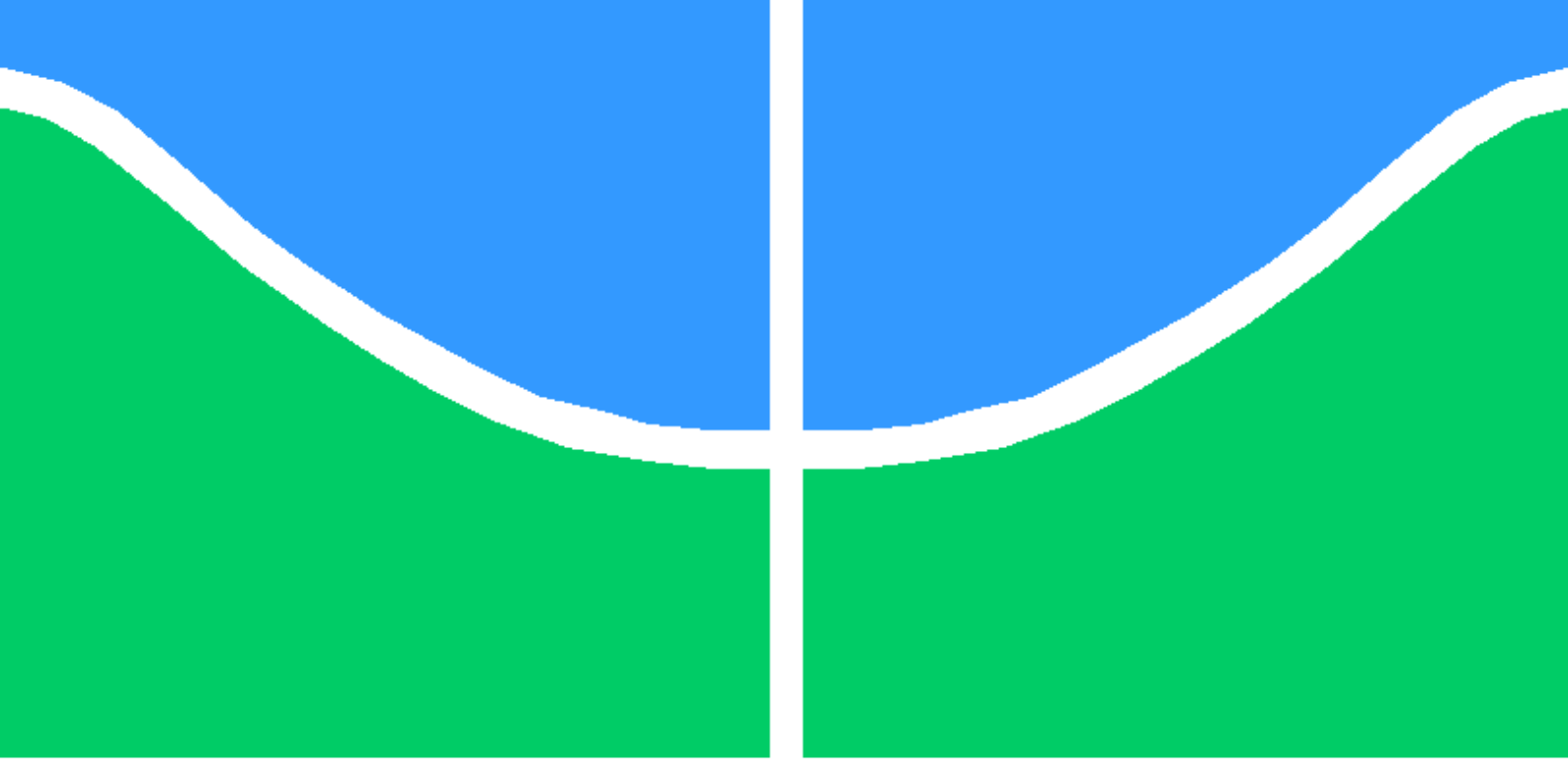


Trabalho Final de Graduação

Análise do uso de Entropia para detecção de tráfego criptografado no payload da camada de transporte

João Fiuza de Alencastro

Brasília, novembro de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Trabalho Final de Graduação

Análise do uso de Entropia para detecção de tráfego criptografado no payload da camada de transporte

João Fiuza de Alencastro

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Flávio Elias Gomes de Deus, ENE/UnB _____
Orientador

Prof. Robson Albuquerque, ENE/UnB _____
Co-Orientador

Prof. Georges Daniel Amvame Nze, ENE/UnB _____
Examinador

Prof. João José Costa Gondim, CIC/UnB _____
Examinador

Agradecimentos

Dedico este trabalho aos meus pais, que sempre deram tudo de sí para me proporcionar uma boa educação. Dedico também, aos meus irmãos que estão sempre comigo, não importa a situação. Dedico também este trabalho a todos os meus amigos que tive a sorte de ter ao meu lado. Gostaria aproveitar essa oportunidade para exaltar a educação pública do nosso país e agradecer a todos os professores da Universidade de Brasília que tiveram alguma influência em minha trajetória. Além disso, o laboratório Latitude e seus projetos são de extrema importância para os alunos e aos professores e causam um tremendo impacto em nosso departamento de forma positiva.

Resumo

Tendo em vista o contexto em que vivemos hoje, onde vazamentos de dados, ataques de ransomware e infecção por *malwares* são cada vez mais frequentes, aplicações estão sendo reestruturadas levando a segurança em consideração. Muitas delas estão sendo padronizadas com a criptografia ponta-a-ponta, sendo realizada na própria aplicação, como o HTTPS. Ao mesmo tempo, torna-se mais comum *appliances* empresariais serem capazes de inspecionar o tráfego criptografado por interceptação, realizando verificações dentro do conteúdo da aplicação. Já as aplicações que não deveriam ser criptografadas por natureza, podem passar despercebidas ao serem criptografadas por implementações clandestinas de atacantes. Esses casos almejam criar fluxos encapsulados, a fim de extrair dados, criar estruturas de comando e controle ou injetar códigos e scripts infectados. Ferramentas de detecção de intrusão por assinatura falham ao tentar analisar fluxos que são disfarçados por uma aplicação legítima, como o DNS, por exemplo. Ao passo que, sistemas de detecção por anomalia também podem ser dribladas por estes mesmos tipos de mimetismo. Portanto, vê-se a necessidade de desenvolver um sistema capaz de verificar se há criptografia a nível de aplicação sem interceptá-lo. O método proposto por este trabalho utiliza a entropia para analisar a possibilidade de classificar as aplicações a partir do payload da camada de transporte. Atingindo assim, uma abstração das camadas acima e sendo capaz de obter resultados significativos sobre aplicações genéricas, independentemente do protocolo de aplicação utilizado.

Abstract

By the context we live in, where loss of data, ransomware attacks and infection by malware seem to be more and more frequent, applications are being restructured while taking security into consideration. Many of those are being standardized with end-to-end encryption at the application layer, as HTTPS. Meanwhile, more and more enterprise appliances are able to intercept traffic and execute scans inside the payload. On the other hand, applications which were not supposed to be encrypted can go unnoticed if they are encrypted by attacker's clandestine implementations. These cases aim to create encapsulated flows, trying to exfiltrate data, create command and control structures or inject malicious codes and scripts. Signature based intrusion detection systems lack when trying to analyse flows that are disguised as a legitimate, such as DNS. Whereas anomaly based intrustion detection systems also can be dribbled by these same mimicry mecanisms. Therefore, there is a need to develop a system capable of verifying encryption at the application layer without having to decipher and cipher again the content. The proposed method by this work resorts to entropy in an attempt to analyse the possibility of classification of applications from the transport payload perspective. Reaching an abstraction of the above layers and being able to achieve meaningful results about generic applications, regardless of the application protocol.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Abreviatura	**Significado**
AES	Advanced Encryption Standard
API REST	Application Programming Interface Representational state transfer
APT	Advanced Persistent Threat
ASCII	American Standard Code for Information Interchange
BiEn	BiEntropia
CDN	Content Delivery Network
C&C	Command and Control
DNS	Domain Name System
DoH	DNS over HTTPS
DPI	Deep Packet Inspection
ELK	Elasticserach Logstash Kibana
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IaC	Infrastructure as Code
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
LGPD	Lei Geral de Proteção de Dados
ML	Machine Learning
NDJSON	Newline Delimited JavaScript Object Notation
NIDS	Network Intrusion Detection System
NIPS	Network Intrusion Prevention System
NIST	National Institute of Standards and Technology
NGFW	Next-Generation Firewall
OpenSSL	Open Secure Sockets Layer
RFC	Request for Comments
SlowDoS	Slow Denial of Service
SSL	Secure Sockets Layer
SRTP	Secure Real-time Transport Protocol
SVM	Support Vector Machine
TBiEn	Tres BiEntropia
TLS	Transport Layer Security

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAF	Web Application Firewall

1 Introdução

A captura e análise de tráfego são técnicas frequente na área de redes. Seja porque é necessário realizar uma checagem de rotina e conferir se o tráfego está normal. Seja por uma finalidade de encontrar algum erro na rede. Seja por motivos de segurança. A inspeção de pacotes é uma técnica útil na área de segurança da informação, controlando ou observando de certa forma os dados que trafegam por uma dada interface de rede. Ao longo do tempo, novas tecnologias foram surgindo, utilizando-se da captura ao seu favor, como, por exemplo, *firewalls*, sistemas de detecção de intrusão, ou até mesmo dispositivos clandestinos de interceptação e *'gathering'* de informações sem autorização. Ferramentas que realizam uma captura em determinada rede são comumente chamados de *'Network Sniffer'* pela comunidade, e são de fácil acesso.

Tendo em vista que pessoas má intencionadas sabem da facilidade de detectar informações trafegadas na Internet, juntamente à facilidade do uso de bibliotecas de criptografia, muitos dos fluxos ilegítimos se tornaram criptografados a nível de aplicação. Segundo [15], em janeiro de 2017, o tráfego criptografado da Internet ultrapassou o tráfego não-criptografado. É explicado também que, apesar de a criptografia aperfeiçoar a segurança, também traz proteção para intrusões e ataques ao bloquear acesso externo ao conteúdo de pacotes.

1.1 Definição do Problema

Já não é dúvida que a segurança cibernética não é mais trabalhada somente na camada de rede como era na década passada. Com o advento de aplicações com criptografia ponta-a-ponta, é simplesmente perigoso permitir que usuários comuns fechem canais de comunicação quaisquer e totalmente privados de supervisão em redes empresariais. Ou pelo menos, para transmissões que por padrão não deveriam conter um conteúdo criptografado. Como, por exemplo, em [11], trabalha-se a ideia de um sistema que detecta por anomalia ataques do tipo *SlowDoS* sobre tráfego criptografado a nível de aplicação. O trabalho se baseia no pressuposto de que tradicionais sistemas de detecção de intrusão por assinatura vêm tendo dificuldades para detectá-los, principalmente quando utilizam-se de tráfego criptografado em HTTP, protocolo que por natureza não é criptografado.

Além disso, protocolos que já fazem uso da criptografia por padrão podem não ser o vetor de atacantes, já que, por naturalmente esconderem o conteúdo de uma comunicação, espera-se que possa haver algo malicioso ali presente. Ou seja, existem métodos de interceptar tais comunicações, descriptografá-las, analisá-las, e as criptografar novamente, como mostra [18] e tecnologias patenteadas do mercado que são amplamente difundidas. Muitos dos novos *appliances* de segurança já se preocupam em fazer a intermediação de fluxos SSL/TLS (Secure Sockets Layer/Transport Layer Security). Tal atividade realizada por estes novos equipamentos se veem restritas ao criarem e re-assinarem certificados, mas que mesmo assim são incapazes de detectar 100% dos ataques realizados todos os dias pelo mundo.

Um problema constante na segurança da informação é o de equipamentos convencionais de

segurança de perímetro como NIDS (*Network Intrusion Detection System*), NIPS (*Network Intrusion Prevention System*), NGFW (*Next Generation Firewall*) e WAF (*Web Application Firewall*) necessitam saber qual o tipo de informação que está sendo trafegada por suas interfaces. Alguns métodos de classificação já são conhecidos e utilizados por muitos. Como, por exemplo, o reconhecimento de assinaturas produzidas pelos metadados presentes nos cabeçalhos. Ou o reconhecimento de padrões no estilo de comunicação, como, por exemplo, os tempos de intervalos entre pacotes ou tamanhos dos pacotes. Ou até, pelo frágil reconhecimento de porta e protocolo de transporte. E ainda assim, utilizar-se de todos estes métodos em conjunto, ainda não se pode ter certeza de que tais fluxos são legítimos e confiáveis. Isso porque, ataques podem fazer um uso correto de uma certa aplicação, de maneira que logicamente aparenta ser legítima, mas que extrai dados de uma rede usando uma simples implementação de criptografia sobre seu conteúdo. A criptografia foi abundantemente adotada, e continuará sendo, pois é de fácil acesso, como especifica [3], mostrando que bibliotecas como a *OpenSSL* [17] são abertas, extremamente difundidas e podem ser utilizadas para criptografar qualquer tráfego.

Pode se observar em [1], um trabalho dedicado a detecção de *'exploits'* nos canais DNS (*Domain Name System*) para exfiltrar dados e também manter um túnel de *'Command and Control'* (C&C) para malwares. Ou seja, protocolos vitais para o funcionamento da Internet, como o DNS, são utilizados de forma distorcida para se extrair dados ou enviar comandos e scripts de execução remota de forma maliciosa.

Como explica [30], a classificação de tráfego criptografado ainda está em estágio inicial de desenvolvimento. E o que é visto em estudos recentes como, [14] [5] [28] dentre outros, é que a entropia pode ser uma grande aliada nesse estudo, o assunto é mais detalhado na seção 2.6.

