Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC0640 - Sistemas Operacionais 1

Relatório - Implementação do Backdoor em C para Linux

Docente: Professor: Vanderlei Bonato

Cauê Pereira Cermak - 8936864
Yudi Asano Ramos - 12873553

Jun
2023

Conteúdo

1 Introdução Geral: Implementação de Backdoors para Monitoramento de Teclado

1.1 Introdução

Neste artigo, discutiremos a implementação de três códigos: 'backdoor.c', 'client.py' e 'server.py'. Esses códigos têm o objetivo de demonstrar um exemplo de implementação de um backdoor para monitoramento de teclado em um sistema Linux. Um backdoor é um tipo de software malicioso que permite o acesso não autorizado a um sistema ou dispositivo, abrindo uma porta dos fundos para ações indevidas.

O código 'backdoor.c' é escrito em linguagem C e é projetado para ser executado no kernel Linux. Ele é responsável por capturar eventos do teclado e armazená-los em um buffer de log. O backdoor é implementado usando recursos do kernel Linux, como manipulação de interrupções e criação de sistema de arquivos de depuração.

O código 'client.py' é escrito em Python e representa o cliente que se conecta ao backdoor para receber as atualizações do log. O cliente estabelece uma conexão TCP/IP com o servidor, recebe os dados do log e os exibe no terminal.

O código 'server.py' também é escrito em Python e representa o servidor que envia as atualizações do log para o cliente. O servidor cria um socket TCP/IP, aguarda a conexão do cliente e envia as atualizações do log assim que são detectadas.

O objetivo dessa implementação é fornecer um exemplo didático de como um backdoor pode ser desenvolvido e utilizado para monitorar as ações do usuário. É importante ressaltar que a utilização de backdoors sem o consentimento dos usuários é considerada uma prática antiética e ilegal. O objetivo deste artigo é apenas educacional, com o intuito de apresentar as técnicas envolvidas na implementação de um backdoor e promover uma melhor compreensão sobre segurança de sistemas.

No decorrer do artigo, abordaremos detalhadamente cada um dos códigos, explicando suas funcionalidades, estrutura e lógica de funcionamento. Além disso, discutiremos a importância da ética e da segurança da informação no contexto de backdoors e possíveis medidas de proteção contra essas ameaças.

1.2 Base de Código

Para a implementação dos códigos apresentados neste artigo, utilizaremos como base o código disponível no seguinte repositório do GitHub: `https://github.com/Yudiaramos/Backdoor_Service_and_client_py`. O

repositório contém a versão completa dos códigos ‘backdoor.c‘, ‘client.py‘ e ‘server.py‘, além de outros recursos relacionados.

O código fornecido no repositório servirá como ponto de partida para a explicação detalhada de cada um dos códigos e suas funcionalidades. Faremos referências específicas a trechos relevantes do código ao longo do artigo, permitindo uma compreensão mais clara e precisa da implementação do backdoor e dos componentes cliente e servidor.

É recomendado que os leitores acessem o repositório e baixem os códigos para acompanharem a explicação de forma prática e interativa. A disponibilidade do código-fonte completo facilitará a análise e a compreensão dos conceitos discutidos ao longo do artigo.

Agora, vamos nos aprofundar na explicação de cada um dos códigos, começando pelo ‘backdoor.c‘, que é o componente principal responsável pela captura dos eventos do teclado e armazenamento em um buffer de log.

2 Parte 1: Implementação do Backdoor em C para Linux

2.1 Introdução

Neste artigo, discutiremos a implementação do código `backdoor.c`, que consiste em um backdoor projetado para o kernel Linux. Abordaremos as funcionalidades, estrutura e a lógica por trás do código. O objetivo deste backdoor é capturar eventos do teclado e armazená-los em um buffer de log, permitindo um possível monitoramento das ações do usuário.

2.2 Importação de Bibliotecas

O código começa importando as bibliotecas necessárias para o funcionamento do backdoor. As bibliotecas *linux/module.h,linux/kernel.h,linux/init.h, linux/keyboard.h, linux/irq.h, linux/interrupt.h, linux/fs.h e linux/debugfs.h fornecem as funcionalidades essenciais para a captura de eventos do teclado, manipulação de interrupções e criação do sistema de arquivos de depuração (`debugfs`).*

2.3 Definição do Backdoor

Em seguida, o código define as informações do backdoor, como a licença, autor, versão e descrição do módulo. Essas informações são importantes para identificar o backdoor e garantir sua compatibilidade e conformidade.

2.4 Estrutura de Dados

O backdoor utiliza uma estrutura de dados chamada `dentry` para representar um diretório no sistema de arquivos de depuração. O código declara duas variáveis do tipo `struct dentry`: `dir` para representar o diretório principal do backdoor e `file` para representar o arquivo de log onde os eventos do teclado serão armazenados.

