UNIVERSIDADE DE BRASÍLIA

Faculdade do Gama

Tópicos Especiais em Engenharia de Software – Desenvolvimento de Software Seguro

Relatório de Projeto

Escalonamento de Privilégios no Kernel Linux

Gustavo Nogueira Rodrigues - 170144259

Pedro Helias - 140158278

Brasília, DF

2022

1. Introdução

1.1 Escalonamento de Privilégios

O escalonamento de privilégios é um tipo de ataque de rede usado para obter acesso não autorizado a sistemas dentro de um perímetro de segurança.

Para os invasores, a escalada de privilégios é um meio para um fim. Ele permite que eles obtenham acesso a um ambiente, persistam e aprofundem seu acesso e executem atividades maliciosas mais graves. Por exemplo, o invasor pode usar os privilégios recém obtidos para roubar dados confidenciais, executar comandos administrativos ou implantar *malware* – e potencialmente causar sérios danos ao seu sistema operacional, aplicativos de servidor, organização e reputação.

1.2 Linux

O Linux está instalado na maioria dos servidores demonstrando que é a melhor escolha com um mínimo de recursos. Até os rivais estão usando o Linux em suas ofertas. À medida que os aplicativos de software estão migrando para plataformas em nuvem, os servidores Windows estão sendo descontinuados para dar espaço aos servidores Linux.

Além disso, Linux está em toda parte. Do menor dispositivo ao maior supercomputador, o Linux está em toda parte. Pode ser um carro, roteador, telefone, dispositivos médicos, avião, TV, satélite, relógio ou tablet.

1.3 Objetivo

Dito isso, o presente trabalho tem como objetivo os seguintes os tópicos: realizar uma revisão a estrutura do kernel Linux; realizar uma revisão sobre os conceitos relacionados ao escalonamento de privilégios; realizar uma revisão sobre os principais vetores de ataque de escalonamento de privilégios em sistemas Linux; analisar a vulnerabilidade *Dirty Pipe* de escalonamento de privilégios do kernel Linux; elaborar uma prova de conceito que explora a vulnerabilidade *Dirty Pipe*.

2. O Kernel do Linux

Inicialmente, um kernel é um núcleo do Sistema Operacional que realiza a comunicação entre a parte gráfica, de interface, e o hardware, uma ponte de interligação entre os sistemas. Todo sistema operacional possui um Kernel. Além disso, o Kernel permite a intercomunicação e troca de mensagens entre esses sistemas.

O kernel possui alguns subsistemas como *Process Management(PM)*, responsável pelo controle e acesso dos processos ao processador. *Memory Management(MM)*, que controla o acesso à memória. Há o *Virtual File System*, que fornece uma interface para os dados armazenados no sistema. A *Network Layer*, que permite a conexão do dispositivo à rede. Por fim, o *IPC* que realiza a intercomunicação entre outros sistemas. Cada subsistema listado possui módulos que facilitam os procedimentos.

Especificamente, o Kernel possui algumas funções a partir de seus subsistemas, como:

Gerenciamento da memória: ele monitora o volume de memória utilizado para armazenar o que no sistema (arquivos, dados etc.) e onde geograficamente no sistema (ambiente).

Gerenciamento de processos: ele determina quais processos podem usar a unidade central de processamento (*CPU*), quando e por quanto tempo.

Drivers de dispositivos: ele atua como intermediário entre o hardware e os processos.

Chamadas do sistema e segurança: recebe solicitações dos processos para a execução de serviços.

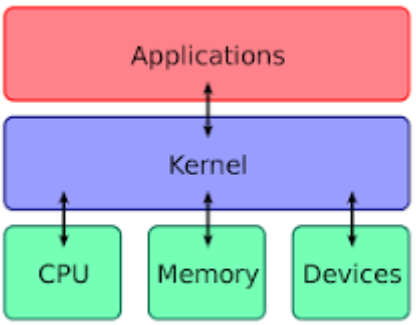


Figura 1 - Esquemático interligação Kernel. Fonte: AHMED, Alexis. [1]

Alguns tipos de Kernel:

- Monolítico: Não há separação clara em serviços na estrutura de um sistema, geralmente estão alocados em uma forma que facilite a distribuição de memória no sistema. Essa distribuição facilita o acesso ao hardware.
- Microkernel: Em oposição ao sistema Monolítico, as atividades do Microkernel são quase todas realizadas fora do núcleo, se tornando uma execução mais minimalista
- Há também o kernel híbrido.

