|  |  |
| --- | --- |
| CURSO DE CIÊNCIA DA COMPUTAÇÃO – TCC | |
| ( x  ) PRÉ-PROJETO     (   ) PROJETO | ANO/SEMESTRE: 2024/1 |

**Ferramenta Para Monitoramento De Chamadas de Sistema em Aplicações Obfuscadas ou Virtualizadas em Sistemas Operacionais Windows**

João Vitor de Oliveira

Prof. Danton Cavalcanti Franco Junior - Orientador

# Introdução

Segundo CISO ADVISOR (2023), o número de ataques cibernéticos registrados no ano de 2022 globalmente foi quase dois quintos (38%) maior que o volume total observado no ano de 2021. Devido a este fato, se enfrenta um desafio cada vez maior relacionado à segurança da informação. Ameaçando a integridade, a confidencialidade e a disponibilidade dos sistemas de informação em todo o mundo. Nesse contexto de acordo com a Kaspersky (2013) um dos principais vetores de ataque que tem preocupado a comunidade de segurança cibernética é o uso de rootkits, que muitas vezes se infiltram no núcleo do sistema operacional, explorando técnicas avançadas de hooking para comprometer a segurança e a privacidade dos sistemas.

Ainda segundo a Kaspersky (2013), historicamente os rootkits eram frequentemente desenvolvidos com o intuito de se esconderem no sistema, tornando-se virtualmente invisíveis para as soluções de segurança convencionais. Uma das técnicas mais utilizadas para esse fim era o *hooking* da *System Service Dispatch Table (SSDT*), permitindo que os rootkits interceptassem chamadas de sistema (*syscall*) e, assim, obtivessem controle sobre o sistema operacional (Demirkapi, 2020). O desenvolvimento dessa técnica representou um sério desafio para a segurança cibernética, pois tornava extremamente difícil a detecção e a mitigação de ameaças avançadas. (CYBERARK, 2022).

De acordo com Reginato (2019), à medida que a ameaça dos rootkits cresceu, também evoluiu a resposta da indústria de segurança cibernética. Um dos marcos nesse avanço foi a introdução do mecanismo de defesa conhecido como PatchGuard (PG) ou Kernel Patch Protection (KPP), desenvolvido pela Microsoft. O PG tornou-se uma barreira eficaz contra os rootkits que exploravam o *hooking* da SSDT, protegendo o núcleo do sistema operacional e garantindo a integridade das chamadas de sistema.

Em aplicações que são obfuscadas ou virtualizadas, frequentemente o processo de identificação de chamadas de sistema torna-se desafiador, pois a aplicação poderá possuir um *bytecode* personalizado que será interpretado posteriormente pela máquina virtual (VM) (OREANS, 2023). Essa complexidade dificulta a análise quando o binário está sendo examinado por ferramentas de engenharia reversa.

## OBJETIVOS

Este trabalho tem como objetivo utilizar de técnicas de hooking e virtualização para a interceptação de chamadas do sistema para facilitar o processo de engenharia reversa principalmente em aplicações obfuscadas ou virtualizadas.

Os objetivos específicos são:

1. criar uma ferramenta eficiente e confiável que seja capaz de monitorar chamadas de sistema em tempo real em aplicações obfuscadas ou virtualizadas em ambientes Windows;
2. catalogar e identificar chamadas de sistema críticas que são comumente usadas em aplicações obfuscadas ou virtualizadas, fornecendo um foco específico para o monitoramento;
3. implementar funcionalidades para analisar o comportamento das aplicações em execução, visando detectar padrões anômalos ou suspeitos relacionados ao uso de chamadas de sistema;
4. desenvolver uma interface de usuário que permita a fácil configuração e visualização dos dados coletados, bem como a geração de relatórios detalhados das atividades monitoradas;
5. realizar testes extensivos para validar a eficácia e confiabilidade da ferramenta em diferentes cenários e configurações de sistema operacional Windows;
6. estabelecer um plano de manutenção contínua para a ferramenta, incluindo atualizações regulares para suportar novas versões do sistema operacional Windows.

