Tutorial de Instalação do Remnux no VirtualBox e Ferramentas

Capítulo 1 - Instalação

1 - Introdução

REMnux é uma distribuição Linux projetada especificamente para a análise de malware e a realização de tarefas relacionadas à segurança cibernética. Criada por Lenny Zeltser e mantida pela comunidade, REMnux fornece uma vasta coleção de ferramentas gratuitas que facilitam o processo de desmontagem de malwares, investigações forenses, análise de documentos maliciosos, e muito mais, sem a necessidade de instalação e configuração complexa dessas ferramentas. Equipada com utilitários para a análise estática e dinâmica de malwares, a REMnux é altamente valorizada por analistas de segurança e pesquisadores de malware por sua conveniência e eficácia. Além de ferramentas específicas para a engenharia reversa de software e análise de artefatos suspeitos, a REMnux também inclui recursos para simular redes e interações de rede, tornando-a uma solução abrangente para profissionais de segurança cibernética enfrentarem desafios modernos de malware.

Neste tutorial vamos apresentar como instalar o REMnux em uma VM utilizando um arquivo pré-configurado (OVA).

2 - Passo-a-Passo

1- Baixar o Arquivo OVA do Remnux

Acesse o site oficial do Remnux (https://remnux.org/) e localize a seção de download do arquivo OVA.
Escolha entre as opções de download disponíveis (por exemplo, Box ou SourceForge) e baixe o arquivo OVA.

2 - Iniciar o VirtualBox

Neste tutorial iremos utilizar o VirtualBox. O processo para outras soluções de VM é análogo.

3 - Importar o Arquivo OVA

No menu do VirtualBox, clique em "Arquivo" e selecione "Importar Appliance".
Na janela que se abre, clique no ícone de pasta (à direita) para procurar o arquivo OVA do Remnux que você baixou.

Selecione o arquivo OVA e clique em “Abrir".

Clique em "Próximo" ou "Continuar" para revisar as configurações da appliance que serão importadas. Aqui, você pode modificar algumas configurações, como o nome da VM, o local de armazenamento no seu sistema, quantidades de CPU e memória RAM alocadas, etc. No entanto, geralmente, as configurações padrão fornecidas pelo arquivo OVA são adequadas.
Após revisar (e possivelmente modificar) as configurações, clique em "Importar" para iniciar o processo de importação.

4 - Iniciar a Máquina Virtual Remnux

Selecione a VM recém-importada na lista do VirtualBox e clique em "Iniciar" para inicializar a máquina virtual.
O Remnux deverá iniciar como qualquer outro sistema operacional numa VM.

5 -Pós-Instalação

Atualize as Ferramentas: É uma boa prática verificar se há atualizações para as ferramentas incluídas no Remnux logo após a instalação. Abra um terminal dentro do Remnux e execute o comando *remnux upgrade* para atualizar as ferramentas.

Capítulo 2 - Ferramentas de Análise Estática

A análise estática de malware é uma técnica utilizada para inspecionar o código de um programa malicioso sem executá-lo. Essa abordagem envolve o exame do código-fonte (quando disponível) ou do código de máquina (binário) para entender o comportamento, as funcionalidades e as intenções do malware. Ferramentas e métodos como descompiladores, disassemblers, e analisadores de strings são frequentemente empregados para extrair informações úteis do código, como rotinas maliciosas, chamadas de sistema, strings ofuscadas, e padrões de comunicação de rede. A análise estática permite aos pesquisadores identificar assinaturas de malware, vulnerabilidades exploradas, e potenciais impactos sem o risco de infecção. Além disso, essa técnica é crucial para a criação de assinaturas de detecção, entendimento de técnicas de evasão e análise de lógica de ataque, fornecendo uma base sólida para a compreensão aprofundada de amostras de malware.

1 - oletools

1.1 - Introdução

O oletools é um pacote de ferramentas Python de código aberto destinado à análise de arquivos do Microsoft Office para fins de segurança, procurando por indicações de malware, especialmente aqueles que utilizam formatos OLE (Object Linking and Embedding). As ferramentas dentro do oletools podem ajudar a detectar malwares escondidos em documentos, planilhas e apresentações, frequentemente camuflados como macros VBA ou formas obfuscadas de scripts.

Embora o oletools já venha instalado no REMnux, pode ser necessário atualizá-lo ou instalá-lo em outros ambientes. Para isso, você pode usar o pip, o gerenciador de pacotes do Python

```
pip install  oletools
```

1.2 - Uso Básico e Funcionalidades

O oletools inclui várias ferramentas úteis, cada uma com seu propósito específico. Aqui estão algumas das principais ferramentas e como usá-las:

1.2.1 - olevba

olevba analisa documentos do Office em busca de macros VBA maliciosas, suportando formatos como OLE, OpenXML (docx, xlsx, pptx), e mesmo arquivos de texto que contenham código VBA extraído.

Sintaxe:

olevba <nome_do_arquivo>

Principais Parâmetros:

-l, --loglevel: Define o nível de log (DEBUG, INFO, WARNING, ERROR, CRITICAL).
-c, --reveal: Converte strings obfuscadas para seu estado original, ajudando na análise.

1.2.2 - oleid

oleid é projetado para identificar propriedades de arquivos OLE que indicam potencial para conter malware, como a presença de macros, objetos Flash, ou estruturas suspeitas.

oleid <nome_do_arquivo>

1.2.3 - mraptor (MacroRaptor):

mraptor é uma ferramenta que analisa macros VBA em busca de comportamentos suspeitos ou maliciosos.

mraptor <nome_do_arquivo>

Principais Parâmetros:
-s, --scan: Escaneia o arquivo fornecido em busca de macros suspeitas.

