Breaking Good: Injeção de *Payloads* Legítimos em Binários Maliciosos para Teste de Robustez de Antivírus contra Evasão

Gabriel Lüders[1], **Marcus Botacin**[1], **Fabrício Ceschin**[1], **André Grégio**[1]

[1]Departamento de Informática
Universidade Federal do Paraná (UFPR)

{gl19, mfbotacin, fjoceschin, gregio}@inf.ufpr.br

Abstract. *Antiviruses (AVs) are important defensive solutions (very often the only available way to protect a system against threats). Therefore, AV testing should consider multiple other aspects than just the detection rate accomplished from known-bad samples scanning. Although the detection rate is a popular metric, AVs resistance against attacks to their own engines is usually overlooked. To bridge this gap, we present a solution able to generate evasive samples, i.e., malware disguised as benign software, and submit them to a committee of AVs so as to verify their robustness. We implemented our proposed solution in Python, enabling multiple techniques for payload injection on it, either into malicious files, as well as into benign ones.*

Resumo. *Antivírus (AVs) são importantes soluções de proteção e, muitas vezes, a única defesa de um sistema contra ameaças. Portanto, o teste de AVs deve considerar diversos aspectos que não apenas a taxa de detecção contra códigos conhecidamente maliciosos. Embora a referida taxa seja uma métrica popular, a resistência dos AVs em relação a ataques contra eles próprios é frequentemente negligenciada. Para suprir esta lacuna, apresentamos uma solução capaz de gerar exemplares evasivos, isto é, códigos maliciosos disfarçados de benignos, e submetê-los a um comitê de AVs de forma a avaliar a robustez destes. A solução proposta é implementada em Python e aplica múltiplas técnicas de injeção de conteúdo, tanto em binários benignos quanto em maliciosos.*

1. Introdução

Diversos ataques ocorrem no ciberespaço a todo momento, o que faz com que soluções de defesa se tornem essenciais e ganhem papel de destaque. Antivírus (AVs) são atualmente a principal solução de defesa para a maioria dos usuários. A avaliação de AVs é essencial para garantir um nível adequado de proteção contra ameaças. Atualmente, avaliações são focadas na capacidade de detecção de novas ameaças enquanto aspectos como a robustez contra variações de ataques conhecidos são negligenciados [Botacin et al. 2020].

Ataques empregando variantes de exemplares de *malware* conhecidos ficam cada vez mais comuns a medida que a corrida armamentista entre atacantes e defensores avança. Ao mesmo tempo em que a automatização de tarefas possibilita ampliar as capacidades de defesa, também possibilita a criação de novos ataques em um curto espaço de tempo. De um lado, agentes de defesa, tais como empresas de antivírus (AVs), pesquisam algoritmos que automatizem seus processos, tais como o uso de aprendizado de má-

quina [Ceschin et al. 2018]. De outro lado, os atacantes automatizam o processo de criação de novas variações de ataques, em resposta à atualização dos mecanismos de defesa, afim de aumentar a eficiência desses ataques, já que possivelmente alguma dessas variantes pode evadir certas soluções com simples alterações nos binários [Ceschin et al. 2019]. Geralmente, essas alterações são feitas inserindo características de binários benignos em binários maliciosos, sem que se altere o comportamento original do ataque, resultando em binários chamados de adversários. Logo, é importante que os antivírus tenham ciência desse tipo de ataque e façam uso dessas variações, geradas por ferramentas de injeção de *payloads* como esta, a seu favor, afim de aumentar suas taxas de detecção.

Visando suprir esta lacuna de avaliação e desenvolvimento, propomos uma solução automática de geração de variantes de *malware* para que pesquisadores de segurança possam avaliar a robustez de soluções antivírus de forma automática e escalável. A solução proposta é capaz de modificar exemplares de *malware* adicionando *bytes* aleatórios do sistema ou *strings* binárias de arquivos não maliciosos de modo a enviesar soluções de antivírus em direção a classificação do exemplar como benigno. Além disso, a solução é capaz de realizar *swap* de instruções (trocá-las de lugar) e substituir *flags* de *debuggers*, como as instruções `int3`, por conjuntos de outras instruções que não alteram o comportamento original do executável.