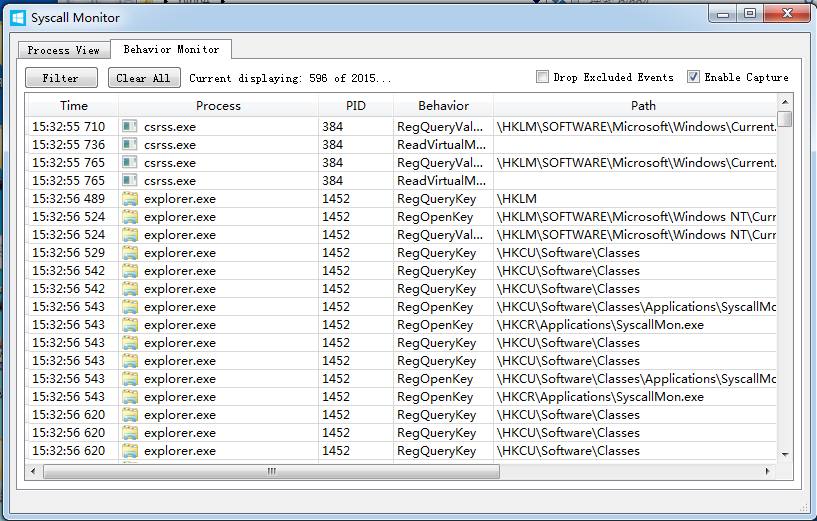
# trabalhos correlatos

Neste capítulo são abordados trabalhos com características similares aos principais objetivos do presente estudo. A seção 2.1 apresenta uma ferramenta para análise de execução de chamadas de sistema com uma interface gráfica (Wang, 2017), a seção 2.2 introduz a utilização de *process instrumentation callback* para obter notificações de chamadas executadas na camada de usuário (Deputation, 2021). Por fim, a seção 2.3 refere-se a uma ferramenta desenvolvida pela Microsoft frequentemente utilizada pela área de DevOPS (Microsoft, 2021).

## SYSCALL-MONITOR

Desenvolvido por Wang (2017), é uma contribuição significativa para a área de segurança da informação e monitoramento de chamadas de sistema em sistemas Windows. Este projeto utiliza a tecnologia de virtualização da Intel, Virtualization Technology (VT-X) e Extended Page Tables (EPT), para implementar uma abordagem inovadora na detecção e modificação de chamadas de sistema, permitindo a criação de stealth hook, que refere-se a uma técnica especializada e avançada de interceptação e monitoramento de chamadas de sistema em um sistema operacional, como o Windows. Essencialmente, elas permitem que o Syscall-Monitor observe e registre as atividades do sistema de forma discreta, sem chamar a atenção das medidas de segurança convencionais (Wang, 2017). Em outras palavras, os stealth hooks agem como ganchos ou pontos de observação que são inseridos no sistema de forma oculta, permitindo que o Syscall-Monitor rastreie e analise as interações entre aplicativos e o kernel do sistema operacional sem ser detectado, proporcionando assim uma visão profunda do comportamento do sistema para fins de segurança e monitoramento avançados. Essa capacidade de operar de maneira discreta é crucial para identificar atividades suspeitas e tomar medidas preventivas ou corretivas sem acionar alarmes de segurança que poderiam alertar potenciais invasores.

Figura 1 – Syscall Monitor



Fonte: Wang (2017)

Através da figura 1 é possível obter uma visualização da ferramenta em execução. Sendo assim, a tecnologia de virtualização da Intel permite a criação de máquinas virtuais e oferece suporte para recursos avançados, como a tradução de endereços de segundo nível (Second Level Address Translation - SLAT), que é essencial para o funcionamento do projeto. A principal funcionalidade do Syscall-Monitor é adicionar uma camada adicional de tradução entre os endereços virtuais e os endereços reais usados pelo sistema operacional Windows. Isso possibilita a modificação de rotinas nativas do kernel do Windows sem ser detectado pelo PG.

Ao usar stealth hooks, o Syscall-Monitor consegue monitorar e interceptar chamadas de sistema de forma discreta, o que é essencial para aplicações de segurança e monitoramento avançadas. Ele permite que os pesquisadores e profissionais de segurança acompanhem o comportamento do sistema, identifiquem atividades suspeitas e até mesmo façam modificações específicas, se necessário, sem acionar alarmes de segurança.

Apesar de ser um método muito eficaz no processo de identificação de chamadas de sistema manuais, há a limitação sobre a utilização de virtualização, neste caso, o modelo de Wang somente implementa tecnologia VT-X, fazendo assim com que não seja possível o monitoramento na tecnologia AMD-V (AMD).

Wang (2017) demonstra um profundo entendimento da virtualização da Intel e do funcionamento interno do sistema Windows. Seu trabalho inovador fornece uma base sólida para o desenvolvimento de ferramentas avançadas de segurança cibernética e demonstra a importância da pesquisa contínua na área de segurança da informação.

Em resumo, o projeto Syscall-Monitor de Wang (2017) é uma contribuição valiosa para a comunidade de segurança da informação, aproveitando a virtualização da Intel para criar um mecanismo eficaz de monitoramento e modificação de chamadas de sistema no Windows, com a capacidade de operar de forma discreta e contornar as medidas de segurança do sistema operacional. Suas inovações são um exemplo inspirador de como a tecnologia pode ser aplicada para aprimorar a segurança cibernética.