1.2 Objetivo

Este trabalho tem como objetivo geral validar, de maneira teórica e prática, a classificação de fluxos de rede em tráfegos criptografados por meio de indicadores de entropia. A classificação deverá guiar administradores de redes a fim de detectarem ameaças. Objetiva-se realizar testes nas formas de texto plano, texto criptografado e ainda textos comprimidos. Textos comprimidos deverão entrar na aferição para que se verifique se haverão falsos positivos por tais dados serem altamente estruturados. Por fim, deve-se concluir se há possibilidade de detectar comportamentos possivelmente anômalos.

1.2.1 Objetivos específicos

A fim de atingir o objetivo proposto, foi necessário realizar os seguintes objetivos específicos:

- Desenvolver métodos de transmissão genéricas de fluxos de pacotes
- Aplicar métodos de captura, extração e filtragem de acordo com a necessidade
- Montar a arquitetura de coleta, ingestão e indexação dos dados
- Implementar métodos de processamento de entropia e geração de gráficos

1.3 Organização do trabalho

O trabalho se organiza da seguinte forma: No capítulo seguinte, é feita a fundamentação teórica dos assuntos abordados, ao mesmo tempo que se traz diversos trabalhos estado-da-arte que foram produzidos recentemente. Em seguida, no terceiro capítulo, é discutido o problema que está sendo abordado e é mostrado a arquitetura, a metodologia, o fluxo dos dados propostos por este trabalho. Logo após, a fim de criar um alicerce empírico e estatístico para as hipóteses, são apresentados os cenários contendo seus respectivos resultados. A seção de cenários é seguida pela conclusão do trabalho, criando a relação entre a prática e a teoria. Juntamente da conclusão, foi realizada a ligação aos trabalhos futuros, abrindo espaço portanto, para uma continuação do problema apresentado e, aprimorando assim, a tecnologia criada.

2 Fundamentação Teórica e Estado da Arte

A padronização funciona como um alicerce para o conhecimento humano. Diversos processos, métodos, sistemas e tecnologias dependem da padronização para seus funcionamentos, que por suas vezes, dependem de outros processos, e assim por diante. A própria Internet é exemplo de uma tecnologia que depende fortemente da padronização entre elementos, parte do motivo da criação da IETF (*Internet Engineering Task Force*). Portanto, nesta seção serão apresentados conceitos-chave para o entendimento geral do sistema proposto. São também, apresentados os trabalhos estado-da-arte relacionados ao estudo realizado.

2.1 Estado da Arte

Em [14] traz-se uma proposta similar à deste trabalho, o qual se propõe uma abordagem à classificação de tráfego de rede criptografado baseado no cálculo de entropia e em técnicas de ML. Foi utilizada a entropia de Shannon e a BiEntropia. O modelo de ML utilizado foi o *Support Vector Machine* (SVM). Contudo, o que não foi feito naquele estudo, diferentemente deste, foi a indexação de metadados dos pacotes após eles terem sido processados, o que garante um ambiente multi-funcional, de caráter observativo e auditável. Isso não só traz uma perspectiva real a uma possível resposta a um ataque de forma instantânea, como também correlaciona eventos na infraestrutura.

Em [5], é também prospota uma classificação de tráfego criptografado onde são utilizados os chamados *"magic header bytes"*, utilizados para detectar muitos algoritmos de compressão com suas *fingerprints* dos primeiros bytes dos pacotes. É dito que isso implica diretamente na classificação de seus formatos, o que leva a uma melhor acurácia de classificação. Porém, este método depende do cabeçalho utilizado para encapsular o *payload* da comunicação, prática desconexa com a proposta por este trabalho, já que parte-se do pressuposto que pouco importa a aplicação que esteja sendo utilizada no momento da análise. Ou seja, não há restrição para a aplicação utilizada. Além disso, as análises feitas em [5] são realizadas em pacotes de maneira individual, sem criar uma correlação entre os pacotes de mesmo fluxo.

Outro artigo que segue a mesma ideia de classificação de tráfegos criptografados e comprimidos por uso de entropia é o [28]. É um trabalho similar ao realizado por [5] ao que se trata da formulação do problema, contudo suas diferenças se acentuam nas metodologias utilizadas. Por exemplo, em [28] o método de cálculo da entropia utilizado foi somente o de Shannon, o qual falha em apontar elevados níveis de desordem entre os caracteres, apesar de uma possível alta distribuição no texto, como descrito em 2.6.1.

Em [30], o artigo propõe uma abordagem semelhante à esta em seus primeiros estágios, no que diz respeito à classificação de tráfego criptografado por meio de entropia. Contudo, duas grandes diferenças se destacam ao ler o artigo. Primeiramente, o artigo citado foca seus estudos em três protocolos principais, estes são, IPsec, SSL/TLS e SRTP. Ou seja. são avaliados protocolos que já

utilizam a criptografia por padrão, deixando de lado portanto, situações peculiares onde atacantes não utilizam tais protocolos. Além disso, são utilizadas métricas como, tamanhos dos pacotes, tempo de chegada entre pacotes e direção para avaliar com mais precisão o tráfego. Contudo, o sistema proposto por [30] deixa de analisar cenários onde há peculiaridades na rede ou na aplicação, o que podem mudar drasticamente tais valores a depender do ambiente.

O trabalho apresentado em [9] propõe um sistema capaz de detectar exfiltração de dados usando características de entropia e criptografia do tráfego de rede. O trabalho em questão é similar a este no sentido de que tem como objetivo ajudar administradores de rede a entenderem melhor o que está se passando, se pode haver algo malicioso trafegando. No caso de [9], o foco foi a exfiltração de dados, porém a criptografia pode ser um vetor para outros tipos de ataques, como, por exemplo, estrutura de comando e controle, injeção de código, *cross site scripting* e afins. Ademais, a abordagem do trabalho citado em relação à entropia se faz presente somente na forma da entropia de Shannon, a qual se limita somente à frequência dos caracteres, como descrito em 2.6.1.

2.2 AES 256

O algoritmo AES (*Advanced Encryption Standard*) é uma especificação para a criptografia de dados e foi introduzido pelo *'U.S. National Institute of Standards and Technology'* (NIST) em 2001 [2]. O documento explica, "O algoritmo AES é uma cifra de blocos simétrica que pode criptografar (cifrar) e descriptografar (decifrar) informação. A criptografia converte os dados para um formato ilegível chamado *'ciphertext'*; descriptografando o *ciphertext* converte os dados de volta para sua forma original, chamada *'plaintext'*".

O algoritmo AES 256, apesar de ser ter sido introduzido a relativamente bastante tempo, considerando o âmbito da tecnologia, ele ainda é utilizado por protocolos recentes e que são padrões *de facto*. Como, por exemplo, a mais nova versão do TLS, 1.3, descrita pela RFC 8446 [25]. Isso acontece porque o algoritmo utiliza-se de princípios matemáticos para criptografar dados, e ainda faz uso de chaves simétricas, característica de algortimos de criptografia muito fortes. Por estas razões, este foi o método escolhido para ser utilizado como base de textos criptografados nos experimentos que virão.

O algoritmo AES 256 é utilizado hoje como padrão de criptografia para diversos protocolos. Sua implementação é de fácil acesso em diversas linguagens de programação e a criação de sua instância não requer conhecimentos avançados. Portanto, torna-se comum a prática de implementações de novas aplicações que fazem uso desse algoritmo, inclusive implementações clandestinas que tem como objetivo exfiltrar dados, ou encapsular o tráfego. Por esses motivos, o AES 256 foi o padrão escolhido para ser utilizado em alguns dos cenários de testes propostos por este trabalho.

2.3 Estrutura de Comando e Controle C&C

Em [12] é esclarecido que, muitos ataques comuns como *Trojan*, *Botnet* e *Advanced Persistent Threats (APT)* precisam estabelecer um canal de comunicação conhecido como comando e controle.

Por esse canal, serão transmitidos comandos remotos, informações sensíveis, e outros dados que enriquecem um atacante. E por tal motivo, para evitarem de serem detectados, ou para evadirem barreiras defensivas, *hackers* constroem canais disfarçados para estabelecerem *C&C*, ou seja, realizam tunelamentos. Ainda em [12], são especificados três métodos de detecção na literatura atual: detecção baseada em assinaturas, detecção baseada em anomalia de protocolo, e detecção baseada em comportamentos e estatísticas.

Em [21] foi feita uma análise comprovando o recrudescimento do uso de criptografia em comunicações de comando e controle, permitindo assim, uma maior taxa de evasão contra sistemas de detecção de intrusão. A criptografia lhe confere uma forma de evadir combinações de assinatura para pacotes desse tipo.

2.4 Exfiltração de dados

O trabalho realizado em [27] define exfiltração de dados como um dos maiores vetores de ataques cibernéticos, cujo objetivo é o vazamento de dados sensíveis internos a uma organização para uma entidade sem autorização. É dito ainda que, é comum que métodos convencionais de comunicação sejam utilizados para realizar tais ataques.

Com o advento de novas leis de proteção de dados pelo mundo, como a LGPD no Brasil, empresas não podem mais correr o risco de vazar informações sigilosas sobre clientes, como, por exemplo, dados de cartões de crédito, CPFs ou documentos confidenciais. Ou até mesmo vazamentos em órgãos governamentais que possuem inteligência sobre futuros projetos, segredos nacionais ou estratégias políticas. Caso não sejam tratados com um alto nível de segurança, podem gerar consequências catastróficas.

Para evadir mecanismos de detecção de intrusões, *firewalls* e afins, exfiltrações podem muitas vezes optar por mascarar tal fluxo por meio do uso de criptografia a nível de aplicação. De forma que não haja assinaturas compatíveis ou inspeções de pacote efetivas no momento de injeção, implantação ou na própria exfiltração dos dados. O trabalho realizado em [9] foca em um método de detecção de exfiltração de dados utilizando entropia e características do tráfego criptografado. É dito que dados podem ser exfiltrados de várias maneiras, mas que os métodos mais comuns são por meio de *botnets*, vírus, *worms*, *rootkits*, FTP, acesso físico como em pen-drives e CDs, e-mails e até ataques de *fishing*. Ainda em [9], são mostradas várias formas de como um administrador de redes pode se situar em relação aos tráfegos presentes em seu ambiente, utilizando ferramentas, como, *IP Helper*, *DNS Extractor*, *Network Top*, dentre outras. Porém, nenhuma deles permite uma visualização completa quando se há criptografia ponta-a-ponta envolvida. Em sua metodologia, o trabalho descreve como foi utilizado a entropia para detectar fluxos criptografados. Contudo, o cálculo utilizado é a conhecida entropia de Shannon, proposta por C.E. Shannon em [23], pioneiro no ramo da teoria da informação, que determina o grau de previsibilidade dos símbolos em um texto, mas que por si só não diz muito sobre a disposição e a ordem desses símbolos. O assunto é abordado com mais clareza na seção 2.6.1.

2.5 Deep Packet Inspection

De acordo com [13], *Deep Packet Inspection* (DPI) é um método melhorado de filtragem de rede aplicado na camada de aplicação da arquitetura OSI. É dito ainda que, o DPI detecta, tanto a parte de dados (conteúdo do pacote), quanto a parte da assinatura (ID do pacote). Essa filtragem faz uso de diversos algoritmos e até mesmo de expressões regulares.

Em [20], diz-se que ao ler a parte de dados do pacote, procura-se falta de *compliance* de protocolos, vírus, *spam*, intrusões ou critérios pré-definidos para decidir se um pacote pode passar, se deve ter outro destino ou se é necessário coletar dados estatísticos. É dito ainda que, o DPI permite gerenciamento avançado de rede, serviço de usuário e funções de segurança, assim como mineiração de dados, *eavesdropping* e censura.

Ou seja, o método DPI engloba diversas ações que podem ser realizadas em cima de dados coletados na rede, pois parte do pressuposto que o pacote está sendo analisado por dentro, da forma mais granular possível. Isso abre infinitas possibilidades do que se pode fazer com essa informação, especialmente para o ramo de segurança da informação.

2.6 Entropia

A entropia pode ser classificada como o grau de incerteza ou aleatoriedade de algum fenômeno. Existem vários métodos de como calcular o nível de ordem/desordem ao nosso redor. No ramo da Teoria da Informação, um método quantitativo extremamente utilizado é o da teoria de Shannon de Entropia. Porém, vemos em [6], outros métodos de diferentes complexidades contribuírem para as análises estatísticas e experimentais.

A entropia de bytes é uma métrica usada para medir a aleatoriedade dos bytes, quanto maior a entropia, maior é a incerteza de premeditar o valor de uma observação.

Em [7], são citados diferentes métodos de se calcular a entropia e chegar a um escalar que defina o grau de aleatoriedade e desordem de *strings* finitas. E que além disso, são divididos em dois grupos os testes e algortimos existentes: aqueles que usam a técnica de janela deslizante para examinar *substrings* de uma *string* original, e aqueles que utilizam o tamanho original da *string*.

2.6.1 Entropia de Shannon

A entropia de Shannon leva em consideração a frequência dos caracteres em um determinado conjunto, contudo não é levado em consideração a ordem na qual esses caracteres se encontram. A equação para a entropia de Shannon se encontra abaixo 2.1:

$$H(x) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{2.1}$$

Porém, de acordo com equação 2.1, não há um limite de quantos bits de informação pode carregar um texto, dependendo de sua codificação e o tamanho de possibilidades do alfabeto. Isso acontece, pois o valor máximo da entropia variará de acordo com o tamanho de possibilidades que

um evento pode assumir. No caso da entropia de Shannon, a entropia terá como máximo $log_2 m$, onde m é o tamanho de símbolos da mensagem. Caso a entropia esteja sendo calculada em bits, m será o número de bits que a mensagem tem. Por tal motivo, independentemente da base em qual será calculada a entropia, para fins comparativos, a entropia de Shannon foi normalizada. Essa normalização foi realizada ao dividirmos o resultado da fórmula, por $log_2 m$, limitando assim, o valor final a um intervalo entre 0 e 1, procedimento realizado também em [28]. Portanto, neste trabalho a probabilidade $P(x_i)$ é probabilidade de determinado símbolo hexadecimal, onde o somatório vinculado ao i percorre as possibilidades do alfabeto hexadecimal que variam de 0 a f. Essa lógica serve também para as definições de entropia de BiEn e TBiEn, explicadas nas seções seguintes.