1.2.4 - pyxswf

pyxswf é uma ferramenta que extrai objetos SWF (Adobe Flash) de documentos do Office, que podem ser usados para ataques maliciosos.

```
pyxswf <nome_do_arquivo>
```

1.2.5 - oleobj

oleobj extrai objetos OLE embutidos em documentos do Office, como arquivos executáveis embutidos ou outros objetos potencialmente maliciosos.

```
oleobj <nome_do_arquivo>
```

2 - peepdf

2.1 - peepdf

O peepdf é uma ferramenta de análise de arquivos PDF escrita em Python, projetada para explorar arquivos PDF em busca de potenciais malwares ou elementos suspeitos. Ela é capaz de identificar estruturas comuns usadas em ataques de PDF, incluindo obfuscations, scripts maliciosos e explorações de vulnerabilidades conhecidas nos leitores de PDF. O peepdf fornece um ambiente interativo que facilita a inspeção profunda dos elementos internos de um PDF, tornando-a uma ferramenta valiosa para analistas de segurança.

O peepdf é uma das muitas ferramentas pré-instaladas no REMnux para a análise de malwares, incluindo análises específicas de arquivos PDF. Se você precisar instalar ou atualizar o peepdf em uma instalação do REMnux ou em outro sistema, você pode fazê-lo através do GitHub ou usando o pip, por exemplo:

```
git clone https://github.com/jesparza/peepdf.git
```

Ou, se preferir usar o pip:

```
pip install peepdf
```

2.2 - Uso Básico e Funcionalidades

O peepdf oferece um poderoso conjunto de funcionalidades para a análise de documentos PDF. Vamos explorar algumas das principais funcionalidades e como utilizá-las:

2.2.1 - Iniciando a Análise Interativa

Para começar uma análise interativa de um arquivo PDF, simplesmente execute o peepdf com o caminho do arquivo:

```
peepdf <nome_do_arquivo.pdf>
```

Isso iniciará o shell interativo do peepdf, onde você pode executar diversos comandos para analisar o documento PDF.

2.2.2 - Visualizando a Árvore de Objetos

Um dos primeiros passos na análise de um PDF pode ser visualizar sua estrutura de objetos. Use o comando:

```
tree
```

Isso exibirá a estrutura de objetos do PDF, ajudando você a identificar elementos suspeitos.

2.2.3 - Examinando Objetos Específicos

Para examinar um objeto específico dentro do PDF, você pode usar o comando object seguido pelo número do objeto:

```
object  <num_do_objeto>
```

Isso exibirá detalhes do objeto, incluindo seu conteúdo e metadados.

2.2.4 - Pesquisando por Palavras-Chave

O peepdf permite pesquisar dentro do PDF por palavras-chave específicas, o que pode ser útil para encontrar scripts maliciosos ou obfuscados:

```
search javascript
```

Isso procurará por ocorrências da palavra "javascript" no documento, uma técnica comum em PDFs maliciosos.

2.2.5 - Analisando Streams

Muitas vezes, o conteúdo malicioso em PDFs está oculto em streams obfuscados. Você pode desobfuscar e analisar esses streams com o comando stream:

```
stream <n>
```

Este comando tentará desobfuscar e mostrar o conteúdo do stream do objeto <n>.

2.2.6 - Principais Parâmetros

-f: Força a análise do arquivo, mesmo que seja detectado como malicioso.
-l: Lista todos os comandos disponíveis no shell interativo do peepdf.
-s <script_file>: Executa um script de análise, permitindo automatizar tarefas.
-c —key <api_key>: Verifica o hash do PDF usando a API do VirusTotal

3 - ExifTool

3.1 - Introdução

O ExifTool é uma ferramenta de software livre e de código aberto, escrita em Perl, que é utilizada para ler, escrever e editar metadados em uma grande variedade de arquivos. Embora seja amplamente conhecido por sua capacidade de manipular metadados de imagens, o ExifTool também suporta muitos outros tipos de arquivos, incluindo documentos do Office, arquivos PDF, arquivos de áudio e vídeo. Ele pode ser usado para extrair informações de metadados, que podem revelar detalhes sobre a origem de um arquivo, alterações feitas, geolocalização (para imagens) e muito mais, o que é particularmente útil em investigações forenses digitais e análises de segurança.

O ExifTool vem pré-instalado no REMnux, caso você precise instalar o ExifTool em um sistema que não seja o REMnux, ou se precisar de uma versão atualizada, você pode instalá-lo através do gerenciador de pacotes da sua distribuição ou baixá-lo diretamente do site oficial. Para a maioria das distribuições Linux, o comando para instalar o ExifTool é:

```
sudo apt-get install libimage-exiftool-perl
```

3.2 - Uso Básico e Funcionalidades

3.2.1 - Lendo Metadados

Para ler os metadados de um arquivo, simplesmente execute o ExifTool seguido pelo nome do arquivo:

```
exiftool <nome_do_arquivo>
```

Esse comando exibirá todos os metadados disponíveis no arquivo especificado.

3.2.2 - Alterando Metadados

O ExifTool permite modificar metadados de forma simples. Por exemplo, para alterar o título de um arquivo PDF, você usaria:

```
exiftool -Title="Novo Título" nome_do_arquivo.pdf
```

Lembre-se de que alterar metadados pode modificar a integridade do arquivo, então use com cuidado.