2.5 Funções de Log e Leitura

A função **write_to_log()** é responsável por adicionar mensagens ao buffer de log. Essa função recebe uma string como parâmetro e a concatena ao buffer de log **log_buffer**. A função **log_read()** é uma função de leitura do arquivo de log que é chamada quando o arquivo é lido. Essa função retorna os dados do log para quem lê o arquivo.

2.6 Manipulação dos Eventos do Teclado

O código implementa duas funções principais para manipular os eventos do teclado. A função `keyboard_interrupt_handler()` é um manipulador de interrupção que é acionado quando um evento do teclado é capturado. Nessa função, o código converte o código do evento em um caractere e o armazena no buffer de log. A função `keyboard_notifier_callback()` é um callback que é acionado quando um evento do teclado é registrado. Essa função chama o manipulador de interrupção para processar o evento.

2.7 Registro do Backdoor

O código registra o backdoor no sistema. Ele cria o diretório principal e o arquivo de log dentro do sistema de arquivos de depuração. Em seguida, registra o callback `keyboard_notifier_callback()` para capturar os eventos do teclado.

2.8 Funções de Inicialização e Encerramento

O código define as funções de inicialização e encerramento do backdoor. A função `keyboard_module_init()` é chamada durante a inicialização e é responsável por criar o diretório e o arquivo de log, além de registrar o callback. A função `keyboard_module_exit()` é chamada durante o encerramento e é responsável por remover o diretório, o arquivo de log e desregistrar o callback.

2.9 Conclusão Backdoor - Parte 1

O código `backdoor.c` apresenta a implementação de um backdoor no kernel Linux, capturando eventos do teclado e armazenando-os em um buffer de log. Essa implementação permite monitorar as ações do usuário. É importante ressaltar que a utilização de backdoors sem o consentimento dos usuários é considerada uma prática antiética e ilegal. Este artigo tem o objetivo de apresentar o funcionamento e as técnicas envolvidas na implementação de um backdoor para fins educativos e de pesquisa.

3 Parte 2: Implementação de um Servidor em Python para Envio de Logs de um Backdoor

3.1 Introdução

Neste artigo, exploraremos o código `server.py`, que consiste na implementação de um servidor em Python responsável por receber os logs gerados por um backdoor e enviar esses logs para os clientes conectados. Discutiremos a lógica por trás do código e sua interação com o backdoor e os clientes.

3.2 Configuração de Conexão

O código começa definindo as configurações de conexão do servidor. A variável `HOST` armazena o endereço IP ou nome do host do servidor. A variável `PORT` define a porta do servidor para aguardar conexões dos clientes.

3.3 Criação do Socket

Em seguida, o código cria um objeto de socket TCP/IP utilizando a biblioteca `socket` do Python. A função `socket.socket()` é utilizada para criar o socket com os devidos parâmetros.

3.4 Definição das Opções do Socket

O servidor define uma opção de socket chamada `SO_REUSEADDR` utilizando a função `s.setsockopt()`. Essa opção permite reutilizar o endereço local, permitindo que o servidor reinicie rapidamente em caso de encerramento repentino.

3.5 Associação do Socket ao Endereço e Porta

O servidor associa o socket ao endereço e porta definidos utilizando a função `s.bind()`. Essa etapa é necessária para que o servidor fique vinculado a um endereço e porta específicos e possa aguardar por conexões dos clientes.

3.6 Colocando o Socket em Modo de Escuta

Após a associação do socket, o servidor coloca o socket em modo de escuta por meio da função `s.listen()`. Isso permite que o servidor aguarde as conexões dos clientes.

3.7 Aceitando Conexões dos Clientes

O servidor aguarda a conexão de um cliente utilizando a função `s.accept()`. Essa função bloqueia a execução do servidor até que um cliente se conecte. Uma vez que uma conexão é estabelecida, o servidor obtém o objeto de conexão e o endereço do cliente.

3.8 Função de Envio dos Logs

O servidor possui uma função chamada `send_log_updates()`, responsável por enviar as atualizações do log para o cliente conectado. Essa função é executada em um loop contínuo. Ela lê o arquivo de log gerado pelo backdoor e verifica se há novas atualizações. Caso haja novas atualizações, envia os dados para o cliente.

3.9 Encerramento da Conexão

Após o envio das atualizações do log, o servidor encerra a conexão com o cliente utilizando a função `conn.close()`. Isso libera os recursos e finaliza a comunicação com o cliente.

3.10 Conclusão Server - Parte 2

O código `server.py` implementa um servidor em Python responsável por receber os logs gerados por um backdoor

4 Parte 3: Desenvolvimento de um Cliente em Python para Receber Logs de um Backdoor

4.1 Introdução

Neste artigo, abordaremos o código `client.py`, que é responsável por implementar um cliente em Python para se conectar a um servidor e receber os logs gerados por um backdoor. Discutiremos a lógica por trás do código e sua interação com o servidor.