Análises preliminares demonstram que muitos antivírus no mercado não são capazes de detectar uma ou mais dessas mudanças de forma estática, permitindo livre circulação desses arquivos modificados e, potencialmente, representando grande risco a sua base de usuários. Como forma de aumentar a robustez dos AVs, sugerimos o uso de nossa ferramenta para incrementar a base de dados dos sistemas de detecção, seja dos AVs comerciais ou das soluções acadêmicas, através da geração de variações de *malware*, ampliando a cobertura de segurança.

O restante deste trabalho está organizado da seguinte maneira: Na Seção 2, posicionamos nossa solução em meio aos trabalhos relacionados; na Seção 3, apresentamos detalhes sobre a implementação da nossa ferramenta; na Seção 4, apresentamos o uso de nossa solução; finalmente, apresentamos nossas considerações finais na Seção 5.

2. Trabalhos Relacionados

A evasão de soluções antivírus não é uma área de pesquisa inédita. Há muito se sabe que, por exemplo, criptografar *payloads* maliciosos leva ao *bypass* de detectores estáticos [Tasiopoulos and Katsikas 2014]. Apesar disto, esta área tem ganhado cada vez mais notoriedade devido ao aumento do número e da complexidade dos ataques a sistemas computacionais.

Ao longo do tempo, sistemas de detecção de ameaças evoluíram para cobrir a detecção de exemplares de *malware* polimórficos [Alam et al. 2014], embora com efetividade ainda limitada. Com a adoção de detectores baseados em aprendizado de máquina, a ameaça imposta por exemplares polimórficos foi realçada, devido a ataques do tipo adversário [Ceschin et al. 2019].

Neste cenário, é imprescindível desenvolver ferramentas de geração de códigos polimórficos para que se possa testar as soluções de detecção de modo efetivo. Infelizmente, há poucas soluções existentes. Além disto, estas são muito limitadas. Por exem-

plo, algumas soluções operam apenas sobre códigos-fonte [R1kk3r 2019], o que limita a aplicação destas em binários já existentes.

As soluções mais recentes para a geração de exemplares adversários se baseiam no uso de técnicas de aprendizado de máquina, como aprendizado por reforço [Filar 2020]. Uma limitação deste tipo de técnica é que o analista/usuário perde controle das modificações introduzidas. Desta forma, propomos neste trabalho uma solução prática, que opere sobre binários compilados, e que permita ao analista total controle sobre as alterações introduzidas em binários já existentes.

3. Projeto e Implementação

Nesta seção, apresentamos a ferramenta desenvolvida e descrevemos a implementação e o fluxo de execução da mesma.

3.1. Arquitetura

A ferramenta proposta foi desenvolvida de forma modular, de modo a permitir o reúso de componentes e facilitar a manutenção e evolução de código. A Figura 1 ilustra os diferentes módulos e componentes desenvolvidos, bem como o fluxo de dados ao longo da solução. Este se dá da seguinte maneira: (i) Uma cópia do arquivo malicioso passado como entrada é criado e um *hash* MD5 desta é criado, visando garantir a integridade do arquivo original; (ii) Após a validação dos arquivos gerados, as sub-rotinas diretamente responsáveis por realizar as modificações nos binários, como a injeção de *payloads* benignos, são executadas; e (iii) ao final, os arquivos originais e modificados são submetidos a um comitê de AVs (VirusTotal) e as taxas de detecção destes são comparadas.