2.6.2 BiEntropia

O conceito de BiEntropia, descrito por [7], propõe um método onde é possível diferenciar o nível de desordem nos dígitos de uma *string* qualquer de tamanho arbitrário. É utilizada uma função chamada *'BiEntropy'*, que envolve calcular uma média ponderada das entropias de Shannon de toda a *string*, exceto em suas duas últimas derivadas binárias. A derivada binária de um determinada *string* de n bits é formada pelo cálculo XOR de pares adjacentes de bits. Portanto, a primeira derivada binária de s, $d_1(s)$, tem $n-1$ bits em tamanho.

Segundo [7], "a BiEntropia é uma média ponderada das entropias binárias de Shannon da *string* e das primeiras $n-2$ derivadas binárias daquela *string* utilizando simples potência". A BiEntropia pode ser calculada pela equação 2.2

$$BiEn(s) = (\frac{1}{2^{n-1}-1})(\sum_{k=0}^{n-2}((-p(k)\log_2 p(k) - (1-p(k))\log_2(1-p(k))))2^k) \tag{2.2}$$

2.6.3 TBiEntropia

É dito também que, caso as maiores derivadas de uma *string* binária arbitrariamente longa sejam periódicas, toda a sequência também exibirá periodicidade. E para o caso contrário, ou para todos os casos, é utilizado uma segunda versão da BiEntropia, a Tres BiEntropia, a qual utiliza ponderação logarítmica para avaliar o conjunto completo de longas séries de derivadas binárias. As conclusões tiradas no trabalho descrito pelo cálculo da BiEn e da TBiEn para *strings* 4 e 8 bits mostraram que suas distribuições diferem entre sí na maneira como ordenam strings de 8 bits parcialmente periódicas. Já para as *strings* de 8 bits aperiódicas e periódicas, são ordenadas da mesma forma. Portanto, a utilidade da comparação entre BiEn e TBiEn deverá ser determinada experimentalmente. A Tres BiEntropia pode ser calculada através da fórmula descrita em 2.3.

$$TBiEn(s) = (\frac{1}{\sum_{k=0}^{n-2}\log_2(k+2)})(\sum_{k=0}^{n-2}(-p(k)\log_2 p(k) - (1-p(k))\log_2(1-p(k)))\log_2(k+2)) \tag{2.3}$$

2.7 Métodos estatísticos

Os métodos estatísticos utilizados neste trabalho são os testes de Shapiro-Wilk, introduzido em [24], utilizado para determinar com certo nível de confiança se uma amostra de dados tem uma distribuição normal (gaussiana). Além do Shapiro-Wilk foi utilizado o teste Wilcoxon Signed Rank Test, descrito em [29], que pode ou não refutar a hipótese nula que demonstra igualdade entre duas distribuições ordenadas não-normais. O teste estatístico é descrito por 2.4.

$$W = \sum_{i=1}^{N_r} [sgn(x_{2,i} - x_{1,i}) R_i] \tag{2.4}$$

Para a equação 2.4, W é o teste estatístico, N_r é o tamanho da amostra, excluindo pares onde $x_1 = x_2$, sgn é a função sinal, $x_{1,i}, x_{2,i}$ são os pares ordenados correpondentes das duas distribuições e R_i é o ranque i. A execução do método consiste em um algoritmo composto por uma série de passos, onde primeiro criam-se ranques das duas distribuições e se pontua cada ranque, de acordo com os valores das amostras. Em seguida, calcula-se o valor de W com a fórmula apresentada em 2.4. Uma vez em posse desse valor, pode-se refutar ou não a hipótese nula, a depender do valor escolhido para α.

3 Discussão do Problema e Proposta de Solução

Em qualquer solução proposta, vê-se a necessidade de discutir os componentes e a arquitetura da solução a fim de contextualizar o projeto. Nesta seção, portanto, serão discutidos quais foram os empecilhos encontrados ao longo do projeto e suas complicações. Serão também abordadas as ferramentas utilizadas para atingir os objetivos específicos. E por fim, como esses componentes se encaixam e se integram de forma a atingir o objetivo geral.

3.1 Descrição do Problema

Um fato é que softwares e hardwares de segurança já podem fazer uma interceptação no tráfego TLS/SSL. Podem portanto, inspecioná-los sem o empecilho da criptografia. Contudo, e quanto aos protocolos que não utilizam criptografia por padrão, mas que por algum motivo estão embaralhados? Por que suas respectivas entropias estão com valores elevados? Poderiam eles ter sido criptografados de forma clandestina? Se sim, o mimetismo do encapsulamento de aplicações legítimas é o suficiente para que atacantes possam passar despercebidos em uma rede empresarial? Os problemas encontrados giram em torno destas perguntas.

3.1.1 Viabilidade da classificação de tráfego criptografado pelo uso da entropia

O problema traz uma questão interessante à tona: Qual é a viabilidade de classificar o tráfego criptografado pela entropia? Pergunta esta que já foi imposta por outros pesquisadores e que, pelo estudo do Estado-da-Arte se mostrou positiva até então. Mas essa dúvida gera outros questionamentos, como, por exemplo: Quais ferramentas podem auxiliar no processo? Quais métodos do cálculo da entropia podemos utilizar e qual será o escopo? Quais são os custos computacionais associados a tais procedimentos? Nesta seção iremos conversar sobre a metodologia do trabalho e como foram realizados os procedimentos que embasam a teoria deste projeto.

3.1.2 Dificuldade em classificar tráfego

Como foi dito em algumas menções a trabalhos relacionados, existe um grande problema na identificação da aplicação que trafega em determinada interface sem ser uma das partes comunicantes. Este é um problema que já é abordado desde os primórdios da Internet e que hoje só se agrava com a maior utilização da criptografia.

A criptografia dificulta a determinação do conteúdo, porque não se pode simplesmente levar em consideração os metadados definidos no cabeçalho de uma pacote e simplesmente acreditar que o que está sendo trafegado é legítimo. Pelo menos não para a segurança, se é feita de forma cautelosa. Métodos já conhecidos que se utilizam de assinaturas já devem ser descartados nesse momento, pois assinaturas não funcionam com algoritmos de criptografia, já que com um texto

totalmente embaralhado, não se há compatibilidade. Se a assinatura é feita no cabeçalho, volta-se ao problema anterior, onde não há certeza do que está no *payload*. É importante lembrar também que, uma simples inspeção de número de portas padrões assinaladas pela IANA, como especifica o RFC 6335 [19], não é um mecanismo confiável de classificação de serviço, já que a troca de portas é uma prática comum e já esperada, como explica [3].

Um empecílio encontrado neste estudo, descrito em [14], [28] e [5], são os dados comprimidos, que por conterem por natureza um alto grau de estruturação em seus dados, podem gerar valores elevados de entropia. Tal relação é verificada empíricamente na seção 4.1.2.

3.2 Arquitetura Proposta

A arquitetura proposta neste trabalho consiste em um sistema que responda às perguntas feitas na última seção. Isso porque foram realizados testes para analisar a viabilidade a utilização das métricas aqui apresentadas para a determinação de que um fluxo pode ser ilegítimo e carregar informações não autorizadas. Em seções a seguir serão explicados com maior profundidade os testes.

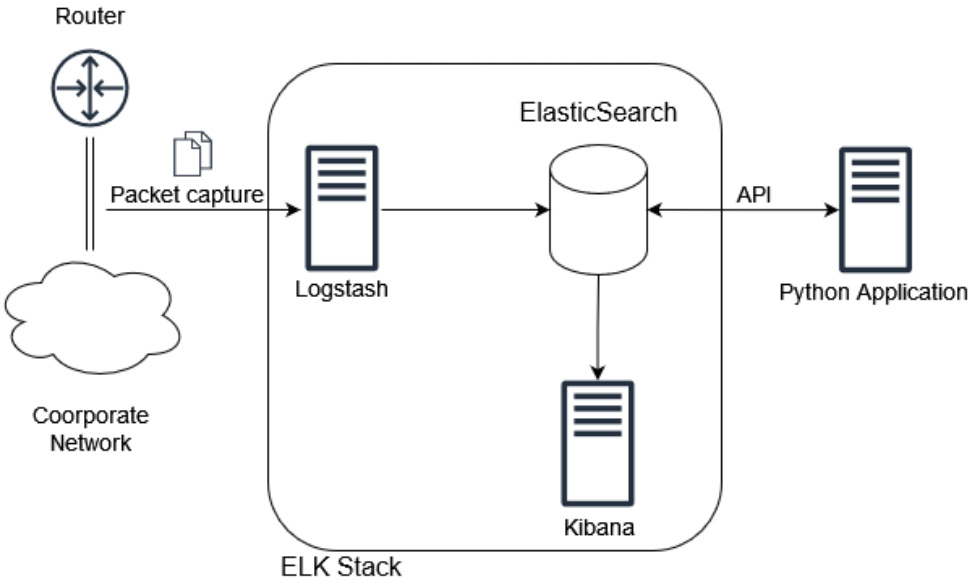


Figura 3.1 – Arquitetura da solução proposta

A figura 3.1 representa a arquitetura proposta neste trabalho como uma visão geral, apontando os componentes ativos que fazem parte do processo de validação. Levando em consideração que o roteador e a rede coorporativa na imagem são genéricas, tal disposição não é necessariamente o padrão para o funcionamento do sistema. A captura de pacotes poderá ser feita em qualquer enlace desejado. Além disso, a captura de pacotes pode ser feita em qualquer ferramenta de preferência, como, por exemplo, *tcpdump*, *wireshark* ou *tshark*. Não importa, desde que esteja no formato *pcap* ou *pcapng*. Dessa forma, é possível fazer a ingestão dos pacotes pelo uso do *tshark* (mais detalhes em 3.2.2) através do logstash (3.2.3 em uma estrutura de dados bem definida chamada *"Newline Delimited JSON (NDJSON)"*. Como define [16], é um formato conveniente para se armazenar ou

transmitir dados estruturados que possam ser processados um por vez (como é o caso dos pacotes). Possui melhor funcionamento em conjunto com ferramentas de textos estilo *unix* e *shell*. Além disso, auxilia no momento de manter logs e de enviar mensagens.

3.2.1 Metodologia

A fim de atingir o objetivo proposto, foi necessário realizar os seguintes métodos:

- Desenvolver métodos de transmissão genéricas de fluxos de pacotes
- Aplicar métodos de captura, extração e filtragem de acordo com a necessidade
- Montar a arquitetura de coleta, ingestão e indexação dos dados
- Implementar métodos de processamento de entropia e geração de gráficos

Os métodos de transmissão foram implementados com um caráter genérico, porém ainda assim, similares aos da realidade. A extração leva em consideração os metadados para que seja possível armazená-los de forma eficiente e por fim, processá-los. A arquitetura de coleta, ingestão e indexação dos pacotes foi realizada de forma estruturada e auditável. Ao mesmo tempo, serviu os dados de maneira fácil e rápida, para que não houvesse latência excessiva. Atingindo os pontos explicitados, foi possível aplicar o método científico e realizar cálculos estatísticos sobre os dados coletados, confirmando ou não a validez da hipótese proposta, levando ao objetivo geral.

3.2.2 Captura e dissecação, *tshark*

O *tshark* é uma ferramenta que trabalha dentro do *wireshark* e em outras ferramentas de captura ou *sniffing*. Por o *tshark* ser uma ferramenta de linha de comando, suas possibilidades de filtragem, edição, captura e modificação são variadas. Neste estudo, ela emprega um papel importante, o de ingerir o arquivo de captura, extrair os metadados e conteúdos, e por fim expelir o NDJSON do pacote para alguma saída. Normalmente essa saída é a *stdout*, saída padrão do *shell*, porém por redirecionamento na linha de comando, envia-se a saída para um diretório específico da máquina local.

A captura e a leitura de arquivos em *pcap* é essencial para qualquer engenheiro, não só de redes, mas qualquer engenheiro na área de TI, principalmente desenvolvedores web, que necessitam ter certeza que suas aplicações funcionam de forma eficiente, aproveitando ao máximo seus recursos de rede sem exaurí-los. Protocolos de redes muitas vezes são abstraídos por bibliotecas e *frameworks* e acabam utilizando de recursos de rede de forma exagerada e gananciosa. Isso pode impactar diretamente a experiência do usuário e a qualidade do serviço ofertado.

A captura manual de pacotes filtrada foi utilizada nesse trabalho a fins experimentais, dessa forma é possível controlar exatamente o que foi ingerido pela aplicação e tirar conclusões apropriadas. Contudo, uma vez que este sistema for migrado para um ambiente mais real, a troca para uma captura e envio contínuo para a ingestão dos sistema pode ser obtida de maneira relativamente fácil. Ao utilizar ferramentas como *'tcpdump'* e ferramentas nativas do linux como *'cronjob'*, que

permite a execução de comandos e binários em certos intervalos de tempo, é possível criar um fluxo contínuo em quase-tempo real para o envio das capturas.

3.2.3 Coleta e Parsing, Indexação, Visualização dos dados, ELK

Uma vez que nossos pacotes foram transformados em uma estrutura de dados bem definida e está salva no disco da máquina, o Logstash, ferramenta *open-source* da Elastic, verifica essa mudança no ambiente e automaticamente ingere esse arquivo estruturado, e de acordo com os filtros e scripts definidos por nós mesmos na implementação do Logstash, esses dados são convertidos e enviados para a indexação.

A indexação por sua vez, é feita pela ferramenta que talvez seja a peça central do quebra-cabeça, o ElasticSearch. O ElasticSearch é um instrumento de pesquisa, também de código aberto, mantido pela Elastic, que tem como base o indexador Lucene. Ele provê funcionalidades de pesquisas rápidas, uma API REST simples, uma boa escalabilidade, dentre outras.

