SACI: Solução para Análise Comportamental Automatizada de Código Infeccioso em SO MS Windows Modernos

Bernardo Tomasi[1], Davi C. Ribeiro[1], Pedro Friedrich[1], Ruibin Mei[1], Yago Furuta[1], Jorge Correia[1], André Grégio[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
SecRET (`secret.inf.ufpr.br`) – Curitiba – PR – Brasil

{bpt22, dcr23, phfr24, rm23, yyvf22, jpcorreia, gregio}@inf.ufpr.br

Abstract. *Although operating systems evolve their defenses, malicious code remains the main way to infect users. In this article, we propose SACI, a solution for automated behavioral analysis of suspicious executables for MS Windows 10 and 11. Through a filter driver and an infrastructure that promotes high availability and scalability, SACI allows obtaining information about malware's lifetime on the monitored system, as well as provides data on its security tokens, I/O operations, Registry changes, process creation tracking and use of threads.*

Resumo. *Embora os sistemas operacionais evoluam suas defesas, os códigos maliciosos continuam sendo a principal forma de infectar usuários. Neste artigo, é proposto o SACI, uma solução para análise comportamental automatizada de executáveis suspeitos para MS Windows 10 e 11. Por meio de um filter driver e uma infraestrutura que promove alta disponibilidade e escalabilidade, SACI permite a obtenção de informações a respeito do tempo de vida do malware no sistema monitorado, além de fornecer dados sobre os seus tokens de segurança, operações de E/S, alterações no Registro, rastreamento da criação de processos e a utilização de threads.*

1. Introdução

A segurança dos sistemas e seus usuários é constantemente desafiada por exemplares de *malware* cada vez mais sofisticados e evasivos. A evolução dos sistemas operacionais nem sempre é suficiente para impedir esse tipo de infecção, ainda mais se o *malware* conseguir acesso privilegiado ao alvo. Ferramentas públicas para análise dinâmica de *malware* sofrem de instabilidade e dificuldade de manutenção causadas por uso de mecanismos de monitoração não-nativos e funções não documentadas, incorrendo em sua descontinuidade. Este artigo apresenta o SACI, uma Solução para Análise Comportamental Automatizada de Código Infeccioso com abordagem baseada em *driver* de filtro e arquitetura modular baseada em microsserviços, cujo foco é na alta disponibilidade, escalabilidade e automatização. O SACI permite registrar atividades maliciosas de forma inovadora, possibilitando análise de ataques como escalada de privilégio por meio da monitoração de *tokens* de segurança, aumentando assim a capacidade de identificar eventos infecciosos que visam manipular as credenciais de segurança do Windows.

2. Trabalhos Relacionados

Nesta seção, discute-se criticamente os trabalhos mais próximos deste, com ênfase nas diferenças das soluções propostas quando comparadas ao SACI.

[Ribeiro et al. 2020] e [Figueiredo et al. 2022] propõem analisadores baseados no Cuckoo Sandbox [Guarnieri 2013], uma ferramenta para análise dinâmica de *malware* em sistemas Linux, Darwin e Windows. Em relação as informações coletadas, nota-se como principais diferenças que o Cuckoo permite a captura de tráfico de rede e obtenção de *dump* da memória do sistema monitorado (ainda não suportadas pelo SACI), porém não permite monitorar *tokens* de acesso, um dos principais atrativos do SACI. Além disso, o monitoramento em sistemas MS Windows nas iterações mais recentes do Cuckoo é feito por meio de *hooks* em modo de usuário, cujos problemas são discutidos na Seção 3.4.1 e agravados pelo fato do projeto estar inativo e descontinuado. Em contrapartida, o SACI tem uma forma de monitoramento atualizada de acordo com a documentação vigente do MS Windows, com projeto que visa o suporte à longo prazo. [Botacin et al. 2021] também desenvolveram uma ferramenta Web (Corvus) que suporta análise dinâmica de executáveis de sistemas MS Windows por meio de um *driver* de filtro [Botacin et al. 2018]. No entanto, a ferramenta não inclui análise de tokens de acesso, tampouco um filtro para carregamento de imagens, como é o caso do SACI. Além disso, a plataforma Corvus está fora do ar e era compatível apenas com Windows 7 e 8. Já [Souza and Silva 2021] propuseram Freki, que utiliza um método de análise dinâmica com base num processo de depuração supervisionada, no qual as instruções propriamente executadas pelo agente malicioso são identificadas. Já o SACI foca em mostrar os efeitos que um dado malware tem de fato no sistema, ao relacionar diferentes subsistemas do *kernel* do Windows com os resultados obtidos na monitoração.

