Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Manual Técnico de Análise Cibernética

Implementação e Análise de Conexões SSH com Criptografia Clássica e Híbrida Pós-Quântica em Ambientes Monitorados

Autora: Katiuscia Scarabottolo de Morais
Orientadora: Profa. Dra. Ana Karina Dourado Salina de Oliveira

Campo Grande – MS
2025

Sumário

Parte I

Introdução e Preparação Geral

Capítulo 1

Introdução

Este manual técnico apresenta a implementação, configuração e análise de conexões SSH em um ambiente controlado de laboratório, com foco na comparação entre criptografia clássica e criptografia híbrida pós-quântica. O propósito central é documentar, de forma estruturada e reprodutível, os procedimentos empregados na avaliação de mecanismos de segurança aplicados ao protocolo SSH, bem como observar seus impactos sobre ferramentas de monitoramento e detecção em ambientes de segurança cibernética.

O ambiente descrito simula um cenário realista de comunicação segura, composto por um servidor SSH real, uma máquina atacante e uma estação dedicada ao monitoramento, possibilitando a análise de parâmetros técnicos como tempo de negociação de chaves, padrões de tráfego, geração de logs e comportamento das soluções de segurança diante de diferentes perfis criptográficos.

1.1 Objetivo do Manual

O objetivo deste manual é descrever de forma sistemática e técnica os procedimentos necessários para:

- Implementar um ambiente isolado e controlado para testes de conexões SSH.
- Configurar e validar mecanismos de criptografia clássica e híbrida pós-quântica no ambiente com OpenSSH.
- Avaliar o comportamento das conexões sob monitoramento ativo.
- Analisar indicadores técnicos relacionados ao desempenho criptográfico e à geração de eventos de segurança.
- Fornecer diretrizes reprodutíveis para estudos comparativos em segurança cibernética com foco em resistência pós-quântica.

O manual destina-se a pessoas interessadas em compreender os efeitos da evolução criptográfica sobre a proteção de canais de comunicação segura, bem como a orientar aqueles que desejam reproduzir o ambiente experimental apresentado, permitindo realizar seus próprios testes e análises.

1.2 Escopo e Limitações

O escopo deste manual abrange exclusivamente o ambiente experimental de laboratório, configurado para fins acadêmicos, contemplando:

- Estabelecimento de conexões SSH em rede local isolada.
- Configuração do ambiente utilizando algoritmos criptográficos clássicos (como ECDH, RSA, AES).
- Configuração do ambiente utilizando algoritmos híbridos pós-quânticos baseados em ML-KEM (Kyber).
- Monitoramento e análise de tráfego por meio de ferramentas SOC (Suricata, Zeek, Splunk).

As principais limitações do estudo são:

- Os resultados não representam ambientes de produção em larga escala.
- A infraestrutura não está exposta à Internet pública.
- O desempenho observado é dependente do hardware e da carga do sistema.
- Não são considerados cenários de ataques externos em escala real.

1.3 Visão Geral do Ambiente

O ambiente experimental é composto por três máquinas principais, cada uma com função específica:

- **Servidor SSH Real**: baseado na imagem *Debian 12.12 netinst.iso*, responsável por hospedar o serviço OpenSSH configurado com diferentes perfis criptográficos.
- **Máquina Atacante**: executando *Kali Linux*, utilizada para simulação de conexões SSH e ações ofensivas controladas.
- **Máquina de Monitoramento**: baseada no *Kali Purple*, destinada à coleta, análise e correlação dos dados de tráfego e eventos de segurança.

A comunicação ocorre em rede local isolada, sendo o tráfego replicado para a máquina de monitoramento por meio de espelhamento de portas, permitindo inspeção passiva sem interferência no fluxo de dados.

1.4 Arquitetura do Laboratório

A arquitetura do laboratório foi concebida com foco em isolamento, controle e reprodutibilidade dos experimentos. A topologia empregada é composta por:

- Comunicação direta entre a máquina atacante e o servidor SSH real.
- Switch gerenciável com espelhamento de portas (Port Mirroring) ativado.
- Captura passiva do tráfego pela máquina de monitoramento.

O fluxo de comunicação pode ser representado da seguinte forma:

- Kali Linux (Atacante) → Debian 12.12 (Servidor SSH)
- Switch com Port Mirroring → Kali Purple (Monitoramento)

Essa configuração viabiliza a análise simultânea do comportamento criptográfico e da resposta das ferramentas de segurança, preservando a integridade do ambiente experimental frente a cenários simulados de ataque.

Capítulo 2

Requisitos

2.1 Hardware Necessário

Para a implementação do ambiente descrito, recomenda-se a seguinte configuração mínima:

- Computador com suporte à virtualização (Intel VT-x ou AMD-V).
- Processador com no mínimo 4 núcleos.
- 16 GB de memória RAM (recomendado).
- Armazenamento com no mínimo 200 GB disponíveis.
- Switch gerenciável com suporte a Port Mirroring, sendo que, no experimento atual documentado neste manual, foi utilizado especificamente o switch gerenciável **TP-Link TL-SG105E**, com recurso de espelhamento de portas habilitado.
- Interfaces de rede dedicadas.

O uso de máquinas físicas ou virtuais é permitido, desde que mantidos os requisitos mínimos de desempenho e isolamento.

2.2 Software e Distribuições Utilizadas

As seguintes distribuições e ferramentas compõem o ambiente utilizado neste estudo:

- **Debian 12.12**: servidor SSH real.
- **Kali Linux**: máquina atacante.
- **Kali Purple**: estação de monitoramento SOC.
- **OpenSSH customizado**: compilado com suporte à criptografia pós-quântica.
- **OpenSSL 3.x com OQS Provider**: implementação dos algoritmos híbridos.
- **Suricata**: IDS/IPS para detecção de tráfego suspeito.

- **Zeek**: análise de protocolos e geração de logs.
- **Splunk**: centralização e correlação de eventos.
- **Wireshark/Tcpdump**: captura e inspeção de pacotes.

Recomenda-se que as versões específicas de cada ferramenta sejam registradas em documentação complementar, a fim de assegurar rastreabilidade e reprodutibilidade dos resultados.

Parte II

Instalação dos Sistemas e Ambiente de Execução

Capítulo 3

Instalação e Configuração das Máquinas Virtuais

3.1 Ferramentas Utilizadas

Nesta seção, serão listadas as ferramentas e recursos necessários para a execução dos ambientes virtuais e das atividades propostas ao longo deste manual. Certifique-se de que todas as ferramentas estejam devidamente instaladas ou acessíveis antes de prosseguir para as próximas etapas.

- **Oracle VM VirtualBox** – Software de virtualização de código aberto, que permite executar máquinas virtuais em seu computador.
- **VirtualBox Extension Pack** – Pacote adicional que habilita funcionalidades como USB 2.0/3.0, RDP, criptografia de disco, entre outras.
- **Imagem de Máquina Virtual (VM)** – Arquivo de máquina virtual contendo o sistema operacional já pré-configurado (Linux, Windows, ou ambos, dependendo do cenário).
- **Navegador de Internet** – Utilizado para acessar os links de download e recursos online.
- **Computador com suporte à virtualização** – Mínimo 8 GB de RAM (ideal 16 GB), 100 GB de espaço livre em disco, sistema operacional hospedeiro Windows 10/11 ou Linux (Ubuntu, Debian, etc)

3.2 Preparação do Ambiente: Instalação do VirtualBox

VirtualBox é um software gratuito de virtualização. Ele permite a execução de sistemas operacionais (como Linux ou Windows) dentro de uma janela, no computador pessoal. Essas "janelas"são chamadas de **máquinas virtuais**, ou VMs.

Nesta etapa, será realizada a instalação completa do Oracle VM VirtualBox e de seu Extension Pack no sistema operacional Windows 11. O VirtualBox será usado para executar as máquinas virtuais utilizadas nas atividades práticas deste manual. Além disso, o Extension Pack é necessário para garantir suporte a funcionalidades como dispositivos USB 2.0/3.0, criptografia de disco, e RDP.

3.2.1 Download do VirtualBox

1. Acessar o site Oficial do virtualBox:

   https://www.virtualbox.org/wiki/Downloads

2. Buscar a seção **"VirtualBox Platform Packages"**.

3. Clicar na opção correspondente ao sistema operacional do seu computador. Para a realização deste manual foi utilizado o sistema operacional Windows 11; portanto, selecionou-se a opção `Windows hosts`, conforme ilustrado na Figura 3.1.

Figura 3.1: Selecionando opção de download do VirtualBox

4. O download do instalador começará automaticamente.

5. Salvar o arquivo em uma pasta de fácil acesso, como a pasta `Downloads`.

6. O arquivo terá um nome semelhante a: `VirtualBox-7.1.8-168469-Win.exe`

3.2.2 Download do Extension Pack

O Extension Pack adiciona funções extras importantes para o funcionamento das máquinas virtuais, como: suporte a dispositivos USB 2.0 e 3.0, a criptografia de disco e a conexões remotas (RDP).

1. Ainda na página de downloads do VirtualBox, localizar a seção:

   "Oracle VM VirtualBox Extension Pack"

2. Clicar em Accept and download para realizar o download do arquivo, conforme mostrado na Figura 3.2.

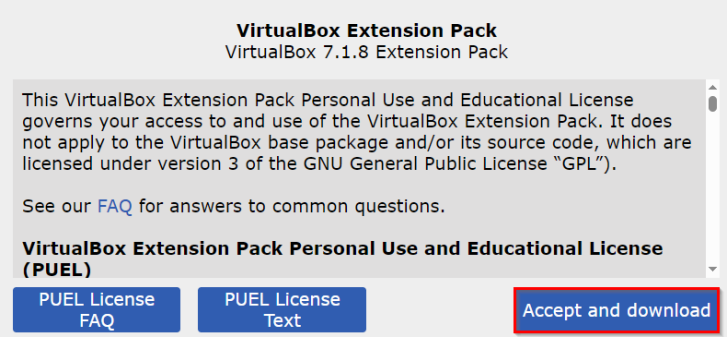

Figura 3.2: Download do Extension Pack

3. Salvar o arquivo com extensão `.vbox-extpack` junto ao instalador do VirtualBox.

4. O nome do arquivo será algo como: `Oracle_VirtualBox_Extension_Pack-7.1.8.vbox-extpack`

3.2.3 Instalação do VirtualBox

1. Localizar o arquivo `.exe` que foi feito download. E dar início à instalação.

2. Um assistente de instalação será aberto.

   - **Tela de Boas-vindas (Welcome)**: Clicar em Next, conforme apresentado na Figura 3.3.

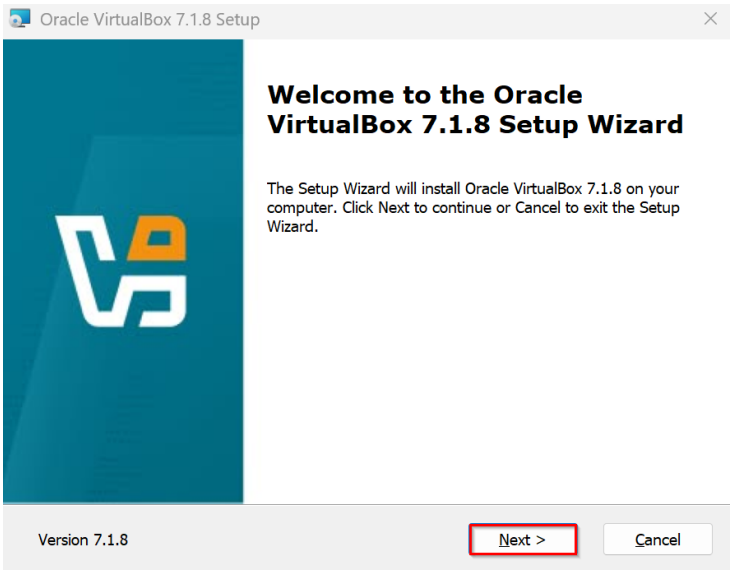

Figura 3.3: Abrindo o Assistente de Instalação do VirtualBox.

   - **Aceitando os Termos da Licença**: Na tela `End-User License Agreement`, após a leitura dos termos, selecionar `I accept the terms in the License Agreement` e, em seguida, clicar em `Next`, como ilustrado na Figura 3.4.

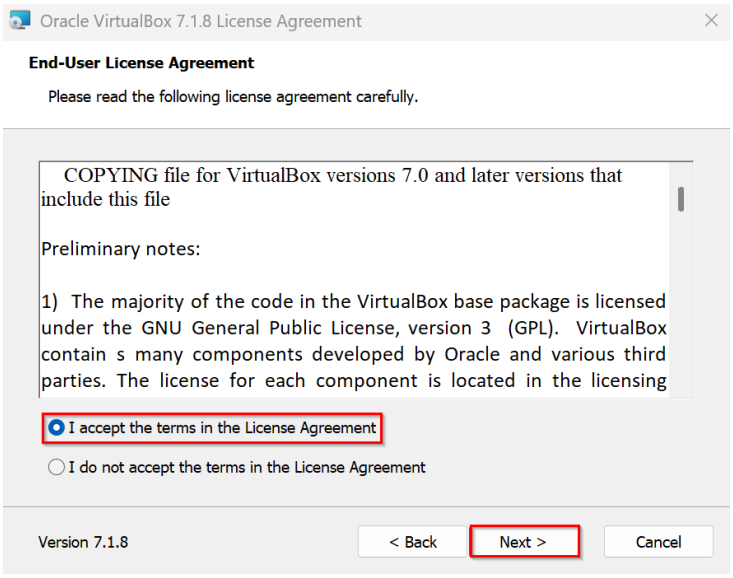

Figura 3.4: Aceitando os Termos da Licença de Uso do VirtualBox

- **Componentes a instalar**: Na tela `Custom Setup`, manter todas as opções padrão habilitadas e prosseguir pressionando `Next`, conforme Figura 3.5.

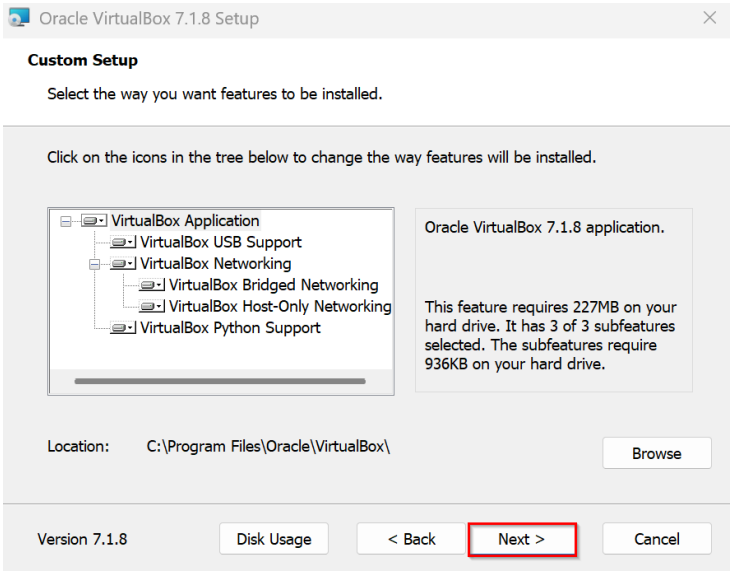

Figura 3.5: Instalação dos Componentes do VirtualBox

- **Aviso sobre conexão de rede**: Na janela **Warning: Network Interfaces**, clicar em `Yes` para autorizar a instalação dos drivers necessários, conforme Figura 3.6.

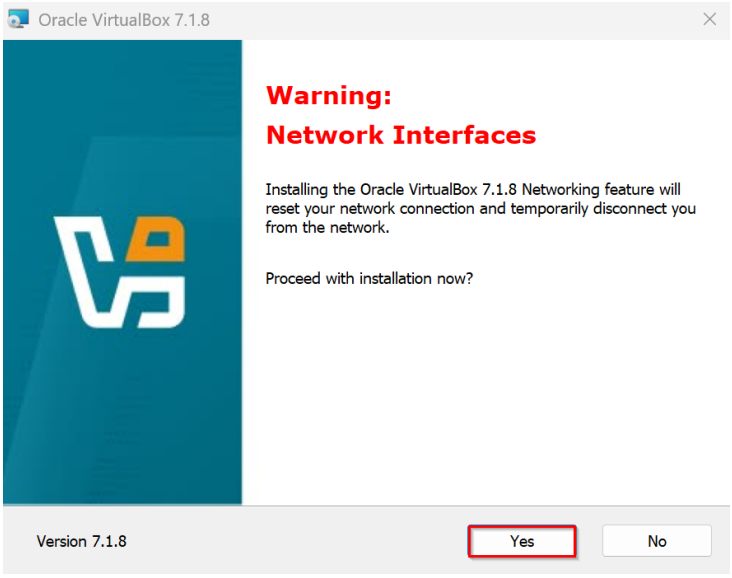

Figura 3.6: Instalação das Interfaces de Rede

- **Atalhos no sistema**: Manter as opções padrão ou marcar apenas as preferidas pelo usuário. Em seguida, clicar em `Next` (Figura 3.7).

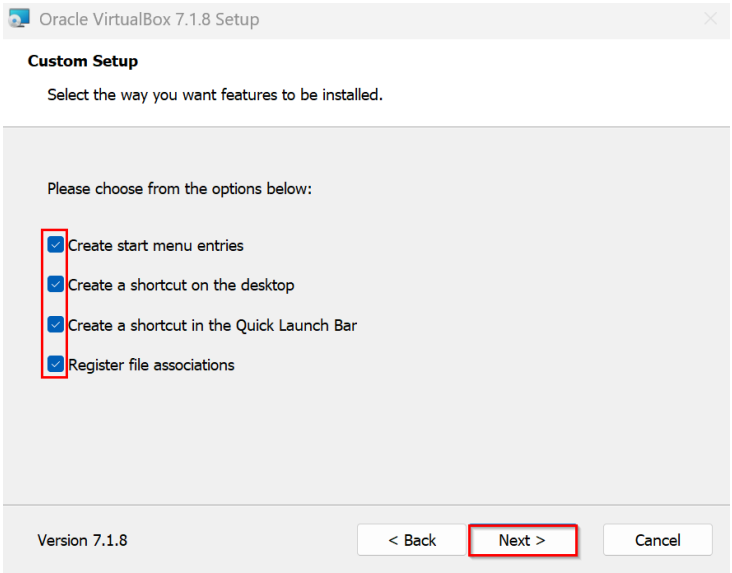

Figura 3.7: Instalação dos Atalhos do VirtualBox

- **Pronto para a Instalação**: Clicar em `Install` para iniciar o processo de instalação (Figura 3.8), e aguardar a conclusão (Figura 3.9).

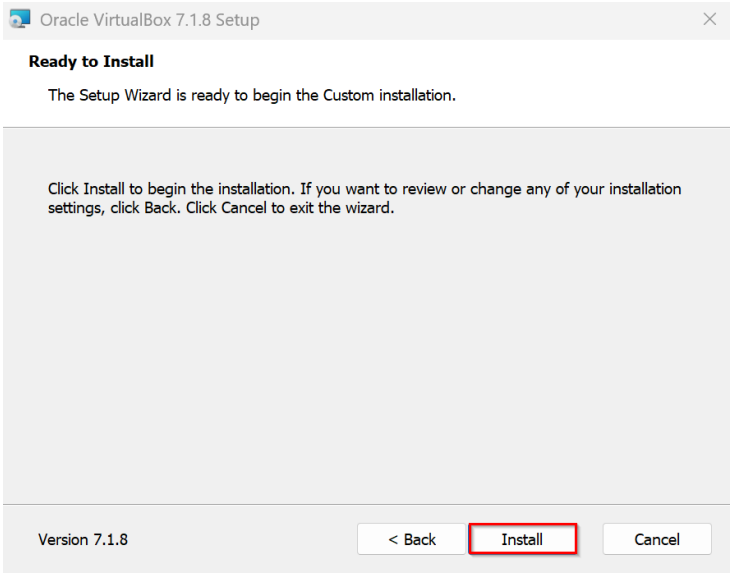

Figura 3.8: Janela de Confirmação para Início da Instalação do VirtualBox

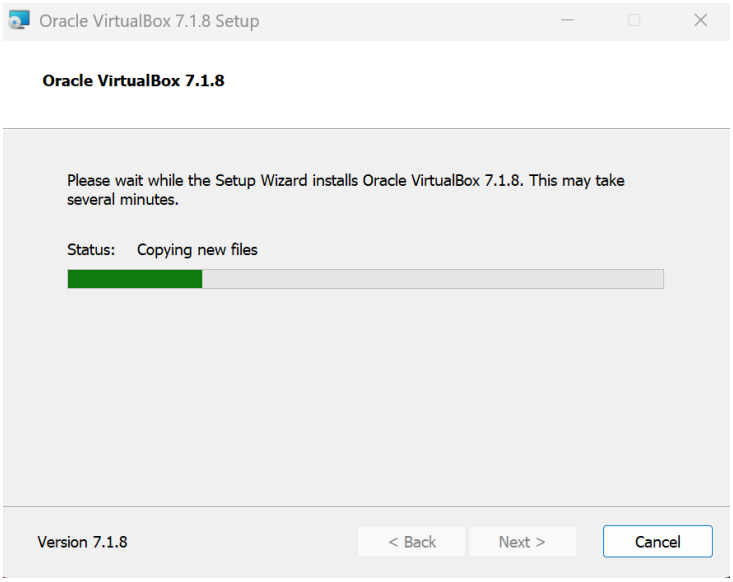

Figura 3.9: Visualização do processo de instalação.

- **Finalização**: Após concluir o processo, clicar em `Finish` para encerrar a instalação (Figura 3.10).

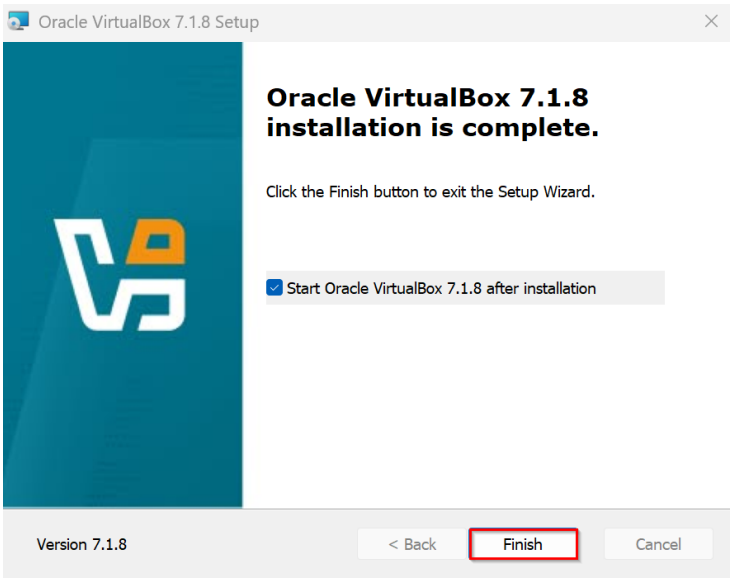

Figura 3.10: Instalação Finalizada.

3. Caso a opção `Start Oracle VirtualBox 7.1.8 after installation` esteja marcada, o VirtualBox será iniciado automaticamente ao final do processo de instalação (Figura 3.11).

Figura 3.11: Janela de Início do VirtualBox.

3.2.4 Instalação do Extension Pack

1. Com o VirtualBox aberto, clicar no menu superior em: `1.Arquivo > 2.Ferramentas > 3.Gerenciador de Pacotes de Extensão` conforme Figura 3.12.

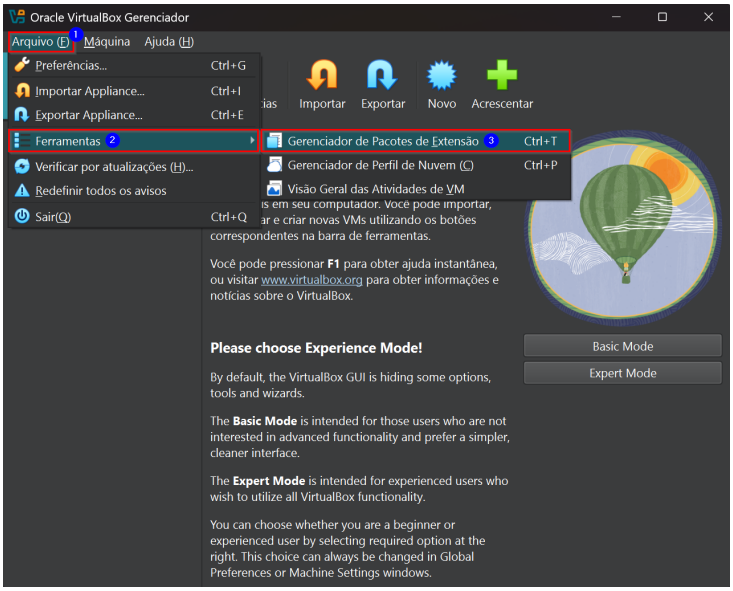

Figura 3.12: Acessando o Gerenciador de Pacotes de Extensão

2. Na nova janela, clicar no ícone de + **(Instalar)** para adicionar um novo pacote (Figura 3.13).

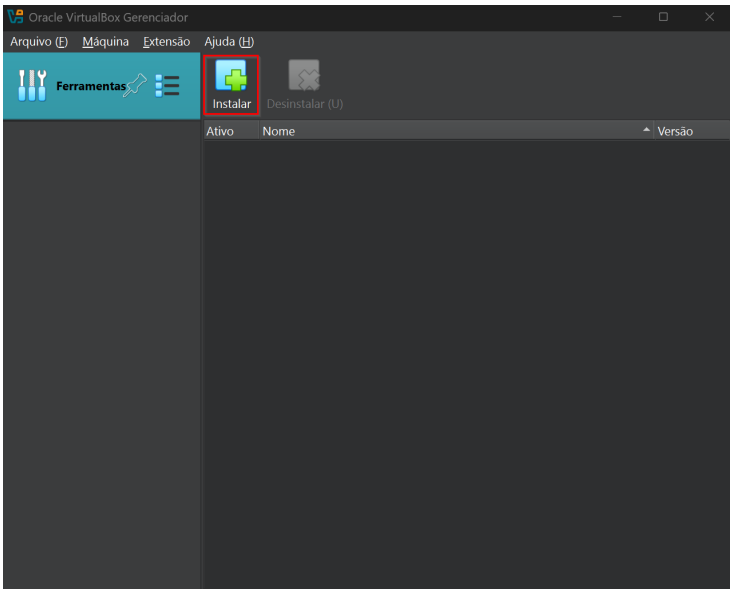

Figura 3.13: Inicializando o Processo de Instalação do Pacote de Extensão.

3. Selecionar o arquivo `.vbox-extpack` previamente baixado e confirmar a escolha (Figura 3.14).

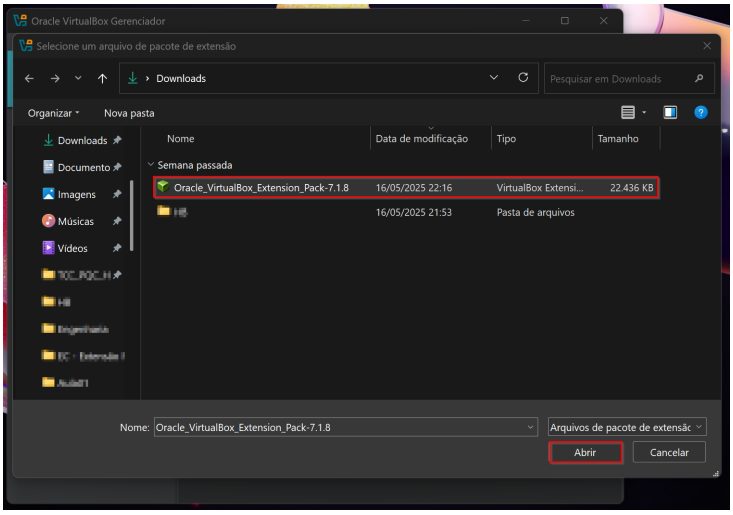

Figura 3.14: Janela de Início do VirtualBox.

4. Um aviso será exibido informando que alguns componentes instalados pelo pacote de extensão podem oferecer riscos ao sistema caso a origem não seja confiável. Se o arquivo foi obtido de fonte segura, clicar em `Instalar` para prosseguir (Figura 3.15).

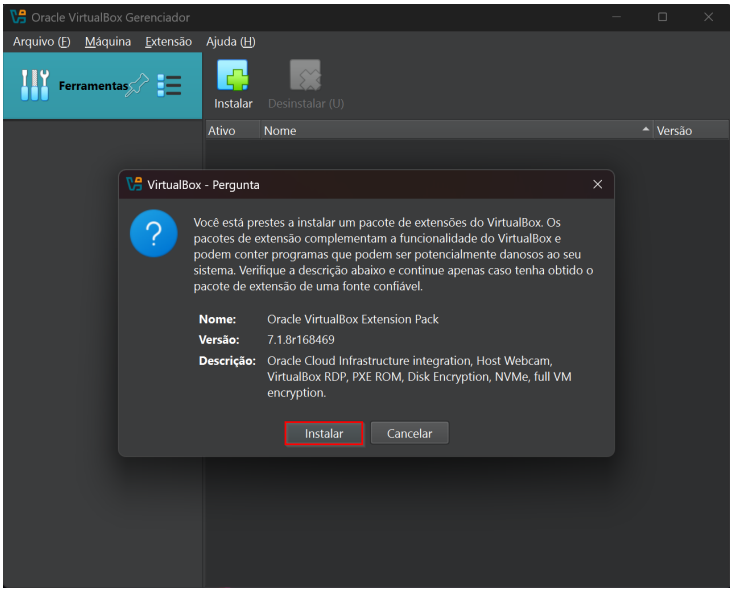

Figura 3.15: Janela de Aviso e Confirmação da Instalação

5. Na tela dos **Termos de Licença**, rolar o texto até o final (requisito do instalador) e, se estiver de acordo com os termos, clicar em `I Agree` para prosseguir, conforme a Figura 3.16.

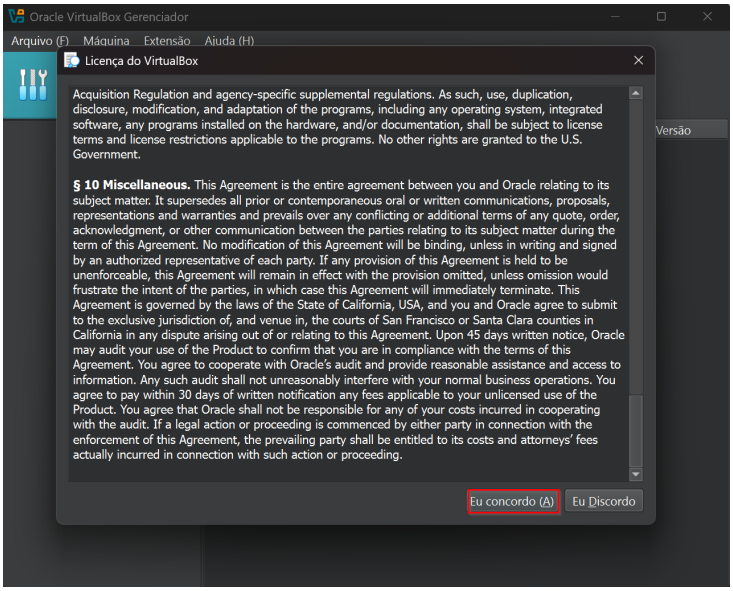

Figura 3.16: Termos de Licença.

6. Após a instalação, o Gerenciador mostrará o pacote listado com sua versão e status indicando que está ativo (Figura 3.17).

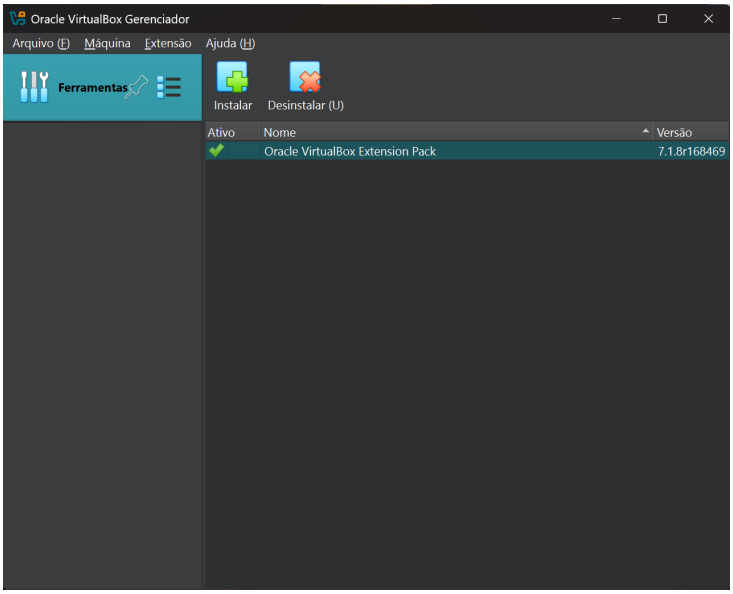

Figura 3.17: Pacote de Extensão Instalado e Ativo.

7. Se o usuário desejar voltar à tela de início, clicar em 1.Ferramentas > 2.Bem-vindo, conforme mostrado na Figura 3.18.

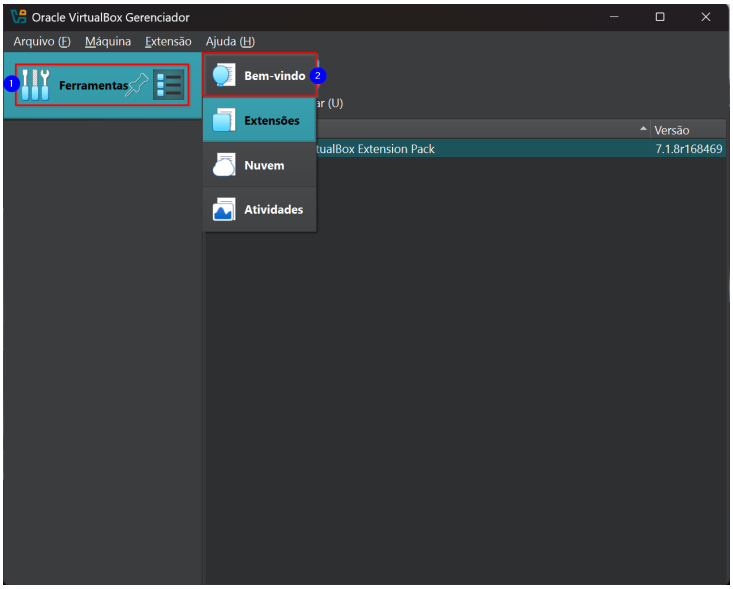

Figura 3.18: Retornando à Janela de Início do VirtualBox.

3.3 Criação das Máquinas Virtuais

Nesta seção são descritos os procedimentos para criação das máquinas virtuais que podem ser utilizadas no laboratório: servidor SSH real (Debian 12.12), máquina atacante (Kali Linux) e máquina de monitoramento (Kali Purple).

As telas apresentadas nas Figuras 3.19 a 3.24 referem-se ao processo de criação da máquina virtual do servidor SSH (Debian). Para as demais máquinas, o procedimento é análogo, alterando-se apenas o nome da VM, a imagem *ISO* e os recursos alocados.

3.3.1 Servidor SSH (VM Debian)

1. Abrir o **Oracle VM VirtualBox**. A tela inicial do gerenciador de máquinas virtuais é exibida conforme a Figura 3.19. O botão **Novo** (*New*), utilizado para criar uma nova máquina virtual, está destacado com o marcador **(1)**.

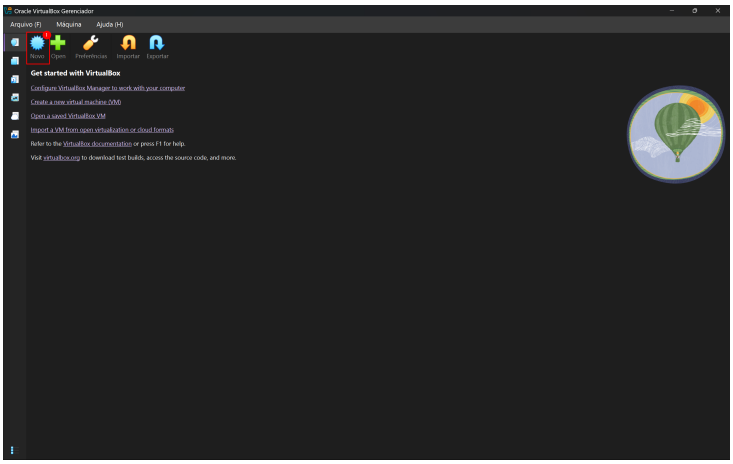

Figura 3.19: Janela inicial do Oracle VM VirtualBox, com o botão **Novo** destacado em **(1)**.

2. Clicar no botão **Novo** para iniciar a criação de uma nova máquina virtual.

3. Na tela **New Virtual Machine**, preencher os campos principais conforme a Figura 3.20, observando os marcadores numéricos:

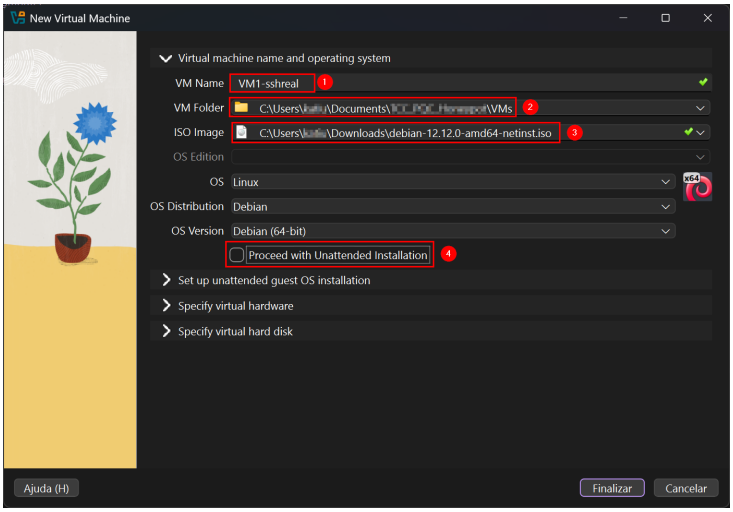

Figura 3.20: Configuração inicial da VM Debian 12.12: (1) nome da VM, (2) pasta de armazenamento, (3) imagem ISO e (4) desativação da instalação desassistida.

   - **(1) VM Name**: definir o nome da máquina virtual, nesse caso foi definido como `VM1-sshreal`.
   - **(2) VM Folder**: selecionar a pasta onde serão armazenados os arquivos da VM (por exemplo, `C:\Users\...\TCC_PQC_SSHReal\VMs`).
   - **(3) ISO Image**: apontar para a imagem `debian-12.12.0-amd64-netinst.iso`.
   - **(4) Proceed with Unattended Installation**: manter esta opção **desmarcada**, pois a instalação será feita manualmente dentro do instalador do Debian.

   Os campos de sistema operacional (*OS*, *OS Distribution* e *OS Version*) são automaticamente ajustados para `Linux`, `Debian` e `Debian (64-bit)` após a seleção da ISO.

4. A seção **Set up unattended guest OS installation** (Figura 3.21) pode ser utilizada para automatizar criação de usuário, senha e opções de instalação. Neste manual, a instalação do Debian será documentada passo a passo, portanto a instalação desassistida não será utilizada.

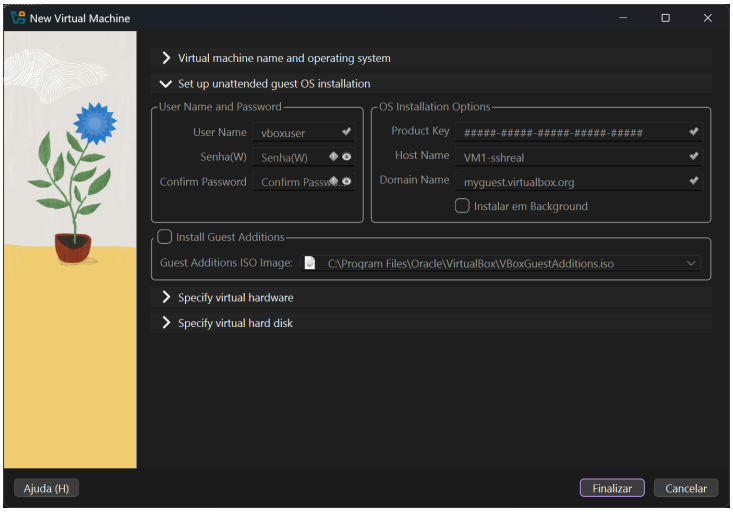

Figura 3.21: Opções de instalação desassistida (*unattended*). Não utilizadas neste manual.

