MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

ALESSANDRA DA SILVA DIAS MALIZIA
LEONARDO GOMES GONÇALVES

ÉGIDE: UMA FERRAMENTA PARA DETECÇÃO DE DESFIGURAÇÕES DE DOMÍNIO

RIO DE JANEIRO
2021

ALESSANDRA DA SILVA DIAS MALIZIA
LEONARDO GOMES GONÇALVES

ÉGIDE: UMA FERRAMENTA PARA DETECÇÃO DE DESFIGURAÇÕES DE DOMÍNIO

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(es): Anderson Pereira Fernandes dos Santos, D.Sc.
Narcélio Rodrigues de Medeiros,

Rio de Janeiro

2021

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ CEP: 22290-270

da Silva Dias Malizia, Alessandra; Gomes Gonçalves, Leonardo.
Égide: uma ferramenta para detecção de desfigurações de domínio / Alessandra da Silva Dias Malizia e Leonardo Gomes Gonçalves. – Rio de Janeiro, 2021.
47 f.

Orientador(es): Anderson Pereira Fernandes dos Santos e Narcélio Rodrigues de Medeiros.

Projeto de Final de Curso (graduação) – Instituto Militar de Engenharia, Engenharia de Computação, 2021.

1. desfiguração. 2. pichação. 3. defesa cibernética. i. Pereira Fernandes dos Santos, Anderson (orient.) ii. Rodrigues de Medeiros, Narcélio (orient.) iii. Título

ALESSANDRA DA SILVA DIAS MALIZIA
LEONARDO GOMES GONÇALVES

Égide: uma ferramenta para detecção de desfigurações de domínio

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(es): Anderson Pereira Fernandes dos Santos e Narcélio Rodrigues de Medeiros.

Aprovado em Rio de Janeiro, 20 de outubro de 2021, pela seguinte banca examinadora:

Prof. Anderson Pereira Fernandes dos Santos - D.Sc.

Prof. Narcélio Rodrigues de Medeiros

Prof. Luiz Henrique da Costa Araújo - D. Sc.

Leandro de Mattos Ferreira

Prof. Leandro de Mattos Ferreira - M. Sc.

Rio de Janeiro
2021

Este trabalho é dedicado a todos os professores que participaram da nossa formação e nos ajudaram a chegar até aqui.

AGRADECIMENTOS

Agradecemos aos nossos amigos e familiares, que nos motivaram, incentivaram e compartilharam conosco cada passo dessa caminhada. Aos nossos colegas de turma, que nos apoiaram durante o curso e estavam presentes nos momentos mais difíceis.

Aos nossos orientadores Cap Narcélio e Cel Anderson pela disponibilidade, atenção e contribuições ao longo do desenvolvimento deste trabalho. Agradecemos a todos os professores que participaram da nossa formação e a todos que contribuíram para a nossa graduação no Instituto Militar de Engenharia.

"O homem não é nada além daquilo que a educação faz dele."
(Immanuel Kant)

RESUMO

A evolução da Internet e a expansão do ciberespaço contribuíram para o aumento no número de ataques cibernéticos. A desfiguração, um dos tipos de ataque, é caracterizada pela mudança maliciosa do conteúdo de um sítio. Ela pode causar sérios danos à imagem de organizações e aos seus usuários, constituindo uma ameaça para o espaço cibernético de órgãos públicos, privados e das forças armadas. Este trabalho propõe uma ferramenta para detectar a ocorrência de desfigurações em domínios selecionados. Foi desenvolvido o sistema Égide, capaz de encontrar recursivamente as referências no domínio, de detectar e de classificar mudanças em seu conteúdo, notificando o administrador caso uma desfiguração seja encontrada. Nele, foram utilizados os métodos de *hash*, a detecção de assinaturas e o reconhecimento ótico de caracteres para encontrar mudanças no conteúdo de documentos HTML e classificá-las como uma desfiguração.

Palavras-chave: desfiguração. pichação. defesa cibernética.