## INSTRUMENTATION\_CALLBACKS

Segundo Deputation (2021), o método empregado no projeto instrumentation\_callbacks é baseado na utilização de callbacks de instrumentação de processo. Esses callbacks permitem a monitorização de eventos específicos, como a execução de syscalls, dentro do espaço de memória de um processo. Isso é realizado através da injeção de uma DLL (Dynamic Link Library) no espaço de memória do processo em questão, permitindo que a DLL registre e responda a eventos específicos.

Ainda de acordo com Deputation (2021) as callbacks de instrumentação de processo oferecem a capacidade de interceptar e registrar a execução de syscalls, o que pode ser extremamente valioso na detecção de atividades maliciosas ou comportamentos anômalos em processos em execução. Essa abordagem é particularmente útil em cenários de segurança cibernética, onde a identificação de syscalls suspeitas pode ser um indicativo importante de atividades maliciosas.

No entanto, é importante destacar algumas limitações inerentes a esse método de detecção. A principal limitação é que ele exige a injeção de uma DLL no espaço de memória do processo que está sendo monitorado. Isso pode ser problemático em ambientes onde a injeção de DLLs é considerada uma atividade suspeita ou maliciosa. Além disso, a injeção de DLLs pode ser bloqueada ou detectada por medidas de segurança avançadas. Também é importante notar que a stack é limpa antes de executar a callback, ou seja, os paramêtros que foram informados para a execução da rotina não estarão mais disponíveis (Deputation, 2021).

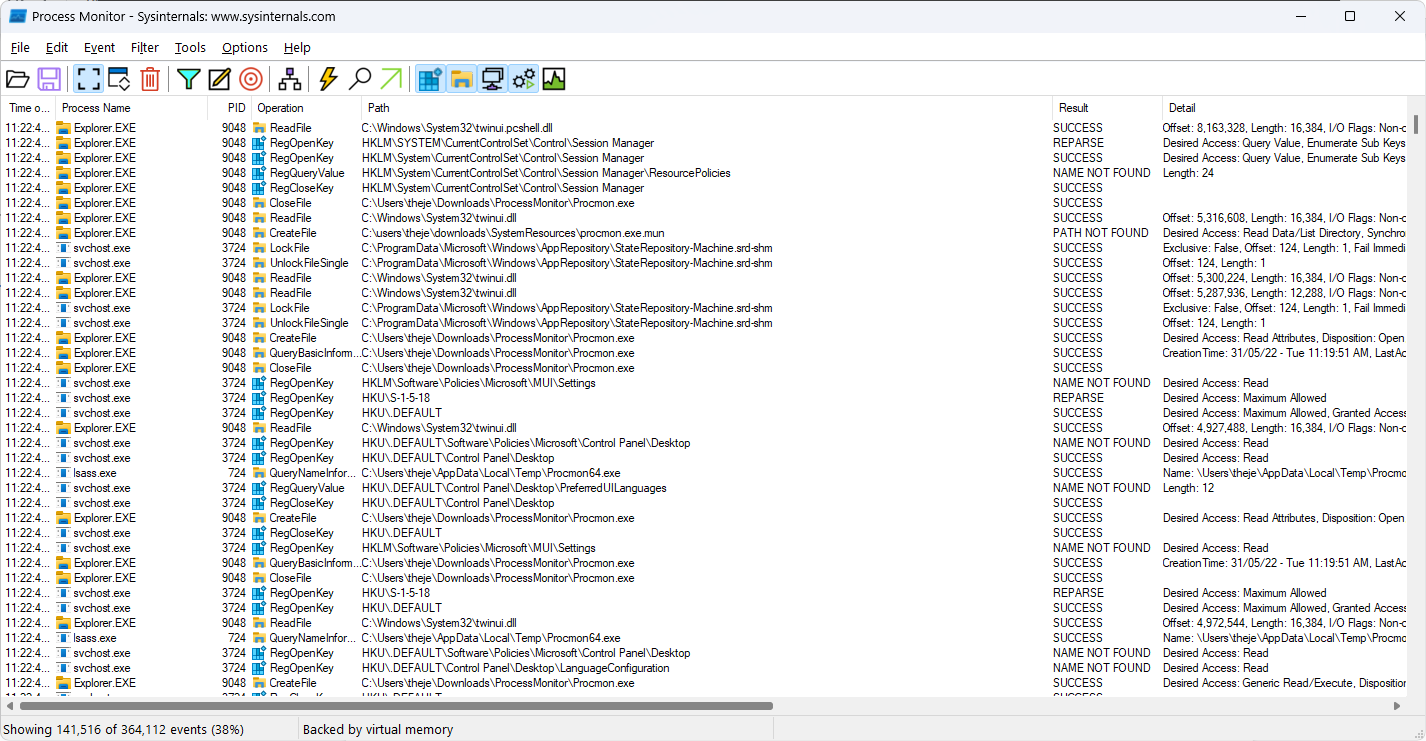
Outra consideração importante é que, como qualquer método de detecção, ele não é imune à possibilidade de burla. Um atacante que tenha conhecimento sobre a presença da DLL de instrumentação no processo monitorado pode tentar desativar ou burlar as callbacks de instrumentação, comprometendo assim a eficácia da detecção de syscalls (Deputation, 2021).

