UNIVERSIDADE DE AVEIRO

CTeSP – Desenvolvimento de Software

Trabalho 6
Software Seguro e Deteção de Vulnerabilidades

Redes e Segurança Informática

Apresentado por:

Douglas Lobo
Número mecanográfico: 127111
Email: douglascnlobo@ua.pt

Orientado por:

Professor: Mário Pinto
Email: mjp@ua.pt

Aveiro, Portugal
20 de junho de 2025

Sumário

Lista de Figuras

PARTE I – Distribuição de *Software*

Obtenha o ficheiro ISO da distribuição Linux Mint

Acedendo ao *website* do Linux Mint e dirigindo-me à aba de *downloads* selecionei o "sabor" Xfce descrito como *light, simple, efficient.*

Verifique a forma de verificação de integridade e autenticidade proposta pela comunidade da distribuição Linux Mint

"Anyone can produce fake ISO images, it is your responsibility to check you are downloading the official ones." avisa-nos a direção do Linux Mint. Assim procedi ao manual fornecido e pus-me a ler e executar as verificações de integridade e autenticidade lá dispostas.[1]

Proceda à verificação de integridade e autenticidade dos ficheiros obtidos

Efetuado o *download* da ISO e das *hashes*, parti para o terminal onde apliquei os passos elencados na documentação.

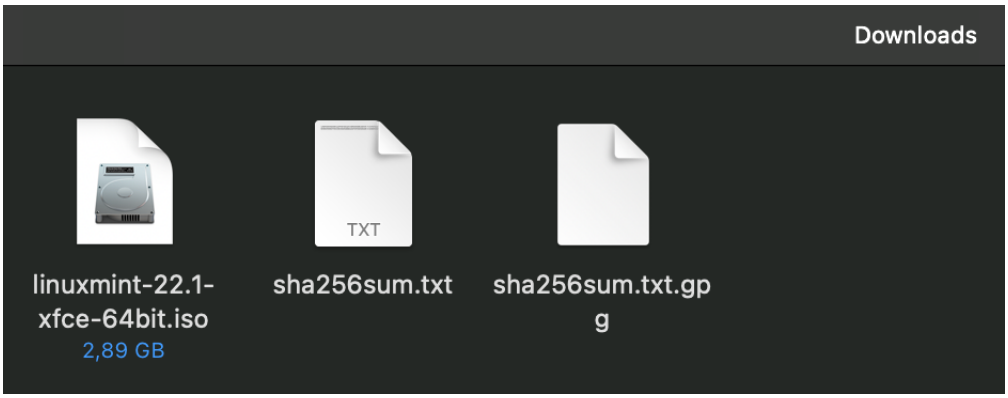

Figura 1.1: Arquivos baixados do *website* oficial do Linux Mint

Primeiro verifiquei a integridade das *hashes* com os seguintes comandos:

[1] `https://linuxmint-installation-guide.readthedocs.io/en/latest/verify.html`

```
[douglaslobo@m0 ~/Downloads]$ sha256sum -b linuxmint-22.1-xfce-64bit.iso
6451496af35e6855ffe1454f061993ea9cb884d2b4bc8bf17e7d5925ae2ae86d *linuxmint-22.1-xf
ce-64bit.iso
[douglaslobo@m0 ~/Downloads]$ sha256sum -c --ignore-missing sha256sum.txt
linuxmint-22.1-xfce-64bit.iso: OK
[douglaslobo@m0 ~/Downloads]$
```

Figura 1.2: Verificando a *hash* com `sha256sum`

```
[douglaslobo@m0 ~/Downloads]$ shasum -a 256 linuxmint-22.1-xfce-64bit.iso
6451496af35e6855ffe1454f061993ea9cb884d2b4bc8bf17e7d5925ae2ae86d  linuxmint-22.1-xf
ce-64bit.iso
[douglaslobo@m0 ~/Downloads]$ shasum -c --ignore-missing sha256sum.txt
linuxmint-22.1-xfce-64bit.iso: OK
```