Mais profundamente no Linux, seu funcionamento se dá semelhantemente a um sistema de 3 camadas. Na primeira camada temos o hardware, composto basicamente de memória *RAM*, *CPU*, dispositivos de entrada e saída (*I/O*), *GPU*, placa de rede, entre outros. Na segunda camada há o Kernel do Linux, o qual coordena a *CPU* conforme há interações e solicitações. Por fim, há os processos de usuário que são os serviços gerenciados e em execução pelo Kernel. Então, O Kernel vai fazer essa gerência entre as camadas.

O código é executado de duas formas: Modo usuário e modo Kernel. Quando executado em modo kernel, há acesso irrestrito ao hardware, ao contrário das restrições delimitadas pelo modo usuário. Neste ponto, há a formação da base da separação de privilégios, plenamente categorizada neste trabalho. Há também a possibilidade de criação de Containers e utilização de máquinas virtuais. O acesso como usuário Root ao sistema se vale da separação de privilégios existentes. Problemas gerados no modo Usuário, não são transferidos para todo o sistema e

pode ser remediado/corrigido. Porém, ações realizadas em modo Kernel atingem todo um sistema, podendo trazer consequências severas.

Quando o Linux está em execução, há uma divisão da memória *RAM* da máquina: Espaço do kernel - onde o kernel é executado; e o espaço do usuário, onde todos os processos dos usuários são executados. Um processo pode ser executado tanto na área do usuário quanto no kernel. Quando executado na área do usuário, possui privilégios normais, não tendo autorização para realizar algumas ações. Em contrapartida, quando um processo é executado no kernel, ele possui privilégios irrestritos.

3. Escalonamento de Privilégios

3.1 O que é o escalonamento de privilégios ?

De acordo com AHMED, o escalonamento de privilégios é o processo de exploração de vulnerabilidades ou configurações errôneas em sistemas para elevar privilégios de um usuário para outro, normalmente para um usuário com acesso administrativo ou *root* em um sistema. A escalada de privilégios bem-sucedida permite aos atacantes aumentar seu controle sobre um sistema ou grupo de sistemas que pertencem a um domínio, dando-lhes a capacidade de fazer mudanças administrativas, roubo de dados, modificar ou danificar o sistema operacional e manter o acesso através da persistência.

3.2 Como os privilégios e as permissões são designadas ?

Antes de abordar as técnicas de escalonamento de privilégios, precisamos entender primeiro os sistemas operacionais projetados para lidar com as contas de usuário e privilégios.

Para AHMED, a segregação de funções é o principal fator por trás das várias filosofias de implementação de contas de usuários que são implementadas nos sistemas operacionais atualmente. Esta abstração das funções e permissões dos usuários em um sistema é configurada e facilitada por um sistema chamado anel de proteção(*ring protection*), como ilustrado na Figura 1. Isto especifica os limites e

reforça a funcionalidade dos usuários em um sistema e seu respectivo acesso aos recursos.

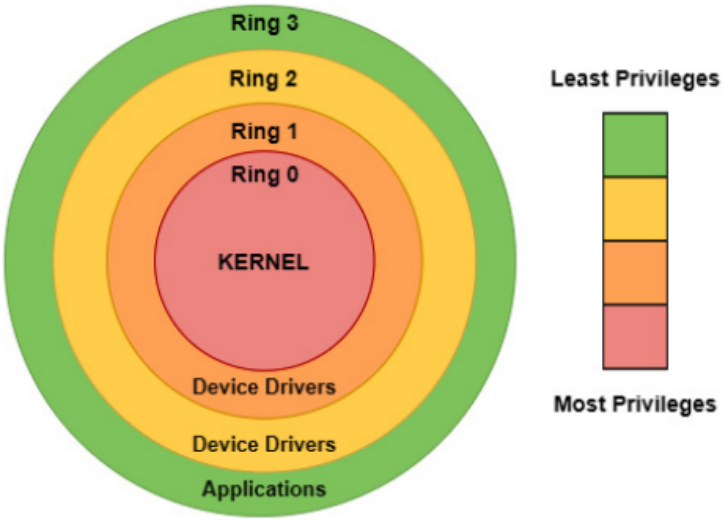


Figura 2 - Anel de Proteção. Fonte: AHMED, Alexis. [1]

