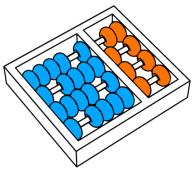
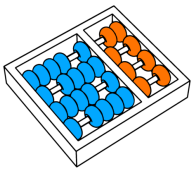


Cicero Silva Luiz Junior

“Segurança em Ambientes Windows”

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Cicero Silva Luiz Junior

"Segurança em Ambientes Windows"

Orientador(a): **Prof. Dr. Paulo Lício de Geus**

Trabalho de Conclusão de Curso apresentado à Comissão de Graduação do Instituto de Computação.

Este exemplar corresponde à versão final do Trabalho de Conclusão de Curso defendido por Cicero Silva Luiz Junior, sob orientação de Prof. Dr. Paulo Lício de Geus.

Assinatura do Orientador(a)

CAMPINAS
2013

Abstract

Operating Systems are a piece of software responsible for managing computer hardware and software, as well as providing an interface between the machine and its users. Many Operating systems target a multitude of architectures, but the most widely known are those based on either UNIX or Windows. One of the main responsibilities of an operating system is to ensure its own security and the security of other software running on it. The aim of this work is to present the Windows NT Operating System (version 6.1) from a security standpoint. Chapter 2 presents briefly, what makes the Windows NT architecture. Chapter 3 explains the most important mechanisms that implement the Windows Security Model up to Windows 7 and Chapter 4 lists some recent vulnerabilities that target Windows operating systems, followed by a brief discussion of what real impact they can potentially have.

Resumo

Sistemas operacionais gerenciam *hardware* e *software* de dispositivos computacionais e proveem uma interface entre os usuários e seus computadores. Há sistemas operacionais para diferentes tipos de plataformas e arquiteturas, podendo-se citar como os mais difundidos aqueles baseados em Unix e Windows. Uma das principais características dos sistemas operacionais modernos é a provisão de segurança para os usuários. Este trabalho tem por objetivo a realização de um levantamento sobre o sistema operacional Windows NT e suas características, com enfoque nos aspectos relacionados às funcionalidades de segurança implementados até a versão 6.1 (Windows 7). O Capítulo 2 apresenta uma introdução ao modelo da arquitetura NT enquanto o Capítulo 3 detalha os principais mecanismos do sistema responsáveis pela segurança que estão implementados e disponíveis até a versão acima citada. Por fim, o Capítulo 4 mostra algumas vulnerabilidades que foram encontradas nas últimas versões do Windows e apresenta uma breve discussão do que elas realmente representam.

Agradecimentos

A Deus, pelo Dom da Vida e pelo Bom Ânimo.

Aos meus pais, por sempre me incentivarem na minha caminhada.

Ao Instituto de Computação e à UNICAMP, pela oportunidade de crescimento acadêmico e profissional.

The 10 Immutable Laws of Security 2.0

Law 1: If a bad guy can persuade you to run his program on your computer, it's not solely your computer anymore.
Law 2: If a bad guy can alter the operating system on your computer, it's not your computer anymore.
Law 3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.
Law 4: If you allow a bad guy to run active content in your website, it's not your website any more.
Law 5: Weak passwords trump strong security.
Law 6: A computer is only as secure as the administrator is trustworthy.
Law 7: Encrypted data is only as secure as its decryption key.
Law 8: An out-of-date antimalware scanner is only marginally better than no scanner at all.
Law 9: Absolute anonymity isn't practically achievable, online or offline.
Law 10: Technology is not a panacea.

The Microsoft Security Response Center

Sumário

Abstract **iii**

Resumo **iv**

Agradecimentos **v**

Epigraph **vi**

1 Introdução **1**

2 Visão geral da arquitetura do Windows NT **3**
2.1 Objetivos de Design . . . 3
2.1.1 Extensibilidade . . . 3
2.1.2 Portabilidade . . . 4
2.1.3 Confiabilidade . . . 5
2.1.4 Compatibilidade . . . 5
2.1.5 Segurança . . . 6
2.1.6 Performance . . . 6
2.2 Arquitetura simplificada do Windows NT . . . 7
2.2.1 Módulos de modo *Kernel* . . . 8
2.2.2 Módulos de modo Usuário . . . 11

3 Principais subsistemas de segurança implementados até o Windows 7 **13**
3.1 SRM - *Security Reference Monitor* (Monitor de referências de Segurança) . 13
3.1.1 Verificações de Acesso . . . 14
3.1.2 Identificadores de Segurança . . . 14
3.2 Níveis de Integridade . . . 15
3.3 *Tokens* . . . 16
3.3.1 *Tokens* Restritos . . . 17
3.3.2 *Tokens* Filtrados . . . 17

3.4 *Impersonation* (Personificação) . . . 18
3.5 Contas Virtuais de Serviço . . . 19
3.6 Descritores de Segurança . . . 19
3.6.1 *ACL*: Lista de controle de acesso . . . 19
3.6.2 Determinação de Acesso . . . 22
3.7 Isolamento de privilégios da Interface de Usuário . . . 28
3.8 Direitos de um Usuário e Privilégios de Sistema . . . 28
3.8.1 Direitos de um Usuário . . . 28
3.8.2 Privilégios . . . 29
3.8.3 Super Privilégios . . . 34
3.9 Sistema de *Logon* . . . 35
3.9.1 *Winlogon* . . . 35
3.10 Controle de Conta do Usuário e Virtualização de Recursos . . . 36
3.10.1 Virtualização do Sistema de Arquivos e do Registro . . . 37
3.10.2 Mecanismos de Elevação . . . 40
3.10.3 Auto-Elevação . . . 41
3.11 *PatchGuard* - Proteção contra alterações no *Kernel* . . . 42
3.12 Integridade de Código . . . 45
3.13 *BitLocker* . . . 45
3.13.1 Descrição . . . 45
3.13.2 Sequência de *Boot* do *BitLocker* . . . 48

4 Análise Histórica e Discussão de Vulnerabilidades 52
4.1 CVE-2013-3940 . . . 52
4.2 CVE-2013-3894 . . . 53
4.3 CVE-2013-3195 . . . 53
4.4 CVE-2013-3175 . . . 53
4.5 CVE-2013-3174 . . . 54
4.6 CVE-2013-0011 . . . 54
4.7 CVE-2012-4774 . . . 54
4.8 CVE-2012-1852 . . . 55
4.9 CVE-2012-0173 . . . 55
4.10 CVE-2012-0001 . . . 55

5 Conclusão 57

Referências Bibliográficas 58

Lista de Tabelas

3.1 Níveis de Integridade do NT 6.1 15
3.2 Lista de Privilégios disponíveis no Windows 29
3.3 Componentes protegidos pelo PatchGuard 43

Lista de Figuras

1.1 *Market Share* de Sistemas Operacionais *Desktop* em Outubro 2013 [3] . . . 2

2.1 Arquitetura simplificada do Windows NT 7

3.1 Exemplo de um objeto do tipo Arquivo e sua *DACL* 21
3.2 Exemplo de um objeto do tipo processo e suas *ACLs* 23
3.3 Ilustração gráfica de uma verificação de acesso 27
3.4 Ilustração gráfica do funcionamento da virtualização de sistema de arquivos do UAC 39
3.5 Exemplo de uma elevação de consentimento no Windows Vista 41
3.6 Exemplo de uma elevação *OTS* no Windows 7 42
3.7 Arquitetura simplificada do *BitLocker* 47
3.8 Ilustração de como o *BitLocker* gerencia suas chaves 49
3.9 Ilustração do processo de verificação em cadeia no tempo de *Boot* 51

Capítulo 1

Introdução

Sistemas operacionais gerenciam *hardware* e *software* de dispositivos computacionais e proveem uma interface entre os usuários e seus computadores. Há sistemas operacionais para diferentes tipos de plataformas e arquiteturas, podendo-se citar como os mais difundidos aqueles baseados em Unix e Windows. Uma das principais características dos sistemas operacionais modernos é a provisão de segurança para os usuários. Tal segurança é alcançada por meio da implantação de mecanismos de proteção tanto no espaço do usuário quanto no espaço do *kernel*. Existem basicamente três classes maiores desses mecanismos. Primeiro tem-se os que visam garantir e controlar permissões como o *Security Reference Monitor*, os Descritores de Segurança e os sistemas de controles de permissão. Em seguida existem os que tentam mitigar possíveis danos causados quando programas ou serviços vulneráveis são atacados e comprometidos. Entre eles podem ser destacados o Controle de Conta do Usuário, o *Kernel PatchGuard* e os diversos filtros de privilégio presentes. Além desses também existem mecanismos que operam em particularidades da arquitetura, como no caso da x86-64 em que tem grande importância o ASLR e o DEP. Na época da escrita desse trabalho 90,66% dos computadores pessoais tradicionais (como *Desktops* e *Notebooks*) conectados à internet são baseados em alguma versão do Windows. Versões não NT representam menos de 0,01% do total, o que é muito bom já que em versões não NT praticamente inexistem sistemas de mitigação de vulnerabilidades.

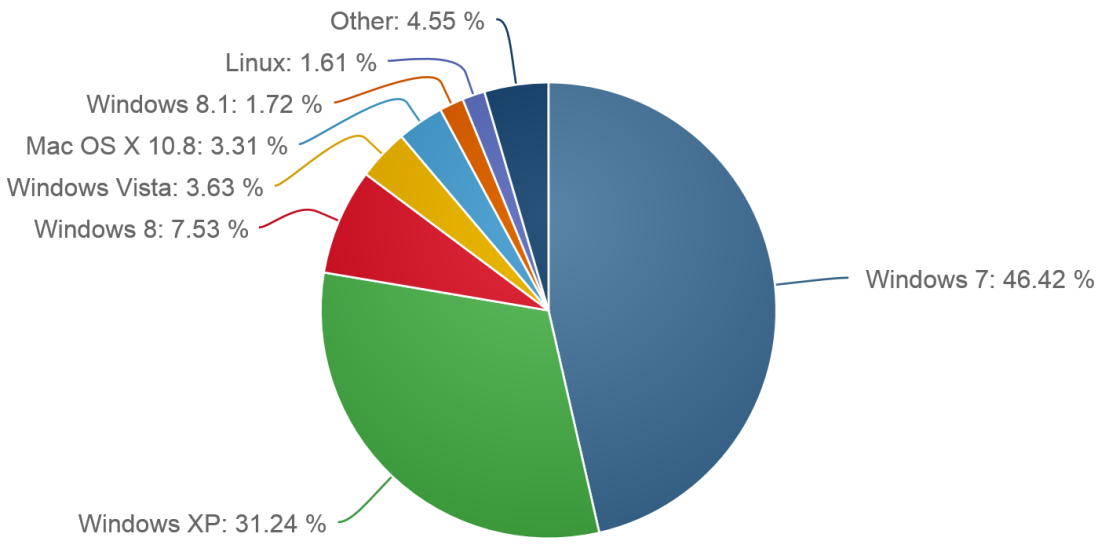


Figura 1.1: *Market Share* de Sistemas Operacionais *Desktop* em Outubro 2013 [3]

Por outro lado tem-se a informação alarmante de quase 32% dos computadores pessoais terem o Windows XP como sistema operacional. Trata-se de um sistema que, embora seja da família NT, foi lançado há mais de uma década e não possui vários dos mecanismos e políticas mínimas de segurança discutidas ao longo desse trabalho e que são essenciais para uma proteção pró-ativa.

No decorrer dos anos a evolução do *hardware* permitiu abandonar as versões não NT do Windows em favor das versões NT que possuem mecanismos de segurança e gerência muito mais sofisticados e compatíveis com a era da "internet das coisas". Dessa forma é possível focar em apenas um único sistema operacional capaz de operar desde *smartphones* até grandes *clusters* e assim desenvolver uma grande estrutura centralizada que cria *patches* e correções de forma facilitada. Esse fato combinado à estrutura modular do Windows NT faz com que a arquitetura NT e seus sistemas derivados sejam um dos mais seguros da atualidade. [13] [15] [19] [14]

Este trabalho tem por objetivo a realização de um levantamento sobre o sistema operacional Windows NT e suas características, com enfoque nos aspectos relacionados às funcionalidades de segurança implementados até a versão 6.1 (Windows 7). O Capítulo 2 apresenta uma introdução ao modelo da arquitetura NT enquanto o Capítulo 3 detalha os principais mecanismos do sistema responsáveis pela segurança que estão implementados e disponíveis até a versão acima citada. Por fim, o Capítulo 4 mostra algumas vulnerabilidades que foram encontradas nas últimas versões do Windows e apresenta uma breve discussão do que elas realmente representam.

Capítulo 2

Visão geral da arquitetura do Windows NT

O Windows NT é uma família de sistemas operacionais de propósito geral desenvolvido pela Microsoft. Trata-se de um sistema operacional orientado a linguagem de alto nível, independente de processador, compatível com multiprocessadores e multiusuário.

2.1 Objetivos de Design

O projeto inicial do Windows NT foi feito com os seguintes objetivos de design: Extensibilidade, portabilidade, confiabilidade, retro compatibilidade, segurança e performance. [12] [4] Para alcançar esses objetivos o Windows NT foi desenvolvido usando-se do conceito de sofware em camadas. Adicionalmente cada camada do sistema operacional pode falar apenas com sua camada imediatamente superior ou imediatamente inferior. [10]

2.1.1 Extensibilidade

Define-se como extensibilidade a capacidade de um sistema operacional se adaptar para dar suporte a novo *hardware* e novos cenários. No Windows NT isso é alcançado através da implementação de subsistemas (ou servidores), dessa forma isolando e controlando as dependências. Ou seja, subsistemas inteiros podem ser reescritos ou trocados, desde que suas interfaces sejam mantidas. Como todas as trocas de informações dentro do sistema operacional são feitas através dessas interfaces, os detalhes de implementação dos subsistemas deixam de ser importantes, desde que ele realize a tarefa à qual foi designado.

Adicionalmente, a arquitetura baseada em subsistemas que foi utilizada no Windows permite adicionar e remover ambientes e interfaces de forma extremamente isolada e programática. Alguns ambientes disponíveis para o Windows NT são o OS/2, Win16,

DOS, POSIX e Win32, sendo o mais amplamente disponível o Win32. Por fim, caso necessário, novos ambientes podem ser construídos e adicionados ao Windows sem que seja necessário fazer qualquer mudança nas camadas inferiores do sistema operacional. O Windows *Runtime* (novo ambiente para aplicativos estilo *tablet*) disponível no Windows 8 é um exemplo de novo ambiente. [17] [16]

Outro ponto importante para a extensibilidade do Windows NT é a possibilidade de carga dinâmica de programas que podem rodar em modo *kernel*, chamados *Device Drivers*. Embora o nome esteja ligado à palavra *Device*, não há necessidade alguma que um *Device Driver* realmente opere um dispositivo. O que vale ressaltar é que ele funciona na camada do *Executive* do Sistema Operacional e que as operações e *APIs* disponíveis para ele são bem diferentes das disponíveis para programas que rodam em Modo Usuário, que muitas vezes apenas conversam com o subsistema Win32. [16] [9]

Além das características já mencionadas deve-se considerar o fato de o Windows NT possuir uma interface de *Syscalls* que permite de forma organizada e precisa a alteração de *Syscalls* presentes no sistema (através de *Hooking*), bem como a adição de novas. [16]

2.1.2 Portabilidade

Portabilidade pode ser definida como a capacidade de um Sistema de ser movido para outras plataformas de *hardware* com o mínimo de esforço. Para conseguir essa característica o time do Windows NT decidiu escrever todo o sistema operacional nas linguagens C / C++, de forma modular, além de criar uma camada especial chamada de *HAL*. *HAL* significa *Hardware Abstraction Layer* e trata-se de uma camada de interface de *software* que fica entre o *Hardware* e o *Kernel*. O objetivo dessa camada é reduzir ao mínimo possível as dependências do resto do sistema operacional com relação ao *hardware*. [16] Ou seja, a todo momento que for possível as camadas superiores do sistema operacional fazem chamadas às interfaces expostas pela *HAL* ao invés de lidar diretamente com o *hardware* em que o sistema está rodando. *Device Drivers*, contudo, podem acessar o *hardware* diretamente sem precisar passar pela *HAL*. A ideia por trás desse arranjo é que, caso desejado, rodar o Windows em uma nova arquitetura de *hardware* exige basicamente apenas reescrever a camada de *HAL*, pois ela é a única dependência das camadas superiores do próprio sistema operacional.