Em resumo, o projeto instrumentation\_callbacks de Deputation (2021) apresenta uma forma de detectar execução de syscalls por meio de callbacks de instrumentação de processo, sem a necessidade da instalação de um driver para efetuar a monitoração, entretanto, é importante destacar as limitações inerentes deste método, visto que é possível desabilitar as callbacks temporariamente e passar despercebido.

## PROCESS MONITOR

Desenvolvido pela Microsoft, é uma ferramenta de monitorização e diagnóstico de processos amplamente reconhecida no ecossistema Windows. Este software desempenha um papel fundamental na observação do comportamento de programas em execução no ambiente Windows e na coleta detalhada de informações sobre suas operações.

Figura 2 – Getting started with Procmon: The Beginner’s Guide to Monitoring Windows Systems



Fonte: Alex Marin. (2022)

O método empregado pelo Process Monitor para monitorar processos é notável, pois envolve a operação em modo kernel, onde um *driver* é carregado no sistema operacional. Esse driver atua como uma camada intermediária que permite ao Process Monitor registrar todas as atividades relevantes. Uma das funcionalidades notáveis do Process Monitor é a capacidade de realizar o *dumping* (registro) de callstacks, permitindo que os usuários localizem com precisão o ponto de origem de uma chamada de sistema (*syscall*) dentro de um aplicativo em execução.

A capacidade de rastrear a origem das chamadas de sistema é uma característica inestimável para profissionais de segurança, desenvolvedores e administradores de sistemas, pois permite a identificação rápida de problemas, comportamentos anômalos e até mesmo atividades maliciosas dentro de aplicativos ou processos em execução no sistema Windows.

No entanto, é importante destacar que, apesar de sua eficácia, o Process Monitor não é capaz de capturar todas as syscalls por meio de callbacks registradas no Windows. Algumas chamadas de sistema podem não ser visíveis por meio deste método, o que limita parcialmente sua capacidade de monitorização. Essa limitação muitas vezes é necessária para estar em conformidade com o PG.

Em resumo, o Process Monitor da Microsoft é uma ferramenta essencial que utiliza um driver carregado no modo kernel para monitorar e registrar o comportamento de processos no ambiente Windows. Sua capacidade de rastrear callstacks torna possível identificar com precisão a origem das chamadas de sistema em aplicativos em execução. Embora tenha algumas limitações inerentes, o Process Monitor representa uma alternativa valiosa para a análise e diagnóstico de processos no ambiente Windows, desempenhando um papel crucial na manutenção da integridade e segurança do sistema.

# proposta

A seguir é apresentada a justificativa para o desenvolvimento da ferramenta, juntamente com os principais requisitos e a metodologia de desenvolvimento a ser utilizada. Também são relacionados os assuntos e as fontes bibliográficas que irão fundamentar o estudo presente.

## JUSTIFICATIVA

Segundo Sikander (2022) o processo de detecção de chamadas de sistemas realizadas de forma manual em aplicações obfuscadas ou virtualizadas é dificultado, pois há a construção de um *bytecode* customizado, fazendo com que este seja somente interpretado pela *Virtual Machine (VM)*. Então, neste contexto, é proposto um projeto para desenvolver uma ferramenta avançada de monitoramento de chamadas de sistema com suporte às tecnologias de virtualização *VT-X* da Intel e *AMD-V* da AMD para estar dentro das conformidades do mecanismo de defesa PG, contribuindo ativamente no processo de rastreabilidade de chamadas de sistema facilitando o processo de análise dinamica comumente destinado aos analistas de malware.

Quadro 1 - Comparativo dos trabalhos correlatos

|  |  |  |  |
| --- | --- | --- | --- |
| Trabalhos Correlatos  Características | Syscall-Monitor | *Instrumentation\_callbacks* | Process Monitor |
| Técnica utilizada | *Hooking* com virtualização *VT-X / EPT* | *Callback* de instrumentação de processo | *Callbacks* registradas no *kernel* do sistema operacional |
| É possível obter os parâmetros da rotina | Sim | Não | Sim |
| Executado na camada | Kernel | Usuário | Kernel |
| É possível burlar através da camada de usuário | Não | Sim | Não |
| Compatível entre os fabricantes | Intel | Intel e AMD | Intel e AMD |
| Possui um driver centralizado fácil de manutenção | Não | Não | Não |

Fonte: elaborado pelo autor.