ABSTRACT

Evolution of the Internet and the expansion of cyberspace contributed to raising the number of cyberattacks. Defacement, one of the different types of cyberattacks, can be explained as a malicious change made to a page's content. This can inflict serious harm on the image of organizations and their public, threatening the cyberspace of public and private organizations, and also the armed forces. This project presents a tool to detect defacements in selected domains. The Égide system was developed, capable of finding recursively all the references within a page, detecting changes made to their content, classifying those changes as dafacements and notifying an administrator of all detected occurrences. In this system, the methods used to detect changes to the HTML document and classify them as defacements were hashing, attack signatures detection and optical character recognition.

Keywords: defacement. cyber secutiry. attack signatures.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BID	Base Industrial de Defesa
CDCiber	Centro de Defesa Cibernética
CMS	Sistema de Gerenciamento de Conteúdo
ComDCiber	Comando de Defesa Cibernética
DCT	Transformada Discreta de Cosseno
DOM	*Document Object Model*
END	Estratégia Nacional de Defesa
KNN	K-ésimo Vizinho mais Próximo
OCR	Reconhecimento Ótico de Caracteres
PND	Política Nacional de Defesa
SIEM	Gerenciamento e Correlação de Eventos de Segurança
SMTP	*Simple Mail Transfer Protocol*
SOC	Centro de Operações de Segurança
SVM	Máquinas de Vetores e Suporte
SERPO	Serviço Federal de Processamento de Dados
TLS	*Transport Layer Security*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Na sociedade brasileira e em todo o mundo, a vida moderna passou a ser caracterizada por um ambiente virtual e sem fronteiras, o espaço cibernético. Ele pode ser entendido como o conjunto de pessoas, empresas, equipamentos e interconexões dos sistemas de informação e das informações que por eles trafegam (JUNIOR, 2010).

A evolução da *Internet* e a expansão do ciberespaço contribuíram para o aumento no número de ataques cibernéticos: no último trimestre de 2020, foram registrados mais de 4,5 bilhões de ataques no Brasil (LABS, 2020). Entre os diversos tipos de ataque, pode-se destacar a pichação ou desfiguração de domínios (*defacement*), que consiste na modificação maliciosa do conteúdo de um sítio da Internet. Neste tipo de ataque, o criminoso substitui o conteúdo original da página, normalmente a página principal do domínio, por outro de sua autoria, que pode conter textos e imagens maliciosos, com mensagens disruptivas ou algum tipo de assinatura.

A desfiguração de domínio difere de outros tipos de ataques cibernéticos por dois motivos principais. Ele é mais visível e é facilmente identificado, já que muitas vezes o objetivo do atacante é divulgar o seu feito publicamente em redes sociais e expor a vulnerabilidade do site. Além disso, é um ataque comumente usado para exibir mensagens de cunho político ou transgressivas, através da infraestrutura de empresas e organizações que não devem ser associadas a este tipo de mensagem (COOKS; OLIVER, 2014).

Neste sentido, outra caraterística dos ataques por pichação é a ação de grupos de ciberativismo, ou hacktivismo. O hacktivismo tem suas origens em meados da década de 1990 e pode ser definido como o uso de ferramentas digitais com fins políticos, que pode ser feito de maneira transgressiva ou disruptiva (MACHADO, 2015).

A sequência de ações para evitar e reagir à ocorrência de um ataque de desfiguração pode ser dividida em três etapas: prevenção, detecção e reação. A primeira consiste na proteção dos sistemas de informação, como Sistemas de Gerenciamento de Conteúdo (CMS), e dos servidores e *proxies*, a segunda, na correta detecção de mudanças maliciosas, minimizando a ocorrência de falsos positivos como mudanças legítimas de conteúdo, e a terceira consiste na emissão de alertas, inspeção e mitigação do ataque. Sistemas de informação como um Centro de Operações de Segurança (SOC) e ferramentas de Gerenciamento e Correlação de Eventos de Segurança (SIEM) podem ser necessários na última etapa, que deve estar sempre em funcionamento (BERGADANO et al., 2019).