5. Em **Specify virtual hardware**, ajustar a quantidade de memória RAM e o número de CPUs virtuais conforme a Figura 3.22:

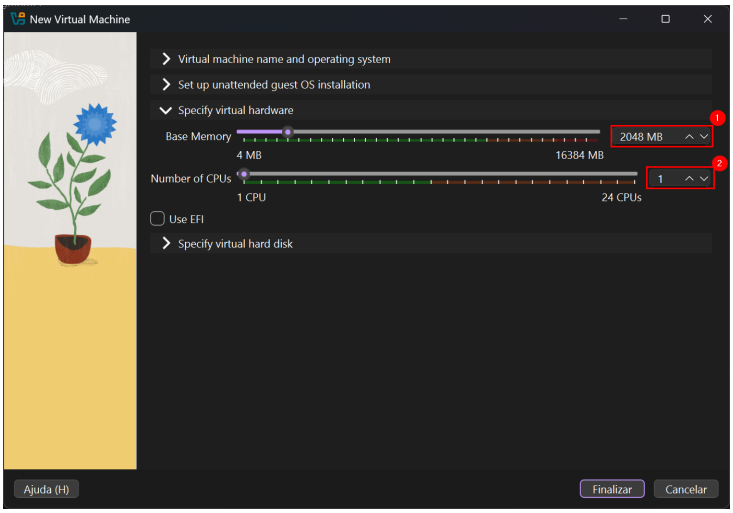

Figura 3.22: Definição de recursos de hardware virtual: (1) memória RAM e (2) número de vCPUs.

- **(1) Base Memory**: definir, por exemplo, `2048 MB` (2 GB), de acordo com o perfil recomendado para o servidor SSH.
- **(2) Number of CPUs**: selecionar a quantidade de vCPUs, por exemplo, `1 CPU` ou `2 CPUs`, conforme os recursos do hospedeiro.

Esses valores devem respeitar os perfis descritos na Seção 3.4.

6. Em **Specify virtual hard disk**, selecionar **Create a New Virtual Hard Disk** e configurar o disco virtual conforme a Figura 3.23:

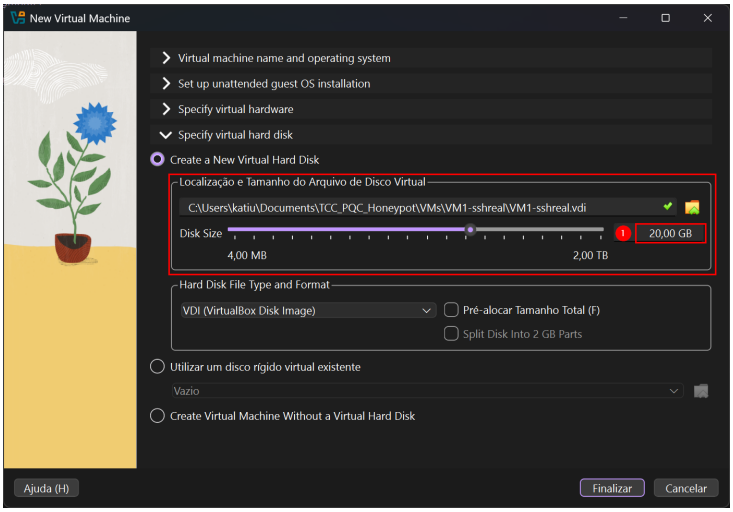

Figura 3.23: Criação e dimensionamento do disco virtual: destaque em (1) para o tamanho do disco.

- **(1) Disk Size**: definir o tamanho do disco, por exemplo, `20,00 GB`, correspondente ao perfil recomendado para o servidor SSH.

O tipo de arquivo de disco `VDI (VirtualBox Disk Image)` com alocação dinâmica é adequado para os testes realizados neste manual.

7. Após revisar as configurações, clicar em **Finalizar**. A nova máquina virtual `VM1-sshreal` será exibida na lista de VMs do VirtualBox com o status **Desligada** (*Powered Off*), conforme a Figura 3.24.

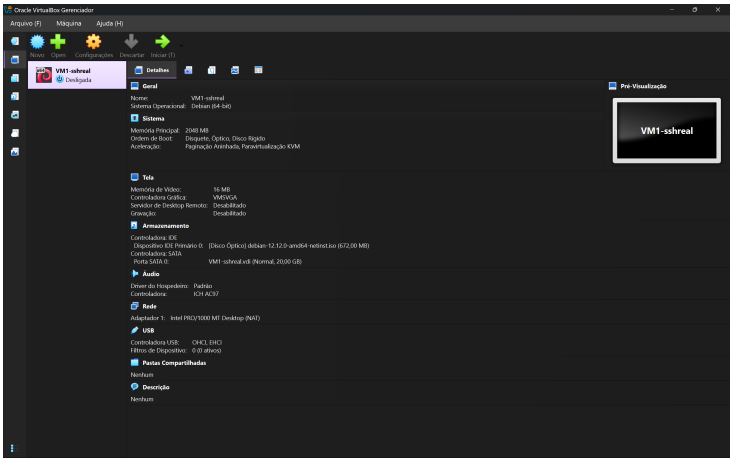

Figura 3.24: Máquina virtual `VM1-sshreal` criada no VirtualBox com os recursos configurados.

3.3.2 Máquina Atacante (VM Kali Linux)

A criação da máquina virtual da **máquina atacante** segue o mesmo procedimento descrito para o servidor SSH, utilizando o mesmo fluxo de telas do VirtualBox. As principais adaptações são:

1. Na tela **New Virtual Machine**, configurar:

- **VM Name**: `VM2-attacker` (ou outro nome padronizado).
- **VM Folder**: mesma pasta base utilizada para as demais VMs do projeto.
- **ISO Image**: imagem do *Kali Linux* utilizada no laboratório.
- **OS** / **OS Distribution** / **OS Version**: ajustados para Linux 64 bits compatível com o Kali.

2. Em **Specify virtual hardware**, utilizar os perfis da Tabela 3.1 para definir RAM e vCPUs.

3. Em **Specify virtual hard disk**, selecionar tamanho de disco de acordo com o perfil (por exemplo, 30–40 GB para o perfil recomendado).

4. Concluir clicando em **Finalizar**.

3.3.3 Máquina de Monitoramento (VM Kali Purple)

A máquina de monitoramento, responsável pela execução das ferramentas de SOC, também é criada a partir do mesmo fluxo:

1. Na tela **New Virtual Machine**, configurar:

   - **VM Name**: `VM3-monitor` ou `VM3-kalipurple`.
   - **VM Folder**: mesma pasta base das demais VMs.
   - **ISO Image**: imagem do *Kali Purple*.
   - **OS** / **OS Distribution** / **OS Version**: Linux 64 bits compatível.

2. Em **Specify virtual hardware**, alocar recursos seguindo, no mínimo, o perfil *Recomendado* para a VM de monitoramento (Tabela 3.1), visto que serão executados Suricata, Zeek e Splunk.

3. Configurar o tamanho do disco virtual conforme a mesma tabela (80–100 GB para o perfil recomendado).

4. Finalizar a criação da VM clicando em **Finalizar**.

3.4 Alocação de Recursos (CPU, RAM, Disco)

A definição adequada de recursos de hardware virtual (CPU, memória RAM e disco) é essencial para garantir que os testes de criptografia e monitoramento ocorram de forma estável, sem sobrecarregar o sistema hospedeiro.

A Tabela 3.1 apresenta três perfis de alocação para cada papel de máquina virtual:

- **Mínimo**: configuração funcional, porém limitada; indicada apenas para testes rápidos ou máquinas hospedeiras com poucos recursos.

- **Recomendado**: configuração utilizada como base para os experimentos deste manual, buscando equilíbrio entre desempenho e consumo de recursos.

- **Ideal**: configuração mais confortável, destinada a cenários com repetição intensa de atos, maior volume de logs e histórico prolongado.

Tabela 3.1: Perfis de alocação de recursos para cada máquina virtual.

Papel da VM	**Perfil**	**RAM**	**CPU**	**Disco**	**Observações técnicas**
SSH real	Mínimo	1 GB	1	15–20 GB	SSH ativo, poucas conexões simultâneas.
SSH real	Recomendado	2 GB	2	20–30 GB	Testes de autenticação e troca criptográfica.
SSH real	Ideal	4 GB	2	40 GB	Execução estável com logging detalhado.
Attacker	Mínimo	2 GB	1–2	20 GB	Execução básica dos atos via SSH, testes simples de conexão.
Attacker	Recomendado	4 GB	2	30–40 GB	Execução dos scripts e testes criptográficos.
Attacker	Ideal	6 GB	4	50 GB	Vários atos em paralelo e maior volume de payload/exfiltração.
Monitor	Mínimo	4 GB	2	40 GB	Suricata ou Zeek isolado, Splunk limitado.
Monitor	Recomendado	8 GB	4	80–100 GB	Suricata + Zeek + Splunk estáveis.
Monitor	Ideal	12–16 GB	6–8	120–200 GB	SOC completo, histórico longo e múltiplos atos.

Capítulo 4

Instalação dos Sistemas Operacionais

4.1 Debian (Servidor SSH)

O servidor SSH do laboratório será baseado na distribuição **Debian GNU/Linux 12** (codinome *Bookworm*), utilizando a imagem de instalação mínima (*netinst*). Este procedimento é válido tanto para máquinas virtuais (VMs) quanto para máquinas físicas; quando houver diferenças entre os dois cenários, elas serão indicadas explicitamente ao longo desta seção.

4.1.1 Preparação da ISO

Para o servidor SSH foi utilizada a seguinte imagem de instalação:

- **Nome do arquivo**: `debian-12.12.0-amd64-netinst.iso`
- **Arquitetura**: 64 bits (`amd64`)
- **Tipo**: imagem *netinst* (instalador mínimo via rede)
- **Origem**: página oficial do instalador Debian Bookworm

A imagem *netinst* é particularmente adequada para este servidor por permitir uma instalação mais enxuta e controlada, trazendo apenas o conjunto básico de pacotes e baixando o restante diretamente dos repositórios oficiais. Isso reduz o tamanho da mídia de instalação e evita a inclusão desnecessária de componentes gráficos ou serviços que não serão utilizados no ambiente de laboratório. Em cenários com conexão de rede estável, como neste projeto, o uso da *netinst* é preferível a imagens maiores que já incluem ambientes de área de trabalho completos.
A imagem pode ser obtida em:

https://www.debian.org/releases/bookworm/debian-installer/

Na época da realização deste manual, a versão `12.12.0` era a versão **estável** mais recente disponível para a arquitetura `amd64`. Recomenda-se sempre utilizar uma imagem oficial da página do Debian, evitando fontes de terceiros.

Uso em máquina virtual (VirtualBox)

Para uso em máquina virtual no Oracle VM VirtualBox:

1. Fazer o *download* do arquivo `debian-12.12.0-amd64-netinst.iso` e salvá-lo em uma pasta de fácil acesso (por exemplo, `Downloads` ou uma pasta específica do projeto).
2. Durante a criação da VM (Seção anterior), selecionar esta ISO no campo **ISO Image** da tela *New Virtual Machine*.
3. Verificar se a ordem de boot da VM está configurada para iniciar pelo **Disco Óptico / Optical** (ou equivalente), garantindo que o instalador do Debian seja carregado na primeira inicialização.

Uso em máquina física (pendrive de instalação)

Para instalação em máquina física, é necessário gravar a ISO em um pendrive de boot:

1. Separar um **pendrive USB** com, no mínimo, **4 GB** de capacidade. Todos os dados existentes serão apagados.
2. Em um computador com Windows ou Linux, utilizar uma ferramenta de criação de mídia de boot, como:
    - Rufus, Balena Etcher, Ventoy ou ferramenta equivalente.
3. Selecionar:
    - A unidade correspondente ao pendrive USB.
    - A imagem `debian-12.12.0-amd64-netinst.iso` como fonte.
    - Esquema de partição e sistema de destino adequados ao equipamento (por exemplo, `GPT + UEFI`, quando aplicável).
4. Iniciar o processo de gravação e aguardar a conclusão.
5. No equipamento onde o Debian será instalado:
    (a) Conectar o pendrive USB.
    (b) Acessar a BIOS/UEFI e configurar a ordem de boot para iniciar pelo pendrive.
    (c) Salvar as alterações e reiniciar o equipamento.

Após esses passos, tanto a máquina virtual quanto a máquina física estarão prontas para iniciar o instalador do Debian a partir da ISO preparada.

4.1.2 Instalação Passo a Passo

Com a mídia de instalação preparada (ISO anexada à VM ou pendrive de boot em máquina física), o próximo passo é executar a instalação do Debian no servidor SSH.

1. **Tela inicial do instalador**

   Inicie o computador (físico ou virtual) a partir da mídia de instalação do Debian. Na tela de boot do instalador, serão exibidas opções como **Install**, **Graphical Install** e outras variantes.

   Neste manual será utilizada a opção **Install** (modo texto), por dois motivos principais:

   - é mais **leve** em termos de consumo de recursos (CPU, RAM e vídeo), o que é mais adequado ao perfil de **servidor**;
   - reflete melhor um cenário de **servidor sem interface gráfica**, comum em ambientes de produção e em laboratórios focados em serviços de rede (como SSH).

   Assim, selecione a opção **Install** e pressione Enter, conforme a Figura 4.1.

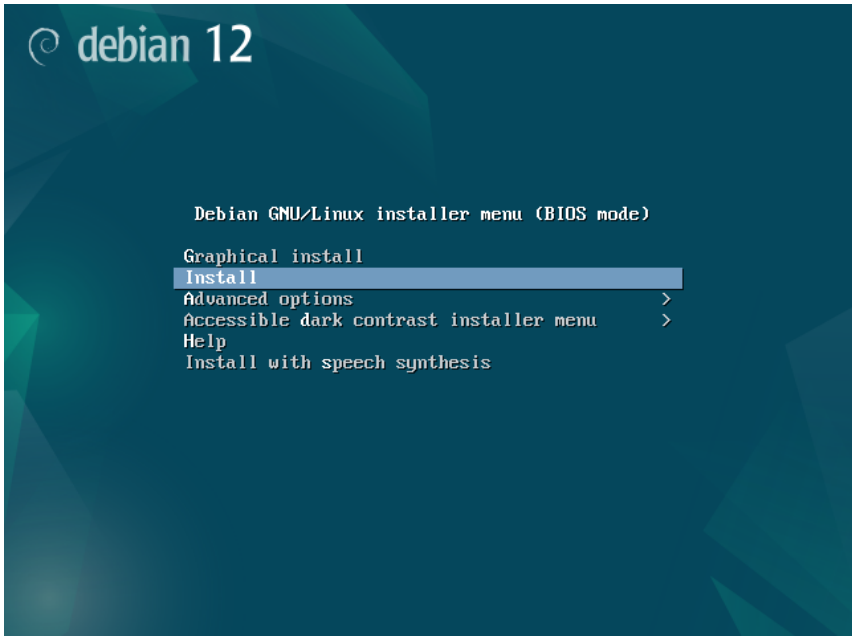

Figura 4.1: Tela inicial do instalador Debian: seleção da opção **Install** (modo texto), recomendada para servidores.

2. **Idioma, localidade e teclado**

   Selecionar o idioma **Portuguese (Brazil)**, país **Brasil** e layout de teclado **Português Brasileiro**. Essas opções afetam apenas o idioma das mensagens e o mapeamento de teclado, não interferindo na configuração de rede ou dos serviços.

3. **Configuração de rede (básica)**

   O comportamento desta etapa varia conforme o tipo de ambiente:

- **Máquina virtual (VirtualBox)**: a interface de rede é detectada automaticamente pelo instalador e não é apresentada nenhuma opção de escolha. A configuração prossegue diretamente utilizando a interface virtual conectada ao adaptador configurado na VM (NAT ou Bridge), normalmente via DHCP.
- **Máquina física**: o instalador pode solicitar que o usuário escolha qual interface será utilizada, podendo aparecer opções como:
  - **Ethernet (rede cabeada)** – geralmente identificada como `eth0`, `enp0s3`, `enpXsY`, entre outros.
  - **Wireless (Wi-Fi)** – identificada como `wlan0` ou similar.

Para ambientes de servidor e laboratório, recomenda-se priorizar a interface **Ethernet**, por oferecer maior estabilidade. Após a seleção (quando aplicável), permitir que o instalador configure automaticamente o endereço IP via DHCP. Os ajustes específicos de rede (endereçamento fixo, gateways e segmentação) serão definidos posteriormente na fase de configuração do laboratório.

4. **Nome da máquina e domínio**

   Nesta etapa, definir o nome que identificará o servidor na rede.

   - **Hostname**: utilizar um nome coerente com o papel da máquina no laboratório, por exemplo:
     - `sshreal`, `vm1-sshreal` ou `debian-ssh`.
   - **Nome de domínio**: pode ser deixado em branco ou configurado com um domínio interno, como `lab.local`, caso se deseje estruturar um namespace local.

5. **Definição de senha do usuário root**

   O instalador solicitará a definição da senha do usuário `root`. Recomenda-se configurar uma senha forte, composta por combinação de letras maiúsculas, minúsculas, números e caracteres especiais, pois esta conta possui privilégios administrativos totais no sistema.

6. **Criação de usuário padrão**

   Em seguida, será solicitado o cadastro de um usuário comum. Para este ambiente, foi adotado o seguinte padrão:

   - **Nome completo do novo usuário**: `sshserver01`
   - **Nome de usuário**: `sshserver`
   - **Senha**: definir uma senha forte e repeti-la quando solicitado, garantindo consistência.

   Este usuário será utilizado para acesso cotidiano ao sistema, podendo executar comandos administrativos via `sudo` quando necessário.

7. **Configuração do relógio do sistema**

   Selecionar a região geográfica correspondente ao local do laboratório. Para este projeto, foi configurado:

- **Fuso horário**: `America/Campo_Grande` (Mato Grosso do Sul)

Essa configuração garante coerência temporal nos logs gerados pelo sistema, o que é essencial para análises forenses e correlação de eventos em ferramentas de monitoramento.

8. **Particionamento de disco**

   Na etapa de particionamento, recomenda-se utilizar o modo assistido, adequado a ambientes de laboratório e testes:

   - Selecionar **Assistido – usar o disco inteiro** (*Guided – use entire disk*).
   - Escolher o disco correto:
     - Em VM: selecionar o disco virtual criado no VirtualBox.
     - Em máquina física: selecionar o disco destinado exclusivamente ao Debian (atenção para não sobrescrever discos com dados importantes).
   - Escolher o esquema **Todos os arquivos em uma única partição**, simplificando a gestão do sistema para fins experimentais.
   - Confirmar a gravação das alterações em disco para início da instalação (Figura 4.2).

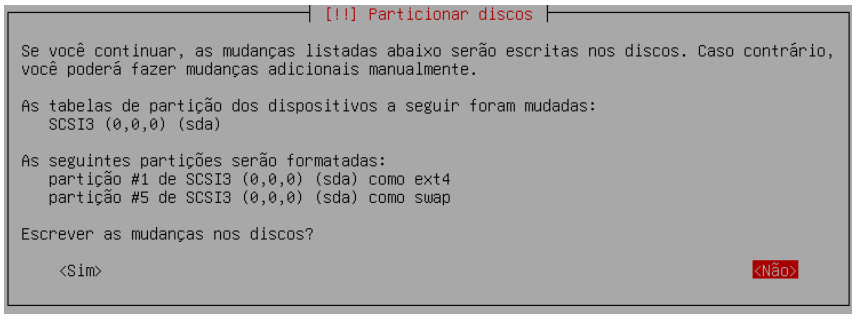

Figura 4.2: Resumo do particionamento assistido antes de gravar as alterações em disco.

9. **Configuração do gerenciador de pacotes (mirror e repositórios)**

   Nesta etapa, o instalador do Debian solicita a configuração do espelho de pacotes (*mirror*), que será utilizado para download e atualização dos softwares do sistema.

   - Selecionar o país do espelho, preferencialmente **Brasil** ou outro geograficamente próximo.
   - Aceitar o espelho padrão sugerido pelo instalador, geralmente `deb.debian.org`, que redireciona automaticamente para servidores confiáveis e atualizados.
   - Caso o ambiente utilize proxy HTTP (situação comum em redes corporativas), informar os dados solicitados. Em ambiente de laboratório, este campo normalmente permanece em branco.

   Essa configuração é responsável por garantir que o sistema utilize repositórios oficiais e estáveis do Debian para instalação e atualização de pacotes.

10. **Configuração do popularity-contest**

    O instalador poderá solicitar a participação no serviço *popularity-contest*, que coleta estatísticas anônimas sobre os pacotes mais utilizados no sistema, auxiliando a comunidade Debian na priorização de melhorias.

    Neste manual, recomenda-se selecionar a opção **Não participar**, uma vez que o ambiente é destinado exclusivamente a testes controlados em laboratório, não sendo necessária a contribuição estatística.

11. **Seleção de softwares**

    Na tela de seleção de softwares, é possível definir quais conjuntos de pacotes serão instalados inicialmente.

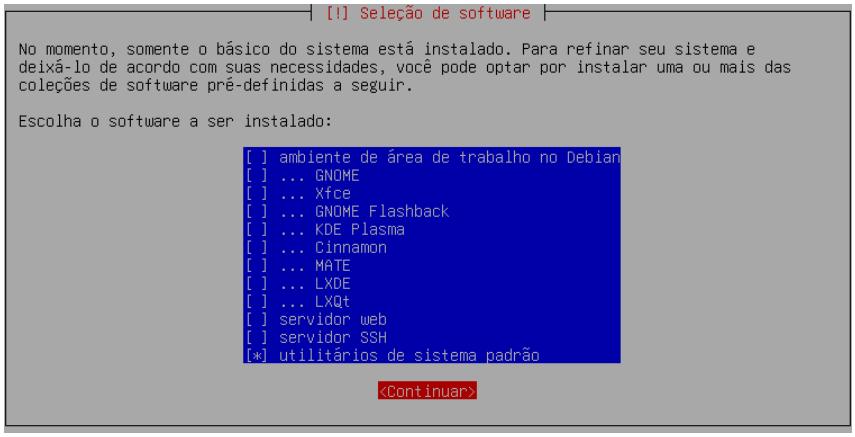

Figura 4.3: Seleção de tarefas: utilitários padrão do sistema e servidor SSH.

Para este ambiente, o objetivo é manter o sistema o mais enxuto possível, instalando apenas o essencial para um servidor base, conforme mostrado na Figura 4.3. Assim, foi adotada a seguinte configuração:

- Manter marcado apenas **Utilitários padrão do sistema** (*standard system utilities*).
- Desmarcar todos os demais componentes, incluindo:
  - Ambientes gráficos (GNOME, KDE, XFCE etc.).
  - Servidores adicionais.
  - Servidor SSH (*SSH server*) e outros serviços de rede.

O servidor SSH **não é instalado neste momento**. Ele será instalado e configurado manualmente em seção específica deste manual, o que permite controle total sobre os pacotes utilizados, versões, parâmetros criptográficos e ajustes de segurança aplicados ao serviço.

Essa escolha resulta em um sistema mais leve, com menor superfície de ataque e melhor desempenho, adequado ao ambiente experimental descrito neste manual.

12. **Instalação do GRUB**

    Na etapa de bootloader, confirmar a instalação do **GRUB** no disco principal:

- Em VM: geralmente o próprio disco virtual selecionado anteriormente.
- Em máquina física: o dispositivo físico onde o Debian foi instalado (por exemplo, `/dev/sda` ou `/dev/nvme0n1`).

13. **Conclusão da instalação**

    Após a cópia dos arquivos e configuração dos pacotes, o instalador exibirá uma mensagem indicando que a instalação foi concluída e solicitará a remoção da mídia de instalação (ISO ou pendrive) antes do primeiro reinício (Figura 4.4).

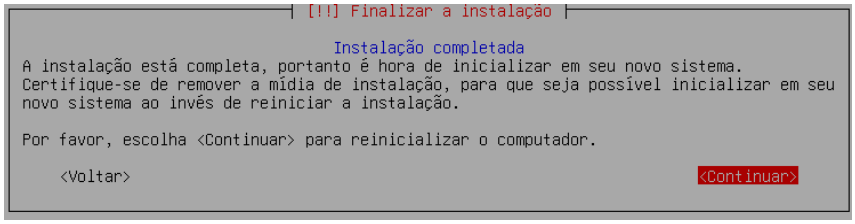

Figura 4.4: Tela de conclusão da instalação do Debian, com solicitação de reinicialização.

Confirmar a reinicialização do sistema para prosseguir com o primeiro boot.

4.2 Kali Linux (Atacante)

A máquina atacante do laboratório será baseada no **Kali Linux**, distribuição voltada para testes de segurança e *penetration testing*. Nesta seção são descritos os passos para preparação da ISO e instalação do sistema, tanto em máquina virtual quanto em máquina física, quando aplicável.

4.2.1 Preparação da ISO

Para a máquina atacante foi utilizada a seguinte imagem de instalação:

- **Nome do arquivo**: `kali-linux-2025.3-installer-amd64.iso`
- **Arquitetura**: 64 bits (`amd64`)
- **Tipo**: instalador completo do Kali Linux
- **Origem**: site oficial do projeto Kali Linux

Optou-se pelo instalador completo do Kali Linux por garantir que as principais ferramentas de segurança já estejam disponíveis desde o primeiro boot, reduzindo a dependência de conexão com a Internet durante a instalação e aumentando a previsibilidade do processo. O uso da versão *netinst* é indicado apenas quando se deseja uma instalação mais enxuta ou quando há necessidade de baixar seletivamente os pacotes a partir dos repositórios.

Uso em máquina virtual (VirtualBox)

No caso de uso como VM (máquina virtual):

1. Realizar o download da ISO `kali-linux-2025.3-installer-amd64.iso` a partir do site oficial do Kali.
2. Salvar a imagem em uma pasta de fácil acesso (por exemplo, a mesma pasta utilizada para as ISOs do projeto).
3. Durante a criação da VM atacante (Seção de criação das VMs), selecionar essa ISO no campo **ISO Image** da tela *New Virtual Machine*.
4. Certificar-se de que a VM está configurada para iniciar a partir da mídia óptica virtual (CD/DVD) para que o instalador do Kali seja carregado no primeiro boot.

Uso em máquina física (pendrive de instalação)

Para instalação do Kali Linux em máquina física, é necessário criar um pendrive de boot:

1. Separar um **pendrive USB** com, no mínimo, **8 GB** de capacidade (todos os dados existentes serão apagados).
2. Em um computador com Windows ou Linux, utilizar uma ferramenta de criação de mídia inicializável, como Rufus, Balena Etcher, Ventoy ou equivalente.
3. Selecionar:
   - A unidade correspondente ao pendrive.
   - A imagem `kali-linux-2025.3-installer-amd64.iso` como fonte.
   - O esquema de partição e alvo (por exemplo, `GPT + UEFI` ou `MBR + BIOS`), conforme o hardware.
4. Iniciar o processo de gravação e aguardar até a conclusão.
5. No equipamento onde o Kali será instalado, conectar o pendrive, ajustar a ordem de boot na BIOS/UEFI para inicializar pelo USB e reiniciar.

4.2.2 Instalação Passo a Passo

A seguir são descritos os passos da instalação do Kali Linux para uso como máquina atacante, contemplando todo o fluxo do instalador desde a inicialização até a conclusão do processo. As imagens são apresentadas apenas nas etapas em que a visualização contribui para melhor compreensão ou evita ambiguidades na configuração.

1. **Tela inicial do instalador**

   Inicie a máquina (virtual ou física) pelo instalador do Kali Linux. Na tela inicial serão exibidas opções como **Graphical install**, **Install**, **Advanced options**, entre outras.

   Para a máquina atacante, recomenda-se utilizar a opção **Graphical install** (Figura 4.5), pois o Kali será utilizado com interface gráfica e múltiplas ferramentas visuais.

Figura 4.5: Tela inicial do instalador Kali Linux: seleção da opção **Graphical install**.

2. **Idioma, localidade e teclado**

   Selecionar o idioma **Portuguese (Brazil)**, a localidade **Brasil** e o layout de teclado **Português Brasileiro**, mantendo coerência com a configuração utilizada no servidor Debian. Isso facilita o uso simultâneo das máquinas no laboratório.

3. **Configuração de rede**

   Assim como no Debian, o comportamento varia conforme o tipo de ambiente:

   - **Máquina virtual (VirtualBox)**: a interface de rede virtual é detectada automaticamente, e a configuração prossegue normalmente via DHCP.
   - **Máquina física**: o instalador pode exibir opções para escolher entre interface Ethernet (cabeada) ou Wi-Fi. Em laboratório, recomenda-se priorizar Ethernet pela estabilidade.

   Neste momento, uma configuração automática (DHCP) é suficiente; configurações detalhadas de rede serão ajustadas posteriormente na fase de configuração do laboratório.

4. **Nome da máquina e domínio**

   Definir um **hostname** coerente com o papel de máquina atacante, por exemplo:

   - `kali`, `vm2-attacker` ou `kali-attacker`.

   O campo de **Nome de domínio** pode ser deixado em branco ou preenchido com o domínio interno do laboratório, como `lab.local`, se desejado.

5. **Criação de usuário e senha**

   O instalador do Kali solicitará a criação de um usuário padrão (Kali, nas versões recentes, não utiliza mais `root` como usuário de login por padrão). Para padronizar o ambiente de testes, pode-se adotar:

- **Nome completo**: `attacker01`
- **Nome de usuário**: `attacker`
- **Senha**: definir uma senha forte e repeti-la quando solicitado.

Este usuário será utilizado para a execução dos atos de ataque, execução de ferramentas de segurança e conexão SSH às demais máquinas do laboratório.

6. **Configuração do relógio do sistema**

   Selecionar a região correspondente ao laboratório, garantindo consistência temporal entre logs das diferentes máquinas. Para este projeto:

   - **Fuso horário**: `America/Campo_Grande` (Mato Grosso do Sul).

   Essa unificação de fuso horário facilita a correlação de eventos entre o atacante, o servidor SSH e a máquina de monitoramento.

7. **Particionamento de disco**

   Para a máquina atacante, também é adequado utilizar o particionamento assistido:

   - Selecionar **Assistido – usar o disco inteiro**.
   - Em VM: escolher o disco virtual criado para a VM do Kali.
   - Em máquina física: selecionar o disco exclusivo para o Kali (atenção para não sobrescrever outros sistemas).
   - Utilizar o esquema **Todos os arquivos em uma única partição**, suficiente para os testes previstos.
   - Confirmar a gravação das alterações no disco para prosseguir com a instalação.

8. **Seleção de ambiente e softwares**

   O instalador do Kali permite escolher perfis de instalação e ambientes gráficos. Como a máquina atacante será utilizada intensivamente com ferramentas gráficas, recomenda-se manter o ambiente padrão sugerido pelo Kali (geralmente **KDE** ou **Xfce**, dependendo da ISO), desde que atenda ao limite de recursos definido na Tabela 3.1.

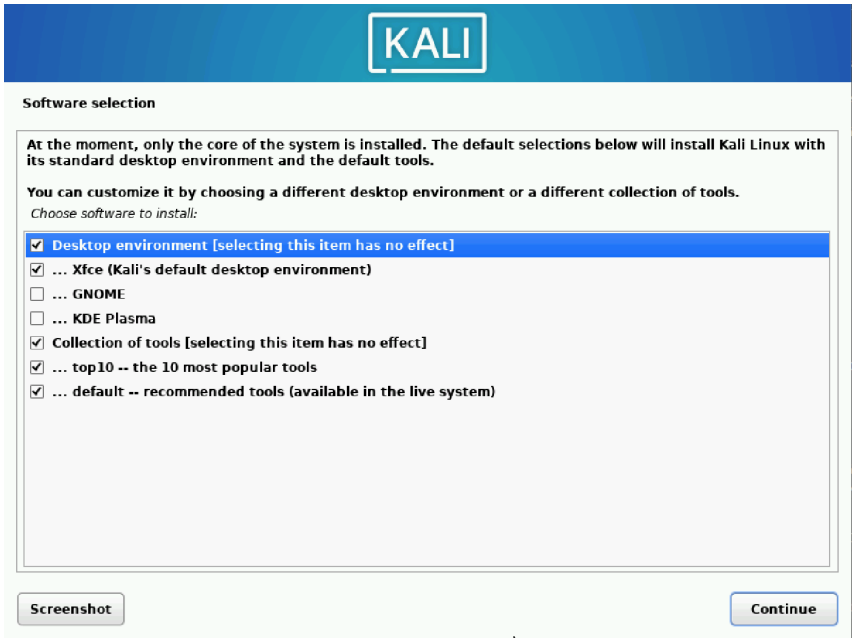

Figura 4.6: Seleção de ambiente e tarefas no instalador do Kali Linux.

Para este manual, recomenda-se seguir as opções exibidas na Figura 4.6:

- Manter o **ambiente gráfico padrão** do Kali.
- Manter as **ferramentas padrão** selecionadas pelo instalador (perfil padrão do Kali), suficientes para execução dos atos planejados.

9. **Instalação do carregador de boot (GRUB)**

   Confirmar a instalação do GRUB no disco principal:

   - Em VM: geralmente o disco virtual utilizado pelo Kali.
   - Em máquina física: o disco onde o Kali foi instalado (por exemplo, `/dev/sda`).

10. **Conclusão da instalação**

    Após a cópia dos arquivos e configuração dos pacotes, o instalador exibirá uma mensagem indicando que a instalação foi concluída e solicitará a remoção da mídia de instalação (ISO ou pendrive), seguida de reinicialização (Figura 4.7).

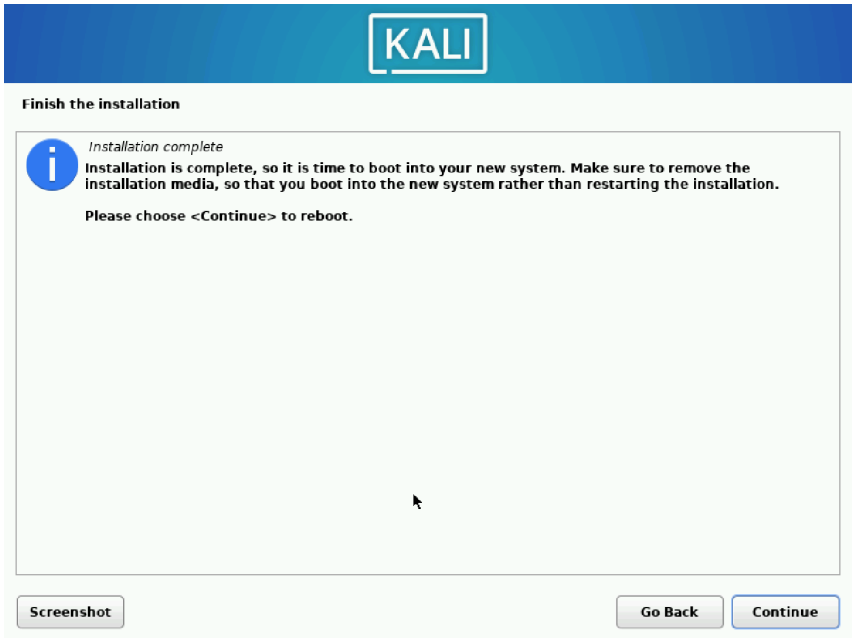

Figura 4.7: Conclusão da instalação do Kali Linux para a máquina atacante.

Confirmar a reinicialização para prosseguir com o primeiro boot do Kali Linux como máquina atacante do laboratório.

4.3 Kali Purple (Monitoramento)

A máquina de monitoramento do laboratório foi baseada no **Kali Purple**, distribuição especializada em operações defensivas, monitoramento contínuo e análise de eventos de segurança, projetada para atuação em ambientes de SOC (*Security Operations Center*). Seu papel é central na captura, inspeção e correlação do tráfego SSH gerado durante os testes envolvendo criptografia clássica e híbrida pós-quântica.

O processo de instalação do Kali Purple é, em sua maior parte, idêntico ao da máquina atacante (Kali Linux), diferindo principalmente na seleção de softwares e no conjunto de ferramentas instaladas por padrão. Dessa forma, nesta seção são destacados apenas os pontos específicos e diferenciadores do Kali Purple.

4.3.1 Preparação da ISO

Para a máquina de monitoramento foi utilizada a seguinte imagem oficial:

- **Arquivo ISO**: `kali-purple-2025.3-installer-amd64.iso`
- **Arquitetura**: 64 bits (`amd64`)
- **Tipo**: instalador completo
- **Origem**: site oficial do Kali Linux

Optou-se pelo instalador completo devido à necessidade de disponibilizar, desde o primeiro boot, ferramentas essenciais de monitoramento e análise, reduzindo dependências externas e garantindo maior previsibilidade do ambiente.

Em ambiente virtual, a ISO foi montada diretamente na máquina virtual. Em máquinas físicas, a ISO deve ser gravada em um pendrive utilizando ferramentas como *Rufus*, *Balena Etcher* ou *Ventoy*, respeitando o modo de boot compatível com o hardware (BIOS ou UEFI).

4.3.2 Instalação Passo a Passo

As seguintes etapas seguem exatamente o mesmo fluxo descrito para o Kali Linux Atacante:

- Inicialização do instalador em modo *Install*.
- Seleção de idioma, localidade e layout de teclado.
- Configuração básica de rede via DHCP.
- Definição de hostname, usuário e senha.
- Configuração de fuso horário.
- Particionamento de disco em modo assistido.
- Instalação do carregador de inicialização GRUB.

A principal diferença ocorre na etapa de seleção de softwares, descrita a seguir.

Seleção de softwares (Kali Purple)

Na etapa *Software selection*, o instalador do Kali Purple apresenta opções estruturadas para ambientes de defesa, organizadas conforme os domínios do **NIST Cybersecurity Framework (CSF)**, permitindo montar um ambiente completo de monitoramento e resposta a incidentes.

Neste laboratório, foram selecionadas as opções exibidas na Figura 4.8:

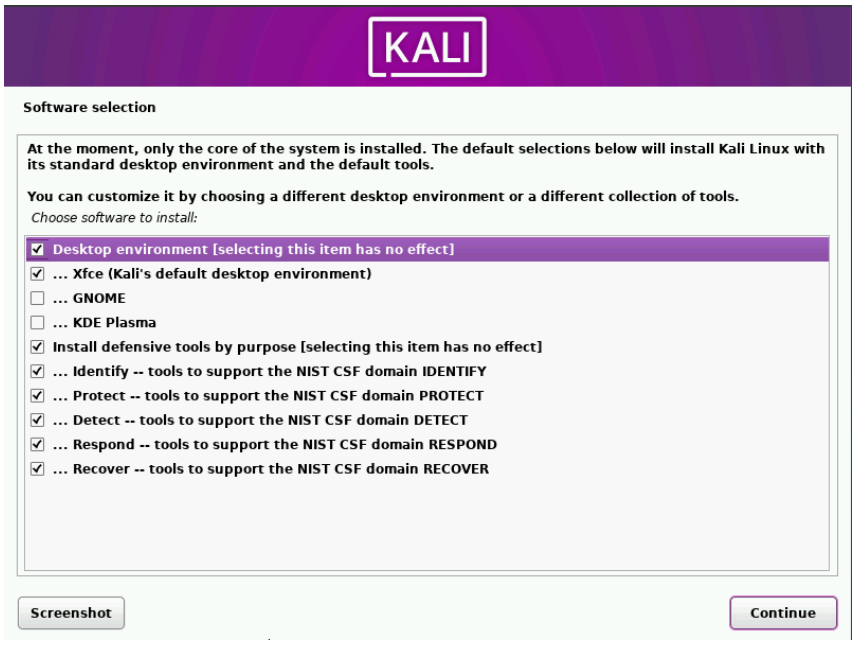

Figura 4.8: Tela de seleção de softwares no Kali Purple, com as ferramentas defensivas organizadas conforme os domínios do NIST CSF.

- **Desktop environment**
  - `Xfce (Kali's default desktop environment)`

  O ambiente XFCE foi escolhido por apresentar baixo consumo de recursos, maior estabilidade e melhor desempenho contínuo, características fundamentais para uma máquina de monitoramento que opera em regime permanente.