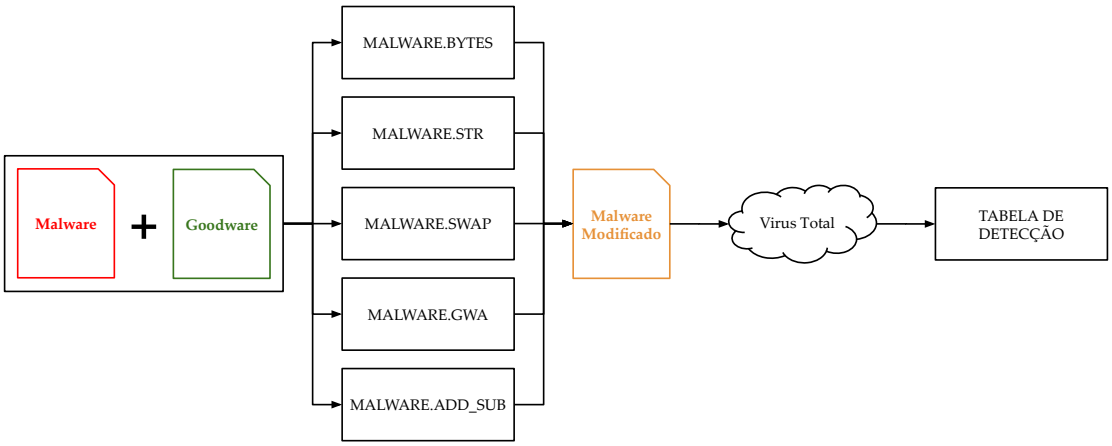

Figura 1. Fluxograma de execução da ferramenta proposta.

As funções responsáveis pelas modificações são definidas em módulos específicos, detalhados a seguir:

Append Data. Este módulo é responsável por adicionar dados de aplicações legítimas aos binários maliciosos de modo a confundir os detectores. Neste módulo, implementa-se os seguintes métodos:

- *Append Raw Data:* Esta rotina adiciona dados brutos (*raw*) aos binários. Sua implementação se dá pelo uso direto da função `os.urandom` para se obter uma

sequência de (pelo menos 50) *bytes* aleatórios. Esta *string* é adicionada ao final do arquivo malicioso (*append*) de modo a não interferir com a operação original do binário. Um resumo desta operação é ilustrado pela Figura 2.

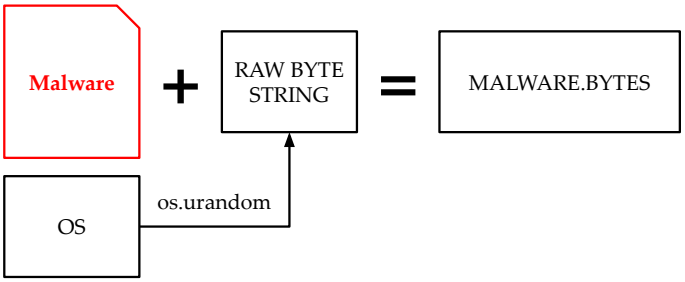

Figura 2. Fluxograma da função `Append Raw Data`.

- *Append Goodware Strings:* Este módulo adiciona *strings* de *goodware* ao binário. Espera-se que as *strings*, por serem dados mais qualificados que dados brutos (*raw bytes*) possam ter uma influência maior sobre os classificadores. A implementação deste módulo se dá pela execução do comando `strings` do *bash* diretamente sobre um arquivo benigno fornecido pelo usuário como entrada. Todas as *strings* presentes no binário original são *appendadas* ao binário malicioso, tal qual exemplificado na Figura 3.

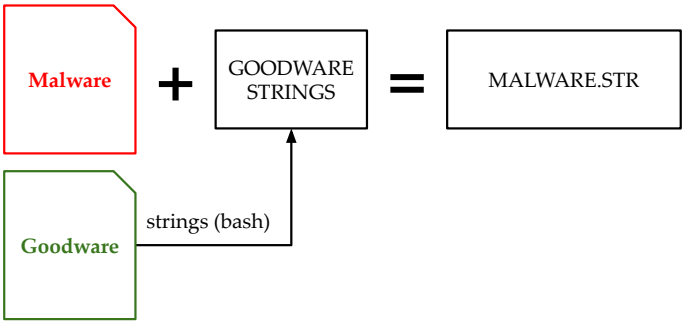

Figura 3. Fluxograma do método `Append Goodware Strings`.