O destino final da análise não é o indexador, ou pelo menos não nessa etapa, porém ele se mostra extremamente importante para o armazenamento dos logs coletados, e também para as consultas que serão feitas posteriormente à coleta. E além disso, enquanto os dados estão indexados no ElasticSearch, a terceira componente da pilha ELK, o Kibana, entra como ferramenta de visualização dos dados. Através de uma GUI, é possível consultar os documentos (amostras indexadas em um índice) e todos os seus dados associados, agregá-los e criar complexos gráficos e tabelas. Dessa forma existe uma tela de monitoramento que já pode ser integrada de forma simples a uma infraestrutura, o que agrega observabilidade ao ambiente. Isso permite que pessoas que ocupem cargos executivos e que não possuem a prática de leitura de pacotes possam entender de forma completa e intuitiva o que está se passando no ambiente. Cada vez mais, empresas utilizam estes métodos de visualização para seus sistemas, seja com o Kibana, ou com ferramentas similires.

Com o Kibana se torna completa a pilha ELK, container representado em volta das três máquinas na figura 3.1. Representação inclusive, que representa a mesma máquina usada no ambiente de testes.

Essa configuração da ELK *stack* é muito utilizada, pois desde que se mostrou muito útil em seus primeiros estágios de desenvolvimento, cresceu com outras tecnologias e se aproveitou de novos nichos. Como, por exemplo, atualmente é possível usar a pilha em forma de container como o Docker, facilitando assim, a sua aplicabilidade, sua escalabilidade e sua imutabilidade, conceitos característicos de Infraestrutura como Código (IaC).

3.2.4 Pós-processamento

O pós-processamento se mostra extremamente importante para a validação desse projeto, já que é nessa etapa onde são calculados os valores de entropia de cada pacote. É a etapa também, onde acontece a reindexação e onde os dados são tratados e testados com testes estatísticos que validam diferenças entre distribuições de fluxos compostos de texto plano e fluxos compostos de texto criptografado.

Ele começa depois que os pacotes passaram por um primeiro processo de indexação, eles já devem estar, inclusive, livres para serem visualizados nos *dashboards* do Kibana. Então, executa-se um arquivo em Python, que pode ser encontrado no apêndice B. Este código recebe como entrada primeiramente o endereço IP do Elasticsearch, aonde ele possa se conectar via API. Em seguida, ele precisa saber qual é o índice daqueles presentes dentre as capturas indexadas que será utilizado. Uma vez em posse do índice, o script realiza o restante da execução, que consiste em extrair das centenas de informações presentes as mais pertinentes para a aplicação. Para que assim, seja possível fazer o cálculo das entropias de cada pacote existente no índice, reestruturá-los e reenviá-los para o ElasticSearch, para que possam ser reindexados. Em seguida, são salvos localmente na forma de CSV para análises futuras.

Cadernos Jupyter foram utilizados para transformar estes dados armazenados em DataFrames, estrutura na qual, torne-se mais simples realizar manipulação e visualização de dados, muito utilizada na ciência de dados. Dessa forma, foi possível se livrar de pacotes "ruidosos", que interfeririam na verdadeira análise da solução. Assim, foram gerados os gráficos presentes na próxima sessão. Mais informações sobre os códigos utilizados podem ser encontrados no apêndice A.

3.2.5 Fluxo dos dados e Descrição da Arquitetura

O fluxo dos dados na arquitetura proposta se dá como é descrito na figura 3.2. Os fluxos enviados são capturados e dissecados de forma que fiquem estruturados no formato NDJSON. Dessa forma, são indexados na pilha ELK de dados e monitoramento, e com isso são gerados gráficos, tabelas e visualizações dos dados. Em seguida, são executados scripts para realizar todo o pós-processamento através da API do ElasticSearch. O pós-processamento engloba todo o processo de cálculo das entropias para cada pacote de cada fluxo, e suas respectivas correlações. Todas as implementações dos diferentes métodos são realizadas nesta *caixa*, como o cálculo por janela deslizante ou o cálculo de uma *string* de tamanho arbitrário, demonstrado em 3.2. Uma vez em posse de novas informações valiosas, é possível reindexá-los e gerar novas possíveis visualizações mais esclarecedoras sobre cada fluxo capturado.

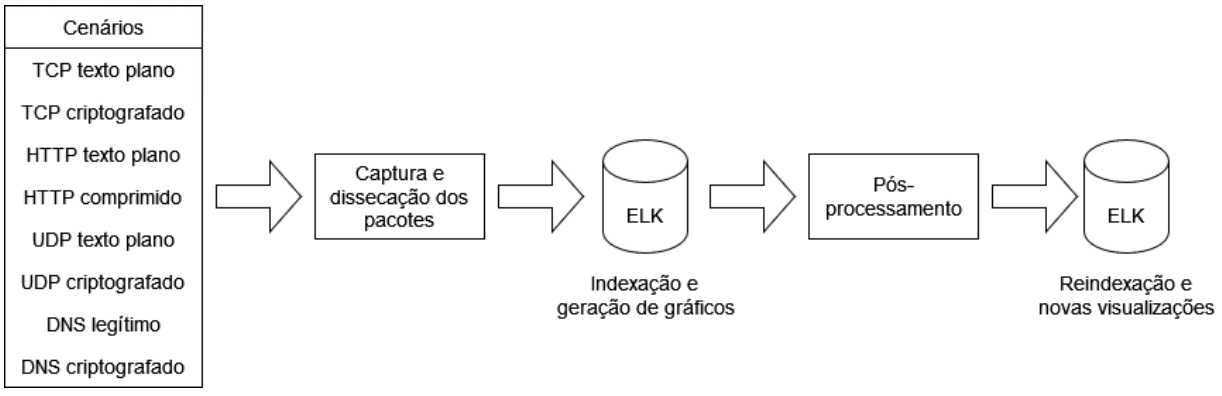


Figura 3.2 – Fluxo dos dados na arquitetura proposta.

Pode parecer inconsistente a indexação dos dados duas vezes, já que costuma ser um processo moroso e custoso. Porém, neste caso foi necessário, já que não existe uma forma de incorporar o código em Python dentro da ELK Stack de forma que o processamento ocorra dentro do container

da figura 3.1. A API REST cumpre um papel importante nesse ponto, já que é possível criar, deletar e atualizar documentos e índices a qualquer momento. Além disso, outros metadados irrelevantes para o cálculo da entropia são descartados durante o pós-processamento. Dados estes, que podem ser relevantes de um ponto de vista forense ao detectar fluxos maliciosos.

4 Resultados e Análises

Resultados e análises são fundamentadas na observação empírica da teoria proposta pelo trabalho. Os resultados aqui apresentados serão em forma de diversos cenários a fim de analisar o método proposto de forma individual. Os tráfegos utilizados foram capturados e orquestrados em um ambiente controlado, fazendo o uso de filtros, *Sniffers* de rede e linguagens de programação para atingir tal finalidade.

Ao implementar a própria aplicação, o desenvolvedor decide o número de bytes que serão utilizados para cada transferência. Pois isso reflete diretamente na responsividade ou tempo de resposta da aplicação. No caso da aplicação desenvolvida para o projeto, só a simples ação de preencher um *buffer* de determinado tamanho (1024 bytes) e enviá-lo, já é suficiente para a finalidade experimental. Aqui, não há a necessidade de enviar pequenos ou médios pacotes, muito menos a necessidade de baixa latência ou tempo rápido de resposta. As quatro aplicações, TCP *plaintext*, TCP *encrypted*, UDP *plaintext* e UDP *encrypted* foram desenvolvidas de forma semelhante à descrita acima. Portanto, o valor 1024 é arbitrário e não carrega nenhum significado mais profundo.

Todos os capturas utilizadas nos cenários foram feitas no *Wireshark*, e os *sockets* foram implementados em Python 3.8. As implementações podem ser encontradas no repositório indicado em A.

4.1 Descrição de cenários

Nesta seção, serão realizados os exérimentos descritos pelos cenários, que servirão como testes para a resolução do problema. Sua resolução é segmentada em cenários para que seja constatado um certo nível de padronização para a resposta do sistema. A melhor forma de se verificar essa padronização é por meio do uso excessivo dos testes, dados e informações.

Será demonstrado à diante, que são realizados dois cenários de testes para diferentes protocolos de transporte utilizados na Internet hoje, assim como dois cenários de testes para diferentes aplicações, também difundidos nos tráfegos da Internet. Dessa forma, primeiramente, poderemos constatar se há diferença na utilização de qualquer um dos dois maiores protocolos de transporte. Além disso, será possível constatar se o sistema será capaz de abstrair a aplicação que roda sobre a camada de transporte e seus cabeçalhos.

4.1.1 Cenário 1 - Aplicação TCP

O primeiro cenário dos resultados consiste na captura dois fluxos TCP que simplesmente enviam pacotes ao preencherem o *buffer* determinado na implementação do *socket*. Portanto, o fluxo é único e só há um início e um término de conexão TCP. Ainda assim, como o TCP faz controle automático da transmissão de pacotes, todos os pacotes que não continham *payload* na camada de transporte foram descartados na análise, ou seja, todos os pacotes de controle de transmissão

TCP.

Foi utilizado um arquivo '.txt' de um texto em inglês suficientemente grande para a análise, a RFC 761, que descreve o funcionamento do protocolo TCP. O arquivo foi escolhido por ter uma extensão em tamanho considerável para gerar uma amostra grande, podendo gerar dados estatísticos consistentes. A língua também fará diferença nos resultados dos testes. Cada língua tem seu nível médio de entropia, e o inglês é uma língua universal e aceita pela comunidade científica.

Neste cenário, os dois fluxos diferem em sua codificação. O primeiro foi transmitido utilizando texto plano, podendo ser lido por qualquer humano que pudesse interceptar esse tráfego. Como é possível verificar na figura 4.1, o tráfego foi reconstituído pela ferramenta *Wireshark* sem a necessidade de senhas ou de decifrar os dados a fim de comprovar o sucesso da transmissão.

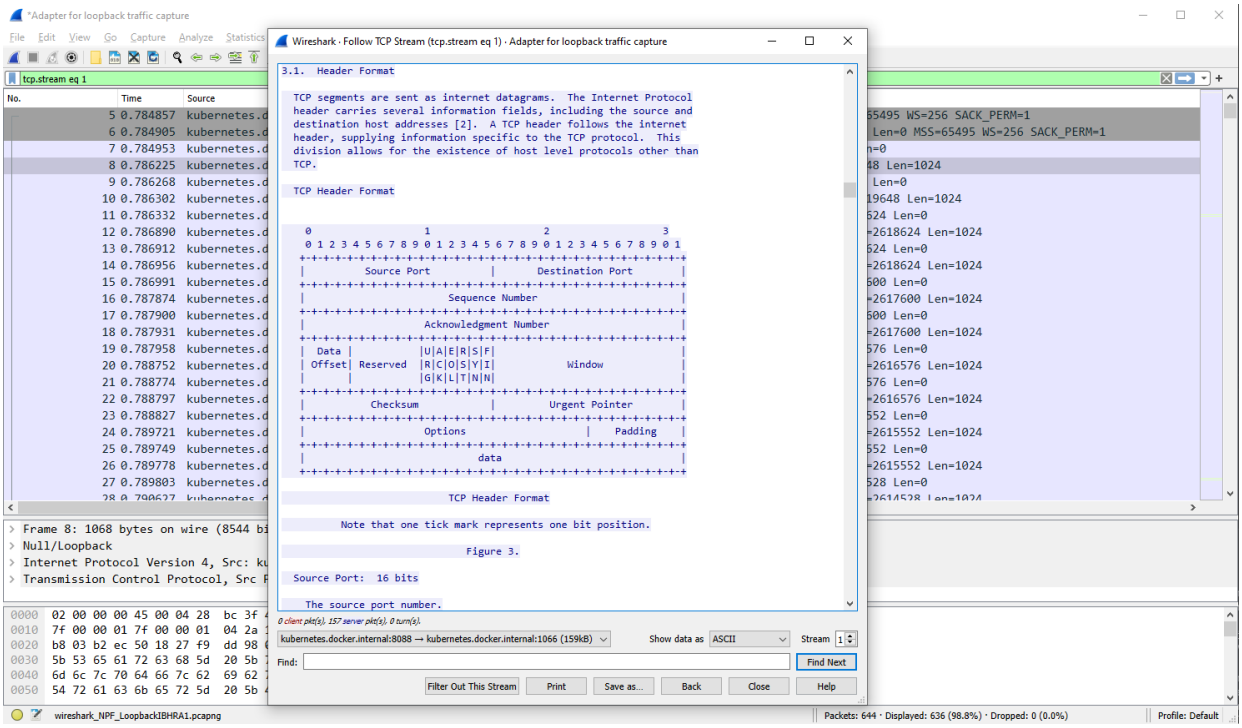


Figura 4.1 – Captura do tráfego em texto plano pelo *Wireshark*.

Já para o segundo fluxo, foi utilizado o mesmo arquivo texto como fonte de informação, porém foi cifrada com AES 256 antes do envio, utilizando-se uma chave aleatória, algoritmo descrito em 2.2 a fim de comprovar a hipótese de que suas entropias resultariam em valores diferentes, já que os caracteres transmitidos seriam completamente embaralhados, e portanto, teriam cada um probabilidades equivalentes de serem utilizados.

Na figura 4.2, foi realizado outro teste para verificar se houve mesmo o envio criptografado dos dados. Como é visto na figura, é praticamente impossível que alguém com acesso a estes pacotes possa decifrá-los sem a chave pré-definida. Uma observação interessante a ser feita para a figura 4.2 é que, quando tenta-se seguir o fluxo da aplicação pelo wireshark utilizando o ASCII, vários pontos aparecem como caracteres, o que dá uma falsa impressão de que este caracter tem uma maior probabilidade de ocorrer. Contudo, o que acontece é que a tabela ASCII comum só possui

128 entradas, das quais nem todas representam símbolos legíveis, o que não é um problema quando tenta-se ler um arquivo que foi feito para ser lido em texto. Mas que apresenta falta de símbolos quando os bytes variam de 0 a 256, e por tal razão são impressos pontos na tela para os bytes que não existem na tabela ou que não possam ser impressos.