1.1 Motivação

A mudança maliciosa do conteúdo de um sítio pode causar sérios danos à imagem de organizações e aos seus usuários. A prevenção contra ataques de desfiguração, assim como a rápida detecção e correção dos mesmos, é de extrema importância para a garantia da integridade e credibilidade dessas organizações. Apenas no ano de 2020, foram registrados quase 500.000 sites desfigurados e, nos últimos 8 anos, ocorreram em média mais de 800.000 desfigurações por ano (ZONEH, 2020).

Nesse contexto, a pichação constitui uma ameaça para o espaço cibernético de órgãos públicos, privados e das forças armadas. No meio civil, a segurança cibernética refere-se à proteção e garantia da utilização de ativos de informação e estratégicos, como sistemas que controlam infraestruturas críticas nacionais. Abrange, por exemplo, a interação com órgãos da administração pública federal. Já no ambiente militar, a defesa cibernética diz respeito ao conjunto de ações defensivas, exploratórias e ofensivas, com o objetivo de proteger os sistemas de informação, obter dados para a produção de conhecimento de inteligência e causar prejuízos aos sistemas de informação do oponente (CRUZ, 2020).

Ainda no contexto militar, a exploração de sistemas de informação e a defesa contra ataques cibernéticos podem levar à superioridade no campo de batalha. De acordo com PARKS; DUGGAN, guerra cibernética é o subconjunto da guerra da informação que envolve ações realizadas no mundo cibernético. Apesar da separação entre os mundos cibernético (ou virtual) e cinético (ou real), as ações adotadas no mundo cibernético afetam o mundo cinético, e vice-versa (DUTRA, 2007).

Ressaltando a importância da defesa cibernética no Brasil, a Estratégia Nacional de Defesa (END), lançada em 2008 e revista em 2012, estabelece o setor como um dos três essenciais para a Defesa Nacional, juntamente com os setores nuclear e espacial (BRASIL, 2008). A END também atribui a responsabilidade do setor cibernético ao Exército e estabelece como prioridade para o país o fomento à pesquisa científica voltada para esse tema (SCHMITT, 2015).

1.2 Objetivo

O objetivo deste trabalho é a implementação de um *software* capaz de detectar a ocorrência de ataques por desfiguração em domínios previamente selecionados. A ferramenta será dividida em três módulos principais e um módulo auxiliar. O três primeiros consistem em: um módulo para a busca por referências em um domínio, um para a detecção de mudanças no conteúdo da página e um para a classificação de mudanças maliciosas, características de pichações. O último componente é um módulo auxiliar para notificar os administradores do domínio sobre as ocorrências encontradas.

O sistema deverá percorrer o domínio recursivamente, analisando seus ativos estáticos como páginas, referências, textos e imagens de forma eficiente. Quando uma mudança for detectada e classificada como maliciosa, a ferramenta deverá rapidamente alertar os administradores para que o ataque seja revertido o mais breve possível, minimizando os danos da pichação. O sistema proposto será implementado e testado, levantando métricas de eficiência, como o tempo de execução e a taxa de acerto das classificações.

1.3 Justificativa

O desenvolvimento da infraestrutura de defesa e segurança do espaço cibernético brasileiro é de extrema relevância para a Defesa Nacional. O planejamento de ações destinadas à Defesa Nacional tem seu documento condicionante de mais alto nível na Política Nacional de Defesa (PND). Esta, por sua vez, tem seus objetivos alcançados por meio da Estratégia Nacional de Defesa, que trata da reorganização e reorientação das Forças Armadas, da organização da Base Industrial de Defesa (BID) e da política de composição dos efetivos da Marinha, do Exército e da Aeronáutica (BRASIL, 2008).