Conforme apresentado na figura 1, existe uma segregação dos anéis desde os menos privilegiados até os mais privilegiados. Esta segregação de privilégios em um sistema leva à adoção de dois papéis principais, como segue:

- Acesso privilegiado: Isto é tipicamente representado ou atribuído à conta *root* ou administrador e fornece acesso completo a todos os comandos e recursos do sistema. Dentre as funcionalidades que podem ser utilizadas, destacamos as seguintes: a capacidade de instalar, desinstalar e modificar o software do sistema ou binários; capacidade de adicionar, modificar ou remover usuários e grupos de usuários; capacidade de acessar e ter controle sobre todo o hardware do sistema.
- Acesso sem privilégios: Isto é normalmente representado ou atribuído a contas não-*root* ou contas de usuário padrão e está limitada a um conjunto específico de privilégios que são projetados e adaptados para o acesso padrão do usuário em um sistema. Ele limita a funcionalidade do usuário a tarefas básicas e acesso aos dados do usuário no sistema.

Esta segregação de permissões destaca a importância do escalonamento de privilégios para *penetration testers* ou atacantes, pois oferece controle total sobre

um sistema ou, potencialmente, sobre um grupo de sistemas se eles puderem obter *root* ou acesso administrativo em um sistema.

Dito isso, dada a natureza dos ataques de escalada de privilégios em relação às contas e permissões dos usuários, há dois métodos principais de realizar escalada de privilégios que podem ser utilizados pelos atacantes com base em suas intenções e objetivos, sendo eles: escalonamento horizontal de privilégios; escalonamento vertical de privilégios.

3.3 Escalonamento horizontal de privilégios

Segundo AHMED, a escalada horizontal de privilégios é o processo de acesso à funcionalidade ou dados de contas de usuários comuns em um sistema, ao contrário de obter acesso a contas com privilégios administrativos ou *root*. Trata-se principalmente de acessar ou autorizar a funcionalidade em um sistema usando contas que estão no mesmo nível de permissões de usuário, ao contrário das contas de usuário que estão mais acima e que têm mais privilégios e permissões.

O propósito desse tipo de ataque de escalada de privilégios é obter acesso a dados de contas de usuários sem privilégios, bem como roubar credenciais de contas de usuários ou hashes de senhas.

3.4 Escalonamento vertical de privilégios

De acordo com AHMED, a escalada vertical de privilégios é o processo de exploração de uma vulnerabilidade em um sistema operacional para obter acesso *root* ou administrativo em um sistema. Este método é geralmente preferido por atacantes e *penetration testers*, pois dadas as permissões e funcionalidades, agora é possível ter total acesso e controle sobre o(s) sistema(s).

3.5 Vetores de ataque em sistemas Linux

3.5.1 Explorando o Kernel

As explorações do kernel são programas ou binários que afetam tanto o Windows quanto Linux e são projetados para explorar vulnerabilidades no kernel subjacente, para executar código arbitrário com permissões elevadas ou *root.*

Para AHMED, o processo de exploração é multifacetado e requer uma boa quantidade de enumeração a fim de determinar a versão do sistema operacional e *patches* ou *hotfixes* instalados e, consequentemente, se é afetado por algum *kernel exploit*, após os quais o código de *kernel exploit* pode ser recuperado através de vários repositórios de exploração, tais como *exploit-db[3]* e *traitor[4]*. O código de exploração deve então ser inspecionado e personalizado com base nos parâmetros necessários e funcionalidade. Após a personalização, o código pode ser compilado em um binário e transferido para o alvo para a execução. Em alguns casos, o código de exploração precisará ser baixado e compilado no alvo, se depender de certas dependências.

Por fim, após a compilação e execução bem sucedida do binário, o *kernel exploit* concederá ao atacante acesso *root* ao sistema alvo sob a forma de um *shell prompt*, onde eles podem executar comandos no sistema com privilégios de *root*.

3.5.2 Explorando binários SUID

SUID(Set User ID) é um recurso embutido do Linux que permite aos usuários executar binários e arquivos com as permissões de outros usuários.