Malware Disassembly. Este módulo é responsável por realizar o *disassembly* dos binários e retornar as instruções identificadas em um formato conveniente para manipulação. Este módulo atua como um auxiliar para as operações de mutação de instruções dos binários, embora também possa ser executado diretamente. O *disassembly* é realizado através do *framework* `Capstone` [Anh Quynh et al. 2019]. Por visarmos implementar polimorfismo de código, optou-se por realizar o *disassembly* apenas das seções executáveis de código. Estas são identificadas pelas *flags* `0x00000020` e `0x20000000` que indicam, respectivamente, a existência de código na seção e a presença de uma seção executável. Estas *flags* são identificadas através da solução `Pefile` [Carrera 2019]. O fluxo completo de execução deste módulo é ilustrado na Figura 4.

Este módulo implementa os seguintes métodos para geração de códigos polimórficos:

- *Swap RET with NOP:* Esta função inverte o posicionamento das instrução `RET` e `NOP` quando estas estão posicionadas de modo consecutivo em um binário. Este

Figura 4. Fluxo de dados do módulo `Malware Disassembly`.

tipo de modificação não altera o funcionamento do binário, mas é capaz de afetar detectores baseados em `hash` e/ou em assinaturas se estas instruções forem consideradas. Este módulo também é capaz de alterar uma sequência de instruções destes tipos, tendo como base um *offset* fornecido pelo usuário.

- *Replace INT3 with ADD_SUB:* Esta função modifica instruções de modo a evadir detectores de forma similar à função apresentada anteriormente. Contudo, esta função implementa modificações mais complexas do que o *swap* de instruções de modo a aumentar a diversidade de código. Mais especificamente, a estratégia aqui adotada é a de substituir instruções inefetivas por um conjunto de instruções diferentes que não produza efeito colateral. Em particular, substitui-se as instruções `INT3`, pois estas são utilizadas apenas quando o binário está sendo depurado, o que não é o cenário de uso aqui apresentado. Como exemplo de um conjunto de instruções que não produz efeito colateral, considere a adição e subsequente subtração de um valor fixo em um registrador. Consideramos, por exemplo, a sequência de instruções `ADD (ebp+4), 1` e `SUB (ebp+4), 1`, dado a elevada frequência desta construção em binários legítimos (embora sem garantias de funcionamento em binários cuja compilação não gerou um *frame pointer*). Como desafio de implementação, destacamos que as instruções `INT3` e `ADD` e `SUB` possuem diferentes tamanhos (1, 4 e 4 *bytes*, respectivamente). Desta forma, cada conjunto `ADD_SUB` é substituído apenas quando uma sequência de oito instruções `INT3` é encontrada no binário.
- *Append Goodware Sections:* Esta função é responsável por adicionar código legítimo ao binário malicioso através da criação de uma novação seção no binário. Esta abordagem tem como vantagem, em relação as anteriores, que qualquer tipo de instrução, sem limitações de tamanho, pode ser adicionada. Este tipo de mecanismo visa evadir classificadores de aprendizado de máquina que utilizam o número de seções presentes no binário como atributo. A função foi implementada fazendo uso de um injetor de seções já existente [Cheron 2017].

AVTester. Este módulo é responsável por implementar a consulta das taxas de detecção dos exemplares gerados através de um comitê de AVs. Este módulo é implementado através da API do serviço `VirusTotal` (VT) [Chronicle Security 2020]. A Figura 5 ilustra o fluxo de dados quando da execução deste módulo: uma vez executadas todas as funções de modificação do *malware*, todos os arquivos gerados são postados um a um para conseguinte análise no VT. A *API* retorna uma série de dicionários relacionadas a cada arquivo, permitindo análises dos impactos que cada modificação teve na detecção ou não do arquivo malicioso. Ressaltamos que este trabalho visa a evasão de detectores estáticos, que estão disponíveis no VT. Apesar disso, destacamos que os exemplares gerados podem ser detectados por componentes dinâmicos que são executados pelos AVs apenas quando instalados em *endpoints*, embora os AVs disponíveis no VT se aproximem com bastante razoabilidade daqueles instalados localmente [Botacin et al. 2020].