Com a aprovação da Política Nacional de Defesa e da Estratégia Nacional de Defesa pelo decreto no 6.703, de 18 de dezembro de 2008, também foram criadas OMs para atuar na defesa e proteção dos ativos de informação do Ministério da Defesa e das Forças Armadas, como o Comando de Defesa Cibernética (ComDCiber) e o Centro de Defesa Cibernética (CDCiber), em Brasília. Um exemplo da atuação do CDCiber foi a parceria firmada com o Serviço Federal de Processamento de Dados (SERPO) (CRUZ, 2020).

O tema deste projeto foi proposto pelo CDCiber e está alinhado aos interesses do setor cibernético nacional. O projeto é uma demanda do CDCiber, em parceria com o Instituto Militar de Engenharia, e propõe uma ferramenta recursiva e eficiente para a detecção de desfiguração de domínios, contribuindo para a pesquisa científica no setor cibernético brasileiro e para a segurança do espaço cibernético do país.

A ferramenta proposta poderá ser usada para monitorar de maneira eficiente o ataque por pichação em domínios de organizações públicas e privadas. A rápida detecção e notificação do ataque permitirá a correção da desfiguração e a minimização de danos para a integridade das organizações.

1.4 Metodologia

Inicialmente, foi feito o estudo sobre os ataques de desfiguração de domínio e suas principais técnicas de detecção, sobre o impacto desses ataques em organizações e sobre a importância do tema para a defesa cibernética nacional. Além disso, foi feito o

levantamento bibliográfico dos principais trabalhos relacionados já publicados, viabilizando a discussão sobre as características desejadas deste projeto e o detalhamento das mesmas.

Em seguida, foi feita a modelagem da ferramenta, detalhando o planejamento de implementação e o funcionamento de cada um de seus quatro módulos. O sistema implementado foi testado em diferentes domínios e em exemplos de desfigurações extraídos da *Internet*. Foi verificado seu tempo de execução e os acertos na classificação dos ataques.

1.5 Estrutura do Texto

Este projeto está estruturado em cinco capítulos, além das referências bibliográficas. Após a Introdução, o Capítulo 2 discorre sobre o estado da arte da área de estudo, apresentando os principais conceitos e trabalhos relacionados. O Capítulo 3 detalha a modelagem proposta a ser implementada no projeto e suas principais características. Já o Capítulo 4 apresenta os resultados obtidos da implementação, os testes e descreve o funcionamento da ferramenta. O Capítulo 5 expõe a conclusão do projeto, propondo melhorias em trabalhos futuros.

2 ESTADO DA ARTE

De acordo com (SHANNON, 1948), um sistema de comunicação é composto por cinco partes essenciais: a fonte de informação, que produz a mensagem a ser enviada, o transmissor, que transforma a mensagem em um sinal no canal de comunicação, o canal em si, por onde o sinal é transmitido e pode receber ruídos, o receptor, que reconstrói a mensagem a partir do sinal, e o destinatário, a quem a mensagem deve ser enviada. Os componentes são representados na Figura 1.

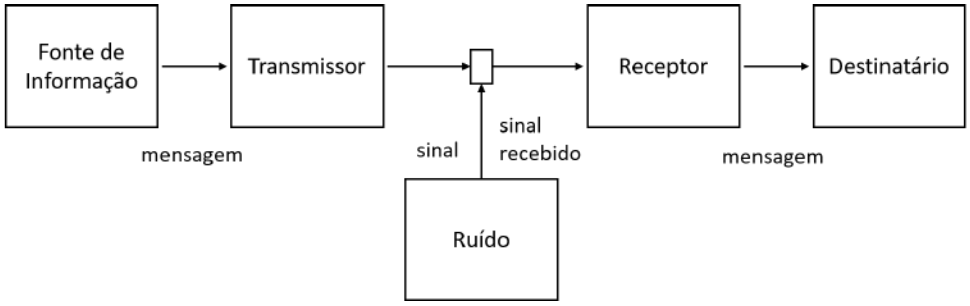

Figura 1 – Diagrama esquemático de um sistema de comunicação, baseada em (SHANNON, 1948).