Conforme constatado no Quadro 1, a aplicação Syscall-Monitor por Wang (2017) apenas utiliza da tecnologia de virtualização VT-X, o projeto instrumentation\_callbacks por Deputation (2021) utiliza de callback de instrumentação de processo, sendo assim, apenas reside na camada de usuário (usermode) do sistema operacional, tornando mais passível de desativação por alguma outra aplicação comum, visto que ambas estarão com os mesmos privilégios, o Process Monitor está presente na camada do kernel (núcleo) do sistema operacional, havendo a limitação onde nem todas as rotinas que estão presentes na SSDT constam para a registração de uma callback.

Como a ferramenta de Wang (2017) utiliza diretamente de hooking para a interceptação das chamadas de sistema, neste momento ainda não houve a limpeza da memoria stack, sendo assim, ainda é possível obter os parâmetros que foram informadas para a rotina, já o projeto instrumentation\_callbacks como previamente detalhado por Deputation (2021) no momento que haver a chamada para essa callback a stack já foi limpa, impossibilitando obter os parâmetros de execução da função em questão. Por fim, a ferramenta Process Monitor é capaz de obter os parâmetros da rotina devido a sua ordem de chamada ser antes da execução em si da rotina com a única limitação de um escopo reduzido.

Previamente dito, a ferramenta de Deputation (2021) instrumentation\_callback reside na camada de usuário, sendo assim, ela é a mais comúm, é a camada onde as aplicações tradicionais são alocadas facilitando o atacante burlar por um executável ou DLL.

Segundo Wang (2017) como hypervisors dependem da implementação de cada CPU, o Syscall-Monitor não foi implementado para a tecnologia AMD-V impossibilitando os usuários de processadores da fabricante AMD de utilizar.

O Syscall-Monitor utiliza o framework de virtualização ddimon por Tanda (2016), onde Tanda (2016) utiliza de mais alguns outros frameworks para a construção do seu projeto, fazendo com que o processo de manutenção do Syscall-Monitor seja mais complicado, já o Process Monitor não é um projeto de código aberto impossibilitando a criação de novas funcionalidades.

Wang (2017) também relata uma lentidão causada proveniente da má otimização no uso de memória, sendo assim, é necessário reiniciar a máquina depois de certo tempo de uso, já o projeto instrumentation\_callbacks por Deputation (2021) como utiliza da camada de usuário o máximo que poderá acontecer é houver um crash na aplicação, o Process Monitor por ser um projeto maduro não apresenta o problema citado.

## REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta a ser desenvolvida deverá:

1. identificar as chamadas de sistema (*syscall*) executadas de forma manual (fora da ntdll) (Requisito Funcional – RF);
2. apresentar o *dumping* de *callstack* com os nomes de módulos presentes no processo e calcular o *relative virtual address (RVA)* (RF);
3. apresentar o tempo de duração da execução da chamada de sistema (*syscall*) (RF);
4. ser compatível entre os principais fabricantes de processadores do mercado (Intel e AMD) (Requisito Não Funcional - RNF);
5. ser compatível com as *builds* recentes do Windows 10 e Windows 11 (RNF);
6. ser performático de modo que o usuário não sofra com problemas de desempenho impossibilitando o uso da máquina (RNF);
7. construir um driver centralizado que facilite a manutenção das funcionalidades de *hooking* (RNF);
8. ser implementado utilizando a linguagem de programação C++ (RNF);
9. desenvolver utilizando a IDE Visual Studio Community (RNF).

## METODOLOGIA

O trabalho será desenvolvido observando as seguintes etapas:

1. levantamento bibliográfico: pesquisar e estudar sobre desenvolvimento *kernel mode,* técnicas de *hooking*, virtualização e trabalhos correlatos;
2. lenvatamento dos requisitos: levando em consideração as informações da etapa anterior, validar os requisitos propostos para a aplicação;
3. desenvolvimento de driver centralizado: implementar uma estrutura de centralização para facilitar o desenvolvimento da ferramenta e de sua manutenção;
4. modelagem do driver de *hooking:* após o desenvolvimento do item (c), desenvolver o driver que conterá as rotinas nativas do *kernel* do sistema operacional e também comunicará a aplicação sobre a execução de uma nova chamada de sistema (*syscall*);
5. implementação de interface gráfica: verificar se a linguagem de programação C++ atenderá os principais requisitos para a implementação da interface gráfica e desenvolver;
6. testes: avaliar a performance, assim como verificar se está sendo realizado todos os *hookings* propostos para a ferramenta e também verificar sobre as compatibilidades entre os sistemas operacionais e processadores, por fim checar a segurança do uso da aplicação.