- **Install defensive tools by purpose**
  - **Identify** — ferramentas voltadas à identificação de ativos, reconhecimento de superfícies monitoradas e inventário de recursos.
  - **Protect** — ferramentas voltadas à aplicação de controles preventivos e endurecimento do sistema.
  - **Detect** — ferramentas responsáveis pela detecção de eventos suspeitos, inspeção profunda de pacotes e análise de tráfego.
  - **Respond** — ferramentas destinadas à resposta a incidentes, contenção e análise inicial.
  - **Recover** — ferramentas relacionadas à recuperação e restauração após incidentes.

Essa seleção garante a instalação dos principais componentes do ambiente SOC, incluindo sensores de rede, sistemas de detecção, agentes de coleta de logs e mecanismos de análise, formando a base necessária para o monitoramento das conexões SSH do laboratório.

Parte III

Configuração Específica das Máquinas

Capítulo 5

Servidor SSH Real (Debian)

5.1 Pós-instalação

5.1.1 Instalação e configuração do *sudo*

A instalação e a configuração do utilitário *sudo* são necessárias para permitir que o usuário padrão execute comandos administrativos de forma controlada, sem a necessidade de permanecer logado como *root*. O procedimento descrito a seguir deve ser realizado imediatamente após a instalação do sistema Debian.

1. **Acessar o modo *root***

   Inicialmente, é necessário alternar para o usuário *root*, que possui privilégios administrativos completos no sistema.

   ```
   su -
   ```

   - `su -`: altera a sessão atual para o usuário *root*, carregando o ambiente completo desse usuário (incluindo variáveis de ambiente e caminho padrão). A senha do usuário *root* foi definida durante a instalação do sistema operacional.

2. **Atualizar a lista de pacotes e instalar o *sudo***

   Em seguida, atualiza-se a lista de pacotes disponíveis nos repositórios e realiza-se a instalação do pacote *sudo*:

   ```
   apt update
   apt install sudo -y
   ```

   - `apt update`: atualiza a lista de pacotes disponíveis nos repositórios configurados, garantindo que o sistema reconheça as versões mais recentes dos pacotes.
   - `apt install sudo -y`: instala o pacote *sudo*.
     - `-y`: aceita automaticamente as confirmações padrão durante o processo de instalação, evitando a necessidade de interação manual.

3. **Adicionar o usuário ao grupo *sudo***

   Após a instalação do *sudo*, o usuário que realizará a administração do sistema deve ser incluído no grupo `sudo`. No contexto deste ambiente, o usuário configurado recebe o nome `sshserver`.

```
usermod -aG sudo sshserver
```

- `usermod -aG sudo sshserver`: altera a configuração do usuário `sshserver`, adicionando-o ao grupo `sudo`.
  - `-aG`: adiciona (*append*) o usuário aos grupos informados, sem removê-lo dos grupos aos quais ele já pertence.
  - `sudo`: nome do grupo ao qual o usuário será adicionado para obter permissão de uso do *sudo*.
  - `sshserver`: conta de usuário que será habilitada para uso do *sudo*.

4. **Encerrar as sessões atuais**

   Para que as alterações de grupos de usuários tenham efeito, é necessário encerrar as sessões em andamento e realizar um novo login com o usuário configurado.

```
exit
exit
```

- `exit`: no primeiro uso, encerra a sessão do usuário *root*, retornando à sessão anterior.
- `exit`: no segundo uso, encerra a sessão de terminal restante, retornando ao gerenciador de login.

5. **Validar o funcionamento do *sudo***

   Após autenticar-se novamente como o usuário `sshserver`, recomenda-se testar o funcionamento do *sudo* executando um comando simples de atualização da lista de pacotes:

```
sudo apt update
```

- `sudo`: executa o comando subsequente com privilégios elevados, solicitando a senha do usuário `sshserver` para autorização.
- `apt update`: atualiza a lista de pacotes disponíveis, permitindo verificar se o *sudo* está funcionando corretamente.

Se o comando for executado sem erros de permissão, conclui-se que a configuração do *sudo* foi realizada com sucesso.

5.2 Preparação do Ambiente e Dependências

Esta etapa prepara o sistema para a compilação dos componentes criptográficos, instalando as ferramentas de desenvolvimento necessárias, definindo um diretório de trabalho e registrando informações básicas do ambiente.

1. **Instalar dependências de compilação**

   Os comandos a seguir atualizam a lista de pacotes e instalam o conjunto de ferramentas e bibliotecas necessárias. Caso o usuário esteja autenticado como *root*, o prefixo `sudo` pode ser omitido.

```
sudo apt update
sudo apt install -y build-essential git cmake ninja-build pkg-config
    autoconf automake libtool zlib1g-dev libgmp-dev liblz4-dev python3-venv
     python3-openssl libpam0g-dev libwrap0-dev libsystemd-dev libaudit-dev
    libseccomp-dev wget ca-certificates libssl-dev
```

- `sudo apt update`: atualiza a lista de pacotes disponíveis nos repositórios configurados, garantindo que o sistema utilize metadados recentes para instalação e atualização.
- `sudo apt install -y ...`: instala, em um único comando, o conjunto de ferramentas de desenvolvimento e bibliotecas necessárias para compilar o OpenSSL, o OpenSSH e demais componentes.
  - `-y`: aceita automaticamente as confirmações padrão do gerenciador de pacotes, permitindo a instalação não interativa.
  - `build-essential`: metapacote que instala o compilador `gcc`, `g++`, `make` e outras ferramentas básicas de compilação.
  - `git`: sistema de controle de versão distribuído, utilizado para obter e gerenciar código-fonte.
  - `cmake`: gerador de sistemas de *build* portátil, utilizado por diversos projetos modernos em C/C++.
  - `ninja-build`: ferramenta de *build* de alto desempenho, frequentemente utilizada em conjunto com o `cmake`.
  - `pkg-config`: utilitário que auxilia na descoberta de bibliotecas instaladas, fornecendo *flags* de compilação e ligação.
  - `autoconf`: ferramenta para geração de scripts `configure`, utilizada em projetos que seguem o padrão *Autotools*.
  - `automake`: gera arquivos `Makefile.in` compatíveis com o *Autotools*, facilitando a portabilidade do processo de compilação.
  - `libtool`: auxilia na criação e no uso de bibliotecas compartilhadas de forma portátil entre diferentes sistemas.
  - `zlib1g-dev`: arquivos de desenvolvimento da biblioteca de compressão `zlib`, utilizada por diversos componentes de rede e criptografia.
  - `libgmp-dev`: biblioteca de aritmética de precisão múltipla (*GNU MP*), empregada em operações matemáticas intensivas, como algumas rotinas criptográficas.
  - `liblz4-dev`: arquivos de desenvolvimento da biblioteca de compressão `LZ4`, usada em rotinas de compressão de alto desempenho.
  - `python3-venv`: suporte para criação de ambientes virtuais Python 3, útil para isolar dependências de ferramentas auxiliares.
  - `python3-openssl`: módulo Python que permite utilizar funcionalidades da biblioteca OpenSSL a partir de scripts Python.
  - `libpam0g-dev`: arquivos de desenvolvimento da biblioteca PAM (*Pluggable Authentication Modules*), necessária para integrar o OpenSSH com mecanismos de autenticação do sistema.

- `libwrap0-dev`: arquivos de desenvolvimento da biblioteca *TCP Wrappers*, utilizada para controle de acesso baseado em endereço de rede.
- `libsystemd-dev`: interfaces de desenvolvimento para integração com o *systemd*, permitindo que serviços reportem estado e usem facilidades do gerenciador de serviços.
- `libaudit-dev`: biblioteca de auditoria do Linux, utilizada para registrar eventos de segurança e acessos ao sistema.
- `libseccomp-dev`: biblioteca de *sandboxing* baseada em *seccomp*, empregada para restringir chamadas de sistema realizadas por processos (por exemplo, o servidor SSH).
- `wget`: utilitário de linha de comando para download de arquivos via HTTP, HTTPS e FTP.
- `ca-certificates`: conjunto de certificados de autoridades certificadoras confiáveis, necessário para validar conexões HTTPS (por exemplo, ao usar `wget`).
- `libssl-dev`: arquivos de desenvolvimento da OpenSSL fornecida pelo repositório da distribuição, utilizados por algumas ferramentas do sistema.

Observação: o pacote `libssl-dev` instalado pelos repositórios da distribuição é distinto da versão de OpenSSL que será compilada manualmente neste manual. Para os *builds* relacionados ao experimento, será indicado explicitamente o caminho da OpenSSL compilada.

2. **Criar o diretório de trabalho**

   Em seguida, é criado um diretório dedicado para os códigos-fonte e artefatos de compilação, de forma a organizar todos os arquivos do experimento em um único local.

```
mkdir -p ~/build-oqs && cd ~/build-oqs
mkdir -p artifacts
```

   - `mkdir -p ~/build-oqs`: cria o diretório `~/build-oqs` no diretório pessoal do usuário. A opção `-p` garante que o comando não produza erro caso o diretório já exista.
   - `cd ~/build-oqs`: entra no diretório de trabalho recém-criado. O operador `&&` faz com que o comando `cd` seja executado apenas se o `mkdir` tiver sido bem-sucedido.
   - `mkdir -p artifacts`: cria o subdiretório `artifacts`, que será utilizado para armazenar arquivos de registro (*logs*) e metadados relacionados ao processo de compilação.

3. **Registrar informações básicas do ambiente**

   Para fins de reprodutibilidade e documentação, recomenda-se registrar a data/hora e informações básicas do sistema em um arquivo de texto dentro do diretório `artifacts`.

```
date > artifacts/build-info.txt
uname -a >> artifacts/build-info.txt
```

- `date > artifacts/build-info.txt`: grava a data e hora atuais no arquivo `build-info.txt`. O operador `>` cria o arquivo (caso não exista) ou sobrescreve seu conteúdo.
- `uname -a » artifacts/build-info.txt`: acrescenta ao mesmo arquivo informações detalhadas sobre o sistema operacional e o kernel em uso. O operador `»` adiciona o texto ao final do arquivo sem apagar o conteúdo previamente registrado.

5.3 OpenSSL Personalizado

5.3.1 Download, compilação e instalação do OpenSSL 3.x

Nesta etapa é realizada a obtenção do código-fonte do OpenSSL 3.x, sua compilação e instalação em `/usr/local/openssl`, além da configuração das variáveis de ambiente necessárias para que essa versão personalizada seja corretamente localizada pelo sistema.

1. **Definir a versão, baixar o *tarball* e registrar o *hash***

```
OPENSSL_VER="3.5.1"
```

- Define, em uma variável de ambiente, a versão do OpenSSL que será utilizada nos comandos subsequentes, facilitando eventual troca de versão sem alterar todas as linhas manualmente.

```
wget https://www.openssl.org/source/openssl-$OPENSSL_VER.tar.gz -O
    artifacts/openssl-$OPENSSL_VER.tar.gz
```

- Utiliza o utilitário `wget` para fazer o download do *tarball* oficial do OpenSSL na versão indicada pela variável `OPENSSL_VER`, salvando o arquivo no diretório `artifacts` com um nome padronizado.

```
sha256sum artifacts/openssl-$OPENSSL_VER.tar.gz > artifacts/openssl-
    $OPENSSL_VER.sha256
```

- Usa o comando `sha256sum` para calcular o *hash* `SHA-256` do arquivo baixado, registrando o valor em um arquivo de texto no diretório `artifacts`, o que permite verificar a integridade do download posteriormente.

2. **Descompactar, configurar, compilar e instalar o OpenSSL**

```
tar xzf artifacts/openssl-$OPENSSL_VER.tar.gz
```

- Descompacta o *tarball* do OpenSSL, criando um diretório chamado `openssl-\$OPENSSL\_VER` com o código-fonte.

```
cd openssl-$OPENSSL_VER
```

- Entra no diretório do código-fonte correspondente à versão definida na variável `OPENSSL_VER`.

```
./Configure --prefix=/usr/local/openssl --openssldir=/usr/local/openssl
   shared linux-x86_64
```

- Executa o script `Configure` do OpenSSL, definindo os parâmetros de compilação e instalação.
- Opção `-prefix=/usr/local/openssl`: define o diretório onde os binários e bibliotecas da versão personalizada serão instalados.
- Opção `-openssldir=/usr/local/openssl`: define o diretório base para arquivos de configuração e demais dados do OpenSSL.
- Palavra-chave `shared`: determina que as bibliotecas sejam geradas na forma de bibliotecas compartilhadas (*shared libraries*).
- `linux-x86_64`: especifica o alvo de compilação para sistemas Linux de 64 bits na arquitetura x86_64.

```
make -j$(nproc) || make
```

- Compila o OpenSSL utilizando o comando `make`. A opção `-j$(nproc)` faz com que a compilação use todos os núcleos de CPU disponíveis, acelerando o processo.
- O trecho `|| make` garante uma recompilação sequencial simples caso a compilação paralela apresente algum erro.

```
sudo make install_sw
```

- Instala o software compilado (binários e bibliotecas) no diretório definido em `-prefix`. O alvo `install_sw` instala apenas os componentes de software principais, sem, por exemplo, a documentação completa.

```
 /usr/local/openssl/bin/openssl version | tee -a ../artifacts/build-info.
    txt
```

- Executa a versão recém-instalada do `openssl` para verificar a versão efetivamente em uso.

- O uso de `tee -a` acrescenta a saída do comando ao arquivo `../artifacts/build-info.txt`, registrando a versão compilada como parte das informações de ambiente.

Em alguns sistemas, ao utilizar essa nova instalação, podem surgir erros informando que determinadas bibliotecas não foram encontradas (*not found*). Isso é comum em ambientes em que bibliotecas de 64 bits são instaladas em diretórios como `lib64`. Nesses casos, é necessário ajustar as variáveis de ambiente, conforme descrito nas etapas seguintes.

3. **Configurar variáveis de ambiente para localizar a OpenSSL personalizada**

```
export PATH=/usr/local/openssl/bin:$PATH
```

- Adiciona o diretório de binários da OpenSSL personalizada ao início da variável `PATH`, fazendo com que essa versão seja encontrada antes da OpenSSL padrão do sistema.

```
export LD_LIBRARY_PATH=/usr/local/openssl/lib64:$LD_LIBRARY_PATH
```

- Informa ao carregador dinâmico de bibliotecas que o diretório `/usr/local/openssl/lib64` deve ser considerado ao procurar bibliotecas compartilhadas, garantindo que as bibliotecas recém-instaladas sejam localizadas.

```
export PKG_CONFIG_PATH=/usr/local/openssl/lib64/pkgconfig:$PKG_CONFIG_PATH
```

- Ajusta o caminho utilizado pelo `pkg-config` para localizar os arquivos `.pc` da OpenSSL personalizada, permitindo que outros projetos encontrem essa biblioteca durante seus próprios processos de compilação.

```
export CPPFLAGS="-I/usr/local/openssl/include"
```

- Acrescenta o diretório de cabeçalhos (`include`) da OpenSSL personalizada à lista de caminhos utilizados pelo compilador C/C++ ao procurar arquivos de cabeçalho.

```
export LDFLAGS="-L/usr/local/openssl/lib64"
```

- Informa ao vinculador (*linker*) o diretório onde estão as bibliotecas da OpenSSL personalizada, para que possam ser ligadas corretamente durante a compilação de outros programas.

Para conferir se as variáveis foram definidas corretamente na sessão atual, recomenda-se verificar seus valores:

```
echo $LD_LIBRARY_PATH
echo $PKG_CONFIG_PATH
echo $LDFLAGS
```

- Cada comando `echo` exibe o valor atual da variável correspondente, permitindo confirmar se os diretórios desejados foram incluídos.

4. **Validar o acesso à OpenSSL personalizada**

```
which openssl
```

- Mostra o caminho completo do executável `openssl` que será utilizado quando o comando for chamado na linha de comando. O resultado esperado é: `/usr/local/openssl/bin/openssl` ou equivalente.

```
 /usr/local/openssl/bin/openssl version -a
```

- Exibe informações detalhadas sobre a versão do OpenSSL em uso, incluindo opções de compilação, diretórios e módulos habilitados, confirmando que a instância personalizada está ativa.

5. **Persistir as variáveis de ambiente**

As variáveis definidas com `export` são válidas apenas para a sessão atual e serão descartadas após logout ou reinicialização. Para torná-las persistentes, é necessário adicioná-las a um arquivo de configuração do *shell*, como `~/.profile`.

```
echo 'export PATH=/usr/local/openssl/bin:$PATH' >> ~/.profile
echo 'export LD_LIBRARY_PATH=/usr/local/openssl/lib64' >> ~/.profile
echo 'export PKG_CONFIG_PATH=/usr/local/openssl/lib64/pkgconfig' >> ~/.
   profile
echo 'export CPPFLAGS="-I/usr/local/openssl/include"' >> ~/.profile
echo 'export LDFLAGS="-L/usr/local/openssl/lib64"' >> ~/.profile
```

- Cada comando `echo '...' >> ~/.profile` adiciona uma linha ao final do arquivo `~/.profile`, de forma que as variáveis sejam configuradas automaticamente nas próximas sessões de login.

Para aplicar imediatamente essas definições na sessão atual, sem reiniciar o sistema, pode-se recarregar o arquivo:

```
source ~/.profile
```

- Faz com que o *shell* leia novamente o arquivo `~/.profile`, aplicando as variáveis de ambiente recém adicionadas.

5.4 Biblioteca *liboqs*

5.4.1 Compilação e instalação da *liboqs*

A biblioteca *liboqs*, mantida pelo projeto Open Quantum Safe, fornece implementações de algoritmos criptográficos resistentes a computadores quânticos. Nesta etapa, ela será obtida a partir do repositório oficial, compilada com *CMake*/*Ninja* e instalada em `/usr/local/liboqs`.

1. **Acessar o diretório de trabalho**

```
cd ~/build-oqs
```

   - Entra no diretório de trabalho previamente criado para agrupar todos os componentes compilados deste ambiente.

2. **Clonar o repositório oficial da *liboqs***

```
git clone --depth 1 https://github.com/open-quantum-safe/liboqs.git
```

   - Utiliza o `git` para clonar o repositório oficial da *liboqs*, hospedado no GitHub.
   - A opção `-depth 1` realiza um *clone* raso, trazendo apenas o *commit* mais recente, o que reduz o volume de dados transferidos e acelera o procedimento.

3. **Entrar no diretório do código-fonte**

```
cd liboqs
```

   - Acessa o diretório `liboqs` recém-criado pelo comando de *clone*, onde se encontram os arquivos de código-fonte da biblioteca.

4. **Registrar o *commit* exato utilizado**

```
git rev-parse HEAD > ../artifacts/liboqs-commit.txt
```

   - Obtém, com `git rev-parse HEAD`, o identificador completo (*hash*) do *commit* atualmente checado.
   - Grava esse identificador no arquivo `../artifacts/liboqs-commit.txt`, permitindo documentar com precisão qual versão do código foi utilizada na compilação.

5. **Criar diretório de *build* e acessá-lo**

```
mkdir -p build && cd build
```

- `mkdir -p build`: cria o diretório `build` para realizar uma compilação fora da árvore principal de código (*out-of-source build*). A opção `-p` evita erro caso o diretório já exista.
- `cd build`: entra no diretório de *build* recém-criado.

6. **Configurar o projeto com *CMake* e gerador *Ninja***

```
cmake -GNinja -DCMAKE_INSTALL_PREFIX=/usr/local/liboqs ..
```

- Executa o `cmake` para gerar os arquivos de *build* necessários à compilação da *liboqs*.
- Opção `-GNinja`: seleciona o *Ninja* como gerador de *build*, conforme recomendado pela documentação do projeto.
- Opção `-DCMAKE_INSTALL_PREFIX=/usr/local/liboqs`: define o diretório de instalação da biblioteca, que será utilizado posteriormente pelos comandos de *install*.
- O argumento `..` indica que o arquivo `CMakeLists.txt` se encontra no diretório imediatamente acima do diretório de *build*.

7. **Compilar a *liboqs* com *Ninja***

```
ninja -j$(nproc) || ninja
```

- Utiliza o `ninja` para compilar a biblioteca. A opção `-j$(nproc)` faz com que a compilação explore todos os núcleos de CPU disponíveis, reduzindo o tempo de *build*.
- O trecho `|| ninja` garante uma tentativa alternativa de compilação sequencial, caso a compilação paralela apresente algum erro.

8. **Instalar a *liboqs* no sistema**

```
sudo ninja install
```

- Executa a etapa de instalação definida pelo `CMake`, copiando bibliotecas e arquivos associados para o diretório especificado em `CMAKE_INSTALL_PREFIX` (`/usr/local/liboqs`), operação que requer privilégios administrativos.

9. **Verificar a presença das bibliotecas instaladas**

```
ls /usr/local/liboqs/lib
```

- Lista o conteúdo do diretório `/usr/local/liboqs/lib`, permitindo confirmar a presença das bibliotecas compartilhadas e demais arquivos produzidos pela instalação da *liboqs*.

10. **Registrar versões e *commit* utilizado**

    Para documentar o ambiente de compilação e possibilitar reprodutibilidade, recomenda-se registrar explicitamente o *commit* da *liboqs* utilizado:

```
cat ../artifacts/liboqs-commit.txt
```

    - Exibe o identificador de *commit* armazenado em `../artifacts/liboqs-commit.txt`, que pode ser anotado ou citado na documentação técnica do experimento.

Conforme a documentação oficial, a *liboqs* adota *CMake* e *Ninja* como ferramentas recomendadas de compilação. Em 2025, o site oficial do projeto indicava a versão `0.14.0` como *release* estável, o que justifica a escolha dessa cadeia de ferramentas neste ambiente.

5.5 Provider *oqs-provider* para o OpenSSL 3

5.5.1 Compilação e instalação do *oqs-provider*

O *oqs-provider* é um *provider* do OpenSSL 3 desenvolvido pelo projeto Open Quantum Safe, permitindo utilizar algoritmos de criptografia pós-quântica via interface de *providers* do OpenSSL. Nesta etapa, o código será obtido a partir do repositório oficial, compilado e instalado de forma compatível com a *liboqs* e o OpenSSL personalizados configurados nas seções anteriores.

1. **Acessar o diretório de trabalho**

```
cd ~/build-oqs
```

    - Retorna ao diretório de trabalho `~/build-oqs`, onde estão organizados os componentes compilados do ambiente criptográfico (OpenSSL e *liboqs*).

2. **Clonar o repositório do *oqs-provider***

```
git clone --depth 1 https://github.com/open-quantum-safe/oqs-provider.git
```

    - Clona, via `git`, o repositório oficial do *oqs-provider* mantido pelo projeto Open Quantum Safe.
    - A opção `-depth 1` realiza um *clone* raso, trazendo apenas o *commit* mais recente, o que reduz o volume de dados transferidos.

3. **Entrar no diretório do código-fonte**

```
cd oqs-provider
```

    - Acessa o diretório `oqs-provider`, criado pelo comando de *clone*, onde se encontram os arquivos de código-fonte do *provider*.

4. **Registrar o *commit* exato utilizado**

```
git rev-parse HEAD > ../artifacts/oqs-provider-commit.txt
```

- Usa `git rev-parse HEAD` para obter o identificador completo (*hash*) do *commit* atualmente checado.
- Grava esse identificador em `../artifacts/oqs-provider-commit.txt`, permitindo documentar com precisão qual versão do código foi utilizada na compilação.

5. **Configurar o projeto com *CMake***

```
cmake -S . -B _build \
  -DOPENSSL_ROOT_DIR=/usr/local/openssl \
  -Dliboqs_DIR=/usr/local/liboqs/lib/cmake/liboqs \
  -DCMAKE_INSTALL_PREFIX=/usr/local/oqs-provider
```

- Executa o `cmake` apontando a árvore de código-fonte atual (`-S .`) e definindo o diretório de *build* como `\_build` (`-B _build`), seguindo a prática de *out-of-source build*.
- Opção `-DOPENSSL_ROOT_DIR=/usr/local/openssl`: informa ao `cmake` o caminho da instalação personalizada do OpenSSL, garantindo que o *provider* seja compilado contra essa instância específica.
- Opção `-Dliboqs_DIR=/usr/local/liboqs/lib/cmake/liboqs`: aponta para os arquivos de configuração `CMake` instalados pela *liboqs*, permitindo que o `cmake` localize corretamente essa biblioteca.
- Opção `-DCMAKE_INSTALL_PREFIX=/usr/local/oqs-provider`: define o diretório-base em que os artefatos do *oqs-provider* serão instalados.

6. **Compilar o *oqs-provider***

```
cmake --build _build -j$(nproc)
```

- Inicia a compilação a partir da árvore de *build* `\_build`, utilizando todos os núcleos de CPU disponíveis (`-j$(nproc)`), o que reduz o tempo de compilação.

7. **Instalar o *oqs-provider* no sistema**

```
sudo cmake --install _build
```

- Executa a etapa de instalação definida pelo projeto, copiando os módulos de *provider* e arquivos auxiliares para os diretórios apropriados, conforme `CMAKE_INSTALL_PREFIX` e a configuração do OpenSSL.

Após a instalação, espera-se que o módulo do *provider* (por exemplo, `oqsprovider.so`) seja instalado em um diretório similar a `/usr/local/openssl/lib64/ossl-modules/`, que é o local padrão de módulos do OpenSSL 3 em sistemas de 64 bits.

8. **Verificar *providers* disponíveis no OpenSSL personalizado**

```
/usr/local/openssl/bin/openssl list -providers |
tee -a ~/build-oqs/artifacts/openssl-providers.txt
```

   - Utiliza o executável do OpenSSL personalizado para listar todos os *providers* reconhecidos pela instalação atual.
   - O uso de `tee -a` registra a saída completa do comando no arquivo `~/build-oqs/artifacts/openssl-providers.txt`, acrescentando (`-a`) as informações ao final do arquivo.
   - O resultado esperado é que, além do *provider* `default`, apareça um *provider* com nome semelhante a `oqs-provider` (ou conforme o repositório escolhido). Caso apenas o `default` seja listado, isso indica que o *provider* não foi corretamente instalado ou não está sendo carregado pela configuração atual do OpenSSL.

5.5.2 Solução de problemas na carga do *oqs-provider*

Se, após a compilação e instalação, o comando

```
/usr/local/openssl/bin/openssl list -providers |
 tee -a ~/build-oqs/artifacts/openssl-providers.txt
```

não listar o *oqs-provider* (ou listar apenas o `default`), é provável que o OpenSSL personalizado não esteja carregando o módulo de *provider* na inicialização. Nesta situação, recomenda-se seguir os passos a seguir.

1. **Garantir a existência do diretório de configuração do OpenSSL customizado**

```
sudo mkdir -p /usr/local/openssl/ssl
sudo nano /usr/local/openssl/ssl/openssl.cnf
```

   - `sudo mkdir -p /usr/local/openssl/ssl`: cria o diretório destinado aos arquivos de configuração do OpenSSL personalizado, caso ainda não exista.
   - `sudo nano /usr/local/openssl/ssl/openssl.cnf`: abre o editor `nano` para criação ou edição do arquivo `/usr/local/openssl/ssl/openssl.cnf`, que será usado como arquivo de configuração principal desse OpenSSL.

2. **Definir configuração mínima para carregar o *oqs-provider***

   No editor, deve-se inserir o conteúdo abaixo no arquivo `openssl.cnf`:

```
openssl_conf = openssl_init

[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
oqsprovider = oqsprovider_sect

[default_sect]
activate = 1

[oqsprovider_sect]
module = /usr/local/openssl/lib64/ossl-modules/oqsprovider.so
activate = 1
```

- A linha `openssl_conf = openssl_init` define a seção inicial de configuração que será utilizada quando o OpenSSL for executado.
- A seção `[openssl_init]` associa a palavra-chave `providers` à seção `[provider_sect]`, na qual são declarados os *providers* disponíveis.
- Em `[provider_sect]`, são definidos o *provider* padrão (`default`) e o *oqsprovider*, cada um apontando para sua respectiva seção de configuração.
- A seção `[default_sect]` com `activate = 1` garante que o *provider* padrão do OpenSSL seja ativado.
- A seção `[oqsprovider_sect]` indica o caminho do módulo `oqsprovider.so` em `/usr/local/openssl/lib64/ossl-modules/oqsprovider.so` e o ativa com `activate = 1`.
- Após inserir o conteúdo, o arquivo deve ser salvo e fechado (por exemplo, em `nano`, utilizando `Ctrl+X`, `Y` e `Enter`).

3. **Apontar o OpenSSL para o arquivo de configuração personalizado**

```
export OPENSSL_CONF=/usr/local/openssl/ssl/openssl.cnf
```

- Define a variável de ambiente `OPENSSL_CONF` para que o executável `/usr/local/openssl/bin/openssl` utilize o arquivo `/usr/local/openssl/ssl/openssl.cnf` como configuração padrão, carregando os *providers* definidos.

4. **Reverificar os *providers* carregados**

```
/usr/local/openssl/bin/openssl list -providers
```

- Lista novamente os *providers* disponíveis na instalação personalizada do OpenSSL. O esperado é que, além do `default`, seja exibido o *provider* correspondente ao módulo instalado (por exemplo, `oqsprovider` ou nome similar).

- Caso o *provider* ainda não apareça, é recomendável verificar se o módulo foi corretamente instalado no diretório indicado, se o caminho em `module = ...` está correto e se a variável `OPENSSL_CONF` está realmente definida na sessão atual.

5.6 OpenSSH Modificado

5.6.1 Download, compilação e instalação do OpenSSH personalizado

Nesta etapa, o OpenSSH é obtido a partir do repositório oficial, recompilado e instalado de forma a utilizar a OpenSSL personalizada configurada em `/usr/local/openssl`. O objetivo é dispor de um cliente e de um servidor SSH que suportem os algoritmos e *providers* definidos nesse ambiente.

1. **Acessar o diretório de trabalho**

```
cd ~/build-oqs
```

   - Retorna ao diretório de trabalho `~/build-oqs`, onde estão organizados os códigos-fonte e artefatos de compilação deste ambiente.

2. **Clonar o repositório do OpenSSH portátil**

```
git clone https://github.com/openssh/openssh-portable.git
```

   - Clona o repositório oficial `openssh-portable`, que contém a versão portável do OpenSSH para sistemas tipo Unix.

3. **Entrar no diretório do código-fonte**

```
cd openssh-portable
```

   - Acessa o diretório `openssh-portable`, criado pelo comando de *clone*, onde se encontram os arquivos de código-fonte do OpenSSH.

4. **Registrar o *commit* exato utilizado**

```
git rev-parse HEAD > ../artifacts/openssh-commit.txt
```

   - Obtém, com `git rev-parse HEAD`, o identificador completo (*hash*) do *commit* atualmente checado.
   - Grava esse identificador em `../artifacts/openssh-commit.txt`, permitindo documentar de forma precisa qual versão do código foi utilizada na compilação.

5. **Preparar o sistema de *build* (*Autotools*)**

```
autoreconf -fi
```

   - Executa `autoreconf` com as opções `-f` (forçar) e `-i` (instalar arquivos auxiliares), regenerando os scripts `configure` e demais arquivos de *Autotools* conforme a configuração atual do sistema.

6. **Configurar o OpenSSH para usar a OpenSSL personalizada**

```
./configure --prefix=/usr/local/openssh \
  --with-ssl-dir=/usr/local/openssl \
  --sysconfdir=/etc/ssh \
  --with-pam --with-md5-passwords CPPFLAGS="$CPPFLAGS" LDFLAGS="$LDFLAGS"
```

   - Executa o script `configure`, gerando os arquivos de *Makefile* de acordo com as opções fornecidas e com as bibliotecas disponíveis no sistema.
   - `-prefix=/usr/local/openssh`: define o diretório de instalação do OpenSSH recompilado (binários, bibliotecas e arquivos auxiliares).
   - `-with-ssl-dir=/usr/local/openssl`: aponta explicitamente para a instalação da OpenSSL personalizada, assegurando que o OpenSSH seja compilado contra essa biblioteca.
   - `-sysconfdir=/etc/ssh`: especifica que os arquivos de configuração do OpenSSH (por exemplo, `sshd_config`) serão armazenados em `/etc/ssh`.
   - `-with-pam`: habilita o suporte a PAM (*Pluggable Authentication Modules*), permitindo integrar a autenticação SSH com os mecanismos do sistema.
   - `-with-md5-passwords`: mantém suporte a hashes de senha baseados em MD5 (útil em ambientes legados onde esse formato ainda está presente).
   - As variáveis de ambiente `CPPFLAGS` e `LDFLAGS` são reaproveitadas na linha de configuração, indicando ao compilador e ao *linker* onde localizar os cabeçalhos e as bibliotecas da OpenSSL personalizada.

   Se o script `configure` não detectar corretamente a OpenSSL compilada, recomenda-se inspecionar o arquivo `config.log` e verificar os valores de `CPPFLAGS`, `LDFLAGS` e `PKG_CONFIG_PATH` definidos nas etapas de preparação do ambiente.

7. **Compilar o OpenSSH**

```
make -j$(nproc) || make
```

   - Compila o OpenSSH utilizando o `make`. A opção `-j$(nproc)` permite compilar em paralelo usando todos os núcleos de CPU disponíveis, reduzindo o tempo de *build*.
   - O trecho `|| make` executa uma compilação sequencial simples caso a compilação paralela apresente algum erro.

8. **Instalar o OpenSSH recompilado**

```
sudo make install
```

   - Instala o OpenSSH recompilado nos diretórios definidos por `-prefix` e demais parâmetros de configuração. Em particular, o servidor SSH (`sshd`) será instalado em `/usr/local/openssh/sbin/sshd` e o cliente SSH (`ssh`) em `/usr/local/openssh/bin/ssh`.

5.6.2 Criação de unidade *systemd* para o OpenSSH personalizado

Após compilar e instalar o OpenSSH em `/usr/local/openssh`, é recomendável criar uma unidade específica do *systemd* para gerenciar o daemon `sshd` recompilado, sem interferir diretamente no serviço SSH padrão da distribuição.

1. **Criar o arquivo de unidade *systemd***

   Deve-se criar o arquivo `/etc/systemd/system/sshd-oqs.service` com o conteúdo a seguir, utilizando o editor de texto de preferência (por exemplo, `nano`, `vim` ou outro):

```
[Unit]
Description=OpenSSH Daemon (oqs  custom build)
After=network.target

[Service]
Environment=LD_LIBRARY_PATH=/usr/local/openssl/lib
ExecStart=/usr/local/openssh/sbin/sshd -D -f /etc/ssh/sshd_config
Restart=on-failure
KillMode=process

[Install]
WantedBy=multi-user.target
```

   - Seção `[Unit]`: define metadados e dependências da unidade.
   - `Description=OpenSSH Daemon (oqs - custom build)`: descrição textual da unidade, indicando que se trata de um `sshd` compilado com suporte a algoritmos pós-quânticos.
   - `After=network.target`: especifica que o serviço deve ser iniciado apenas após a inicialização da pilha de rede do sistema.
   - Seção `[Service]`: define como o serviço será executado.
   - `Environment=LD_LIBRARY_PATH=/usr/local/openssl/lib`: ajusta a variável de ambiente `LD_LIBRARY_PATH` para que o `sshd` encontre as bibliotecas da OpenSSL personalizada.
   - `ExecStart=/usr/local/openssh/sbin/sshd -D -f /etc/ssh/sshd_config`: comando utilizado pelo *systemd* para iniciar o daemon SSH recompilado, em modo de primeiro plano (`-D`) e utilizando o arquivo de configuração `/etc/ssh/sshd_config`.

- `Restart=on-failure`: instrui o *systemd* a reiniciar o serviço automaticamente caso ele termine com falha.
- `KillMode=process`: define que apenas o processo principal do serviço será encerrado quando o *systemd* emitir sinais de parada.
- Seção `[Install]`: define como a unidade será vinculada aos *targets* do sistema.
- `WantedBy=multi-user.target`: indica que o serviço deve estar ativo no nível de execução multiusuário, permitindo habilitá-lo para iniciar automaticamente durante o *boot*.

2. **Recarregar o *systemd* e substituir o serviço SSH padrão**

   Após a criação do arquivo de unidade, é necessário recarregar a configuração do *systemd*, desativar o serviço SSH padrão da distribuição e habilitar o novo serviço `sshd-oqs.service`.

```
sudo systemctl daemon-reload
```

   - Recarrega os arquivos de unidade do *systemd*, tornando o novo serviço `sshd-oqs.service` visível para o gerenciador de serviços.

```
sudo systemctl stop sshd
```

   - Interrompe o serviço `sshd` padrão da distribuição. Recomenda-se não executar este comando se houver sessões SSH críticas ativas sem um meio alternativo de acesso ao sistema.

```
sudo systemctl disable sshd
```

   - Desabilita o serviço SSH padrão para que não seja iniciado automaticamente nas próximas inicializações.

```
sudo systemctl enable --now sshd-oqs.service
```

   - Habilita o serviço `sshd-oqs.service` para iniciar automaticamente no *boot* (`enable`) e o inicia imediatamente na sessão atual (`-now`).

```
sudo systemctl status sshd-oqs.service
```

   - Exibe o estado atual do serviço `sshd-oqs.service`, permitindo verificar se ele foi iniciado corretamente (`Active: active (running)`) ou se houve alguma falha de inicialização (`Active: failed`).

   Caso o estado exibido seja `Active: failed`, será necessário seguir para o passo seguinte do manual, analisando os registros de log e a configuração do serviço para identificar a causa do erro.

5.6.3 Ajuste do *sshd_ config* para ambiente de testes

Para realizar os testes com o OpenSSH recompilado, é conveniente ajustar o arquivo de configuração do servidor SSH, criando uma cópia de segurança e definindo parâmetros específicos (porta, endereço de escuta, autenticação e nível de log).

1. **Fazer *backup* do arquivo de configuração atual**

```
sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.back
```

   - Cria uma cópia de segurança do arquivo `/etc/ssh/sshd_config` em `/etc/ssh/sshd_config.back`, permitindo restaurar a configuração original se necessário.

2. **Substituir o *sshd_ config* por uma configuração mínima de testes**

```
sudo tee /etc/ssh/sshd_config > /dev/null <<'EOF'
Port 22222
ListenAddress 192.168.50.40
PermitRootLogin no
PasswordAuthentication yes
LogLevel VERBOSE
# coloque HostKey e KexAlgorithms conforme teste (veja ssh -Q kex)
EOF
```

   - Utiliza o comando `tee` com *heredoc* para sobrescrever o conteúdo de `/etc/ssh/sshd_config` com uma configuração de testes.
   - `Port 22222`: define que o servidor SSH escutará na porta `22222`, evitando conflito com a porta padrão (22) da instalação da distribuição.
   - `ListenAddress 192.168.50.40`: limita a escuta do servidor ao endereço IP específico do host de testes.
   - `PermitRootLogin no`: desabilita o login direto do usuário `root` via SSH, reforçando a segurança.
   - `PasswordAuthentication yes`: mantém a autenticação por senha habilitada, facilitando os testes (especialmente quando chaves ainda não foram configuradas).
   - `LogLevel VERBOSE`: aumenta o nível de detalhe dos registros de log, útil para depuração e análise de negociação criptográfica.
   - Comentário `# coloque HostKey e KexAlgorithms ...`: indica que chaves de host e algoritmos de troca de chaves (*KexAlgorithms*) devem ser ajustados conforme os cenários de teste, consultando a lista retornada por `ssh -Q kex`.

3. **Reiniciar o serviço e verificar o estado**

```
sudo systemctl restart sshd-oqs.service
sudo systemctl status sshd-oqs.service
```

- Reinicia o serviço `sshd-oqs.service` para aplicar a nova configuração.
- Exibe o status do serviço, permitindo verificar se ele foi iniciado corretamente (`Active: active (running)`) ou se entrou em estado de falha (`Active: failed`).

Caso o serviço ainda esteja em estado de falha, é recomendável rodar o servidor em primeiro plano com nível máximo de depuração:

```
sudo /usr/local/openssh/sbin/sshd -D -f /etc/ssh/sshd_config -ddd
```

- Executa o `sshd` recompilado em primeiro plano (`-D`), utilizando o arquivo `/etc/ssh/sshd_config` e com nível de depuração `-ddd`, exibindo mensagens detalhadas que ajudam a identificar a causa da falha.

4. **Tratar erro de usuário de separação de privilégios ausente**

   Se a saída indicar mensagem semelhante a *"privilege separation user sshd does not exist"*, é necessário criar o usuário de sistema `sshd`:

```
sudo useradd --system --no-create-home --shell /usr/sbin/nologin sshd
```

- Cria um usuário de sistema chamado `sshd`, sem diretório pessoal e sem shell de login, destinado exclusivamente à separação de privilégios do daemon SSH.