3.2.3 - Extraindo Metadados para um Arquivo

Para salvar os metadados extraídos em um arquivo separado, você pode redirecionar a saída do comando:

```
exiftool <nome_do_arquivo> > metadados.txt
```

Isso criará um arquivo chamado metadados.txt com todos os metadados extraídos do arquivo especificado.

3.3 - Principais Parâmetros

-r: Executa o ExifTool de maneira recursiva, processando todos os arquivos em um diretório e seus subdiretórios.
-tagsFromFile <arquivo>: Copia metadados de um arquivo para outro.
-all=: Remove todos os metadados de um arquivo.
-GPSLatitude, -GPSLongitude: Permite ajustar as informações de geolocalização em arquivos de imagem.

4 - FLOSS (FireEye Labs Obfuscated String Solver)

4.1 - Introdução

FLOSS, ou FireEye Labs Obfuscated String Solver, é uma ferramenta avançada de código aberto projetada para detectar e desobfuscar/decodificar strings em binários. Ela é especialmente útil na análise de malware, pois muitas variantes de malware contêm strings ofuscadas para ocultar URLs maliciosas, comandos, nomes de arquivo e outras informações críticas que podem ser fundamentais para entender o comportamento do malware. FLOSS utiliza técnicas avançadas de análise estática e dinâmica para identificar automaticamente e desobfuscar strings, facilitando significativamente o processo de análise de malware.

O FLOSS está disponível e pré-instalado no REMnux. Se por algum motivo você precisar instalar o FLOSS em um sistema que não seja o REMnux, ou precisar de uma versão mais atualizada, você pode baixá-lo e instalá-lo a partir do repositório oficial no GitHub:

```
git clone https://github.com/fireeye/flare-floss.git
cd flare-floss
python setup.py install
```

4.2 - Uso Básico

O FLOSS é uma ferramenta poderosa para revelar strings ofuscadas em binários. Aqui está como você pode começar a usar o FLOSS para análise:

4.2.1 - Analisando um Binário

Para analisar um arquivo binário e extrair strings ofuscadas, simplesmente execute o FLOSS com o caminho do arquivo como argumento:

```
floss <nome_do_binario.exe>
```

Esse comando irá processar o binário, identificar e tentar desobfuscar as strings contidas nele.

4.2.2 - Exportação de Resultados

Você pode redirecionar a saída do FLOSS para um arquivo para análise posterior:

floss nome_do_binario.exe > output.txt

4.3 - Parâmetros

-q, --quiet: Executa o FLOSS em modo silencioso, exibindo apenas as strings desobfuscas.
-a, --all-strings: Extrai todas as strings, incluindo as que não estão ofuscadas.
-g, --group: Agrupa strings desobfuscas por algoritmo de ofuscação.
—help: Para mais informações sobre parâmetros.

5 - Detect It Easy (DiE)

5.1 - Introdução

Detect It Easy, ou DiE, é uma ferramenta de análise de binários que visa fornecer uma visão detalhada sobre a natureza de qualquer arquivo binário. É capaz de identificar muitos aspectos de arquivos, como o tipo de arquivo, se ele foi compactado ou empacotado (e com qual empacotador), a presença de criptografia, e muito mais. DiE é especialmente útil para analistas de malware, pesquisadores de segurança e qualquer pessoa interessada em engenharia reversa, fornecendo informações cruciais para a compreensão de como um arquivo desconhecido pode se comportar.

Detect It Easy vem pré-instalado no REMnux por padrão. Se por algum motivo você precisar instalar o FLOSS em um sistema que não seja o REMnux, ou precisar de uma versão mais atualizada, você pode baixá-lo e instalá-lo a partir do repositório oficial no GitHub:

git clone https://github.com/horsicq/Detect-It-Easy.git

5.2 - Analisando um Arquivo (GUI)

Para analisar um arquivo com o DiE, basta iniciar o programa e abrir o arquivo desejado através da interface gráfica do usuário. DiE fornecerá uma análise detalhada sobre o arquivo, incluindo seu tipo, se ele foi empacotado e com qual empacotador, indicações de criptografia, entre outras informações.

5.3 - Analisando um Arquivo via CLI

DiE também oferece uma interface de linha de comando (CLI) para automação ou integração em workflows de análise mais amplos. Para usar a CLI:

diec <nome_do_arquivo>

Este comando básico analisará o arquivo especificado e exibirá as informações detectadas no terminal.

5.3.1 - Parâmetros

-r : scan recursivo.
-d : scan profundo.
-u : scan heurístico.
-a : scanneia todos os tipos.
—json: exporta o arquivo em json.
—help : para ver mais parâmetros.

6 - Ghidra

6.1 - Introdução

Ghidra é uma suíte de software de engenharia reversa (SRE) de código aberto desenvolvida pela National Security Agency (NSA) dos Estados Unidos. Ela é projetada para auxiliar analistas de segurança na análise de programas maliciosos e software em geral. Com Ghidra, é possível descompilar binários para uma forma de código mais legível, entender a lógica de software desconhecido, identificar potenciais vulnerabilidades e muito mais. Ghidra suporta uma ampla variedade de formatos de arquivo em diversas plataformas, incluindo Windows, macOS, Linux e dispositivos móveis.

O Ghidra vem pré-instalado no REMnux. Se por algum motivo você precisar instalar o FLOSS em um sistema que não seja o REMnux, ou precisar de uma versão mais atualizada, você pode baixá-lo e instalá-lo seguindo os seguintes passos:

Acesse o site oficial do Ghidra (https://ghidra-sre.org/) e baixe a versão mais recente.
Extraia o arquivo baixado em um diretório de sua escolha.
Ghidra requer o JDK (Java Development Kit) para ser executado. Certifique-se de que você tem uma versão compatível do JDK instalada no seu sistema.

