Universidade do Minho
Escola de Engenharia

Tecnologias de Segurança

Trabalho Prático 3

Deteção de Modificações Não-Autorizadas no Sistema Operativo

Grupo 4

Bruno Filipe de Sousa Dias PG47068
Guilherme da Silva Amorim Martins PG47225
Luís Enes Sousa A89597



11 de Junho de 2022

Conteúdo

1 Introdução

Neste projeto é pretendido desenvolver e implementar uma componente de software capaz de detetar modificações não-autorizadas num conjunto de ficheiros considerados críticos para a segurança de um dado sistema operativo Linux (ex: configurações de sistema, de serviços, programas executáveis, etc).

Assim foi acrescentado um novo sistema de ficheiros baseado em **libfuse**, sendo adicionado essencialmente um sistema F2A na abertura de ficheiros. Assim, é adicionada a funcionalidade de autenticação de 2 fatores do Utilizador quando o mesmo pretende efetuar a abertura de um ficheiro. Assim, iremos neste relatório explicar todo o processo para a construção desta funcionalidade.

2 Proposta de Resolução

2.1 Abordagem

Sabemos neste momento o nosso objetivo, no entanto, não são impostas quaisquer regras quanto à implementação desta funcionalidade. Entre as escolhas fornecidas, a equipa optou pelo envio de mensagens como forma de envio do código de autenticação. Desta forma, sempre que o Utilizador pretender aceder a um ficheiro, este terá de ser alvo de uma autenticação de 2 fatores, inserindo primeiro a sua password e seguidamente o código de autenticação (recebido numa mensagem SMS). Desta forma, a verificação de acesso a um ficheiro pode ser efetuada. No caso de sucesso, o Utilizador poderá aceder então ao ficheiro e no caso de falha, o Utilizador obtém um erro de *Permission Denied*, não tendo assim acesso ao mesmo.

2.2 Arquitetura

Como podemos perceber, a implementação desta funcionalidade passa por vários passos. Deste modo, nem tudo teria a mesma eficiência e simplicidade se fosse escrito na linguagem C. Desta forma, a equipa utilizou como recurso 3 linguagens de modo a implementar esta funcionalidade: **python**, **bash** e **C**.

No ficheiro **python** possuímos todas as componentes necessárias para a geração do código PIN de autenticação aleatório, bem como o seu envio em forma de SMS para o Utilizador. Este PIN gerado aleatoriamente é essencial para uma autenticação mais segura e protegida de ataques. Nos ficheiros **bash** possuímos pequenos scripts que são responsáveis pela captura da password e do PIN que o Utilizador introduz na autenticação de abertura de um ficheiro e os envia ao nodo principal (através de FIFOs). Finalmente, no ficheiro **C**, que é o central e principal do sistema, possuímos toda a informação responsável por implementar a nova funcionalidade de autenticação de 2 fatores. Este ficheiro é baseado essencialmente no ficheiro *passthrough.c* do *libfuse*, possuindo informação extra relativamente ao processo de autenticação e de abertura de um ficheiro. Apenas utilizando estes 3 conjuntamente conseguimos implementar este processo de autenticação.

É ainda importante relembrar que para além destes ficheiros, esta funcionalidade utilizada ainda como recurso um ficheiro que é guardado em disco com apenas permissões de leitura e escrita do dono, onde estão guardadas informações sobre o username, password e número de telefone utilizado para receção das mensagens SMS com código de autenticação (informação de cada Utilizador em cada linha do ficheiro). Tal como este ficheiro recorremos ainda a PIPEs com nome, mais especificamente, FIFOs. Foram ainda construídas condições de segurança face a estes ficheiros explicadas no tópico de Segurança mais à frente.

2.3 Funcionamento

Estando explicada a arquitetura, passamos agora a explicar como é o funcionamento desta nova funcionalidade. No arranque deste sistema, classificamos todos e quaisquer Utilizadores como não autorizados. Isto deve-se ao facto de estes ainda não possuírem qualquer informação guardada no Sistema. Deste modo não será possível efetuar a abertura de qualquer ficheiro, dado o Sistema não possuir quaisquer informações para efetuar a autenticação de 2 fatores de um certo Utilizador. Para além destas informação, sabemos também que este processo fica a correr no background e é esperada a sua invocação através da abertura de ficheiros.

Assim, podemos perceber que o primeiro passo após a inicialização deste sistema de ficheiros, é fazer o cadastro e introdução de informações do Utilizador (vários Utilizadores distintos podem efetuar o seu cadastro, não apenas um). Após efetuarem o seu cadastro, estas informações serão guardadas num ficheiro seguro e com os devidos mecanismos de segurança, que irá ser mais tarde utilizado na autenticação de um Utilizador para a abertura de um ficheiro.

Estando feito o cadastro do Utilizador, este poderá começar a efetuar pedidos de abertura de ficheiros. A tentativa de abertura de um ficheiro irá invocar a função *open()* e todo o sistema irá entrar em ação. Inicialmente criamos dois pipes anónimos, utilizados para a troca de informação entre processos de informações. Estas informações dizem respeito à password e ao PIN de autenticação (cada tipo de informação tem o seu pipe). Posteriormente efetuamos a *system call fork*. O processo filho originado deste fork será responsável por executar o código python construído, e enviar o PIN construído, bem como a password do Utilizador para o processo pai, através dos *pipes* criados anteriormente. Não esquecer que, para efetuar tais operações, será necessário aceder ao ficheiro que possui as informações relativas ao diferentes Utilizadores.