Em seguida, tente iniciar novamente o serviço:

```
sudo systemctl restart sshd-oqs.service
sudo systemctl status sshd-oqs.service
```

- Reinicia e verifica novamente o serviço `sshd-oqs.service`. Se ainda houver falhas, é necessário revisar os logs e a configuração de chaves e algoritmos.

5. **Listar algoritmos de troca de chaves (KEX) disponíveis**

   Para preparar os testes criptográficos, é útil registrar os algoritmos de troca de chaves (*key exchange* – KEX) suportados pelo cliente e oferecidos pelo servidor.

```
/usr/local/openssh/bin/ssh -Q kex | \
tee ~/build-oqs/artifacts/ssh-kex-client.txt
```

- Lista, com o cliente SSH recompilado, todos os algoritmos de KEX suportados localmente e grava a saída em `~/build-oqs/artifacts/ssh-kex-client.txt` para futura referência.

```
sudo /usr/local/openssh/sbin/sshd -T | grep kex
```

- Exibe a configuração efetiva do `sshd` recompilado (`-T`) e filtra as linhas relacionadas a KEX, mostrando os algoritmos de troca de chaves que o servidor está oferecendo.

Os nomes exatos dos algoritmos de KEX pós-quânticos ou híbridos dependem do *provider* em uso. A saída de `ssh -Q kex` deve ser utilizada como referência para identificar os nomes válidos (por exemplo, `mlkem_kyber768x25519-sha256`, entre outros, conforme o *build* do *provider*).

Capítulo 6

Máquina Atacante (Kali Linux)

6.1 Pós-instalação

6.1.1 Pós-instalação no Kali Linux (máquina física)

Esta seção descreve os ajustes iniciais recomendados após a instalação do Kali Linux na máquina física que atuará como atacante no ambiente experimental.

1. **Atualização do sistema**

```
sudo apt update && sudo apt full-upgrade -y
```

   - Atualiza a lista de pacotes disponíveis nos repositórios configurados (`apt update`) e aplica todas as atualizações disponíveis para o sistema (`apt full-upgrade -y`).
   - O uso de `-y` automatiza a confirmação das alterações, garantindo que o sistema esteja com os pacotes mais recentes e correções de segurança aplicadas.

2. **Instalação de firmware e utilitários essenciais**

```
sudo apt install firmware-linux firmware-linux-nonfree \
firmware-misc-nonfree firmware-realtek firmware-iwlwifi \
network-manager gparted
```

   - Instala pacotes de *firmware* e ferramentas essenciais para o funcionamento adequado do hardware e a administração da máquina atacante.
   - `firmware-linux`, `firmware-linux-nonfree`, `firmware-misc-nonfree`: coleções gerais de *firmware* para diversos dispositivos, incluindo controladores de vídeo, rede e outros periféricos.
   - `firmware-realtek`, `firmware-iwlwifi`: *firmware* específico para controladores de rede Realtek e Intel (iwlwifi), comumente utilizados em interfaces de rede cabeadas e sem fio.
   - `network-manager`: serviço de gerenciamento de conexões de rede, facilitando a configuração de interfaces Ethernet e Wi-Fi.

- `gparted`: ferramenta gráfica para manipulação de partições de disco, útil para ajustes de armazenamento e organização de mídias.
- Caso o Kali tenha sido instalado a partir de uma imagem que já inclui esses componentes, a instalação adicional pode não ser necessária, mas o comando é seguro e garante a presença dos pacotes.

3. **Habilitação do serviço SSH**

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

- `sudo systemctl enable ssh`: configura o serviço `ssh` para iniciar automaticamente a cada inicialização do sistema.
- `sudo systemctl start ssh`: inicia imediatamente o servidor SSH, permitindo o acesso remoto à máquina atacante, o que é útil para administração e execução de testes a partir de outros hosts.

4. **Correção do Bluetooth inativo (opcional)**

```
sudo systemctl enable bluetooth
sudo systemctl start bluetooth
sudo systemctl status bluetooth
bluetoothctl list
```

- Habilita (`enable`) e inicia (`start`) o serviço `bluetooth`, verificando em seguida o estado do serviço com `systemctl status bluetooth`.
- O comando `bluetoothctl list` lista os controladores Bluetooth detectados, permitindo confirmar se o hardware foi reconhecido.

Caso o Bluetooth continue inativo, recomenda-se identificar o chipset e instalar o *firmware* apropriado:

```
lsusb
lspci
dmesg | grep -i bluetooth
# Instalar conforme fabricante:
# Intel: sudo apt install firmware-iwlwifi
# Realtek: sudo apt install firmware-realtek
# Broadcom: sudo apt install firmware-brcm80211
```

- `lsusb` e `lspci`: listam dispositivos USB e PCI, respectivamente, auxiliando na identificação do fabricante do adaptador Bluetooth.
- `dmesg | grep -i bluetooth`: filtra mensagens do `dmesg` relacionadas a Bluetooth, ajudando a detectar erros de *firmware* ou inicialização.
- Dependendo do chipset identificado (Intel, Realtek, Broadcom, etc.), deve-se instalar o pacote de *firmware* correspondente (como indicado nos comentários), e reiniciar o sistema caso necessário para que o dispositivo seja corretamente inicializado.

6.2 Ferramentas necessárias

6.2.1 Instalação de ferramentas ofensivas no Kali Attacker

Esta etapa aplica-se apenas quando o Kali Linux foi instalado utilizando o *NetInstaller* (*All packages are downloaded during installation*). Caso a instalação tenha sido realizada a partir do instalador completo (*Installer — Complete offline installation with customization*), a maior parte das ferramentas ofensivas já terá sido incluída durante o processo e esta etapa pode ser dispensada.

Objetivo da etapa: garantir que o sistema Kali persistente da máquina atacante disponha de um conjunto atualizado de ferramentas ofensivas, incluindo utilitários para ataques de senha, exploração de vulnerabilidades e reconhecimento de rede utilizados nos experimentos.

6.2.1.1 Atualizar os repositórios

Antes de instalar qualquer ferramenta, recomenda-se atualizar os índices de pacotes do sistema:

```
sudo apt update
```

- Atualiza a lista de pacotes disponíveis nos repositórios configurados, garantindo acesso às versões mais recentes para instalação e atualização.
- Se a máquina atacante estiver fora da rede isolada, é necessário assegurar conectividade com a internet para que o comando seja bem-sucedido. Em ambiente isolado, presume-se que os pacotes necessários já tenham sido obtidos previamente durante a preparação do laboratório.

6.2.1.2 Instalar metapacotes de ferramentas do Kali

O Kali oferece conjuntos funcionais de ferramentas chamados *metapacotes*, que agrupam utilitários por categoria. Para os experimentos deste projeto, recomenda-se a instalação dos seguintes grupos:

```
sudo apt install -y kali-tools-top10 kali-tools-passwords kali-tools-wireless
    kali-tools-web kali-tools-exploitation kali-tools-information-gathering kali
    -tools-vulnerability
```

- Realiza a instalação de metapacotes que abrangem as principais categorias de ferramentas ofensivas necessárias para testes de penetração e simulações.
- `kali-tools-top10`: reúne algumas das ferramentas mais utilizadas em *pentests*, incluindo scanners e exploradores.
- `kali-tools-passwords`: agrupa ferramentas para ataques de senha e quebra de *hashes*, como *Hydra* e *John the Ripper*.
- `kali-tools-wireless`: inclui ferramentas voltadas a ataques e análise de redes sem fio.

- `kali-tools-web`, `kali-tools-exploitation`, `kali-tools-information-gathering`, `kali-tools-vulnerability`: reúnem ferramentas para exploração de aplicações web, coleta de informações, varredura e exploração de vulnerabilidades (por exemplo, *Metasploit*, *SQLMap*, scanners e ferramentas de reconhecimento).
- Em conjunto, essas categorias cobrem força bruta de credenciais, análise de redes, exploração de serviços, scanners de vulnerabilidades e ferramentas de apoio aos testes.

6.2.1.3 Instalar ferramentas ofensivas específicas recomendadas

Além dos metapacotes, recomenda-se garantir explicitamente a instalação de algumas ferramentas ofensivas amplamente utilizadas:

```
sudo apt install -y \
hydra \
john \
hashcat \
nmap \
nikto \
gobuster \
sqlmap \
aircrack-ng \
ettercap-graphical \
wireshark \
tcpdump \
net-tools \
whois \
dnsutils \
resolvconf \
curl \
wget \
vim \
tmux \
gnome-terminal
```

- `hydra`, `john`, `hashcat`: ferramentas voltadas a ataques de senha e quebra de *hashes*, úteis em cenários de força bruta e recuperação de credenciais.
- `nmap`, `whois`, `dnsutils`, `resolvconf`: utilitários de varredura, reconhecimento de rede e manipulação de DNS.
- `nikto`, `gobuster`, `sqlmap`: ferramentas para avaliação de aplicações web, descoberta de conteúdo (força bruta de diretórios) e testes de injeção SQL.
- `aircrack-ng`, `ettercap-graphical`: voltadas a ataques e análise de redes sem fio e interceptação de tráfego em redes comutadas.
- `wireshark`, `tcpdump`: ferramentas de captura e análise detalhada de tráfego de rede, fundamentais para monitoramento e correlação com os experimentos.
- `net-tools`, `curl`, `wget`: utilitários gerais de rede para diagnóstico, transferência de dados e testes de conectividade.

- `vim`, `tmux`, `gnome-terminal`: ferramentas de apoio ao trabalho em linha de comando, permitindo editar arquivos de configuração, gerenciar múltiplas sessões e trabalhar de forma mais organizada durante os testes.
- De forma geral, algumas dessas ferramentas são usadas diretamente em ataques, outras em captura de tráfego, reconhecimento, tunelamento ou manipulação de serviços de nomes (DNS), compondo o *toolkit* ofensivo da máquina atacante.

6.2.1.4 (Opcional) Atualizar o sistema após a instalação das ferramentas

Após a instalação de um conjunto extenso de pacotes, pode ser conveniente realizar uma atualização completa do sistema para harmonizar dependências e aplicar correções recentes:

```
sudo apt update && sudo apt full-upgrade -y
```

- Garante que todas as ferramentas recém-instaladas e suas dependências estejam nas versões mais recentes disponíveis nos repositórios configurados, reduzindo problemas de incompatibilidade.

6.2.1.5 Confirmar as ferramentas instaladas

Para verificar rapidamente quais metapacotes relacionados às ferramentas do Kali estão instalados:

```
dpkg -l | grep kali-tools
```

- Lista, entre os pacotes instalados, aqueles cujo nome contém `kali-tools`, permitindo confirmar quais metapacotes foram efetivamente instalados no sistema.

Também é possível verificar a presença de ferramentas específicas conferindo seus caminhos no `PATH`:

```
which hydra john sqlmap metasploit-framework nmap
```

- Exibe o caminho dos executáveis `hydra`, `john`, `sqlmap`, `metasploit-framework` e `nmap`. Caso algum comando não retorne caminho, indica que a ferramenta correspondente pode não estar instalada ou não estar acessível no `PATH`.

6.2.1.6 Organização das ferramentas no menu gráfico (XFCE)

Quando a máquina atacante utiliza o ambiente gráfico padrão do Kali (XFCE), as ferramentas instaladas são organizadas no menu principal em:

Applications → Kali Linux → [Categoria]

Exemplos de categorização:

- **Information Gathering**: ferramentas como *nmap*, *whois*.
- **Password Attacks**: *hydra*, *john*.

- **Exploitation Tools**: *metasploit-framework*, *sqlmap*.
- **Wireless Attacks**: *aircrack-ng*, *kismet* (quando instalado).
- **Web Applications**: *nikto*, *gobuster*, entre outras ferramentas focadas em aplicações web.

Essa organização no menu gráfico complementa o uso em linha de comando, facilitando o acesso às ferramentas ofensivas por usuários que preferem ou precisam localizar os programas via interface gráfica.

6.3 Pilha criptográfica compartilhada com o Servidor SSH Real

A máquina atacante reutiliza a mesma pilha criptográfica construída para o Servidor SSH Real, a fim de garantir comparabilidade entre os cenários de teste (criptografia clássica, pós-quântica e híbrida). Dessa forma, todos os passos de preparação de ambiente e compilação de bibliotecas criptográficas são idênticos aos descritos no Capítulo do Servidor SSH Real, até a etapa de *troubleshooting* do *oqs-provider*.

Na máquina atacante (Kali Linux), execute exatamente as mesmas etapas a seguir, na mesma ordem, ajustando apenas o usuário e diretórios de trabalho conforme necessário:

1. **Preparação do ambiente e dependências** Reproduzir a Seção 5.2, instalando no Kali as mesmas dependências de compilação (compiladores, *autotools*, *cmake*, *ninja*, bibliotecas de desenvolvimento etc.).
2. **OpenSSL personalizada** Reproduzir a Seção 5.3 , compilando e instalando o OpenSSL 3.x em `/usr/local/openssl` e configurando as variáveis de ambiente (`PATH`, `LD_LIBRARY_PATH`, `PKG_CONFIG_PATH`, `CPPFLAGS`, `LDFLAGS`) da mesma forma que na máquina Servidor.
3. **Biblioteca *liboqs*** Reproduzir a Seção 5.4, clonando o repositório *liboqs*, compilando com *cmake*/*ninja* e instalando em `/usr/local/liboqs`, com registro do *commit* utilizado.
4. **Provider *oqs-provider* para o OpenSSL 3** Reproduzir a Seção 5.5 , compilando e instalando o *oqs-provider* compatível com a *liboqs* e o OpenSSL personalizados e ajustando o arquivo de configuração `openssl.cnf` e a variável `OPENSSL_CONF`, se necessário.

Após essas etapas, a máquina atacante passa a dispor da mesma pilha criptográfica da máquina Servidor SSH Real, porém será utilizada apenas como *cliente* SSH nos experimentos (não é necessário repetir as etapas específicas de serviço *systemd* nem as configurações de `sshd` descritas para o servidor).

Capítulo 7

Máquina de Monitoramento (Kali Purple)

7.1 Pós-instalação

7.1.1 Atualização inicial do sistema

Após a instalação do Kali Purple, é essencial realizar uma atualização completa do sistema. Isso garante que o ambiente de monitoramento esteja com todas as correções de segurança aplicadas, versões recentes dos pacotes e estabilidade adequada para executar ferramentas críticas como Zeek, Suricata e Splunk.

Abra o terminal e execute:

```
sudo apt update
sudo apt full-upgrade -y
sudo apt autoremove -y
```

- `apt update`: atualiza a lista de pacotes disponíveis para download, sincronizando o sistema com os repositórios oficiais.
- `apt full-upgrade`: instala todas as atualizações disponíveis, incluindo atualizações de kernel, módulos, bibliotecas e correções de vulnerabilidades.
- `apt autoremove`: remove dependências antigas que não são mais necessárias, prevenindo acúmulo de pacotes obsoletos.

Opcionalmente, reinicie:

```
sudo reboot
```

A reinicialização é recomendada caso haja atualização do kernel ou serviços essenciais ao monitoramento.

Após o reboot, faça login novamente e abra o terminal para continuar.

7.1.2 Instalar ferramentas básicas de rede

Estas ferramentas são fundamentais para diagnóstico, testes iniciais, verificação de conectividade, captura manual de pacotes e análises auxiliares. Servem como base para

validações antes, durante e após a ativação dos sensores Zeek e Suricata, bem como para depuração de eventuais problemas na topologia monitorada.

```
sudo apt install -y \
  curl wget git vim nano net-tools iproute2 iputils-ping tcpdump ethtool lsof
   htop lsb-release unzip
```

- `curl`: ferramenta de linha de comando para requisições HTTP/HTTPS e outros protocolos; será usada para baixar arquivos de configuração, regras de IDS ou testar endpoints HTTP expostos no laboratório.
- `wget`: semelhante ao `curl`, porém focado em download de arquivos; útil para obter pacotes, scripts e arquivos de log de referência.
- `git`: sistema de controle de versão; permite clonar repositórios contendo scripts de coleta, configurações padronizadas e outros artefatos do laboratório.
- `vim` e `nano`: editores de texto em modo terminal; usados para editar arquivos de configuração do Suricata, Zeek, Splunk e scripts auxiliares.
- `net-tools`: conjunto tradicional de utilitários de rede (como `ifconfig`, `netstat`, `route`), ainda amplamente utilizado em diagnósticos e validações rápidas.
- `iproute2`: conjunto moderno de ferramentas (`ip`, `ss`, etc.) para configuração e inspeção de interfaces, rotas e sockets; é a base para diagnósticos de rede em distribuições atuais.
- `iputils-ping`: provê o comando `ping`, utilizado constantemente para testar conectividade entre o monitor, o servidor SSH e a máquina atacante.
- `tcpdump`: ferramenta de captura de pacotes em linha de comando; essencial para verificar se o tráfego espelhado pelo switch está chegando corretamente à interface de monitoramento.
- `ethtool`: usado para inspecionar e ajustar parâmetros da interface de rede (modo, velocidade, duplex); importante para validar a interface conectada à porta espelhada do switch TL-SG105E.
- `lsof`: lista arquivos abertos por processos; muito útil para identificar quais serviços estão ouvindo em determinadas portas ou mantendo arquivos de log em uso.
- `htop`: monitor interativo de processos e recursos; permite acompanhar o impacto de Suricata, Zeek e Splunk em CPU e memória durante execuções intensivas.
- `lsb-release`: exibe informações de identificação da distribuição (nome, versão); útil para documentação e verificação de compatibilidade de pacotes.
- `unzip`: utilitário para descompactar arquivos no formato `.zip`; facilita o manuseio de pacotes, conjuntos de regras ou coleções de logs distribuídos nesse formato.

7.1.3 Instalar ferramentas de apoio e utilitários

Além das ferramentas essenciais, estes utilitários adicionais ajudam nas análises forenses, diagnósticos avançados e validações de rede durante a operação do laboratório. Eles permitem observar o comportamento do ambiente em tempo real e rastrear gargalos de rede, CPU e disco.

```
sudo apt install -y htop iftop iotop net-tools tcpdump ethtool traceroute git
    curl unzip lsb-release zmap nmap arp-scan
```

- `htop`: reforça o papel de monitor de processos, facilitando a análise de uso de recursos quando Suricata, Zeek e Splunk estão em execução simultânea.
- `iftop`: exibe, em tempo real, o volume de tráfego por conexão na interface selecionada; muito útil para ver, imediatamente, se o espelhamento de porta está entregando tráfego ao monitor.
- `iotop`: monitora o uso de I/O em disco por processo; importante para avaliar o impacto da gravação de logs (especialmente pelo Splunk) no desempenho do sistema.
- `net-tools`, `tcpdump`, `ethtool`, `git`, `curl`, `unzip`, `lsb-release`: complementam as ferramentas já instaladas anteriormente, garantindo redundância e flexibilidade para diagnósticos e automação.
- `traceroute`: permite mapear o caminho que um pacote percorre até um determinado destino; útil se houver múltiplos segmentos de rede ou roteamento diferenciado na topologia.
- `zmap`: ferramenta de varredura rápida em larga escala; pode ser utilizada para gerar tráfego de teste e validar se o monitor está capturando corretamente grandes volumes de pacotes.
- `nmap`: scanner de portas e serviços; pode ser utilizado tanto para validar a exposição de serviços (como o SSH real) quanto para verificar se o monitor está observando o tráfego desses scans.
- `arp-scan`: realiza varreduras na rede local usando ARP; útil para identificar dispositivos presentes no segmento monitorado e validar a visão do monitor sobre a camada 2.

Essas ferramentas complementam a operação e facilitam o diagnóstico de toda a topologia monitorada, permitindo validar tanto o funcionamento dos sensores quanto o comportamento do tráfego de teste.

7.1.4 Instalar pacotes Python úteis (via `pip`)

Em vários momentos da análise, pode ser útil manipular arquivos de log, gerar gráficos e consolidar estatísticas. Para isso:

```
sudo apt install -y python3-pip
pip3 install matplotlib pandas requests jupyterlab
```

- `pandas`: ideal para consolidar registros de Zeek e Suricata em tabelas.
- `matplotlib`: permite criação de gráficos e visualizações.
- `jupyterlab`: oferece ambiente interativo para relatórios exploratórios.

7.1.5 Limpeza do sistema

Após instalar e configurar tudo:

```
sudo apt autoremove -y
sudo apt clean
```

- Remove pacotes obsoletos.
- Reduz uso de disco, especialmente importante nesta máquina, já que os logs do Splunk tendem a ocupar bastante espaço.

7.2 Verificar ferramentas principais (pré-instaladas)

Antes de prosseguir com instalações adicionais, é importante checar se as principais ferramentas de monitoramento já estão disponíveis na imagem do Kali Purple: Zeek, Suricata e Wireshark. Elas formam a base da análise de tráfego deste laboratório.

Execute:

```
which zeek
which suricata
which wireshark
```

- Se algum comando retornar um caminho (por exemplo, `/usr/bin/zeek`), significa que o executável correspondente está instalado e acessível no `PATH`.
- Se o comando não retornar nada (linha em branco) ou exibir `command not found`, indica que a ferramenta não está presente e precisará ser instalada manualmente nas seções seguintes.

No contexto deste laboratório:

- **Zeek**: será utilizado como *Network Security Monitor*, gerando logs detalhados sobre conexões, sessões SSH e padrões de tráfego que serão correlacionados posteriormente no Splunk.
- **Suricata**: atuará como IDS/IPS, aplicando regras de detecção sobre o tráfego espelhado e produzindo alertas que também serão enviados para análise centralizada.
- **Wireshark**: servirá como ferramenta gráfica de apoio, permitindo inspeção pontual e detalhada de pacotes em casos específicos de depuração ou validação de comportamento.

Suricata e Zeek são os sensores principais do laboratório e, portanto, sua presença e funcionamento são fundamentais para que o monitoramento das conexões SSH em criptografia clássica e híbrida pós-quântica seja efetivamente registrado e analisado.

7.3 Instalar ferramentas que estiverem ausentes

Nesta etapa, são instaladas manualmente as ferramentas que não vieram pré-instaladas na imagem do Kali Purple, em especial o Zeek e, se necessário, o Suricata. É obrigatório que a máquina esteja com acesso à internet e que os comandos sejam executados com privilégios `sudo`.

7.3.1 Zeek (Network Security Monitor)

O Zeek é um *Network Security Monitor* voltado à inspeção profunda de tráfego, geração de logs detalhados e suporte a análises forenses. Em algumas instalações do Kali Purple ele já está presente; em outras, precisa ser adicionado manualmente.

Quando o Zeek não estiver disponível (`which zeek` não retorna caminho), pode ser instalado a partir do repositório oficial mantido para Debian 12:

```
echo 'deb http://download.opensuse.org/repositories/security:/zeek/Debian_12/ /'
    | sudo tee /etc/apt/sources.list.d/security:zeek.list
curl -fsSL https://download.opensuse.org/repositories/security:zeek/Debian_12/
   Release.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/security_zeek.
   gpg > /dev/null
sudo apt update
sudo apt install -y zeek
```

- A primeira linha adiciona um arquivo de lista de repositórios específico para o Zeek (`security:zeek`) ao sistema, apontando para a versão destinada ao Debian 12 (*bookworm*).
- A segunda linha faz o download da chave de assinatura do repositório e a converte para o formato aceito pelo `apt`, permitindo que os pacotes instalados desse repositório tenham sua integridade e autenticidade verificadas.
- Em seguida, a lista de pacotes é atualizada e o Zeek é instalado a partir desse repositório, garantindo uma versão adequada ao ambiente.

Após a instalação, verifique a versão:

```
zeek --version
```

- A saída desse comando confirma não apenas a instalação, mas também qual versão está em uso, informação relevante para comparar resultados e reproduzir o laboratório em outros ambientes.

7.4 Suricata (IDS/IPS)

O Suricata é o IDS/IPS responsável por inspecionar o tráfego espelhado para a máquina de monitoramento, aplicando regras de detecção de ataques e gerando alertas. Em muitas imagens do Kali Purple ele já está presente, mas, se não estiver, sua instalação é direta via repositório.

7.4.1 Instalar Suricata

Quando o comando `which suricata` não retornar caminho, faça:

```
sudo apt update
sudo apt install -y suricata
```

- `apt update`: garante que a lista de pacotes esteja sincronizada com os repositórios.
- `apt install suricata`: instala o Suricata e suas dependências, integrando-o ao sistema como serviço gerenciado pelo `systemd`.

Verifique a compilação e a versão:

```
suricata --build-info
suricata -V
```

- `-build-info`: exibe informações detalhadas sobre como o Suricata foi compilado (bibliotecas, aceleração, módulos ativos), o que ajuda a entender os recursos disponíveis (por exemplo, suporte a NFLOG, AF_PACKET, etc.).
- `-V`: mostra a versão exata instalada, importante para documentar o ambiente e comparar resultados com outros trabalhos.

7.4.2 Habilitar no boot e iniciar o serviço

Para que o Suricata seja ativado automaticamente a cada inicialização do sistema e comece a inspecionar o tráfego sem intervenção manual:

```
sudo systemctl enable suricata.service
sudo systemctl start suricata.service
```

- `systemctl enable`: registra o serviço `suricata.service` para ser iniciado automaticamente no boot.
- `systemctl start`: inicia imediatamente o serviço na sessão atual, sem necessidade de reiniciar o sistema.

7.4.3 Verificar status do Suricata

Para confirmar que o serviço foi iniciado corretamente:

```
sudo systemctl status suricata.service
```

- A saída mostra se o serviço está *active (running)* ou se houve falha na inicialização, exibindo também as últimas mensagens de log relevantes.
- Se houver erros relacionados à interface de rede ou ao arquivo de configuração, eles serão destacados nessa saída, facilitando o diagnóstico.
- Pressione `q` para sair da tela de status e voltar ao terminal.

7.5 Reiniciar e verificar ambiente

Após instalar e configurar o Zeek e o Suricata, é recomendável reiniciar o sistema para garantir que todos os serviços e dependências sejam carregados adequadamente na inicialização.

Reinicie o sistema:

```
sudo reboot
```

Depois do boot, realize as verificações:

```
zeek --version
suricata --build-info
suricata -V
```

- Esses comandos confirmam que o Zeek continua acessível após o reboot e que o Suricata permanece instalado com a mesma versão e configuração de compilação.

Em seguida, verifique o status do serviço Suricata:

```
sudo systemctl status suricata
```

- A presença de *active (running)* indica que o IDS está operativo e pronto para receber o tráfego espelhado do *switch*.

7.5.1 Ajuste de PATH para o Zeek (se necessário)

Em alguns cenários, o Zeek pode ter sido instalado em um diretório fora do `PATH` padrão, como `/opt/zeek/bin`. Nessa situação, o binário existe e funciona, mas o comando `zeek` não é encontrado diretamente.

Primeiro, localize o binário:

```
whereis zeek
```

Supondo que o caminho retornado inclua `/opt/zeek/bin/zeek`, teste:

```
/opt/zeek/bin/zeek --version
```

- Se a versão for exibida corretamente, confirma-se que o problema é apenas de `PATH`, e não de instalação.

Verifique qual shell está sendo usado:

```
echo $SHELL
```

O resultado esperado, neste ambiente, é algo como:

```
/usr/bin/zsh
```

Isso indica que o arquivo de configuração adequado é `~/.zshrc`. Para incluir o Zeek no `PATH`:

```
echo 'export PATH=$PATH:/opt/zeek/bin' >> ~/.zshrc
```

Em seguida, recarregue a configuração:

```
source ~/.zshrc
```

Teste novamente:

```
which zeek
zeek --version
```

- Agora, `which zeek` deve apontar para `/opt/zeek/bin/zeek`, e `zeek -version` deve exibir a mesma versão utilizada nos experimentos (por exemplo, `8.0.4`).

7.6 Instalação e Uso do Splunk Enterprise (One-shot na Máquina de Monitoramento)

O Splunk Enterprise será utilizado nesta máquina como plataforma central de análise, recebendo logs gerados por Suricata, Zeek e pelos atos de conexão SSH. O modo de uso será *one-shot*, ou seja, com ingestão dirigida de arquivos de log gerados nos experimentos, em vez de coleta contínua via *forwarders*.

7.6.1 Download do Splunk Enterprise

O instalador deve ser obtido diretamente do site oficial da ferramenta, garantindo que a versão utilizada seja confiável e compatível com o ambiente Linux adotado.

Abra o navegador no Kali Purple e acesse:

https://www.splunk.com/en_us/products/splunk-enterprise.html

- Crie ou faça login em uma conta Splunk.
- Escolha a opção de download para **Linux**, em formato `.tgz`, adequada para extração em `/opt`.

A partir da URL disponibilizada pelo site, o download pode ser feito via terminal. O diretório `/tmp` é utilizado como área de trabalho temporária:

```
cd /tmp
```

- Altera o diretório de trabalho atual para `/tmp`, que será usado como local temporário para armazenar o arquivo `.tgz` baixado.

Em seguida, realiza-se o download do instalador. O exemplo abaixo corresponde a uma versão específica do Splunk; a URL pode ser ajustada conforme a versão selecionada no site oficial:

```
wget -O splunk-10.0.2-e2d18b4767e9-linux-amd64.tgz "https://download.splunk.com/
    products/splunk/releases/10.0.2/linux/splunk-10.0.2-e2d18b4767e9-linux-amd64
    .tgz"
```

- `wget`: realiza o download do arquivo a partir da URL informada.

- A opção `-O` define explicitamente o nome do arquivo salvo localmente (`splunk-10.0.2-e2d18b4767e9-linux-amd64.tgz`), facilitando a identificação e o uso nos comandos seguintes.

Após o download, o pacote é extraído para o diretório `/opt`, onde ficará instalada a aplicação:

```
sudo tar -xvzf splunk-10.0.2-e2d18b4767e9-linux-amd64.tgz -C /opt
```

- `tar -xvzf`: descompacta o arquivo `.tgz` (tar + gzip), exibindo os arquivos extraídos (`-v`) e preservando a estrutura do pacote.
- A opção `-C /opt` direciona a extração para o diretório `/opt`, local típico para instalação de softwares de terceiros.

Por fim, ajusta-se a propriedade dos arquivos para o usuário atual, facilitando a administração em ambiente de laboratório:

```
sudo chown -R $(whoami):$(whoami) /opt/splunk
```

- `chown -R`: altera o proprietário (*owner*) e o grupo de todos os arquivos dentro de `/opt/splunk`.
- `$(whoami)` garante que o diretório pertença ao usuário logado, permitindo executar comandos administrativos do Splunk sem problemas de permissão durante os testes.

Após esses passos, o diretório:

`/opt/splunk`

passa a conter a instalação completa do Splunk Enterprise utilizada neste laboratório.

7.6.2 Iniciar Splunk pela primeira vez (aceitar licença e criar admin)

Na primeira inicialização, o Splunk exige a aceitação da licença de uso e a criação de um usuário administrador, que será utilizado para acessar a interface web e configurar índices, buscas e ingestões de dados.

Primeiro, acesse o diretório de binários do Splunk:

```
cd /opt/splunk/bin
```

- Define o diretório atual para `/opt/splunk/bin`, onde se encontram os executáveis principais, incluindo o comando `splunk`.

Em seguida, execute:

```
sudo ./splunk start --accept-license
```

- `./splunk start`: inicia o serviço Splunk.

- A opção `-accept-license` indica que o usuário concorda com os termos da licença exibidos, permitindo prosseguir com a inicialização.
- Durante essa primeira execução, o Splunk solicitará a criação de um usuário administrador (geralmente `admin`) e uma senha forte.

A saída padrão inclui mensagens semelhantes a:

```
This appears to be your first time running this version of Splunk.

Splunk software must create an administrator account during startup. Otherwise,
    you cannot log in.
Create credentials for the administrator account.
Characters do not appear on the screen when you type in credentials.

Please enter an administrator username:
Password must contain at least:
   * 8 total printable ASCII character(s).
Please enter a new password:
```

- Nesta etapa, são definidos o nome de usuário e a senha do administrador da instância, que serão utilizados para acessar a interface web do Splunk e administrar o ambiente de análise.

Após a inicialização bem-sucedida, a interface web do Splunk fica disponível em:

`https://<ip-do-monitor>:8000`

Ao acessar esse endereço no navegador do Kali Purple, é exibida a tela de autenticação do Splunk Enterprise, conforme ilustrado na Figura 7.1.

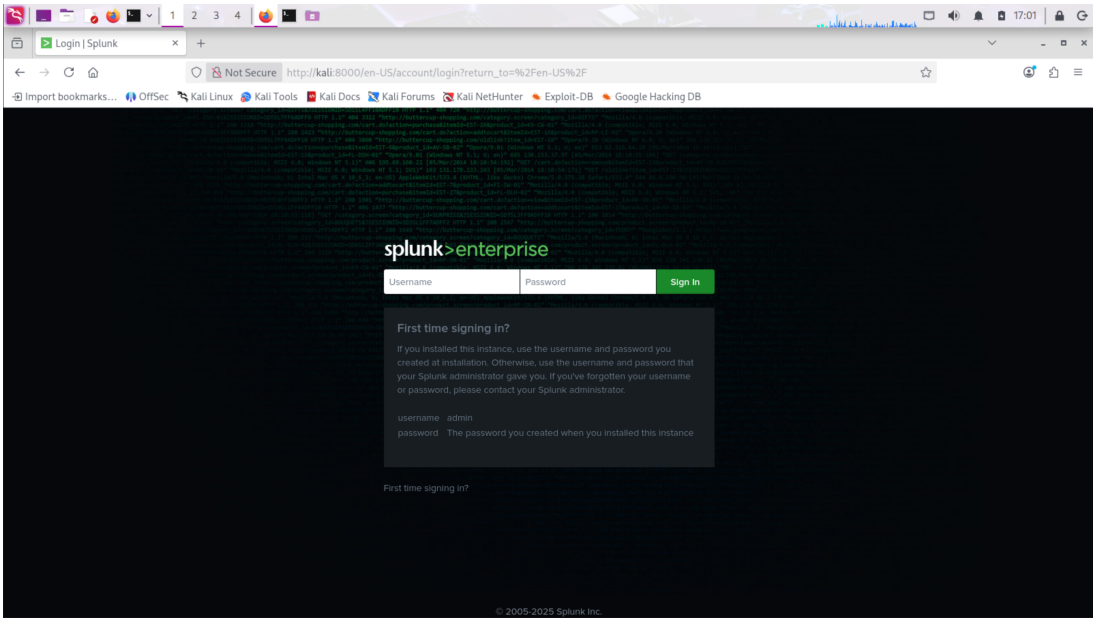

Figura 7.1: Tela de login do Splunk Enterprise acessada a partir do Kali Purple.

Utilize o nome de usuário e a senha definidos na primeira execução do comando `splunk start` (tipicamente `admin` e a senha cadastrada). Após o login bem-sucedido, o Splunk apresenta a página inicial do ambiente administrativo, como mostrado na Figura 7.2.

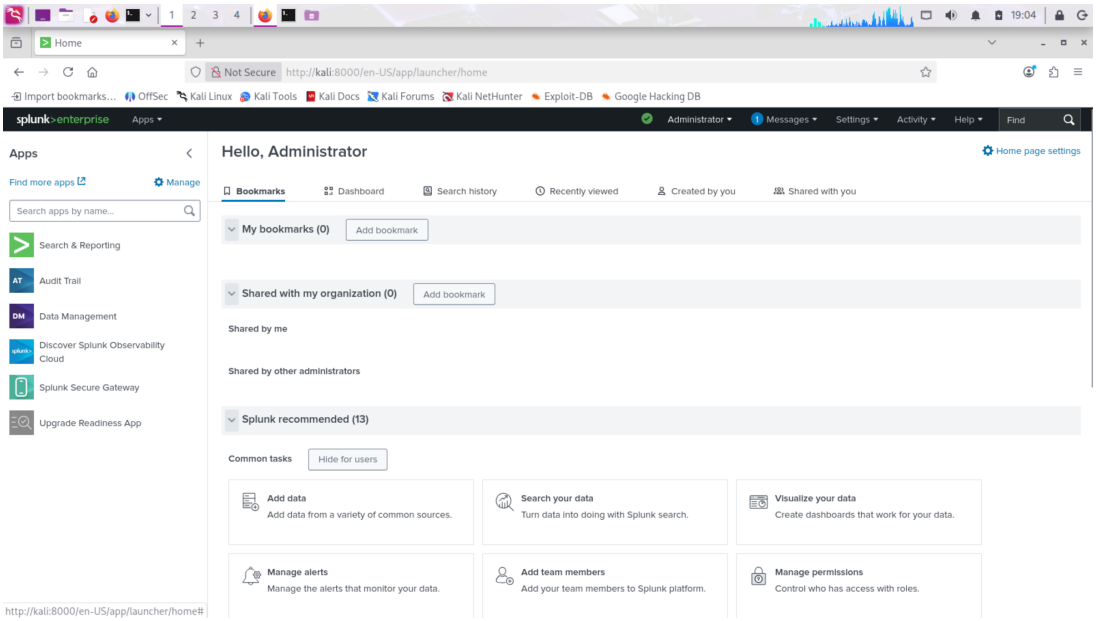

Figura 7.2: Página inicial do Splunk Enterprise após o login do administrador.

7.6.3 Verificar Splunk Enterprise (no monitor – Kali Purple)

O Splunk Enterprise utilizado aqui é o serviço completo, com interface web, responsável pela análise centralizada dos logs gerados no laboratório. É importante verificar se ele está em execução antes de qualquer ingestão (*one-shot*) de arquivos de log.

7.6.3.1 Verificar status via linha de comando

Para checar o estado do serviço:

```
sudo /opt/splunk/bin/splunk status
```

- Consulta o estado atual da instância Splunk, indicando se o processo principal (`splunkd`) e os processos auxiliares estão em execução.

Se o Splunk estiver em execução, a saída será semelhante a:

```
splunkd is running (PID: 1234).
splunk helpers are running (PID: 1250).
```

Se estiver parado:

```
splunkd is not running.
splunk helpers are not running.
```

- Recomenda-se executar essa verificação antes de rodar qualquer script ou ato do laboratório que dependa de envio de logs ao Splunk, de forma a evitar perda de dados por ausência do serviço.

7.6.3.2 Iniciar, parar e reiniciar o Splunk

As operações de controle do serviço podem ser feitas diretamente pelo binário do Splunk.

Para iniciar:

```
sudo /opt/splunk/bin/splunk start
```

- Inicia o serviço Splunk caso esteja parado, tornando novamente acessível a interface web e as funcionalidades de busca e análise.

Para parar:

```
sudo /opt/splunk/bin/splunk stop
```

- Encerra o serviço Splunk de forma controlada, garantindo que os arquivos de índice e configurações sejam fechados corretamente antes de desligar a máquina ou realizar manutenções.

Para reiniciar:

```
sudo /opt/splunk/bin/splunk restart
```

- Interrompe e inicia novamente o serviço Splunk, aplicando alterações em configurações internas (como criação de novos índices ou ajustes de ingestão) sem necessidade de reiniciar o sistema operacional.

7.7 Captura e Armazenamento de Logs

A máquina de monitoramento é responsável por registrar, de forma estruturada, todo o tráfego gerado nos experimentos, incluindo as conexões SSH entre atacante e servidor real. Para isso, foi desenvolvido o script `prepare_monitor.sh`, que inicializa simultaneamente:

- a captura de pacotes via `tcpdump`;
- a análise em tempo real pelo Zeek (*Network Security Monitor*);
- a análise em tempo real pelo Suricata (IDS/IPS).

Cada execução do script gera uma nova estrutura de diretórios, bem como um identificador único de execução (`RUN_ID`), permitindo correlacionar PCAPs, logs de Zeek, logs de Suricata e metadados do experimento.