Figura 1.3: Verificando a *hash* com `shasum`

Segundo, verifiquei-lhe a autenticidade com os seguinte comandos:

```
[douglaslobo@m0 ~/Downloads]$ gpg --keyserver hkp://keys.openpgp.org:80 --recv-key
27DEB15644C6B3CF3BD7D291300F846BA25BAE09
gpg: key 300F846BA25BAE09: public key "Linux Mint ISO Signing Key <root@linuxmint.c
om>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Figura 1.4: Adquirindo a chave pública do Linux Mint

```
[douglaslobo@m0 ~/Downloads]$ gpg --verify sha256sum.txt.gpg sha256sum.txt

gpg: Signature made Tue Jan 14 11:35:18 2025 WET
gpg:                using RSA key 27DEB15644C6B3CF3BD7D291300F846BA25BAE09
gpg: Good signature from "Linux Mint ISO Signing Key <root@linuxmint.com>" [unknown
]
```

Figura 1.5: Verificando autenticidade: `good signature` e a *fingerprint* condiz com a provida no *website*

Após este processo consigo assegurar dentro dos limites dos meus conhecimentos e presumindo que o *website* do Linux Mint não foi adulterado em segredo por um agente malicioso que, sim, se trata de uma ISO oficial inalterada.

PARTE II – *Software* Seguro

Entrega segura de software instalável e suas atualizações

Concebamos um *software* aos moldes de um gestor de frota para uma entidade que oferece serviços de manutenção de refrigeradores que será utilizado pelas equipes de técnicos para reservarem veículos dinamicamente para os seus serviços. O sistema é distribuído como aplicação instalável nos dispositivos das equipas e deve suportar atualizações futuras, tanto funcionais como de segurança.

A aplicação centraliza a gestão das reservas, permitindo que os técnicos consultem a disponibilidade da frota e reservem veículos conforme a agenda de intervenções. Os dados são sincronizados com um servidor central, onde também são processadas as atualizações e os acessos administrativos. O ambiente de uso prevê mobilidade, ligação remota e operação contínua durante o horário de expediente.

Processos de segurança considerados

Autenticação e controle de acesso
Cada técnico tem credenciais próprias e só pode ver e reservar veículos. Supervisores e administradores têm permissões ampliadas, como aprovar ou rejeitar uma reserva e configurar o sistema de modo *ad hoc*. O sistema valida sessões com `tokens` e bloqueia acessos não autorizados. A autenticação baseia-se em sessões geridas por `JWT` (JSON Web Tokens), com verificação periódica do estado da sessão.

Validação de dados e uso legítimo
Nenhuma reserva entra no sistema sem passar por validações no servidor. Evita abusos, inconsistências e tentativas de entrada de dados maliciosos, como *SQL injections*[1] (atacando diretamente a base de dados, mitigado com consultas preparadas) e *XSS*[2] (execução de scripts maliciosos, mitigado com sanitização e codificação de entradas).

Comunicação segura
Toda a comunicação com o servidor é feita via `HTTPS`, utilizando o protocolo `TLS` (Transport Layer Security)[3]. Não há tráfego exposto. Dados sensíveis, como credenciais e reservas, são protegidos em trânsito, ou seja, no transporte pela rede.

Integridade e assinaturas
O instalador e o binário da aplicação são assinados digitalmente com certificados `X.509`[4]

[1] `https://owasp.org/www-community/attacks/SQL_Injection`
[2] `https://owasp.org/www-community/attacks/xss`
[3] `https://datatracker.ietf.org/doc/html/rfc8446`