Nesse exato momento o NT está apto a rodar nas arquiteturas Intel x86, x86-64, Intel Itanium e ARM. Anteriormente o Windows NT também já rodou nas arquiteturas MIPS e Alpha. [16] [10] [12]

2.1.3 Confiabilidade

Para esse cenário, confiabilidade é a capacidade de um sistema operacional de lidar com erros de aplicação, de usuário e de *hardware* sem que esses erros afetem o funcionamento do próprio sistema operacional na medida do possível. Na arquitetura do Windows NT isso é obtido através do isolamento completo entre aplicações rodando em modo usuário e isolamento entre aplicações modo usuário e aplicações modo *kernel*. O isolamento entre processos em nível de usuário se beneficia em grande parte devido ao esquema de memória virtual implementado, onde cada processo possui seu próprio espaço de endereçamento virtual. Apenas o sistema operacional tem autoridade para controlar o diretório de páginas, enquanto processos rodam como se fossem o único processo da máquina. Por questões de *performance* e aproveitamento de recursos, contudo, o espaço de endereçamento de um processo de 32bits é dividido em duas porções de 2GB. Nos 2GB superiores estão mapeadas seções de código somente-leitura correspondente às bibliotecas compartilhadas entre os vários processos. Para evitar duplicação do uso de memória, caso dois ou mais processos mapeiem uma mesma biblioteca, o sistema operacional carrega essa biblioteca em páginas físicas apenas uma vez e redireciona as páginas virtuais de todos os processos que precisam dessa biblioteca para essas páginas físicas já fixadas. Os 2GB inferiores são para uso do próprio processo.

Por outro lado, o isolamento entre aplicações modo usuário e aplicações modo *kernel* é feito através de uma distinção, onde código modo *kernel* roda no modo supervisor do processador. Para fins de compatibilidade com as mais variadas arquiteturas o Windows NT usa apenas dois níveis de privilégio de processador. Nas arquiteturas Intel eles são o nível 0 e o nível 3. [16] [10] [17]

2.1.4 Compatibilidade

Compatibilidade aqui é definida tanto como retro compatibilidade, onde versões mais novas de um mesmo sistema é capaz de executar programas feitos para versões anteriores, como também compatibilidade entre programas escritos para diferentes sistemas operacionais.

O Windows NT foi desenvolvido para ser um sistema extremamente compatível. Assim como nas propriedades de extensibilidade do sistema que já foram mencionadas, aqui o conceito de subsistemas de ambiente é fundamental, pois isso lhe confere a capacidade de ser um sistema operacional que possui personalidades. O Windows NT foi arquitetado para ser vários sistemas operacionais em um [4] [8] podendo rodar aplicações que não foram escritas para o Windows NT. Isso porque aplicações de usuário rodam sobre um dos subsistemas de ambiente (personalidade) disponíveis no Windows, e esse subsistema em questão se comunica com as camadas inferiores para permitir a execução do pro-

grama. O subsistema de ambiente nativo do Windows NT é o Win32 e ele roda em modo *kernel* por questões de *performance*, mas vários outros estão disponíveis como o Win16, WinRT, POSIX e OS/2, todos funcionando em modo usuário. E nada impede que novas personalidades sejam desenvolvidas, mesmo por empresas que não a própria Microsoft.

Adicionalmente cada personalidade roda independente das outras, ou seja, os subsistemas de ambiente não se comunicam entre si de forma direta.

2.1.5 Segurança

Define-se um sistema operacional como seguro caso ele seja efetivo ao se proteger de qualquer tipo de uso não autorizado de recursos, tanto locais como remotos. O Windows NT é um sistema operacional *Secure by design* [5] [16] [2] [4] [10], ou seja, ele implementa os princípios de menor privilégio de acesso, possui testes de prova automática para os componentes do *microkernel*, implementa testes unitários para componentes do *Executive*, é implementado no paradigma de Defesa em profundidade, onde mais de um subsistema precisa ser comprometido para comprometer todo o sistema, possui reforço de políticas de segurança seguras por padrão e verificação de traços de atividades centralizado. Adicionalmente a Microsoft se compromete a divulgar os *bugs* encontrados no Windows bem como seus *patches* o mais rápido possível, fazendo com que a norma de *Full Disclosure* também seja respeitada.

2.1.6 Performance

O Windows NT foi desenvolvido com segurança em primeiro lugar, mas vários desses procedimentos causam *overhead*. Por isso o Windows consegue bom desempenho através da melhor gestão possível do *hardware*, o que inclui o uso de estruturas de dados adequadas a cada situação e também rodar não só o *kernel*, mas todo o *Executive* em modo supervisor. As principais partes do *Executive* são o *microkernel* (responsável por agendar *threads*, lidar com interrupções de *software* e *hardware*, sincronização de processadores em sistemas multiprocessados e criação de objetos de *kernel*), a porção de código que implementa o subsistema de ambiente nativo (Win32) e os subsistemas que implementam I/O em geral (incluindo rede). Adicionalmente as partes mais críticas do *microkernel* são programas direto em Assembly. [16]

Por fim criou-se um método de comunicação entre *threads* chamado de LPC (*Local Procedure Call*), implementado em modo *kernel* que garante comunicação síncrona entre *threads*. [17] [10]

2.2 Arquitetura simplificada do Windows NT

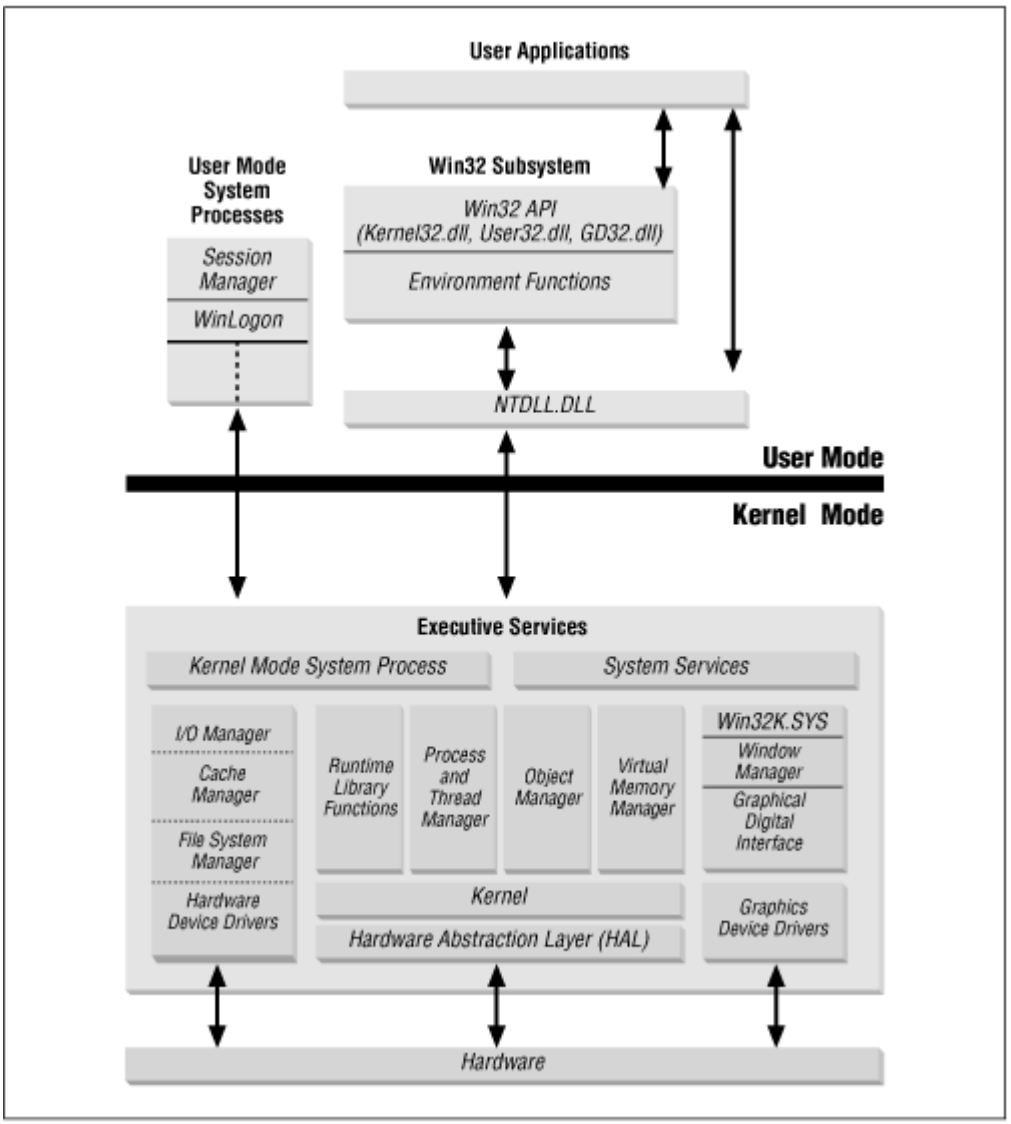


Figura 2.1: Arquitetura simplificada do Windows NT

O *Design* do Windows NT é montado em camadas, de tal forma que as partes que mais dependem do processador em que se está rodando podem ser isoladas. Assim, cria-se uma interface padrão para as camadas superiores, que não precisam se preocupar com nuances da arquitetura de *hardware* utilizada.

Além disso há uma quebra entre as várias tarefas que devem ser executadas em modo supervisor. O *Kernel*, que é a peça fundamental, é na verdade bastante minimalista enquanto as outras atividades mais complexas são feitas em outros subsistemas. Sendo

assim, trata-se de uma arquitetura *microkernel* híbrida.

2.2.1 Módulos de modo *Kernel*

Executive

É a parte mais externa dos serviços implementados em *Kernel Mode*, e na verdade serve como um *wrapper* dos outros subsistemas de *kernel* presentes. O *Executive* é responsável pelas funções que podem ser chamadas por processos rodando em modo usuário, ou seja, os serviços providos pelo sistema. A assinatura dessas funções está disponível na biblioteca Ntdll.dll.

Vale lembrar, contudo, que existem bibliotecas que também rodam em modo usuário e essas sim chamam as funções disponíveis na Ntdll.dll, facilitando assim a programação. Ou seja, um caminho mais usual é que um programa chame uma função alto-nível em alguma *API* (como uma chamada da *API* Win32) e o subsistema Win32 realize várias chamadas à Ntdll.dll para cumprir sua função.

Adicionalmente o *Executive* também prove funções para programas modo usuário que precisam conversar diretamente com *drivers*, através da chamada *DeviceIoControl*.

HAL (*Hardware Abstraction Layer*)

Camada de abstração de hardware é um módulo de *kernel* que prove interface de baixo nível entre o *hardware* e as camadas superiores do *kernel*. Essa camada é responsável por normalizar o tratamento das interfaces de I/O, controladores de interrupção, comunicação entre processadores e eventuais outras rotinas que sejam fortemente dependentes de arquitetura.

Kernel

Parte fundamental, responsável por tarefas como agendamento de *threads*, sincronização de objetos, controle das interrupções e disparo de exceções.

Device Drivers

Device Drivers (Drivers de dispositivo) são programas / módulos que podem ser carregados em modo *kernel*. Existem vários tipo de *drivers* de dispositivo, mas a maior parte deles pode ser englobada nas seguintes classes:

- *Hardware Device Drivers*, que manipulam o *hardware* para escrever ou ler de um dispositivo físico. Exemplos: *Mouse*, teclado, *pendrive*, placa de som.

- *File System drivers*, que aceitam comandos que lidam com arquivos e os traduz em pedidos de I/O para um dispositivo específico
- *File System Filter Drivers*, que ficam um nível abaixo dos *File System Drivers*. Os File *System Filter Drivers* são responsáveis por atividades como espelhamento, *RAID*, criptografia e afins.
- *Drivers* Redirecionadores de rede, que são responsáveis por fazer operações com arquivos pela rede
- *Drivers* de Protocolo, tais como TCP/IP
- *Kernel Streaming Filter Drivers*, que fazem processamento em *streams* de dados.

Cache Manager

Trata-se de um conjunto de subsistemas que funciona em modo *kernel*, com o objetivo de auxiliar o gerenciador de memória na execução de *caches* globais para todo o sistema. Devido ao fato de esse gerenciador ser centralizado, todos os tipos de arquivos e tipos de sistemas de arquivos são, por efeito colateral, gerenciados pelo *cache* sem que seja necessário a cada sistema de arquivos ou subsistema implementar sua própria rotina de *cache*.

O gerenciador de *cache* usa o gerenciador de memória para decidir quais arquivos e quais de seus pedaços devem ser mantidos em memória. O *cache* é feito de *cluster* em *cluster*, ou seja, a unidade mínima de alocação para o *cache* é o tamanho da unidade de alocação lógica utilizada na formatação do disco. Adicionalmente esse gerenciador suporta parâmetros de otimização que informam, por exemplo, se o arquivo aberto será lido de forma sequencial ou aleatória, se o arquivo é temporário, se é uma transação que não deve ser *cacheada*, entre outros.

I/O Manager

O gerenciador de *I/O* é responsável por gerenciar os dispositivos de *hardware* e prover interfaces que possam ser acessadas pelo resto do sistema. Trata-se de um componente essencial, porque é ele que define as regras que devem ser usadas para entregar pacotes de *I/O* aos *drivers* de dispositivo. Trata-se, portanto, de *I/O* direcionado a pacotes.

O gerenciador de *I/O* também é responsável por prover funções que permitem a um *driver* chamar outros *drivers*. Ele também controla os *buffers* para os pedidos de *I/O*, detecta situações de erros e *time-out* e mantém uma lista ativa de todos os sistemas de arquivos que estão instalados e em funcionamento no sistema operacional.

Virtual Memory Manager

Trata-se de um subsistema que roda em modo *kernel*. Entre suas tarefas estão traduzir e mapear os endereços de memória virtual referenciados pelos processos em endereços reais de memória e também fazer o *swapping* de páginas de memória para disco e de disco para memória, conforme o necessário e a disponibilidade de memória física.

Object Manager

O *Object Manager* é um componente modo *kernel* responsável por gerenciar todos os objetos ativos no Sistema Operacional. Dessa forma ele implementa uma centralização dos controles de recursos do sistema, facilitando a manutenção e corroborando para uma política de segurança mais sólida. Os objetivos do *Object Manager* são prover um mecanismo uniforme para o uso de qualquer tipo de recurso do sistema, isolar a proteção de objetos em um único subsistema, permitir rastrear quais recursos estão alocados para quais processos, suportar outros ambientes e ser versátil o suficiente para acomodar vários tipos de recursos, tais como dispositivos físicos, arquivos, processos, etc.

Process Manager

Trata-se de um componente que reside em modo *kernel* e é responsável por alocar os devidos recursos aos processos (*threads*) ativos no momento. O Windows implementa um sistema de escalonamento de *threads* dirigido a prioridade e preemptivo. Além disso pelo menos uma *thread* do grupo de maior prioridade que esteja pronta para execução sempre está executando, por grupo de processadores.

Runtime Library Functions

Essas bibliotecas são porções de código que podem ser instanciadas em nível *kernel* para auxiliar outros procedimentos que também executam em modo *kernel*. Alguns exemplos são bibliotecas para manipulação de *strings* ou instruções aritméticas complexas.

Kernel Mode System Process

Trata-se de um processo especial (identificado como *System*) cujo objetivo é agrupar todas as *threads* que estão rodando em modo *kernel*. Ou seja, toda a parte privilegiada do sistema operacional (subsistemas, *drivers*, etc) roda dentro deste processo, cada tarefa como uma *thread*.

System Services

São serviços de *kernel* providos pelo Windows que são chamados por processos e / ou bibliotecas rodando em modo usuário. Qualquer método chamado de qualquer uma das *APIs* do Windows que implicam alguma tarefa de *kernel* são na verdade redirecionadas para essas funções de serviço.

Window Manager

É o serviço em modo *kernel* responsável por prover a funcionalidade necessária para que as aplicações possam desenhar suas próprias janelas. Até a versão 5.2 o *Window Manager* era um sistema de janelas conhecido como *"Stacking Window Manager"*. A partir da versão 6.0 foi introduzida uma nova capacidade para que o *Window Manager* se comportasse também como um *"Compositing Window Manager"*. A partir da versão 6.2 o *Window Manager* é apenas *Compositing* e a funcionalidade *stacking* foi removida completamente.

Graphics Device Driver

Essa é a parte em que deve ser carregado o *driver* responsável por controlar diretamente a placa de vídeo. *Drivers* de vídeo não passam pelo *HAL*, mas tem acesso completo e irrestrito ao *hardware*. Ele deve interfacear com a *GDI*, que possui chamadas normalizadas e que servem de apoio para o *Window Manager*.

2.2.2 Módulos de modo Usuário

Session Manager

Esse processo é responsável por realizar as funções de inicialização que definem uma sessão no Windows. É ele que cria variáveis de ambiente, define portas de comunicação e carrega a instância do subsistema do Win32 para esta sessão. A sua instancia em sessão 0 se marca como um processo crítico, ou seja, caso ele termine o *kernel* do Windows emite um Bugcheck. Caso a inicialização seja correta e não se trate da sessão 0 ele termina de forma correta.