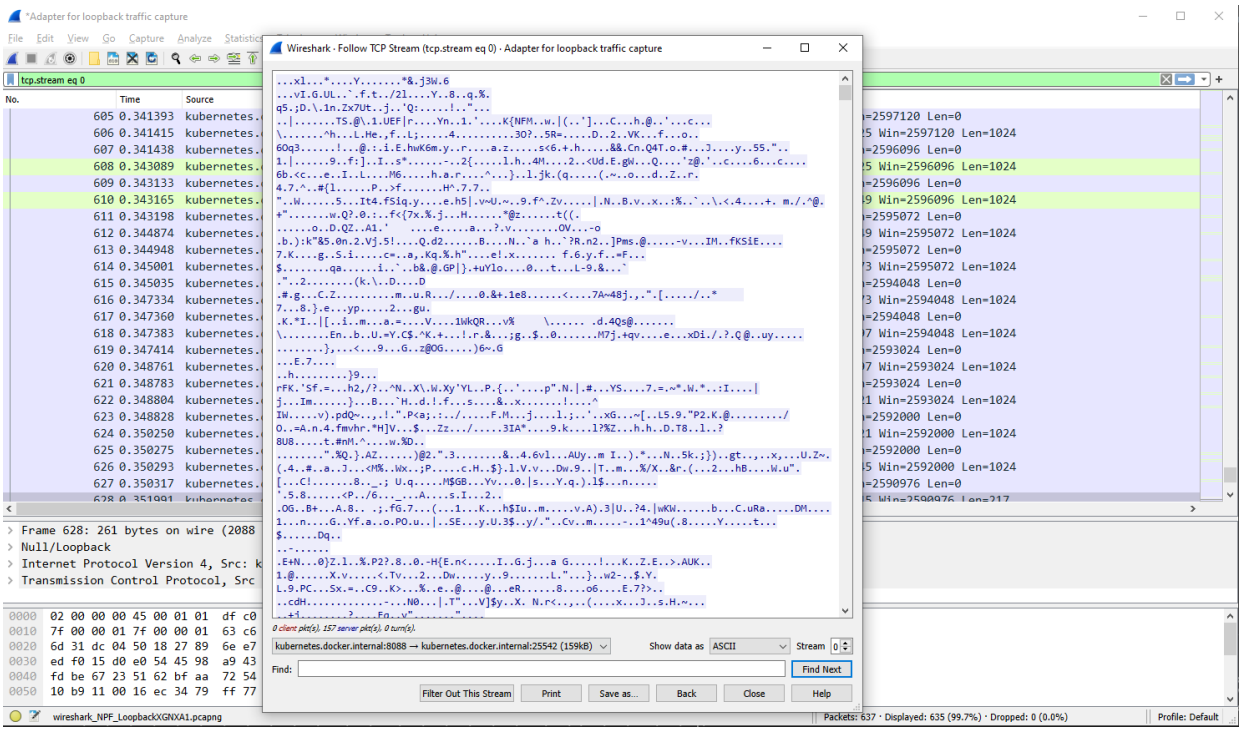
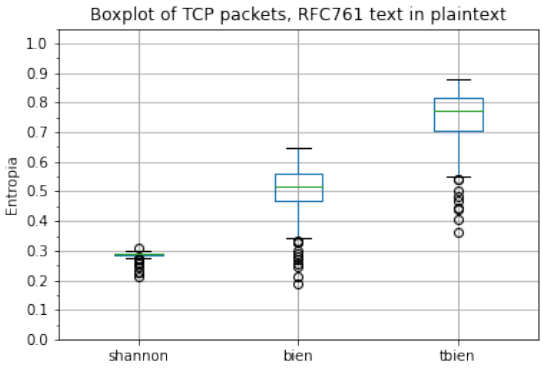


Figura 4.2 – Captura do tráfego criptografado pelo *Wireshark*.

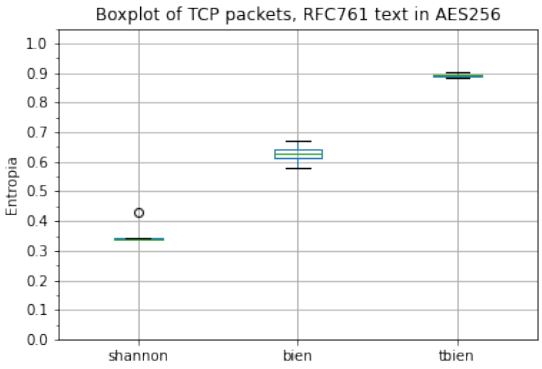
Como descrito no capítulo anterior, uma vez que os pacotes já foram indexados e separados pelos metadados em diferentes campos em uma estrutura de chave-valor, é possível calcular as entropias sob os campos de carga da camada de transporte, que nesse caso é o TCP. Ou seja, para cada pacote, são computadas as entropias, seja por janela deslizante, seja de toda a carga, e então criam-se diferentes distribuições para cada variável dos dois fluxos. Elas foram plotadas em forma de *'boxplot'* no gráfico das figuras 4.3. À esquerda, em 4.3a, é possível verificar a distribuição das amostras para o Fluxo TCP não-criptografado, enquanto que à direita, em 4.3b, vê-se a distribuição das amostras para o Fluxo TCP criptografado.

Um 'boxplot' é uma forma padronizada de se apresentar a distribuição de dados baseada em cinco valores principais: o mínimo, o primeiro quartil (Q1), a mediana, o terceiro quartil (Q3), e o máximo. Além disso, é possível ver *outliers* e seus valores. Esses e outros valores também podem ser encontrados nas tabelas a seguir, para as distribuições em texto plano e em texto criptografado, respectivamente. Como, por exemplo, *mean* é média estatística das distribuições, enquanto que, *std* é o desvio padrão.

Dessa forma, podemos comparar os valores de entropia para cada um dos cenários. Já é nítido antes mesmo da comparação que existe um foco de amostras para tbien próximas a 1 para o tráfego criptografado, de forma que não existem pontos espalhados pelo espectro de entropia de tbien, ou seja, sem *outliers*, assim como bien. É possível observar ainda que, as três distribuições para o



(a) *Boxplot* do *payload* TCP em *plaintext*.



(b) *Boxplot* do *payload* TCP em AES 256.

Figura 4.3 – Ambos os fluxos TCP em termos de entropia

Tabela 4.1 – Resultados para o fluxo TCP genérico em texto plano.

	shannon	bien	tbien
mean	0.28589934351966767	0.5009640383783863	0.7494548108262054
std	0.012748509708130107	0.08720973946196223	0.1042087232510738
min	0.2133439856461027	0.1872620403421369	0.364402459713449
25%	0.2842723093998457	0.4703735000141512	0.7071781941212045
50%	0.288077117198409	0.5170928445524017	0.7728628638643767
75%	0.2918028643786063	0.5595659973988824	0.817883939093314
max	0.3078522651038391	0.64899850224028	0.8797150136627504

Tabela 4.2 – Resultados para o fluxo TCP genérico criptografado.

	shannon	bien	tbien
mean	0.3409630331811487	0.6272585531888561	0.8928553799538232
std	0.007168132529200126	0.01915496095581587	0.0038407382344750047
min	0.337321648822051	0.5790851653485742	0.882875502488567
25%	0.3395254376828379	0.615178636132422	0.8903176153375233
50%	0.3403057564098546	0.6259273676959002	0.8925342466821108
75%	0.3412012875535716	0.6406867077671287	0.8955630021750781
max	0.4288486442758329	0.6701324704104317	0.9031620644834704

fluxo criptografado apresentam baixa variância graficamente.

Além disso, percebe-se que para as três grandezas do fluxo criptografado, as medianas, os máximos e os mínimos estão acima dos seus correpondentes no fluxo em texto plano. Portanto, visivelmente a primeira conclusão é de que há sim uma diferença entre as duas distribuições para todos os pacotes presentes nos fluxos.

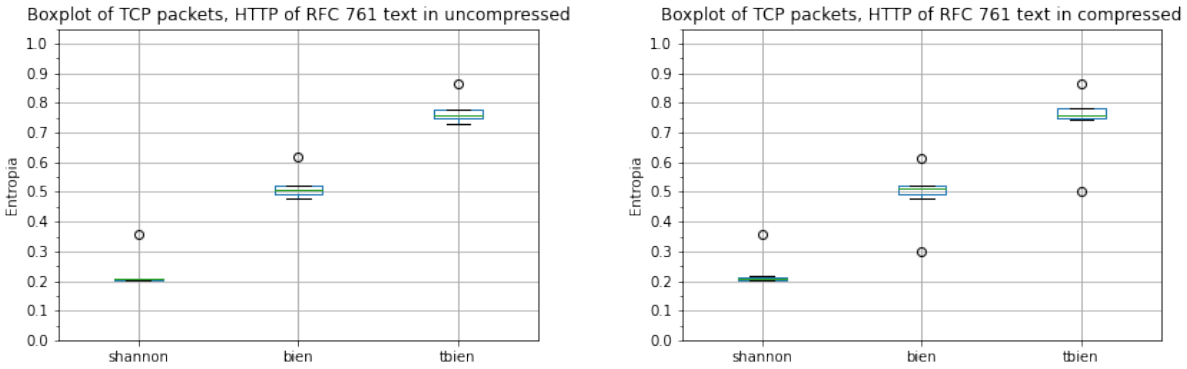
Porém, por mais que os gráficos sejam convincentes, ainda não existe uma confirmação estatística da hipótese proposta. Portanto, foi realizado o teste de 'Shapiro-Wilk' em todas as distribuições, usando $\alpha = 0,05$, para determinar se seguem uma distribuição normal, já que algumas distribuições parecem seguir esse padrão pelo *boxplot* e para determinar qual teste será utilizado em seguida. Nenhuma das distribuições foi confirmada pelo teste como uma distribuição normal. Em seguida, foi utilizado o teste 'Wilcoxon Signed-Rank Test' para verificar se há igualdade entre as diferenças das medianas. Para as três grandezas de cálculo de entropia, a hipótese nula, que

representa uma diferença nula entre as medianas, foi refutada.

4.1.2 Cenário 2 - Fluxo HTTP

Para este cenário, foram capturados, de forma legítima e exclusiva, dois fluxos de uma das aplicações mais comuns na Internet, extremamente empregada no uso das APIs, o HTTP (*Hypertext Transfer Protocol*). Este cenário teve como objetivo a observação do comportamento do protocolo em questão, o HTTP, em relação ao uso de entropia. Os dois fluxos consistiam em duas transmissões mantendo ainda como fonte o texto da página da RFC 761: A primeira, descrito na figura 4.4a, transimitindo somente um objeto HTML do texto por completo e original. E outra, descrita pela figura 4.4b, utilizando o módulo de compressão do *Apache*, descrito pelo protocolo, que permite que o servidor utilize um método que o navegador ou a aplicação similar do cliente suporte. O método aceito e utilizado durante a transmissão foi o *gzip*.

Este teste teve um objetivo bem definido, algo que interfere diretamente, como dito na seção 3.1.2, que é a utilização de algoritmos de compressão que o servidor oferece ao cliente durante o início da comunicação, como determinado pelo IETF em [10].



(a) *Boxplot* do tráfego HTTP sem compressão.

(b) *Boxplot* do tráfego HTTP comprimido.

Figura 4.4 – Ambos os fluxos HTTP em termos de entropia

É importante lembrar também que, o arquivo utilizados na primeira transmissão deste cenário, que apesar de ser texto plano, segue uma linguagem de marcação, o que lhe confere um caráter mais estruturado do que uma linguagem humana, como o inglês.

Ao observar os dois gráficos em 4.4, pode-se perceber que não houve uma diferença significativa entre as distribuições. Além disso, pelas tabelas 4.3 e 4.4, é possível confirmarmos que não há diferenças que suportem a distinção de um texto plano para um texto comprimido.

Tabela 4.3 – Resultados para o fluxo HTTP original.

	shannon	bien	tbien
mean	0.22668450138813104	0.5196020571874913	0.7725317772895501
std	0.0582343376592372	0.047321585077425704	0.04512686794710547
min	0.2006464842695695	0.4766907848779502	0.730958769649727
25%	0.20328043304183574	0.4946688544510033	0.7477083595239781
50%	0.2052610930012222	0.50797888805517	0.7589409731831658
75%	0.20787420521720465	0.5216196673976659	0.7778722001096292
max	0.3585746559280448	0.6199676836819811	0.8666615789267448

Tabela 4.4 – Resultados para o fluxo HTTP comprimido.

	shannon	bien	tbien
mean	0.22587520452980964	0.49440517174476406	0.7427227949301323
std	0.053723980287352986	0.08791207373208508	0.10480260153594163
min	0.2016481469175915	0.2996771570837513	0.501550821618103
25%	0.20468733673386563	0.4944358842106419	0.7496916095477519
50%	0.20623789730484265	0.5099702456213502	0.759938902119963
75%	0.21051790406366014	0.5217327345184273	0.7800794265154087
max	0.3583517386757779	0.6124255202074489	0.8660542155335877

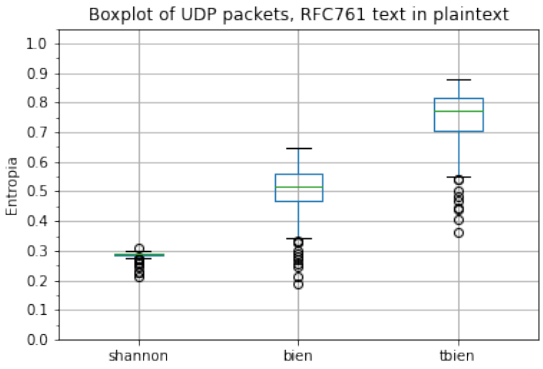
4.1.3 Cenário 3 - Aplicação UDP

O terceiro cenário é análogo ao primeiro cenário, onde tem-se dois fluxos utilizando o mesmo protocolo da camada de transporte, porém que diferem na carga do pacote, onde uma utiliza um conteúdo em texto plano, e a outra, o mesmo conteúdo em texto criptografado. O arquivo de texto utilizado foi o mesmo que o do cenário 1.