6.2 - Uso Básico

6.2.1 - Iniciando um Projeto

Após iniciar o Ghidra, o primeiro passo é criar um novo projeto. Isso pode ser feito através da janela inicial, selecionando "File" > "New Project". Você pode escolher entre um projeto não compartilhado (para uso local) ou compartilhado (para colaboração).

6.2.2 - Importando Binários

Para analisar um arquivo binário, você primeiro precisa importá-lo para o seu projeto no Ghidra. Isso pode ser feito usando "File" > "Import File" e navegando até o arquivo desejado.

6.2.3 - Analisando o Binário

Depois de importar o arquivo, clique duas vezes sobre ele na janela "Project" para abri-lo. Ghidra perguntará se você deseja analisar o arquivo agora. Confirme para iniciar a análise automática, que preparará o binário para a descompilação e exame detalhado.

6.2.4 - Usando o Decompiler

Com o binário aberto, você pode visualizar o código descompilado selecionando uma função na janela "Functions" e visualizando o código fonte aproximado na janela "Decompiler". Isso é especialmente útil para entender o que o binário faz sem necessidade de analisar o assembly.

6.3 - Principais Funcionalidades

Descompilação de código: Ghidra oferece uma descompilação poderosa, transformando código de máquina em uma representação de alto nível.
Análise de código: Ferramentas integradas permitem a análise de fluxo de controle, estruturas de dados e mais.
Scripts e Extensões: Ghidra suporta scripts em Python e Java, permitindo a automação de tarefas repetitivas e a extensão das funcionalidades do Ghidra.

7 - Yara

7.1 - Introdução

Yara é uma ferramenta de grande importância para analistas de segurança e pesquisadores de malware, utilizada para identificar e classificar malware com base em regras definidas pelo usuário. Ela permite a criação de descrições (regras Yara) que procuram por padrões específicos em arquivos ou fluxos de dados. Essas regras podem detectar sequências de bytes, strings e até padrões comportamentais dentro de amostras de malware, facilitando a identificação e a análise de famílias de malware e variantes.

O Yara já vem pré-instalado no REMnux. Caso você não esteja utilizando o REMnux ou precise de uma versão mais recente do Yara, você pode instalá-lo seguindo as instruções no GitHub oficial do Yara:

```
sudo apt-get install yara
```

Ou, para compilar a partir do código-fonte:

```
git clone https://github.com/VirusTotal/yara.git
cd yara
./bootstrap.sh
./configure
```

7.2 - Uso Básico

7.2.1 - Regras

Para usar o Yara, você primeiro precisa criar uma regra. Segue um exemplo de uma regra básica:

```
rule ExampleMalwareDetection
{
   strings:
      $a = "maliciousstring" nocase
      $b = {6A 40 68 00 30 00 00 6A 14 8D 91}

   condition:
      $a or $b
}
```

Salve isso em um arquivo com a extensão .yara, por exemplo, example_rule.yara.

7.2.1 - Analisando um Arquivo com Yara

Para analisar um arquivo utilizando a regra que você criou, execute o seguinte comando no terminal:

```
yara example_rule.yara name_of_file_to_scan
```

Isso irá verificar se o arquivo name_of_file_to_scan corresponde a qualquer uma das condições definidas em example_rule.yara.

7.3 - Parâmetros

-m: Mostra os metadados das regras correspondentes.
-s: Mostra as strings correspondentes nas regras.
-r: analise recursiva de diretórios
-f: Interrompe após a primeira correspondência.

7.4 - Alguns exemplos de regras

Vamos apresentar a seguir uma série de exemplos de regras YARA. Vale ressaltar que estes exemplos são meramente ilustrativos, servindo como ponto de partida para o entendimento das técnicas de detecção. A efetividade dessas regras em identificar ameaças específicas depende fortemente do contexto em que são aplicadas.

7.4.1 - Deteção de Executáveis Empacotados

```
rule PackedExecutable {
    meta:
        description = "Detecta executáveis empacotados comumente usados por malwares"
    strings:
        $upx = "UPX!"
        $mpress = "MPRESS"
    condition:
        $upx or $mpress
}
```

7.4.2 - Identificação de Shellcode

```
rule ShellcodeDetection {
    meta:
        description = "Detecta a presença de shellcode comum"
    strings:
        $shellcode = { 6A 30 59 FE CD FE CD FE CD 8B C1 51 52 }
    condition:
        $shellcode
}
```

7.4.3 - Detecção de URLs HTTP

```
rule HTTPUrls {
    meta:
        description = "Detecta strings de URLs HTTP que podem ser usadas para C&C"
    strings:
        $http = "http://" nocase
    condition:
        $http
}
```

7.4.4 - Detecção de Malware Baseado em PowerShell

```
rule PowerShellMalwareEnhanced {
    meta:
        description = "Detecta scripts PowerShell maliciosos com base em parâmetros e técnicas comuns de ofuscação"
    strings:
        $powershell = "powershell.exe" nocase
        $exec_bypass = "-Exec Bypass" nocase
        $encoded_command = "-EncodedCommand" nocase
        $base64_pattern = /[A-Za-z0-9+\/]{20,}={0,2}/ nocase
    condition:
        $powershell and ($exec_bypass or $encoded_command or $base64_pattern)
}
```