Já no processo pai, este irá esperar que a execução dos processos filhos acabe, de forma ao processo ser mais eficiente e irá ler os pipes que possui, de forma a ir buscar as informações relativas tanto à password, bem como ao PIN gerado aleatoriamente e enviado sob a forma de SMS para o Utilizador. Neste momento o processo pai irá mais uma vez reccorer *system call fork* e nos processos filho irá efetuar os processos de pedir *input* ao Utilizador da password, bem como do PIN recebido. Este pedido de input é feito através da geração de uma nova janela *bash* com recurso ao *xterm*. As informações introduzidas serão depois enviadas para os pipes com nome, ou seja, o FIFOs que serão posteriormente lidos pelo pai.

Mais uma vez, o processo pai irá esperar pelo término da realização dos processos filho e seguidamente irá buscar aos FIFOs (lendo dos mesmos) as informações relativas tanto à password, bem como ao PIN. Finalmente, podemos efetuar o processo de autenticação de 2 fatores, onde o processo pai irá comparar tanto o PIN gerado com o PIN introduzido, bem como a password inserida no cadastro com aquela introduzida no processo de verificação. Se ambas as informações se revelarem iguais às que efetivamente deveriam ser, o Utilizador possui finalmente acesso ao ficheiro e pode efetuar a sua abertura sem qualquer problema. Se por ventura, algumas das duas informações introduzidas não se verificar verdadeira, então o Utilizador não terá acesso ao ficheiro recebendo um *Acess Denied*.

2.4 Segurança

Percebemos como é o funcionamento deste processo, e inclusive apontamos que efetuamos alguns métodos de segurança neste sistema. Assim, iremos neste momento explicar quais as preocupações e métodos de segurança implementados ao longo de toda a implementação desta funcionalidade e sistema.

Como sabemos um dos dois processos de autenticação será a geração de um código PIN aleatório. Na geração deste PIN teríamos de possuir a segurança necessária para o sistema, mas ao mesmo tempo, não possuir um PIN excessivamente grande, uma vez que seria cansativo a sua introdução na abertura dos ficheiros. Assim, a equipa decidiu que a melhor opção seria a geração de um PIN com 5 dígitos, que irá formar um total de 10^5 combinações, o que torna o ambiente suficientemente seguro nas condições propostas.

Sabemos também que, o outro processo de autenticação é a introdução de uma password. Esta password é, tal como referido anteriormente, introduzida no cadastro do utilizador e guardada num ficheiro ao qual apenas o dono possui acesso. Tanto o username, como o número telefónico, apesar de sensíveis, não são alvo de grande importância comparadas à password, uma vez que, à partida, podemos saber estas informações de outras formas (como por exemplo, conhecer o Utilizador na vida real). A password no entanto, e apesar de o ficheiro apenas poder ser lido pelo seu *owner*, não deverá estar guardada explicitamente neste ficheiro. Desta forma, o que o nosso grupo decidiu, foi efetuar o seu hash, através de um salt, e guardar essa informação no ficheiro. Assim, teríamos no lugar da password, o tipo de codificação usada, o *salt* usado, bem como a *hash* da *password* introduzida. Assim, mesmo que face a um ataque, a informação relativamente à password continuaria oculta e encriptada.

Para além deste mecanismo de segurança, é importante também realçar que tanto os FIFOs, bem como este ficheiro com informações sobre os Utilizadores, apenas iria ter permissões de leitura e escrita para o seu *owner*. Nenhum outro user terá sequer a permissão de leitura destes ficheiros, tornando assim o sistema mais seguro e livre de ataques. Para além disso, estes dois ficheiros irão possuir o *sticky bit* ativo na sua criação. Deste modo, temos a certeza que nenhum utilizador que não o seu dono possa alterar o conteúdo destes ficheiros, e dessa forma, o funcionamento do sistema nunca será quebrado.

Como acabamos de perceber, estes ficheiros serão impenetráveis, e por isso, se qualquer outro Utilizador que não o seu *owner* quisesse correr este programa, não poderia efetuá-lo. Assim, adicionamos neste sistema, mais propriamente no seu executável o bit de *setuid* (como poderemos ver na Makefile), de modo a qualquer Utilizador conseguir correr este sistema. Desta forma, um Utilizador que não possuiria permissões para aceder aos ficheiros mencionados anteriormente, irá neste momento (por causa do bit de *setuid*) possuir as permissões do *owner* temporariamente e irá dessa forma ser contornado o problema de outros Utilizadores conseguirem correr este sistema.

Finalmente, um outro mecanismo de segurança aplicado, foi a introdução de *waits* por parte dos processos pai, de forma a todo o processo ser efetuado da forma mais eficiente possível, melhorando assim operações como a leitura e escrita em pipes e fazendo com que os processos terminassem no estado de Zombie, roubando recursos desnecessários ao Sistema.

3 Conclusão

Finalizamos assim o desenvolvimento e explicação do projeto no âmbito da Unidade Curricular de **Tecnologias de Segurança**. Sentimos que foi um projeto bastante relevante, uma vez que nos forneceu a oportunidade de aprender ainda mais e obter mais conhecimentos sobre a área de Tecnologias de Segurança, principalmente na área relativa à Gestão de Ficheiros.

Apesar de não ser um trabalho muito extenso, uma vez que a quantidade de código acrescentado e alterado não ser muito vasta, o grupo acredita que foi um trabalho que nos conseguiu pôr á prova, uma vez que a maior parte do tempo foi utilizado a "puxar pela cabeça". Ainda assim, a equipa revela-se contente com o produto final e acredita ter alcançado todos os objetivos propostos, bem como ter acrescentado alguns pormenores que se revelam interessantes e importantes no que toca à Segurança do Sistema. Para além disso, é sempre agradável transformar algo já existente, numa versão ainda melhor.