Nestes sistemas de comunicação, a segurança da informação é aplicada baseada em cinco pilares: confidencialidade, integridade, autenticidade, irretratabilidade e disponibilidade. A disponibilidade garante que a informação poderá sempre ser acessada quando necessária, a irretratabilidade, que um emissor não possa negar o envio de uma mensagem que já emitiu, a autenticidade, que o emissor seja quem o receptor acredita que é, a integridade, que a informação não seja modificada sem o conhecimento do receptor e a confidencialidade garante que o acesso à informação seja feito apenas por pessoas autorizadas.

Nesse contexto, a desfiguração de domínios é um ataque que afeta o pilar da integridade, pois modifica o conteúdo de páginas da *Internet* sem o conhecimento do administrador. Além disso, um ataque pode afetar também o pilar da disponibilidade, já que é capaz de impossibilitar o acesso dos usuários à página original, comprometendo as informações e funcionalidades do sítio. Os atacantes exploram vulnerabilidades desses sistemas e podem ter diversas motivações, como objetivos políticos, econômicos e pessoais.

2.1 Recursividade

Para detectar ataques por desfiguração em um domínio e também nas páginas a que faz referência, pode ser usada uma busca recursiva. A recursividade não é necessária

para a detecção de desfiguração de um domínio e não é empregada frequentemente nos trabalhos relacionados, recebendo-se normalmente uma lista de páginas ou apenas uma delas, na qual será feita a detecção.

A aplicação da recursividade pode ser feita de maneira a percorrer todas as páginas e referências listadas no domínio inicial recursivamente, de modo a realizar a procura e detecção de uma possível desfiguração em todos os sítios pertencentes ao escopo do domínio definido inicialmente.

Dentre as diversas maneiras de se realizar uma recursão dessas, é importante ressaltar a existência de ferramentas no mercado que seriam capazes de realizar uma recursão de sites por força bruta, como o DirBuster(OWASP, 2021), e de ferramentas avançadas de extração capazes de extrair todo tipo de informação de uma página web, usando inclusive artifícios de inteligência artificial.

Para o escopo do trabalho proposto, essa recursão pode ser feita de modos mais simples, como por exemplo utilizando-se o lynx, em comandos linux, que é um navegador baseado em texto capaz de fazer a busca por referências em uma página. Ainda pode ser usada recursivamente através de APIs (HACKERTARGET, 2021) para se obter essa lista, o que poderia implicar em uma menor eficiência do processo. Por último, a recursão também pode ser realizada através de uma biblioteca em *Python* chamada *lxml* (BEHNEL, 2021), com um foco em extrair dados de arquivos HTML e XML, podendo facilmente navegar e analisar todo o conteúdo de uma página, identificando redirecionamentos e aplicando a recursão a partir desses domínios.

2.2 Detecção de Mudanças

Existem diversas técnicas e ferramentas propostas e implementadas comercialmente para a detecção de desfiguração de domínio (KUMAR, 2020). Uma das formas mais simples de detectar pichações é através da detecção de mudanças no conteúdo das páginas monitoradas.

Nesta abordagem, uma versão original do conteúdo é armazenado localmente e comparado periodicamente com a página monitorada, emitindo notificações sempre que qualquer diferença é encontrada (HOANG; NGUYEN, 2019). A maior parte das ferramentas comerciais para detecção de desfiguração, como o Nagios, Site 24x7 e WebOrizon são baseados em técnicas de comparação direta do histórico das páginas e podem gerar uma alta taxa de falsos positivos.

Uma das formas de comparação pode ser feita usando a ferramenta diff, disponível em ambientes Linux. O comando diff do Linux permite a comparação entre dois arquivos linha a linha (KERRISK, 2020). Dessa forma, um documento HTML é comparado perio-

dicamente com uma versão padrão previamente armazenada, alertando quando houver mudanças no arquivo. Assim como as outras técnicas de comparação, essa abordagem é eficiente em páginas estáticas, mas gera uma grande quantidade de alarmes falsos para páginas com conteúdo dinâmico (HOANG; NGUYEN, 2019).