Winlogon

Trata-se de um processo que lida com a autenticação de sessões interativas. Ele é responsável por iniciar o primeiro processo do usuário depois que suas credenciais são confirmadas, gerenciar a finalização, travamento e desconexão da sessão atual bem como permitir a alteração de senhas. É papel do *Winlogon* garantir que durante as suas operações

nenhum outro processo seja capaz de interceptar as credenciais que estão sendo manipuladas.

Client-Server Runtime Subsystem

Esse processo é responsável por implementar a parte modo usuário do ambiente Win32. Quando um processo em modo usuário chama procedimentos que requerem janelas de console, criação de processos ou *threads* ou outras chamadas que normalmente requerem uma transição para modo *kernel*, caso essa chamada tenha sido feita pela Win32API da forma documentada, então a Win32API primeiro chama o CSRSS que faz boa parte do trabalho em modo usuário e apenas muda para o modo *kernel* quando for absolutamente necessário.

Capítulo 3

Principais subsistemas de segurança implementados até o Windows 7

Esse capítulo apresentará alguns detalhes de como são implementados os subsistemas que garantem a execução segura das tarefas dentro da arquitetura do Windows NT. [10] As informações são baseadas na versão NT 6.1 (Windows 7), mas refletem em grande parte o que está implementado na versão mais recente disponível no momento da escrita desse trabalho (NT 6.3). De acordo com o *Trusted Computer System Evaluation Criteria*, o Windows NT é certificado para o nível C2 [4] [18], em ambientes online. Mais recentemente a TCSEC foi substituída pela *Common Criteria for Information Technology Security Evaluation* e, nesse sistema, o Windows Server 2003 (NT 5.2) foi certificado como EAL4+, a categoria mais alta possível para um sistema operacional de propósito geral. [7]

3.1 SRM - *Security Reference Monitor* (Monitor de referências de Segurança)

Trata-se de um componente do *Executive* responsável por definir a estrutura de dados do *token* de acesso que representa um contexto de segurança. Ele também é responsável por fazer verificações de segurança nos objetos instanciados e por manipular os privilégios. Adicionalmente ele gera mensagens de auditoria capturadas por outras partes do sistema. Durante a inicialização do Sistema é criado um canal de comunicação entre o LSASS (*User-Mode*) e o SRM. Tal conexão é feita através do ALPC e o resultado é um canal de comunicação privado apenas entre esses dois processos. Essa comunicação é permitida ser criada apenas uma vez na inicialização, de tal forma que outros processos não conseguem, por exemplo, terminar o LSASS e refazer o pedido de comunicação. Terminar o LSASS resulta em um *BugCheck* imediado.

3.1.1 Verificações de Acesso

O modelo de segurança do Windows exige que quando uma *thread* quer acessar um objeto ela deve, antes de realizar qualquer acesso, expor quais operações ela quer fazer nesse objeto. Nesse momento o *Object Manager* faz uma chamada ao SRM, que verifica se essa *thread* tem as credenciais necessárias para realizar as operações que ela deseja no objeto em questão. Se permitido o SRM libera uma autorização para o processo que contém a *thread* que fez o pedido. Dessa forma não apenas a *Thread* que fez o pedido, mas todas as *Threads* do mesmo processo podem fazer as operações requisitadas sobre o objeto mencionado.

A essência de toda a política de funcionamento do SRM pode ser resumida em uma regra que recebe três entradas e responde com uma saída. As três entradas são 1- O objeto que representa a identidade de uma *thread*; 2- Um objeto que indique quais operações essa *thread* quer fazer sobre um objeto e 3- Um objeto que mostre as configurações de segurança do objeto que está sendo acessado. A saída é um "Acesso Permitido" ou um "Acesso Negado".

3.1.2 Identificadores de Segurança

No Windows NT, entidades que servem como base para tomada de decisões possuem SIDs (*Security Identifiers*). Um SID é uma estrutura que contém os campos Versão do SID, um número de 48 bits que representa uma autoridade e vários números de 32 bits que representam uma sub-autoridade ou um identificador relativo. Uma autoridade é um agente que distribui SIDs. Sub-autoridades são agentes que são considerados confiáveis pelos agentes que os criaram. Cada máquina ou domínio Windows possui um SID, e o Windows toma as medidas possíveis para que cada máquina possua um SID diferente.

Quando o Windows é instalado o instalador emite um SID para a máquina. Depois disso o instalador gera SIDs para as contas locais no computador. O código dos SIDs dessas contas locais são baseados no SID da máquina com um RID (identificador relativo) no final. Quando um computador se torna um controlador de domínio o SID dessa máquina se torna o SID do domínio.

Além disso o Windows define alguns SIDs para contas ou grupos que já são padrão do Windows. O SID S-1-1-0, por exemplo, representa "Qualquer usuário".

Entende-se como "entidades que servem como base para tomada de decisões" contas de usuários, grupos de usuários, sessões de logon, serviços, domínios, computadores, membros de domínio e níveis de integridade (que são um tipo de serviço).

Exemplo de SID: S-1-5-21-1463437245-1224812800-863842198-1128

- S: Todo SID começa com S

- 1: Versão / revisão do SID
- 5: Identificador de Autoridade (Nesse caso, *Windows Security Authority*)
- 21: Subautoridade 1
- 1463437245: Subautoridade 2
- 1224812800: Subautoridade 3
- 863842198: Subautoridade 4
- 1128: Identificador Relativo

3.2 Níveis de Integridade

O objetivo da introdução de níveis de integridade foi permitir usuários executarem processos com níveis de permissão inferiores aos que realmente estão disponíveis. Dessa forma objetos ao serem questionados por suas permissões apresentam seus níveis de integridade e não as permissões do usuário que o está instanciando. As permissões do usuário são consultadas apenas na criação do objeto e somente se um processo/objeto deseja criar um outro processo/objeto com um nível de integridade mais alto que seu próprio. Além disso, consultas às credenciais do usuário normalmente requerem uma intervenção manual através de uma janela de confirmação. Existem 5 níveis de integridade no NT 6.1, com a introdução de um sexto no NT 6.2:

Tabela 3.1: Níveis de Integridade do NT 6.1

SID	**Nível**	**Exemplo de Uso**
S-1-16-0x0	Não Confiável	Acesso somente-leitura
S-1-16-0x1000	Baixo	Permissão restrita.
S-1-16-0x2000	Médio	Permissão Usuário padrão
S-1-16-0x3000	Alto	Permissão Administrador
S-1-16-0x4000	Sistema	Serviços de Sistema

Todo objeto tem um nível de integridade. Além disso:

- Um processo herda o nível de integridade do seu processo pai.
- Caso um processo invoque um outro executável através da sua imagem e no objeto que representa essa imagem já tenha um parâmetro definindo o nível de integridade desse processo que será criado, o nível de integridade do processo criado é o menor entre o executável pai e o nível de integridade armazenado na imagem.

- Um processo pode criar processos filhos que possuem o mesmo nível de integridade que ele próprio ou menor. Criar processos com níveis de integridade mais altos precisam de uma chamada de elevação / *impersonation*.
- Caso um objeto não possua um nível de integridade definido em si próprio, ao ser instanciado esse objeto ganha implicitamente o nível de integridade médio.
- Existem regras que impedem a comunicação plena entre objetos com níveis de integridade baixo e níveis de integridade altos.

3.3 *Tokens*

Tokens são objetos que são usados pelo SRM para identificar o "Contexto de Segurança" de um processo ou *thread*. Um Contexto de Segurança é composto por:

- Conta de Usuário
- Grupo de Usuário
- Lista de Privilégios
- *ID* da Sessão
- Nível de Integridade
- Estado de Virtualização do *UAC*

Durante a operação de *Login*, o processo *LSASS* cria um *Token* inicial que representa o estado "Usuário fazendo logon". Depois disso, o *LSASS* verifica quais as credenciais do usuário que está fazendo *login*, e verifica se ele pertence a um Grupo de Alto Privilégio. Caso o usuário faça parte de um Grupo Alto de Privilégio ou então possua um privilégio alto, O *LSASS* cria dois *Tokens*: Um que contém todas as permissões desse usuário e um outro *Token* chamado de "*Token* Restrito" ou "*Token* Filtrado". Para cada um desses *tokens* é inicializada uma Sessão, mas o *Token* dado pelo Winlogon ao Userinit.exe é o *Token* restrito. Como processos ao serem criados herdam as permissões do processo que os criou, todos os processos do Usuário começam, portanto, executando com um *Token* Restrito. Caso algum processo precise ser elevado ele precisa passar pelo procedimento padrão de elevação, que já foi brevemente mencionado na sessão anterior.

O Mecanismo de segurança do Windows, de posse de um *Token*, precisa responder a duas perguntas: Quais objetos podem ser acessados e quais operações podem ser executadas. Essa decisão é tomada usando-se dois conjuntos de informações: O primeiro é o

SID do Usuário e os *SIDs* do Grupo do Usuário. O *SRM* usa os *SIDs* para determinar se uma *Thread* / Processo pode ter acesso a um objeto. Com os *SIDs* de grupo do *Token* é possível determinar de quais grupos o Usuário atual faz parte e daí liberar as permissões conforme os grupos. O segundo conjunto avaliado pelo *SRM* é o vetor de privilégios, que é uma estrutura que vai no final do *Token*. Nesse vetor está presente uma lista de privilégios associados a esse *Token*.

Cada *Token* pode ser de dois tipos: Um *token* primário (aquele que é responsável por identificar o contexto de segurança de um processo) e um *token* de Personificação (*Impersonation*). Um *Token* de Personificação é diferente, pois é um *token* temporário que uma *Thread* usa para mudar de contexto de segurança. *Tokens* de Personificação carregam um "Nível de Personificação" para identificar o tipo de personificação que está sendo usado.

Um *Token* também inclui uma política obrigatória para o processo, que define como o verificador de integridade deve agir ao instanciar o objeto. Existem duas políticas:

- TOKEN_MANDATORY_NO_WRITE_UP: Ativo por padrão, deixa claro que um processo não pode acessar outros processos de nível de integridade mais altos para acessos de escrita
- TOKEN_MANDATORY_NEW_PROCESS_MIN: Ativo por padrão, significa que o instanciador deve usar o menor nível de integridade possível entre o processo pai que está criando o processo filho e o valor de integridade guardado na imagem do executável.

3.3.1 *Tokens* Restritos

Um *token* restrito é criado a partir de um *token* primário ou um token já personificado. Trata-se de uma cópia do *Token* do qual ele deriva, mas com a possível remoção de privilégios do vetor de privilégios, ou então com alguns *SIDs* marcados como negados, ou então com alguns *SIDs* marcados como restritos.

Tokens restritos são úteis quando uma aplicação quer fazer operações em nome de um outro usuário, mas nem todas as permissões desse outro usuário são necessárias. Dessa forma pode-se reduzir o possível impacto ao executar código não confiável.

3.3.2 *Tokens* Filtrados

Trata-se de um tipo especial de *Token* restrito criado em máquinas que estão configuradas para usar o *UAC* (*User Account Control* = Controle de Conta do Usuário). Um *token* filtrado tem as seguintes características:

- Nível de Integridade médio
- *SIDs* de Grupo de Administradores marcados como Negados
- Todos os privilégios do vetor de privilégios são removidos, exceto Notificações, Desligar, Desacoplar, aumentar *Working Set* e Relógio.

Por fim, os campos de um *Token* são imutáveis (estão em espaço de *Kernel*, inacessíveis a processos de usuário). Apenas alguns campos podem ser modificados e apenas mediante as chamadas de sistema corretas.

3.4 *Impersonation* (Personificação)

Impersonation é um mecanismo usado no Windows que permite um certo processo / *Thread* adotar o perfil de segurança de um outro usuário. Esse outro usuário pode ter mais ou menos privilégios do que os privilégios atuais do processo. Esse mecanismo permite, portanto, avisar o *SRM* que, nesse momento, um processo está executando em nome de um outro usuário que possui um outro contexto de segurança. Quando o *SRM* faz as verificações por permissão ele faz essa verificação no *token* de personificação e não no *token* que representa o contexto de segurança do processo.

Impersonation não significa que o processo mudou de contexto de segurança. As estruturas de controle de *Thread* têm um parâmetro opcional que informam se elas possuem um *token* de *Impersonation*, mas o *token* original da *Thread* é mantido durante todo o processo. Nesse *token* original estão as permissões originais que a *Thread* possuía ao ser criada.

Essas formas de *Impersonation* são bastante úteis quando um certo processo quer fazer uma atividade específica em nome de um outro cliente, mas não é possível executar um programa inteiro dessa forma. Além disso, *tokens* de personificação não podem acessar arquivos / impressoras / diretórios de rede a não ser que ele próprio possua as credenciais necessárias. Caso seja necessário executar uma aplicação inteira no contexto de um outro usuário, esse outro usuário precisa estar logado com uma sessão ativa.

Para evitar casos de *Spoofing* em que um processo de baixa integridade pudesse simplesmente perguntar as credenciais de um usuário (como Usuário e Senha, por exemplo) e depois fizesse uma chamada de sistema para se auto-elevar, o Windows não permite que *Threads* se auto-elevem. Uma *Thread* só pode pedir *tokens* de permissão iguais ou menores do que ela própria, independente das credenciais de usuário que estão sendo utilizadas.

3.5 Contas Virtuais de Serviço

O Windows provê um tipo especial de contas chamados de "Contas Virtuais de Serviço", cujo objeto é aumentar o isolamento de segurança e o controle de acesso de serviços do Windows com o mínimo de esforço possível. Como todo processo precisa obrigatoriamente rodar em uma conta de Usuário, sem esse mecanismo um processo de serviço ou rodaria necessariamente em uma conta padrão do sistema ou então em uma conta de domínio, mas isso não seria apropriado já que as contas padrão do Sistema possuem privilégios muito altos e não podem ser alteradas, pois isso causaria problemas em alguns dos serviços que realmente precisam dessas permissões. Por isso, decidiu-se por um modelo em que cada serviço deveria, para o melhor isolamento possível, ter sua própria credencial.

O nome de uma conta de serviço virtual é sempre da forma NT SERVICE\nomedoservico. Contas virtuais de serviço podem ter acessos permitidos / revogados e podem ser gerenciadas pela ferramenta de política de grupo. Por outro lado, não podem ser criadas ou deletadas através da interface padrão que cria e deleta contas de usuário normais.

3.6 Descritores de Segurança

Um descritor de segurança é uma estrutura de dados que armazena quais as permissões que um dado usuário tem sobre um objeto. Os campos presentes nessa estrutura são:

- Número de Revisão: Versão do *SRM* que foi usada para gerar o descritor
- *Flags*: Campos opcionais que modificam as características / comportamento do descritor
- *Owner SID*: O *SID* do dono desse objeto
- *Group SID*: O *SID* do grupo primário para esse objeto
- *DACL*: Lista de Controle de Acesso Discreta: Especifica quem tem acesso a esse objeto e as operações permitidas
- *SACL*: Lista de Controle de Acesso do Sistema: Especifica quais operações feitas por quais usuários devem ficar armazenadas no *log* de auditoria de segurança e também guarda o nível de integridade explícito desse objeto

3.6.1 *ACL*: Lista de controle de acesso

Uma *ACL* é composta de um cabeçalho seguido de 0 ou mais entradas de controle de acesso. Existem dois tipos de *ACLs*: *DACLs* (Lista de Controle de Acesso Discreta) e *SACLs* (Lista de controle de acesso do sistema).

DACL

Em uma *DACL* cada entrada de controle de acesso contém um *SID* e uma máscara de acesso que especifica os direitos de acesso (Leitura, escrita, excluir, executar, etc) que são permitidos ou negados para o *SID*. Existem 9 tipos de *ACEs*:

1. Acesso Permitido
2. Acesso Negado
3. Objeto Permitido
4. Objeto Negado
5. Permitido Chamada
6. Negado Chamada
7. Permitido chamada de objeto
8. Negado chamada de objeto
9. Permissões condicionais

As permissões que estão relacionadas a Objetos (como Objeto permitido) são para uso dentro do *Active Directory*. Entradas de Controle de acesso desse tipo possuem um *GUID* que indica que essa *ACE* se aplica apenas para um conjunto particular de objetos ou subobjetos.

A soma de direitos de acesso de todas as *ACEs* dentro de uma *ACL* definem as permissões garantidas por essa *ACL*. Se não existe uma *DACL* em um descritor de segurança então o objeto pode ser acessado com permissão total por todos os usuários. Por outro lado, se existe uma *DACL* mas que não possui nenhuma *ACE*, significa que nenhum usuário tem permissão alguma para esse objeto.

Adicionalmente uma *DACL* possui uma lista de *flags* que definem como o sistema deve se comportar ao propagar essas informações de segurança quanto a herança.