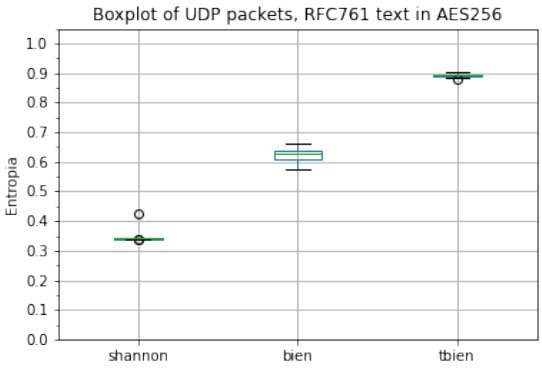
Os fluxo analisados neste cenário foram capturados do começo ao fim e se utilizam do protocolo UDP. O conteúdo da comunicação se resume à simples transmissão dos bytes presentes no arquivo de texto por intermédio de um *buffer* que os libera sempre que atinge sua capacidade. Como o protocolo UDP não dispõe de controle de pacotes, a filtragem feita para os pacotes foi a simples seleção da porta de destino.

O gráficos com os valores das entropias para este cenário se encontram na figura 4.5. À esquerda, em 4.5a, é possível observar o *boxplot* do fluxo UDP não-criptografado. Enquanto que, à direita, em 4.5b, vê-se *boxplot* do fluxo UDP criptografado.

Aqui é possível verificar resultados similares ao primeiro cenário, o que faz sentido, pois foram abstraídos os pacotes de controle de transmissão dos fluxos TCP, faz-se a extração correta do conteúdo do pacote. Os textos foram criptografados utilizando a mesma chave e o conteúdo transmitido é exatamante igual. Incongruências entre os cenários 1 e 3 indicariam alguma falha no processo. Para os resultados dos testes estatísticos, os resultados também tiveram as mesmas conclusões. Os resultados gerados podem ser observados nas tabelas 4.5 e 4.6.



(a) *Boxplot* do *payload* UDP em *plaintext*.



(b) *Boxplot* do *payload* UDP em AES 256.

Figura 4.5 – Ambos os fluxos UDP em termos de entropia

Tabela 4.5 – Resultados para o fluxo UDP genérico em texto plano.

	shannon	bien	tbien
mean	0.2858993435196677	0.5009640383783863	0.7494548108262054
std	0.012748509708130107	0.08720973946196223	0.1042087232510738
min	0.2133439856461027	0.1872620403421369	0.364402459713449
25%	0.2842723093998457	0.4703735000141512	0.7071781941212045
50%	0.288077117198409	0.5170928445524017	0.7728628638643767
75%	0.2918028643786063	0.5595659973988824	0.817883939093314
max	0.3078522651038391	0.64899850224028	0.8797150136627504

Tabela 4.6 – Resultados para o fluxo UDP genérico criptografado.

	shannon	bien	tbien
mean	0.3409895356078522	0.6240100608067853	0.8921855662918359
std	0.006937905054680069	0.01851762524061573	0.003804287099702688
min	0.3371728705868034	0.574230753680473	0.8779685364435982
25%	0.3396894109844364	0.6105820491633421	0.8893912055956427
50%	0.3404668978858552	0.6255584716087524	0.8924500793298364
75%	0.3413158022163255	0.6377253319838487	0.8950055598786943
max	0.4261069456535426	0.6634717290756792	0.9010461591294804

4.1.4 Cenário 4 - DNS

Já no cenário 4, foi utilizado uma aplicação equivalentemente difundida como o HTTP, porém que faz uso do protocolo de transporte UDP, o DNS. Seu gráfico pode ser encontrado na figura 4.6. Em [22] é citado como é possível calcular a entropia dos caracteres em um *payload* DNS a fim de descobrir a aleatoriedade dos nomes utilizados. Ainda na referência, é explicado que, em uma *query* benígna, a maioria das *labels* consistem de palavras de dicionário, enquanto que caracteres codificados em uma *query* maliciosa tendem a ser mais aleatórias, mostrando portanto, um maior nível de entropia. É explicado em [22] que, essa pode não ser a regra para todos os casos, pois *queries* em *Content Delivery Networks (CDNs)* podem apresentar altas entropias, apesar de completamente legítimas.

Em [4], foi realizada uma abordagem mais dependente da detecção de anomalias pela entropia, e são utilizadas alguns tipos diferentes de entropias. Focando em dados finitos e discretos de

elementos, [4] utiliza-se, ou a comparação da entropia de duas distribuições, ou considera-se a entropia relativa entre elas.

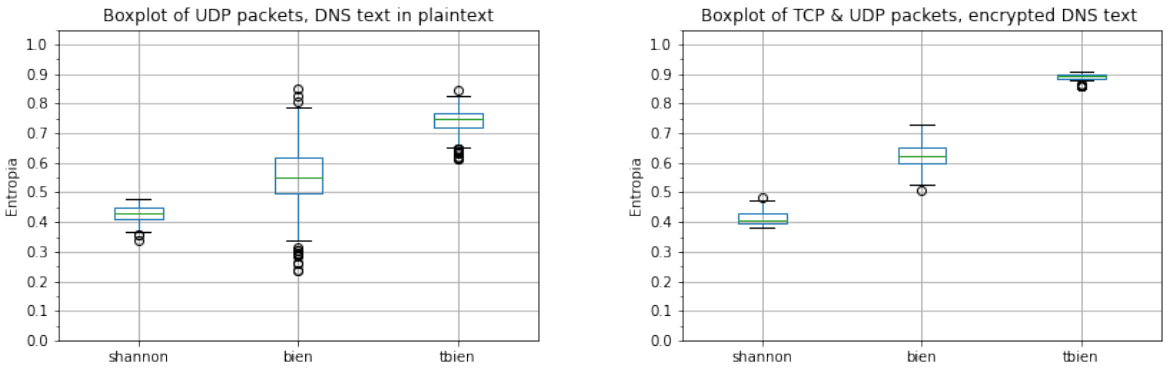
A entropia pode se mostrar uma ferramenta valiosa durante a análise dos dados após a coleta final, portanto é levada em consideração caso seja necessária.

Foi utilizado, portanto, uma ferramenta *open-source* e disponível para criptografar as requisições DNS realizadas sob a utilização de um protocolo não-homologado por qualquer orgão significativo, chamado de *DNSCrypt* [8]. Protocolo este que é diferente do conhecido *DNS over HTTPS* (DoH), bem estabelecido nos maiores servidores DNS responsáveis pela tradução da maior parte do fluxo da Internet. Apesar de não ter sido possível comparar os fluxos baseados em um mesmo arquivo de texto como fonte neste caso, tentou se ao máximo manter os nomes requisitados similares.

Ao analisar as figuras em 4.6, é possível observar que, mais uma vez, a tbien se sobressai em relação às outras. Sua distribuição para a situação de criptografia é mais certeira e varia pouco ao redor da mediana, que tem valor maior do que o próprio máximo da entropia tbien do DNS legítimo.

Já a entropia bien demonstra um achatamento para o DNS criptografado em relação ao seu correspondente legítimo, e sua mediana também é mais elevada. Porém, é nítido que não há uma comparação justa entre bien e tbien para este cenário.

Por último, a entropia de Shannon não teve qualquer resultado positivo para a determinação da criptografia neste caso, inclusive causou confusão com uma mediana menor para o caso do DNS legítimo. É possível conferir os valores e suas discrepâncias nas tabelas 4.7 e 4.8.



(a) *Boxplot* das entropias para requisições DNS legítimas.

(b) *Boxplot* das entropias para requisições DNS utilizando DNSCrypt.

Figura 4.6 – Distribuições antagônicas de DNS em termos de entropia

Tabela 4.7 – Resultados para o fluxo DNS legítimo.

	shannon	bien	tbien
mean	0.4293707964596123	0.5509560033323627	0.7453710657322272
std	0.023283725718624273	0.097507329744652	0.03566470828078016
min	0.3381507811879212	0.2385994848401414	0.6139754547006596
25%	0.4125943085529603	0.4964557242655849	0.7217069678632029
50%	0.4310304259926569	0.5514376246725304	0.7486942116932923
75%	0.4469793819308449	0.616078910442003	0.7689696963285514
max	0.4800013161826775	0.8512550808188706	0.8446159813830804

Tabela 4.8 – Resultados para o fluxo DNS criptografado.

	shannon	bien	tbien
mean	0.41451451791655064	0.6231454283855351	0.888896017075708
std	0.020497329918703767	0.04341528082469843	0.012695590035616601
min	0.3808203542328771	0.5046366397503639	0.8574419123821649
25%	0.395939790178668	0.5992612726487438	0.8856310124332056
50%	0.4077765989934165	0.6235837454152576	0.8925321701119102
75%	0.4293017794578516	0.6507518715844068	0.8967846872087536
max	0.4832780809012839	0.7272715640650157	0.9097584273168832

4.2 Comparativos

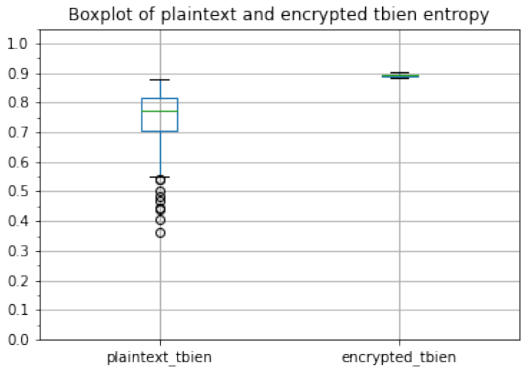
Para fins comparativos, nesta seção são apresentados gráficos que foram gerados relacionando entropias de mesma natureza, porém com cenários opostos. Como, por exemplo, valores de entropia de Shannon para o fluxo TCP descriptografado com fluxo TCP criptografado.

4.2.1 Comparativos dos fluxos TCP genéricos

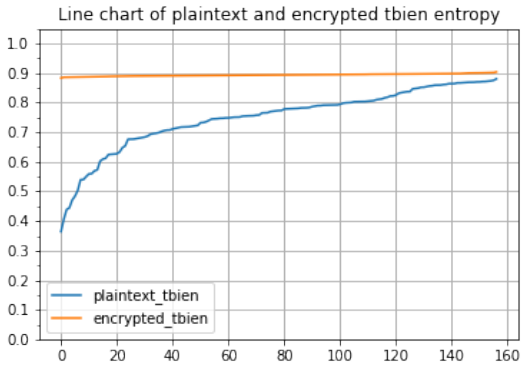
Começando pela entropia de TBien, representado pela figura 4.7a, onde foi percebida a maior diferença entre as duas distribuições, percebe-se que para as amostras provenientes do texto criptografado, não há variância, valor representado pela linha verde na plotagem. Enquanto que, para o texto plano os valores são mais espalhados e mais baixos. Além de existirem vários *outliers* para a distribuição de tbien para texto plano.

Para uma visão mais completa das amostras, foi gerado também o gráfico de comparação para o mesmo cenário em linhas, como pode ser visto na figura 4.7b. Como dito anteriormente, a entropia permanece alta para todos os pacotes do fluxo de forma uniforme. No gráfico, os pacotes foram ordenados em ordem crescente de entropia, assim é possível verificar que existem pacotes que possuem valores altos de entropia. Isso acontece pois, naturalmente existirão amostras que decaem para a distribuição de probabilidade, mas mesmo assim não há uma zona crítica onde as amostras se confundem. Na prática, isso pode significar um pacote onde existiam muitos espaços em branco, ou no caso da RFC uma parte do texto onde foi feita uma representação gráfica com os carecteres.

Ao analisar a entropia bien em 4.8a, ainda existe uma diferença entre as medianas e valores significativos como máximos, mínimos, primero e terceiro quartis. Porém, já existe aqui uma zona crítica de *overlapping* entre as amostras, o que influencia no cálculo do teste de hipótese. Mas ainda sim, pelo teste de Wilcoxon Signed Rank Test há diferença entre as medianas das distribuições.



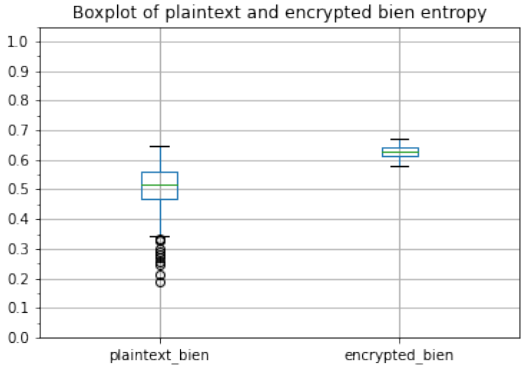
(a) Comparação entre distribuições de tbien em TCP para tráfego em texto plano e criptografado em *boxplot*.



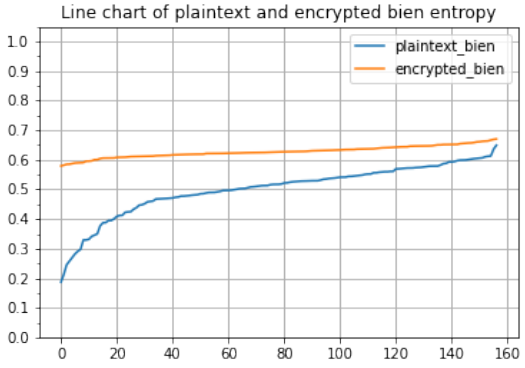
(b) Comparação entre distribuições de tbien em TCP para tráfego em texto plano e criptografado em linha.

Figura 4.7 – Comparativos entre distribuições de entropias TBiEn para fluxos TCP.

Já no gráfico de linha, ainda é notado o comportamento mais distribuído para os valores da entropia, demonstrando maior variância nos dados.



(a) Comparação entre distribuições de bien em TCP para tráfego em texto plano e criptografado em *boxplot*.