Uma alternativa para detectar mudanças em documentos HTML é através da comparação da estrutura lógica da página, utilizando a API *Document Object Model* (DOM). O DOM é um padrão da World Wide Web Consortium (W3C) para estruturas de documentos em árvore (CONSORTIUM, 2020), e a análise dessa árvore permite identificar mudanças na estrutura de sítios da *Internet*, de forma indiferente a seu conteúdo. Nessa abordagem, a estrutura DOM é extraída do conteúdo da página original e armazenada. Essa árvore é então periodicamente comparada com uma extraída da versão atual da página e são emitidos alarmes quando há mudanças estruturais (HOANG; NGUYEN, 2019).

Por fim, outra técnica baseada no monitoramento de mudanças muito utilizada para a detecção de desfigurações de domínio, principalmente pelas ferramentas comerciais existentes, são as funções de hash. Uma função hash é um algoritmo que mapeia dados de comprimento variável em um comprimento fixo, sendo uma transformação de uma grande quantidade de dados em uma pequena quantidade de informações e tendo uma ampla escala de aplicabilidade e escalabilidade em diversas áreas da computação.

Nesta abordagem, a verificação em soma (*checksum*) com algoritmos de hash, como MD5 ou SHA1, é coletada e armazenada em um perfil. Em seguida, a página é monitorada e são geradas, periodicamente, novas verificações de soma com o conteúdo da página, que são então comparadas com o perfil armazenado. Se os valores não coincidirem, é então gerado um alerta. A técnica de detecção apenas pelo uso de função hash funciona bem para páginas estáticas, mas não é aplicável em páginas dinâmicas, como comércios e publicidade, onde o conteúdo muda com frequência (HOANG; NGUYEN, 2019).

Essas técnicas tradicionais de função hash, como o MD5 e o SHA (1-6), são eficientes em garantir a integridade de dados, mas não é factível para informações de multimídia, que enfatizam mudanças na percepção da retenção da informação, ao invés de apenas a mudança de bits. Além disso, as funções tradicionais de hash são muito sensíveis a mudanças de dados, isto é, a mudança de um bit da entrada provoca mudança significativa na saída da função. Uma função hash de imagem, deveria ser robusta a mudanças poucas de brilho, contraste, correção gamma, compressão JPEG e ajustes de escala, por exemplo.

Dentre as principais funções de hash de imagens, temos o hash médio, de blocos, diferencial, da mediana, perceptivo, de ondaleta. Um estudo em (BLOCKCHAIN, 2020) testou em uma grande base de imagens a eficiência de cada uma dessas funções para mudanças de brilho, contraste, marca d'água, conversão da imagem em tons de cinza, mudança de escala e compressão JPEG. A função de hash perceptivo, a qual utiliza

transformada discreta de cosseno (DCT), apresentou os melhores resultados no geral, sendo ainda mais rápida que as demais.

2.3 Classificação de Mudanças

Além dos métodos de detecção de mudanças e comparações simples, é possível classificar as técnicas de detecção de desfiguração entre abordagens baseadas na detecção de assinaturas e na detecção de anomalias. A primeira é mais eficiente para detectar ataques já conhecidos, enquanto a segunda é capaz de construir um perfil das páginas monitoradas e detectar novos tipos de ataque, quando o perfil lido é classificado como uma anomalia (HOANG; NGUYEN, 2019).

2.3.1 Detecção de Assinaturas

As técnicas de detecção de assinaturas usam um conjunto de regras previamente definidas a partir de ataques conhecidos de pichação para buscar páginas desfiguradas. O conjunto de assinaturas é extraído manualmente de ataques que já aconteceram e pode conter padrões de texto, como o nome do atacante ou regras para a página como: conter o fundo preto, conter uma única imagem, não conter imagens, não conter nenhuma *tag* HTML ou não conter nenhum texto (BARTOLI; DAVANZO; MEDVET, 2010). Uma tabela com exemplos de assinaturas extraídas de páginas desfiguradas, apresentada em (HOANG; NGUYEN, 2019), é ilustrada abaixo.