Segundo AHMED, este recurso é comumente usado para permitir que contas não-*root* executem utilitários e binários de sistema com permissões de *root*. Você pode definir a permissão *SUID* do programa ou utilitário com o proprietário como *root*. Dessa forma, isto permitirá que o programa ou utilitário seja executado com privilégios de root sempre que um usuário não-*root* o executar. Os atacantes podem explorar ou tirar vantagem das configurações erradas de *SUID* e executar comandos arbitrários como *root*. Por exemplo, programas ou binários que permitem a execução de comandos arbitrários como o *vim* não devem ter seu dono *SUID*

definido como *root*, pois usuários não-*root* podem aproveitar a funcionalidade de execução de comandos dentro do *vim* para executar comandos com *root*.

3.5.3 Explorando serviços vulneráveis e permissões

Os serviços oferecem a maior superfície de ameaça para os atacantes, dada a variabilidade e diversidade de programas e serviços que podem ser encontrados rodando em sistemas Linux.

De acordo com AHMED, os atacantes normalmente terão como objetivo identificar serviços e programas mal configurados ou vulneráveis que possam facilitar a escalada dos privilégios. Por exemplo, em sistemas Linux, os atacantes tentarão identificar e explorar configurações errôneas com *cron jobs* e aproveitar a funcionalidade para executar código arbitrário ou binários maliciosos.

3.5.4 Explorando o SUDO

Neste cenário, os atacantes geralmente têm como alvo os usuários que têm privilégios *SUDO*. O *SUDO* permite aos usuários executar comandos como outro usuário, normalmente o usuário *root*.

Segundo AHMED, os privilégios *SUDO* são normalmente configurados manualmente pelos administradores, o que deixa a porta aberta para possíveis erros de configuração. Assim, por exemplo, um administrador pode atribuir permissões *SUDO* a um usuário não-*root* para certos utilitários de linha de comando (tais como *find* ou *vim*) que podem executar comandos *shell* ou código arbitrário.

Dessa forma, isto pode ser alavancado por atacantes para executar código arbitrário ou executar comandos com privilégios de *root*.

3.5.5 Credenciais inseguras

AHMED cita que esta técnica envolve a busca de credenciais inseguras que tenham sido armazenadas em um sistema pelos usuários ou pela realização de um processo de rachadura de credenciais fracas de usuários. Muitos usuários, e até mesmo administradores de sistema, anotam senhas em texto puro em documentos, planilhas e arquivos de configuração para várias contas de serviços. Estes arquivos

podem ser localizados executando consultas especializadas de busca com vários utilitários de linha de comando.

Assim, um exemplo disso é o uso do utilitário de linha de comando *find* no Linux para localizar arquivos com extensões e nomes de arquivos específicos.

4. Vulnerabilidade Dirty Pipe (CVE-2022-0847)

4.1 O que é ?

Divulgada em 7 de março de 2022 por Max Kellerman[2], a CVE-2022-0847 é uma vulnerabilidade no kernel Linux (versões entre a 5.8 e 5.16.11) que permite que invasores substituam arquivos somente leitura ou imutáveis e aumentem seus privilégios no sistema da vítima. A CVE-2022-0847 foi apelidada de Dirty Pipe e possui uma pontuação CVSS de 7,8 (considerada alta).

4.2 Como foi descoberta ?

A vulnerabilidade foi encontrada pelo pesquisador de segurança Max Kellerman. Segundo o pesquisador, tudo começou a partir de uma solicitação de suporte relacionada à corrupção de arquivos de *log* feita por um cliente.

O pesquisador conta que o arquivo de *log* estava de fato corrompido e que o mesmo podia ser descompactado, porém o *gzip* havia relatado um erro de *CRC(Cyclic Redundancy Check)*. Em um primeiro momento, o pesquisador não entendeu o motivo da corrupção do arquivo e decidiu apenas corrigir o *CRC* do arquivo manualmente para fechar a solicitação de suporte do problema.

Alguns meses depois, o pesquisador constatou que esse problema aconteceu mais algumas vezes. Segundo ele, toda vez que o conteúdo do arquivo parecia correto, apenas o *CRC* no final do arquivo estava errado. Agora, com a posse de vários arquivos corrompidos, o pesquisador conseguiu procurar mais fundo e encontrou um tipo inusitado de corrupção de memória.