[4] `https://datatracker.ietf.org/doc/html/rfc5280`

(padrão de infraestrutura de chave pública) emitidos por uma autoridade certificadora (*CA*) interna ou reconhecida. A aplicação cliente valida a assinatura antes da instalação ou atualização. Só é possível instalar versões autenticadas e autorizadas. Ao molde daquilo que o Linux Mint oferecia, será também possível que técnicos de sistema na entidade possam, como último, verificar manualmente a `hash` da aplicação (ex.: `SHA-256`, algoritmo que gera um identificador único de 256 bits). Isso garante que o *software* não foi adulterado e que provém de uma fonte confiável.

Assinaturas de ações e verificação automática
Para além da autenticação inicial, cada ação relevante executada na aplicação (como reservar, alterar ou cancelar uma reserva) é localmente assinada digitalmente utilizando uma chave secreta única por utilizador. Esta assinatura, gerada com `HMAC-SHA-256` (código de autenticação baseado em *hash*, usando a função `SHA-256` com uma chave secreta), assegura que apenas o legítimo titular da conta poderia ter originado a ação, e que os dados não foram manipulados no caminho. O servidor valida automaticamente cada requisição comparando o valor enviado com aquele que seria esperado a partir da chave conhecida. Tentativas de falsificação ou adulteração são detectadas e rejeitadas de imediato, permitindo resposta automatizada a incidentes.

Atualizações seguras
As atualizações são distribuídas em pacotes assinados digitalmente com a chave privada associada ao certificado `X.509`. O cliente valida a assinatura usando a chave pública incluída no certificado. O processo segue o modelo de segurança aplicado em sistemas como o `APT` (Advanced Package Tool, usado em distribuições como Debian/Ubuntu) e é automático, sem requerer intervenção do utilizador. Em caso de falha, existe mecanismo de *rollback* para garantir o funcionamento contínuo com a versão anterior validada.

Logs e rastreabilidade
Cada ação relevante (*sign in*, reserva, cancelamento, falhas) gera logs com data, hora, IP e utilizador. Os registos são protegidos contra edição (`WORM` – *write once, read many*)[5] e podem ser assinados localmente com o certificado `X.509`, usando algoritmos como `SHA-256 + RSA` (comprovação de integridade e autenticidade). Posteriormente, são enviados ao servidor por canal `TLS` para consolidação e auditoria.

Tolerância a falhas
Se o servidor estiver inacessível, a aplicação entra em modo *offline* com acesso restrito e operação local segura. As alterações são armazenadas em ficheiros temporários assinados digitalmente (com `HMAC` e `timestamp`) para posterior validação. Quando a ligação é restabelecida, os dados são sincronizados com o *backend*, mantendo a integridade no sistema.

[5] `https://en.wikipedia.org/wiki/Write_once_read_many`

PARTE III – Deteção de Vulnerabilidades

Explore a ferramenta Nmap

Acedendo ao *website* do Nmap (*Network Mapper*) aprende-se que se trata de um utilitário FOSS[1] para descoberta de rede e auditoria de segurança. O Nmap usa pacotes IP brutos de novas maneiras para determinar quais *hosts* estão disponíveis na rede, quais serviços (nome e versão do aplicativo) entre outros dados.

Utilize a ferramenta Nmap para obter informações do servidor scanme.nmap.org

Com a permissão do Nmap de escaneá-los utilizo o comando provido por eles:

```
[douglaslobo@m0 ~/Documents/CTeSP/1o ano/2o semestre/RSI/RSI trabalho 6]$ nmap -A s
canme.nmap.org
Starting Nmap 7.97 ( https://nmap.org ) at 2025-06-20 13:46 +0100
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.17s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 993 closed tcp ports (conn-refused)
PORT      STATE    SERVICE      VERSION
22/tcp    open     ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; p
rotocol 2.0)
| ssh-hostkey:
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (ED25519)
80/tcp    open     http         Apache httpd 2.4.7 ((Ubuntu))
|_http-server-header: Apache/2.4.7 (Ubuntu)
|_http-title: Go ahead and ScanMe!
|_http-favicon: Nmap Project
135/tcp   filtered msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
9929/tcp  open     nping-echo   Nping echo
31337/tcp open     tcpwrapped
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.or
g/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 48.19 seconds
```

Figura 1.1: Resultado do comando

[1] https://en.wikipedia.org/wiki/Free_and_open-source_software

Interpretando os resultados tenho que o servidor está ativo e respondendo pelo IP `45.33.32.156`, existe um IPv6, este, contudo, não foi escaneado. Foram escaneadas 1000 portas TCP, destas destaco:

- 993 destas estão fechadas (conn-refused);
- 7 portas mostradas: 5 abertas, 2 filtradas.

22/tcp ssh
O serviço SSH está aberto, rodando OpenSSH 6.6.1p1, versão típica de um sistema Ubuntu antigo. É possível inferir que esta máquina executa Ubuntu Linux, o que ajuda a descobrir o SO do alvo. Há também algumas chavees púbiclas úteis para verificar integridade da identidade do servidor.

80/tcp http
A página feita para fins educativos como suscitado no *header* com o texto "Go ahead and ScanMe!", em Apache httpd 2.4.7.

135, 139, 445/tcp serviços Windows
Esses serviços típicos de Windows possivelmente estão sendo utilizados para proteção ou redirecionamento, conquanto se trata de um sistema Ubuntu.