3. Arquitetura e Implementação do SACI

Projetou-se SACI como uma arquitetura modular composta por microsserviços de análise e *frontend*, gerenciados por um orquestrador de *container*. Os ambientes de execução são provisionados pelo *Proxmox* [Proxmox 2024] e pré-carregados com um *daemon* cuja função é iniciar o *driver* de monitoração, fazer requisições de eventos e criar os registros dos resultados, e um *driver* de filtragem com quatro filtros e um componente de comunicação que fazem a captura e transmissão das informações coletadas para o *daemon*.

3.1. Microsserviços

A arquitetura do SACI, ilustrada na Figura 1, é baseada em microsserviços, isto é, componentes independentes e interconectados responsáveis por diferentes funcionalidades da aplicação, permitindo uma distribuição eficiente das tarefas e uma escalabilidade modular da aplicação. Cada microsserviço desempenha um papel específico na cadeia de processamento de análise de *malware*, garantindo um sistema robusto e ágil para os usuários, bem como a facilidade de manutenção e atualização contínuas.

Frontend. Este microsserviço constitui a interface de usuário da aplicação SACI e integra todas as funcionalidades disponíveis para os usuários finais, incluindo o envio de arquivos executáveis para análise. O *Frontend* atua como o ponto de entrada para os usuários, encaminhando as amostras de *malware* para o microsserviço responsável pela análise.

Análise. Este microsserviço é o núcleo da funcionalidade do SACI, e implementa rotas para receber os arquivos enviados pelos usuários através do *Frontend* e para a retirada de relatórios. Além disso, esse microsserviço se comunica com a API do Proxmox para pedir um ambiente de execução.

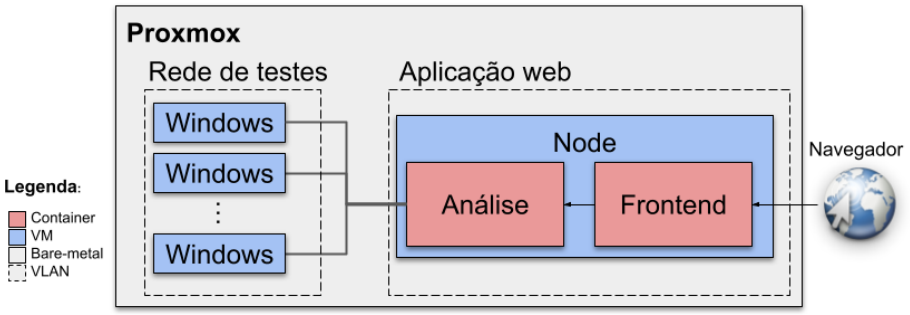

Figura 1. Arquitetura da aplicação SACI.

Ambiente de execução. A plataforma Proxmox é utilizada para virtualização sob demanda. Ela permite a criação de novas máquinas virtuais a partir de um *template* configurado com o necessário para a execução controlada das amostras de malware. Este ambiente, apagado ao final da execução, inclui o *driver* de filtro, que monitora a execução do malware e registra as atividades relevantes em um formato JSON.

3.2. Orquestração de containers

A arquitetura de microsserviços do SACI requer um agente orquestrador de *containers* para coordenar a execução dos componentes da aplicação Web de forma que estes se comuniquem. Cada microsserviço pode sofrer uma carga variável de requisições, fazendo com que os com maior demanda tenham mais réplicas do container distribuído nos *Nodes*. O orquestrador também possui um balanceador de carga que distribui as requisições entre as réplicas, diminuindo gargalos e aumentando a escalabilidade das requisições. Além disso, caso um container precise ser atualizado, o orquestrador consegue atualiza-lo individualmente, de modo que a aplicação Web tenha alta disponibilidade.