SACL

Uma *SACL* pode ter dois tipos de *ACEs*: *ACE* de auditoria do sistema e *ACE* de auditoria de objeto do sistema. Essas entradas de acesso especificam quais operações feitas no objeto por usuários ou grupos específicos devem ser armazenadas em um *log*. Informações como essa são chamadas informações de auditoria, e estão armazenadas no *log* de auditoria do sistema. Assim como no caso da *DACL*, *ACEs* de objetos possuem um *GUID* indicando

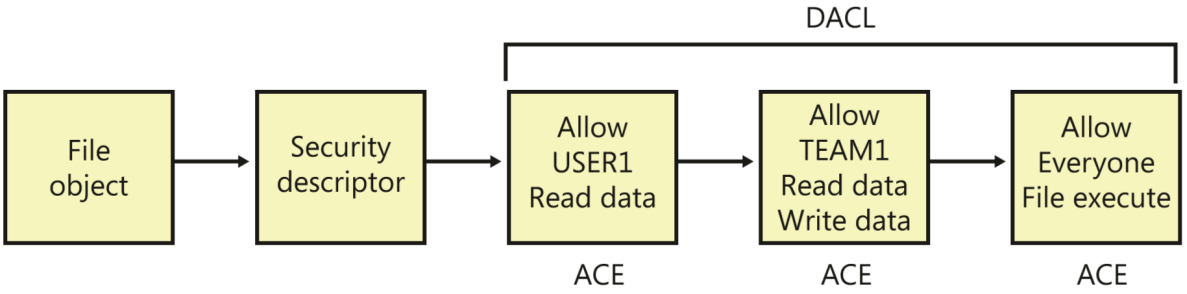


Figura 3.1: Exemplo de um objeto do tipo Arquivo e sua *DACL*

os tipos de objetos ou subobjetos aos quais ela se aplica. Se uma *SACL* não está presente, não há auditoria do objeto. Assim como no caso de *DACL*, *SACLs* podem especificar parâmetros para herança de suas *ACEs*.

Distribuição de ACL

Quando um objeto é criado, o sistema precisa determinar quais permissões colocar na *DACL* desse novo objeto. O Windows usa a primeira regra que se encaixar dentre as seguintes:

1. Se um chamador de forma explícita prover um descritor de segurança, então o sistema aplica esse descritor ao objeto. Se o objeto tiver um nome e estiver dentro de um contêiner o sistema copia as entradas do *DACL* do chamador para o objeto de acordo com as *flags* de herança.

2. Se um chamador não prover um descritor de segurança e o objeto tiver um nome, o sistema de segurança olha para o descritor de segurança do contêiner em que o objeto será armazenado. Algumas das *ACEs* do diretório em que ele será guardado possivelmente estão marcadas como herdáveis, ou seja, devem ser aplicadas a novos objetos criados nesse diretório. O sistema então propaga essas *ACEs* herdáveis.

3. Se não há descritor de segurança e o objeto não herda nenhuma *ACE*, o sistema busca a *DACL* padrão do *token* de acesso do chamador e a coloca no objeto. Alguns subsistemas do Windows possuem *DACLs* padrão *hard-coded*, e elas são então herdadas a esses objetos.

4. Se não há descritor de segurança, nem *ACEs* que podem ser herdadas, nem *DACLs* padrão, o sistema cria o objeto sem uma *DACL* o que garante acesso por qualquer um.

As regras para distribuição de *SACLs* são praticamente as mesmas, com algumas exceções. A primeira é que a *flag* que define a não propagação por herança é diferente. A segunda é que caso não haja herança de nenhuma *ACE* para o novo objeto então ele é logo criado sem nenhuma *SACL*, já que não existem *SACLs* padrão como no caso das *DACLs*.

Quando um novo descritor de segurança que contém *ACEs* que podem ser herdáveis é aplicado em um contêiner o sistema automaticamente propagada essas *ACEs* para os objetos filhos, desde que eles estejam marcados para obter *ACEs* por herança. Caso haja algum tipo de conflito entre as *ACEs* que estão sendo propagadas por herança e uma *ACE* que já esteja no objeto, então *ACEs* que já estão no objeto têm preferência. Resumidamente:

- Se um objeto filho sem *DACL* herda *ACEs*, o resultado é esse objeto filho com uma *DACL* que contém as *ACEs* herdadas
- Se um objeto filho com uma *DACL* vazia herda *ACEs*, o resultado é um objeto filho com uma *DACL* que contém as *ACEs* herdadas.
- Para objetos que estiverem no *Active Directory*, se uma *ACE* é removida de um objeto pai, o sistema de herança também remove essa *ACE* dos objetos filhos
- Para objetos que estiverem no *Active Directory*, se uma operação de herança resulta na remoção de todas as *ACEs* da *DACL* de um objeto, o objeto filho fica com uma *DACL* vazia ao invés de ficar sem *DACL* alguma.

3.6.2 Determinação de Acesso

O Windows determina o acesso a um objeto de duas maneiras:

- Através da verificação do nível de integridade obrigatório, que determina se o nível de integridade do processo chamador é alto o suficiente para acessar o recurso, baseado no nível de integridade do recurso e na sua política de obrigatoriedade.
- Verificação de acesso discreto, que determina que um usuário / conta possui acesso a um objeto

Por razões de *performance*, quando um processo tenta abrir um objeto primeiro verifica-se a compatibilidade dos níveis de integridade. Caso isso passe o sistema procede e verifica as informações da *DACL*. Dado que um processo possui um nível de integridade ele pode abrir objetos para escrita desde que seu nível de integridade seja maior ou igual ao nível de

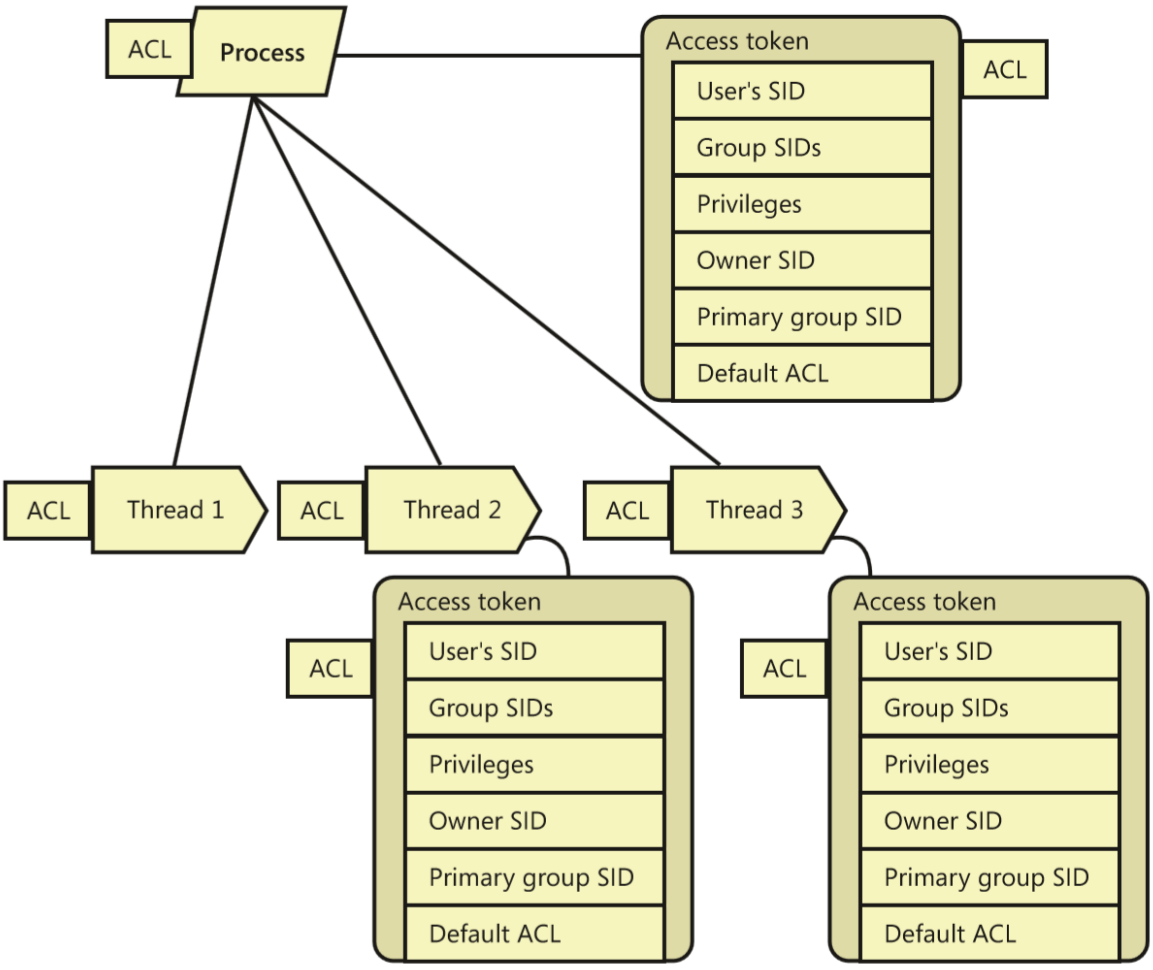


Figura 3.2: Exemplo de um objeto do tipo processo e suas *ACLs*

integridade do objeto que ele deseja escrever. Verificações de Integridade tem precedência, ou seja, se um processo não possui acesso por incompatibilidade de nível de integridade não importa ele ter armazenado na *DACL* que ele possui permissão para acessar esse objeto. O acesso para escrita será negado.

Com as políticas padrão de integridade um processo de integridade qualquer pode abrir qualquer objeto para leitura (exceto objetos do tipo Processo, *Thread* e *Token*), dado que sua *DACL* o permita tal acesso. Objetos do tipo Processo, *Thread* e *Token* estão definidos como *No-Read-Up*.

Durante a etapa de verificação da *DACL*, o sistema procede na seguinte ordem para determinar o maior acesso possível a um objeto

- Se o objeto não tiver *DACL*, o objeto não tem proteção e o sistema libera acesso irrestrito
- Se o chamador tem privilégio de "tomar como dono", o sistema de segurança permite direito de "sobrescrever dono" antes de examinar a *DACL*.
- Se o chamador é o dono do objeto, o sistema procura por um *SID* de "Direitos do Dono" (*Owner Rights*) e procede as demais avaliações de acordo com esse *SID*, não o do chamador. Se não for possível, libera acessos de leitura, controle e escrita de *DACL*.
- Para cada *ACE* de acesso negado que contém um *SID* que casa com um dos *SIDs* do token de acesso do chamador, a máscara de acesso permitido é removida da *ACE*. (Ou seja, nega os acessos marcados na *ACE* que devem ser negados)
- Para cada *ACE* de acesso permitido que contém um *SID* que casa com um dos *SIDs* do *token* de acesso do chamador, a máscara de acesso permitido é adicionada na *ACE* sendo computada, a não ser que ela já tenha sido negada no passo anterior (conflitos de permitir/negar sempre são definidos como negar).

Ao final, a máscara contendo a maior quantidade de acessos permitidos é retornada ao processo chamador. Essa máscara representa a permissão efetiva daquele processo para com aquele objeto naquele momento.

Adicionalmente, existe um outro passo de verificação que serve para determinar se um acesso ou operação específica está permitida, baseado no *Token* de acesso do chamador. Nessa situação o sistema procede da seguinte maneira:

1. Se o objeto não tiver uma *DACL*, o objeto não tem segurança e o sistema libera essa operação

2. Se o chamador do procedimento tiver o privilégio "tornar-se dono", o sistema de segurança garante permissão de escrita no campo de "dono" do objeto.

3. Se o chamador é o dono do objeto, o sistema procura por um *SID* de dono do objeto e passa a usar esse *SID* para os próximos passos.

4. Cada entrada de controle de acesso na *DACL* é examinada da primeira até a última. Uma *ACE* é processada se uma das seguintes condições forem satisfeitas:

   - Existe uma *ACE* que é de negar acesso, e o *SID* nessa *ACE* casa com um *SID* ativo ou então com um *SID* de sempre negar no *token* de acesso do chamador
   - Existe uma *ACE* de acesso permitido, e o *SID* nessa *ACE* casa com um *SID* ativo do chamador que não é do tipo sempre negar
   - Esta é a segunda verificação por permissões de acesso ou operação e o *SID* na *ACE* casa com um *SID* restrito do *token* de acesso do chamador
   - A *ACE* não está marcada como "herdar somente"
   - Se existe uma *ACE* de acesso permitido, os direitos presentes na máscara dessa *ACE* são liberados. Se todas as permissões foram aceitas, a verificação de acesso funcionou. Se existe uma *ACE* de acesso negado e algum dos acessos pedidos estão negados, a verificação de acesso nega a permissão
   - Se acabou a *DACL* e ainda não foi possível determinar direitos suficientes para alguma operação, o acesso é negado.
   - Se todos os acessos foram permitidos, mas o *Token* de acesso do chamador contém pelo menos um *SID* restrito, o sistem reverifica as *ACEs* da *DACL* procurando por *ACEs* com máscaras de acesso que casam com os pedidos de permissão feitos pelo chamador e tenta casá-las com algum *SID* da lista de *SIDs* restritos. Se for possível achar essa *ACE* e casá-la com um *SID* restrito, o acesso é negado.

O comportamento dos algoritmos de validação de permissão dependem da ordem relativa das *ACEs* dentro de uma *DACL*. Dessa forma, *ACEs* que negam acesso sempre são armazenadas antes de *ACEs* que permitem acesso. Vale lembrar que, por motivos de eficiência, o *SRM* faz uma verificação de permissão apenas quando um *Handle* é aberto. Ou seja, se uma *ACE* mudar enquanto um processo já possui um *Handle* o processo continua com os privilégios que ele tinha no momento em que o *Handle* foi criado. Além disso componentes que rodam no nível do *Executive* não passam por verificações de acesso, já que todos os componentes que estão no *Executive* são considerados seguros ao ponto de

fazerem parte do próprio sistema operacional, mesmo que não o sejam originalmente (no caso de *Drivers* e outros programas de *Executive*, por exemplo).

Além disso, devido ao fato de o dono de um objeto sempre ter direito de escrita na *DACL*, usuários não podem ser negados acesso a objetos que eles mesmos criaram a não ser que o dono desse objeto também seja mudado (por exemplo, por um usuário que possua permissão de "tornar-se dono"). Caso uma *ACE* seja ingenuamente alterada por um usuário cujo dono do objeto é na verdade um outro usuário, esse outro usuário poderá simplesmente alterar as permissões presentes na *DACL*, já que ele é o dono.

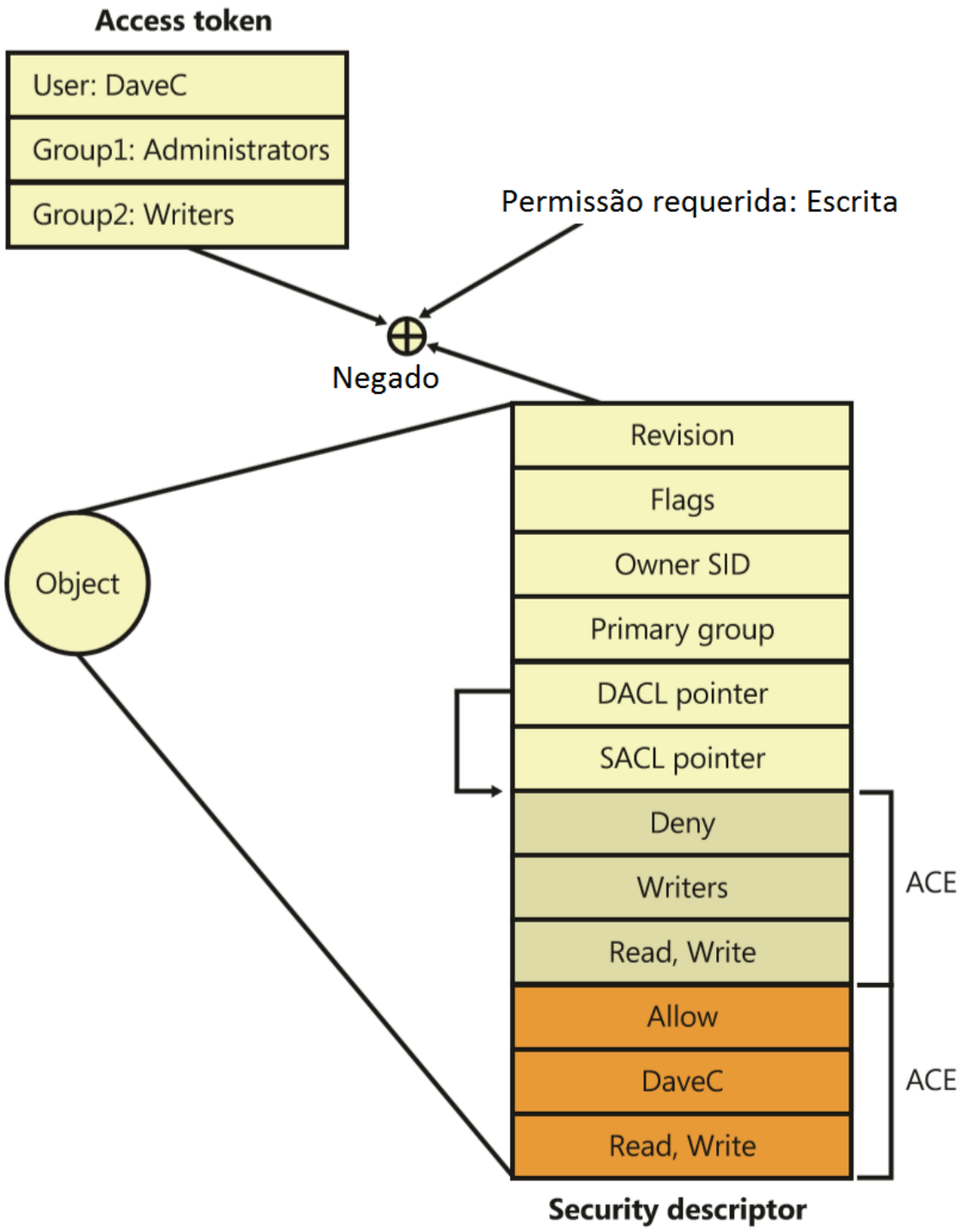


Figura 3.3: Ilustração gráfica de uma verificação de acesso

3.7 Isolamento de privilégios da Interface de Usuário

Devido ao isolamento criado pelo sistema de controle de integridade, não é possível que um processo de integridade menor envie mensagens de janela para um processo de integridade maior. Contudo, para facilitar alguns cenários, algumas mensagens foram adicionadas em uma lista de mensagens permitidas. Tratam-se basicamente, de mensagens que permitem a um processo de baixa integridade obter algumas informações sobre processos de maior integridade, mas nenhuma delas permite escrita e nenhuma delas permite interceptação de mensagens potencialmente confidenciais.