7.4.5 - Detecção de Uso de Criptografia

```
rule EncryptionRoutinesEnhanced {
    meta:
        description = "Detecta o uso de rotinas de criptografia comuns, focando em contextos suspeitos"
    strings:
        $aes = "AES" nocase
        $des = "DES" nocase
        $cryptolib = "CryptEncrypt" nocase
    condition:
        ($aes or $des) and $cryptolib
}
```

7.4.6 - Identificação de Documentos Maliciosos do Office

```
rule MaliciousOfficeDocumentEnhanced {
    meta:
        description = "Detecta documentos do Office com macros potencialmente maliciosas,
usando estruturas de macros suspeitas"
    strings:
        $ole = {D0 CF 11 E0 A1 B1 1A E1}
        $autoopen = "AutoOpen" nocase
        $autoexec = "AutoExec" nocase
        $vba_macro = /CreateObject\("WScript.Shell"\)/ nocase
    condition:
        $ole at 0 and ($autoopen or $autoexec or $vba_macro)
}
```

7.4.7 - Detecção de Anti-Debugging

```
rule AntiDebuggingTechniquesEnhanced {
    meta:
        description = "Detecta técnicas comuns de anti-debugging além de IsDebuggerPresent"
    strings:
        $isDebuggerPresent = "IsDebuggerPresent" nocase
        $checkRemoteDebugger = "CheckRemoteDebuggerPresent" nocase
        $outputDebugString = "OutputDebugString" nocase
        $debugSelf = {E8 00 00 00 00 C3} // Padrão que tenta chamar a si mesmo, técnica comum
para detectar debuggers.
    condition:
        any of them
}
```

7.4.8 - Detecção de Código Ofuscado

```
rule ObfuscatedCodeEnhanced {
    meta:
        description = "Detecta padrões de ofuscação de código, incluindo técnicas além do
base64_decode"
    strings:
        $base64_decode = "base64_decode" nocase
        $eval = "eval" nocase // Muito usado em PHP para executar código ofuscado.
```

```
        $exec = "exec" nocase // Pode ser usado para execução de código ofuscado
dinamicamente.
        $gzinflate = "gzinflate" nocase // Usado para descomprimir strings codificadas.
    condition:
        any of them
}
```

7.4.9 - Identificação de Uso de Comandos do Sistema

```
rule SystemCommandsUsageEnhanced {
    meta:
        description = "Detecta o uso de comandos do sistema em arquivos, com foco em variações
comuns"
    strings:
        $cmd = "cmd.exe" nocase
        $powershell = "powershell.exe" nocase // Inclusão do PowerShell devido ao seu uso comum
em ataques.
        $wscript = "wscript.exe" nocase // Executor de scripts do Windows, frequentemente usado
em malware.
    condition:
        any of them
}
```

7.4.10 - Detecção de Acesso a Registro do Windows

```
rule WindowsRegistryAccessEnhanced {
    meta:
        description = "Detecta acesso a chaves de registro específicas do Windows, considerando
chaves comumente abusadas"
    strings:
        $regCurrentUser = "HKEY_CURRENT_USER" nocase
        $regLocalMachine = "HKEY_LOCAL_MACHINE" nocase // Inclusão devido ao seu potencial
de impacto no sistema.
        $regClassesRoot = "HKEY_CLASSES_ROOT" nocase // Frequentemente modificado por
malware para persistência ou hijacking.
        $regUsers = "HKEY_USERS" nocase
    condition:
        any of them
}
```

Capítulo 3 - Ferramentas de Análise Dinâmica

A análise dinâmica de malware é uma técnica que envolve executar o malware em um ambiente controlado ou sandbox para observar seu comportamento em tempo real. Diferente da análise estática, que examina o código do malware sem executá-lo, a análise dinâmica permite aos pesquisadores observarem como o malware interage com o sistema operacional, redes, outros programas e usuários. Isso inclui monitorar chamadas de sistema, modificações de arquivos e registros, comunicações de rede, e outros comportamentos maliciosos. A análise dinâmica é essencial para entender ameaças complexas que podem não ser totalmente reveladas pela análise estática, ajudando na criação de medidas de detecção e remediação mais eficazes.

1 - Wireshark

1.1 - Introdução

Wireshark é a ferramenta de análise de rede mais reconhecida e amplamente utilizada no mundo. Funciona como um analisador de protocolos de rede, ou "sniffer", capturando e exibindo em tempo real o tráfego de dados que passa por redes de computadores em formato detalhado e decodificado. Com o Wireshark, profissionais de TI, engenheiros de rede e analistas de segurança podem examinar detalhadamente o que está acontecendo em uma rede, permitindo a detecção de problemas, análises de segurança, desenvolvimento e aprendizado de protocolos de rede.

O REMnux inclui o Wireshark por padrão. Caso você não esteja utilizando o REMnux ou precise de uma versão mais recente do wireshark, você pode instalá-lo seguindo as instruções:

```
sudo apt-get install wireshark
```

Durante a instalação, pode ser solicitado que você configure as permissões para usuários não privilegiados capturarem pacotes. Siga as instruções na tela para fazer sua escolha.

1.2 - Uso Básico

1.2.1 - Iniciando o Wireshark

Após a instalação, você pode iniciar o Wireshark através do menu de aplicações do sistema ou pela linha de comando simplesmente digitando wireshark.

1.2.2 - Capturando Pacotes

Selecionar a Interface de Rede: Ao abrir o Wireshark, você verá uma lista de interfaces de rede disponíveis para captura. Escolha a interface pela qual o tráfego de rede relevante está passando.

1.2.3 - Iniciar Captura

Clique em "Start" ao lado da interface escolhida para começar a capturar os pacotes.