3.3. Fluxo do *driver*

A seguir, explica-se o fluxo de dados da arquitetura do SACI. A Figura 2 ilustra o fluxo da interação entre a amostra monitorada pelo *driver* de filtro (setas vermelhas) e o componente *daemon* do SACI (setas verdes).

Fluxo Malware -> *driver* de Filtro (setas vermelhas):

1. O *malware* submetido ao SACI será executado no ambiente virtualizado e irá começar a interagir com o SO, mais especificamente com o Executivo do Windows.
2. Os filtros irão receber notificações sobre as ações do *malware* dos componentes executivos ao qual este está associado.
3. O *driver* cria uma representação interna do evento reportado e anexa à lista de eventos.

Fluxo SACI *daemon* -> *driver* de Filtro (setas verdes):

1. O SACI Daemon irá fazer requisições de eventos para o *driver* por meio de interrupções (IRP) para as quais Executivo do Windows irá criar uma lista de entrada e saída usada para o retorno dos dados monitorados.
2. O *driver*, ao receber a IRP, retira o evento mais antigo da lista configurada pelo *IO Control* e copia seus componentes para o *buffer* provido pelo *Daemon*.
3. Uma vez que o *driver* complete a IRP, o *Daemon* pode então acessar seu *buffer*, para onde os dados de um evento foram copiados, e armazená-los em um JSON.

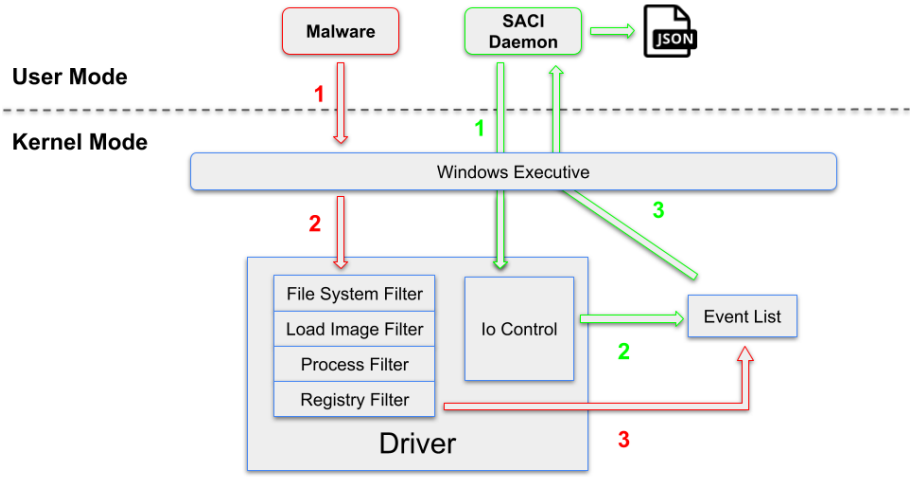

Figura 2. Fluxo das interações entre *malware*, SACI *Daemon* e *driver* de filtro.

3.4. Componentes do *driver*

Filtro de Sistema de Arquivos. Toda vez que uma aplicação em modo usuário interage com o sistema de arquivos, o *I/O Manager* cria uma estrutura contendo todas as informações necessárias para a identificação e descrição da operação solicitada. Essa estrutura, denominada *I/O Request Packet* (IRP), então, percorre a *device stack* e passa de *driver* à *driver* até que requisição é finalizada por um dos *device objects*, que ou chama a função *IoCompleteRequest()* ou registra uma rotina de finalização através da função *IoSetCompletionRoutineEx()*. Nesse processo, o *filter driver* do SACI intercepta determinadas IRPs, de forma indireta, pelos callbacks registrados no *filter manager*; cria uma estrutura interna e, através do parâmetro FLT_CALLBACK_DATA dos callbacks, obtêm os dados relevantes que incluem: o nome do arquivo envolvido, a major function, o pid do processo que realizou a requisição, o pid do processo pai, thread id, session id, path completo do executável, bem como os tokens de segurança. A respeito dos IRPs, o *filter driver* do SACI foca nos três principais e mais comuns *I/O Request Packets*: IRP_MJ_CREATE, IRP_MAJOR_READ e IRP_MAJOR_WRITE, que correspondem às operações de criação, leitura e escrita, respectivamente. Assim, é possível obter informações crucias a respeito do funcionamento de malwares ao observar como esses processos maliciosos interagem com o sistema de arquivos e impactam o sistema como um todo.