4.3 Pipes, Páginas e Buffers no Kernel Linux

Antes de abordar a exploração da vulnerabilidade, serão apresentados conceitos importantes, nos parágrafos a seguir, que serão utilizados na seção de exploração da vulnerabilidade.

Um ***pipe*** nada mais é que uma ferramenta para comunicação unidirecional entre processos. Uma extremidade é para enviar dados para ele, a outra extremidade pode puxar esses dados. O kernel do Linux implementa isso utilizando um *array/buffer* circular de *struct pipe_buffer* [6][7], cada um referindo-se a uma página. Em outras palavras, *pipe* é um *buffer* de dados na memória de um sistema Linux que pode ser usado como se fosse um arquivo.

Uma **página** é a menor unidade de memória gerenciada pela *CPU(Central Processing Unit)* e consiste em um bloco de dados que geralmente possui o tamanho de de 4096 bytes(4 kB). O kernel do Linux divide os dados em páginas e opera em páginas em vez de lidar com o arquivo inteiro de uma só vez.

Segundo KELLERMANN, toda E/S(entrada/saída) de arquivo é sobre páginas, assim, ao ler dados de um arquivo, o kernel primeiro copia um número de pedaços de 4 kB do disco rígido para a memória do kernel, gerenciado por um subsistema chamado **cache de página**. E então, a partir daí, os dados serão copiados para o espaço do usuário. A cópia no cache da página permanece por algum tempo, onde pode ser usada novamente, assim evitando E/S desnecessárias no disco rígido, até que o kernel decida que tem um melhor uso para aquela memória.

Dito isso, existem algumas *syscalls*(chamadas de sistema) que permitem mapear as páginas gerenciadas pelo cache de página diretamente no espaço do usuário, ou seja, sem a necessidade de copiar os dados do arquivo para a memória do espaço do usuário. O kernel implementa isso passando **referências de página**, assim não copiando nada. Como exemplo desse tipo de chamada, temos as syscalls *mmap*, *sendfile* e *splice.*

Para o contexto da vulnerabilidade, destacamos a *syscall splice.* Esta chamada move dados entre dois descritores de arquivo(onde um dos descritores de

arquivo deve referir-se a um *pipe*) sem copiar entre o espaço de endereço do kernel e o espaço de endereço do usuário. Dessa forma, em vez de copiar uma página toda vez, ela fornece uma referência à página que deve ser transferida para o *pipe*.

4.4 Como funciona a exploração da vulnerabilidade ?

A vulnerabilidade Dirty Pipe é descrita como uma falha na maneira como o atributo *flags* da *struct pipe_buffer[7]* **não continha inicialização adequada** nas funções *copy_page_to_iter_pipe* e *push_pipe* no kernel do Linux e, portanto, poderia conter valores obsoletos.

Uma das flags que pode ser configurada no atributo flags da *struct pipe_buffer* é a flag chamada *PIPE_BUF_FLAG_CAN_MERGE.* Ela indica se a mesclagem de mais dados no *pipe_buffer* é permitida ou não. Dessa forma, quando os dados são copiados para um *pipe_buffer*, mais dados **podem ser adicionados** ao *pipe_buffer* se a página copiada tiver menos de 4096 bytes de tamanho.

Ao injetar *PIPE_BUF_FLAG_CAN_MERGE* em uma referência de cache de página, tornou-se possível **sobrescrever dados no cache de página**, simplesmente escrevendo novos dados no *pipe* preparado de maneira especial(este processo será descrito nos parágrafos a seguir). Dessa forma, um usuário local sem privilégios pode usar essa falha para sobreescrever/gravar em páginas no cache de página apoiadas por arquivos somente leitura e, assim, aumentar seus privilégios no sistema.

Dito isso, para explorar a vulnerabilidade, as seguintes etapas devem ser seguidas:

1. Crie um *pipe*.
2. Preencha o *pipe* com dados arbitrários (para definir a *flag PIPE_BUF_FLAG_CAN_MERGE* em todas as entradas do *array* circular).
3. Drenar o *pipe* (deixando a flag *PIPE_BUF_FLAG_CAN_MERGE* definida em todas as instâncias de *struct pipe_buffer* do *array* circular). Normalmente, a *flag* deve ser redefinida. No entanto, a vulnerabilidade Dirty Pipe faz com que a *flag* permaneça definida como 1.