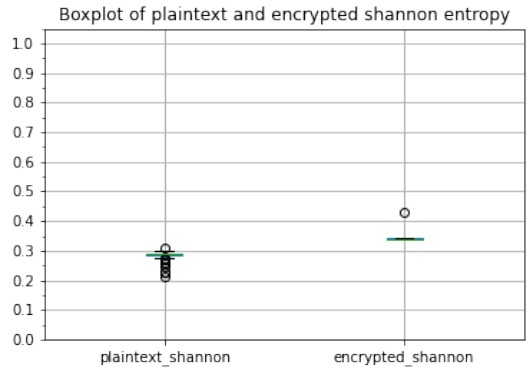
(b) Comparação entre distribuições de bien em TCP para tráfego em texto plano e criptografado em linha.

Figura 4.8 – Comparativos entre distribuições de entropias BiEn para fluxos TCP.

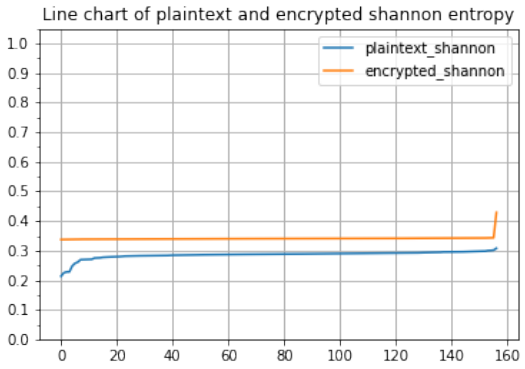
Na entropia de Shannon, é onde pode-se observar a menor distinção entre as duas distribuições em linha. Não há um crescimento significativo da linha para o tráfego em texto plano, exceto pelos *outliers*. Esse comportamento demonstra que o indicador não é a melhor detecção de tráfegos criptografados, já que os testes aqui realizados foram baseados em fluxos. Ou seja, o fluxo como um todo traz a informação completa, diferentemente de como foi realizado em [5] e [28].

4.2.2 Comparativos dos fluxos UDP genéricos

Já para o cenário dos fluxos UDP genéricos, é possível observar um comportamento similar. Começando pela entropia que melhor demonstrou as distinções entre os cenários em texto plano e criptografados, a tbien. Como era de se esperar, ainda há grandes distinções entre as duas dis-



(a) Comparação entre distribuições de Shannon em TCP para tráfego em texto plano e criptografado em *boxplot*.

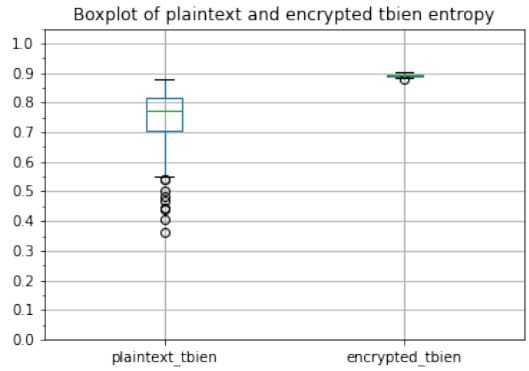


(b) Comparação entre distribuições de Shannon em TCP para tráfego em texto plano e criptografado em linha.

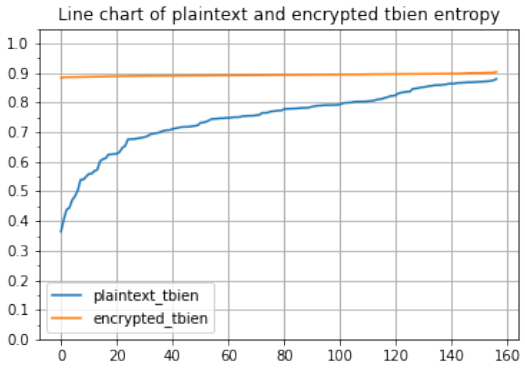
Figura 4.9 – Comparativos entre distribuições de entropias de Shannon para fluxos TCP.

tribuições, representadas pelos gráficos em 4.10. Afinal, como os pacotes foram filtrados a ponto de só sobrar aqueles que continham realmente o conteúdo transmitido, e o dados foram retirados do payload da camada de transporte, era de se esperar que houvesse uma similaridade entre os cenários.

Na figura 4.10a, é possível observar uma concentração da distribuição das amostras criptografadas em torno da mediana, que têm um valor mais elevado que o valor máximo da distribuição das amostras em texto plano. Ao mudar o foco para 4.10b, no gráfico de linha, fica mais claro ainda uma distribuição mais instável entre as amostras, de forma que os valores variam desde baixas entropias, até valores mais próximos do que é considerado esperado para a distribuição criptografada.



(a) Comparação entre distribuições de TBien em UDP para tráfego em texto plano e criptografado em *boxplot*.

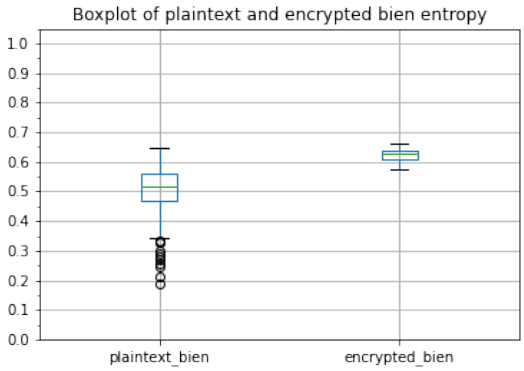


(b) Comparação entre distribuições de TBien em UDP para tráfego em texto plano e criptografado em linha.

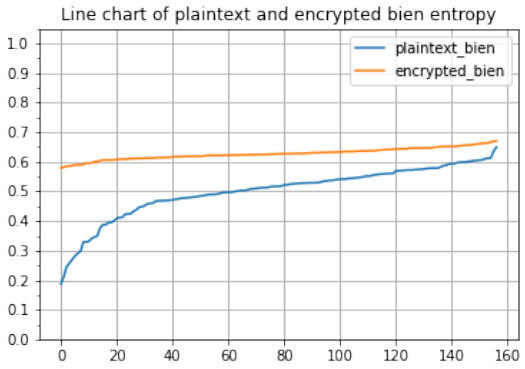
Figura 4.10 – Comparativos entre distribuições de entropias TBiEn para fluxos UDP.

Para as entropias Bien, descritas pelas figuras em 4.11, observa-se um comportamento similar ao de tbien em 4.10, porém de forma menos acentuada, o que não significa que não é possível

fazer a distinção estatística das duas distribuições. Pois, de acordo com os testes realizados, neste comparativo ainda é constatada uma falta de semelhança entre as medianas pelo teste de Wilcoxon.



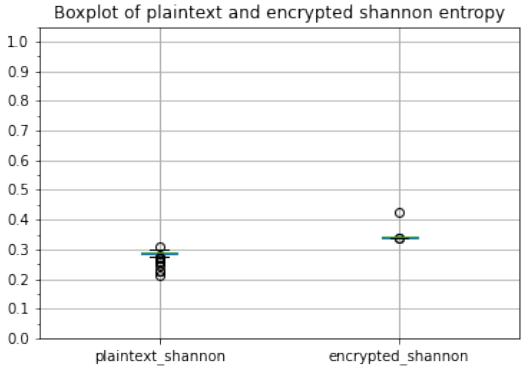
(a) Comparação entre distribuições de Bien em UDP para tráfego em texto plano e criptografado em *boxplot*.



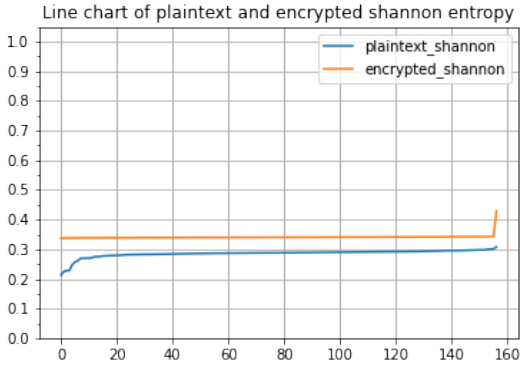
(b) Comparação entre distribuições de Bien em UDP para tráfego em texto plano e criptografado em linha.

Figura 4.11 – Comparativos entre distribuições de entropias BiEn para fluxos UDP.

Quando é realizado o comparativo entre as distribuições de entropia de Shannon, constata-se maior dificuldade em distinguir o comportamento entre elas. Apesar de o teste de Wilcoxon demonstrar ainda a distinção necessária para classificar as amostras, percebe-se pelo comportamento da distribuição em texto plano em 4.12b que a maioria dos pacotes têm baixa variabilidade em relação à mediana. Este resultado apresenta uma incerteza em relação à grandeza de Shannon para a finalidade de distinção do tráfego em texto plano do criptografado. Ainda que seja importante ser coletada sempre que possível, já que é um escalar importante para a Teoria da Informação e pode atribuir informações para outras finalidades.



(a) Comparação entre distribuições de Shannon em UDP para tráfego em texto plano e criptografado em *boxplot*.



(b) Comparação entre distribuições de Shannon em UDP para tráfego em texto plano e criptografado em linha.

Figura 4.12 – Comparativos entre distribuições de entropias de Shannon para fluxos UDP.

5 Conclusão e Trabalhos Futuros

Ao final da análise, levando em consideração que os resultados foram positivos, já que para todos os testes estatísticos, com exceção do tráfego comprimido, houveram distinções entre as distribuições comparadas, pode se dizer que a diferenciação e classificação de tráfegos em texto plano e criptografados é possível dentro do escopo analisado. A diferença em termos de valores parece pequena, mas ainda é distinguível. Portanto, é possível alertar administradores de redes, desenvolvedores e técnicos da existência de tráfego possivelmente criptografado quando não houve um processo padronizado pelos protocolos comumente utilizados. Isso pode indicar uma possível fraude, estrutura de comando e controle, exfiltração de dados, dentre outras práticas maliciosas como explicado na seção 2.

Para os cenários estabelecidos, realizados à partir das implementações de uma aplicação genérica, pode se dizer que o objetivo geral foi alcançado e que o sistema proposto pode oferecer mais uma barreira de segurança para possíveis atacantes em qualquer rede. Além disso, esses resultados podem ser obtidos em uma rede empresarial sem causar instabilidade ou *bottleneck* no tráfego. Não há *downtime* em aplicações ou perda de pacotes, pois não existe a necessidade de processamento *on-the-fly*, apenas o espelhamento do que já está trafegando na rede. De uma outra perspectiva, um pacote que foi detectado como possível fraude por criptografia quando não deveria haver, não poderá ser parado, mas servirá como embasamento para que o seu fluxo seja detectado e possivelmente bloqueado.

Alertas poderão ser criados a partir dos resultados destes testes, e dependerão assim, ou de uma intervenção humana para o bloqueio, ou poderá ser criado algum processo que fique em execução, responsável pela criação de regras de bloqueio em um eventual *firewall*. Ademais, trabalhos futuros podem continuar melhorando a detecção do tráfego e melhor aferindo se um conteúdo é realmente criptografado através de um classificador baseado em *Machine Learning* que possa utilizar a aplicação detectada ou não, para ter maior eficácia.

Para que o sistema pudesse ficar cada vez mais robusto e acurado, seria essencial a criação de um banco de dados que pudesse receber constantemente valores de entropia para cada aplicação de tráfegos comuns na Internet. Tais fluxos poderiam ser provenientes de várias amostras de arquivos *.pcap* ou *.pcapng*, que são constantemente compartilhados em sites como o do *wireshark*, ou mesmo dos usuários do sistema. Dessa forma, seria possível ter uma base comparativa mais fidedígna aos valores normais para cada aplicação, sendo mais fácil classificá-las. Outra implementação futura interessante, seria fazer o uso da linguagem P4 como fonte para criar um ambiente SDN de alta performance, deixando verificações como as propostas neste trabalho mais performáticas e em tempo real.

Bibliografia

[1] Jawad Ahmed et al. "Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks". Em: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 649–653.

[2] *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. [Online; acessado 04-Novembro-2021]. URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.

[3] Laurent Bernaille e Renata Teixeira. "Early Recognition of Encrypted Applications". Em: *Passive and Active Network Measurement*. Ed. por Steve Uhlig, Konstantina Papagiannaki e Olivier Bonaventure. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 165–175. ISBN: 978-3-540-71617-4.

[4] Christian Callegari, Stefano Giordano e Michele Pagano. "Entropy-based network anomaly Detection". Em: *2017 International Conference on Computing, Networking and Communications (ICNC)*. 2017, pp. 334–340. DOI: <10.1109/ICCNC.2017.7876150>.

[5] Fran Casino, Kim-Kwang Raymond Choo e Constantinos Patsakis. "HEDGE: Efficient Traffic Classification of Encrypted and Compressed Packets". Em: *IEEE Transactions on Information Forensics and Security* 14.11 (2019), pp. 2916–2926. DOI: <10.1109/TIFS.2019.2911156>.

[6] Weiting Chen et al. "Measuring complexity using FuzzyEn, ApEn, and SampEn". Em: *Medical Engineering Physics* 31.1 (2009), pp. 61–68. ISSN: 1350-4533. DOI: <https://doi.org/10.1016/j.medengphy.2008.04.005>. URL: <https://www.sciencedirect.com/science/article/pii/S1350453308000726>.

[7] Grenville J. Croll. "BiEntropy - The Approximate Entropy of a Finite Binary String". Em: *CoRR* abs/1305.0954 (2013). arXiv: <1305.0954>. URL: <http://arxiv.org/abs/1305.0954>.

[8] *DNSCrypt*. [Online; acessado 04-Novembro-2021]. URL: <https://www.dnscrypt.org/>.

[9] *ExFILD: a tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic*. [Online; acessado 04-Novembro-2021]. URL: <http://udspace.udel.edu/handle/19716/5838>.