As etapas serão realizadas nos períodos relacionados no Quadro 2.

Quadro 2 - Cronograma

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2024 | | | | | | | | | |
|  | jul. | | ago. | | set. | | nov. | | dez. | |
| etapas / quinzenas | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| levantamento bibliográfico |  | X | X |  |  |  |  |  |  |  |
| levantamento dos requisitos |  |  | X |  |  |  |  |  |  |  |
| desenvolvimento do driver centralizado |  |  |  | X |  |  |  |  |  |  |
| modelagem do driver de *hooking* |  |  |  |  | X | X | X |  |  |  |
| implementação da interface gráfica |  |  |  |  | X | X | X | X |  |  |
| testes |  |  |  |  |  |  |  |  | X | X |

Fonte: elaborado pelo autor.

# Tecnologias relevantes

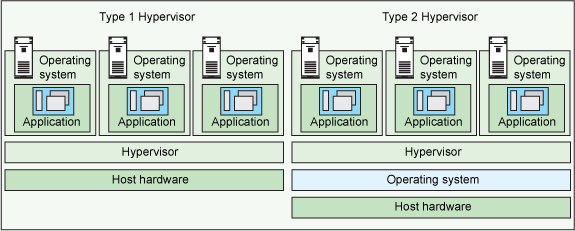
Este capítulo descreve brevemente as tecnologias relevantes para o presente estudo:

## Hypervisors

Segundo Martin (2017), hypervisors fazem parte da ampla área da computação conhecida como virtualização, um conceito que existe há certo tempo. No meio da década de 1960, os mainframes dominavam o mundo da computação corporativa. No entanto, segundo Martin (2017) esses mainframes eram relativamente raros e extremamente caros. Muita das vezes servindo a mais de um usuário, a qual compartilhavam uma fátia do tempo para executar alguns trabalhos. Significando que, do ponto de vista de cada usuário, parecia que eles tinham um mainframe exclusivamente destinado a eles. Neste contexto, um hypervisor é um software que gerencia o hardware disponível no computador em que está sendo executado para os demais sistemas operacionais que está sendo controlado (guests).

Também segundo Martin (2017), existem dois tipos de virtualização baremetal ou type-1 e guest ou type-2. A virtualização baremetal, é uma técnica de virtualização na qual um hypervisor é instalado diretamente no hardware físico de um servidor, sem a necessidade de um sistema operacional hospedeiro (host). Isso significa que o hypervisor funciona como sistema operacional primário, controlando o acesso direto aos recursos físicos da máquina e permitindo a execução de múltiplas máquinas virtuais (VMs) independentes. Já de acordo com Martin (2017) um hypervisor do tipo guest é um software de virtualização que é instalado e executado sobre um sistema operacional host existente.

Figura 3 – Learn about hypervisors, system virtualization, and how it works in a cloud environment

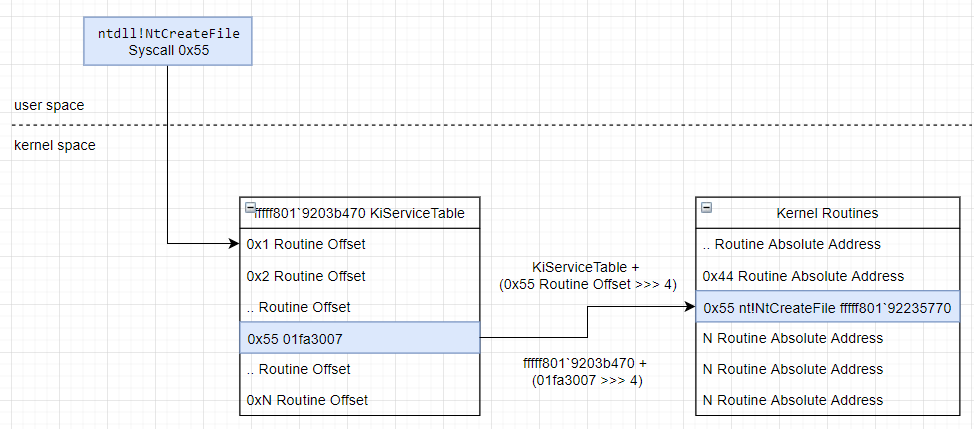


Fonte: Bhanu P Tolheti (2011).