Tabela 1 – Amostra de assinaturas extraídas de pichações.

Assinaturas
By Cyberpunks
XrillZed004
Mr.PlugIn
MrMoonz
ABD3LOS
GO1B 1D1OT
./LoliSecID
HACKED BY STUDENTS
IND CYBER ARMY
Mr.Joker366
Cyb3rCl4y

Nesta abordagem, o conjunto já definido de assinaturas é buscado na página monitorada, e um alarme é emitido sempre que um padrão for encontrado. Esta técnica possui grande eficiência na detecção de ataques conhecidos e funciona bem em páginas

dinâmicas e estáticas, mas não é capaz de detectar novos tipos de ataque (HOANG; NGUYEN, 2019).

2.3.2 Detecção de Anomalias

A detecção de anomalias é uma técnica utilizada em vários campos de dados, como na saúde, detecção de fraudes, processamento de imagens e detecção de intrusão. Esta abordagem pode ser entendida como a definição de uma região de comportamento normal dos dados e a classificação de anomalias como sendo os pontos que não pertencem a essas regiões (BERGADANO FABIO CARRETO; RAGNO, 2014). Um exemplo de anomalias é ilustrado na Figura 2. Na figura, N1 e N2 são regiões de comportamento padrão, enquanto O1, O2 e O3 são anomalias.

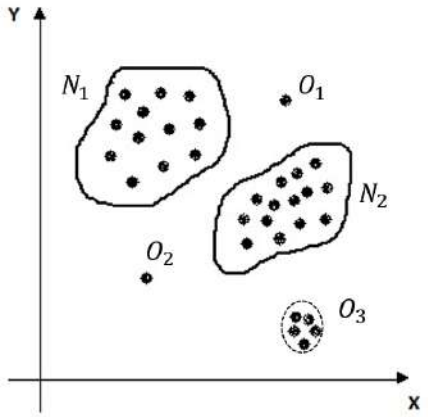

Figura 2 – Ilustração de anomalias, baseada em (BERGADANO FABIO CARRETO; RAGNO, 2014).

Apesar de ter uma definição simples, o uso da detecção de anomalias para encontrar desfigurações de domínio é um problema complexo e possui algumas dificuldades: o comportamento normal dos dados pode não ser estático e mudar ao longo do tempo; as fronteiras entre o comportamento normal e anômalo podem não estar bem definidas, dependendo do domínio; e adversários passivos podem determinar o comportamento normal e encontrar comportamentos maliciosos que se encaixem nesta região. O último ponto será tratado separadamente na próxima seção.

Os métodos de detecção de anomalias podem ser divididos nos grupos abaixo.

- Técnicas de classificação: consistem na construção de modelos de classificação através do aprendizado a partir de um conjunto de dados de treino e utilizam algoritmos como redes neurais, redes Bayesianas e máquinas de vetores de suporte (SVM);

- Técnicas baseadas nas K menores distâncias (KNN): consistem na classificação de pontos anômalos baseado em sua distância a seu k-ésimo vizinho, ou a todos os outros pontos;
- Técnicas de clusterização: consistem no agrupamento dos dados em categorias (clusters). Um ponto é classificado como anomalia se não pertencer a um dos grupos;
- Técnicas baseadas na teoria da informação: utilizam medidas de informação do conjunto de dados, como a complexidade de Kolmogorov e a entropia. Um ponto é considerado uma anomalia se mudar a quantidade de informação do conjunto de dados; e
- Técnicas baseadas em estatística: se baseiam na modelagem matemática de um conjunto de dados, normalmente estocástico. Um ponto é considerado uma anomalia se ocorrer em uma região de baixa probabilidade do modelo.