9929/tcp nping-echo
Usado como um simples utilitário de ping para detectar *hosts* ativos, também pode ser usado como um gerador de pacotes brutos para testes de estresse de pilha de rede, envenenamento de `ARP`, ataques de negação de serviço, rastreamento de rotas, etc. O novo modo de eco do Nping permite que os usuários vejam como os pacotes mudam em trânsito entre os *hosts* de origem e de destino.

31337/tcp tcpwrapped
O `tcpwrapped` significa que o serviço está protegido por uma camada genérica de TCP wrapperk, o Nmap não consegue identificar o que realmente roda lá. Também é uma porta famosa por comumente ser utilizada como *backdoor*.[2]

Utilize a ferramenta Nmap para obter informação de uma máquina na rede

Decidi então mapear o meu servidor pessoal e obtive os seguintes resultados.

[2] `https://en.wikipedia.org/wiki/Back_Orifice`

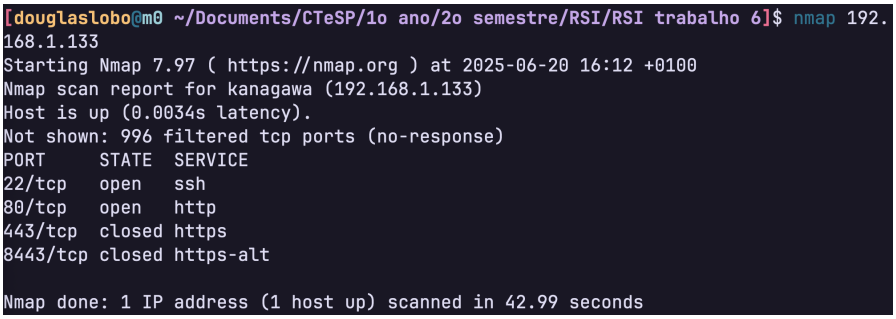

Figura 1.2: Resultado do escaneamento do *host* do meu servidor o kanagawa

Interpretando os resultados,[3] constato que a máquina alvo está ativa e a responder com o nome de rede *kanagawa*. O tempo de resposta foi bastante baixo (cerca de 3ms), indicando proximidade na rede local – como espearado. Foram escaneadas 1000 portas TCP, das quais destaco:

- 996 estão filtradas (sem resposta), o que indica presença de *firewall* ou filtragem por IP;
- 4 portas identificadas: 2 abertas, 2 fechadas.

Identifique possíveis fragilidades detectadas

22/tcp ssh
Porta aberta com o serviço `OpenSSH 9.6p1` em sistema Ubuntu recente. A autenticação é feita via protocolo `SSH 2.0` e as chaves públicas `ECDSA` e `ED25519` foram identificadas, o que permite validar posteriormente a identidade do servidor.

80/tcp http
Porta aberta com um servidor HTTP baseado em `Golang net/http`. A página de entrada tem como título *CasaOS*, uma interface de gestão de dispositivos e serviços domésticos.

443 e 8443/tcp https e https-alt