4. Transfira um arquivo somente leitura para o *pipe* usando a *syscall splice*.
5. Modifique o arquivo somente leitura. Como a chamada do sistema *splice* usa o método *pass-by-reference* (passagem por referência), o invasor pode sobrescrever o arquivo devido a *flag PIPE_BUF_FLAG_CAN_MERGE*.

4.5 Mitigações

A vulnerabilidade foi corrigida no Linux 5.16.11, 5.15.25 e 5.10.102, portanto, para solucionar o problema, deve ser necessário obter essas versões ou uma posterior [8].

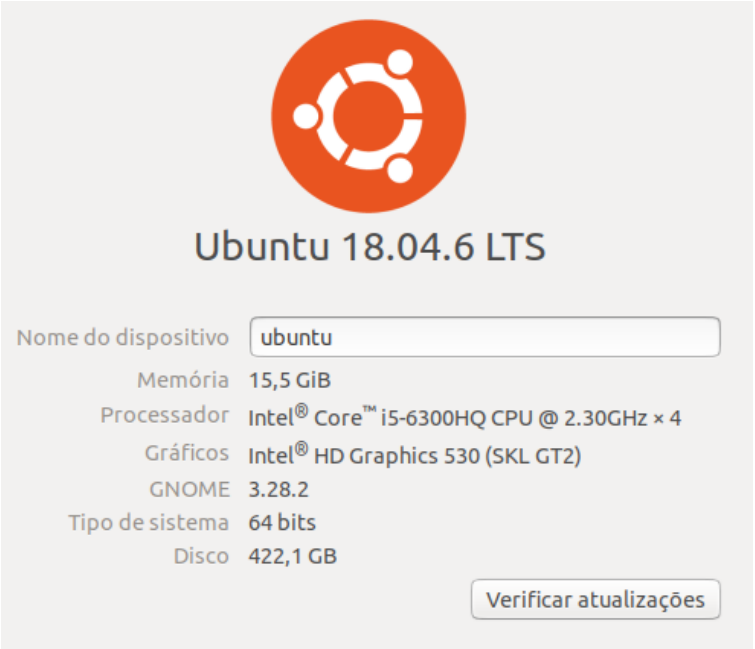
Segundo ARNTZ, para plataformas Android, foi constatado que, nas versões 5.x, a vulnerabilidade só será encontrada nos modelos mais recentes. Dessa forma, os usuários do Android com versões 5.x devem verificar se estão vulneráveis e, em caso afirmativo, estar atentos a uma atualização a ser lançada para corrigir essa vulnerabilidade. Por fim, a vulnerabilidade não afeta as versões 4.x, que representam a maioria dos dispositivos do Google e de outros fornecedores.