## Chamada de sistema (syscall)

De acordo com Brizendine (2023), uma syscall é feita de algumas funções na biblioteca da NTDLL como uma maneira de requisitar um serviço do kernel do sistema operacional. Uma syscall é o último passo entre a camada de usuário e a camada de kernel e uma syscall não é feita para ser usada por programadores, sendo assim, programadores irão chamar outras funções, geralmente provindas da biblioteca KERNEL32. Uma syscall também utiliza um número especial de serviço, nomeado de system service number (SSN) que quando é executada tem o seu número atribuído ao registrador EAX. No momento da execução, a rotina presente na kernel é obtida através da resolução da SSN utilizando a SSDT que é a tabela responsável por conter todas as rotinas que poderão ser requisitadas através da instrução SYSCALL (Ired Team, 2019).

Figura 4 – System Service Descriptor Table

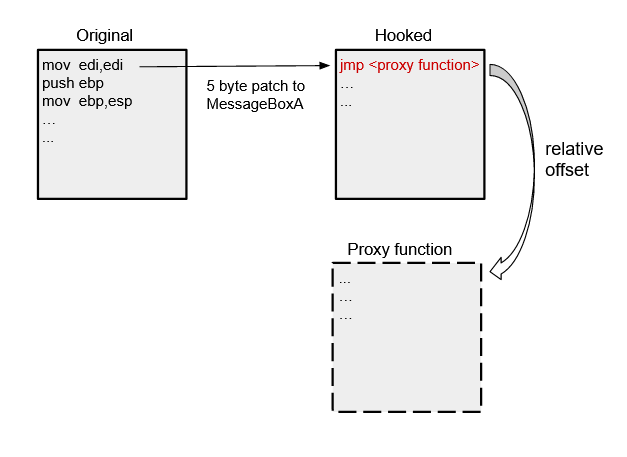


Fonte: ired.team (2020)

## Técnica de *hooking*

Segundo Shaid e Maarof (2015), a técnica de *hooking* é uma técnica que permite monitorar, modificar ou estender o comportamento de funções ou rotinas de uma *Application Programming Interface (API*) em um software ou sistema operacional. Essa técnica é frequentemente utilizada para aprimorar a funcionalidade de programas, adicionar recursos de segurança, rastrear a atividade do sistema ou até mesmo depurar o software em questão. O uso de hooking é poderoso, mas também apresenta desafios. Pode ser abusado para atividades maliciosas, como spyware ou malware. Portanto, é importante que o uso dessa técnica seja ético e legal. Muitos sistemas operacionais e aplicativos de segurança possuem medidas para detectar atividades suspeitas sobre o uso desta técnica.

Figura 5 – Basic Windows API Hooking



Fonte: Jayson Hurst (2021).

## ROOTKITS

Segundo Pedro (2021), rootkit é um software usado por hackers para assumir o controle do computador ou rede. Inicialmente, ele parece ser um único software, mas apresentam uma série de ferramentas que permitem com que cibercriminosos tenham acesso administrativo a um dispositivo em questão, sendo possível o roubo de dados pessoais, informações financeiras e até mesmo usar a máquina para enviar spam ou ataques de negação de serviço (DDoS). De acordo com Burdova (2021) isso acontece porque os rootkits executam na camada de maior privilégio do sistema (kernel) dessa forma eles podem interferir no comportamento do sistema operacional em um escopo global.

## PATCHGUARD

De acordo com Ermolov e Shishkin (2014), PatchGuard ou Kernel Patch Protection é um mecanismo de defesa do sistema operacional, sua principal função é proteger o núcleo do sistema operacional contra modificações não autorizadas e ataques de rootkits, o PG é uma resposta direta aos desafios representados por rootkits que segundo Pedro (2021) conseguem se infiltrar no sistema e esconder o seu comportamento do kernel tornando dificil o processo de detecção e remoção do mesmo.

## obfuscação por virtualização de código

De acordo com Blazytko (2021) a obfuscação por virtualização de código ocorre quando um conjunto de instruções (bytecode) compatíveis com o processador alvo é submetido a criação de um novo conjunto de instruções e também é construido uma camada na aplicação para a tradução e interpretação deste código em questão, dificultando o processo de engenharia reversa pois o analista teria de reconstruir um interpretador para o bytecode que está sendo analisado.

## TeCnologia de virtualização vt-x

Segundo Horta (2010) a tecnologia de virtualização VT-X, se propõe a diminuir a complexidade dos VMM (monitores de máquina virtual), aumentar o desempenho de sistemas virtualizados baseados em software e permitir que sistema operacionais não modificados sejam executados, com desempenho igual ou superior ao alcançado pela paravirtualização, na qual os sistemas operacionais são modificados ou sofrem translações binárias para serem executados sobre um VMM.