Ambas as portas encontram-se fechadas. Isso indica ausência de canal seguro para acesso ao `CasaOS`. O tráfego web está atualmente desprotegido, o que representa um risco à confidencialidade e integridade da informação transmitida na minha rede local.

Análise adicional
Apesar da resposta `HTTP` ter sido parcialmente reconhecida, o `Nmap` não conseguiu identificar totalmente o serviço *web*, tendo sugerido o envio da assinatura para futura identificação. Isto pode dever-se ao uso de tecnologias menos convencionais ou personalizadas.

[3] Aqui mostro os resultados do comando "`$ nmap 192.168.1.133`" sem a flag `-A`, pois ao utilizá-la vinha muito código `html` junto do *output*, irei disponibilizar este *output* na íntegra junto do atual documento

Identifique mitigações possíveis de aplicar

Ativação de HTTPS com certificado válido
O serviço web `CasaOS` deve ser protegido com TLS. Para isso, pode-se configurar um *reverse proxy*, como o `Nginx`, na porta `443/tcp`, com suporte a certificados válidos (ex.: `Let's Encrypt`). Embora a configuração pareça acessível, reconheço que ainda não tenho experiência prática com esse tipo de infraestrutura, o que torna o processo algo desafiante para mim neste momento.

Fortalecer acesso SSH
Já utilizo autenticação apenas por chave pública, sem uso de senhas, para aceder ao serviço `SSH`, o que é uma boa prática de segurança. Como melhorias adicionais, poderia restringir o acesso a IPs específicos através de firewall, e configurar ferramentas como o `fail2ban`[4] para mitigar ataques de força bruta.

Monitorização e atualizações contínuas
É importante garantir que serviços como o `OpenSSH` e o servidor HTTP em Go estejam atualizados com as últimas correções de segurança. A configuração de logs de acesso e alertas também ajuda na deteção precoce de comportamentos suspeitos.

Validação do software CasaOS
Sendo o `CasaOS` um software de terceiros, é recomendável validar sua origem e integridade. Isso inclui verificar assinaturas de binários, aplicar atualizações com frequência e auditar permissões de acesso locais.