4.6 Construção da Prova de Conceito (POC)

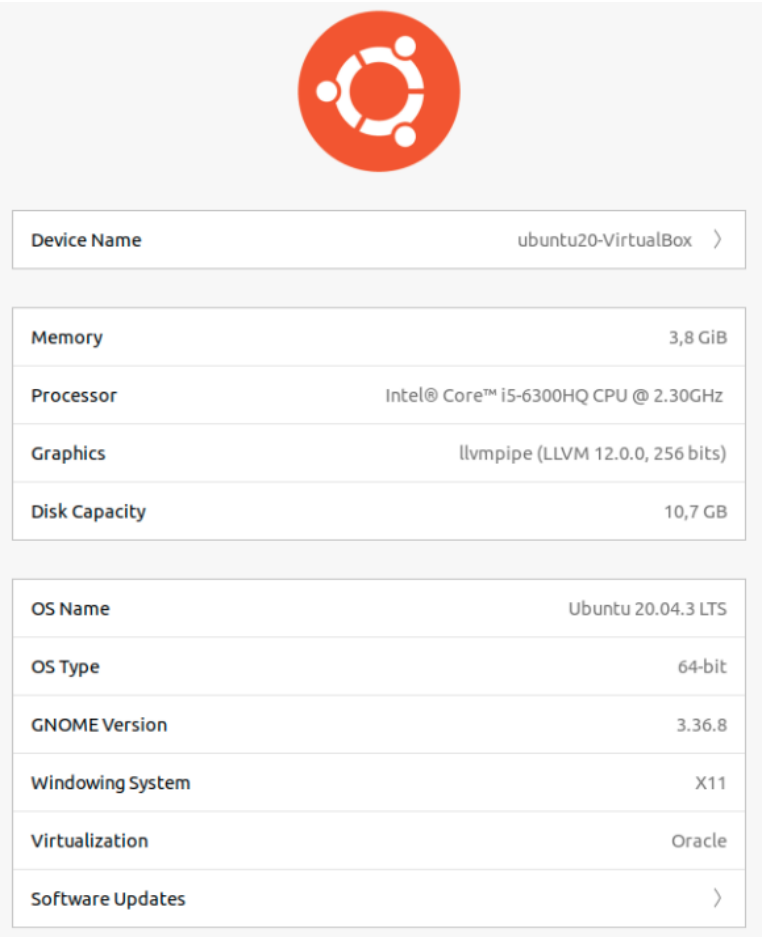
Para construção de uma prova de conceito que explora a vulnerabilidade foi utilizado o VirtualBox para criar uma Máquina Virtual. Além disso, foi utilizado como base o código de exploração fornecido pelo KELLERMANN em seu artigo "*The Dirty Pipe Vulnerability*" [2][10]. Os *scripts* da *POC* podem ser acessados a partir do seguinte repositório: https://github.com/Gustavo-Nogueira/Dirty-Pipe-Exploits.

4.6.1 Configuração do ambiente

O sistema operacional host é um Ubuntu 18.04.6 LTS e possui as seguintes configurações:



O sistema operacional convidado é um Ubuntu 20.04.3 LTS (versão kernel Linux 5.11.0-27-generic) e possui as seguintes configurações:



4.6.2 Exploit 1

O *exploit* 1, presente no arquivo *exploit-1.c* do repositório, consiste em sobrescrever a senha do usuário *root* presente no arquivo */etc/passwd* (este arquivo contém informações sobre a conta de usuário, como a senha criptografada) e por fim, fornecer um terminal com acesso *root*. Para tanto, foi necessário:

1. Realizar um *backup* do arquivo */etc/passwd.*
2. Preparar o *pipe* (conforme apresentado nas etapas 1, 2 e 3 da seção 4.4) com a flag *PIPE_BUF_FLAG_CAN_MERGE.*
3. Transferir o arquivo */etc/passwd* para o *pipe* e sobrescrever o conteúdo deste arquivo com uma nova senha de root conhecida (conforme apresentado nas etapas 4 e 5 da seção 4.4).
4. Criar uma sessão como *root* a partir da nova senha definida.
5. Em seguida, a senha é redefinida para o valor original a partir do *backup,* assim sem deixar rastros.
6. Por fim, ainda com a sessão *root* estabelecida, um terminal *shell* com acesso *root* é liberado.

4.6.3 Exploit 2

O *exploit* 2, presente no arquivo *exploit-2.c* do repositório, consiste em sobrescrever a sequestrar um binário *SUID*(esse tipo de ataque foi relatado na seção 3.5.2) e por fim, fornecer um terminal com acesso *root*. Para tanto, foi necessário:

1. Preparar um código executável *ELF(Executable and Linking Format)* que será utilizado para sobrescrever o binário *SUID* recebido a ser recebido como parâmetro. No *exploit*, o executável *ELF* utilizado cria um novo um arquivo executável(*/tmp/sh*), que por sua vez, cria uma terminal *shell* com acesso *root.*
2. Receber o *path* de um binário *SUID* como parâmetro.
3. Realizar um *backup* do binário recebido.

4. Preparar o *pipe* (conforme apresentado nas etapas 1, 2 e 3 da seção 4.4) com a flag *PIPE_BUF_FLAG_CAN_MERGE.*
5. Transferir o arquivo o binário recebido para o *pipe* e sobrescrever o conteúdo deste arquivo com o executável *ELF* da comentado na etapa 1 (conforme apresentado nas etapas 4 e 5 da seção 4.4).
6. Em seguida, o binário recebido é restaurado para o seu conteúdo original a partir do *backup,* assim sem deixar rastros.
7. Por fim, o arquivo executável */tmp/sh* citado na etapa 1 é executado, assim abrindo um novo *shell* com acesso *root*.