Filtro de Criação Processos e Threads. Em sistemas Windows, cada processo é representado por uma estrutura opaca interna ao *kernel*. Essa estrutura, conhecida como EPROCESS, embora não formalmente documentada, apresenta uma série de atributos que tanto identifica quanto caracteriza um dado processo. Nesse contexto, o *filter driver* do SACI, através dos handles recebidos pelos callbacks registrados nas funções *PsSetCreateProcessNotifyRountine()* e *PsSetCreateThreadNotifyRoutine()*, obtêm a estrutura EPROCESS a partir da função *PsLookupProcessByProcessId()*. Com isso, uma estrutura interna é criada contendo os tokens de segurança, bem como o path completo de tanto o processo recém criado ou deletado, quanto o path completo do processo pai. A obtenção dos paths ocorre por meio da estrutura EPROCESS através da função *SeLocateProcessImageName()*. Em relação ao monitoramento de threads, por sua vez, apenas um único PID e path são reportados, correspondendo ao processo que a thread atual está execu-

tando. Nesse contexto, o *filter driver* do SACI tem um papel importante na detecção da criação de processos por processos maliciosos. Além disso, a partir das funcionalidades relacionadas à criação e deleção de threads, junto do filtro de Bibliotecas Dinamicamente Ligadas (DLLs), é possível a identificação do uso de threads remotas por agentes maliciosos.

Filtro de Registros. Os registros do Windows são uma base de dados hierárquica que é usada por componentes do sistema e aplicações, para armazenar dados de configurações. Os arquivos de registro estão espalhados pelo sistema, e são carregados em memória durante o boot, pelo *Configuration Manager*, componente do executivo do Windows. Por isso há a necessidade de um monitoramento dos registros separados do sistema de arquivos, além do fato de que os registros voláteis nunca adentram o disco. A monitoração dos registros é fundamental para a análise de *malware*, pois estes são alvos de várias técnicas implementadas por códigos maliciosos. Um exemplo é o ransomware CryptoLocker, que esteve em atividade durante 5 de setembro de 2013 até maio de 2014, que fazia modificações nos registros para garantir que o código malicioso fosse executado automaticamente durante a inicialização do sistema. O monitoramento dos registros é feito a partir de um *callback*, que é implementado em cima do mecanismo geral de *callbacks* executivo, que pode ser invocado antes ou depois de uma operação de registro. Este *callback* permite que tenhamos acessos aos *buffers* de entradas e saída fornecidos para as operações. Focamos nossa análise em tipos específicos de operações de registro, `RegNtSetValueKey` e `RegNtDeleteValueKey`, que são mais comumente encontradas em técnicas usadas por códigos maliciosos. Ao ser ativado o *callback* irá armazenar, o caminho do valor modificado, e o conteúdo da modificação.

Filtro de Carregamento de Imagens. Uma imagem é um arquivo executável (EXE), uma DLL, ou um *driver* (`.sys`) que pode ser carregado em memória para que seja executado. O carregamento de uma imagem em memória é essencial para o funcionamento do SO, principalmente do MS Windows que depende fortemente das DLLs para implementação das seus subsistemas(Win32, POSIX, etc.). No entanto, essa mesma funcionalidade também é frequentemente explorada por códigos maliciosos para realizar ataques como injeção de DLL e *Process Hollowing*. Para monitorar tais ataques, registra-se uma rotina por meio da função `PsSetLoadImageNotifyRoutine()`, invocada pela função `PsCallImageNotifyRoutines()`, por sua vez invocada pela função `MapViewOfImageSection()` após ela ter terminado de mapear a imagem em questão na memória, mas antes de retornar o fluxo para a função original (provavelmente a `LoadLibrary()` ou a `CreateProcess()`). Assim, tem-se acesso às informações da imagem carregada antes do ponto de entrada ser chamado. Dessa forma, quando o filtro for ativado, ele irá registrar o caminho da imagem que foi carregada e o processo no qual a imagem foi mapeada, e copiará a estrutura *IMAGE_INFO*, da qual se pode ter acesso a diversas informações adicionais que permitem distinguir carregamentos de rotina e carregamentos maliciosos (por exemplo, *ImageSignatureLevel*, que permite ver o código de integridade associado à imagem).

