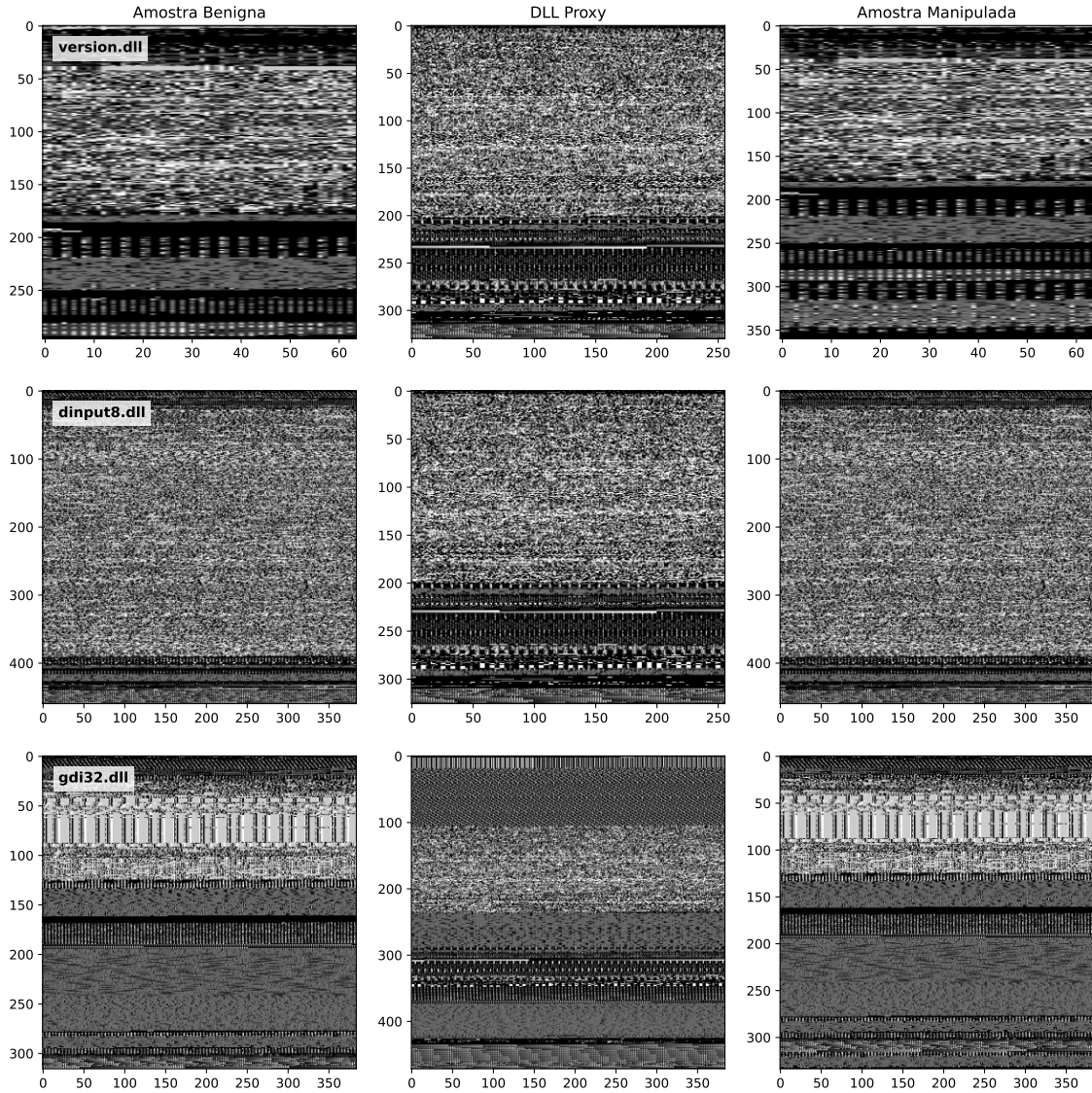


Figura 2. Mapeamento em tons de cinza das amostras benignas originais, das DLLs *proxy* de referência, e das amostras após manipulação em byte pela ferramenta WrappEm.

A ferramenta implementada é funcionalmente viável em testes preliminares e está disponível para reprodução, bem como os artefatos gerados. No Windows 11, as amostras manipuladas conseguiram executar sua carga auxiliar em tempo de execução (Figura 3), exatamente como a DLL *proxy* de referência (tanto para arquiteturas $\times 86$ quanto para $\text{amd}64$). A figura 2 apresenta, da coluna esquerda para a direita, a representação em escala de cinza baseada em bytes das amostras originais; das DLLs *proxy*, com todas as exportações de encaminhamento; e das amostras manipuladas pela ferramenta WrappEm. Essa representação em texturas é mais resiliente para classificação de malwares [Nataraj et al. 2011] e requer que a largura da imagem seja fixa, podendo a altura variar dependendo do tamanho do arquivo, conforme a tabela 1.