7.7.1 Script `prepare_monitor.sh`

O script abaixo deve ser copiado para a máquina de monitoramento (Kali Purple), tornado executável (`chmod +x prepare_monitor.sh`) e chamado antes da execução de cada *Ato* de exfiltração/conexão SSH.

```
#!/usr/bin/env bash
# prepare_monitor.sh (FINAL)
# Uso: ./prepare_monitor.sh <interface> <attacker_ip> <target_ip> <mode> <
    profile> <label>
# Ex.: ./prepare_monitor.sh eth0 192.168.50.20 192.168.50.40 classico leve
    sshreal
set -euo pipefail

IFACE="${1:-eth0}"
ATTACKER_IP="${2:-192.168.50.20}"
TARGET_IP="${3:-192.168.50.40}"
MODE="${4:-classico}"        # classico | hibrido
PROFILE="${5:-leve}"         # leve | medio | pesado (metadados)
LABEL="${6:-target}"         # cowrie | sshreal | outro

BASE="$HOME/monitoramento1/$MODE/exfil_compare/$LABEL/$PROFILE"
PCAPDIR="$BASE/pcap"
ZEEKDIR="$BASE/zeek"
SURICATADIR="$BASE/suricata"
PIDDIR="$BASE/pids"
LOGDIR="$BASE/logs"
METAFILE="$BASE/meta_run.txt"

mkdir -p "$PCAPDIR" "$ZEEKDIR" "$SURICATADIR" "$PIDDIR" "$LOGDIR"

TS="$(date +%Y%m%d-%H%M%S)"
RUN_ID="${LABEL}_${MODE}_${PROFILE}_${TS}"
PCAP_FILE="${PCAPDIR}/${RUN_ID}.pcap"

# Registra metadados deste run
cat > "$METAFILE" <<EOF
RUN_ID=$RUN_ID
TIMESTAMP=$TS
IFACE=$IFACE
ATTACKER_IP=$ATTACKER_IP
TARGET_IP=$TARGET_IP
MODE=$MODE
PROFILE=$PROFILE
LABEL=$LABEL
PCAP_FILE=$PCAP_FILE
ZEEK_DIR=$ZEEKDIR
SURICATA_DIR=$SURICATADIR
LOG_DIR=$LOGDIR
EOF

echo "[*] RUN_ID=$RUN_ID"
echo "[*] Diretório base do Ato: $BASE"
echo

# 1) tcpdump (sempre dentro do Ato)
echo "[*] Iniciando tcpdump -> $PCAP_FILE"
```

```
nohup sudo tcpdump -i "$IFACE" "(host $ATTACKER_IP and host $TARGET_IP)" -w "
    $PCAP_FILE" >/dev/null 2>&1 &
echo $! > "$PIDDIR/tcpdump.pid"
sleep 0.3
echo "[OK] tcpdump pid: $(cat "$PIDDIR/tcpdump.pid")"

# 2) Zeek live (forçar CWD e logdir do Ato)
ZEEKBIN="${ZEEKBIN:-/opt/zeek/bin/zeek}"
ZEEK_LOG="$LOGDIR/zeek-startup-${TS}.log"
if [[ -x "$ZEEKBIN" ]]; then
  echo "[*] Iniciando Zeek live (CWD=$ZEEKDIR, Log::default_logdir=$ZEEKDIR)"
  nohup sudo bash -c "cd '$ZEEKDIR' && exec '$ZEEKBIN' -i '$IFACE' local Log::
    default_logdir='$ZEEKDIR'" &> "$ZEEK_LOG" &
  echo $! > "$PIDDIR/zeek.pid"
  sleep 0.3
  echo "[OK] zeek pid: $(cat "$PIDDIR/zeek.pid") | startup: $ZEEK_LOG"
else
  echo "[WARN] Zeek não encontrado em $ZEEKBIN; Zeek live não será iniciado (
    fallback offline no collect)."
fi

# 3) Suricata live (sempre dentro do Ato)
if command -v suricata >/dev/null 2>&1; then
  echo "[*] Iniciando Suricata live (logdir=$SURICATADIR)"
  SURICATA_LOG="$LOGDIR/suricata-startup-${TS}.log"
  nohup sudo suricata -c /etc/suricata/suricata.yaml -i "$IFACE" -l "
    $SURICATADIR" &> "$SURICATA_LOG" &
  echo $! > "$PIDDIR/suricata.pid"
  sleep 0.3
  echo "[OK] suricata pid: $(cat "$PIDDIR/suricata.pid") | startup:
    $SURICATA_LOG"
else
  echo "[*] Suricata não instalada; pulando Suricata live (fallback offline no
    collect)."
fi

echo
echo "[OK] Monitoramento iniciado (exfil_compare/$LABEL)."
echo "    pcap: $PCAP_FILE"
echo "    zeek dir: $ZEEKDIR"
echo "    suricata dir: $SURICATADIR"
echo "    meta: $METAFILE"
```

7.7.1.1 Parâmetros de execução

O script recebe seis parâmetros posicionais, permitindo reutilizar o mesmo procedimento para diferentes cenários de experimento:

- `<interface>` (`IFACE`): interface de rede do Kali Purple que recebe o espelhamento de tráfego (ex.: `eth0`, `eth1`). É a interface onde o `tcpdump`, o Zeek e o Suricata irão capturar os pacotes.

- <attacker_ip> (ATTACKER_IP): endereço IP da máquina atacante (Kali Linux), usado no filtro do tcpdump.
- <target_ip> (TARGET_IP): endereço IP do alvo monitorado (tipicamente o Servidor SSH Real).
- <mode> (MODE): modo criptográfico sob teste, por exemplo classico ou hibrido. Este valor é refletido na estrutura de diretórios e nos metadados.
- <profile> (PROFILE): perfil de carga usado como metadado (leve, medio, pesado), indicando a intensidade/quantidade de operações daquele *run*.
- <label> (LABEL): rótulo lógico do alvo (sshreal, cowrie, etc.), permitindo diferenciar cenários mesmo com IPs ou modos semelhantes.

Um exemplo típico de execução é:

```
./prepare_monitor.sh eth0 192.168.50.20 192.168.50.40 classico leve sshreal
```

Esse comando inicia a captura para conexões entre o atacante 192.168.50.20 e o servidor real 192.168.50.40, no modo classico, perfil leve, rotulado como sshreal.

7.7.1.2 Estrutura de diretórios e metadados gerados

Logo após a leitura dos parâmetros, o script define a base de armazenamento:

- BASE = $HOME/monitoramento1/$MODE/exfil_compare/$LABEL/$PROFILE

A partir desse caminho, são criados automaticamente os subdiretórios:

- pcap: arquivos .pcap gerados pelo tcpdump.
- zeek: diretório de trabalho e saída de logs do Zeek.
- suricata: diretório onde o Suricata grava seus logs (inclusive eve.json).
- pids: arquivos contendo os PIDs dos processos iniciados (tcpdump, Zeek, Suricata).
- logs: arquivos de *startup* e mensagens de erro (zeek-startup-.log, suricata-startup-.log).

O script gera ainda:

- um **timestamp** no formato YYYYMMDD-HHMMSS (TS);
- um **identificador de execução** RUN_ID, no formato LABEL_MODE_PROFILE_TS, que é usado no nome do arquivo .pcap;
- o arquivo meta_run.txt, contendo os principais metadados (IPs, interface, modo, perfil, caminhos gerados).

O bloco:

```
cat > "$METAFILE" <<EOF
...
EOF
```

registra, em texto plano, campos como RUN_ID, TIMESTAMP, ATTACKER_IP, TARGET_-IP, diretórios de saída e o caminho completo para o arquivo .pcap. Esse arquivo é fundamental para correlação posterior no Splunk e em análises manuais.

7.7.1.3 Inicialização do `tcpdump`

O primeiro sensor iniciado é o `tcpdump`, responsável pela captura bruta de pacotes:

```
nohup sudo tcpdump -i "$IFACE" "(host $ATTACKER_IP and host $TARGET_IP)" -w "
    $PCAP_FILE" \
  >/dev/null 2>&1 &
echo $! > "$PIDDIR/tcpdump.pid"
```

- `-i "$IFACE"`: define a interface de captura (por exemplo, `eth0`) — deve ser a interface que recebe o espelhamento do switch.
- o filtro `(host $ATTACKER_IP and host $TARGET_IP)` limita a captura ao tráfego entre atacante e alvo, reduzindo o volume de dados.
- `-w "$PCAP_FILE"`: grava os pacotes diretamente em um arquivo `.pcap`, com nome baseado no `RUN_ID`.
- `nohup` e `&`: permitem que o processo continue em *background*, mesmo que o terminal de origem seja fechado.
- `>/dev/null 2>&1`: suprime a saída padrão e de erro, direcionando qualquer mensagem para o `/dev/null`.
- `$!`: PID do último processo em *background*, armazenado em `pids/tcpdump.pid` para possível encerramento ou inspeção posterior.

7.7.1.4 Execução do Zeek em modo *live*

Em seguida, é iniciado o Zeek para análise em tempo real:

```
ZEEKBIN="${ZEEKBIN:-/opt/zeek/bin/zeek}"
ZEEK_LOG="$LOGDIR/zeek-startup-${TS}.log"
if [[ -x "$ZEEKBIN" ]]; then
  echo "[*] Iniciando Zeek live (CWD=$ZEEKDIR, Log::default_logdir=$ZEEKDIR)"
  nohup sudo bash -c "cd '$ZEEKDIR' && exec '$ZEEKBIN' -i '$IFACE' local Log::
    default_logdir='$ZEEKDIR'" \
    &> "$ZEEK_LOG" &
  echo $! > "$PIDDIR/zeek.pid"
  ...
fi
```

- `ZEEKBIN`: permite ajustar o caminho do binário do Zeek via variável de ambiente; por padrão, assume `/opt/zeek/bin/zeek`.
- o `if [[ -x "$ZEEKBIN"]]` garante que o binário é executável antes de tentar iniciar o serviço.
- o comando é executado via `bash -c`, alterando o diretório atual para `$ZEEKDIR` (`cd '$ZEEKDIR'`) e forçando o Zeek a escrever seus logs nesse mesmo diretório através do parâmetro `Log::default_logdir='$ZEEKDIR'`.

- `-i "$IFACE"`: mesma interface usada pelo `tcpdump`, mantendo coerência entre captura e análise.
- a saída de *startup* é redirecionada para `$LOGDIR/zeek-startup-$TS.log`, facilitando *debug* em caso de erro de configuração.
- o PID é salvo em `pids/zeek.pid`, permitindo o encerramento controlado.

Caso o Zeek não seja encontrado no caminho configurado, o script emite um `[WARN]` indicando que a análise *live* não será iniciada, podendo ser feita posteriormente de forma *offline* com base no `.pcap`.

7.7.1.5 Execução do Suricata em modo *live*

O terceiro componente inicializado é o Suricata, também em modo *live*:

```
if command -v suricata >/dev/null 2>&1; then
  echo "[*] Iniciando Suricata live (logdir=$SURICATADIR)"
  SURICATA_LOG="$LOGDIR/suricata-startup-${TS}.log"
  nohup sudo suricata -c /etc/suricata/suricata.yaml -i "$IFACE" -l "
    $SURICATADIR" \
    &> "$SURICATA_LOG" &
  echo $! > "$PIDDIR/suricata.pid"
  ...
fi
```

- `command -v suricata`: verifica se o binário do Suricata está disponível no `PATH`.
- `-c /etc/suricata/suricata.yaml`: define o arquivo de configuração principal do Suricata.
- `-i "$IFACE"`: utiliza a mesma interface de monitoramento configurada para o Zeek e o `tcpdump`.
- `-l "$SURICATADIR"`: define o diretório de saída para os logs, incluindo o arquivo `eve.json`, que posteriormente será ingerido pelo Splunk.
- toda a saída de *startup* é registrada em `$LOGDIR/suricata-startup-$TS.log`, sendo o PID armazenado em `pids/suricata.pid`.

Se o Suricata não estiver instalado, o script emite uma mensagem informando que a captura *live* será pulada, com possibilidade de análise *offline* a partir dos PCAPs.

7.7.1.6 Resumo e uso nos experimentos

Ao final da execução, o script imprime um resumo com os caminhos utilizados:

- arquivo `.pcap` gerado (`pcap/$RUN_ID.pcap`);
- diretório de logs do Zeek (`zeek/`);
- diretório de logs do Suricata (`suricata/`);

- arquivo de metadados `meta_run.txt`.

Dessa forma, cada *run* de experimento (associado a um determinado modo criptográfico, perfil de carga e alvo) possui um conjunto completo de artefatos de monitoramento, organizados de forma consistente, facilitando:

- a ingestão *one-shot* de logs no Splunk;
- a comparação entre cenários (`classico` vs. `hibrido`);
- a reprodução de resultados e auditoria do ambiente.

7.7.2 Script `collect_monitor.sh`

Após a execução de um Ato de exfiltração/conexão SSH, é necessário encerrar os sensores de monitoramento e consolidar os artefatos gerados (PCAP, logs do Zeek, logs do Suricata, metadados, hashes) em um diretório único, pronto para análises e ingestão no Splunk. Essa tarefa é automatizada pelo script `collect_monitor.sh`, apresentado a seguir.

```
#!/usr/bin/env bash
# collect_monitor.sh (FINAL)
# Uso: ./collect_monitor.sh <interface> <attacker_ip> <target_ip> <mode> <
    profile> <label>
# Ex.: ./collect_monitor.sh eth0 192.168.50.20 192.168.50.40 classico leve
    sshreal
set -euo pipefail

IFACE="${1:-eth0}"
ATTACKER_IP="${2:-192.168.50.20}"
TARGET_IP="${3:-192.168.50.40}"
MODE="${4:-classico}"
PROFILE="${5:-leve}"
LABEL="${6:-target}"

BASE="$HOME/monitoramento1/$MODE/exfil_compare/$LABEL/$PROFILE"
PCAPDIR="$BASE/pcap"
PIDDIR="$BASE/pids"
LOGDIR="$BASE/logs"
ZEEKDIR="$BASE/zeek"
SURICATADIR="$BASE/suricata"

TS="$(date +%Y%m%d-%H%M%S)"
COLLECT_DIR="$BASE/collected_$TS"

ZEEKBIN="${ZEEKBIN:-/opt/zeek/bin/zeek}"
SURICATABIN="$(command -v suricata || true)"
SPLUNK_BIN_DEFAULT="/opt/splunk/bin/splunk"

mkdir -p "$COLLECT_DIR"

echo "[*] collect_monitor: BASE=$BASE"
echo
```

```
# 1) parar processos
stop_pid(){
  local f="$1"
  local name="$2"
  if [[ -f "$f" ]]; then
    local pid; pid="$(cat "$f")"
    echo "[*] Parando $name (pid $pid) ..."
    sudo kill "$pid" >/dev/null 2>&1 || true
    sleep 0.5
  fi
}
stop_pid "$PIDDIR/tcpdump.pid" "tcpdump"
stop_pid "$PIDDIR/zeek.pid" "zeek"
stop_pid "$PIDDIR/suricata.pid" "suricata"

# 2) copiar artefatos do Ato (apenas deste Ato)
echo "[*] Copiando artefatos deste Ato para $COLLECT_DIR ..."
mkdir -p "$COLLECT_DIR/pids" "$COLLECT_DIR/logs" "$COLLECT_DIR/pcap"

# pcap
if compgen -G "$PCAPDIR/*.pcap" > /dev/null; then
  cp -a "$PCAPDIR/." "$COLLECT_DIR/pcap/" 2>/dev/null || true
fi

# logs locais (texto, startups)
cp -a "$LOGDIR/." "$COLLECT_DIR/logs/" 2>/dev/null || true

# pids (conteúdo, sem criar pids/pids)
cp -a "$PIDDIR/." "$COLLECT_DIR/pids/" 2>/dev/null || true

# meta
[[ -f "$BASE/meta_run.txt" ]] && cp "$BASE/meta_run.txt" "$COLLECT_DIR/" || true

# 3) Zeek live logs (só do diretório do Ato)
if [[ -d "$ZEEKDIR" && -n "$(ls -A "$ZEEKDIR" 2>/dev/null || true)" ]]; then
  cp -a "$ZEEKDIR/." "$COLLECT_DIR/zeek_logs/" 2>/dev/null || true
fi

# 4) Suricata live logs (só do diretório do Ato)
if [[ -d "$SURICATADIR" && -n "$(ls -A "$SURICATADIR" 2>/dev/null || true)" ]];
    then
  cp -a "$SURICATADIR/." "$COLLECT_DIR/suricata_logs/" 2>/dev/null || true
fi

# 5) sha256 dos PCAPs (integridade)
if compgen -G "$COLLECT_DIR/pcap/*.pcap" > /dev/null; then
  echo "[*] Calculando sha256 dos PCAPs ..."
  mkdir -p "$COLLECT_DIR/hashes"
  for p in "$COLLECT_DIR"/pcap/*.pcap; do
    sha256sum "$p" > "$COLLECT_DIR/hashes/$(basename "$p").sha256" || true
```

```
  done
fi

# 6) Fallback: Zeek offline APENAS se não existir zeek_logs ou estiver vazio
need_zeek_offline=0
if [[ ! -d "$COLLECT_DIR/zeek_logs" || -z "$(ls -A "$COLLECT_DIR/zeek_logs" 2>/
    dev/null || true)" ]]; then
  need_zeek_offline=1
fi

if [[ $need_zeek_offline -eq 1 ]]; then
  if [[ -x "$ZEEKBIN" ]] && compgen -G "$COLLECT_DIR/pcap/*.pcap" > /dev/null;
    then
    echo "[*] Zeek live ausente/vazio -> processando PCAPs com Zeek offline ..."
    for pcap in "$COLLECT_DIR"/pcap/*.pcap; do
      [[ -f "$pcap" ]] || continue
      outdir="$COLLECT_DIR/zeek_offline_$(basename "$pcap" .pcap)"
      mkdir -p "$outdir"
      ( cd "$outdir" && sudo "$ZEEKBIN" -r "$pcap" local ) \
        || echo "[WARN] Zeek offline falhou para $pcap"
    done
  else
    echo "[WARN] Zeek offline não pôde rodar (binário ausente ou não há PCAP)."
  fi
else
  echo "[*] Zeek live presente; offline não necessário."
fi

# 7) Fallback: Suricata offline APENAS se não existir suricata_logs ou estiver
    vazio
need_suricata_offline=0
if [[ ! -d "$COLLECT_DIR/suricata_logs" || -z "$(ls -A "$COLLECT_DIR/
    suricata_logs" 2>/dev/null || true)" ]]; then
  need_suricata_offline=1
fi

if [[ $need_suricata_offline -eq 1 ]]; then
  if [[ -n "$SURICATABIN" ]] && compgen -G "$COLLECT_DIR/pcap/*.pcap" > /dev/
    null; then
    echo "[*] Suricata live ausente/vazio -> processando PCAPs com Suricata
    offline ..."
    for pcap in "$COLLECT_DIR"/pcap/*.pcap; do
      [[ -f "$pcap" ]] || continue
      outdir="$COLLECT_DIR/suricata_offline_$(basename "$pcap" .pcap)"
      mkdir -p "$outdir"
      sudo "$SURICATABIN" -r "$pcap" -l "$outdir" -c /etc/suricata/suricata.yaml
     \
        || echo "[WARN] Suricata offline falhou para $pcap"
    done
  else
    echo "[WARN] Suricata offline não pôde rodar (binário ausente ou não há PCAP
```

```
    )."
  fi
else
  echo "[*] Suricata live presente; offline não necessário."
fi

# 8) Gerar script de ingestão do Splunk dentro do COLLECT_DIR (dedup por SHA,
    maxdepth, move p/ ingested/)
INGEST_SH="$COLLECT_DIR/splunk_ingest.sh"
cat > "$INGEST_SH" <<'SPLUNK_INGEST'
#!/usr/bin/env bash
set -euo pipefail

SPLUNK_BIN="${SPLUNK_BIN:-/opt/splunk/bin/splunk}"
ADMIN_AUTH="${ADMIN_AUTH:-}"   # opcional: admin:Senha
RUN_ID="unknown_run"
[[ -f meta_run.txt ]] && RUN_ID="$(grep -m1 '^RUN_ID=' meta_run.txt | cut -d'='
    -f2- || echo "$RUN_ID")"

INGESTED_DB=".ingested.hashes"     # armazena "sha256  caminho"
mkdir -p ingested
touch "$INGESTED_DB"

log(){ printf '%s %s\n' "$(date -Iseconds)" "$*"; }
file_sha(){ sha256sum "$1" | awk '{print $1}'; }

oneshot(){
  local f="$1" idx="$2" st="$3" host="$4"
  if [[ -z "$ADMIN_AUTH" ]]; then
    "$SPLUNK_BIN" add oneshot "$f" -index "$idx" -sourcetype "$st" -host "$host"
  else
    "$SPLUNK_BIN" add oneshot "$f" -index "$idx" -sourcetype "$st" -host "$host"
     -auth "$ADMIN_AUTH"
  fi
}

# varre só até profundidade 3 para evitar reprocessar coleções antigas
mapfile -t files < <(find . -maxdepth 3 -type f \
  \( -name 'conn.log' -o -name 'dns.log' -o -name 'http.log' -o -name 'files.log
    ' -o -name 'weird.log' -o -name 'notice.log' -o -name 'eve.json' -o -name '
    *.txt' \) -print)

if [[ ${#files[@]} -eq 0 ]]; then
  log "Nenhum arquivo a indexar."
  exit 0
fi

for f in "${files[@]}"; do
  f="${f#./}"
  sha="$(file_sha "$f")"
  if grep -q "^${sha} " "$INGESTED_DB" 2>/dev/null; then
```

```
    log "Ignorando $f (já ingestado)."
    continue
  fi

  base="$(basename "$f")"
  case "$base" in
    conn.log|dns.log|http.log|files.log|weird.log|notice.log)
      idx="zeek"; st="zeek:${base%.log}";;
    eve.json)
      idx="suricata"; st="suricata:eve";;
    *.txt)
      idx="experiments"; st="metadata";;
    *)
      idx="experiments"; st="generic";;
  esac

  log "Ingesting $f -> index=$idx sourcetype=$st host=$RUN_ID"
  if oneshot "$f" "$idx" "$st" "$RUN_ID"; then
    printf '%s %s\n' "$sha" "$f" >> "$INGESTED_DB"
    # mover mantendo subpastas
    mkdir -p "ingested/$(dirname "$f")" 2>/dev/null || true
    if mv -f "$f" "ingested/$f" 2>/dev/null; then
      log "Ingestado e movido: ingested/$f"
    else
      touch "$f.ingested"
      log "Ingestado; marcação criada: $f.ingested"
    fi
  else
    log "Falha ao ingestar $f"
  fi
done

log "Ingestão completa."
SPLUNK_INGEST
chmod +x "$INGEST_SH" || true

echo
echo "[OK] Coleta finalizada: $COLLECT_DIR"
echo " Para indexar no Splunk (no mesmo host):  ADMIN_AUTH='admin:SENHA' \"
    $INGEST_SH\""
```

7.7.2.1 Objetivo e parâmetros

O `collect_monitor.sh` é o par complementar do `prepare_monitor.sh`. Enquanto o primeiro inicia a captura e os sensores em tempo real, este segundo script:

- encerra com segurança os processos do `tcpdump`, Zeek e Suricata relacionados àquele Ato;
- copia os artefatos gerados para um diretório de coleta (`collected_$TS`);
- garante a integridade dos PCAPs via `sha256`;

- executa Zeek e Suricata em modo *offline* se os logs *live* estiverem ausentes;
- gera um script auxiliar (`splunk_ingest.sh`) para ingestão controlada dos logs no Splunk, com deduplicação por `sha256`.

Os parâmetros de linha de comando são os mesmos do `prepare_monitor.sh`:

- `<interface>` (`IFACE`): interface de captura usada no Ato;
- `<attacker_ip>` (`ATTACKER_IP`): IP do atacante;
- `<target_ip>` (`TARGET_IP`): IP do alvo (SSH real);
- `<mode>` (`MODE`): modo criptográfico (`classico`, `hibrido`);
- `<profile>` (`PROFILE`): perfil de carga (`leve`, `medio`, `pesado`);
- `<label>` (`LABEL`): rótulo lógico do alvo (`sshreal`, `cowrie`, etc.).

Isso garante que `prepare_monitor.sh` e `collect_monitor.sh` sempre operem sobre a mesma árvore de diretórios (`$HOME/monitoramento1/$MODE/exfil_compare/$LABEL/$PROFILE`).

7.7.2.2 Etapas principais do `collect_monitor.sh`

1) Encerramento controlado dos processos de monitoramento. A função `stop_pid` lê o PID armazenado em cada arquivo dentro do diretório `pids` e envia um `kill` para o processo correspondente:

- `tcpdump.pid`: encerra a captura de pacotes;
- `zeek.pid`: encerra o Zeek em modo *live*;
- `suricata.pid`: encerra o Suricata em modo *live*.

Isso evita processos órfãos consumindo CPU e disco após o término do Ato.

2) Consolidação dos artefatos do Ato. O script cria um diretório de coleta exclusivo, com sufixo de timestamp:

- `COLLECT_DIR = $BASE/collected_$TS`

Em seguida, copia:

- todos os `.pcap` de `pcap/` para `COLLECT_DIR/pcap/`;
- os arquivos de *log* locais (`logs/`) para `COLLECT_DIR/logs/`;
- os arquivos de PID (`pids/`) para `COLLECT_DIR/pids/`;
- o arquivo de metadados `meta_run.txt` para a raiz de `COLLECT_DIR`.

Os logs produzidos diretamente pelo Zeek e Suricata em modo *live* (`$ZEEKDIR` e `$SURICATADIR`) são copiados, se existirem, para:

- `COLLECT_DIR/zeek_logs/`;
- `COLLECT_DIR/suricata_logs/`.

3) Cálculo de `sha256` dos PCAPs. Para cada `.pcap` em `COLLECT_DIR/pcap/`, o script gera um arquivo `.sha256` correspondente em `COLLECT_DIR/hashes/`, registrando o hash de integridade. Isso permite:

- verificar se o arquivo não foi alterado entre coletas e análises;
- detectar duplicações acidentais de PCAPs em diferentes execuções.

4) Fallback Zeek *offline*. Se o diretório `zeek_logs` estiver ausente ou vazio, o script assume que o Zeek *live* não produziu logs e aciona um processamento *offline*:

- para cada `.pcap`, é criado um diretório `zeek_offline_<nome-do-pcap>` dentro de `COLLECT_DIR`;
- o Zeek é executado com `-r <pcap>` a partir desse diretório, gerando os logs a partir do traço capturado.

Dessa forma, mesmo que o Zeek não tenha conseguido operar em modo *live*, os artefatos de alto nível (`conn.log`, `dns.log`, etc.) ainda são produzidos, o que é essencial para as análises comparativas.

5) Fallback Suricata *offline*. O mesmo raciocínio é aplicado ao Suricata:

- se `suricata_logs` estiver ausente ou vazio, cada `.pcap` é processado em modo *offline* com o comando `suricata -r <pcap>`, gerando logs em `suricata_offline_<nome-do-pcap>`.

Isso garante a geração do `eve.json` e de demais arquivos de log, mesmo quando o Suricata não esteve ativo em tempo real.

7.7.3 Script de ingestão para o Splunk (`splunk_ingest.sh`)

Na etapa final, o `collect_monitor.sh` cria, dentro do `COLLECT_DIR`, um script auxiliar chamado `splunk_ingest.sh`. Esse script é responsável por:

- localizar automaticamente arquivos relevantes para análise (`conn.log`, `dns.log`, `http.log`, `files.log`, `weird.log`, `notice.log`, `eve.json`, arquivos `.txt` de metadados);
- deduplicar a ingestão usando um banco local de hashes (`.ingested.hashes`);
- enviar os arquivos ao Splunk via comando `splunk add oneshot`, direcionando-os para índices e *sourcetypes* específicos;
- mover arquivos já ingeridos para uma subárvore `ingested/`, mantendo a organização original.

O `splunk_ingest.sh` infere o `RUN_ID` a partir do `meta_run.txt` (quando presente) e o utiliza como `host` na ingestão, permitindo que todas as entradas daquele Ato sejam identificadas como pertencentes ao mesmo experimento.

Para utilizar esse script, no diretório de coleta correspondente, basta executar, por exemplo:

```
cd "$COLLECT_DIR"
ADMIN_AUTH='admin:SENHA' ./splunk_ingest.sh
```

onde `admin:SENHA` corresponde às credenciais do administrador do Splunk configuradas anteriormente.

Parte IV

Infraestrutura de Rede e Switch

Capítulo 8

Configurações de Rede

8.1 Endereçamento e estrutura da rede isolada

Nesta seção é descrita a estrutura lógica da rede isolada utilizada no laboratório, bem como o endereçamento IP estático configurado em cada máquina participante do ambiente de testes.

8.1.1 Topologia lógica e endereçamento IP

A Tabela 8.1 resume o esquema de endereçamento adotado na rede isolada do experimento.

Tabela 8.1: Endereçamento da rede isolada

Dispositivo	**IP estático**	**Função**
Máquina Atacante	`192.168.50.20/24`	Geração dos ataques
Máquina Monitor	`192.168.50.30/24`	Análise e defesa
Servidor SSH	`192.168.50.40/24`	Alvo dos ataques
Gateway fictício	`192.168.50.1`	Referência de gateway na rede isolada (sem roteamento externo)

8.2 Servidor SSH (Debian 12) – IP estático via `/etc/network/interfaces`

No servidor SSH (Debian), a configuração de IP estático foi realizada de forma direta no arquivo `/etc/network/interfaces`. Essa abordagem utiliza o mecanismo clássico `ifup-down`, dispensando ferramentas como NetworkManager para a interface de laboratório, o que torna o comportamento mais previsível em um ambiente de servidor.

8.2.1 Edição do arquivo `/etc/network/interfaces`

1. Abrir o arquivo de configuração de rede com privilégios de superusuário:

```
sudo nano /etc/network/interfaces
```

2. Ajustar o conteúdo para definir o *loopback* e a interface cabeada com IP estático. As demais linhas podem ser comentadas ou removidas, mantendo apenas a configuração relevante:

```
auto lo
iface lo inet loopback

auto enp2s0
iface enp2s0 inet static
    address 192.168.50.40/24
    gateway 192.168.50.1
    dns-nameservers 1.1.1.1 8.8.8.8
```

   - `enp2s0` é a interface Ethernet conectada ao switch utilizado no laboratório. Caso o nome da interface seja diferente, deve-se ajustá-lo conforme o resultado do comando `ip a`.
   - `address` define o IP estático do servidor SSH real na sub-rede `192.168.50.0/24`.
   - `gateway` indica o roteador/gateway do segmento `192.168.50.0/24` (`192.168.50.1`), permitindo que o servidor alcance outras redes quando necessário.
   - `dns-nameservers` especifica servidores DNS públicos, suficientes para resolução de nomes quando houver acesso à Internet (por exemplo, durante atualizações ou instalação de pacotes).

3. Salvar o arquivo (`Ctrl+O`, `Enter`) e sair do editor (`Ctrl+X`).

4. Reiniciar o serviço de rede ou a máquina para aplicar as alterações:

```
sudo systemctl restart networking
#ou
sudo reboot
```

5. Após o reinício, confirmar o endereço configurado:

```
ip a
```

   A interface `enp2s0` deve exibir:

```
inet 192.168.50.40/24
```

6. Opcionalmente, testar a conectividade com as demais máquinas assim que elas estiverem configuradas, utilizando `ping` ou `ssh`.

Essa configuração torna o servidor SSH um alvo estável e previsível para os testes, evitando variações de endereço que poderiam comprometer os scripts de coleta e a correlação de eventos.

8.3 Máquina Atacante e Máquina de Monitoramento (Kali Linux / Kali Purple)

Nas máquinas Kali (atacante e monitoramento), a configuração de IP estático foi realizada por meio do NetworkManager, utilizando um perfil dedicado `LabControlado`. Esse modelo permite:

- manter uma configuração estática consistente para a rede isolada;
- alternar rapidamente para um perfil de Internet via DHCP, quando necessário, sem reescrever arquivos de configuração;
- centralizar a gerência de rede em um mecanismo suportado pelo ambiente gráfico e pelos utilitários do Kali.

8.3.1 Identificação da interface de rede

1. Listar as interfaces de rede:

```
ip a
```

2. Localizar a interface cabeada conectada ao switch da rede isolada, normalmente marcada com `LOWER_UP`, por exemplo:

```
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
```

3. Anotar o nome exato da interface (por exemplo, `enp2s0` ou `eth0`), pois ele será utilizado no arquivo de configuração do NetworkManager e em comandos de validação (`arp-scan`, `nmap`, etc.).

8.3.2 Criação/edição do perfil `LabControlado` (IP estático)

1. Abrir (ou criar) o arquivo de conexão do NetworkManager:

```
sudo nano /etc/NetworkManager/system-connections/LabControlado.nmconnection
```

2. Inserir o conteúdo padrão do perfil, ajustando o nome da interface e o endereço IP conforme o papel da máquina:

```
[connection]
id=LabControlado
type=ethernet
interface-name=eth0   # substituir pelo nome correto (ex.: enp2s0)
autoconnect=true

[ipv4]
method=manual
address1=192.168.50.30/24,192.168.50.1
```

```
dns=1.1.1.1;
ignore-auto-dns=true

[ipv6]
method=ignore
```

- Em `interface-name`, substituir `eth0` pelo nome real identificado na etapa anterior (por exemplo, `enp2s0`).
- Em `address1`, ajustar o endereço IP conforme o host:
  - Máquina atacante: `192.168.50.20/24,192.168.50.1`
  - Máquina de monitoramento: `192.168.50.30/24,192.168.50.1`
- O campo `dns` define o servidor DNS desejado (no exemplo, `1.1.1.1`), suficiente para resolução de nomes em situações em que a máquina tenha saída para a Internet.
- O parâmetro `ignore-auto-dns=true` impede que configurações de DNS oferecidas por DHCP sobrescrevam essa definição, preservando a previsibilidade do ambiente.
- `[ipv6] method=ignore` desativa IPv6, simplificando a análise e evitando confusão com endereços não utilizados nos testes.

3. Salvar as alterações e sair do editor.

8.3.3 Ajuste de permissões do arquivo de conexão

O NetworkManager exige que arquivos `.nmconnection` sejam de propriedade do usuário `root` e tenham permissão `600`. Caso essas permissões não sejam respeitadas, o perfil pode ser ignorado ou marcado como inválido.

```
sudo chmod 600 /etc/NetworkManager/system-connections/LabControlado.nmconnection
sudo chown root:root /etc/NetworkManager/system-connections/LabControlado.
    nmconnection
```

Esses comandos garantem que apenas o `root` possa ler e modificar o perfil de conexão, atendendo aos requisitos de segurança do NetworkManager.

8.3.4 Recarregar o NetworkManager e ativar o perfil estático

1. Recarregar as conexões conhecidas pelo NetworkManager:

```
sudo nmcli connection reload
```

2. Listar as conexões disponíveis:

```
nmcli connection show
```

O perfil `LabControlado` deve aparecer associado ao tipo `ethernet` e à interface selecionada.

3. Ativar o perfil na interface de rede:

```
sudo nmcli connection up LabControlado
```

4. Confirmar que o IP estático foi aplicado:

```
ip a
```

A interface deve exibir o endereço configurado, por exemplo, na máquina de monitoramento:

```
inet 192.168.50.30/24
```

5. Na máquina atacante, o procedimento é idêntico, variando apenas o endereço IP (`192.168.50.20/24`).

8.4 Perfis para acesso à Internet (DHCP) e recuperação de rede

Em alguns momentos (instalação de pacotes, atualizações etc.), pode ser necessário conectar as máquinas Kali à Internet fora da rede isolada. Para evitar alterações manuais frequentes nos arquivos de configuração, é conveniente criar um segundo perfil, baseado em IPv4 via DHCP.

8.4.1 Criação de um perfil DHCP (`Internet-dhcp`)

1. Criar o perfil uma única vez, associando-o à mesma interface física:

```
sudo nmcli connection add type ethernet con-name Internet-dhcp ifname
    enp2s0 ipv4.method auto
```

- Substituir `enp2s0` pelo nome real da interface usada para acesso externo, se diferente.
- O método `ipv4.method auto` indica que o IP será obtido via DHCP, recebendo endereço, gateway e DNS automaticamente do roteador.

8.4.2 Alternar entre a rede isolada e a Internet

Uma vez definidos os dois perfis, a alternância entre eles torna-se simples:

- Voltar à Internet (perfil DHCP):

```
sudo nmcli connection up Internet-dhcp
```

- Voltar à rede isolada (perfil estático):

```
sudo nmcli connection up LabControlado
```

Essa abordagem evita que ajustes da rede externa interfiram na topologia controlada do laboratório, e vice-versa.

8.4.3 Recuperação rápida via DHCP

Caso o perfil estático apresente problemas de conectividade e seja necessário restaurar provisoriamente o acesso à rede via DHCP (por exemplo, para recuperar pacotes ou corrigir configurações), pode-se usar:

```
sudo dhclient enp2s0
```

Substituir `enp2s0` pela interface correta, se necessário. Esse comando solicita um endereço IP temporário ao servidor DHCP, útil para operações de manutenção e correção sem alterar permanentemente o perfil `LabControlado`.

8.5 Validação da conectividade na rede isolada

Após configurar os IPs estáticos, é fundamental validar se as máquinas estão se enxergando corretamente dentro da sub-rede `192.168.50.0/24`. Essa validação garante que o tráfego gerado pelos atos de experimento efetivamente circulará pelo switch e poderá ser capturado pela máquina de monitoramento.

8.5.1 Verificar IP local

Na máquina de monitoramento (Kali Purple), confirmar o endereço configurado:

```
ip a
```

A interface utilizada no laboratório deve exibir:

```
inet 192.168.50.30/24
```

Se o endereço não corresponder ao esperado, é necessário revisar o perfil `LabControlado` e garantir que ele esteja ativo.

8.5.2 Testar conectividade com o servidor SSH e com a máquina atacante

Os testes a seguir assumem que o servidor SSH real e a máquina atacante já estão ligados e configurados com IPs estáticos:

- Testar comunicação com o servidor SSH (`192.168.50.40`):

```
ping -c 4 192.168.50.40
```

A saída esperada inclui respostas ICMP com tempos de latência da ordem de milissegundos. Perdas de pacotes ou ausência de resposta indicam problemas de cabeamento, IP incorreto ou firewall.

- Testar comunicação com a máquina atacante (`192.168.50.20`):

```
ping -c 4 192.168.50.20
```

Se ambos os testes forem bem-sucedidos, a conectividade básica entre os três nós do laboratório está assegurada.

8.5.3 Verificação opcional com ferramentas de varredura

Para confirmar que os dispositivos aparecem na rede e que as portas básicas estão acessíveis (por exemplo, porta 22/TCP para SSH), podem ser utilizados comandos como:

```
sudo nmap -sP 192.168.50.0/24
```

ou, alternativamente:

```
sudo arp-scan --interface=enp2s0 192.168.50.0/24
```

Substituir `enp2s0` pela interface correta, se necessário. Essas verificações complementares ajudam a confirmar que todos os nós da rede isolada (servidor SSH, atacante e monitor) estão visíveis, com endereços coerentes, e aptos a trocar tráfego que será capturado e analisado pelos sensores de monitoramento (tcpdump, Zeek, Suricata e Splunk).

Capítulo 9

Configuração do Switch Gerenciável

9.1 Port-mirroring no switch TP-Link TL-SG105E

9.1.1 Objetivo do espelhamento de porta

Nesta seção é descrita a configuração de *port mirroring* no switch gerenciável TP-Link TL-SG105E, de modo que a máquina de monitoramento (*Kali Purple* / Blue Team) receba, de forma passiva, todo o tráfego entre o servidor SSH (*sshserver*) e a máquina atacante. O objetivo é permitir o monitoramento contínuo e a análise forense do tráfego gerado nos experimentos, sem interferir diretamente na comunicação entre as máquinas.

9.1.2 Pré-requisitos e mapeamento de portas

Antes da configuração lógica do *port mirroring*, é necessário garantir que a conexão física das portas do switch esteja de acordo com o layout definido para o laboratório. A Tabela 9.1 apresenta o mapeamento adotado.

Tabela 9.1: Mapeamento de portas do switch TL-SG105E no laboratório

Porta	**Dispositivo**	**Função**
1	Servidor SSH	Máquina alvo dos ataques
2	Máquina Atacante	Geração e envio dos ataques
3	Máquina de Monitoramento	Porta de espelhamento (RX+TX) para análise de tráfego

Adicionalmente, considera-se que:

- A interface de rede da máquina de monitoramento (por exemplo, `eth0` no Kali Purple) esteja configurada com IP estático na rede isolada (por exemplo, `192.168.50.30/24`).
- O acesso ao painel de administração do switch seja realizado pelo endereço `http://192.168.0.1`.