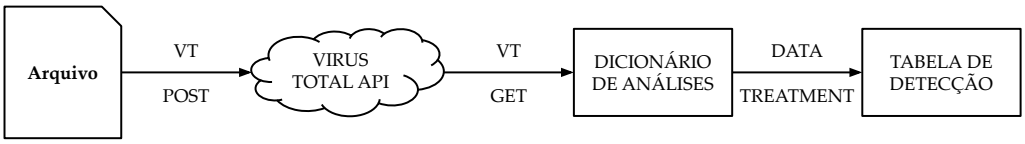

Figura 5. Fluxo de dados do módulo de detecção por comitê utilizando da API do `VirusTotal`.

Data Treatment. Este módulo é responsável por tratar os dicionários retornados pelo VT e representá-los na forma de uma tabela, de modo a permitir a comparação das taxas de detecção antes e depois das modificações.

4. Aplicação

Nesta seção, apresentamos como a solução desenvolvida pode ser obtida, instalada e utilizada.

Obtenção da ferramenta. A ferramenta pode ser obtida diretamente do repositório: `https://github.com/ludersGabriel/BreakingGood`.

Instalação da ferramenta. Para instalar a ferramenta, deve-se executar o *script* `install.sh`. É necessário também ter as seguintes dependências instaladas: `Python3.6` com `pip3` e `Python2.7` com `pip2`. Deve-se também adicionar a chave de usuário do serviço VT no arquivo `vtapi.py`.

Utilização da ferramenta. A ferramenta pode ser executada com o seguinte comando: `python3 main.py <diretório de saída> <caminho do malware> <caminho do goodware>`. O usuário deverá visualizar uma saída como a ilustrada na Tabela 1. A primeira coluna é referente ao arquivo original (ORIG). As seguintes correspondem respectivamente as modificações do tipo *Append Raw Data (BYTES)*, *Append Goodware Strings (STR)*, *Swap RET with NOP (SWAP)*, *Replace INT3 with ADD_SUB (ADD_SUB)* e *Append Goodware Sections (GWA)*.

Tabela 1. Exemplo de saída da ferramenta quando executada com o exemplar cujo *hash* MD5 é `3dd221b0ea6f863e086868b246a6a104`.

AV	ORIG	BYTES	STR	SWAP	ADD_SUB	GWA
Bkav	True	True	True	True	True	False
DrWeb	True	True	True	True	True	True
MicroWorld-eScan	True	True	True	True	True	False
FireEye	True	True	True	True	True	True
McAfee	True	True	True	True	True	False
Cylance	True	True	True	True	True	True
Zillya	True	True	True	True	True	False
SUPERAntiSpyware	False	False	False	False	False	False
Sangfor	False	False	False	False	False	False
K7AntiVirus	True	True	False	True	False	False
Alibaba	True	False	False	True	True	False

Na Listagem 1, exemplifica-se a diferença entre os binários do arquivo original e sua variante, obtida após aplicação do módulo *Replace INT3 with ADD_SUB* em um arquivo malicioso. Já na Listagem 2, exemplifica-se as diferenças alcançadas nos binários dos arquivos resultantes da aplicação do módulo *Swap RET with NOP* no mesmo *malware* supracitado. Vale a pena ressaltar que, caso as mudanças-alvo de um módulo não sejam possíveis de serem efetivadas ou causem alguma corrupção, a integridade do arquivo original é mantida (i.e., o arquivo resultante da aplicação de qualquer um dos módulos é uma cópia do original passado como entrada).

```
MD5.original          | MD5.addSub
0x25328: ret 0xc      | 0x25328: ret 0xc
0x25331: int3         | 0x25331: add dword ptr [ebp -4], 1
0x25332: int3         |
0x25333: int3         |
0x25334: int3         |
0x25335: int3         | 0x25335: sub dword ptr [ebp -4], 1
0x25336: int3         |
0x25337: int3         |
0x25338: int3         |
0x25339: int3         | 0x25339: int3
```