Gestão de acessos com UFW
O sistema utiliza o `UFW` (*Uncomplicated Firewall*) como camada de controle de tráfego de rede. As portas que tenho abertas explicitam que andei a testar coisas, decerto convém fechá-las.

[4] `https://wiki.archlinux.org/title/Fail2ban`

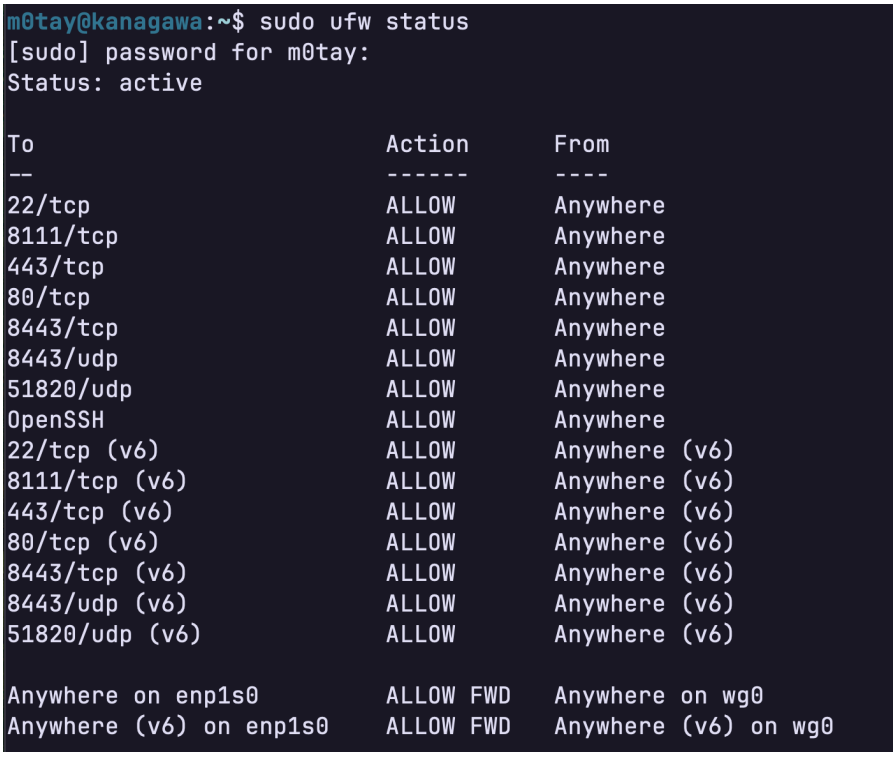

Figura 1.3: Evidência do estado do meu *firewall* no meu servidor pessoal

PARTE IV - Deteção de Vulnerabilidades

Explore a ferramenta `snort`

O `Snort` é um dos principais Sistemas de Prevenção de Intrusão (*IPS*) de código aberto do mundo. Ele usa uma série de regras que ajudam a definir atividades maliciosas na rede, e aplica essas regras para identificar pacotes suspeitos, gerando alertas para os utilizadores.

A ferramenta `snort` pode funcionar como *sniffer* para pacotes da camada 2

Ao utilizar o comando "`$ sudo snort -vde`", a ferramenta é executada em modo *sniffer*, ou seja, apenas captura e exibe os pacotes da rede, sem aplicar regras nem gerar alertas. Essa execução mostra detalhes da camada de enlace (como endereços `MAC`), da camada de rede (como cabeçalhos IP), e ainda o conteúdo da carga útil (*payload*) dos pacotes, em representação textual e hexadecimal.

É uma forma prática de observar o tráfego bruto da rede em tempo real, especialmente útil para análise de segurança, testes de visibilidade e para compreender o comportamento dos protocolos.

A seguir apresento uma amostra dos pacotes capturados na minha rede:

Figura 1.1: *Output* inicial do `snort`

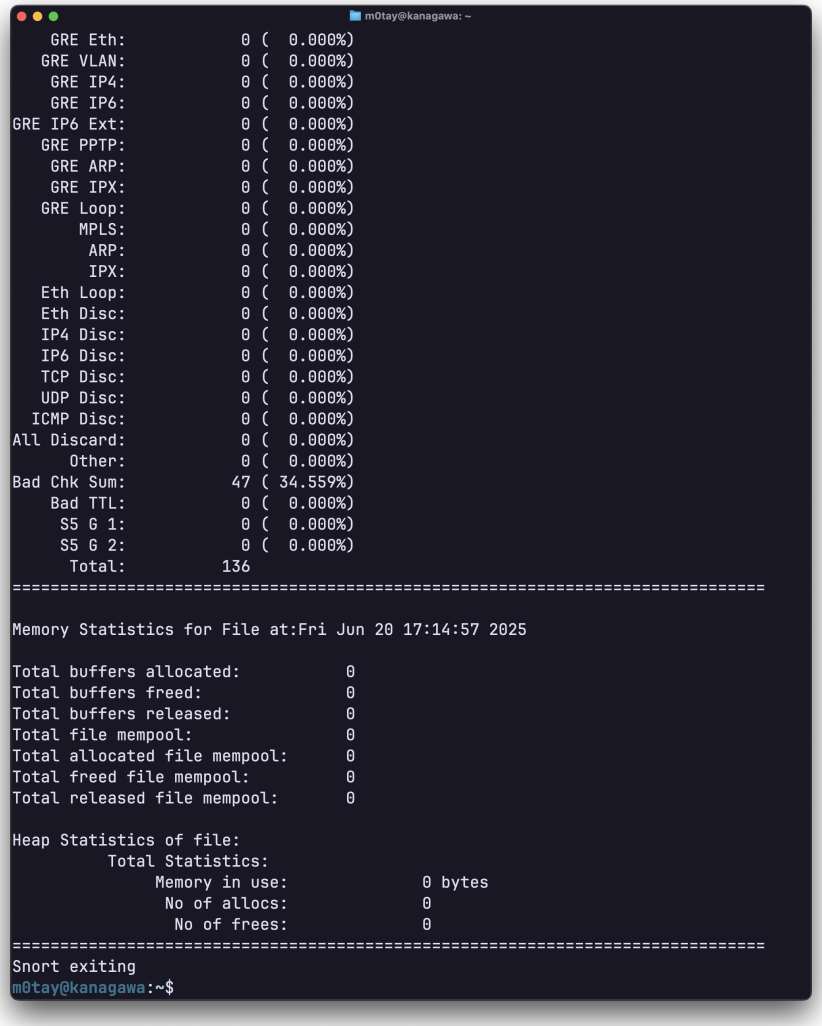

Figura 1.2: *Output* final do `snort`

Utilização do `snort` como IDS pressupõe a utilização de regras que podem ser personalizadas

Utilizando a regra de exemplo oferecida pelo professor, inserimo-la no ficheiro responsável por conter as regras locais:

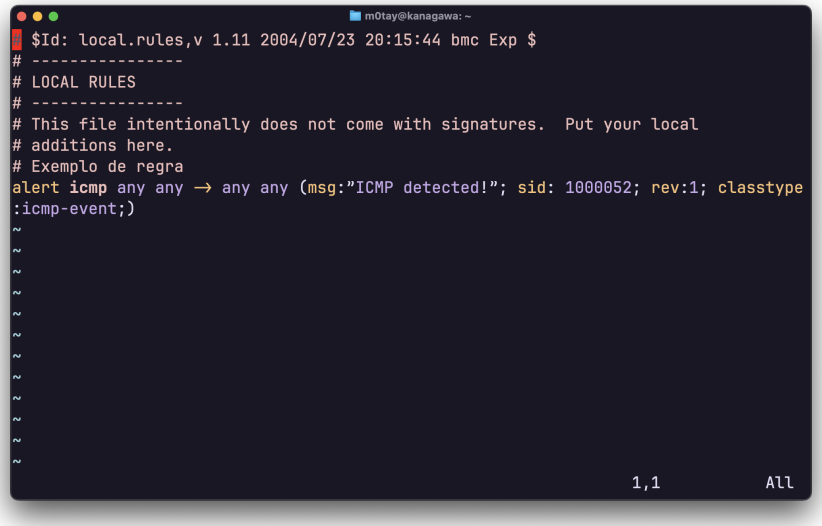

Figura 1.3: Adicionando a regra oferecida pelo professor

Iniciar o `snort` como IDS e visualizar os alertas