Tokens. O token de acesso do Windows é um objeto que descreve os atributos (como a identidade) ou as regras de segurança de um processo ou *thread* (como os privilégios do usuário associado), realizando assim uma função significativa na proteção do SO. Por exemplo, quando um usuário faz *login*, o sistema compara a senha digitada com os da-

dos armazenados em um banco de dados. Caso a senha seja válida, é produzido um token de acesso contendo informações de identificador de segurança (SID) da conta do usuário, seus privilégios, o tipo do token (primário ou *impersonation token*), entre outras. Além disso, todo processo executado em nome do usuário possui uma cópia do token de acesso e seus privilégios associados. Assim, sempre que se tenta realizar operações privilegiadas ou acessar recursos do sistema, o token de acesso é usado para verificação e controle de acesso. O identificador de segurança (SID) é fundamental no modelo de segurança do Windows, sendo usado para identificar a entidade de segurança (conta de usuário/computador, processo executado ou *thread* autenticada pelo SO) ou um grupo de segurança. O token primário descreve o contexto de segurança da conta de usuário ligada ao processo. Por padrão do sistema, quando uma *thread* do processo interage com um objeto protegido, esse token é utilizado. Porém quando uma *thread* representa um identificador do cliente com a mesma funcionalidade de interagir com objetos protegidos ele é conhecido como *impersonation token*. Um privilégio pode controlar o acesso a recursos ou das tarefas do próprio sistema, enquanto os direitos de acesso controlam o acesso aos objetos protegidos como os processos e *threads*. Quando o usuário executa uma operação privilegiada, o sistema faz uma verificação no token de acesso para determinar quais privilégios podem ser executados; caso os privilégios não estejam habilitados, o sistema não executará a operação. Neste contexto, o monitoramento dos tokens é crucial para análise da extensão do dano de um *malware* ao sistema alvo, pois muitos deles podem fazer elevação de privilégios e manipulação de tokens durante o processo de infecção.

3.4.1. Por que usar filtro e não *hooking* de chamadas de sistema?

O uso de *hooks* para o monitoramento de códigos maliciosos, seja em modo usuário ou em modo de *kernel*, é uma técnica comumente usada por mecanismos de proteção. Entretanto, existem muitas deficiências de segurança e implementação nesse tipo de técnica: enquanto os *kernel hooks* (como SSDT *hooking* ou *inline patching*) comprometem a representatividade, estabilidade e segurança do ambiente, pois precisam alterar o código do *kernel* e, consequentemente, implicam na desativação de mecanismos nativos de proteção como o KPP (*Kernel Patch Protection*), os *user mode hooks* (como Microsoft Detours) sofrem de problemas diversos para implementação correta e segura, devido à necessidade de *patching* nas bibliotecas nativas dos subsistemas Windows, bem como de visibilidade limitada na monitoração de *malware* privilegiado que interage diretamente com o executivo do Windows. Além disso, a evolução dos mecanismos de segurança do sistema e as formas muitas vezes não documentadas de se fazer *hooking* os tornam facilmente obsoletos. Por outro lado, a implementação do SACI faz uso de mecanismos pré-estabelecidos para realizar o monitoramento do sistema: o uso de filtros específicos baseados em mecanismos nativos do sistema e funções documentadas, atuais e suportadas pela Microsoft permite maior estabilidade na execução do sistema de monitoração e melhor manutenibilidade e suporte à longo prazo.