A separação entre processos por níveis de integridade, nesse caso, impede que processos de baixa integridade insiram dados em processos de maior integridade. Além disso, torna-se muito mais difícil executar ataques que causem *buffer overflows* nos processos de maior integridade, o que poderia levar a execução de algum código arbitrário.

Também tomou-se cuidado para que *hooks* que estejam em uma janela de baixa integridade não consigam capturar informações sensíveis de janelas que estejam rodando em maior prioridade, como por exemplo as teclas que estão sendo digitadas.

Processos como o teclado virtual e outros que precisam inserir entrada em outros processos possuem uma *flag* especial, e na inicialização do sistema eles têm integridade entre 0x2000 e 0x3000 para um usuário comum, ou então integridade alta para um usuário administrador. Para evitar que um processo qualquer possa se passar pelo *OSK*, existe uma rotina de verificação que testa pela autenticidade da aplicação mediante uma assinatura digital, e é obrigatório que o processo esteja em um dos "lugares seguros" do sistema operacional.

3.8 Direitos de um Usuário e Privilégios de Sistema

Existem algumas operações que não podem ser validadas através do modelo de verificação de permissão de objeto, já que tratam-se de operações que não interagem com objeto algum. Um exemplo muito comum é um usuário operador de *backup*: Esse usuário precisa ter permissão para copiar um arquivo, embora não seja necessário que ele possua direito para ler o conteúdo do arquivo. Dessa forma, criou-se um outro paradigma de autorização para os sistemas Windows que permitem tratar situações como essas.

3.8.1 Direitos de um Usuário

Os Direitos de um Usuário são controlados por um subsistema próprio, o *LSA* (*Local Security Authority* - Autoridade local de Segurança). Quando um Usuário faz Login em um sistema o *LSA* vai até sua base de dados e busca os Direitos que esse usuário deve

ter. Existem 5 direitos:

- *Logon* Interativo: Fazer *Login* fisicamente na máquina em questão
- *Logon* pela Rede: Fazer logon que se origina por um computador remoto, para um acesso a arquivo compartilhados, por exemplo
- *Logon* pelo *Terminal Service*: Fazer logon por uma sessão de *Terminal Service*, que permite a um usuário operar remotamente uma máquina através de um *Shell* gráfico completo
- *Logon* como Serviço: Fazer *logon* como uma conta de serviço, sem sessão interativa
- *Logon* como executador de *Batch*: Fazer *logon* e submeter tarefas não interativas ao agendador de tarefas de modo sequencial.

3.8.2 Privilégios

Ao contrário dos Direitos de um Usuário que é verificado por apenas um subsistema *(LSA)*, os privilégios de execução de um usuário são definidos por vários componentes, no momento em que eles vão realizar alguma atividade. Dessa forma, cada componente passa a ser responsável por verificar o privilégio desejado. Além disso, diferentemente dos Direitos de um Usuário, privilégios podem ser ativados e desativados enquanto um usuário está ativo. Para que um usuário seja marcado como possuidor de um certo privilégio, o privilégio deve estar no *token* sendo analisado e deve estar marcado como ativo. A ideia por trás disso é que privilégios deveriam ser liberados apenas quando seu uso é realmente necessário, impedindo assim que um processo mal intencionado faça uma operação que cause algum impacto de segurança.

Abaixo segue uma lista de todos os privilégios disponíveis no Windows:

Tabela 3.2: Lista de Privilégios disponíveis no Windows

Privilégio	Direitos do Usuário	Momento de verificação
SeAssignPrimaryTokenPrivilege	Muda um *token* de processo	Verificado ao criar e modificar *tokens* de processos
SeAuditPrivilege	Faz auditoria de Segurança	Requerido para escrever no *log* de eventos de segurança, verificado pela *API ReportEvent*

SeBackupPrivilege	Faz *Backup* de arquivos e diretórios	Ao acessar um arquivo, faz com que o *NTFS* atribua as permissões *READ_CONTROL, ACESS_SYSTEM_SECURITY, FILE_GENERIC, FILE_TRAVERSE* independentemente do descritor de segurança presente
SeChangeNotifyPrivilege	Ignorar verificação de travessia de caminho	Usado ao acessar arquivos, para dizer ao *NTFS* para não olhar as permissões de todos os diretórios do caminho, mas apenas o do arquivo em questão. Também serve para liberar aplicativos para receber notificações de quando há alguma alteração do sistema de arquivos
SeCreateGlobalPrivilege	Criar objetos globais	Requerido quando um processo quer criar *links* simbólicos nos diretórios que estão contidos no espaço de nomes do *Object Manager* e que estão atribuídos a uma sessão diferente do processo chamador
SeCreatePageFilePrivilege	Criar um arquivo de paginação (área de *Swap*)	Verificado pelo *NTCreatePagingFile*, que é responsável por criar arquivos de *swap*
SeCreatePermanentPrivilege	Criar objetos compartilhados permanentes	Verificado pelo *Object Manager* quando se cria objetos permanentes
SeCreateSymbolicLinkPrivilege	Criar links simbólicos	Verificado pelo *NTFS* na criação de *links* simbólicos
SeCreateTokenPrivilege	Criar um objeto do tipo *Token*	Verificado pelo *NtCreateToken* quando alguém pede para criar um objeto *token*

SeDebugPrivilege	Depurar programas	Se o chamador tiver esse privilégio ele pode acessar qualquer outro processo ou *thread* independentemente do conteúdo armazenado no descritor de segurança do processo alvo e que estejam no mesmo nível de integridade
SeEnableDelegationPrivilege	Permite considerar um sistema / usuários desse sistema confiáveis o suficiente para delegação de nomes	Usado pelo *Active Directory* para determinar quem pode criar novas credenciais
SeImpersonatePrivilege	Permite personificar um cliente após autenticação	O *Process Manager* verifica esse privilégio quando uma *Thread* quer usar um *token* diferente para personificação (caso do *UAC*, por exemplo) e o *token* alvo é de um usuário diferente do *token* do usuário atual
SeIncreaseBasePriorityPrivilege	Aumenta a prioridade de um processo no escalonador	Verificado quando alguém quer elevar a prioridade de um processo
SeIncreaseQuotaPrivilege	Ajusta cotas de memória e *CPU* para um processo	Verificado quando um processo ultrapassa seu limite máximo pré definido de *working set* e de uso de *CPU*
SeIncreaseWorkingSetPrivilege	Aumenta o *working set* de um processo	Requerido para se aumentar o *Working Set* mínimo de um processo. Se o processo fizer um *VirtualLock* nessa área, o escalonador de memória passa a não considerar essa área como passível de *swap* para disco

SeLoadDriverPrivilege	Permite carregar *drivers*	Verificado ao se carregar *drivers*
SeLockMemoryPrivilege	Trava páginas em memória	Usado pelo *NtLockVirtualMemory*, que é um equivalente do *VirtualLock* só que em modo *kernel*
SeMachineAccountPrivilege	Adicionar estaçoes ao domínio	Verificado pelo *SAM* em um controlador de domínio quando alguém cria uma conta representando uma máquina nesse domínio
SeManageVolumePrivilege	Fazer tarefas que envolvam manipulação de volumes	Verificado pelos *drivers* de sistemas de arquivos quando é feita uma operação do tipo "abrir volume", necessária por exemplo para fazer *CheckDisk* ou *Defrag*.
SeProfileSingleProcessPrivilege	Fazer uma medição de performance de um único processo	Verificado pelo *Prefetcher* e pelo *Superfetch* quando eles precisam de informações de performance de algum processo, que é obtida pela chamada *NtQuerySystemInformation*
SeRelabelPrivilege	Modificar o nome de um objeto	Verificado pelo *SRM* quando se faz uma elevação de integridade
SeRemoteShutdownPrivilege	Permitir desligamento por *logon* remoto	Verificado quando se quer desligar o sistema
SeRestorePrivilege	Restaurar arquivos e diretórios	Libera as seguintes permissões, independente do descritor de segurança presente: *WRITE_OWNER, ACESS_SYSTEM_SECURITY, FILE_GENERIC_WRITE, FILE_ADD_FILE, FILE_ADD_SUBDIRECTORY, DELETE*

SeSecurityPrivilege	Gerenciar o *log* de auditoria	Verificado para se acessar o *SACL* de um descritor de segurança e para ler ou escrever no *log* de segurança
SeShutdownPrivilege	Desligar o computador	Verificado ao se iniciar o desligamento de um sistema
SeSyncAgentPrivilege	Sincronizar dados de um diretório	Requerido para ser usar o serviço de sincronização *LDAP (Active Directory)*. Permite ler todos os objetos e propriedades no diretório independentemente da proteção marcada nos objetos e propriedades
SeSystemEnvironmentPrivilege	Modificar variáveis de ambiente de *firmware*	Requerido pelo *NtSetSystemEnvironmentValue* e *NtQuerySystemEnvironmentValue* para modificar variáveis de ambiente de *firmware* através da *HAL*
SeSystemProfilePrivilege	Permite fazer medições de *performance* do sistema	Verificado sempre que alguém quiser perguntar ao sistema informações de performance
SeSystemtimePrivilege	Mudar a data/hora do sistema	Requerido ao se mudar a data/hora
SeTakeOwnershipPrivilege	Tornar-se dono de objetos	Necessário quando se quer tornar dono de um objeto mas não possui permissão suficiente no objeto

SeTcbPrivilege	Agir como parte do Sistema Operacional	Verificado pelo *SRM* quando um *ID* de sessão é atribuído a um *token*, quando o gerenciador *Plug and Play* cria objetos, quando *BroadcastSystemMessageEx* é chamado, quando *LsaRegisterLogonProcess* é chamado e quando se marca a execução de um processo em uma *VDM*
SeTimeZonePrivilege	Mudar a zona de tempo	Requerido ao se mudar a zona de tempo
SeTrustedCredManAccessPrivilege	Acessar o gerenciador de credenciais como um chamador confiável	Verificado pelo *Credential Manager* para verificar se ele deve confiar no chamador com as credenciais que foram fornecidas, que podem inclusive terem vindo de texto puro. Por padrão, apenas o *Winlogon* se encaixa nesse requisito.
SeUndockPrivilege	Remover um computador de uma estação de *Dock*	Verificado pelo gerenciador *Plug and Play* quando alguém tenta desconectar um dispositivo ou o próprio computador
SeUnsolicitedInputPrivilege	Receber dados não solicitados por um dispositivo terminal	Privilégio não usado

3.8.3 Super Privilégios

Alguns dos privilégios mencionados na sub-seção anterior são muito poderosos, e caso um usuário os tenha, na prática ele pode fazer qualquer modificação no sistema de forma praticamente irrestrita e dessa forma conseguir acesso a qualquer tipo de recurso disponível na máquina.

- ***SeDebugPrivilege* - *Debugar Programas*:** Um usuário que tenha esse privilégio pode acessar qualquer outro processo no sistema (com exceção de processos protegidos) independentemente do descritor de segurança do processo alvo.
- ***SeTakeOwnershipPrivilege* - Tornar-se dono:** Esse privilégio permite a um usuário tornar-se dono de qualquer tipo de objeto (inclusive processos ou *threads* protegidos). Como um usuário dese tipo sempre tem autorização para ler e escrever nas *DACLs*, então ele pode simplesmente colocar seu *SID* nessa *DACL*, o que permite a ele se tornar dono desse objeto e daí em diante fazer qualquer operação desejada.
- ***SeRestorePrivilege* - Restaurar arquivos e Diretórios:** Com essa autorização, um usuário pode substituir arquivos do sistema sem ter permissão efetiva para tal.
- ***SeLoadDriverPrivilege* - Carregar dispositivos de *driver*:** Esse privilégio permite a um usuário instalar qualquer tipo de programa que rode dentro do *Executive*, ou seja, em *kernel mode*.
- ***SeCreateTokenPrivilege* - Criar um objeto de *Token*:** Com esse privilégio, um usuário pode gerar *tokens* que representam usuários com permissões arbitrárias.
- ***SeTcbPrivilege* - Agir como parte do sistema operacional:** Esse é um privilégio usado por um mecanismo de comunicação entre processos comuns e o *LSASS*. Usuários que tiverem esse privilégio podem interagir com o *LSASS* como se fizessem parte do Sistema Operacional

3.9 Sistema de *Logon*

O momento do Logon em sistemas Windows é crítico. Para logons interativos (aqueles em que se deseja acesso físico à máquina) existe uma cooperação entre o *Winlogon*, o *LogonUI*, o *LSASS* e o *SAM* (ou *Active Directory*).

3.9.1 *Winlogon*

O *Winlogon* é um processo considerado confiável responsável por gerenciar operações relacionadas a segurança para usuários interativos. Ele é responsável por coordenar o *Logon*, iniciar o primeiro processo de um usuário logo após um *login*, faz os procedimento de *logoff*, dispara o *LogonUI* (interface em que o usuário digita suas credenciais), alterar senhas, travar e destravar a estação, entre outras funções. O *Winlogon* deve garantir que operações relativas à segurança do sistema não sejam visíveis a outros processos.

Para diminuir a chance de *bugs* causarem falhas no processo do *Winlogon* (o que resultaria em um *crash* de sistema, já que o *Winlogon* é considerado processo crítico), um processo separado (*LogonUI*) que é o responsável por mostrar a interface do usuário e carregar os provedores de identidade. Sempre que o *Winlogon* precisa de credenciais que o usuário deve digitar, ele instancia uma cópia do *LogonUI*. Quando o *LogonUI* obtém essas informações, ele as passa para o *Winlogon* e finaliza sua própria execução.

O *Winlogon* é o único processo que intercepta mensagens de *logon* do teclado, que são enviadas por uma mensagem *RPC* vinda direto do Win32k.sys. Tão cedo quanto ele consiga as informações de *login* que devem ser usadas de um provedor de credenciais, ele chama o *LSASS* para tentar autenticar o usuário que está pedindo autorização para *Login*. Se essa autenticação for positiva então o *Winlogon* dispara o primeiro processo do usuário de acordo com as regras já descritas nas seções anteriores.

Sequencia de inicialização do *Winlogon*

Durante a inicialização do sistema o *Winlogon* procede da seguinte maneira:

1. Cria e abre uma estação interativa que representa o *mouse*, monitor e teclado. Essa estação possui apenas uma *ACE* com apenas o *SID* do sistema. Dessa forma, nenhum outro processo consegue tomar o controle, a não ser que o *Winlogon* permita.

2. Cria e abre duas áreas de trabalho: A área de trabalho interativa (comum, onde devem rodar os processos do usuário) e uma área de trabalho separada para o próprio *Winlogon* (conhecida como área de trabalho segura). Apenas o *Winlogon* tem acesso à área de trabalho do *Winlogon*. A área de trabalho interativa pode ser acessada tanto por processos do usuário como pelo *Winlogon*. Dessa forma, quando o *Winlogon* toma o controle e exibe a área de trabalho protegida, apenas o próprio *Winlogon* consegue capturar os dados que lá são inseridos.

3. Estabelece uma conexão segura com o *LSASS*.

4. Registra o Servidor de Mensagens *RPC* do *Winlogon*, que escuta pela *SAS* (*Secure Attention Sequence - Ctrl+Alt+Delete*, ou *Windows+Power*), sinais de *Logoff* e travamento da área de trabalho.