Em ambiente controlado e sem acesso à internet, pode ser necessário conectar temporariamente a interface de rede do Kali Purple diretamente a uma porta do switch e atribuir manualmente um endereço IP na faixa `192.168.0.0/24` (por exemplo, `192.168.0.2`) para acessar a interface de gerenciamento do equipamento.

9.1.3 Acesso ao painel de administração do switch

1. No Kali Purple (máquina de monitoramento), abrir o navegador web de preferência.
2. Acessar o endereço de gerenciamento do switch:

```
http://192.168.0.1
```

3. Efetuar login com as credenciais de administração configuradas. Caso o equipamento esteja com configuração de fábrica, o par padrão costuma ser `admin` / `admin`, devendo ser alterado em um contexto de produção.

9.1.4 Configuração do port-mirroring (porta 3 como destino)

Uma vez autenticado no painel do TL-SG105E, a configuração de *port mirroring* é realizada por meio do menu de monitoramento.

9.1.4.1 Ativar o port-mirroring e definir a porta de destino

1. Navegar até o menu de espelhamento de portas, geralmente localizado em:

   `Monitoring` → `Port Mirror`

2. Na seção **Port Mirror**:
   - Definir a opção de espelhamento como `Enable`.
   - Em `Mirroring Port`, selecionar a **Porta 3**, correspondente à porta física na qual está conectada a máquina de monitoramento (Kali Purple).
   - Confirmar as alterações clicando em `Apply`.

9.1.4.2 Definir as portas a serem espelhadas (fontes de tráfego)

Em seguida, é necessário indicar quais portas terão seu tráfego copiado (*mirrored*) para a porta de monitoramento (Porta 3).

1. Na seção **Mirrored Port**, configurar as portas que conectam o servidor SSH e a máquina atacante:
   - Para a **Porta 1** (Servidor SSH):
     - `Ingress`: `Enable` (espelhar tráfego de entrada).
     - `Egress`: `Enable` (espelhar tráfego de saída).
   - Para a **Porta 2** (Máquina Atacante):
     - `Ingress`: `Enable`.
     - `Egress`: `Enable`.
2. Após configurar ambas as portas, clicar em `Apply` para salvar as alterações.

9.1.5 Validação da configuração de espelhamento

Com essa configuração, todo o tráfego de entrada e saída das portas 1 (Servidor SSH) e 2 (Máquina Atacante) será copiado para a porta 3, onde está conectada a máquina de monitoramento. Dessa forma, ferramentas como Zeek, Suricata, *tcpdump* e Wireshark podem capturar e analisar, de forma passiva, o tráfego completo entre atacante e alvo, sem introduzir interferência direta na comunicação original.

Parte V

Execução dos Cenários

Capítulo 10

Cenário Payload Probe

Objetivo do cenário

O script `ssh_payload_probe.sh` implementa um cenário controlado de geração de *payload* real, medição local de métricas criptográficas (clássicas e pós-quânticas/híbridas) e execução de uma sonda de negociação SSH contra o servidor `sshreal`. Os artefatos gerados (dados, chaves, *logs*, métricas e resumos) são organizados em uma estrutura de diretórios por modo e perfil, servindo como exemplo prático de aplicação e instrumentação de criptografia clássica e híbrida no contexto deste laboratório.

10.1 Forma de execução e parâmetros

10.1.0.1 Sintaxe geral

```
./ssh_payload_probe.sh <mode> <profile> <target_ip> <target_port> <ssh_user/
   label> <target_kind> <ssh_password>
```

- `<mode>`:
  - `classico`: o script força a negociação de KEX puramente clássico (`curve25519-sha256`), servindo como linha de base.
  - `hibrido`: o script oferta KEX híbrido (`mlkem768x25519-sha256`, quando disponível) além do clássico, permitindo comparar o impacto do componente pós-quântico.
- `<profile>`:
  - `leve`: simula muitas unidades pequenas de dados (1000 arquivos de 1 MiB), aproximando um fluxo fragmentado.
  - `medio`: 100 arquivos de 10 MiB, carga intermediária.
  - `pesado`: 10 arquivos de 100 MiB, carga concentrada, útil para observar impacto em CPU, memória e I/O.
- `<target_ip>`:

    - Endereço IP do servidor SSH alvo (por exemplo, `192.168.50.40`), conforme a configuração de rede isolada.
- `<target_port>`:
    - Porta TCP em que o `sshd` recompilado está escutando (por exemplo, `22222`), permitindo isolar o experimento do SSH padrão da distribuição.
- `<ssh_user/label>`:
    - Rótulo do alvo, usado também como nome de usuário SSH (tipicamente `sshserver`), permitindo registrar no meta qual alvo foi usado sem expor detalhes de IP/porta isoladamente.
- `<target_kind>`:
    - Identifica o tipo de alvo; neste script deve ser obrigatoriamente `sshreal`, garantindo que o experimento está sendo conduzido contra o servidor real definido.
- `<ssh_password>`:
    - Senha do usuário SSH no servidor alvo, usada apenas no passo final para enviar uma notificação via `wall` (se `sshpass` estiver disponível). Não é usada na sonda de KEX.

10.1.0.2 Exemplos de execução

- **Parâmetros típicos do projeto**:
    - IP do servidor: `192.168.50.40`.
    - Porta SSH: `22222`.
    - Usuário: `sshserver`.
    - Tipo do alvo: `sshreal`.
    - Senha: simbolizada como `SENHA_SSH`.

Modo clássico:

```
./ssh_payload_probe.sh classico leve   192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
./ssh_payload_probe.sh classico medio  192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
./ssh_payload_probe.sh classico pesado 192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
```

Modo híbrido:

```
./ssh_payload_probe.sh hibrido leve   192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
./ssh_payload_probe.sh hibrido medio  192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
./ssh_payload_probe.sh hibrido pesado 192.168.50.40 22222 sshserver sshreal
   SENHA_SSH
```

- Cada execução gera um diretório `run-<profile>-<TS>` distinto, facilitando a comparação `classico` vs `hibrido` para um mesmo perfil (`leve`, `medio` ou `pesado`).

10.1.1 Bloco 0 — Cabeçalho, função auxiliar e argumentos

```
#!/usr/bin/env bash
# ssh_payload_probe.sh - Payload REAL + Sonda SSH + Métricas Comparativas (clá
    ssico x híbrido)
# Uso obrigatório:
#   ./ssh_payload_probe.sh <mode> <profile> <target_ip> <target_port> <ssh_user/
    label> <target_kind> <ssh_password>
set -euo pipefail

now_ms() { date +%s%3N; }
```

- Define explicitamente o uso de `bash`, evitando ambiguidades com `/bin/sh`.
- Documenta no cabeçalho a finalidade exata do script e o formato obrigatório de uso, reduzindo risco de execução com parâmetros errados.
- `set -euo pipefail` endurece o comportamento:
  - `-e`: interrompe em erros não tratados.
  - `-u`: impede uso acidental de variáveis não definidas.
  - `pipefail`: garante que erros em *pipes* não sejam silenciosamente ignorados.
- A função `now_ms` padroniza a captura de timestamps em milissegundos, centralizando medições de tempo (especialmente em blocos criptográficos).

```
# ---------- 0. args ----------
if [[ $# -ne 7 ]]; then
  echo "Uso: $0 <mode> <profile> <target_ip> <target_port> <ssh_user/label> <
    target_kind> <ssh_password>" >&2
  exit 1
fi

MODE="$1"          # classico | hibrido
PROFILE="$2"       # leve | medio | pesado
TARGET_IP="$3"
TARGET_PORT="$4"
TARGET_LABEL="$5"  # será também o usuário SSH
TARGET_KIND="$6"   # deve ser sshreal
SSH_PASS="$7"
```

- Garante que a execução sempre receba exatamente sete argumentos, evitando estados parcialmente configurados.
- Associa cada argumento a uma variável com nome significativo, melhorando a legibilidade e reduzindo erros na manutenção.

```
if [[ "${MODE}" != "classico" && "${MODE}" != "hibrido" ]]; then
  echo "[ERRO] mode inválido: ${MODE}. Use 'classico' ou 'hibrido'." >&2
  exit 2
fi

if [[ "${TARGET_KIND}" != "sshreal" ]]; then
  echo "[ERRO] target_kind inválido: ${TARGET_KIND}. Este script só roda contra
    sshreal." >&2
  exit 2
fi

SSH_USER="${TARGET_LABEL}"
```

- Impõe explicitamente os modos válidos (`classico` e `hibrido`), o que simplifica a análise estatística pois só há dois cenários comparáveis.
- Restringe a execução ao alvo `sshreal`, evitando que o script seja usado por engano contra outras máquinas do laboratório.
- Usa `TARGET_LABEL` como `SSH_USER`, ligando o conceito lógico de "alvo" ao usuário SSH efetivamente utilizado na conexão.

10.1.2 Bloco 1 — Localização do OpenSSL, diretórios e metadados básicos

```
# ---------- 1. openssl e paths ----------
if [[ -x "/usr/local/openssl/bin/openssl" ]]; then
  OPENSSL_BIN="/usr/local/openssl/bin/openssl"
else
  OPENSSL_BIN="$(command -v openssl || true)"
fi
if [[ -z "${OPENSSL_BIN}" ]]; then
  echo "[ERRO] openssl não encontrado." >&2
  exit 3
fi
OQS_MODULE_PATH="/usr/local/lib/ossl-modules"
```

- Dá prioridade ao OpenSSL customizado em `/usr/local/openssl`, garantindo que as features pós-quânticas estejam disponíveis quando configuradas.
- Só recorre ao OpenSSL do sistema caso o customizado não exista, preservando compatibilidade em ambientes sem PQC.
- Se nenhum `openssl` for encontrado, aborta com mensagem clara, pois todo o fluxo criptográfico depende dele.
- Define `OQS_MODULE_PATH` como caminho padrão dos módulos OQS, o que facilita depuração e inspeção manual.

```
TS="$(date +%Y%m%d-%H%M%S)"
START_EPOCH=$(date +%s)

BASE_DIR="$HOME/ataques1/${MODE}/payload_probe/${TARGET_KIND}/${PROFILE}"
RUN_DIR="${BASE_DIR}/run-${PROFILE}-${TS}"
mkdir -p "${RUN_DIR}"

meta_append() { echo "$1" >> "${RUN_DIR}/meta-${TS}.txt"; }
```

- Usa `TS` (timestamp legível) e `START_EPOCH` (segundos desde Epoch) para identificar a rodada e medir duração total.
- Estrutura o diretório de execução de forma hierárquica:
  - `$HOME/ataques1/<mode>/payload_probe/sshreal/<profile>/run-<profile>-<TS>`.
  - Essa estrutura permite localizar facilmente os experimentos por modo (`classico/hibrido`) e perfil (`leve/medio/pesado`).
- A função `meta_append` centraliza o registro de metadados, evitando duplicação de lógica de escrita de arquivo.

10.1.3 Bloco 1.1 — Amostrador de CPU, memória e I/O (sampler)

```
# ---------- sampler infra ----------
SAMPLER_PID=""
SAMPLER_FILE=""
SAMPLER_RUNNING=0
```

- Inicializa o estado do amostrador, que será usado para acompanhar o consumo de recursos locais durante as fases mais pesadas do script (geração de payload, cifra, PQC).

```
_read_cpu_totals() {
  read -r _cpu user nice system idle iowait irq softirq steal guest guest_nice <
    /proc/stat
  total=$((user + nice + system + idle + iowait + irq + softirq + steal + guest
   + guest_nice))
  echo "${idle} ${total}"
}
```

- Lê a linha global de CPU em `/proc/stat` e calcula o tempo total de CPU consumido e o tempo de CPU ociosa.
- Esses dois valores são suficientes para estimar o percentual de uso de CPU em cada intervalo de amostragem.

```
_read_proc_utime_stime() {
  pid="$1"
  if [[ ! -r "/proc/${pid}/stat" ]]; then
    echo "0 0"
    return
  fi
  statline=$(</proc/"${pid}"/stat)
  rest=${statline#*)}
  set -- $rest
  utime=${12:-0}
  stime=${13:-0}
  echo "${utime} ${stime}"
}
```

- Lê `/proc/<pid>/stat` para extrair os tempos de CPU em modo usuário (`utime`) e sistema (`stime`) do processo indicado.
- Ao ser chamado com `$$`, monitora especificamente o consumo de CPU do próprio script ao longo do tempo.

```
start_sampler() {
  SAMPLER_FILE="${RUN_DIR}/sampler-${TS}.log"
  : > "${SAMPLER_FILE}"
  SAMPLER_RUNNING=1

  read prev_idle prev_total < <(_read_cpu_totals)
  read prev_proc_utime prev_proc_stime < <(_read_proc_utime_stime $$)
```

- Cria o arquivo `sampler-<TS>.log` dentro do `RUN_DIR` e o zera.
- Captura valores iniciais de CPU global e do processo para servir de base aos deltas subsequentes.

```
  (
    while :; do
      ts_ms=$(date +%s%3N)

      read idle total < <(_read_cpu_totals)
      total_delta=$((total - prev_total))
      idle_delta=$((idle - prev_idle))
      busy_delta=$((total_delta - idle_delta))

      read proc_utime proc_stime < <(_read_proc_utime_stime $$)
      proc_delta=$(((proc_utime + proc_stime) - (prev_proc_utime +
   prev_proc_stime)))
```

- Laço infinito em *background* que:
  - captura timestamp em milissegundos;

- calcula o delta de tempo de CPU total, ocioso e ocupado;
- calcula o delta de CPU do processo (`proc_delta`).

```
    if [[ $total_delta -gt 0 ]]; then
      cpu_sys_pct=$(awk -v b="$busy_delta" -v t="$total_delta" 'BEGIN{printf
"%.2f", (b*100)/t}')
      proc_cpu_pct=$(awk -v p="$proc_delta" -v t="$total_delta" 'BEGIN{printf
"%.2f", (p*100)/t}')
    else
      cpu_sys_pct="0.00"
      proc_cpu_pct="0.00"
    fi
```

- Converte os deltas em percentuais, tanto para a CPU global quanto para o processo monitorado, com duas casas decimais.
- Em caso de delta zero, evita divisão por zero e força percentuais a 0.

```
    mem_total=$(awk '/MemTotal:/ {print $2*1024}' /proc/meminfo)
    mem_avail=$(awk '/MemAvailable:/ {print $2*1024}' /proc/meminfo)
    mem_used=$((mem_total - mem_avail))
```

- Lê a memória total e disponível a partir de `/proc/meminfo`, calculando o uso de memória em bytes; isso permite relacionar tamanho de payload com pressão de memória.

```
    read_bytes=0
    write_bytes=0
    if [[ -r /proc/$$/io ]]; then
      read_bytes=$(awk '/read_bytes:/ {print $2; exit}' /proc/$$/io 2>/dev/
null || echo 0)
      write_bytes=$(awk '/write_bytes:/ {print $2; exit}' /proc/$$/io 2>/dev/
null || echo 0)
    fi
```

- Lê contadores de I/O do processo (`read_bytes`, `write_bytes`), úteis para correlacionar operações de disco (escrita dos blobs TAR e ciphertext) com impacto total de I/O.

```
    printf '%s %s %s %s %s %s %s %s\n' \
      "${ts_ms}" "${cpu_sys_pct}" "${proc_cpu_pct}" "${mem_used}" "${
read_bytes}" "${write_bytes}" "${total_delta}" "${proc_delta}" \
      >> "${SAMPLER_FILE}"

    prev_idle=${idle}
    prev_total=${total}
    prev_proc_utime=${proc_utime}
    prev_proc_stime=${proc_stime}
```

```
      sleep 1
    done
  ) &
  SAMPLER_PID=$!
}
```

- Registra, a cada segundo, um vetor de métricas (tempo, percentuais de CPU, memória, I/O, deltas de CPU) em formato simples, facilmente processável depois.
- Atualiza as variáveis "anteriores" e dorme por 1 segundo, criando um *perfil temporal* do consumo de recursos.

```
stop_sampler() {
  if [[ "${SAMPLER_RUNNING:-0}" -ne 1 ]]; then
    return
  fi
  SAMPLER_RUNNING=0
```

- Garante que o *stop* só será processado se o amostrador estiver realmente ativo.

```
  if [[ -n "${SAMPLER_PID}" ]] && kill -0 "${SAMPLER_PID}" 2>/dev/null; then
    kill "${SAMPLER_PID}" 2>/dev/null || true
    wait "${SAMPLER_PID}" 2>/dev/null || true
  fi

  if [[ ! -f "${SAMPLER_FILE}" ]]; then
    meta_append "SAMPLER_FILE=absent"
    return
  fi
```

- Tenta encerrar o processo de amostragem de forma limpa (`kill` + `wait`); caso o arquivo de log não exista, registra ausência de dados.

```
  awk 'NR>1 {
    ts=$1; cpu_sys=$2+0; proc_cpu=$3+0; mem=$4+0; r=$5+0; w=$6+0;
    sum_cpu+=cpu_sys; sum_proc+=proc_cpu; sum_mem+=mem;
    if(cpu_sys>max_cpu) max_cpu=cpu_sys;
    if(proc_cpu>max_proc) max_proc=proc_cpu;
    if(mem>max_mem) max_mem=mem;
    samples++;
    last_r=r; last_w=w;
  }
  END {
    if(samples==0) exit;
    printf "SAMPLER_SAMPLES=%d\nCPU_SYS_PCT_AVG=%.2f\nCPU_SYS_PCT_MAX=%.2f\
    nPROC_CPU_PCT_AVG=%.2f\nPROC_CPU_PCT_MAX=%.2f\nMEM_USED_BYTES_AVG=%d\
    nMEM_USED_BYTES_MAX=%d\nPROC_IO_READ_BYTES=%d\nPROC_IO_WRITE_BYTES=%d\n",
      samples, sum_cpu/samples, max_cpu, sum_proc/samples, max_proc, sum_mem/
    samples, max_mem, last_r, last_w
  }' "${SAMPLER_FILE}" > "${SAMPLER_FILE}.summary"
```

- Consolida as amostras em estatísticas agregadas:
  - número de amostras (duração);
  - média e máximo de uso de CPU do sistema e do processo;
  - média e máximo de memória usada;
  - bytes lidos e escritos ao final da execução.
- Gera um arquivo `sampler-<TS>.log.summary`, que resume o comportamento da máquina durante o trecho "pesado".

```
  while IFS= read -r line; do
    meta_append "$line"
  done < "${SAMPLER_FILE}.summary"

  meta_append "SAMPLER_FILE=$(basename "${SAMPLER_FILE}")"
  meta_append "SAMPLER_SUMMARY=$(basename "${SAMPLER_FILE}.summary")"
}
```

- Copia todas as linhas do resumo para o arquivo de meta principal, para que todas as métricas fiquem concentradas num único ponto de consulta por rodada.

```
trap stop_sampler EXIT
```

- Garante que, independentemente de erro ou finalização normal, o amostrador será encerrado e o resumo será produzido, evitando vazamento de processos em *background*.

10.1.4 Bloco 2 — Verificação de dependências básicas

```
# ---------- 2. deps básicos ----------
command -v dd >/dev/null 2>&1 || { echo "[ERRO] instale dd"; exit 4; }
command -v sha256sum >/dev/null 2>&1 || { echo "[ERRO] instale sha256sum"; exit
    5; }
command -v xxd >/dev/null 2>&1 || { echo "[ERRO] instale xxd"; exit 6; }
command -v tar >/dev/null 2>&1 || { echo "[ERRO] instale tar"; exit 7; }
SSH_BIN="$(command -v ssh || true)"
if [[ -z "${SSH_BIN}" ]]; then
  echo "[ERRO] ssh não encontrado no PATH." >&2
  exit 8
fi
```

- `dd`: necessário para gerar blobs de dados aleatórios com o tamanho exato especificado em cada perfil.
- `sha256sum`: usado para calcular a `SHA-256` do TAR, permitindo verificar integridade posteriormente.

- `xxd`: utilizado para converter chave binária em representação hexadecimal compacta, compatível com a interface do OpenSSL.
- `tar`: essencial para compactar os blobs em um único artefato, simulando um *payload* agregador.
- `ssh`: cliente SSH necessário para a sonda de KEX e para a eventual mensagem `wall` no alvo.
- Em caso de ausência de qualquer uma dessas ferramentas, o script falha cedo, com código específico, facilitando correção no ambiente.

10.1.5 Bloco 3 — Metadados iniciais da execução

```
# ---------- 3. meta inicial ----------
{
  echo "RUN_DIR=${RUN_DIR}"
  echo "MODE=${MODE}"
  echo "PROFILE=${PROFILE}"
  echo "TARGET_KIND=${TARGET_KIND}"
  echo "TARGET_LABEL=${TARGET_LABEL}"
  echo "TARGET_IP=${TARGET_IP}"
  echo "TARGET_PORT=${TARGET_PORT}"
  echo "SSH_USER=${SSH_USER}"
  echo "DATE_TS=${TS}"
  echo "START_TS=${START_EPOCH}"
  "${OPENSSL_BIN}" version 2>/dev/null || true
  hostname
  uname -a
} > "${RUN_DIR}/meta-${TS}.txt"
```

- Registra num único arquivo:
  - parâmetros de contexto (modo, perfil, alvo, IP, porta, usuário SSH);
  - versionamento da pilha criptográfica (versão do OpenSSL efetivamente usado);
  - informações de ambiente (hostname e kernel), úteis para reproduzir a execução em máquinas diferentes.
- Esse arquivo também passa a ser enriquecido ao longo do script via `meta_append`, servindo como "registro mestre" da rodada.

10.1.6 Bloco 4 — Métricas de criptografia clássica (X25519)

```
# ---------- 4. métricas clássicas ----------
CLASSIC_PRIV="${RUN_DIR}/classic-priv-${TS}.pem"
CLASSIC_PUB_DER="${RUN_DIR}/classic-pub-${TS}.der"

CLASSIC_KEYGEN_START_MS=$(now_ms)
"${OPENSSL_BIN}" genpkey -algorithm X25519 -out "${CLASSIC_PRIV}" >/dev/null
    2>&1
```

```
"${OPENSSL_BIN}" pkey -in "${CLASSIC_PRIV}" -pubout -outform DER -out "${
    CLASSIC_PUB_DER}" >/dev/null 2>&1
CLASSIC_KEYGEN_END_MS=$(now_ms)
CLASSIC_KEYGEN_TIME_MS=$((CLASSIC_KEYGEN_END_MS - CLASSIC_KEYGEN_START_MS))
```

- Gera um par de chaves de curva elíptica X25519 e exporta a pública em DER, simulando o comportamento de um KEX clássico típico de SSH.
- Mede o tempo de geração de chave (`CLASSIC_KEYGEN_TIME_MS`), permitindo comparar com a geração de chaves PQC (`mlkem768`) em modo híbrido.

```
CLASSIC_SECRET_FILE="${RUN_DIR}/classic-secret-${TS}.bin"
CLASSIC_DERIVE_START_MS=$(now_ms)
# correção: derivar com a pública DER como peer
set +e
"${OPENSSL_BIN}" pkeyutl -derive \
  -inkey "${CLASSIC_PRIV}" \
  -peerkey "${CLASSIC_PUB_DER}" -peerform DER \
  -out "${CLASSIC_SECRET_FILE}" >/dev/null 2>&1
CLASSIC_DERIVE_RC=$?
set -e
CLASSIC_DERIVE_END_MS=$(now_ms)
CLASSIC_DERIVE_TIME_MS=$((CLASSIC_DERIVE_END_MS - CLASSIC_DERIVE_START_MS))
```

- Realiza uma derivação `X25519` local (privada vs pública) para gerar um segredo simulado, permitindo medir o custo da operação *Diffie–Hellman*.
- O `set +e` garante que, mesmo em caso de erro na derivação, o script possa registrar o código de retorno e continuar para fins de estatística.

```
if [[ ${CLASSIC_DERIVE_RC} -ne 0 ]]; then
  # garante que o script não quebre na estatística
  : > "${CLASSIC_SECRET_FILE}"
  meta_append "CLASSIC_DERIVE_RC=${CLASSIC_DERIVE_RC}"
fi

CLASSIC_PUB_SIZE_DER=$(stat -c%s "${CLASSIC_PUB_DER}")
CLASSIC_PUB_SIZE=$((CLASSIC_PUB_SIZE_DER - 12))
CLASSIC_SECRET_SIZE=$(stat -c%s "${CLASSIC_SECRET_FILE}")

meta_append "CLASSIC_KEYGEN_TIME_MS=${CLASSIC_KEYGEN_TIME_MS}"
meta_append "CLASSIC_DERIVE_TIME_MS=${CLASSIC_DERIVE_TIME_MS}"
meta_append "CLASSIC_PUB_SIZE_DER=${CLASSIC_PUB_SIZE_DER}"
meta_append "CLASSIC_PUB_SIZE=${CLASSIC_PUB_SIZE}"
meta_append "CLASSIC_SECRET_SIZE=${CLASSIC_SECRET_SIZE}"
```

- Em caso de falha, zera o arquivo de segredo, evitando leituras inconsistentes, e registra `CLASSIC_DERIVE_RC`.
- Calcula tamanhos de chave pública (bruto e aproximado sem cabeçalho) e do segredo gerado, o que permite comparar volumes de material público e de segredo entre X25519 e KEM PQC.

10.1.7 Bloco 5 — Sonda SSH (negociação de KEX/host key)

```
# ---------- 5. sonda SSH ----------
SSH_DBG="${RUN_DIR}/ssh-probe-${TS}.log"
if [[ "${MODE}" == "classico" ]]; then
  KEX_OFFER="curve25519-sha256"
else
  KEX_OFFER="mlkem768x25519-sha256,curve25519-sha256"
fi
```

- Define o arquivo de log detalhado (`ssh-probe-<TS>.log`) para registrar toda a negociação SSH (`-vvv`).
- Ajusta a lista de KEX ofertados de acordo com o modo:
  - `classico`: apenas `curve25519-sha256`.
  - `hibrido`: oferta `mlkem768x25519-sha256` seguido de `curve25519-sha256`, permitindo que o servidor escolha.

```
SSH_PROBE_START_MS=$(now_ms)
set +e
${SSH_BIN} -p "${TARGET_PORT}" \
  -oBatchMode=yes \
  -oConnectTimeout=5 \
  -oStrictHostKeyChecking=no \
  -oUserKnownHostsFile=/dev/null \
  -oPreferredAuthentications=none \
  -oKexAlgorithms="${KEX_OFFER}" \
  -vvv "${SSH_USER}@${TARGET_IP}" < /dev/null 2> "${SSH_DBG}"
SSH_PROBE_RC=$?
set -e
SSH_PROBE_END_MS=$(now_ms)
SSH_PROBE_DURATION_MS=$((SSH_PROBE_END_MS - SSH_PROBE_START_MS))
```

- Realiza uma conexão SSH de teste com:
  - `-oPreferredAuthentications=none` e `-oBatchMode=yes`: foco na negociação inicial, sem interação de usuário.
  - `-oStrictHostKeyChecking=no` e `-oUserKnownHostsFile=/dev/null`: eliminam erros de *known hosts* que não são relevantes ao experimento.
  - `-vvv`: gera saída altamente detalhada, necessária para extrair KEX e host key.
- Mede o tempo total da tentativa (`SSH_PROBE_DURATION_MS`), permitindo avaliar se o modo híbrido provoca aumento perceptível na duração da fase de *handshake*.

```
meta_append "SSH_PROBE_RC=${SSH_PROBE_RC}"
meta_append "SSH_PROBE_DURATION_MS=${SSH_PROBE_DURATION_MS}"

if [[ ${SSH_PROBE_RC} -ne 0 && ${SSH_PROBE_RC} -ne 255 ]]; then
```

```
  meta_append "SSH_NEGOTIATION_STATUS=failed_connect"
else
  if grep -q "Unable to negotiate with" "${SSH_DBG}"; then
    client_kex=$(grep -m1 -E "^debug2: KEX algorithms:" "${SSH_DBG}" | sed 's/^
    debug2: KEX algorithms: //')
    server_kex=$(grep -A6 -m1 "peer server KEXINIT proposal" "${SSH_DBG}" | grep
     -m1 -E "^debug2: KEX algorithms:" | sed 's/^debug2: KEX algorithms: //')
    meta_append "SSH_NEGOTIATION_STATUS=failed_no_common_kex"
    meta_append "SSH_CLIENT_KEX=${client_kex}"
    meta_append "SSH_SERVER_KEX=${server_kex}"
    meta_append "SSH_KEX_IS_HYBRID=0"
  else
    kex_line=$(grep -m1 -Ei "kex: algorithm:" "${SSH_DBG}" | sed 's/^debug1: //;
    s/^debug2: //')
    hostkey_line=$(grep -m1 -Ei "Server host key:|Host key:" "${SSH_DBG}" | sed
    's/^debug1: //;s/^debug2: //')
    meta_append "SSH_NEGOTIATION_STATUS=ok"
    meta_append "SSH_KEX=${kex_line:-unknown}"
    meta_append "SSH_HOSTKEY=${hostkey_line:-unknown}"
    if echo "${kex_line:-}" | grep -qi "mlkem"; then
      meta_append "SSH_KEX_IS_HYBRID=1"
    else
      meta_append "SSH_KEX_IS_HYBRID=0"
    fi
  fi
fi
meta_append "SSH_PROBE_LOG=$(basename "${SSH_DBG}")"
```

- Classifica o resultado da sonda em três grandes grupos:
  - `failed_connect`: problemas de rede ou serviço indisponível.
  - `failed_no_common_kex`: cliente e servidor não compartilham KEX compatível para o modo testado.
  - `ok`: negociação bem-sucedida, com KEX e host key identificados.
- Registra as listas de KEX ofertados por cliente e servidor quando não há interseção, auxiliando a ajustar o conjunto de algoritmos suportados.
- Em caso de sucesso, marca explicitamente se o KEX contém `mlkem` (`SSH_KEX_IS_HYBRID=1`) ou não (`SSH_KEX_IS_HYBRID=0`), ligação direta com a análise "clássico vs híbrido".

10.1.8 Bloco 6 — Perfil, geração de payload e amostragem

```
# ---------- 6. perfil/payload ----------
case "${PROFILE}" in
  leve)  FILE_COUNT=1000; FILE_SIZE_MB=1 ;;
  medio) FILE_COUNT=100;  FILE_SIZE_MB=10 ;;
  pesado)FILE_COUNT=10;   FILE_SIZE_MB=100 ;;
  *) echo "[ERRO] perfil inválido: ${PROFILE}" >&2; exit 9 ;;
```

```
esac
meta_append "FILE_COUNT=${FILE_COUNT}"
meta_append "FILE_SIZE_MB=${FILE_SIZE_MB}"
```

- Mapeia o conceito lógico de perfil (`leve`, `medio`, `pesado`) para parâmetros quantitativos concretos (número de arquivos e tamanho por arquivo).
- Essa parametrização permite comparar o impacto da mesma "carga lógica" em ambientes com hardware ou pilha criptográfica distintos.

```
# sampler só no trecho pesado
start_sampler

PAYLOAD_DIR="${RUN_DIR}/payloads"
mkdir -p "${PAYLOAD_DIR}"
for i in $(seq 1 "${FILE_COUNT}"); do
  dd if=/dev/urandom of="${PAYLOAD_DIR}/blob-${i}.bin" bs=1M count="${
    FILE_SIZE_MB}" status=none
done

PAYLOAD_TAR="${RUN_DIR}/payload-${TS}.tar.gz"
tar -czf "${PAYLOAD_TAR}" -C "${PAYLOAD_DIR}" .
PAYLOAD_BYTES=$(stat -c%s "${PAYLOAD_TAR}")
PAYLOAD_SHA256=$(sha256sum "${PAYLOAD_TAR}" | awk '{print $1}')
meta_append "PAYLOAD_TAR=$(basename "${PAYLOAD_TAR}")"
meta_append "PAYLOAD_BYTES=${PAYLOAD_BYTES}"
meta_append "PAYLOAD_SHA256=${PAYLOAD_SHA256}"
```

- Inicia o amostrador imediatamente antes da geração de *payload*, capturando o custo real da criação e compactação dos dados.
- Gera arquivos binários pseudoaleatórios, evitando compressão trivial e simulando dados "difíceis" do ponto de vista criptográfico (alta entropia).
- Compacta todos os blobs em um único TAR+GZIP, simulando um volume que poderia ser exfiltrado em um ataque real.
- Calcula e registra tamanho total e `SHA-256` do TAR, permitindo checar integridade ou comparar expansão após cifragem.

10.1.9 Bloco 7 — AES-256-CBC + HMAC

```
# ---------- 7. AES-256-CBC + HMAC ----------
SYM_KEY_FILE="${RUN_DIR}/symkey-${TS}.bin"
IV_FILE="${RUN_DIR}/iv-${TS}.bin"
"${OPENSSL_BIN}" rand -out "${SYM_KEY_FILE}" 32
"${OPENSSL_BIN}" rand -out "${IV_FILE}" 12
KEY_HEX=$(xxd -p "${SYM_KEY_FILE}" | tr -d '\n')
meta_append "SYM_KEY_FILE=$(basename "${SYM_KEY_FILE}")"
meta_append "IV_FILE=$(basename "${IV_FILE}")"
meta_append "SYM_KEY_HEX=${KEY_HEX}"
```

- Gera uma chave simétrica forte de 256 bits, essencial para o uso de `AES-256-CBC`.
- Converte a chave para hexadecimal sem quebras de linha, formato aceito diretamente pelos comandos `enc` e `dgst` com HMAC no OpenSSL.

```
ENC_PAYLOAD="${RUN_DIR}/payload-${TS}.enc"
CBC_IV_FILE="${RUN_DIR}/iv-cbc-${TS}.bin"
"${OPENSSL_BIN}" rand -out "${CBC_IV_FILE}" 16
CBC_IV_HEX=$(xxd -p "${CBC_IV_FILE}" | tr -d '\n')

set +e
"${OPENSSL_BIN}" enc -aes-256-cbc -in "${PAYLOAD_TAR}" -out "${ENC_PAYLOAD}" -K
    "${KEY_HEX}" -iv "${CBC_IV_HEX}" 2> "${RUN_DIR}/enc-cbc-${TS}.err"
ENC_RC=$?
set -e
```

- Gera um IV de 16 bytes, como exigido pelo modo CBC de 128 bits de bloco.
- Criptografa o TAR, gerando `payload-<TS>.enc` e registrando eventuais mensagens de erro em `enc-cbc-<TS>.err` para depuração.

```
if [[ ${ENC_RC} -ne 0 ]]; then
  meta_append "ENC_RC=${ENC_RC}"
  meta_append "ENC_ALG_USED=FAILED"
  echo "[ERRO] falha na encriptação CBC" >&2
  stop_sampler
  exit 10
fi
ENC_PAYLOAD_BYTES=$(stat -c%s "${ENC_PAYLOAD}")
meta_append "IV_CBC_FILE=$(basename "${CBC_IV_FILE}")"
meta_append "ENC_PAYLOAD=$(basename "${ENC_PAYLOAD}")"
meta_append "ENC_PAYLOAD_BYTES=${ENC_PAYLOAD_BYTES}"
meta_append "ENC_ALG_USED=AES-256-CBC"
meta_append "ENC_RC=${ENC_RC}"
```

- Em caso de falha, interrompe cedo e registra o erro, evitando métricas incoerentes com cifragem incompleta.
- Quando bem-sucedido, registra tamanho do ciphertext, IV e algoritmo de cifra usado, permitindo analisar expansão e sobrecarga.

```
ENC_HMAC_FILE="${RUN_DIR}/payload-${TS}.enc.hmac"
"${OPENSSL_BIN}" dgst -sha256 -mac HMAC -macopt hexkey:"${KEY_HEX}" -out "${
    ENC_HMAC_FILE}" "${ENC_PAYLOAD}"
HMAC_RC=$?
if [[ ${HMAC_RC} -ne 0 ]]; then
  meta_append "ENC_HMAC_RC=${HMAC_RC}"
  echo "[ERRO] falha ao gerar HMAC" >&2
  stop_sampler
  exit 11
```

```
fi
meta_append "ENC_HMAC_FILE=$(basename "${ENC_HMAC_FILE}")"
meta_append "ENC_HMAC_RC=${HMAC_RC}"
```

- Calcula HMAC-SHA256 do ciphertext, simulando confidencialidade + integridade/autenticidade (modelo clássico de canal seguro).
- Em caso de falha no HMAC, aborta, pois a integridade deixa de ser garantida.

10.1.10 Bloco 8 — PQC em modo híbrido (KEM via oqsprovider)

```
# ---------- 8. PQC (modo hibrido) ----------
if [[ "${MODE}" == "hibrido" ]]; then
  if "${OPENSSL_BIN}" list -providers 2>/dev/null | grep -qi "oqsprovider"; then
    meta_append "OQSPROVIDER=present"
    KEM_CHOICE=""
    if "${OPENSSL_BIN}" list -kem-algorithms -provider oqsprovider 2>/dev/null |
     grep -q "mlkem768"; then
      KEM_CHOICE="mlkem768"
    fi
```

- Executa apenas em modo `hibrido`, mantendo o modo `classico` como linha de base sem PQC.
- Confirma se o *oqsprovider* está realmente carregado e se o KEM `mlkem768` está disponível, alinhando-se à configuração feita nos capítulos anteriores.

```
    if [[ -n "${KEM_CHOICE}" ]]; then
      meta_append "PQC_KEM_CHOICE=${KEM_CHOICE}"
      KEM_PRIV="${RUN_DIR}/kem-priv-${TS}.pem"
      KEM_PUB_DER="${RUN_DIR}/kem-pub-${TS}.der"

      PQC_KEYGEN_START_MS=$(now_ms)
      if "${OPENSSL_BIN}" genpkey -provider oqsprovider -algorithm "${KEM_CHOICE
   }" -out "${KEM_PRIV}" 2>/dev/null; then
        "${OPENSSL_BIN}" pkey -provider oqsprovider -in "${KEM_PRIV}" -pubout -
   outform DER -out "${KEM_PUB_DER}" 2>/dev/null || true
        PQC_KEYGEN_END_MS=$(now_ms)
        PQC_KEYGEN_TIME_MS=$((PQC_KEYGEN_END_MS - PQC_KEYGEN_START_MS))
        meta_append "PQC_KEYPAIR_GENERATED=1"
        meta_append "PQC_KEYGEN_TIME_MS=${PQC_KEYGEN_TIME_MS}"
```

- Gera par de chaves KEM PQC (`mlkem768`) via OQS, medindo o custo de geração e permitindo comparação com `X25519`.
- Exporta a chave pública em DER para uso nas operações de encapsulação e, se necessário, no modo alternativo de `-encrypt`.

```
      PQC_CT_FILE="${RUN_DIR}/symkey-pqc-${TS}.bin"
      PQC_SECRET_FILE="${RUN_DIR}/symkey-pqc-${TS}.secret"

      PQC_ENCAP_START_MS=$(now_ms)
      if "${OPENSSL_BIN}" pkeyutl -provider oqsprovider \
          -inkey "${KEM_PUB_DER}" -pubin \
          -encap -out "${PQC_CT_FILE}" \
          -secret "${PQC_SECRET_FILE}" 2>/dev/null; then
        PQC_ENCAP_END_MS=$(now_ms)
        PQC_ENCAP_TIME_MS=$((PQC_ENCAP_END_MS - PQC_ENCAP_START_MS))
        meta_append "PQC_KEM_ENCAP=ok_encap"
        meta_append "PQC_ENCAP_TIME_MS=${PQC_ENCAP_TIME_MS}"
        meta_append "PQC_SYMKEY_FILE=$(basename "${PQC_CT_FILE}")"
        meta_append "PQC_SECRET_FILE=$(basename "${PQC_SECRET_FILE}")"
```

- Tenta o fluxo "ideal" de KEM: encapsulação produzindo:
  - *ciphertext* KEM (`PQC_CT_FILE`);
  - nova chave secreta derivada (`PQC_SECRET_FILE`).
- Mede o tempo de encapsulação (`PQC_ENCAP_TIME_MS`), comparável com o custo de `X25519` e com o *handshake* SSH.

```
      else
        PQC_ENCAP_START_MS=$(now_ms)
        if "${OPENSSL_BIN}" pkeyutl -provider oqsprovider \
            -inkey "${KEM_PUB_DER}" -pubin \
            -encrypt -in "${SYM_KEY_FILE}" \
            -out "${PQC_CT_FILE}" 2>/dev/null; then
          PQC_ENCAP_END_MS=$(now_ms)
          PQC_ENCAP_TIME_MS=$((PQC_ENCAP_END_MS - PQC_ENCAP_START_MS))
          meta_append "PQC_KEM_ENCAP=ok_encrypt_mode"
          meta_append "PQC_ENCAP_TIME_MS=${PQC_ENCAP_TIME_MS}"
          meta_append "PQC_SYMKEY_FILE=$(basename "${PQC_CT_FILE}")"
        else
          meta_append "PQC_KEM_ENCAP=failed_both"
        fi
      fi
```

- Se o modo KEM "puro" falhar, recorre a um modo compatível com versões mais antigas: cifra diretamente a chave simétrica `SYM_KEY_FILE` com a chave pública PQC.
- Registra claramente se foi possível usar modo KEM padrão ou se foi necessário o modo `-encrypt`, o que é importante ao comparar comportamentos entre versões do OpenSSL/OQS.

```
        if [[ -f "${KEM_PUB_DER}" ]]; then
          PQC_PUB_SIZE_DER=$(stat -c%s "${KEM_PUB_DER}")
          PQC_PUB_SIZE=$((PQC_PUB_SIZE_DER - 22))
          meta_append "PQC_PUB_SIZE_DER=${PQC_PUB_SIZE_DER}"
          meta_append "PQC_PUB_SIZE=${PQC_PUB_SIZE}"
        fi
        if [[ -f "${PQC_CT_FILE}" ]]; then
          PQC_CT_SIZE=$(stat -c%s "${PQC_CT_FILE}")
          meta_append "PQC_CT_SIZE=${PQC_CT_SIZE}"
        fi

      else
        meta_append "PQC_KEYPAIR_GENERATED=failed"
      fi
    else
      meta_append "PQC_KEM_STATUS=no_supported_kem_found"
    fi
  else
    meta_append "OQSPROVIDER=absent"
  fi
else
  meta_append "PQC_SKIPPED=mode_classico"
fi
```

- Registra tamanhos de chave pública PQC e do *ciphertext* KEM, essenciais para comparar impacto em largura de banda e tamanho de *handshake*.
- Em caso de ausência de KEM ou de *oqsprovider*, marca claramente o motivo (`no_supported_kem_found`, `OQSPROVIDER=absent`).
- Em modo `classico`, registra explicitamente `PQC_SKIPPED=mode_classico`, evitando ambiguidades na interpretação dos metadados.

```
# acabou trecho pesado
stop_sampler
```

- Encerra o amostrador imediatamente após o bloco de operações intensivas (payload + cifra + PQC), isolando o consumo de recursos desse trecho do restante do script.