4. Estudo de Caso

Para avaliar a granularidade da informação capturada por meio do SACI, escolheu-se como amostra de análise o malware *HermeticWiper* [CISA 2022], usado contra a Ucrânia por forças Russas em 2022. Este *malware* tem como objetivo principal a fragmentação e

sobrescrita de arquivos no disco para impossibilitar sua recuperação posterior. Dado que o *HermeticWiper* é um malware multifacetado, o estudo de caso foca no momento inicial da infecção, onde a amostra faz a preparação para fragmentar o disco (Figura 3).

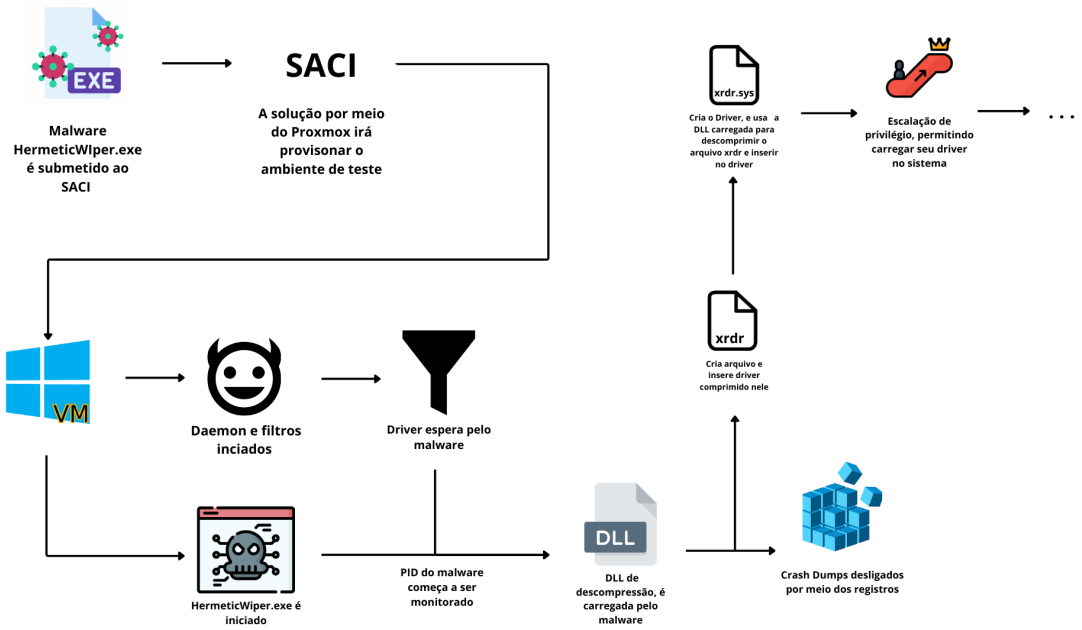

Figura 3. Submissão de amostra ao SACI e etapas iniciais de infecção.

O *malware HermeticWiper* é enviado através do microsserviço de *Frontend* para o de Análise. Ressalta-se que as requisições podem ser distribuídas para diferentes *containers* pelo *load balancer* do orquestrador. Em sequência, o microsserviço de Análise interage com a API do Proxmox para provisionar uma máquina virtual Windows a partir de um *template* com o *driver* já carregado. Portanto, novas mudanças na máquina virtual serão registradas em um *overlay*. Com isso feito, o *malware* é executado remotamente e, ao finalizar, o *driver* irá gerar um *log* em JSON com informações sobre esta execução. A Tabela 1 mostra algumas informações importantes registradas pelo SACI.