[10] Roy T. Fielding, Mark Nottingham e Julian Reschke. *HTTP Semantics*. Internet-Draft draft-ietf-httpbis-semantics-19. Work in Progress. Internet Engineering Task Force, set. de 2021. 252 pp. URL: <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[11] Norberto Garcia et al. "Distributed real-time SlowDoS attacks detection over encrypted traffic using Artificial Intelligence". Em: *Journal of Network and Computer Applications* 173 (2021), p. 102871. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2020.102871>. URL: <https://www.sciencedirect.com/science/article/pii/S1084804520303362>.

[12] Huaqing Lin, Gao Liu e Zheng Yan. "Detection of Application-Layer Tunnels with Rules and Machine Learning". Em: *Security, Privacy, and Anonymity in Computation, Communication, and Storage*. Ed. por Guojun Wang et al. Cham: Springer International Publishing, 2019, pp. 441–455. ISBN: 978-3-030-24907-6.

[13] Reham El-Maghraby, Nada Mostafa e Ayman Bahaa-Eldin. "A survey on deep packet inspection". Em: dez. de 2017, pp. 188–197. DOI: <10.1109/ICCES.2017.8275301>.

[14] Mohammad Saiful Islam Mamun, Ali A. Ghorbani e Natalia Stakhanova. "An Entropy Based Encrypted Traffic Classifier". Em: *Information and Communications Security*. Ed. por Sihan Qing et al. Cham: Springer International Publishing, 2016, pp. 282–294. ISBN: 978-3-319-29814-6.

[15] Fares Meghdouri, Félix Iglesias Vázquez e Tanja Zseby. "Cross-Layer Profiling of Encrypted Network Data for Anomaly Detection". Em: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 2020, pp. 469–478. DOI: <10.1109/DSAA49011.2020.00061>.

[16] *NDJSON*. [Online; acessado 04-Novembro-2021]. URL: <ndjson.org>.

[17] *OpenSSL*. [Online; acessado 04-Novembro-2021]. URL: <https://www.openssl.org/>.

[18] Tamara Radivilova et al. "Decrypting SSL/TLS traffic for hidden threats detection". Em: *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. 2018, pp. 143–146. DOI: <10.1109/DESSERT.2018.8409116>.

[19] *RFC 6335*. [Online; acessado 04-Novembro-2021]. URL: <https://tools.ietf.org/html/rfc6335>.

[20] K. Roebuck. *Deep Packet Inspection: High-impact Strategies - What You Need to Know*. Amazon Distribution, 2011. ISBN: 9781743049921. URL: <https://books.google.com.br/books?id=Jh64pwAACAAJ>.

[21] Christian Rossow e Christian J. Dietrich. "ProVeX: Detecting Botnets with Encrypted Command and Control Channels". Em: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. por Konrad Rieck, Patrick Stewin e Jean-Pierre Seifert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 21–40. ISBN: 978-3-642-39235-1.

[22] Saeed Shafieian, Daniel Smith e Mohammad Zulkernine. "Detecting DNS Tunneling Using Ensemble Learning". Em: *Network and System Security*. Ed. por Zheng Yan et al. Cham: Springer International Publishing, 2017, pp. 112–127. ISBN: 978-3-319-64701-2.

[23] C. E. Shannon. "A mathematical theory of communication". Em: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: <10.1002/j.1538-7305.1948.tb01338.x>.

[24] S. S. Shapiro e M. B. Wilk. "An Analysis of Variance Test for Normality (Complete Samples)". Em: *Biometrika* 52.3/4 (1965), pp. 591–611. ISSN: 00063444. URL: <http://www.jstor.org/stable/2333709>.

[25] *The Transport Layer Security (TLS) Protocol Version 1.3*. [Online; acessado 04-Novembro-2021]. URL: <https://datatracker.ietf.org/doc/html/rfc8446>.

[26] *tshark ELK VM.* [Online; acessado 04-Novembro-2021]. URL: <https://github.com/H21lab/tsharkVM>.

[27] Faheem Ullah et al. "Data exfiltration: A review of external attack vectors and countermeasures". Em: *Journal of Network and Computer Applications* 101 (2018), pp. 18–54. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2017.10.016>. URL: <https://www.sciencedirect.com/science/article/pii/S1084804517303569>.

[28] Yipeng Wang et al. "Using Entropy to Classify Traffic More Deeply". Em: *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*. 2011, pp. 45–52. DOI: <10.1109/NAS.2011.18>.

[29] R. F. Woolson. "Wilcoxon Signed-Rank Test". Em: *Wiley Encyclopedia of Clinical Trials*. American Cancer Society, 2008, pp. 1–3. ISBN: 9780471462422. DOI: <https://doi.org/10.1002/9780471462422.eoct979>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780471462422.eoct979>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9780471462422.eoct979>.

[30] Kun Zhou et al. "Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks". Em: *ETRI Journal* 42.3 (2020), pp. 311–323. DOI: <https://doi.org/10.4218/etrij.2019-0190>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.4218/etrij.2019-0190>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.2019-0190>.

ANEXO A – Repositório e Dependências

O repositório git para o projeto se encontra no seguinte link:

<https://github.com/joaoalencastro/pythonSocket>

Para poder rodar o sistema como um todo, será necessário primeiramente criar uma máquina virtual que receba a ELK Stack. Para tal, foi utilizado um projeto como base para a máquina virtual com a ELK Stack, chamado de tshark ELK VM appliance [26]. Algumas configurações deverão ser alteradas para que funcione corretamente.

Já para a execução dos códigos em Python, segue relação de quais bibliotecas serão necessárias localmente. Incluíndo os 'jupyter notebooks' que fazem a análise posterior ao pós-processamento, ao ler os arquivos CSV e criar DataFrames, Plots e Tabelas que nos permitiram fazer uma análise mais profunda.

Listing A.1 – Lib requirements for entire Python runtime.

```
elasticsearch==7.14.0
bitstring==3.1.9
pandas==1.2.3
BiEntropy==1.1.4
pycryptodome==3.11.0
matplotlib==3.3.4
matplotlib-inline==0.1.3
numpy==1.20.1
scipy==1.6.1
```

ANEXO B – Código

Listing B.1 – Script de pós-processamento responsável pelo cálculo das entropias, reindexação e geração de CSVs.

```
# -*- coding: utf-8 -*-

from elasticsearch import Elasticsearch
from bientropy import bien, tbien
from bitstring import Bits
from math import log2
from pandas import DataFrame

def connect_elasticsearch():
  host = '192.168.0.138' #input('Please, enter elastic host: ')

  es = None
  es = Elasticsearch([{'host':host, 'port': 9200}])
  if es.ping():
    print('Yay Connected')
  else:
    print('Awww it could not connect!')
  return es

def get_packet_indices(es):
  packets_indices = []
  for index in es.indices.get('*'):
    if "packets-" in index:
      packets_indices.append(index)
  return packets_indices

def get_packets_from_index(es, index):
  # Me retorna uma lista de dicionarios, onde cada dicion rio    um pacote
  response = es.search(index=index, body={"size":10000,"query":{"match_all":{}}},
    scroll='1s')
  print("Querying {} packets from index '{}'...".format(response["hits"]["total"]
    ["value"], index))
  return response["hits"]["hits"]

def get_ids_from_index(es, index):
  response = es.search(index=index, body={"query":{"match_all":{}},"stored_fields": []})
  print("========== IDs ==========")
  print(response)

def is_udp(packet):
  if "udp_raw" in packet["_source"]["layers"]:
    return packet["_source"]["layers"]["udp_raw"]
  else:
    return None
```

```
def get_transport_protocol(packet):
  if "udp" in packet["_source"]["layers"]:
    return "udp"
  elif "tls" in packet["_source"]["layers"]:
    return "tls"
  elif "tcp" in packet["_source"]["layers"]:
    return "tcp"
  else:
    return None

def get_payload(protocol, packet):
  # Returns the payload
  if protocol == 'udp':
    if "dns_raw" in packet["_source"]["layers"]:
      payload = packet["_source"]["layers"]["dns_raw"]
    else:
      payload = packet["_source"]["layers"]["data_raw"]
    pass
  elif protocol == 'tcp' and "tcp_tcp_payload" in packet["_source"]["layers"]["tcp"]:
    payload = packet["_source"]["layers"]["tcp"]["tcp_tcp_payload"]
    payload = payload.split(':')
    payload = ''.join(payload)
  elif protocol == 'tls' and "tls_tls_app_data" in packet["_source"]["layers"]["tls"]:
    payload = packet["_source"]["layers"]["tls"]["tls_tls_app_data"]
    payload = payload.split(':')
    payload = ''.join(payload)
  else:
    payload = None
  return payload

def get_ports(protocol, packet):
  ports = []
  if protocol == 'udp':
    ports.append(packet['_source']['layers']['udp']['udp_udp_srcport'])
    ports.append(packet['_source']['layers']['udp']['udp_udp_dstport'])
  else:
    ports.append(packet['_source']['layers']['tcp']['tcp_tcp_srcport'])
    ports.append(packet['_source']['layers']['tcp']['tcp_tcp_dstport'])
  return ports

def get_payload_size(protocol, packet):
  # Returns the payload size of the Transport Layer
  if protocol == 'udp':
    size = packet["_source"]["layers"]["udp"]["udp_udp_length"]
  elif protocol == 'tcp' or protocol == 'tls' and "tcp_tcp_payload"
    in packet["_source"]["layers"]["tcp"]:
    size = packet["_source"]["layers"]["tcp"]["tcp_tcp_len"]
  else:
    size = None
  return size
```

```
def calculate_shannon_entropy(string):
    """
    Calculates the standardized Shannon entropy for the given string

    :param string: String to parse.
    :type string: str

    :returns: Shannon entropy (min bits per hex-character).
    :rtype: float
    """
    ent = 0.0
    alphabet = ['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f']
    if len(string) < 2:
      return ent
    size = float(len(string))
    for b in range(15): # Para usar ASCII, troque para 127
      freq = string.count(alphabet[b])
      if freq > 0:
        freq = float(freq) / size
        ent = ent + freq * log2(freq)
    return -ent/log2(size)

def calculate_bien(string):
  return bien(Bits(string))

def calculate_tbien(string):
  return tbien(Bits(string))

def write_df_to_csv(df, file):
  df.to_csv('Outputs/'+file, index=False)

def truncate_hex(data, hex):
  '''
  Make it divisible by number of hex symbols provided
  This only works for hex strings
  '''
  if len(data) < hex:
    return None # not big enough
  while (len(data) % hex != 0):
      data = data[:-1]
  return data

def sliding_window(data, test, windowsize=7):
  '''
    Receives the data in hex and the type of test it has to work
    (probably entropy) with and returns the mean/average of all windows
    of data entropy
    n is the size of the sliding window in bytes
    Returns the average of the vector formed by the results of the tests
    made in each chunk of string
  '''
  sum = 0
```

```
  if test != 'shannon' and test != 'bien' and test != 'tbien':
    print("Non existing type of test or null data.")
    return None
  data = truncate_hex(data, windowsize)
  # this will make the hex raw data be divisible by 8 hex
  # symbols so we can use sliding window
  if data is None:
    return None
  elif len(data) == 0:
    return None
  else: pass
  windowsnum = int(len(data)/float(windowsize))
  for i in range(1, windowsnum + 1):
    hex_data = '0x' + data[(i - 1) * windowsize : i * windowsize]
    if test == 'bien':
      sum += float(calculate_bien(hex_data))
    else:
      sum += float(calculate_tbien(hex_data))
  avg = sum/windowsnum
  return avg

if __name__ == '__main__':
  es = connect_elasticsearch()

  indices = get_packet_indices(es)
  print(indices)
  index = input("Chose from above which index to process: ")

  # packets is a list of dictionaries, with each dictinary being a packet
  packets = get_packets_from_index(es, index)

  # index_metadata will be the list of dictionaries of the processed data
  records = []

  for packet in packets:
    # Collect packet's intel
    tp = get_transport_protocol(packet)
    raw_data = get_payload(tp, packet)
    ports = get_ports(tp, packet)
    payload_size = get_payload_size(tp, packet)

    if raw_data != None and payload_size != None and 'ip' in packet['_source']['layers']:
      new_doc = {}

      new_doc['id'] = packet['_id']
      new_doc['proto'] = tp
      new_doc['srcip'] = packet['_source']['layers']['ip']['ip_ip_src']
      new_doc['dstip'] = packet['_source']['layers']['ip']['ip_ip_dst']
      new_doc['srcport'] = ports[0]
      new_doc['dstport'] = ports[1]
      new_doc['payload_size'] = payload_size
      new_doc['shannon'] = calculate_shannon_entropy(raw_data)
```

```
    new_doc['bien'] = sliding_window(raw_data, 'bien')
    new_doc['tbien'] = sliding_window(raw_data, 'tbien')

    records.append(new_doc)

index_metadata = DataFrame.from_records(records)

request_body = {
  'mappings': {
    'properties': {
      'id': {'type': 'keyword'},
      'proto': {'type': 'keyword'},
      'srcip': { 'type': 'keyword'},
      'dstip': {'type': 'keyword'},
      'srcport': {'type': 'keyword'},
      'dstport': {'type': 'keyword'},
      'payload_size' : {'type': 'long'},
      'shannon': {'type': 'long'},
      'bien': {'type': 'long'},
      'tbien': {'type': 'long'}
    }
  }
}
new_index = index+'-processed'
print("creating {} index...".format(new_index))

# Before writing data in Elastic, write data to a csv
write_df_to_csv(index_metadata, new_index)

# Now, create index pattern on Elastic
es.indices.create(index=new_index, body = request_body)

# preparing data to be sent to elastic
bulk_data = []

for index, row in index_metadata.iterrows():
  data_dict = {}
  for j in range(len(row)):
      data_dict[index_metadata.columns[j]] = row[j]
  op_dict = {
      "index": {
          "_index": new_index,
          "_type": '_doc',
          "_id": data_dict['id']
      }
  }
  bulk_data.append(op_dict)
  bulk_data.append(data_dict)

print(bulk_data)
res = es.bulk(index = new_index, body = bulk_data)
```

```
# check data is in there, and structure in there
#es.search(index = new_index, body={"query": {"match_all": {}}})
#es_indices.get_mapping(index = new_index)
```