1.2.4 - Filtrando Tráfego

Wireshark oferece poderosos filtros de exibição para ajudá-lo a isolar o tráfego de interesse. Por exemplo, para exibir apenas o tráfego HTTP:

```
http
```

Para filtrar o tráfego por um endereço IP específico:

```
ip.addr == 192.168.1.1
```

1.2.5 - Analisando Pacotes

Cada pacote capturado é listado na janela principal. Clicar em um pacote expandirá os detalhes na janela do meio, divididos por camada de protocolo. A janela inferior mostra os dados do pacote em formato hexadecimal e ASCII.

1.2.6 - Salvando Capturas

Para salvar uma sessão de captura para análise posterior ou compartilhamento, vá para "File" > "Save As" e escolha um local e nome para o arquivo.

1.3 - Alguns Exemplos de Filtros Úteis

Filtrar por IP:	ip.addr==<ip>
Filtrar subnet:	ip.addr==255.255.255.0/24
Filtrar portas TCP:	tcp.port==<n> tcp.port!=<n> tcp.port==80 or tcp.port == 443 tcp.port in {80,443,8000,8080}
Comunicação entre IPs:	ip.addr==<ip> && tcp.port == <n> && ip.addr==<ip> && tcp.port == <n>
Filtrar por protocolo:	<protocolo>
Buscar por strings:	<protocolo> contains “<string>" <protocolo> matches “<string>" (regex)
Filtrar flags:	tcp.analysis.flags

Identificar tempo de resposta: dns.time > <t>

Geolocalização:	ip.geoip.country_iso == “<code_country>”

1.4 - Mais Informações

Top 10 Real World Wireshark Filters you need to know (https://www.youtube.com/watch?v=26MAaX2ldnI)

Udemy: Getting Started with Wireshark: The Ultimate Hands-On Course - Chris Greer

2 - Monitoramento de Processos

Existem várias ferramentas disponíveis para monitoramento de processos em execução e consumo de recursos. Não iremos focar em nenhuma ferramenta especifica neste caso, mas é importante ressaltar a importância da análise de processos e recursos em uma análise dinâmica de malware. Malwares que utilizam recursos da máquina infectada para minerar criptomoedas, por exemplo, utilizam muitos recursos da máquina e podem ser facilmente detectados utilizando o monitoramento de recursos.

3 - GDB

3.1 - Introdução

O GNU Debugger (GDB) é a ferramenta de depuração padrão do ecossistema GNU. É uma ferramenta de linha de comando extremamente poderosa que permite a programadores analisar o que acontece "por dentro" de um programa enquanto ele executa — ou o que o programa estava fazendo no momento em que ele falhou. O GDB pode ser usado para depurar programas escritos em C, C++, Rust, Go, entre outras linguagens de programação suportadas pelo GCC (GNU Compiler Collection). Ele é essencial para identificar falhas de segurança, vazamentos de memória, violações de acesso e outros problemas.

O GDB vem pré-instalado no REMnux. Caso o GDB não esteja disponível ou você deseje instalar o GDB em outra distro, siga os seguintes passos:

```
sudo apt-get install gdb
```

3.2. Uso Básico e Funcionalidades

3.2.1 - Iniciando o GDB

Para começar a depurar um programa com o GDB, primeiro certifique-se de que o programa foi compilado com informações de depuração (usualmente, adicionando a flag -g ao comando de compilação). Em seguida, inicie o GDB passando o nome do programa como argumento:

```
gdb ./nome_do_programa
```

3.2.2 - Definindo Breakpoints

Um "breakpoint" é um ponto no programa onde a execução será interrompida, permitindo que você examine o estado do programa. Para definir um breakpoint em uma função:

```
(gdb) break nome_da_funcao
```

Para definir um breakpoint em uma linha específica do arquivo:

```
(gdb) break nome_do_arquivo.c:numero_da_linha
```

3.2.3 - Iniciando a Execução

Para iniciar a execução do programa sob o GDB, use:

```
(gdb) run
```

Se o programa necessitar de argumentos, eles podem ser passados após o comando run.

3.2.4 - Examinando Variáveis

Para examinar o valor de uma variável após a execução ser interrompida em um breakpoint:

```
(gdb) print nome_da_variavel
```

3.2.5 - Continuando a Execução

Para continuar a execução até o próximo breakpoint ou até a conclusão do programa:

```
(gdb) continue
```

3.2.6 - Saindo do GDB

Para sair do GDB, use:

```
(gdb) quit
```

3.3 - Principais Parâmetros

Step Into (s): Executa a próxima linha de código, entrando em funções.

Next (n): Executa a próxima linha de código, mas não entra em funções.

Backtrace (bt): Mostra a pilha de chamadas atual.

Watchpoints: Permite observar o valor de uma variável e interromper a execução sempre que o valor muda.

3.4 - Exemplo Prático

Suponha que você quer depurar um programa chamado example que está acessando um ponteiro nulo e causando uma falha de segmentação:

Compile example.c com a flag -g: gcc -g example.c -o example.
Inicie o GDB: gdb ./example.
Defina um breakpoint na função suspeita: (gdb) break suspeita_funcao.
Execute o programa: (gdb) run.
Quando o breakpoint for atingido, use next ou step para avançar linha por linha.
Examine variáveis com print para entender o estado que levou ao erro.

4- Radare2

4.1 - Introdução

Radare2 é uma poderosa suíte de ferramentas de engenharia reversa de código aberto. Ela oferece funcionalidades para desmontagem (disassembly), depuração (debugging), análise de binários, manipulação de arquivos e muito mais. Projetado para oferecer uma experiência abrangente na análise de software, desde simples scripts até programas complexos, Radare2 suporta uma ampla gama de arquiteturas e formatos de arquivo, tornando-o uma ferramenta indispensável para pesquisadores de segurança, analistas de malware e entusiastas de engenharia reversa.