Embora uma análise sistemática de evasão de scanners de múltiplos motores esteja planejada para validação futura, a divergência matemática e estrutural da amostra

manipulada pela ferramenta em relação ao binário original é evidencialmente menor que a divergência entre a DLL *proxy* e ao binário original (observada na figura 2). Portanto, espera-se que os métodos implementados apresentem taxas de detecção mais baixas em comparação com a DLL *proxy* de referência.

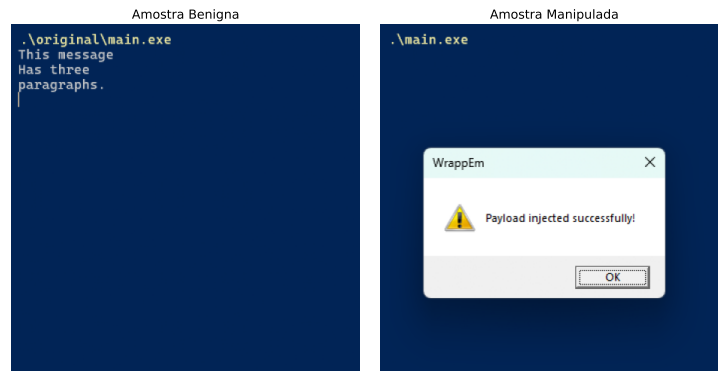


Figura 3. Resultado da execução da amostra original e da amostra manipulada.

4. Conclusão

Este trabalho demonstra que, assim como as técnicas tradicionais baseadas em *proxy*, os métodos de injeção de carga auxiliar baseados em manipulação em nível de byte podem ser utilizados para explorar o Windows Image Loader em ataques adversariais. No entanto, ao contrário das tradicionais, a manipulação do Import Directory e da Import Section evita deixar vestígios passíveis de detecção, uma vez que não requer redirecionamento (i.e. encaminhamento de exportação), o qual gera artefatos com divergências evidentes do original (Figura 2). Como próximos passos imediatos, pretende-se implementar e executar testes de evasão de soluções de segurança em todas as amostras selecionadas para quantificar os resultados.

Referências

- Aryal, K., Gupta, M., Abdelsalam, M., and Saleh, M. (2024). Intra-section code cave injection for adversarial evasion attacks on windows pe malware file.
- Cheng, B., Ming, J., Leal, E. A., Zhang, H., Fu, J., Peng, G., and Marion, J.-Y. (2021). *Obfuscation-Resilient Executable Payload Extraction From Packed Malware*. USE-NIX Association.
- Demetrio, L., Coull, S. E., Biggio, B., Lagorio, G., Armando, A., and Roli, F. (2021). Adversarial examples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM TOPS*.
- Fernández-Álvarez, P. and Rodríguez, R. J. (2023). Module extraction and dll hijacking detection via single or multiple memory dumps. *Forensic Science International: Digital Investigation*, 44.
- Hauser, M. and Cheptsov, V. (2020). Securing the edk ii image loader. In *Proceedings - 2020 Ivannikov Ispras Open Conference, ISPRAS 2020*, pages 16–25. Institute of Electrical and Electronics Engineers Inc.

- Lad, S. S. and Adamuthe, A. C. (2022). Improved deep learning model for static pe files malware detection and classification. *International Journal of Computer Network and Information Security*, 14:14–26.
- Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J., and Wu, Y. (2023). Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. *Computers and Security*, 128.
- Microsoft (2010). Microsoft portable executable and common object file format specification. Technical report, Microsoft Corporation.
- Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: Visualization and automatic classification. *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, pages 1–7.
- Nguyen, M. T., Nguyen, V. H., and Shone, N. (2024). Using deep graph learning to improve dynamic analysis-based malware detection in pe files. *Journal of Computer Virology and Hacking Techniques*, 20:153–172.
- Nisi, D., Graziano, M., Fratantonio, Y., and Balzarotti, D. (2021). Lost in the loader: The many faces of the windows pe file format. In *ACM International Conference Proceeding Series*, pages 177–192. Association for Computing Machinery.
- Rizvi, S. K. J., Aslam, W., Shahzad, M., Saleem, S., and Fraz, M. M. (2022). Proud-mal: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable. *Complex and Intelligent Systems*, 8:673–685.
- Yosifovich, P., Russinovich, M. E., Solomon, D. A., and Ionescu, A. (2017). *Windows internals. Part 1: System architecture, processes, threads, memory management, and more*. Microsoft Press.
- Yousuf, M. I., Anwer, I., Riasat, A., Zia, K. T., and Kim, S. (2023). Windows malware detection based on static analysis with multiple features. *PeerJ Computer Science*, 9.
- Zhang, Y., Chang, X., Lin, Y., Misic, J., and Misic, V. B. (2020). Exploring function call graph vectorization and file statistical features in malicious pe file classification. *IEEE Access*, 8:44652–44660.