4.6.4 Exploit 3

O *exploit* 3, presente no arquivo *exploit-3.c* do repositório, utiliza as mesmas etapas do *exploit 1* para obter o acesso a *shell root*. A diferença é que exploit 3, após obter acesso ao *shell root*, carrega uma chave pública *SSH(Secure Shell)* do invasor e cria um túnel *TCP(Transmission Control Protocol)* para acesso remoto ao *SSH* utilizando a plataforma Ngrok.

5. Conclusão

O escalonamento de privilégios é o processo de exploração de vulnerabilidades ou configurações errôneas em sistemas para elevar privilégios de um usuário para outro, normalmente para um usuário com acesso administrativo ou *root* em um sistema. Dessa maneira, a escalada de privilégios torna-se uma etapa essencial para alguns tipos de invasão. Sendo assim, para os invasores, a escalada de privilégios é um meio para um fim. Tal fim pode ser desde roubo de dados confidenciais até a criação de uma chave *SSH* para permitir acesso remoto à máquina atacada.

O kernel Linux, apesar de ser reconhecido pela sua segurança e integridade, também não está livre de apresentar problemas de segurança. A CVE-2022-0847 divulgada em 7 de março de 2022 pelo pesquisador Max Kellerman, revelou uma vulnerabilidade que afeta as versões entre 5.8 e 5.16.11 do kernel Linux. A CVE-2022-0847, apelidada de *Dirty Pipe*, permite que invasores substituam

arquivos somente leitura ou imutáveis e aumentem seus privilégios no sistema da vítima.

A partir deste trabalho foi possível realizar uma revisão sobre o funcionamento do kernel Linux e os seus e principais vetores de ataque de escalonamento de privilégios. Por fim, foi possível realizar uma análise da vulnerabilidade *Dirty Pipe*, bem como elaborar uma prova de conceito para explorar a vulnerabilidade.

6. Referências

[1] AHMED, Alexis. Privilege Escalation Techniques: Learn the art of exploiting Windows and Linux systems. Birmingham: Packt Publishing, outubro de 2021.

[2] KELLERMANN, Max. The Dirty Pipe Vulnerability. Disponível em: <https://dirtypipe.cm4all.com/>. Acesso em: 23 de julho de 2022.

[3] Exploit Database. Disponível em: <https://www.exploit-db.com/>. Acesso em: 03 de agosto de 2022.

[4] Traitor: Automatically exploit low-hanging fruit to pop a root shell. Disponível em: <https://github.com/liamg/traitor>. Acesso em: 03 de agosto de 2022.

[5] PayloadsAllTheThings. Linux - Privilege Escalation. Disponível em: <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Linux%20-%20Privilege%20Escalation.md#cve-2022-0847-dirty pipe>. Acesso em: 03 de agosto de 2022.

[6] Repositório do Kernel do Linux - Struct pipe_inode_info. Disponível em: <https://github.com/torvalds/linux/blob/v5.8/include/linux/pipe_fs_i.h#L76>. Acesso em: 03 de agosto de 2022.

[7] Repositório do Kernel do Linux - Struct pipe_buffer. Disponível em: <https://github.com/torvalds/linux/blob/v5.8/include/linux/pipe_fs_i.h#L26-L32>. Acesso em: 03 de agosto de 2022.

[8] ARNTZ, Pieter. Linux "Dirty Pipe" vulnerability gives unprivileged users root access. Malwarebytes Lab. Disponível em:

<https://www.malwarebytes.com/blog/news/2022/03/linux-dirty-pipe-vulnerability-gives-unprivileged-users-root-access>. Acesso em: 03 de agosto de 2022.

[9] Picus Security. Linux “Dirty Pipe” CVE-2022-0847 Vulnerability Exploitation Explained. Disponível em: <https://www.picussecurity.com/resource/linux-dirty-pipe-cve-2022-0847-vulnerability-exploitation-explained>. Acesso em: 03 de agosto de 2022.

[10] Exploit Database - Local Privilege Escalation (Dirty Pipe). Disponível em: <https://www.exploit-db.com/exploits/50808>. Acesso em: 03 de agosto de 2022.