10.1.11 Bloco 9 — Envelope JSON e resumo humano

```
# ---------- 9. envelope e summary ----------
ENVELOPE="${RUN_DIR}/envelope-${TS}.json"
cat > "${ENVELOPE}" <<EOF
{
  "mode": "${MODE}",
  "profile": "${PROFILE}",
  "target_label": "${TARGET_LABEL}",
  "target_kind": "${TARGET_KIND}",
```

```
  "target_ip": "${TARGET_IP}",
  "target_port": "${TARGET_PORT}",
  "payload_enc": "$(basename "${ENC_PAYLOAD}")",
  "payload_sha256": "${PAYLOAD_SHA256}",
  "payload_bytes": ${PAYLOAD_BYTES},
  "enc_alg_used": "AES-256-CBC",
  "enc_hmac_file": "$(basename "${ENC_HMAC_FILE}")",
  "date_ts": "${TS}"
}
EOF
meta_append "ENVELOPE=$(basename "${ENVELOPE}")"
```

- Gera um "envelope" em JSON que resume os dados essenciais da rodada: modo, perfil, alvo, tamanho e integridade do payload cifrado.
- Esse arquivo é apropriado para uso por ferramentas automáticas (por exemplo, scripts de exfiltração ou pós-processamento).

```
SUMMARY="${RUN_DIR}/summary.txt"
{
  echo "Criação de payload"
  echo "Data/Hora: ${TS}"
  echo "Modo: ${MODE}"
  echo "Perfil: ${PROFILE}"
  echo "Alvo: ${TARGET_LABEL} (${TARGET_IP}:${TARGET_PORT}) tipo=${TARGET_KIND}"
  echo
  echo "Arquivos gerados: ${FILE_COUNT} x ${FILE_SIZE_MB} MiB"
  echo "Payload tar: $(basename "${PAYLOAD_TAR}") (${PAYLOAD_BYTES} bytes)"
  echo "Payload cifrado: $(basename "${ENC_PAYLOAD}") (${ENC_PAYLOAD_BYTES}
    bytes)"
  echo "AES usado: AES-256-CBC"
  echo "SHA256 do tar: ${PAYLOAD_SHA256}"
  echo "HMAC do ciphertext: $(basename "${ENC_HMAC_FILE}")"
  echo
  echo "Sonda SSH: ver meta para SSH_*"
  echo "Métricas: ver meta para CLASSIC_*, PQC_*, SSH_*, ENC_*, SAMPLER_*"
} > "${SUMMARY}"
```

- Cria um resumo textual amigável, voltado à leitura humana, que pode ser utilizado diretamente na análise ou citado no texto.
- Orienta explicitamente onde encontrar os detalhes (metadados `CLASSIC_*`, `PQC_*`, `SSH_*`, `ENC_*`, `SAMPLER_*`), facilitando navegação nos arquivos.

10.1.12 Bloco 10 — Notificação remota e encerramento

```
# ---------- 10. mensagem no sshreal ----------
MSG="[SSH_PAYLOAD_PROBE][${MODE}/${PROFILE}] payload ${PAYLOAD_BYTES} bytes,
    sha256=${PAYLOAD_SHA256}, hora=${TS}"
```

```
if command -v sshpass >/dev/null 2>&1; then
  set +e
  sshpass -p "${SSH_PASS}" \
    "${SSH_BIN}" -p "${TARGET_PORT}" -oConnectTimeout=5 \
    -oStrictHostKeyChecking=no -oUserKnownHostsFile=/dev/null \
    -oKexAlgorithms="${KEX_OFFER}" \
    "${SSH_USER}@${TARGET_IP}" "echo '${MSG}' | wall" >/dev/null 2>&1 || true
  set -e
  meta_append "REMOTE_WALL_SENT=1"
else
  meta_append "REMOTE_WALL_SENT=0_no_sshpass"
fi
```

- , Se `sshpass` estiver presente, o script envia uma notificação para todos os terminais logados no `sshreal` via `wall`, contendo resumo da rodada (modo, perfil, tamanho e hash).
- Essa mensagem ajuda a validar que o servidor está ativo e a identificar, em tempo real, que um experimento específico foi executado.
- Se `sshpass` não estiver instalado, o script não falha: apenas registra que a notificação não foi enviada.

```
END_EPOCH=$(date +%s)
RUN_DURATION_S=$((END_EPOCH - START_EPOCH))
meta_append "END_TS=${END_EPOCH}"
meta_append "RUN_DURATION_S=${RUN_DURATION_S}"

echo "[OK] SSH_Payload_Probe finalizado. Artefatos em: ${RUN_DIR}"
```

- Calcula a duração total da execução, de `START_EPOCH` a `END_EPOCH`, em segundos, oferecendo uma métrica global de "custo" da rodada.
- A mensagem final mostra o diretório exato onde todos os arquivos (payloads, ciphertexts, HMAC, metas, sampler, logs SSH, envelope, summary) foram armazenados, facilitando a coleta e o uso posterior (inclusive para ingestão no Splunk).

10.2 Artefatos gerados e estrutura de saída

10.2.1 Organização de diretórios

A cada execução, o script `ssh_payload_probe.sh` cria um diretório dedicado à rodada, organizado por modo (`MODE`), tipo de cenário (`payload_probe`), tipo de alvo (`sshreal`) e perfil (`PROFILE`). A raiz lógica dos ataques é:

```
$HOME/ataques1/<mode>/payload_probe/sshreal/<profile>/
```

Dentro dessa raiz, para cada execução é criado um diretório `run` com carimbo de data e hora:

```
$HOME/ataques1/<mode>/payload_probe/sshreal/<profile>/run-<profile>-<TS>/
```

em que:

- `<mode>` ∈ {`classico`, `hibrido`};
- `<profile>` ∈ {`leve`, `medio`, `pesado`};
- `<TS>` é um carimbo temporal no formato `YYYYMMDD-HHMMSS`.

Todas as saídas relevantes (dados, chaves, métricas, resumos e *logs*) de uma rodada ficam contidas nesse `RUN_DIR`, o que permite correlacionar todos os artefatos com um único experimento.

10.2.2 Metadados e resumos principais

No diretório `run-<profile>-<TS>`, os arquivos de controle e resumo geral são:

- `meta-<TS>.txt`
  - arquivo de metadados principal, criado no início da execução e atualizado ao longo do script por meio da função `meta_append`;
  - concentra os parâmetros de execução (modo, perfil, alvo, IP, porta, usuário);
  - registra a versão do OpenSSL utilizado, o `hostname` e o resultado de `uname -a`;
  - agrega as métricas da parte clássica (`CLASSIC_*`);
  - agrega as métricas PQC/híbridas (`PQC_*`);
  - registra os resultados da sonda SSH (`SSH_*`);
  - registra as informações de cifragem simétrica e HMAC (`ENC_*`);
  - inclui as estatísticas de amostragem de recursos (`SAMPLER_*`);
  - armazena a duração total da execução (`RUN_DURATION_S`).
- `summary.txt`
  - resumo textual em linguagem natural da rodada;
  - contém data/hora, modo, perfil e identificação do alvo;
  - apresenta a quantidade e o tamanho dos arquivos de *payload*;
  - indica nomes e tamanhos do TAR e do *ciphertext*;
  - explicita o algoritmo AES utilizado, a `SHA-256` do TAR e o arquivo de HMAC gerado;
  - faz referência às famílias de métricas presentes em `meta-<TS>.txt`, orientando a análise detalhada.
- `envelope-<TS>.json`
  - estrutura em formato JSON que funciona como "descritor" sintético da rodada;

- inclui `mode`, `profile`, `target_label`, `target_ip` e `target_port`;
- registra o nome do arquivo cifrado (`payload_enc`);
- informa o tamanho do *payload* em bytes (`payload_bytes`);
- registra o algoritmo simétrico utilizado (`enc_alg_used`);
- associa o arquivo de HMAC correspondente (`enc_hmac_file`);
- guarda o carimbo de data/hora da execução (`date_ts`).

Esses três arquivos constituem os pontos de entrada recomendados para uma inspeção rápida da rodada e para integração com outras ferramentas (por exemplo, scripts de exfiltração ou rotinas de correlação de *logs*).

10.2.3 Payloads gerados e artefatos de cifragem

A parte de geração de *payload* e cifragem produz os seguintes artefatos:

- Diretório `payloads/`
  - contém os arquivos binários pseudoaleatórios gerados com `dd` a partir de `/dev/urandom`;
  - segue o padrão de nomenclatura:
    - `blob-1.bin`, `blob-2.bin`, ..., `blob-$FILE_COUNT.bin`;
  - os valores `FILE_COUNT` e `FILE_SIZE_MB` dependem do perfil selecionado:
    - perfil `leve`: 1000 arquivos de 1 MiB;
    - perfil `medio`: 100 arquivos de 10 MiB;
    - perfil `pesado`: 10 arquivos de 100 MiB.
- `payload-<TS>.tar.gz`
  - arquivo TAR compactado com GZIP contendo todos os `blob-.bin` do diretório `payloads/`;
  - o tamanho em bytes (`PAYLOAD_BYTES`) e a `SHA-256` (`PAYLOAD_SHA256`) desse arquivo são registrados em `meta-<TS>.txt` e em `summary.txt`;
  - serve como *payload* "bruto" de referência, antes da cifragem simétrica.
- `payload-<TS>.enc`
  - arquivo resultante da cifragem do TAR com `AES-256-CBC`, utilizando chave simétrica gerada aleatoriamente;
  - o tamanho em bytes é armazenado na meta em `ENC_PAYLOAD_BYTES`;
  - representa o *ciphertext* efetivamente sujeito a análise (por exemplo, para exfiltração ou comparação de volume cifrado).
- `payload-<TS>.enc.hmac`
  - arquivo contendo o HMAC-SHA256 do *ciphertext* `payload-<TS>.enc`;

    - é calculado com a mesma chave simétrica utilizada na cifragem, em formato hexadecimal (`SYM_KEY_HEX`);
    - permite verificar integridade/autenticidade do *ciphertext* de forma independente.

- `symkey-<TS>.bin` e `iv-cbc-<TS>.bin`
    - armazenam, respectivamente:
        - a chave simétrica de 256 bits em formato binário;
        - o IV de 16 bytes utilizado no modo `AES-256-CBC`.
    - os nomes desses arquivos, bem como a forma hexadecimal da chave, são registrados na meta por meio dos campos `SYM_KEY_FILE`, `IV_CBC_FILE` e `SYM_KEY_HEX`;
    - permitem reproduzir ou validar localmente a cifragem, se necessário.

10.2.4 Artefatos da parte clássica e pós-quântica

As operações de chave clássica (X25519) e PQC/híbrida (quando ativada) geram os seguintes conjuntos de arquivos:

- **Clássico (X25519)**
    - `classic-priv-<TS>.pem`
        - chave privada X25519 gerada via `genpkey`;
        - utilizada como insumo para derivação de segredo com a respectiva chave pública.
    - `classic-pub-<TS>.der`
        - chave pública correspondente, exportada em formato DER;
        - utilizada como *peer key* na derivação (`pkeyutl -derive`).
    - `classic-secret-<TS>.bin`
        - segredo resultante da derivação de chave (`pkeyutl -derive`);
        - em caso de falha, o arquivo é criado vazio para não quebrar a coleta de métricas.
    - Metadados associados
        - tempos de geração e derivação: `CLASSIC_KEYGEN_TIME_MS`, `CLASSIC_DERIVE_TIME_MS`;
        - tamanhos de chave pública e de segredo: `CLASSIC_PUB_SIZE_*`, `CLASSIC_SECRET_SIZE`;
        - código de retorno da derivação, quando aplicável (`CLASSIC_DERIVE_RC`).
- **PQC/híbrido (modo `hibrido`)**
    - `kem-priv-<TS>.pem`
        - chave privada do KEM pós-quântico (por exemplo, `mlkem768`);
        - gerada via `genpkey` com o *oqsprovider*.

- `kem-pub-<TS>.der`
  - chave pública KEM em formato DER, derivada da chave privada;
  - utilizada na operação de encapsulação (`-encap`) ou, em modo compatível, na cifragem direta (`-encrypt`).
- `symkey-pqc-<TS>.bin`
  - *ciphertext* KEM (quando `-encap` é suportado); ou
  - resultado da cifragem da chave simétrica clássica com a chave pública KEM (modo de compatibilidade);
  - o tamanho desse artefato é registrado em `PQC_CT_SIZE`.
- `symkey-pqc-<TS>.secret` (quando `-encap` é usado)
  - segredo simétrico gerado pela operação de encapsulação KEM;
  - permite comparar a derivação PQC com o segredo gerado em X25519.
- Metadados associados
  - tempo de geração de par KEM (`PQC_KEYGEN_TIME_MS`);
  - tempo de encapsulação (`PQC_ENCAP_TIME_MS`);
  - tamanhos de chave pública e *ciphertext* (`PQC_PUB_SIZE_*`, `PQC_CT_SIZE`);
  - estado da operação KEM (`PQC_KEM_ENCAP`);
  - presença e escolha do provedor (`OQSPROVIDER`, `PQC_KEM_CHOICE`);
  - registro de falhas ou ausência de KEM compatível (`no_supported_kem_found`, `PQC_KEYPAIR_GENERATED=failed`, entre outros).

10.2.5 Logs de sonda SSH e erros de cifragem

Para análise do comportamento do *handshake* SSH e de eventuais problemas de cifragem, os seguintes arquivos são relevantes:

- `ssh-probe-<TS>.log`
  - contém a saída detalhada (`-vvv`) da sonda SSH;
  - registra as listas de algoritmos ofertados pelo cliente e apresentados pelo servidor;
  - evidencia o algoritmo de KEX efetivamente escolhido;
  - traz informações sobre a *host key* utilizada;
  - inclui mensagens de erro em caso de falha de negociação;
  - o nome do arquivo é registrado em `SSH_PROBE_LOG`;
  - o resultado da negociação aparece em campos como `SSH_NEGOTIATION_STATUS`, `SSH_KEX`, `SSH_HOSTKEY` e `SSH_KEX_IS_HYBRID`.
- `enc-cbc-<TS>.err`
  - arquivo de erro da fase de cifragem `AES-256-CBC`;
  - em execuções bem-sucedidas tende a permanecer vazio;
  - em caso de falha, contém mensagens emitidas pelo OpenSSL, auxiliando na depuração de problemas de cifragem.

10.2.6 Amostrador de recursos locais

Por fim, o amostrador de recursos (CPU, memória e I/O) produz:

- `sampler-<TS>.log`
  - arquivo de amostras brutas, em que cada linha (após o cabeçalho) contém:
    - timestamp em milissegundos;
    - porcentagem de uso de CPU do sistema;
    - porcentagem de CPU do processo monitorado;
    - memória utilizada em bytes;
    - contadores de bytes lidos e escritos;
    - deltas de tempo de CPU (global e do processo).
  - é adequado para análises temporais mais finas ou para construção de gráficos de consumo de recursos.
- `sampler-<TS>.log.summary`
  - arquivo de resumo gerado por `awk`, a partir de `sampler-<TS>.log`;
  - suas linhas são importadas para `meta-<TS>.txt`, o que centraliza as estatísticas num único ponto;
  - inclui, entre outros:
    - número de amostras (`SAMPLER_SAMPLES`);
    - médias e máximos de uso de CPU do sistema e do processo;
    - médias e máximos de memória utilizada;
    - contadores finais de I/O em bytes (`PROC_IO_READ_BYTES`, `PROC_IO_WRITE_BYTES`).

Em conjunto, essa estrutura de saída permite que cada execução do `ssh_payload_probe.sh` seja completamente reconstituída e comparada com outras rodadas (diferentes modos, perfis ou parâmetros), tanto do ponto de vista criptográfico quanto do ponto de vista de consumo de recursos e de negociação de algoritmos em SSH.

10.3 Análise de um arquivo de metadados `meta-*.txt`

Este exemplo apresenta um arquivo `meta-*.txt` real gerado pelo `ssh_payload_probe.sh` em modo `hibrido` e perfil `medio`, durante uma execução contra o servidor `sshreal`. O objetivo é ilustrar como os campos são organizados e como podem ser interpretados posteriormente na análise dos experimentos.

10.3.1 Registro bruto do arquivo de metadados

```
RUN_DIR=/home/attacker/ataques1/hibrido/payload_probe/sshreal/medio/run-medio
    -20251112-004419
MODE=hibrido
PROFILE=medio
```

```
TARGET_KIND=sshreal
TARGET_LABEL=sshserver
TARGET_IP=192.168.50.40
TARGET_PORT=22222
SSH_USER=sshserver
DATE_TS=20251112-004419
START_TS=1762922659
OpenSSL 3.5.3 16 Sep 2025 (Library: OpenSSL 3.5.3 16 Sep 2025)
facom
Linux facom 6.16.8+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.16.8-1kali1
    (2025-09-24) x86_64 GNU/Linux
CLASSIC_KEYGEN_TIME_MS=24
CLASSIC_DERIVE_TIME_MS=10
CLASSIC_PUB_SIZE_DER=44
CLASSIC_PUB_SIZE=32
CLASSIC_SECRET_SIZE=32
SSH_PROBE_RC=255
SSH_PROBE_DURATION_MS=102
SSH_NEGOTIATION_STATUS=ok
SSH_KEX=kex: algorithm: mlkem768x25519-sha256
SSH_HOSTKEY=Server host key: ssh-ed25519 SHA256:GjAPcMzh04k4o+81RzEi+
    hcaIYCIoARnLoumk3V53Rw
SSH_KEX_IS_HYBRID=1
SSH_PROBE_LOG=ssh-probe-20251112-004419.log
FILE_COUNT=100
FILE_SIZE_MB=10
PAYLOAD_TAR=payload-20251112-004419.tar.gz
PAYLOAD_BYTES=1048759350
PAYLOAD_SHA256=88e5ac710d50c97f6983e9e9b7dbe223c183978547778348df48435f2f05f5a9
SYM_KEY_FILE=symkey-20251112-004419.bin
IV_FILE=iv-20251112-004419.bin
SYM_KEY_HEX=2f5f7b5f1a018d7fb86c3cf221fb6bcb74ca1790cad930010caf5938d67c7f9b
IV_CBC_FILE=iv-cbc-20251112-004419.bin
ENC_PAYLOAD=payload-20251112-004419.enc
ENC_PAYLOAD_BYTES=1048759360
ENC_ALG_USED=AES-256-CBC
ENC_RC=0
ENC_HMAC_FILE=payload-20251112-004419.enc.hmac
ENC_HMAC_RC=0
OQSPROVIDER=present
PQC_KEM_CHOICE=mlkem768
PQC_KEYPAIR_GENERATED=1
PQC_KEYGEN_TIME_MS=26
PQC_KEM_ENCAP=ok_encap
PQC_ENCAP_TIME_MS=9
PQC_SYMKEY_FILE=symkey-pqc-20251112-004419.bin
PQC_SECRET_FILE=symkey-pqc-20251112-004419.secret
PQC_PUB_SIZE_DER=1206
PQC_PUB_SIZE=1184
PQC_CT_SIZE=1088
SAMPLER_SAMPLES=57
```

```
CPU_SYS_PCT_AVG=38.96
CPU_SYS_PCT_MAX=62.19
PROC_CPU_PCT_AVG=0.01
PROC_CPU_PCT_MAX=0.25
MEM_USED_BYTES_AVG=1185134627
MEM_USED_BYTES_MAX=1221373952
PROC_IO_READ_BYTES=144703488
PROC_IO_WRITE_BYTES=3146145792
SAMPLER_FILE=sampler-20251112-004419.log
SAMPLER_SUMMARY=sampler-20251112-004419.log.summary
ENVELOPE=envelope-20251112-004419.json
REMOTE_WALL_SENT=1
END_TS=1762922718
RUN_DURATION_S=59
```

10.3.2 Bloco 1 – Identificação da execução e contexto básico

- `RUN_DIR`: caminho completo do diretório raiz da rodada. Consolida todos os artefatos desta execução (payloads, chaves, logs, resumos e arquivos de amostragem), permitindo localizar exatamente o experimento:
  - `/home/attacker/ataques1/hibrido/payload_probe/sshreal/medio/run-medio-20251112-004419`
- `MODE=hibrido` e `PROFILE=medio`: indicam que se trata de uma rodada em modo híbrido (KEX pós-quântico + clássico) com carga de perfil `medio` (100 arquivos de 10 MiB, conforme definido no script).
- `TARGET_KIND=sshreal`, `TARGET_LABEL=sshserver`, `TARGET_IP=192.168.50.40`, `TARGET_PORT=22222`, `SSH_USER=sshserver`: identificam o alvo lógico e físico: servidor SSH real (`sshreal`) executando em `192.168.50.40:22222`, com usuário `sshserver`.
- `DATE_TS=20251112-004419` e `START_TS=1762922659`: carimbo legível (data e hora local) e marca em segundos desde a época Unix no início da execução, usados para indexação cronológica e cálculo de duração total.

10.3.3 Bloco 2 – Versão de software e ambiente de execução

- Linha `OpenSSL 3.5.3 16 Sep 2025 (...)`: registra a versão exata do OpenSSL utilizada, incluindo a versão da biblioteca. É importante para reprodutibilidade, principalmente em cenários envolvendo algoritmos pós-quânticos.
- Linha `facom`: *hostname* da máquina atacante onde o script foi executado.
- Linha `Linux facom 6.16.8+kali-amd64 ...`: saída do `uname -a`, descrevendo kernel, arquitetura e versão da distribuição (Kali). Esses dados permitem correlacionar diferenças de desempenho com o ambiente de sistema operacional.

10.3.4 Bloco 3 – Métricas da parte clássica (X25519)

- `CLASSIC_KEYGEN_TIME_MS=24`: tempo, em milissegundos, para gerar o par de chaves X25519 (linha de base clássica).
- `CLASSIC_DERIVE_TIME_MS=10`: tempo de derivação de segredo (`pkeyutl -derive`) usando a chave privada e a chave pública correspondente.
- `CLASSIC_PUB_SIZE_DER=44` e `CLASSIC_PUB_SIZE=32`: tamanho da chave pública em DER (44 bytes) e tamanho aproximado do conteúdo útil sem cabeçalhos (32 bytes), coerente com X25519.
- `CLASSIC_SECRET_SIZE=32`: tamanho do segredo derivado em bytes, resultante da operação de Diffie–Hellman clássica.

10.3.5 Bloco 4 – Resultado da sonda SSH (handshake e KEX)

- `SSH_PROBE_RC=255` e `SSH_PROBE_DURATION_MS=102`: código de retorno da sonda SSH (255 é típico de encerramento após tentativa sem autenticação) e duração total da tentativa (102 ms).
- `SSH_NEGOTIATION_STATUS=ok`: indica que a fase de negociação de algoritmos foi bem-sucedida, mesmo que a conexão não tenha prosseguido para autenticação.
- `SSH_KEX=kex: algorithm: mlkem768x25519-sha256`: registra o algoritmo de troca de chaves efetivamente escolhido no handshake. Neste exemplo, trata-se de um KEX híbrido PQC+clássico.
- `SSH_HOSTKEY=Server host key: ssh-ed25519 ...`: descreve o tipo de *host key* utilizado (`ssh-ed25519`) e seu identificador `SHA256`, permitindo verificar consistência do servidor ao longo de múltiplas rodadas.
- `SSH_KEX_IS_HYBRID=1`: marca explicitamente que a negociação utilizou KEX híbrido (contém componente `mlkem`).
- `SSH_PROBE_LOG=ssh-probe-20251112-004419.log`: nome do arquivo que contém o log detalhado (`-vvv`) da sonda SSH, armazenado dentro do `RUN_DIR`.

10.3.6 Bloco 5 – Perfil de payload e cifragem simétrica

- `FILE_COUNT=100` e `FILE_SIZE_MB=10`: definem o perfil de carga `medio`: 100 arquivos de 10 MiB cada, gerados a partir de `/dev/urandom`.
- `PAYLOAD_TAR=payload-20251112-004419.tar.gz`: nome do arquivo TAR+GZIP que agrega todos os blobs de *payload* desta rodada.
- `PAYLOAD_BYTES=1048759350`: tamanho total do TAR, em bytes (aproximadamente 1 GiB).
- `PAYLOAD_SHA256=88e5ac71...f5a9`: `SHA-256` do TAR, usada como referência de integridade antes da cifragem.

- `SYM_KEY_FILE` e `IV_FILE`: arquivos contendo, respectivamente, a chave simétrica em formato binário e um IV auxiliar gerado pelo script.
- `SYM_KEY_HEX=2f5f7b5f...`: representação hexadecimal da mesma chave simétrica, compatível com os parâmetros `-K` e `-macopt hexkey:` do OpenSSL.
- `IV_CBC_FILE=iv-cbc-20251112-004419.bin`: IV de 16 bytes utilizado especificamente na cifragem `AES-256-CBC`.
- `ENC_PAYLOAD=payload-20251112-004419.enc` e `ENC_PAYLOAD_BYTES=1048759360`: nome e tamanho do arquivo cifrado; o tamanho tende a ser ligeiramente maior que o TAR original devido ao preenchimento do modo de bloco.
- `ENC_ALG_USED=AES-256-CBC` e `ENC_RC=0`: algoritmo simétrico efetivamente utilizado e código de retorno da cifragem (0 indica sucesso).
- `ENC_HMAC_FILE=payload-20251112-004419.enc.hmac` e `ENC_HMAC_RC=0`: arquivo contendo o HMAC-SHA256 do *ciphertext* e código de retorno da operação de HMAC (0 indica sucesso na geração da marca de integridade).

10.3.7 Bloco 6 – Indicadores da parte pós-quântica (PQC)

- `OQSPROVIDER=present`: confirma que o *provider* `oqsprovider` foi carregado corretamente pelo OpenSSL.
- `PQC_KEM_CHOICE=mlkem768`: algoritmo KEM PQC selecionado para a rodada híbrida.
- `PQC_KEYPAIR_GENERATED=1` e `PQC_KEYGEN_TIME_MS=26`: indicam que o par de chaves KEM foi gerado com sucesso e registram o tempo de geração em milissegundos.
- `PQC_KEM_ENCAP=ok_encap` e `PQC_ENCAP_TIME_MS=9`: sinalizam que a operação de encapsulação KEM (`-encap`) foi bem-sucedida e registram o seu tempo.
- `PQC_SYMKEY_FILE` e `PQC_SECRET_FILE`: nomes dos arquivos que armazenam, respectivamente, o *ciphertext* KEM e o segredo simétrico resultante da encapsulação.
- `PQC_PUB_SIZE_DER=1206` e `PQC_PUB_SIZE=1184`: tamanhos da chave pública KEM em DER e estimativa do conteúdo útil, evidenciando o aumento significativo em relação à chave pública clássica (32 bytes).
- `PQC_CT_SIZE=1088`: tamanho, em bytes, do *ciphertext* KEM gerado na operação de encapsulação. Esse valor é relevante para a análise de impacto em largura de banda e tamanho de *handshake*.

10.3.8 Bloco 7 – Amostrador de recursos (CPU, memória e I/O)

- `SAMPLER_SAMPLES=57`: quantidade de amostras coletadas pelo amostrador interno durante o trecho considerado "pesado" (geração de payload, cifragem e PQC).
- `CPU_SYS_PCT_AVG=38.96` e `CPU_SYS_PCT_MAX=62.19`: uso médio e máximo de CPU do sistema, em porcentagem, durante o período amostrado.

- `PROC_CPU_PCT_AVG=0.01` e `PROC_CPU_PCT_MAX=0.25`: uso médio e máximo de CPU atribuído especificamente ao processo do script, indicando que a maior parte do custo de CPU pode estar distribuída em outras atividades do sistema ou em processos auxiliares.
- `MEM_USED_BYTES_AVG=1185134627` e `MEM_USED_BYTES_MAX=1221373952`: consumo médio e máximo de memória, em bytes, observados durante a execução do trecho pesado.
- `PROC_IO_READ_BYTES=144703488` e `PROC_IO_WRITE_BYTES=3146145792`: contadores de bytes lidos e escritos pelo processo, medidos via `/proc/<pid>/io`, refletindo o volume de I/O associado à criação, compactação e cifragem dos arquivos.
- `SAMPLER_FILE` e `SAMPLER_SUMMARY`: nomes do arquivo de amostras brutas (`sampler-*.log`) e do resumo calculado por `awk` (`sampler-*.log.summary`), ambos armazenados no `RUN_DIR` e referenciados pelo `meta-*.txt`.

10.3.9 Bloco 8 – Envelope, notificação remota e encerramento

- `ENVELOPE=envelope-20251112-004419.json`: nome do arquivo JSON que resume os principais parâmetros e artefatos da rodada, pensado para consumo por ferramentas automatizadas.
- `REMOTE_WALL_SENT=1`: indica que a mensagem de notificação (`wall`) foi enviada com sucesso ao servidor `sshreal`, sinalizando a execução do cenário.
- `END_TS=1762922718` e `RUN_DURATION_S=59`: marca temporal de término (em segundos desde a época Unix) e duração total da execução (59 segundos), calculada como diferença entre `END_TS` e `START_TS`.

Este exemplo mostra como o arquivo `meta-*.txt` concentra, em formato chave-valor, todos os elementos necessários para reconstruir o contexto criptográfico, de carga, de recursos e de negociação SSH de uma rodada específica no cenário `payload_probe`.

Capítulo 11

Cenário de Exfiltração

11.1 Objetivo do cenário

O script `ssh_exfil_compare.sh` implementa um cenário controlado de *exfiltração de dados* sobre SSH, com o propósito de comparar, em condições equivalentes, dois modos de negociação criptográfica:

- **modo** `classico`: apenas algoritmos clássicos de troca de chaves (por exemplo, `curve25519-sha256`, `ecdh-sha2-nistp256`);
- **modo** `hibrido`: algoritmos híbridos que combinam componentes pós-quânticos (por exemplo, `mlkem768`) com algoritmos clássicos (por exemplo, `mlkem768x25519-sha256`).

Em cada execução, o script:

- negocia uma sessão SSH com o servidor `sshreal` sob controle estrito de KEX, cifras e *host keys*;
- gera, no servidor, arquivos binários de tamanho conhecido (*payloads* de teste) e registra `checksums` remotos;
- exfiltra todos os arquivos para a máquina atacante, medindo o tempo, o uso de CPU e de memória durante a transferência;
- calcula `checksums` locais e compara com os remotos, verificando integridade fim a fim;
- registra, em um arquivo de metadados, todos os parâmetros relevantes (modo, perfil, autenticação) e as principais métricas da rodada.

11.2 Execução do script de exfiltração

11.2.1 Visão geral e assinatura do script

O script possui o cabeçalho:

```
#!/usr/bin/env bash
# ssh_exfil_compare.sh - Exfiltração Comparável (classico|hibrido) + checksums
    remotolocal
# Assinatura rígida:
#   ./ssh_exfil_compare.sh <classico|hibrido> <leve|medio|pesado> <target_ip> <
    target_port> <ssh_user> <target_kind> <ssh_password_or_key>
set -euo pipefail
```

Os pontos principais desse cabeçalho são:

- `#!/usr/bin/env bash`: garante que o script seja executado com o `bash`, usando o `env` do sistema para localizá-lo.
- comentário descritivo: resume a função do script: exfiltração comparável entre os modos `classico` e `hibrido`, com verificação de `checksums` remoto versus local.
- **assinatura rígida**: a linha de comentário documenta a forma exata de invocação esperada, reforçada pelo código que valida o número de argumentos.
- `set -euo pipefail`:
  - `-e`: encerra o script se qualquer comando simples retornar código de erro;
  - `-u`: trata variáveis não definidas como erro;
  - `pipefail`: propaga código de erro em `pipelines`, caso qualquer comando do pipeline falhe.

  Esses parâmetros tornam a execução mais robusta e evitam que erros silenciosos contaminem as métricas.

11.2.2 Bloco 0 – validação de argumentos e perfis

O primeiro bloco verifica a quantidade de argumentos, inicializa as variáveis de entrada e define o perfil de carga:

```
if [[ $# -ne 7 ]]; then
  echo "uso: $0 <classico|hibrido> <leve|medio|pesado> <target_ip> <target_port>
     <ssh_user> <target_kind> <ssh_password_or_key>" >&2
  exit 2
fi

MODE="$1"
PROFILE="$2"
TARGET_IP="$3"
TARGET_PORT="$4"
TARGET_USER="$5"
TARGET_KIND="$6"
ARG7="$7"
```

- exige exatamente 7 argumentos posicionais; caso contrário, imprime uma mensagem de uso (na `stderr`) e encerra com `exit 2`;

- associa cada argumento a uma variável com nome semântico (`MODE`, `PROFILE`, `TARGET_IP`, etc.), facilitando a leitura e o registro posterior em metadados.

Em seguida, o script valida o modo, o tipo de alvo e define o perfil:

```
if [[ "$MODE" != "classico" && "$MODE" != "hibrido" ]]; then
  echo "modo inválido: $MODE (use classico|hibrido)" >&2
  exit 2
fi
if [[ "$TARGET_KIND" != "sshreal" ]]; then
  echo "TARGET_KIND deve ser 'sshreal'. Recebido: ${TARGET_KIND}" >&2
  exit 2
fi

case "$PROFILE" in
  leve)  FILE_COUNT=10; PER_FILE_MB=1;;
  medio) FILE_COUNT=10; PER_FILE_MB=10;;
  pesado) FILE_COUNT=10; PER_FILE_MB=50;;
  *) echo "perfil inválido: $PROFILE (use leve|medio|pesado)" >&2; exit 2;;
esac
```

- **validação do modo**:
  - aceita apenas `classico` ou `hibrido`;
  - qualquer outro valor implica término imediato (`exit 2`).
- **validação do alvo lógico**:
  - exige `TARGET_KIND = sshreal`; essa restrição garante que o cenário seja sempre executado contra o servidor SSH real, e não contra outros alvos de laboratório;
- **perfil de carga**:
  - `leve`: 10 arquivos de 1 MiB;
  - `medio`: 10 arquivos de 10 MiB;
  - `pesado`: 10 arquivos de 50 MiB;
  - valores inválidos geram mensagem de erro e `exit 2`.

11.2.3 Bloco 1 – estrutura de diretórios e arquivos de controle

O bloco seguinte define a estrutura de saída no lado atacante:

```
BASE_DIR="$HOME/ataques1/${MODE}/exfil/${TARGET_USER}/${PROFILE}"
TS="$(date +%Y%m%d-%H%M%S)"
RUN_DIR="${BASE_DIR}/run-${PROFILE}-${TS}"
LOCAL_OUT="${RUN_DIR}/exfil_payloads"
mkdir -p "$LOCAL_OUT" "$RUN_DIR"
SCP_LOG="${RUN_DIR}/scp.log"
META_FILE="${RUN_DIR}/meta-${TS}.txt"
```

- `BASE_DIR`: raiz lógica de exfiltração para um dado modo, usuário e perfil:

```
$HOME/ataques1/<mode>/exfil/<ssh_user>/<profile>/
```

- `TS`: timestamp da execução no formato `YYYYMMDD-HHMMSS`.
- `RUN_DIR`: diretório específico da rodada:

```
$HOME/ataques1/<mode>/exfil/<ssh_user>/<profile>/run-<profile>-<TS>/
```

- `LOCAL_OUT`: diretório que armazenará os arquivos exfiltrados.
- `SCP_LOG`: arquivo de *log* com saída detalhada das operações `scp`.
- `META_FILE`: arquivo de metadados no qual serão registrados todos os parâmetros relevantes e métricas coletadas ao longo da execução.

11.2.4 Bloco 2 – seleção de binários e parâmetros criptográficos

O script define os binários de `ssh`/`scp` e configura explicitamente KEX, cifras e *host keys*:

```
SSH_BIN="${SSH_BIN:-ssh}"
SCP_BIN="${SCP_BIN:-scp}"

FORCED_CIPHERS="aes256-ctr,aes256-gcm@openssh.com"
FORCED_HOSTKEYS="ssh-ed25519,ecdsa-sha2-nistp256,rsa-sha2-512"
if [[ "$MODE" == "hibrido" ]]; then
  FORCED_KEX="mlkem768x25519-sha256,curve25519-sha256"
  EXPECT_KEX_PATTERN="mlkem"
else
  FORCED_KEX="curve25519-sha256,ecdh-sha2-nistp256"
  EXPECT_KEX_PATTERN="curve25519|ecdh"
fi
```

- `SSH_BIN`, `SCP_BIN`: permitem sobrepor os binários padrão via variáveis de ambiente, mas assumem `ssh` e `scp` caso não sejam definidas.
- `FORCED_CIPHERS`: restringe as cifras a `aes256-ctr` e `aes256-gcm@openssh.com`.
- `FORCED_HOSTKEYS`: restringe os algoritmos de *host key* a `ssh-ed25519`, `ecdsa-sha2-nistp256` e `rsa-sha2-512`.
- `FORCED_KEX` e `EXPECT_KEX_PATTERN`:
  - no modo `hibrido`, força KEX híbrido `mlkem768x25519-sha256` (com `curve25519-sha256` de apoio) e exige que o algoritmo negociado contenha `mlkem`;
  - no modo `classico`, força apenas KEX clássicos `curve25519-sha256` e `ecdh-sha2-nistp256`, exigindo que o negociado corresponda a um desses padrões.