Listagem 1. Diferença do disassembly da região alvo do arquivo original (60ed52917b7fa2332e736f0126f12af1) e do arquivo resultante utilizando a técnica *Replace INT3 with ADDSUB*.

```
MD5.original          | MD5.swap
0x11277: leave        | 0x11277: leave
0x11278: ret 4        | 0x11278: nop
0x11281: nop          | 0x11279: ret 4

0x28062: leave        | 0x28062: leave
0x28063: ret          | 0x28063: nop
0x28064: nop          | 0x28064: ret
0x28065: cli          | 0x28065: cli
```

Listagem 2. Diferença do disassembly da região alvo do arquivo original e do arquivo resultante utilizando a técnica *Replace RET with NOP*.

5. Conclusão

Neste trabalho, apresentamos uma solução automatizada para a criação de exemplares de *malware* evasivos baseada na injeção de *payloads* benignos. O objetivo é possibilitar a realização de testes de robustez adequados para mecanismos de segurança baseados em varredura estática de binários e casamento de assinaturas. Diversas técnicas foram estudadas e avaliadas em um comitê de antivírus a fim de validar a solução proposta. Os resultados preliminares indicam que há um campo em aberto para a aplicação deste tipo de solução nos mecanismos existentes, pois muitas vezes os testes realizados nos mecanismos de segurança lidam com amostras mais diretas (não evasivas, como aqui

apresentado). Como trabalho futuro, planejamos estender a ferramenta para lidar com mais exemplares em paralelo e possibilitar a geração de gráficos das taxas de detecção. Finalmente, espera-se que este trabalho fomente o desenvolvimento de mecanismos de segurança mais robustos, bem como incentive testes mais abrangentes para esse tipo de mecanismo.

Referências

Alam, S., Traore, I., and Sogukpinar, I. (2014). Current trends and the future of metamorphic malware detection. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, page 411–416, New York, NY, USA. Association for Computing Machinery.

Anh Quynh, N., Sheng Di, T., Nagy, B., and Hoang Vu, D. (2019). Capstone engine. `https://www.capstone-engine.org/`.

Botacin, M., Ceschin, F., de Geus, P., and Grégio, A. (2020). We need to talk about antiviruses: Challenges & pitfalls of av evaluations. *Computers & Security*, page 101859.

Carrera, E. (2019). Pefile python handler. `https://pypi.org/project/pefile/`.

Ceschin, F., Botacin, M., Gomes, H. M., Oliveira, L. S., and Grégio, A. (2019). Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors. In *Proceedings of the 3rd Reversing and Offensive-Oriented Trends Symposium*, ROOTS'19, New York, NY, USA. Association for Computing Machinery.

Ceschin, F., Pinage, F., Castilho, M., Menotti, D., Oliveira, L. S., and Gregio, A. (2018). The need for speed: An analysis of brazilian malware classifers. *IEEE Security Privacy*, 16(6):31–41.

Cheron, A. (2017). Code injection with python. `https://axcheron.github.io/code-injection-with-python/`.

Chronicle Security (2020). Virus total api. `https://developers.virustotal.com/reference#getting-started`.

Filar, B. (2020). Malware bypass research using reinforcement learning. `https://github.com/bfilar/malware_rl`.

R1kk3r (2019). Obfuscator-llvm. `https://github.com/obfuscator-llvm/obfuscator/wiki`.

Tasiopoulos, V. G. and Katsikas, S. K. (2014). Bypassing antivirus detection with encryption. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, page 1–2, New York, NY, USA. Association for Computing Machinery.