3.10 Controle de Conta do Usuário e Virtualização de Recursos

O Controle de Conta de usuário foi implementado com o objetivo de forçar usuários a executarem seus programas com nível de privilégio reduzido ao invés de executá-los

como administrador o tempo todo. Sem esses direitos administrativos processos não conseguem alterar configurações do sistema, desativar anti-vírus, nem alterar dados de outros usuários. Dessa forma, trata-se de um sistema de mitigação de *malware*, que tenta restringir as possíveis alterações maliciosas que um *malware* teria a sua disposição.

Por outro lado, várias tarefas realmente precisam de direitos administrativos para serem executadas, tais como alterar a hora do sistema, instalar programas, alterar configurações de impacto global, etc. Sendo assim, o mecanismo de Controle de Conta do Usuário permite executar todos os processos em modo restrito e, apenas quando há necessidade, esse processo é elevado.

3.10.1 Virtualização do Sistema de Arquivos e do Registro

Embora alguns programas realmente precisem de direitos administrativos para executarem suas funções a maior parte dos programas que acabam precisando dessas permissões apenas precisam delas para armazenar informações em locais globais, que normalmente não permitem escrita por um usuário restrito. Isso acontece muito em programas mais antigos, que não levam em conta o fato de que um computador pode ser compartilhado por vários usuários. O correto seriam os programas armazenarem suas informações no contexto do usuário que estão o executando e não no contexto da máquina toda.

Para diminuir uma grande quantidade de elevações desnecessárias o Windows passou a implementar uma virtualização de registro e de sistema de arquivos. Quando um programa sem permissão tenta alterar dados em um repositório (tanto arquivo como chave de registro) global, ao invés do sistema devolver uma sinalização de "Acesso Negado" ele silenciosamente redireciona tais chamadas para um armazenamento virtual, em que cada usuário possui o seu. Igualmente, no momento da leitura de um dado de um repositório global o sistema primeiro verifica se alguma instância desse dado está disponível no repositório local do usuário. Caso não exista tal instância, então a informação é lida a partir do repositório global.

O Windows sempre implementa esse tipo de virtualização, exceto nos seguintes casos:

- O aplicativo é de 64bit. A idéia é que, como esse recurso de virtualização serve apenas para compatibilidade com programas antigos, programas que são mais novos devem seguir as regras de implementação para o Windows de forma correta. Sendo assim, programas de 64bit que tentarem acessar repositórios globais receberão acesso negado.

- O aplicativo já está executando com direitos de administrador. Supõe-se, nesse caso, que realmente se deseja fazer uma alteração ao repositório global.

- A operação foi originada por um chamador rodando em modo *Kernel*

- A operação está sendo feita enquanto o programa chamador está sendo personificado.
- O aplicativo sendo executado possui um *Manifest* dizendo que ele não deve ser virtualizado
- O usuário administrador também não teria permissão para modificar o repositório global em questão: Para manter consistência, já que nesse caso o programa falharia mesmo se estivesse rodando em modo administrativo
- Se o aplicativo estiver rodando como um serviço

Além disso, o Windows cuida para que operações básicas que dependam de um "Status" de Administrador não interfiram com a execução do programa. Por exemplo, pedidos para consultar se o usuário atual pertence ao grupo de administradores sempre retorna verdadeiro, mesmo que ele ainda não tenha de fato os privilégios de um administrador.

Virtualização do Sistema de Arquivos

Os locais que são considerados repositórios globais e que devem ser virtualizados para programas antigos são o *%ProgramFiles%, %ProgramData% e %SystemRoot%*. Arquivos com extensão .exe, .bat, .scr, .vbs e outros executáveis não são virtualizados.

A implementação da virtualização do sistema de arquivos é feita através de um *Filter Driver*, ou seja, ela é completamente transparente para processos operando em modo usuário. Dessa forma, o processo que teve seus acessos virtualizados não tem nenhuma pista que o permita saber de tal coisa, e ele opera normalmente como se estivesse trabalhando com os recursos reais.

Virtualização de Registro

Diferentemente da virtualização de sistema de arquivos, a virtualização de registro é implementada no *Configuration Manager*. Além disso, há uma tabela que determina quais chaves podem ser virtualizadas e quais não podem, já que uma virtualização no estilo da usada no sistema de arquivos causaria problemas de compatibilidade e interoperabilidade.

Chaves que são virtualizadas são redirecionadas para *HKCU\Software\Classes\VirtualStore*.

Programas que forem compatíveis com o *UAC* não participam de virtualização, ou seja, se eles não tiverem acesso eles receberão uma mensagem de acesso negado.

Cada chave do registro carrega consigo uma *flag*, que diz como ela deve se comportar no caso de uma virtualização. Os possíveis valores passa essa *flag* são:

- *REG_KEY_DONT_VIRTUALIZE*: Se estiver marcada, significa que essa chave não deve ser virtualizada

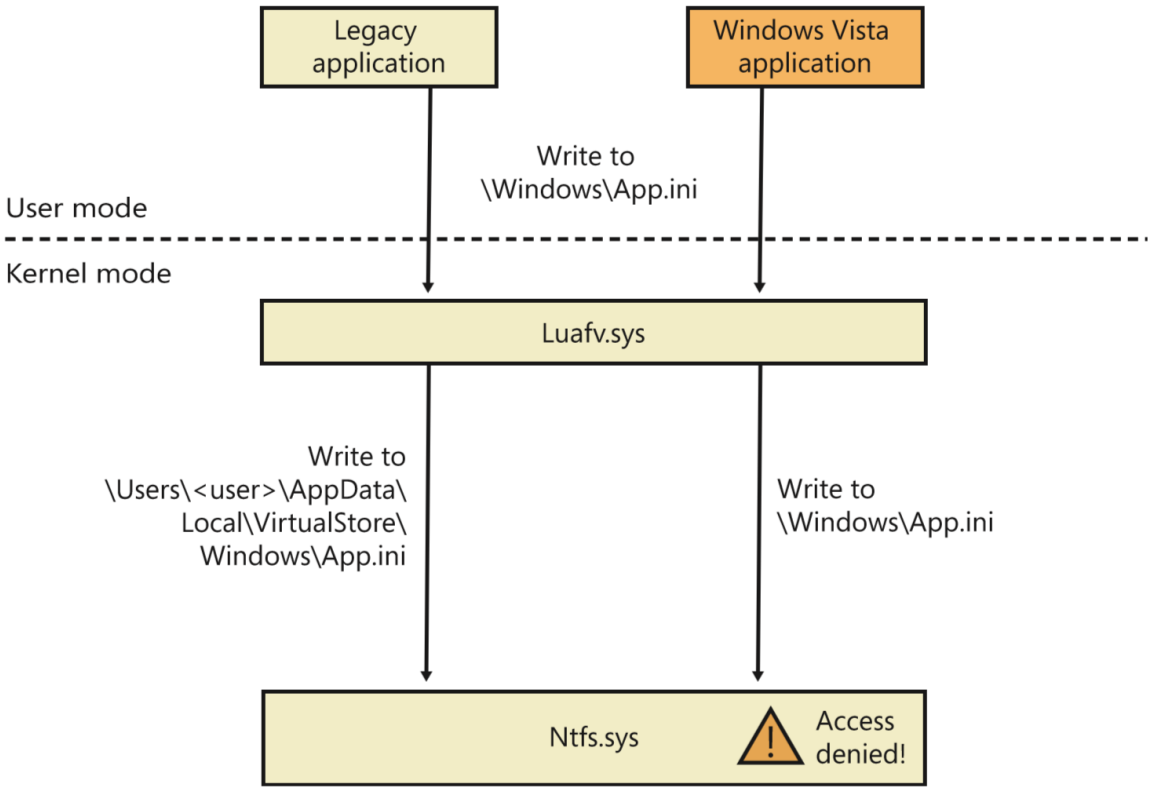


Figura 3.4: Ilustração gráfica do funcionamento da virtualização de sistema de arquivos do UAC

- *REG_KEY_DONT_SILENT_FAIL*: Se *REG_KEY_DONT_VIRTUALIZE* estiver marcada (e portanto a virtualização está desativada) essa *flag* especifica que um programa que normalmente seria negado acesso à chave na verdade recebe acesso máximo.

- *REG_KEY_RECURSE_FLAG*: Se estiver marcada, significa que as configurações de virtualização dessa chave devem se propagar para as sub-chaves.

3.10.2 Mecanismos de Elevação

Mesmo que todos os programas rodando pelo usuário sejam compatíveis com o *UAC*, eventualmente será necessário requisitar mais privilégios, já que algumas tarefas do sistema realmente precisam de privilégios elevados. Pode ser, por exemplo, que o objetivo do programa realmente seja alterar configurações de impacto global do sistema, e nessas horas é necessário que exista um mecanismo que permita tais alterações.

O mecanismo disponibilizado pelo Windows para resolver esse problema é a funcionalidade de "Executar como", que permite um usuário criar processos com credenciais de outros usuário. Dessa forma um usuário limitado (ou filtrado) pode iniciar um processo em nome de um usuário administrador, ou então em seu próprio nome mas sem os filtros. Como mencionado anteriormente, durante o Logon de usuários administradores o Windows na verdade cria duas identidades para esse usuário: Uma identidade de um usuário limitado (chamada de identidade filtrada) e uma identidade que possui todos os direitos administrativos. Por padrão, todos os programas rodam com a identidade do usuário filtrada, e apenas quando necessário executa-se com a identidade de administração.

Esse processo de "dar direitos administrativos a um processo" é chamado de elevação. Além disso, cada um dos tipos de elevação mencionados acima também recebem um nome: Uma elevação feita a partir de um usuário realmente restrito mediante apresentação de credenciais de um administrador é chamada de elevação *OTS (Over-the-shoulder)*. Uma elevação feita a partir de um usuário administrador mas que está executando com um *token* filtrado é chamada de consentimento.

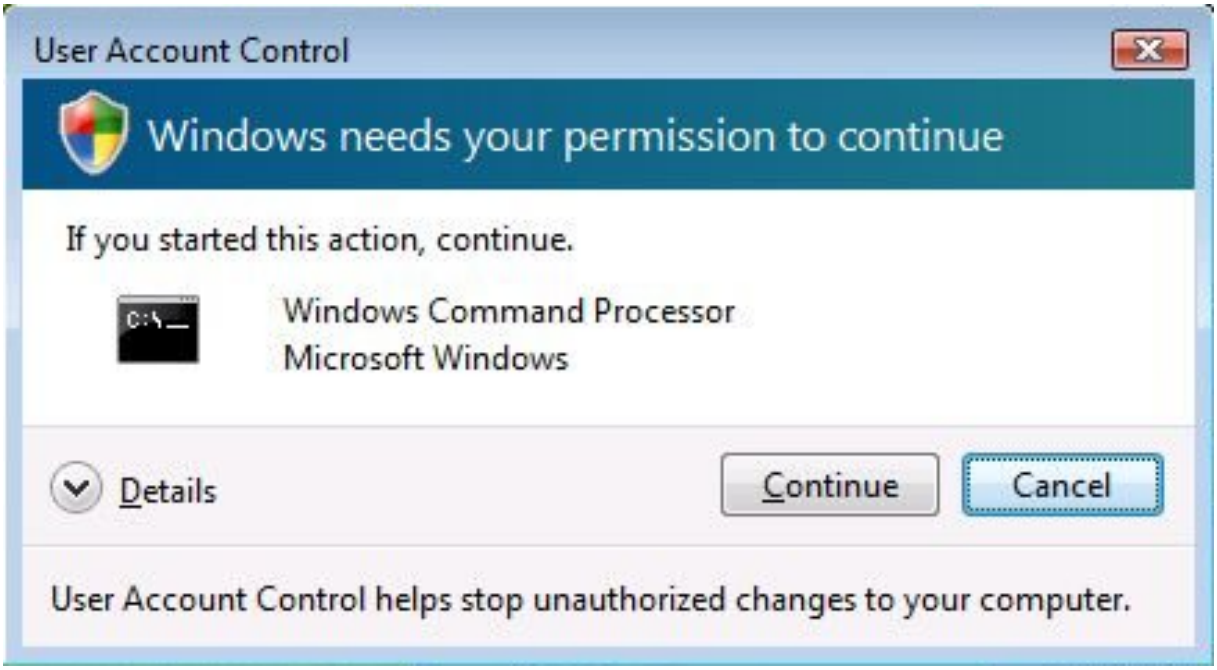


Figura 3.5: Exemplo de uma elevação de consentimento no Windows Vista

Se um usuário negar a elevação, então o sistema retorna um "Acesso negado" ao processo que pediu a elevação. Se o usuário permitir a elevação, então o sistema faz uma chamada de "Executar como" com as credenciais apropriadas que permitirão o processo rodar com direitos administrativos.

3.10.3 Auto-Elevação

Na configuração padrão do Windows, a maioria dos executáveis e itens do painel de controle não resultam em pedidos de elevação, mesmo que eles precisem de direitos administrativos. Isso porque eles são considerados processos passíveis de "auto-elevação".

Para um processo ser passível de auto-elevação ele deve cumprir vários requisitos, tais como ser um executável do Windows (ou seja, assinado digitalmente com o certificado do Windows) e deve estar em algum dos sub-diretórios considerados "seguros" (*%SystemRoot%\System32*) e seus sub-diretórios, ou *%ProgramFiles%*)

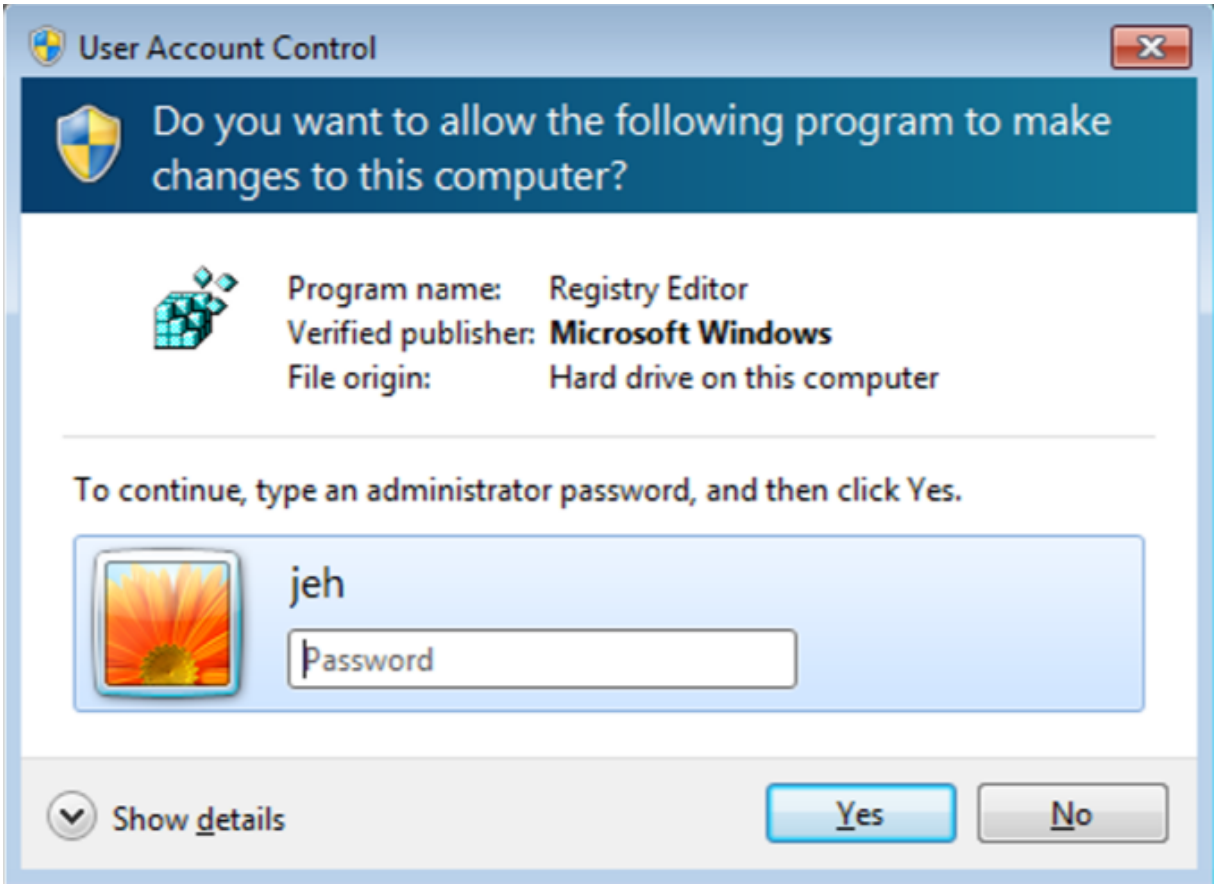


Figura 3.6: Exemplo de uma elevação *OTS* no Windows 7

3.11 *PatchGuard* - Proteção contra alterações no *Kernel*

Ao longo do tempo descobriu-se que vários componentes eram instalados no *Executive* com o objetivo de alterar o comportamento do sistema operacional de forma não suportada. Um exemplo seria alterar ou fazer *Hookings* na tabela de *System Calls*. Devido a isso foi implementado em todas as versões do Windows de 64 bits um mecanismo que evita alterações e *hooking* em imagens / estruturas do *kernel*. Abaixo segue uma tabela com os tipos mais comuns de *kernel patching* que são barrados pelo *PatchGuard*