## TeCnologia de VIRTUALIZAçÃO EPT

Segundo Bhatia (2010), a tecnologia de virtualização Extended Page Tables (EPT) foi implementada para a diminuição do overhead causado pela virtualização, onde ela implementa Second Level Address Translation (SLAT) possibilitando assim com que seja feito uma nova camada de tradução de endereços virtuais em endereços reais, então as vezes que serão necessárias as traduções não será necessário preciso solicitar ao VMM reduzindo o impacto na Memory Management Unit (MMU).

Referências

Blazytko. Workshop: Analysis of Virtualization-based Obfuscation. Disponível em: https://synthesis.to/presentations/r2con2021-deobfuscation.pdf. Acesso em 14 abr de 2024.

Brizendine. Windows Syscalls in Shellcode: Advanced Techniques for Malicious Functionality. Disponível em: https://conference.hitb.org/hitbsecconf2023ams/materials/D1T2%20-%20Windows%20Syscalls%20in%20Shellcode%20-%20Advanced%20Techniques%20for%20Malicious%20Functionality%20-%20Bramwell%20Brizendine.pdf. Acesso em: 14 abr de 2024.

Bhatia. Performance Evaluation of Intel EPT Hardware Assist. Disponível em: https://www.vmware.com/pdf/Perf\_ESX\_Intel-EPT-eval.pdf. Acesso em 14 abr de 2024.

Burdova. O que é um rootkit? Disponível em: https://www.avast.com/pt-br/c-rootkit | Avast. Acesso em 14 abr de 2024.

CISO ADVISOR. Volume global de ciberataques teve alta de 38% em 2022. Disponível em: https://www.cisoadvisor.com.br/volume-global-de-ataques-ciberneticos-aumenta-38-em-2022. Acesso em: 28 set 2023.

Deputation, instrumentation\_callbacks. Disponível em: https://github.com/Deputation/instrumentation\_callbacks. Acesso em: 27 set de 2023.

Henrique. Intel Virtualization Technology. Disponível em: https://www.gta.ufrj.br/ensino/eel879/trabalhos\_vf\_2010\_2/apresentacoes/VT.pdf. Acesso em 01 de dez de 2023.

IRED TEAM. System Service Descriptor Table – SSDT. Disponível em: https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/glimpse-into-ssdt-in-windows-x64-kernel. Acesso em: 14 abr de 2024.

Marin. Getting started with Procmon: The Beginner’s Guide to Monitoring Windows Systems. Disponível em: https://www.advancedinstaller.com/process-monitor-beginner-guide.html. Acesso em: 14 abr de 2024.

Martin. Hypervisor Part 1 – What is a Hypervisor and How Does it Work?. Disponível em: https://blackberry.qnx.com/content/dam/qnx/whitepapers/2017/what-is-a-hypervisor-and-how-does-it-work-pt1.pdf. Acesso em: 14 abr de 2024.

Oreans. Total Obfuscation against Reverse Engineering. Disponível em: https://www.oreans.com/CodeVirtualizer.php. Acesso em: 16 abr de 2024.

Pedro. O que é Rookit?. Disponível em: https://tecnoblog.net/responde/o-que-e-rootkit/. Acesso em: 16 abr de 2024.

Reginato. Updated Analysis of PatchGuard on Microsoft Windows 10 RS4. Disponível em: https://blog.tetrane.com/downloads/Tetrane\_PatchGuard\_Analysis\_RS4\_v1.01.pdf. Acesso em: 16 abr de 2024.

Russinovich. Process Monitor V3.96. Disponível em: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon. Acesso em: 16 abr de 2024.

Shaid; Maroof. In memory detection of Windows API call hooking technique. Disponível em: https://www.researchgate.net/publication/283480063\_In\_memory\_detection\_of\_Windows\_API\_call\_hooking\_technique. Acesso em: 16 abr de 2024.

Sikander. AV/EDR Evasion Using Direct System Calls (User-Mode vs kernel-Mode). Disponível em: https://medium.com/@merasor07/av-edr-evasion-using-direct-system-calls-user-mode-vs-kernel-mode-fad2fdfed01a. Acesso em 16 abr de 2024.

Tanda. DdiMon. Disponível em: https://github.com/tandasat/DdiMon. Acesso em 16 abr de 2024.

Tholeti, Bhanu. Learn about hypervisors, system virtualization and how it works in a cloud environment. Disponível em: https://developer.ibm.com/articles/cl-hypervisorcompare/. Acesso em: 16 abr de 2024.

Wang. Syscall-Monitor. Disponível em: https://github.com/hzqst/Syscall-Monitor. Acesso em: 16 abr de 2024.