Radare2 geralmente vem pré-instalado no REMnux devido à sua relevância e utilidade na análise de malware. No entanto, dada a rápida evolução do Radare2, pode ser benéfico atualizar para a versão mais recente diretamente do repositório oficial. Para instalar ou atualizar o Radare2 no REMnux ou em qualquer distribuição Linux baseada em Debian, você pode utilizar o seguinte comando:

```
sudo apt-get install radare2
```

Ou para a versão mais atualizada:

```
git clone https://github.com/radareorg/radare2.git
cd radare2
sys/install.sh
```

4.2. Uso Básico e Funcionalidades

4.2.1 - Iniciando com Radare2

Para começar a análise de um binário com Radare2, use o comando:

```
radare2 nome_do_binario
```

Isso abrirá o binário no modo de análise do Radare2, pronto para ser examinado.

4.2.2 - Análise Básica

Após abrir um binário, você pode começar uma análise automática com o comando:

```
aaa
```

Este comando realiza uma análise profunda do binário, identificando funções, strings, referências cruzadas e muito mais.

4.2.3 - Exibindo Funções

Para listar todas as funções identificadas na análise, use:

```
afl
```

4.2.4 - Desmontagem de Funções

Para desmontar (disassemblar) uma função específica e ver seu código, primeiro mude o foco para a função com s nome_da_funcao e depois desmonte com pdf:

```
s main
pdf
```

4.2.5 - Depurando um Programa

Para iniciar o Radare2 em modo de depuração, use o comando:

```
radare2 -d nome_do_binario
```

Você pode então definir breakpoints, executar o programa e inspecionar o estado do programa conforme necessário.

4.2.6 - Manipulação de Memória e Registros

Para ler e escrever em memória e registros, você pode usar os comandos dr para registros e px para exibir a memória.

4.2.7 - Exportação de Análises

Radare2 suporta a exportação de análises em vários formatos, incluindo JSON, com comandos específicos baseados no que você deseja exportar.

4.3 - Exemplo Prático

Analisando um binário e buscando por strings potencialmente interessantes:

Abra o binário no Radare2: radare2 nome_do_binario
Realize uma análise automática: aaa
Busque por strings: iz
Foque em uma função específica: s main
Desmonte a função: pdf

5 - INetSim

5.1 - Introdução

INetSim é uma ferramenta de software para simulação de serviços de rede. Ela é projetada para oferecer um ambiente de rede controlado em que um cliente de rede pode interagir como se estivesse na internet, mas sem os riscos associados à exposição real. O INetSim permite que analistas de segurança e pesquisadores de malware observem o comportamento do malware em um ambiente seguro, analisando tentativas de conexão com servidores C2 (comando e controle), downloads de payload adicional, entre outras atividades maliciosas. A ferramenta pode simular diversos protocolos e serviços de rede, incluindo HTTP, HTTPS, SMTP, DNS, e muitos outros.

O INetSim é uma das muitas ferramentas pré-instaladas no REMnux, dada sua utilidade fundamental na análise de malware e na criação de ambientes de rede simulados para fins de pesquisa.
Se por algum motivo o INetSim não estiver instalado no REMnux ou você desejar instalá-lo em outra distribuição Linux, o processo é simples. No Ubuntu ou em distros baseadas no Debian, use o comando:

```
sudo apt-get install inetsim
```

5.2 - Uso Básico e Funcionalidades

5.2.1 - Configurando o INetSim

Antes de iniciar o INetSim, você pode querer personalizar a configuração para adequá-la às suas necessidades de análise. A configuração é feita no arquivo /etc/inetsim/inetsim.conf. Aqui, você pode habilitar ou desabilitar serviços específicos, configurar endereços IP de escuta, entre outras opções.

5.2.2 - Iniciando o INetSim

Para iniciar o INetSim com a configuração padrão, simplesmente execute:

```
sudo inetsim
```

Para um ambiente de teste, pode ser necessário executar o INetSim com privilégios root para que ele possa escutar nas portas padrão dos serviços simulados.

5.2.3 - Redirecionando o Tráfego para o INetSim

Para analisar o comportamento do malware, é comum redirecionar todo o tráfego de saída do sistema ou da máquina virtual para o INetSim. Isso pode ser feito configurando as regras de NAT (Network Address Translation) no firewall do sistema. Por exemplo, no iptables:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 80
```

Este comando redireciona todo o tráfego destinado à porta 80 (HTTP) para a porta 80 local, onde o INetSim estará escutando.

5.3 - Principais Funcionalidades

Simulação de Múltiplos Protocolos de Rede: O INetSim suporta a simulação de uma ampla gama de serviços de rede, permitindo uma análise abrangente do comportamento de rede do malware.
Logs Detalhados: O INetSim gera logs detalhados de todas as atividades de rede simuladas, facilitando a análise posterior das ações do malware.

Configuração Flexível: Os usuários podem habilitar ou desabilitar serviços específicos e configurar detalhes da simulação através do arquivo de configuração.

6 - FakeNet-NG

6.1 - Introdução

FakeNet-NG é uma ferramenta avançada de simulação de rede que intercepta e redireciona todo o tráfego de saída do sistema para si mesma, simulando respostas de protocolos de rede. Isso permite que analistas de segurança e pesquisadores de malware observem e analisem o comportamento de comunicação de rede de softwares maliciosos em um ambiente controlado.

O FakeNet-NG é uma ferramenta popular entre os analistas de malware e, embora não venha pré-instalado no REMnux, pode ser facilmente adicionado devido à sua utilidade na análise de malware.

FakeNet-NG é primariamente desenvolvido para o Windows, mas pode ser executado em Linux através do Wine. Para a versão Windows, você pode baixá-lo diretamente do GitHub oficial do FakeNet-NG.
Para usar no Linux, incluindo o REMnux, primeiro garanta que o Wine está instalado:

```
sudo apt-get install wine
```

Em seguida, baixe o FakeNet-NG e execute-o através do Wine.

6.2 - Configuração e Inicialização

6.2.1 - Iniciando o FakeNet-NG

No Linux, use o Wine para executar o FakeNet-NG:

wine fakenet-ng.exe

6.2.2 - Configuração Básica

O FakeNet-NG vem com um arquivo de configuração fakenet.ini, que permite aos usuários personalizar como o FakeNet-NG responde ao tráfego interceptado. Você pode modificar este arquivo para simular diferentes ambientes de rede e configurar respostas específicas de protocolo.

6.3 - Funcionalidades Chave

Interceptação de Tráfego de Rede: O FakeNet-NG pode interceptar tráfego de rede HTTP, HTTPS, DNS, e muitos outros protocolos, simulando a internet para o software em análise.

Logs Detalhados: Fornece logs detalhados do tráfego de rede interceptado, permitindo uma análise aprofundada das comunicações de rede do malware.

DNS Dinâmico: Pode simular respostas DNS para qualquer domínio solicitado pelo malware, direcionando todas as requisições de volta para o FakeNet-NG.

6.4 - Exemplo Prático

1 - Para analisar um malware que tenta se conectar a um domínio específico:
Configure o fakenet.ini para simular o domínio desejado, garantindo que o FakeNet-NG responderá às solicitações para esse domínio.

2 - Inicie o FakeNet-NG antes de executar o malware.

3 - Execute o malware em análise. O FakeNet-NG interceptará qualquer tentativa de comunicação de rede, permitindo que você observe as solicitações feitas.

4 - Analisando a Saída:
Verifique os logs do FakeNet-NG para entender as solicitações de rede feitas pelo malware, incluindo quaisquer tentativas de exfiltração de dados, download de payloads adicionais ou comunicações com servidores C2.

7 - Volatility para Análise de Malware

7.1 - Introdução

Volatility é uma ferramenta de análise forense de memória de código aberto, projetada para extrair informações de dumps de memória (memory dumps) de sistemas Windows, Linux, Mac, e Android. Na análise de malware, o Volatility é fundamental para investigar malwares que residem na memória, auxiliando na identificação de processos suspeitos, conexões de rede, DLLs injetadas e muito mais, sem a necessidade do sistema operacional estar em execução. Isso torna o Volatility uma ferramenta poderosa para desvendar os aspectos mais obscuros dos malwares que se escondem na memória do sistema.

O Volatility geralmente vem pré-instalado no REMnux, dada a sua importância na análise de malware e investigações forenses digitais. Caso o Volatility não esteja instalado ou você deseje uma versão atualizada no REMnux ou em outro sistema, você pode instalar o Volatility seguindo as instruções no GitHub oficial do projeto. Para a maioria das distribuições Linux, o processo pode ser tão simples quanto:

```
sudo apt-get install volatility
```

7.2 - Uso Básico e Funcionalidades

7.2.1 - Identificando a Versão do Sistema Operacional

Antes de começar a análise, é crucial identificar a versão do sistema operacional do dump de memória para utilizar os plugins apropriados. Use o comando:

```
volatility -f memory_dump.img imageinfo
```

7.2.2 - Listando Processos em Execução

Para listar os processos em execução no momento do dump de memória, o que é útil para identificar atividades suspeitas ou malwares escondidos:

```
volatility -f memory_dump.img --profile=Win7SP1x64 pslist
```

7.2.3 - Analisando Conexões de Rede

Para investigar as conexões de rede ativas ou em escuta, o que pode revelar tentativas de comunicação com servidores de comando e controle:

```
volatility -f memory_dump.img --profile=Win7SP1x64 netscan
```

7.2.4 - Extraindo Strings de um Processo

Para extrair strings de um processo específico, o que pode ajudar na identificação de C2s, chaves de criptografia, e outros indicadores:

```
volatility -f memory_dump.img --profile=Win7SP1x64 memdump --pid=1234 -D output_directory/
```

7.2.5 - Procurando por DLLs Injetadas

Para encontrar DLLs injetadas, que é uma técnica comum usada por malwares para esconder sua presença:

```
volatility -f memory_dump.img --profile=Win7SP1x64 malfind
```

7.2.6 - Principais Parâmetros

-f / --file: Especifica o arquivo de dump de memória a ser analisado.
--profile: Indica o perfil do sistema operacional do dump de memória. Este parâmetro é crucial para a análise correta.
pslist, psscan, pstree: Comandos para listar processos de diferentes maneiras.
netscan: Lista conexões de rede ativas e em escuta.
memdump: Extrai a memória de um processo específico.
malfind: Localiza e analisa processos e/ou regiões de memória suspeitos de abrigar malwares.

7.3 - Exemplo Prático

Suponha que você está investigando um dump de memória de um sistema Windows 7 SP1 x64 suspeito de infecção por malware. Após identificar o perfil correto com imageinfo, você decide listar os processos em execução usando pslist, analisar as conexões de rede com netscan, e verificar a presença de DLLs injetadas com malfind.