Tabela 3.3: Componentes protegidos pelo PatchGuard

Componente	Função no SO	Potencial uso malicioso
Ntoskrnl.exe, Hal.dll, Ci.dll, Kdcom.dll, Pshed.dll, Clfs.sys, Ndis.sys, Tcpip.sys	*Kernel, HAL* e suas dependencias e parte baixo-nível do *stack* de rede	Alterações a esses componentes pode subverter a operação normal dos mesmos, nesse caso subvertendo a operação de todo o sistema.
Tabela de Descritores Globais *(GDT)*	Proteção de hardware que implementa em qual *Ring* do *CPU* se está executando	Ter acesso a essa *GDT* significa poder fazer um ataque tipo *callgate*, em que um processo rodando em modo usuário pode fazer operações com privilégios de modo *kernel*
Tabela de descritores de Interrupção *(IDT)*	Vetor das rotinas de interrupções	Programas maliciosos poderiam interceptar I/O no nível de interrupção, ou então usar isso para esconder partes da memória. *Rootkits* poderiam modificar a entrada INT2E do vetor e dessa forma interceptar todas as *Syscalls*.
Tabela de descritores de serviço de sistema *(SSDT)*	Tabela de *Syscalls* do Windows	*Rootkits* poderiam modificar essa tabela e ao invés de executar uma rotina do sistema executar uma rotina maliciosa
Registradores MSR	Esses registradores são os que guardam qual das entradas da tabela de *Syscalls* deve ser executada	Da mesma forma que anteriormente, um *Rootkit* com acesso a esse registrador pode interceptar / alterar todas as *Syscalls* do sistema

Apontadores das funções *KdpStub, KiDebugRoutine, KdpTrap*	São funções usadas pelos depuradores para captar exceções que acontecem em modo *kernel* e que só funcionam se o sistema estiver configurado para modo de depuração	Valores desses apontadores poderiam ser alterados por *rootkits* para tomar controle do sistema e fazer isso de forma completamente oculta
PsInvertedFunctionTable	Tabela para acesso rápido das exceções, ajuda a rastrear onde cada exceção ocorreu	Poderia ser usada para tomar o controle do sistema durante o tratamento de uma exceção qualquer.
Kernel Stacks	Armazena argumento de funções, o *call stack* e as variáveis de uma *thread*	Um *driver* malicioso poderia alocar memória, se marcar como o *kernel stack* dessa *thread* e depois manipular seu conteúdo para alterar chamadas e parâmetros de forma arbitrária
Definições de Tipos de Objetos	Definições para os vários tipos de objetos suportados pelo *Executive* e que são tratados pelo *Object Manager*	Poderia ser usada como parte de uma técnica chamada *DKOM (Direct Kernel Object Modification)* para modificar o comportamento do sistema.
Outros	Códigos relacionados às rotinas de *bugcheck, DPCs, PatchGuard,* etc	Se programas maliciosos conseguirem alterar pedaços do código que mantém o *PatchGuard*, eles podem efetivamente desativar ou enfraquecer seu funcionamento.

Sistemas de 64bits não podem ter o *PatchGuard* desligado, com exceção a máquinas que estejam rodando com um depurador de *Kernel* ligado a elas. Nesse caso o *PatchGuard* é automaticamente desligado.

3.12 Integridade de Código

O Windows possui um método centralizado e bem definido para autenticar a veracidade e integridade de arquivos e processos. Essa verificação é feita através de um certificado digital, que está embutido na imagem do arquivo. Entretando, por motivos de compatibilidade, apenas as versões de 64bits do Windows implementam o sistema de *Code Integrity* completo. Uma das políticas baseadas na verificação da integridade de código é a *KMCS (Kernel Mode Code Signing)*, que determina que todo código que roda no *Executive* deve, obrigatoriamente, ser assinado.

3.13 *BitLocker*

Todas as políticas definidas anteriormente só podem ser mantidas enquanto o sistema operacional estiver ativo e no controle da máquina[11]. Quando esse não é o caso (como por exemplo quando a máquina está desligada) outras medidas precisam ser tomadas para garantir a segurança e a integridade do sistema.

3.13.1 Descrição

O *Bitlocker* é uma tecnologia implementada a partir da versão 6.0 do NT, e permite criptografar volumes inteiros, incluindo o próprio sistema operacional. Existem dois modos de operação suportados pelo *Bitlocker*:

- Padrão: Criptografa unidades físicas do sistema em questão
- Portátil: Criptografa discos removíveis (como pendrives)

No modo padrão, o *BitLocker* impede acesso não autorizado ao sistema combinando dois procedimentos:

- Todo o volume é criptografado, o que inclui o próprio sistema operacional
- Verificando a integridade dos componentes de inicialização durante a inicialização, bem como seus parâmetros de configuração

Para que o *Bitlocker* funcione totalmente é necessário que o computador possua um chip TPM 1.2. Trata-se de um coprocessador que implementa várias ferramentas de criptografia, tais como criptografia de chave pública. É através desse mecanismo que o Windows sabe, durante a inicialização, que nada foi alterado. Os principais componentes que compõe o sistema do *BitLocker* são:

- *Driver TPM*, que roda no *Executive*, responsável por manipular o chip *TPM*.

- O Serviço de Base do *TPM*, que é um serviço de modo usuário que permite controlar a configuração do chip
- Uma entrada no Gerenciador de *Boot*, que autentica acesso ao disco e cuida do destravamento durante a inicialização
- O *Filter Driver* do *Bitlocker*, que faz criptografia e descriptografia *on-the-fly* de um volume
- O Provedor de gerenciamento do Windows do *Bitlocker*, que permite configurar o *Bitlocker*.

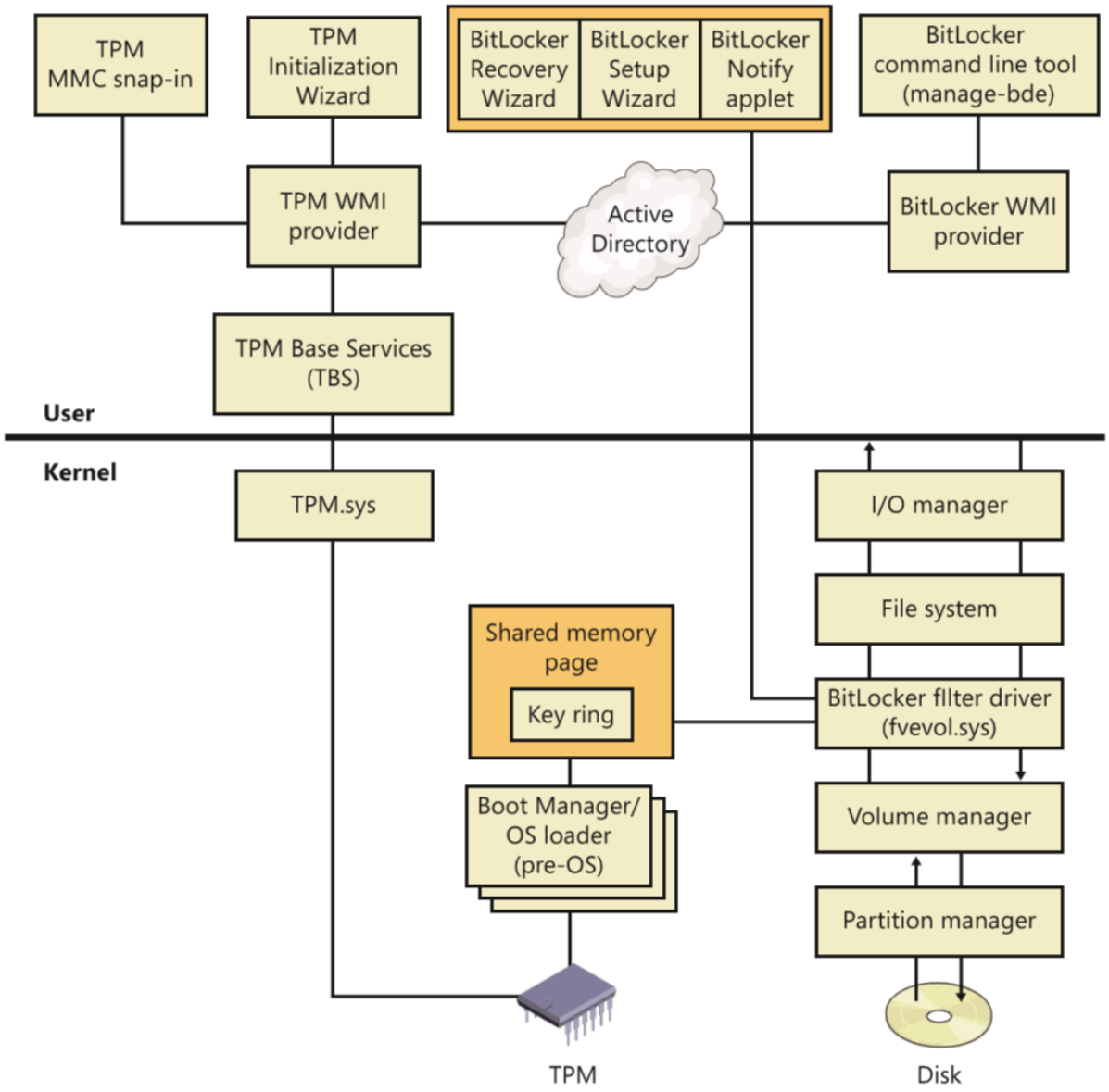


Figura 3.7: Arquitetura simplificada do *BitLocker*

O *BitLocker* criptografa o conteúdo de um volume usando uma chave simétrica *(FVEK)* com o algoritmo AES256-CBC. A *FVEK*, por outro lado, é criptografada usando uma Chave Mestre do Volume *(VMK)* e armazenada na região de controle desse volume. Quando o *BitLocker* é configurado e o sistema possui um chip *TPM*, pode-se criptografar a *VMK* usando o *TPM*. Mais especificamente existem quatro formas de fazer essa criptografia com o TPM:

1. Criptografar apenas com o *TPM*: O próprio *TPM* é a chave para destravar o sistema. Se alguém conseguir acesso ao chip *TPM*, contudo, pode ser que seja possível destravar o sistema de forma não autorizada.

2. Criptografar com o *TPM + PIN*: A chave usada para destravar o sistema é a combinação da chave do *TPM* mais um *PIN* digitado sempre que o sistema inicializa. É mais resistente que a proteção apenas por *TPM*.

3. Criptografar com o *TPM* + Chave *USB*: A chave usada para destravar o sistema é a combinação da chave *TPM* mais uma chave *USB* armazenada em um *Pendrive*. Esse *Pendrive* deve ser conectado sempre que o sistema inicializa. É muito mais complicada de ser quebrada, já que o acesso não autorizado depende de acesso ao *TPM* e ao *Pendrive* com a chave de destravamento.

4. Criptografar com *TPM + PIN* + Chave *USB*: A chave usada para destravar o sistema é a combinação da chave do *TPM*, de um *PIN* digitado e de uma chave armazenada em um *pendrive*. Esse é o nível de proteção mais alto oferecido pelo *BitLocker*.

Além da criptografia comum dada pelo algoritmo *AES* o *Bitlocker* também implementa um esquema de autenticação baseado em criptografia para garantir a integridade dos dados do volume. Embora o *AES* seja supostamente inquebrável atualmente, suponhamos que alguém consiga quebrá-lo e alterar algum arquivo. Pode ser que alterar esse arquivo mude o comportamento do sistema, destruindo todo o propósito da criptografia. O algoritmo de autenticação do *BitLocker* se chama *Elephant* e ele faz com que a modificação de um único bit do volume resulte em uma saída completamente aleatória quando descriptografada pelo algoritmo de *AES*. Ou seja, caso uma alteração seja bem sucedida o sistema se tornará completamente inútil.

3.13.2 Sequência de *Boot* do *BitLocker*

Durante a Inicialização do Sistema o *TPM* coleta e armazena várias características do ambiente pré-OS. Com base nesses dados ele calcula um *hash* que representa o estado da

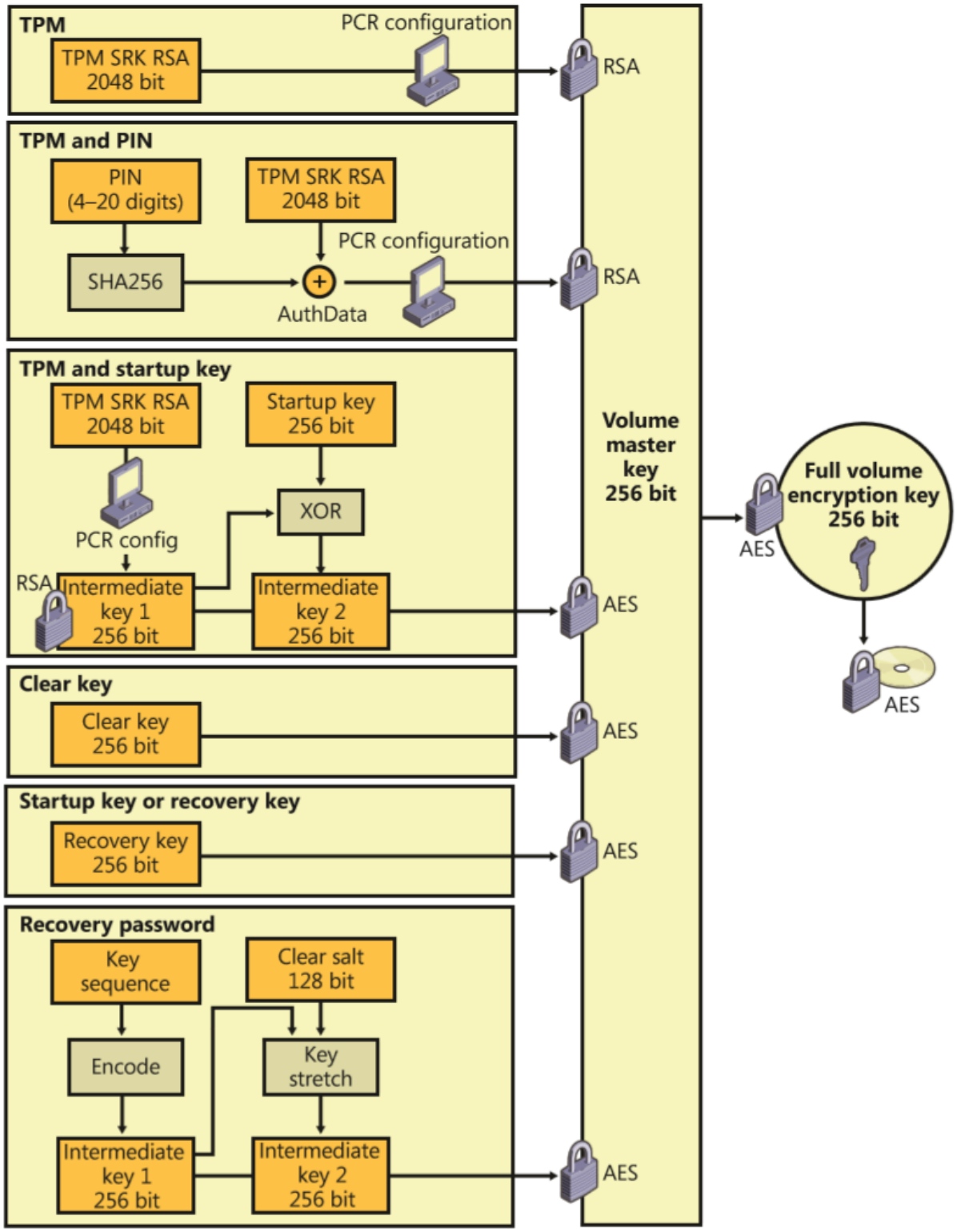


Figura 3.8: Ilustração de como o *BitLocker* gerencia suas chaves

máquina durante a inicialização. O *BitLocker* usa essa função de *hash* para selar a *VMK*. Se algum dos componentes pré-OS tiverem sido alterados o *hash* não será o correto e não é possível destravar o *BitLocker* do sistema. Caso o *hash* esteja correto o *BitLocker* é destravado e desse momento em diante o sistema operacional está ativo, e passa a ser dele a responsabilidade da segurança.