Ao inicializar o `snort` como um IDS, abri outro terminal e realizei pings ao servidor e a outros dispositivos na rede. O resultado obtido foi o seguinte:

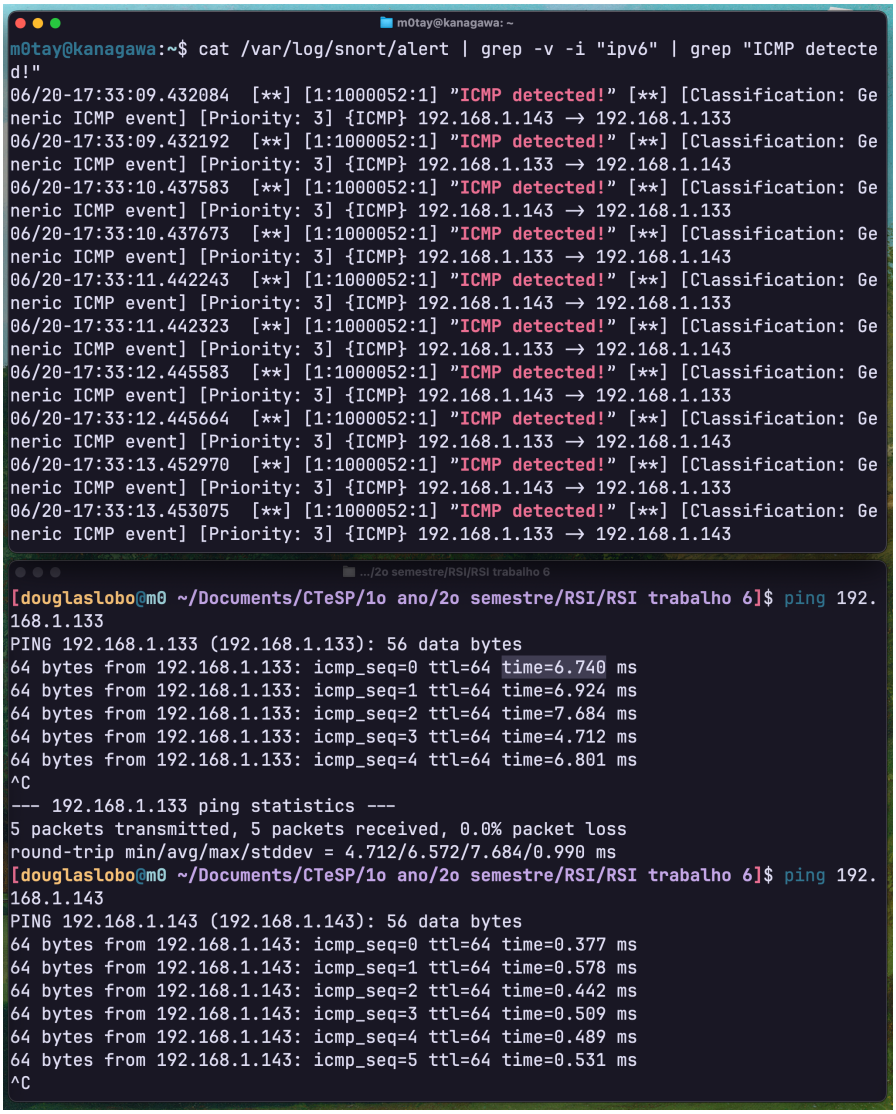

Figura 1.4: Evidência dos resultados dos logs da IDS do `snort` e dos *pings* feitos

Com base no abordado diga como poderia utilizar um NIDS para garantir mais segurança numa rede informática

Como vimos na UC, a utilização de um *Network-based Intrusion Detection System* (`NIDS`) é uma forma prática de reforçar a segurança da rede, sobretudo quando se pretende acompanhar vulnerabilidades conhecidas ao nível dos protocolos ou da prestação de serviços ataques a DNS, spoofing ARP ou simples scans de porta são bons exemplos disso.

O NIDS funciona bem posicionado junto ao ponto de entrada da rede (por exemplo,

junto ao router) e permite observar o tráfego em tempo real, quer se trate de assinaturas específicas de ataques quer de padrões anómalos. Um cenário típico como os que discutimos (reconhecimento externo, ataque, recrutamento) pode ser identificado com base nas tentativas de *port scanning*, ligações suspeitas a serviços vulneráveis ou contactos a C&C.

Também é importante lembrar que o NIDS, para além de gerar alertas, pode integrar-se com outras ferramentas como *honeypots* e mecanismos de logging, funcionando como mais uma fonte de eventos no ecossistema de deteção.

A nível dos objetivos da segurança, a sua utilização contribui diretamente para:

- preservar a **confidencialidade**, sinalizando fugas de dados;
- garantir a **autenticidade**, como nos casos de spoofing;
- manter a **disponibilidade**, detetando ataques como o DoS.

Uma ferramenta particularmente eficaz neste contexto é o `Snort`, um `NIDS` comportamental *open source* amplamente utilizado. A sua flexibilidade na definição de regras e a capacidade de inspeção em múltiplas camadas tornam-no uma escolha robusta para ambientes que exigem deteção de intrusões em tempo real.