11.2.5 Bloco 3 – autenticação (senha ou chave privada)

O bloco seguinte decide se a autenticação será feita com senha ou chave privada:

```
SSH_KEY=""
USE_SSHPASS=0
if [[ "${ARG7#key:}" != "$ARG7" ]]; then
  SSH_KEY="${ARG7#key:}"
  if [[ -z "$SSH_KEY" || ! -f "$SSH_KEY" ]]; then
    echo "[ERRO] chave privada inválida ou não encontrada: $SSH_KEY" >&2
    exit 3
  fi
else
  SSHPASS_VAL="$ARG7"
  if [[ -z "$SSHPASS_VAL" ]]; then
    echo "[ERRO] senha vazia. Passe senha ou key:/caminho" >&2
    exit 3
  fi
  if ! command -v sshpass >/dev/null 2>&1; then
    echo "[ERRO] sshpass não encontrado. Instale para usar senha." >&2
    exit 3
  fi
  USE_SSHPASS=1
fi

MONITOR_PCAP_PATH="${MONITOR_PCAP_PATH:-}"
```

- **autenticação por chave**:
  - se o argumento 7 começa com `key:`, tudo após esse prefixo é tratado como caminho para a chave privada;
  - o script valida se o arquivo existe; caso contrário, encerra com `exit 3`.
- **autenticação por senha**:
  - se não houver prefixo `key:`, o valor é tratado como senha (`SSHPASS_VAL`);
  - senha vazia resulta em erro;
  - exige a presença do utilitário `sshpass`;
  - `USE_SSHPASS=1` indica que a autenticação é por senha.
- `MONITOR_PCAP_PATH`: opção para referenciar, nos metadados, um arquivo `pcap` gerado externamente pelo monitor (Kali Purple); é opcional e pode ser deixado em branco.

11.2.6 Bloco 4 – gravação de metadados iniciais

Neste ponto, o script grava os parâmetros-chave no arquivo de metadados:

```
{
  echo "MODE=${MODE}"
  echo "PROFILE=${PROFILE}"
```

```
  echo "TARGET_IP=${TARGET_IP}"
  echo "TARGET_PORT=${TARGET_PORT}"
  echo "TARGET_USER=${TARGET_USER}"
  echo "TARGET_KIND=${TARGET_KIND}"
  echo "TS=${TS}"
  echo "AUTH_MODE=password"
  echo "USE_SSHPASS=${USE_SSHPASS}"
  echo "SCP_BIN=${SCP_BIN}"
  echo "SSH_BIN=${SSH_BIN}"
  echo "FORCED_KEX=${FORCED_KEX}"
  echo "FORCED_CIPHERS=${FORCED_CIPHERS}"
  echo "FORCED_HOSTKEYS=${FORCED_HOSTKEYS}"
  echo "FILE_COUNT=${FILE_COUNT}"
  echo "PER_FILE_MB=${PER_FILE_MB}"
  echo "MONITOR_PCAP_PATH=${MONITOR_PCAP_PATH}"
} > "$META_FILE"
```

- consolida, em um único arquivo, as informações de contexto da rodada (modo, perfil, alvo, autenticação, algoritmos forçados, carga);
- esse arquivo será posteriormente enriquecido com informações de negociação, exfiltração, `checksums` e amostragem de recursos.

11.2.7 Bloco 5 – funções de conexão SSH e exfiltração de arquivos

Para evitar duplicação, o script define funções reutilizáveis:

```
ssh_base_opts=( -p "$TARGET_PORT" -o "StrictHostKeyChecking=no" -o "
    KexAlgorithms=${FORCED_KEX}" -o "Ciphers=${FORCED_CIPHERS}" -o "
    HostKeyAlgorithms=${FORCED_HOSTKEYS}" )

ssh_run() {
  local cmd="$1"
  if [[ -n "$SSH_KEY" ]]; then
    "$SSH_BIN" -i "$SSH_KEY" "${ssh_base_opts[@]}" "${TARGET_USER}@${TARGET_IP}"
     "$cmd"
  else
    sshpass -p "$SSHPASS_VAL" "$SSH_BIN" "${ssh_base_opts[@]}" "${TARGET_USER}@$
    {TARGET_IP}" "$cmd"
  fi
}

exfil_one() {
  local remote="$1"
  local localdest="$2"
  local scp_opts=( -v -P "$TARGET_PORT" -o "StrictHostKeyChecking=no" -o "
    KexAlgorithms=${FORCED_KEX}" -o "Ciphers=${FORCED_CIPHERS}" -o "
    HostKeyAlgorithms=${FORCED_HOSTKEYS}" )
  mkdir -p "$(dirname "$localdest")"
  if [[ -n "$SSH_KEY" ]]; then
```

```
    "$SCP_BIN" -i "$SSH_KEY" "${scp_opts[@]}" "${TARGET_USER}@${TARGET_IP}:${
    remote}" "$localdest" >>"$SCP_LOG" 2>&1 || echo "[WARN] falhou exfil de ${
    remote}" >>"$SCP_LOG"
  else
    sshpass -p "$SSHPASS_VAL" "$SCP_BIN" "${scp_opts[@]}" "${TARGET_USER}@${
    TARGET_IP}:${remote}" "$localdest" >>"$SCP_LOG" 2>&1 || echo "[WARN] falhou
    exfil de ${remote}" >>"$SCP_LOG"
  fi
}
```

- `ssh_base_opts` concentra opções de segurança e algoritmos que devem ser usados em todas as conexões;
- `ssh_run` executa um comando remoto único (para criação de arquivos, `checksums`, etc.);
- `exfil_one` baixa um arquivo remoto para o diretório local de saída, registrando as operações e eventuais avisos de falha em `scp.log`.

11.2.8 Bloco 6 – amostrador de recursos do host (sampler)

O script inclui um amostrador leve, focado em métricas do host atacante:

```
SAMPLER_INTERVAL="${SAMPLER_INTERVAL:-0.2}"
SAMPLER_SAMPLES="${SAMPLER_SAMPLES:-200}"
SAMPLER_CSV="${RUN_DIR}/sampler.csv"

sampler_loop() {
  local interval="$1"
  local maxs="$2"
  local csv="$3"

  echo "ts;sys_cpu_pct;mem_used_bytes" > "$csv"
  ...
}
```

- permite ajustar, via variáveis de ambiente, o intervalo entre amostras e o número total de amostras;
- grava um CSV com três colunas:
  - `ts`: timestamp em segundos com fração;
  - `sys_cpu_pct`: porcentagem de uso de CPU do sistema;
  - `mem_used_bytes`: memória utilizada (estimada a partir de `/proc/meminfo`).
- esse CSV é posteriormente processado para extrair médias e máximos, que são adicionados ao `META_FILE`.

11.2.9 Bloco 7 – negociação SSH e verificação de KEX

O script realiza uma conexão SSH de teste, em modo verboso, apenas para medir a negociação e extrair os algoritmos efetivamente escolhidos:

```
RUN_START_TS="$(date +%s.%N)"
DEBUG_NEG="${RUN_DIR}/ssh-negotiate.log"
NEG_HANDSHAKE_START_TS="$(date +%s.%N)"
if [[ -n "$SSH_KEY" ]]; then
  "$SSH_BIN" -i "$SSH_KEY" -vvv "${ssh_base_opts[@]}" "${TARGET_USER}@${
    TARGET_IP}" true >"$DEBUG_NEG" 2>&1 || true
else
  sshpass -p "$SSHPASS_VAL" "$SSH_BIN" -vvv "${ssh_base_opts[@]}" "${TARGET_USER
    }@${TARGET_IP}" true >"$DEBUG_NEG" 2>&1 || true
fi
NEG_HANDSHAKE_END_TS="$(date +%s.%N)"
```

- `RUN_START_TS`: marca o início global da rodada;
- `ssh-negotiate.log`: armazena a saída detalhada do SSH em modo `-vvv`;
- os timestamps de início e fim da negociação são usados para calcular `NEG_HANDSHAKE_SEC` via `python3`.

Em seguida, o script extrai KEX, cifra e *host key* e grava no `META_FILE`, verificando se o KEX negociado condiz com o modo solicitado. Caso a verificação falhe, o script aborta, mantendo o cenário coerente com a configuração criptográfica pretendida.

11.2.10 Bloco 8 – criação dos payloads remotos e SHA-256 no servidor

O script cria, no servidor `sshreal`, uma pasta temporária e os arquivos que serão exfiltrados:

```
REMOTE_DIR="/tmp/payloads_${TARGET_USER}_${PROFILE}_${TS}"
PER_FILE_BYTES=$(( PER_FILE_MB * 1024 * 1024 ))
...
ssh_run "$MKCMD"
```

- `REMOTE_DIR`: diretório remoto em `/tmp` contendo os arquivos de teste.
- `MKCMD`: bloco de comandos remotos responsável por:
  - criar `REMOTE_DIR`;
  - gerar os arquivos `file_NNNN.bin` com `fallocate` ou `dd`;
  - registrar a lista de arquivos em `remote_files.txt`.

Logo após, o script calcula `checksums` remotos (SHA-256) em `remote_shas.txt`, usando `sha256sum` ou, em alternativa, `openssl dgst -sha256`.

11.2.11 Bloco 9 – download da lista remota e dos checksums

Usando a função `exfil_one`, o script baixa para o atacante:

- a listagem remota `remote_files.txt`, salva como `remote_files_listing.txt`;
- o arquivo de `checksums` remotos `remote_shas.txt`, salvo como `remote_shas_remote.txt`.

Se qualquer um desses arquivos não puder ser obtido, o script encerra com erro, garantindo que a etapa de comparação de integridade apenas ocorra quando a referência remota estiver disponível.

11.2.12 Bloco 10 – exfiltração cronometrada com sampler

Esta é a fase central de exfiltração:

- o script registra `EXFIL_START`;
- inicia o `sampler_loop` em *background*, gravando métricas de CPU e memória;
- cria um script temporário (`EXFIL_SCRIPT`) que percorre cada arquivo listado em `remote_files_listing.txt` e usa `scp` para transferi-lo para `LOCAL_OUT`;
- executa `EXFIL_SCRIPT` envolvido por `/usr/bin/time`, que produz `exfil_time.txt` com:
  - CPU%, tempos de usuário e sistema;
  - memória máxima residente (RSS);
  - tempo de parede da exfiltração.
- ao final, registra `EXFIL_END`, `EXFIL_ELAPSED_SEC` e anexa o relatório do `time` ao `META_FILE`; o sampler é interrompido de forma ordenada.

11.2.13 Bloco 11 – checksums locais, diff e contagem de discrepâncias

Depois de concluída a exfiltração, o script:

- calcula `checksums` SHA-256 dos arquivos locais em `LOCAL_SHAS`;
- normaliza o formato dos `checksums` remotos em `remote_shas_normalized.txt`;
- ordena ambas as listas e gera um `diff` em `shas_diff.txt`;
- extrai:
  - número de linhas de diferença (`MISMATCH_DIFF_LINES`);
  - quantidade de entradas apenas no remoto (`REMOTE_ONLY_COUNT`);
  - quantidade de entradas apenas no local (`LOCAL_ONLY_COUNT`).
- registra esses indicadores no `META_FILE`, servindo como métrica de integridade da exfiltração.

11.2.14 Bloco 12 – agregação final do sampler e encerramento

Por fim, o script:

- conta o número de arquivos exfiltrados e o total de bytes em `LOCAL_OUT`;
- calcula a duração total da rodada (`RUN_ELAPSED_SEC`);
- processa `sampler.csv` para extrair médias e máximos de CPU e memória;
- registra eventuais informações de `pcap` (`HANDSHAKE_PCAP_BYTES`, `HANDSHAKE_PCAP_PKTS`);
- grava todos esses dados em `META_FILE` e imprime uma mensagem final indicando o caminho desse arquivo.

11.2.15 Exemplos de execução (todas as variações)

A seguir, são apresentadas todas as combinações relevantes de modo (`MODE`), perfil (`PROFILE`) e autenticação, supondo:

- `TARGET_IP = 192.168.50.40`;
- `TARGET_PORT = 22222`;
- `TARGET_USER = sshserver`;
- `TARGET_KIND = sshreal`.

Autenticação por senha (`sshpass`)

- modo clássico, perfil leve:

```
./ssh_exfil_compare.sh classico leve 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

- modo clássico, perfil médio:

```
./ssh_exfil_compare.sh classico medio 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

- modo clássico, perfil pesado:

```
./ssh_exfil_compare.sh classico pesado 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

- modo híbrido, perfil leve:

```
./ssh_exfil_compare.sh hibrido leve 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

- modo híbrido, perfil médio:

```
./ssh_exfil_compare.sh hibrido medio 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

- modo híbrido, perfil pesado:

```
./ssh_exfil_compare.sh hibrido pesado 192.168.50.40 22222 \
  sshserver sshreal MinhaSenhaForteAqui
```

11.3 Artefatos gerados e estrutura de saída

11.3.1 Organização de diretórios

A cada execução, o `ssh_exfil_compare.sh` organiza os artefatos de forma determinística, ancorada na raiz lógica dos ataques de exfiltração:

```
$HOME/ataques1/<mode>/exfil/<ssh_user>/<profile>/
```

em que:

- `<mode>` ∈ {`classico`, `hibrido`};
- `<ssh_user>` corresponde ao usuário remoto (`TARGET_USER`, por exemplo, `sshserver`);
- `<profile>` ∈ {`leve`, `medio`, `pesado`}.

Dentro dessa raiz, para cada execução é criado um diretório de rodada (`RUN_DIR`) com carimbo de data e hora:

```
$HOME/ataques1/<mode>/exfil/<ssh_user>/<profile>/run-<profile>-<TS>/
```

em que `<TS>` é um *timestamp* no formato `YYYYMMDD-HHMMSS`. Nesse diretório de rodada ficam concentrados:

- os arquivos exfiltrados;
- metadados da execução;
- registros da negociação SSH;
- relatórios de tempo, *logs* de `scp`;
- artefatos da etapa de `checksums`;
- resultados da amostragem de recursos do host atacante.

No lado remoto, o script utiliza um diretório temporário em `/tmp`:

```
/tmp/payloads_<TARGET_USER>_<PROFILE>_<TS>/
```

que contém os arquivos construídos para exfiltração e é utilizado apenas durante a rodada.

11.3.2 Metadados e logs principais

Os arquivos de controle e *logs* centrais no `RUN_DIR` são:

- `meta-<TS>.txt` Arquivo de metadados principal da rodada. É criado no início da execução e enriquecido ao longo do script. Entre as informações registradas, destacam-se:
  - parâmetros de contexto: `MODE`, `PROFILE`, `TARGET_IP`, `TARGET_PORT`, `TARGET_USER`, `TARGET_KIND`, `TS`, `FILE_COUNT`, `PER_FILE_MB`;
  - detalhes de autenticação e binários utilizados: `AUTH_MODE`, `USE_SSHPASS`, `SSH_BIN`, `SCP_BIN`;
  - parâmetros criptográficos forçados: `FORCED_KEX`, `FORCED_CIPHERS`, `FORCED_HOSTKEYS`;
  - métricas da negociação SSH (`NEGOTIATED_KEX`, `NEGOTIATED_CIPHER`, `NEGOTIATED_HOSTKEY`, `NEG_HANDSHAKE_*`);
  - métricas de exfiltração (`EXFIL_START`, `EXFIL_END`, `EXFIL_ELAPSED_SEC`, `EXFIL_TIME_REPORT`);
  - campos derivados do `/usr/bin/time`, prefixados com `TIME:` (CPU%, tempo usuário/sistema, memória máxima);
  - resultados de `checksums` e diffs: arquivos utilizados, contadores de discrepâncias (`MISMATCH_DIFF_LINES`, `REMOTE_ONLY_COUNT`, `LOCAL_ONLY_COUNT`);
  - agregados do `sampler` (`SAMPLER_SAMPLES`, `SAMPLER_INTERVAL`, `SYS_CPU_PCT_AVG`, `SYS_CPU_PCT_MAX`, `MEM_USED_BYTES_AVG`, `MEM_USED_BYTES_MAX`);
  - indicadores finais da rodada: número de arquivos exfiltrados, total de bytes, tempo total de execução.
- `scp.log` *Log* detalhado das operações de cópia `scp`, incluindo:
  - comandos de exfiltração por arquivo (modo verboso);
  - mensagens de aviso sobre falhas pontuais de exfiltração;
  - confirmações de transferência bem-sucedida.

  Esse arquivo é útil para depuração de problemas de conectividade ou autenticação.
- `ssh-negotiate.log` Registro completo, em modo `-vvv`, da conexão SSH utilizada para medir a negociação de algoritmos. Permite:
  - inspecionar o KEX efetivamente negociado;
  - verificar cifras e algoritmos de *host key*;
  - analisar mensagens de erro em caso de incompatibilidade entre o modo solicitado (`classico` ou `hibrido`) e o algoritmo de KEX escolhido pelo servidor.

11.3.3 Payloads remotos e arquivos exfiltrados

A carga manipulada pelo cenário de exfiltração se distribui entre o servidor remoto e o atacante.

No servidor `sshreal` (lado remoto)

No diretório temporário `/tmp/payloads_<TARGET_USER>_<PROFILE>_<TS>`, o script cria:

- arquivos binários de teste:
  - `file_0001.bin`, `file_0002.bin`, ..., `file_NNNN.bin`;
  - o número de arquivos (`FILE_COUNT`) e o tamanho de cada um (`PER_FILE_MB`) dependem do perfil (`leve`, `medio`, `pesado`).
- `remote_files.txt` arquivo texto contendo a listagem nominal dos arquivos gerados (`file_*.bin`), usado como referência para a exfiltração.
- `remote_shas.txt` arquivo com `checksums` SHA-256 dos arquivos remotos, calculados com `sha256sum` ou, em alternativa, com `openssl dgst -sha256`.

Na máquina atacante (lado local)

No diretório `exfil_payloads/`, dentro do `RUN_DIR`, são armazenados os arquivos exfiltrados:

- cópias locais dos arquivos `file_NNNN.bin` obtidos via `scp`;
- a integridade de cada arquivo é verificada por meio dos `checksums` locais e da comparação com os `checksums` remotos.

O `META_FILE` registra, ao final, o número de arquivos presentes em `exfil_payloads/` e o total de bytes, permitindo verificar se a quantidade e o volume transferido correspondem ao planejado para o perfil executado.

11.3.4 Checksums e arquivos de comparação

Para validar a integridade fim a fim, o script mantém um conjunto de arquivos auxiliares relacionados a `checksums`:

- `remote_files_listing.txt` cópia local do `remote_files.txt` gerado no servidor. É utilizada tanto para o laço de exfiltração quanto como referência nominal dos arquivos esperados.
- `remote_shas_remote.txt` cópia local de `remote_shas.txt`, contendo os `hashes` SHA-256 calculados no lado remoto.
- `remote_shas_normalized.txt` versão normalizada dos `checksums` remotos. Esse arquivo:
  - converte diferentes formatos de saída (`sha256sum`, `openssl`) em uma forma uniforme `<hash> <arquivo>`;
  - prepara os dados para comparação direta com os `checksums` locais.
- `remote_shas_sorted.txt` versão ordenada da lista normalizada de `checksums` remotos. É utilizada como entrada no `diff`.

- `local_shas.txt` arquivo contendo os `checksums` SHA-256 calculados localmente sobre os arquivos em `exfil_payloads/`. Quando possível, é usado `sha256sum`; caso contrário, o script recorre ao `openssl dgst -sha256`.
- `local_shas_sorted.txt` versão ordenada dos `checksums` locais, no mesmo formato dos remotos, utilizada na comparação linha a linha.
- `shas_diff.txt` resultado do `diff` unificado entre as listas ordenadas remota e local. Permite identificar:
  - arquivos presentes apenas no remoto (*remote-only*);
  - arquivos presentes apenas no local (*local-only*);
  - diferenças de `hash` para um mesmo nome de arquivo.

  Os contadores `MISMATCH_DIFF_LINES`, `REMOTE_ONLY_COUNT` e `LOCAL_ONLY_COUNT` são derivados desse arquivo e gravados em `meta-<TS>.txt`.

11.3.5 Métricas de exfiltração e amostragem de recursos

Para caracterizar o comportamento da exfiltração e o impacto no sistema, o script produz ainda artefatos específicos de tempo e recursos:

- `exfil_time.txt` arquivo gerado pelo `/usr/bin/time`, contendo métricas como:
  - percentual de CPU utilizado (`CPU_PERCENT`);
  - tempo de CPU em modo usuário (`USER_SEC`) e sistema (`SYS_SEC`);
  - memória máxima residente (`MAX_RSS_KB`);
  - tempo real decorrido (`ELAPSED_REAL`).

  As linhas desse arquivo são incorporadas ao `META_FILE` com o prefixo `TIME:`, facilitando a inspeção posterior.
- `sampler.csv` arquivo CSV produzido pela função `sampler_loop`, em que cada linha (após o cabeçalho) contém:
  - `ts`: carimbo de tempo em segundos;
  - `sys_cpu_pct`: uso de CPU do sistema em porcentagem;
  - `mem_used_bytes`: memória utilizada em bytes.

  Esse arquivo registra a evolução do consumo de recursos da máquina atacante durante a exfiltração.
- `sampler.parsed` arquivo intermediário, derivado do `sampler.csv` por meio de `awk`, contendo agregados como:
  - média e máximo de `sys_cpu_pct`;
  - média e máximo de `mem_used_bytes`.

  Esses valores são lidos e registrados em `meta-<TS>.txt`.

Adicionalmente, quando fornecido um caminho de captura externa no MONITOR_PCAP_PATH, o script inclui nos metadados indicadores como HANDSHAKE_PCAP_BYTES e HANDSHAKE_PCAP_PKTS, permitindo correlacionar os resultados de exfiltração com os registros de tráfego capturados pela máquina de monitoramento.

11.4 Análise de um arquivo de metadados meta-*.txt

Este exemplo apresenta um arquivo meta-*.txt real gerado pelo ssh_exfil_compare.sh em modo hibrido e perfil medio, durante uma execução de exfiltração controlada contra o servidor sshreal. O objetivo é ilustrar como os campos são organizados e como podem ser interpretados na análise dos experimentos.

11.4.1 Registro bruto do arquivo de metadados

```
MODE=hibrido
PROFILE=medio
TARGET_IP=192.168.50.40
TARGET_PORT=22222
TARGET_USER=sshserver
TARGET_KIND=sshreal
TS=20251112-023927
AUTH_MODE=password
USE_SSHPASS=1
SCP_BIN=scp
SSH_BIN=ssh
FORCED_KEX=mlkem768x25519-sha256,curve25519-sha256
FORCED_CIPHERS=aes256-ctr,aes256-gcm@openssh.com
FORCED_HOSTKEYS=ssh-ed25519,ecdsa-sha2-nistp256,rsa-sha2-512
FILE_COUNT=10
PER_FILE_MB=10
MONITOR_PCAP_PATH=
NEGOTIATED_KEX=mlkem768x25519-sha256
NEGOTIATED_CIPHER=aes256-ctr
NEGOTIATED_HOSTKEY=ssh-ed25519
NEG_HANDSHAKE_START_TS=1762929567.816349902
NEG_HANDSHAKE_END_TS=1762929568.064650086
NEG_HANDSHAKE_SEC=0.248
EXFIL_START=1762929570.329621180
EXFIL_END=1762929573.912558406
EXFIL_ELAPSED_SEC=3.583
EXFIL_TIME_REPORT=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-
    medio-20251112-023927/exfil_time.txt
FILES_LIST=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-medio
    -20251112-023927/remote_files_listing.txt
TIME: CPU_PERCENT=40%
TIME: USER_SEC=0.74
TIME: SYS_SEC=0.66
TIME: MAX_RSS_KB=10816
TIME: ELAPSED_REAL=3.50
```

```
REMOTE_SHAS_FILE=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-medio
    -20251112-023927/remote_shas_remote.txt
LOCAL_SHAS_FILE=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-medio
    -20251112-023927/local_shas.txt
REMOTE_SHAS_NORMALIZED=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run
    -medio-20251112-023927/remote_shas_normalized.txt
REMOTE_SHAS_SORTED=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-
    medio-20251112-023927/remote_shas_sorted.txt
LOCAL_SHAS_SORTED=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-
    medio-20251112-023927/local_shas_sorted.txt
SHAS_DIFF=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-medio
    -20251112-023927/shas_diff.txt
MISMATCH_DIFF_LINES=0
REMOTE_ONLY_COUNT=0
LOCAL_ONLY_COUNT=0
SAMPLER_SAMPLES=200
SAMPLER_INTERVAL=0.2
SAMPLER_CSV=/home/attacker/ataques1/hibrido/exfil/sshserver/medio/run-medio
    -20251112-023927/sampler.csv
SYS_CPU_PCT_AVG=12.562500
SYS_CPU_PCT_MAX=24
MEM_USED_BYTES_AVG=1961261568.000000
MEM_USED_BYTES_MAX=1976066048
HANDSHAKE_PCAP_BYTES=pcap_not_provided
HANDSHAKE_PCAP_PKTS=pcap_not_provided
LOCAL_COUNT=10
LOCAL_TOTAL_BYTES=104857600
RUN_START_TS=1762929567.814370265
RUN_END_TS=1762929574.302799342
RUN_ELAPSED_SEC=6.488
```

11.4.2 Bloco 1 – Contexto da rodada de exfiltração

- `MODE=hibrido` e `PROFILE=medio`: a rodada usa KEX híbrido (componentes pós-quânticos + clássicos) e o perfil de carga `medio`. Neste cenário, o perfil `medio` foi configurado como 10 arquivos de 10 MiB (ver campos abaixo).
- `TARGET_IP=192.168.50.40`, `TARGET_PORT=22222`, `TARGET_USER=sshserver`, `TARGET_KIND=sshreal`: identificam o servidor SSH real que participa do cenário de exfiltração:
  - IP e porta da instância recompilada de `sshd`;
  - usuário remoto (`sshserver`);
  - rótulo lógico `sshreal`, garantindo que o experimento está associado ao servidor definido.
- `TS=20251112-023927`: carimbo de data e hora (formato `YYYYMMDD-HHMMSS`) da rodada, utilizado na composição do `RUN_DIR` e na correlação com outras capturas (por exemplo, `pcap` e *logs*).

- `FILE_COUNT=10` e `PER_FILE_MB=10`: definem o volume de dados remotos a serem exfiltrados:
  - 10 arquivos;
  - 10 MiB por arquivo;
  - total previsto de aproximadamente 100 MiB no lado remoto.

11.4.3 Bloco 2 – Autenticação e binários utilizados

- `AUTH_MODE=password` e `USE_SSHPASS=1`: indicam que a autenticação foi realizada com senha, via `sshpass`. Em execuções com `key:...`, `AUTH_MODE` seria coerente com autenticação por chave privada e `USE_SSHPASS` tenderia a 0.
- `SCP_BIN=scp` e `SSH_BIN=ssh`: binários usados para transferência de arquivos e sessões SSH. Esses campos permitem documentar eventuais variações (por exemplo, um `scp` ou `ssh` customizados em outro caminho).

11.4.4 Bloco 3 – Parâmetros criptográficos forçados (KEX, cifras, host keys)

- `FORCED_KEX=mlkem768x25519-sha256,curve25519-sha256`: lista de algoritmos de troca de chaves (KEX) explicitamente forçados na linha de comando:
  - `mlkem768x25519-sha256` (híbrido PQC+clássico);
  - `curve25519-sha256` como alternativa clássica.
- `FORCED_CIPHERS=aes256-ctr,aes256-gcm@openssh.com`: restringe as cifras simétricas a `aes256-ctr` e `aes256-gcm@openssh.com`, alinhando a exfiltração com o foco em AES de 256 bits.
- `FORCED_HOSTKEYS=ssh-ed25519,ecdsa-sha2-nistp256,rsa-sha2-512`: define os algoritmos de *host key* aceitos para o servidor, evitando que a sessão utilize chaves de servidor fora desse conjunto.

11.4.5 Bloco 4 – Negociação SSH (handshake) e algoritmos efetivos

- `NEGOTIATED_KEX=mlkem768x25519-sha256`: algoritmo de KEX efetivamente negociado durante o handshake SSH. Neste exemplo, o servidor aceitou o KEX híbrido, compatível com o modo `hibrido` solicitado.
- `NEGOTIATED_CIPHER=aes256-ctr`: cifra simétrica escolhida para proteger o canal de dados da exfiltração (modo CTR com chave de 256 bits).
- `NEGOTIATED_HOSTKEY=ssh-ed25519`: algoritmo de *host key* utilizado pelo servidor, baseado em curvas elípticas (`ed25519`).
- `NEG_HANDSHAKE_START_TS=1762929567.816349902` e `NEG_HANDSHAKE_END_TS=1762929568.064650086`: timestamps de início e fim da negociação SSH em segundos (com fração), medidos a partir da época Unix.

- `NEG_HANDSHAKE_SEC=0.248`: duração aproximada, em segundos, da fase de handshake utilizada para negociar KEX, cifra e *host key*. Este valor é apropriado para comparação direta entre modos `classico` e `hibrido`.

11.4.6 Bloco 5 – Janela temporal da exfiltração e métricas de tempo

- `EXFIL_START=1762929570.329621180` e `EXFIL_END=1762929573.912558406`: marcam, em segundos com fração, o início e o fim da fase de exfiltração dos arquivos via `scp`.
- `EXFIL_ELAPSED_SEC=3.583`: duração da exfiltração em segundos, calculada como diferença entre `EXFIL_END` e `EXFIL_START`. Esta é a métrica principal a ser comparada entre rodadas com KEX clássico e híbrido.
- `EXFIL_TIME_REPORT=.../exfil_time.txt`: caminho do arquivo gerado por `/usr/bin/time`, contendo uma visão mais detalhada do uso de CPU e memória durante a exfiltração.
- `FILES_LIST=.../remote_files_listing.txt`: listagem local dos arquivos remotos que foram exfiltrados. Esse arquivo é utilizado para percorrer a carga alvo e para validar se o conjunto exfiltrado está completo.

11.4.7 Bloco 6 – Saída do `/usr/bin/time` (`TIME:*`)

Os campos prefixados por `TIME:` são derivados do `exfil_time.txt` e sumarizam o comportamento da exfiltração sob a ótica do `/usr/bin/time`:

- `TIME: CPU_PERCENT=40%`: porcentagem aproximada de CPU utilizada pelo processo de exfiltração (incluindo `scp` e auxiliares) durante o período medido.
- `TIME: USER_SEC=0.74` e `TIME: SYS_SEC=0.66`: tempo de CPU gasto em modo usuário e em modo kernel, respectivamente.
- `TIME: MAX_RSS_KB=10816`: memória máxima residente (RSS), em kilobytes, utilizada pelo processo medido durante a exfiltração.
- `TIME: ELAPSED_REAL=3.50`: tempo real decorrido reportado pelo `time` (em segundos), que deve ser consistente com `EXFIL_ELAPSED_SEC` dentro de pequenas variações de arredondamento.

11.4.8 Bloco 7 – Artefatos de `checksums` e verificação de integridade

Este bloco descreve os arquivos auxiliares usados na comparação remoto versus local:

- `REMOTE_SHAS_FILE` e `LOCAL_SHAS_FILE`: caminhos para os arquivos de `checksums` SHA-256 calculados no lado remoto e no lado local, respectivamente.

- REMOTE_SHAS_NORMALIZED: versão normalizada dos checksums remotos (formato uniforme <hash> <arquivo>), capaz de acomodar saídas de sha256sum ou de openssl dgst -sha256.
- REMOTE_SHAS_SORTED e LOCAL_SHAS_SORTED: versões ordenadas (por nome de arquivo) das listas de hashes remotos e locais, preparadas para comparação com diff.
- SHAS_DIFF: arquivo contendo o diff entre as listas ordenadas, identificando discrepâncias de integridade ou diferenças no conjunto de arquivos.
- MISMATCH_DIFF_LINES=0: número total de linhas de diferença no diff unificado. Valor igual a zero indica que não houve divergência de hash nem de presença/ausência de arquivos.
- REMOTE_ONLY_COUNT=0 e LOCAL_ONLY_COUNT=0: contadores de entradas presentes apenas no remoto ou apenas no local. Ambos em zero significam que o conjunto de arquivos remotos foi totalmente exfiltrado, sem sobras nem faltas.

11.4.9 Bloco 8 – Amostrador de recursos do host atacante

- SAMPLER_SAMPLES=200 e SAMPLER_INTERVAL=0.2: indicam que o amostrador registrou 200 amostras com intervalo de 0,2 s entre elas, cobrindo aproximadamente 40 s de observação de recursos do host atacante.
- SAMPLER_CSV=.../sampler.csv: arquivo CSV contendo as amostras brutas, com colunas de tempo, uso de CPU e memória em bytes.
- SYS_CPU_PCT_AVG=12.562500 e SYS_CPU_PCT_MAX=24: uso médio e máximo de CPU do sistema, em porcentagem, ao longo da janela de amostragem.
- MEM_USED_BYTES_AVG=1961261568.000000 e MEM_USED_BYTES_MAX=1976066048: consumo médio e máximo de memória em bytes durante a execução da rodada, permitindo comparar o impacto de diferentes modos de KEX sobre o comportamento de recursos.

11.4.10 Bloco 9 – Correlação com capturas no monitor e contadores finais

- MONITOR_PCAP_PATH= (vazio), HANDSHAKE_PCAP_BYTES=pcap_not_provided e HANDSHAKE_PCAP_PKTS=pcap_not_provided: indicam que, nesta execução específica, não foi fornecido um caminho para captura pcap gerada pelo monitor (Kali Purple). Em execuções onde um pcap é produzido e referenciado, esses campos podem ser preenchidos com tamanho em bytes e número de pacotes da captura correspondente ao handshake/exfiltração.
- LOCAL_COUNT=10 e LOCAL_TOTAL_BYTES=104857600: confirmam que 10 arquivos foram exfiltrados para o diretório local e que o volume total corresponde a 104 857 600 bytes (10 arquivos × 10 MiB), como esperado para o perfil medio.
- RUN_START_TS=1762929567.814370265 e RUN_END_TS=1762929574.302799342: timestamps globais de início e fim da rodada, englobando desde a negociação SSH até o pós-processo final.

- `RUN_ELAPSED_SEC=6.488`: duração total da rodada, em segundos, calculada como diferença entre `RUN_END_TS` e `RUN_START_TS`. Este valor é útil para uma visão de "custo global da execução", complementando a métrica focada apenas na exfiltração (`EXFIL_ELAPSED_SEC`).

Em conjunto, este arquivo `meta-*.txt` fornece uma visão compacta e reprodutível da rodada de exfiltração: modo criptográfico utilizado, parâmetros de KEX/cifra/*host key*, janela temporal de exfiltração, integridade fim a fim dos arquivos e impacto em recursos da máquina atacante.

Parte VI

Referências

Referências Bibliográficas

[1] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Post-quantum cryptography – project overview*. Gaithersburg, 2024. Disponível em: https://csrc.nist.gov/projects/post-quantum-cryptography. Acesso em: 28 nov. 2025.

[2] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)*. Gaithersburg, 2024. Disponível em: https://csrc.nist.gov/pubs/fips/203/final. Acesso em: 28 nov. 2025.

[3] OPEN QUANTUM SAFE. *liboqs – C library for quantum-resistant cryptography*. 2025. Disponível em: https://openquantumsafe.org/liboqs/. Acesso em: 28 nov. 2025.

[4] OPEN QUANTUM SAFE. *Open Quantum Safe project – home page*. 2025. Disponível em: https://openquantumsafe.org. Acesso em: 28 nov. 2025.

[5] OPEN QUANTUM SAFE. *oqs-provider – OpenSSL 3 provider for post-quantum cryptography*. 2025. Disponível em: https://github.com/open-quantum-safe/oqs-provider. Acesso em: 28 nov. 2025.

[6] OPENSSH. *OpenSSH – secure shell connectivity tools*. 2025. Disponível em: https://www.openssh.com. Acesso em: 28 nov. 2025.

[7] OPENSSH. *OpenSSH release notes*. 2025. Disponível em: https://www.openssh.com/releasenotes.html. Acesso em: 28 nov. 2025.

[8] OPENSSL PROJECT. *OpenSSL – cryptography and SSL/TLS toolkit*. 2025. Disponível em: https://www.openssl.org. Acesso em: 28 nov. 2025.

[9] OPENSSL PROJECT. *OpenSSL: TLS/SSL and crypto library – GitHub repository*. 2025. Disponível em: https://github.com/openssl/openssl. Acesso em: 28 nov. 2025.

[10] DEBIAN PROJECT. *Debian – the universal operating system*. 2025. Disponível em: https://www.debian.org. Acesso em: 28 nov. 2025.

[11] DEBIAN PROJECT. *Debian "bookworm" – release information*. 2025. Disponível em: https://www.debian.org/releases/bookworm/. Acesso em: 28 nov. 2025.

[12] DEBIAN PROJECT. *Atualização Debian 12: 12.12 lançado*. 2025. Disponível em: https://www.debian.org/News/2025/2025090602.pt.html. Acesso em: 28 nov. 2025.

[13] DEBIAN PROJECT. *Updated Debian 12: 12.12 released*. 2025. Disponível em: https://www.debian.org/News/2025/2025090602. Acesso em: 28 nov. 2025.

[14] OFFENSIVE SECURITY. *Kali Linux – official website*. 2025. Disponível em: https://www.kali.org. Acesso em: 28 nov. 2025.

[15] OFFENSIVE SECURITY. *Kali Linux documentation*. 2025. Disponível em: https://www.kali.org/docs/. Acesso em: 28 nov. 2025.

[16] OFFENSIVE SECURITY. *Installing Kali Linux on hard disk*. 2025. Disponível em: https://www.kali.org/docs/installation/hard-disk-install/. Acesso em: 28 nov. 2025.

[17] OFFENSIVE SECURITY. *Get Kali – download Kali Linux images*. 2025. Disponível em: https://www.kali.org/get-kali/. Acesso em: 28 nov. 2025.

[18] OFFENSIVE SECURITY. *Kali Linux 2023.1 release: Kali Purple & Python changes*. 2023. Disponível em: https://www.kali.org/blog/kali-linux-2023-1-release/. Acesso em: 28 nov. 2025.

[19] OFFENSIVE SECURITY. *Kali Purple documentation*. 2024. Disponível em: https://gitlab.com/kalilinux/kali-purple/documentation. Acesso em: 28 nov. 2025.

[20] THE OPEN INFORMATION SECURITY FOUNDATION. *Suricata – home page*. 2025. Disponível em: https://suricata.io. Acesso em: 28 nov. 2025.

[21] THE OPEN INFORMATION SECURITY FOUNDATION. *Suricata user guide*. 2023. Disponível em: https://docs.suricata.io. Acesso em: 28 nov. 2025.

[22] THE OPEN INFORMATION SECURITY FOUNDATION. *Suricata documentation portal*. 2023. Disponível em: https://suricata.io/documentation/. Acesso em: 28 nov. 2025.

[23] THE ZEEK PROJECT. *Zeek – the network security monitor*. 2020. Disponível em: https://zeek.org. Acesso em: 28 nov. 2025.

[24] THE ZEEK PROJECT. *Book of Zeek – Zeek documentation*. 2017. Disponível em: https://docs.zeek.org/en/master/. Acesso em: 28 nov. 2025.

[25] THE ZEEK PROJECT. *Zeek – GitHub repository*. 2012. Disponível em: https://github.com/zeek/zeek. Acesso em: 28 nov. 2025.

[26] SPLUNK INC. *Splunk Enterprise – product page*. 2025. Disponível em: https://www.splunk.com/en_us/products/splunk-enterprise.html. Acesso em: 28 nov. 2025.

[27] SPLUNK INC. *Splunk Docs – documentation portal*. 2024. Disponível em: https://help.splunk.com/en. Acesso em: 28 nov. 2025.

[28] SPLUNK INC. *Splunk Enterprise documentation*. 2024. Disponível em: https://help.splunk.com/en/splunk-enterprise. Acesso em: 28 nov. 2025.

[29] SPLUNK INC. *About Splunk Enterprise*. 2025. Disponível em: https://help.splunk.com/en/splunk-enterprise/get-started/overview/10.0/about-splunk-enterprise/about-splunk-enterprise. Acesso em: 28 nov. 2025.

[30] WIRESHARK FOUNDATION. *Wireshark documentation portal*. 2019. Disponível em: https://www.wireshark.org/docs/. Acesso em: 28 nov. 2025.

[31] WIRESHARK FOUNDATION. *Wireshark user's guide*. 2019. Disponível em: https://www.wireshark.org/docs/wsug_html/. Acesso em: 28 nov. 2025.

[32] NMAP PROJECT. *Nmap documentation – official site*. 1997. Disponível em: https://nmap.org/docs.html. Acesso em: 28 nov. 2025.

[33] NMAP PROJECT. *Nmap reference guide*. 2009. Disponível em: https://nmap.org/book/man.html. Acesso em: 28 nov. 2025.

[34] TCPDUMP GROUP. *Tcpdump and libpcap – official website*. 2025. Disponível em: https://www.tcpdump.org. Acesso em: 28 nov. 2025.

[35] TCPDUMP GROUP. *Tcpdump man pages – online manual*. 2025. Disponível em: https://www.tcpdump.org/manpages/. Acesso em: 28 nov. 2025.