Mais especificamente, as tarefas executadas pelo sistema *TPM / BitLocker* na inicialização são:

1. *TPM* faz um auto-teste
2. *BIOS* faz auto-teste e calcula o *hash* do seu sistema de gravação em *TPM* e escreve esse *hash* no primeiro registro do *TPM*.
3. *BIOS* calcula um *hash* do seu código de configuração e também escreve esse *hash* no primeiro registro do *TPM* (levando em conta o *hash* anterior e o substituindo)
4. *BIOS* calcula *hash* do próximo elemento responsável pelo *boot* e também grava o resultado no *TPM*.
5. Para cada um dos componentes de *Boot* seguintes, o próprio componente é responsável por olhar o *hash* já presente no *TPM*, levá-lo em consideração, calcular seu próprio *hash* e rearmazená-lo no *TPM*.
6. No momento da inicialização do sistema operacional esse "*hash* acumulado" é passado como argumento para liberar a *VMK*, necessária para ler o volume em que o Sistema Operacional está presente.

Uma outra característica desse método é que ele não garante apenas que a máquina é idêntica àquela que gerou os dados de *hash*, mas sim que toda a instalação do sistema é única. Mesmo uma outra instalação idêntica do mesmo sistema operacional na mesma máquina não gera a mesma chave acumulada ao final do processo. [11]

Na prática, trata-se de um esquema de verificação em cadeia.

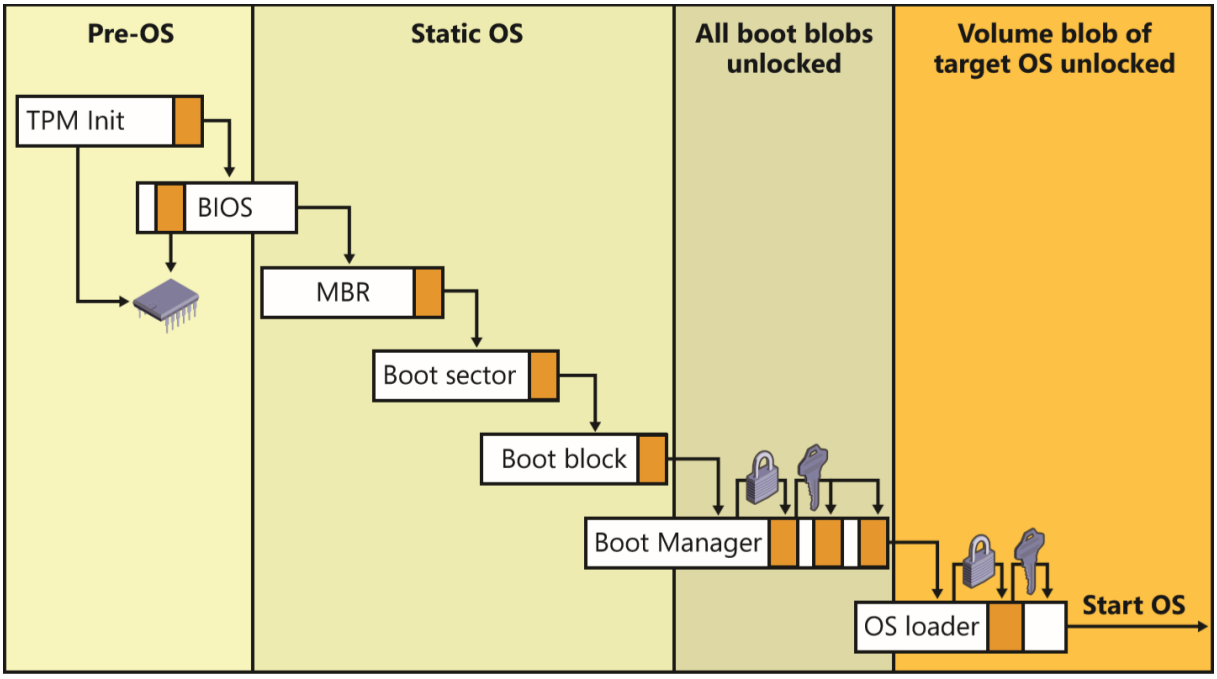


Figura 3.9: Ilustração do processo de verificação em cadeia no tempo de *Boot*

Capítulo 4

Análise Histórica e Discussão de Vulnerabilidades

O Windows NT é um sistema desenvolvido pensando-se em segurança. Como já elucidado ao longo desse trabalho, houve grande preocupação em se implementar inúmeros subsistemas com a finalidade de garantir que cada usuário ou componente deve ter apenas o acesso mínimo necessário ao seu funcionamento, dessa forma mitigando problemas de segurança. Ao longo do tempo contudo, falhas foram descobertas em vários serviços do Sistema Operacional. Algumas dessas falhas não podem ser completamente exploradas caso os sistemas de mitigação estejam funcionando, enquanto outras acabam driblando os sistemas de mitigação. Segue abaixo uma breve lista de algumas vulnerabilidades encontradas recentemente no Windows e nos seus serviços, acompanhadas de uma breve discussão do que elas causam e alguns fatores de mitigação. Todas as vulnerabilidades abaixo possuem CVSS[1] acima de 9 e estão ordenadas cronologicamente da mais recente para a mais antiga.

4.1 CVE-2013-3940

Descoberta em 12/11/2013, trata-se de uma vulnerabilidade no *GDI*, que é um componente instanciado no *Executive*. Essa vulnerabilidade afeta todas as versões do Windows, e, no pior caso, permite que um usuário remoto consiga acesso completo ao sistema através da criação de um arquivo de imagem especial, causando um *Buffer Overflow* no *GDI*. Esse *Buffer Overflow*, por sua vez, pode permitir ao usuário intruso executar código arbitrário com as mesmas permissões que o usuário logado.

No momento da escrita desse trabalho não há *exploits* catalogados que tirem proveito dessa vulnerabilidade. Embora potencialmente o efeito de um *Buffer Overflow* no *GDI* possa significar uma possível execução de código arbitrário, em um sistema Windows

habilitado com o nível padrão de segurança isso resultaria mais provavelmente em um *DoS*, já que *buffer overflow* em componente do *Executive* causaria um *Bugcheck* 0xC1, que causaria *crash* do sistema.

4.2 CVE-2013-3894

Descoberta em 09/10/2013, essa vulnerabilidade caso explorada corretamente permite que usuários intrusos consigam executar código malicioso de forma remota no contexto do *Kernel*. A vulnerabilidade é causada devido a um erro na forma em que o Windows trata arquivos de fonte *TrueType*. Mais especificamente, o Windows falha ao fazer o *parsing* da tabela *CMAP* desse arquivo especialmente projetado. Infectar-se por um programa malicioso que implemente essa vulnerabilidade pode ser tão simples quanto entrar em um site que use uma fonte com essa tabela *CMAP*. Enquanto o sistema estiver tentando renderizar a fonte ele acabará liberando o acesso ao usuário intruso.

4.3 CVE-2013-3195

Trata-se de uma vulnerabilidade no componente *Comctl32.dll* descoberta em 09/10/2013. O *Comctl32.dll* é responsável por implementar a *"Windows Common Control Library"*, que é responsável por definir um conjunto de *APIs* para interação com o usuário. Essa falha pode ser explorada caso um usuário intruso envie uma requisição para um aplicativo *ASP.NET* executando no sistema alvo. O erro propriamente dito é causado por um problema na forma em que a função *DSA_InsertItem* requisita memória ao sistema operacional. Caso completamente explorada, o resultado é um usuário malicioso com os mesmos direitos da conta do usuário que está executando o programa *ASP.NET*. Caso boas práticas de segurança sejam tomadas o comprometimento é apenas do *Pool* da aplicação em questão.

4.4 CVE-2013-3175

Descoberta em 14/08/2013 é uma vulnerabilidade que se explorada corretamente leva a uma elevação de privilégio não autorizada. A falha é causada pela forma em que o Windows lida com as chamadas de *RPC* assíncronas. Essa falha permite que um usuário execute um código no contexto de um outro usuário, e caso esse outro usuário tenha direitos administrativos o usuário intruso passa a poder fazer operações nesse novo contexto. A partir desse ponto ele pode alterar configurações de âmbito global do sistema.

Servidores de aplicações distribuídas públicas são os que mais estão em risco, já que nesse modelo não há nenhuma necessidade de autenticação prévia.

4.5 CVE-2013-3174

Descoberta em 09/07/2013, trata-se de uma uma vulnerabilidade no sistema do *DirectShow*, mais especificamente na porção de código que trata de imagens *GIF*. Essa vulnerabilidade se explorada corretamente permite a um usuário invasor executar código arbitrário na máquina alvo com os mesmos direitos do usuário / processo que abriu o arquivo *GIF*. De forma prática, contudo, essa vulnerabilidade não é fácil de ser explorada, pois depende do usuário da máquina executar propositalmente o arquivo *GIF* infectado. O *DirectShow* é um componente do Microsoft *DirectX*, que é uma biblioteca usada para manipulação de conteúdo digital de alta performance, tais como jogos, *video streaming, video sourcing*, renderização e afins. O *DirectShow*, nesse caso, é usado por inúmeros tocadores de mídia, pois é ele que permite executar tais arquivos usando aceleração de *hardware* no Windows.

4.6 CVE-2013-0011

Descoberta em 09/01/2013, trata-se de uma vulnerabilidade do serviço de *Spooler* de Impressão do Windows. Ela é causada quando o *Spooler* falha ao lidar com um documento de impressão feito especificamente para esse propósito. Caso essa falha seja explorada corretamente o usuário intruso pode executar código arbitrário no contexto da conta de usuário em que o *Spooler* de impressão executa. Na maioria das instalações, contudo, o *Spooler* de impressão roda com a conta *SYSTEM*. Nessas instalações o intruso conseguiria controle completo do sistema alvo. Servidores de impressão que não necessitam de autenticação são os mais vulneráveis. O Windows, contudo, por padrão exige autenticação para se ter acesso aos serviços de impressão.

4.7 CVE-2012-4774

Trata-se de uma vulnerabilidade na forma em que o Windows faz o *Parsing* dos nomes de arquivos. Descoberta em 11/12/2012, se explorada com sucesso permite que um usuário invasor execute código arbitrário na máquina alvo com as mesmas credenciais do usuário que foi atacado. Por outro lado, essa vulnerabilidade é difícil de ser executada, pois depende do usuário abrir um diretório que contenha um arquivo com um nome feito de

forma especial para causar essa falha. Uma forma seria, por exemplo, enviar um arquivo com esse nome anexado em um *e-mail*.

4.8 CVE-2012-1852

O *Windows NT LanMan Workstation Service (LanManWorkstation)* é o serviço do lado do cliente que provê a funcionalidade de rede para redes Microsoft. Em 14/08/2012 descobriu-se uma vulnerabilidade nesse serviço que permite a um usuário invasor executar código remoto no sistema alvo com as mesmas credenciais do serviço *Workstation*. Como o serviço *Workstation* executa com as credenciais *SYSTEM*, o usuário invasor consegue controle praticamente completo da máquina alvo. A falha é causada devido a um erro na forma em que o serviço *Workstation* trata pedidos *RAP (Remote Administration Protocol)*, e um pedido feito de uma forma especial permite a execução de código arbitrário. Essa falha é muito grave, pois não exige autenticação do usuário invasor e a forma mais eficaz de mitigação dela é o uso de um *firewall* que impeça requisições ao *LanManWorkstation*.

4.9 CVE-2012-0173

O *RDP (Remote Desktop Procotol)* é o protocolo usado pelas versões mais recentes do Windows para implementar o *shell* gráfico remoto *(Terminal Services)*. Através dele é possível fazer *login* remoto em uma máquina e operá-la como se o *login* fosse local. Em 12/06/2012 foi descoberta uma falha no serviço que provê o *Remote Desktop Protocol*. Caso um usuário malicioso envie uma sequencia especial de pacotes para um servidor com o serviço habilitado é possível que ele consiga executar código remoto no contexto do serviço que provê o *Terminal Services*. Essa vulnerabilidade afeta principalmente servidores já que versões cliente do Windows vem com o *Terminal Services* desativado por padrão. Para servidores, contudo, essa falha é muito grave, já que ela não exige autenticação e serviços terminais frequentemente estão expostos à internet.

4.10 CVE-2012-0001

Como já foi mencionado anteriormente, o Windows aplica um conceito de "Defesa em Profundidade". Um dos mecanismos implementados para garantir esse conceito é o *SafeSEH (Safe Structured Exception Handling)*. A idéia é criar uma tabela somente-leitura com os ponteiros para as rotinas de exceção. [6] Quando alguma dessas exceções acontecem em tempo de execução o Windows consulta essa tabela para verificar se o endereço está mesmo certo antes de passar o controle para o código que trata da exceção. Em 10/01/2012 foi

descoberta uma falha no módulo de *Kernel* que implementa o *SafeSEH* que permite a um programa malicioso escrever nessa tabela supostamente "somente-leitura". A vulnerabilidade em si não causa danos diretos ao sistema, contudo derruba essa barreira de proteção e abre possibilidade para explorar outras vulnerabilidades em serviços ativos no sistema alvo.

Capítulo 5

Conclusão

Ao longo desse trabalho foi feita uma apresentação sobre o Windows NT usando a versão 6.1 (Windows 7) como base. Como se pode ver, a família de sistemas operacionais NT da Microsoft é desenvolvida de forma extremamente modular, o que facilita a modificação de partes do sistema operacional. Adicionalmente também nota-se uma preocupação profunda com a segurança de todo o ambiente e pode-se perceber que inúmeras medidas são tomadas para tentar evitar e/ou mitigar o comprometimento do sistema em um caso de ataque de malware.

Ao mesmo tempo evidencia-se, contudo, que embora essas medidas e proteções evitem uma boa parte das pragas virtuais, algumas poucas conseguiram descobrir pequenas falhas em alguns dos módulos e se utilizaram de tais falhas para atividades maliciosas.

Por fim, pode-se dizer que a composição de um ambiente seguro não é só responsabilidade do Sistema Operacional, mas principalmente do administrador / usuário. Como se pode ver no capítulo anterior quase todas as vulnerabilidades dependiam de algum sistema de mitigação estar desativado, de serviços possivelmente supérfluos em execução ou então de alguma ação direta do usuário.

Referências Bibliográficas

[1] Common vulnerability scoring system. Available at http://www.first.org/cvss.

[2] Computer security. Available at http://en.wikipedia.org/wiki/Computer_security_industry.

[3] Market share statistics for internet technologies. Available at http://marketshare.hitslink.com/.

[4] Ms windows nt - the foundation. Available at http://technet.microsoft.com/en-us/library/cc767896.aspx.

[5] Reactos wiki. Available at http://www.reactos.org/wiki/ReactOS.

[6] Safe structured exception handling. Available at http://www.tortall.net/projects/yasm/manual/html/objfmt-win32-safeseh.html.

[7] Versions of windows obtain common criteria eal level 4+. Available at http://www.nist.org/news.php?extend.37.

[8] Windows nt server course. Available at http://scilnet.fortlewis.edu/tech/NT-Server/default.htm.

[9] GDCL. Windows nt device drivers. Available at http://www.gdcl.co.uk/winnt.htm.

[10] Alex Ionescu Mark Russinovich, David A. Solomon. *Windows Internals Part 1*. Microsoft Press, 6th edition, 2012.

[11] Alex Ionescu Mark Russinovich, David A. Solomon. *Windows Internals Part 2*. Microsoft Press, 6th edition, 2012.

[12] Daniel A. Menascé. Windows nt architecture. Available at http://cs.gmu.edu/ menasce/osbook/nt/.

[13] Bogdan Popa. Avast users rate windows 8 as the safest windows ever. Available at http://news.softpedia.com/news/Avast-Users-Rate-Windows-8-as-the-Safest-Windows-Ever-305134.shtml.

[14] Bogdan Popa. Softpedia exclusive interview: Jacqueline beauchere on microsoft's security efforts. Available at http://news.softpedia.com/news/Softpedia-Exclusive-Interview-Jacqueline-Beauchere-on-Microsoft-s- Security-Efforts-302483.shtml.

[15] Bogdan Popa. Windows 8 is the most secure doperating system on the market security expert. Available at http://news.softpedia.com/news/Windows-8-Is-the-Most-Secure-Operating-System-on-the-Market-Security-Expert-305566.shtml.

[16] Milind Borate Prasad Dabak, Sandeep Phadke. *Undocumented Windows NT*. Hungry Minds, 1999.

[17] Steven Roman. Windows architecture. Available at http://technet.microsoft.com/en-us/library/cc768129.aspx.

[18] Mark Russinovich. Windows nt security, part 1. Available at http://www.windowsecurity.com/whitepapers/windows_security/Windows_NT_Security_Part_1.html.

[19] Moriah Sargent. Despite windows 8 zero-day, vendors laud security of new microsoft os. Available at http://searchsecurity.techtarget.com/news/2240170624/Despite-Windows-8-zero-day-vendors-laud-security-of-new-Microsoft-OS.