No primeiro momento (16:34:38:622), o *malware* carrega uma DLL (lz32.dll), capturada pelo filtro de carregamento de imagens e que, após investigação, descobre-se servir para compressão/descompressão para o malware usar durante a infecção. No segundo momento (16:34:38:637), o filtro de registro notifica sobre uma alteração para 0 no valor do \REGISTRY\MACHINE\SYSTEM\ControlSet001\Control\CrashControl, o que fará com que se o sistema falhar, ele não irá gerar um *crash dump*, dificultando a análise e o diagnóstico do culpado. Em seguida (16:34:38:638) ocorre uma elevação de privilégios, onde o *malware* irá adquirir o privilégio `SeBackupPrivilege`. Logo após, tem-se uma série de interações com o sistema de arquivos (16:34:38:639-16:34:38:655), capturadas pela *callback* registrada com o *Filter Manager*, as quais serão usados para criação de um *driver*, que por sua vez será usado para fazer a fragmentação do disco: ele irá criar o arquivo `xrdr` e, por meio de um WRITE, colocar *driver* comprimido neste arquivo para então criar o arquivo do *driver* de fato (`xrdr.sys`), lendo `xrdr` e usando a DLL carregada no início para descomprimir o arquivo dentro do `xrdr.sys`. Após ter terminado de criar o *driver*, outra elevação de privilégios ocorre (16:34:38:715), esta detectada a partir

dos *tokens* de acesso, onde o malware irá conseguir o `SeLoad`*`driver`*`Privilege` que permite que ele carregue um *driver* no sistema.

Tabela 1. Ações e objetos alvos do *HermeticWiper* no sistema monitorado.

Horário	Ação	Alvo
16:34:38:622	LOAD_DLL	lz32.dll
16:34:38:637	SetKeyValue	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Control\CrashControl
16:34:38:638	PRIVILEGE_ESCALATION	SeBackupPrivilege
16:34:38:639	CREATE	\Device\HarddiskVolume4\Windows\System32*drivers*\xrdr
16:34:38:640	WRITE	\Device\HarddiskVolume4\Windows\System32*drivers*\xrdr
16:34:38:653	CREATE	\Device\HarddiskVolume4\Windows\System32*drivers*\xrdr.sys
16:34:38:654	READ	\Device\HarddiskVolume4\Windows\System32*drivers*\xrdr
16:34:38:655	WRITE	\Device\HarddiskVolume4\Windows\System32*drivers*\xrdr.sys
16:34:39:715	PRIVILEGE_ESCALATION	SeLoad*driver*Privilege

5. Considerações Finais

Este artigo introduziu o SACI, um sistema de análise de dinâmica de *malware* para Windows 10 e 11, com projeto focado em estabilidade e manutenção. Sua arquitetura e implementação, baseados em orquestrador com balanceamento de carga e *drivers* de filtragem, inovam ao permitir a detecção de escaladas de privilégio por meio da monitoração de *tokens*, bem como análise de *malware* privilegiado pela monitoração de *drivers*, coisa que até então não era possível com o estado da arte.

Agradecimentos

Este trabalho teve apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001 e do C3SL - Centro de Computação Científica e Software Livre (`https://www.c3sl.ufpr.br`).

Referências

Botacin, M., Ceschin, F., and Grégio, A. (2021). Corvus: Uma solução sandbox e de threat intelligence para identificação e análise de malware. In *Anais Estendidos do XXI SBSeg*, pages 50–57, Porto Alegre, RS, Brasil. SBC.

Botacin, M., de Geus, P. L., and Grégio, A. R. A. (2018). The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques*, 14:87–98.

CISA (2022). Mar-10375867-1.v1 – hermeticwiper. `https://www.cisa.gov/news-events/analysis-reports/ar22-115a`.

Figueiredo, G. V., Cattelan, R. G., and Miani, R. S. (2022). Sandbox as a service: automatizando a configuracão do cuckoo sandbox e a geração de dados para análise de malware. In *WTICG, SBSeg*. SBC.

Guarnieri, C. (2013). Cuckoo sandbox. https://cuckoo.readthedocs.io/en/latest/.

Proxmox (2024). Proxmox virtual environment. `https://www.proxmox.com`.

Ribeiro, A. d. S., Canedo, E. D., Mendonça, F. L. L., and Junior, R. T. d. S. (2020). Malware analysis using the unbox tool. In *17th International Conference on Information Technology–New Generations (ITNG 2020)*, pages 127–135. Springer.

Souza, C. and Silva, F. (2021). Freki: Uma ferramenta para análise automatizada de malware. In *Anais Estendidos do XXI SBSeg*, pages 58–65, Porto Alegre, RS, Brasil. SBC.