4.2 Configuração de Conexão

O código começa definindo as configurações de conexão do cliente. A variável `HOST` armazena o endereço IP ou nome do host do servidor ao qual o cliente se conectará. A variável `PORT` define a porta do servidor para estabelecer a conexão.

4.3 Criação do Socket

Em seguida, o código cria um objeto de socket TCP/IP utilizando a biblioteca `socket` do Python. A função `socket.socket()` é utilizada para criar o socket com os devidos parâmetros.

4.4 Conexão ao Servidor

O cliente estabelece a conexão com o servidor por meio da função `s.connect()`, passando o endereço IP e a porta como argumentos. Essa etapa é crucial para estabelecer uma comunicação entre o cliente e o servidor.

4.5 Recebimento dos Logs

Após a conexão ser estabelecida, o cliente inicia um loop para receber os dados do servidor. A função `s.recv()` é utilizada para receber os dados em blocos de tamanho definido (neste caso, 1024 bytes). O loop continua até que não haja mais dados a serem recebidos.

4.6 Exibição dos Logs

Os logs recebidos do servidor são exibidos no terminal do cliente por meio da função `print()`. A função `data.decode()` é utilizada para converter os

dados recebidos, que estão em formato de bytes, para uma representação de string legível.

4.7 Encerramento da Conexão

Após receber todos os logs do servidor, o cliente encerra a conexão utilizando a função `s.close()`. Isso garante a liberação dos recursos e encerra a comunicação com o servidor.

4.8 Conclusão Client - Parte 3

O código `client.py` implementa um cliente em Python que se conecta a um servidor e recebe os logs gerados por um backdoor. Esse cliente permite monitorar e visualizar os logs em tempo real. É importante ressaltar que a utilização de backdoors sem o consentimento dos usuários é considerada uma prática antiética e ilegal. Este artigo tem o objetivo de apresentar o funcionamento e as técnicas envolvidas no desenvolvimento de um cliente para receber logs de um backdoor, visando fins educativos e de pesquisa.

5 Sintetizando: Desenvolvimento de um Backdoor com Envio de Logs para Clientes Remotos

5.1 Introdução

Neste artigo, exploramos a união dos códigos `backdoor.c`, `server.py` e `client.py` para desenvolver um sistema completo de backdoor com envio de logs para clientes remotos. Discutimos a funcionalidade de cada código e como eles se interconectam para criar um ambiente de monitoramento e coleta de informações.

5.2 1. Desenvolvimento do Backdoor

O código `backdoor.c` consiste no núcleo do backdoor. Ele é desenvolvido como um módulo de kernel para Linux e possui funcionalidades para capturar pressionamentos de teclas e armazenar essas informações em um buffer de log. O backdoor utiliza a interface `debugfs` para criar um diretório e um arquivo de log acessíveis pelo sistema de arquivos do kernel.

5.3 2. Implementação do Servidor

O código `server.py` é responsável por implementar o servidor que se comunica com o backdoor e envia os logs para os clientes remotos. Ele cria um socket TCP/IP e aguarda conexões dos clientes. Uma vez que um cliente se conecta, o servidor lê as atualizações do log do backdoor e envia esses dados para o cliente.

5.4 3. Desenvolvimento do Cliente

O código `client.py` é responsável por implementar o cliente que se conecta ao servidor e recebe os logs enviados pelo backdoor. O cliente estabelece uma conexão com o servidor por meio de um socket TCP/IP e recebe os logs em tempo real. Os logs recebidos são exibidos no terminal do cliente.

5.5 4. Funcionamento Integrado

Ao unir esses três códigos, criamos um sistema completo de backdoor com envio de logs para clientes remotos. O backdoor é implantado no sistema alvo e captura os pressionamentos de teclas, armazenando-os em um buffer de log.

O servidor se comunica com o backdoor, lê as atualizações do log e envia esses dados para os clientes remotos conectados. Os clientes recebem os logs em tempo real e os exibem em seus terminais.

5.6 Conclusão Final

A união dos códigos `backdoor.c`, `server.py` e `client.py` nos permite desenvolver um sistema completo de backdoor com envio de logs para clientes remotos. É importante enfatizar que o uso de backdoors sem o consentimento e a autorização dos usuários é estritamente proibido e considerado ilegal. Este artigo teve o objetivo de fornecer uma visão geral dos componentes e da lógica por trás desse sistema, com fins educativos e de pesquisa. É fundamental utilizar esses conhecimentos de forma ética e responsável, respeitando sempre a privacidade e a segurança das pessoas.