Diversas características (*features*) dos documentos HTML podem ser usadas em funções de similaridade para definir uma região de comportamento normal. A função mais simples é a contagem, que pode medir características como: a quantidade de caracteres por *tag* do documento; a quantidade de *tags* por tipo, como por exemplo elementos de definição de bloco, elementos de tabela e elementos de título; dimensões, como o tamanho em bytes do documento, o tamanho médio dos blocos de texto, o número de linhas e o comprimento do texto; atributos do texto, como a quantidade de mudanças de letras maiúsculas para minúsculas e vice-versa; e a quantidade de itens como imagens, tabelas, referências e formulários (BARTOLI; DAVANZO; MEDVET, 2010).

A partir da contagem de uma determinada característica do documento, pode-se calcular a sua média e o consequente desvio de uma observação em relação à mesma, seguindo a abordagem estatística. Similarmente, pode-se tomar a frequência relativa dos itens da página. Neste caso, a frequência poderia ser medida, por exemplo, de cada um dos caracteres ASCII em relação a todas as suas ocorrências nas páginas já observadas, ou de cada elemento HTML (BARTOLI; DAVANZO; MEDVET, 2010). Outras medidas, como a distância Kolmogorov e a entropia de Shannon, baseadas na quantidade de informação, poderiam ser feitas de forma análoga (BORGOLTE; KRUEGEL; VIGNA,).

Além disso, também é possível utilizar a máxima árvore comum entre as observações armazenadas da página monitorada, calculando-a a partir da estrutura HTML extraída por uma árvore DOM de cada observação. Após a determinação da árvore comum, uma nova observação poderá ser classificada como anomalia caso não tenha a árvore como subárvore de sua estrutura (BARTOLI; DAVANZO; MEDVET, 2010).

2.3.3 Adversários Passivos

Adversários passivos são aqueles que podem monitorar, interceptar e analisar cada etapa de um sistema de segurança da informação (BAI et al., 2008). Em um sistema de detecção de anomalias, um adversário passivo pode evitar os mecanismos de defesa influenciando ou prevendo o resultado da etapa de aprendizado da região de comportamento normal.

No primeiro caso, o adversário poderá influenciar o aprendizado do sistema inserindo exemplos maliciosos no conjunto de dados de treino, por exemplo, para que a classificação de anomalias seja ineficiente. No segundo, o adversário poderá prever o comportamento da classificação e fazer uso de um ataque que se encaixe abaixo do nível mínimo de anomalia necessário para a intervenção humana, passando pelo mecanismo de defesa sem ser detectado. Nas técnicas de detecção de desfigurações, considera-se que o atacante não será capaz de manipular o conjunto de dados de treino, mas que poderá causar danos com ataques como no segundo caso acima (BERGADANO et al., 2019)

Uma das formas de evitar um ataque deste tipo é através de sistemas imprevisíveis, em que as etapas do aprendizado e seus componentes sejam públicos e visíveis, mas alguns de seus parâmetros são mantidos em segredo. Esse tipo de aprendizado é chamado de aprendizado com chave (ou *keyed learning*) (BERGADANO et al., 2019).

Um sistema de aprendizado tradicional recebe como entrada um conjunto de dados de treino, de características (*features*) e de restrições e preferências sobre o espaço de hipóteses. Um algoritmo de aprendizado com chave receberá os mesmos parâmetros, mais uma chave secreta. A chave não é visível para o adversário e é gerada aleatoriamente, de forma que não seja possível reproduzi-la.

O processo de aprendizado com chave é ilustrado na Figura 3. A etapa de aprendizado pode se repetir ao longo do tempo, e cada repetição poderá usar diferentes entradas de dados, características e chave. Em cada repetição, a chave chave poderá ser usada para selecionar aleatoriamente as características que serão usadas no aprendizado, os subconjuntos de dados de treino e vieses como os instantes de execução de cada etapa e as restrinções sobre o espaço de hipóteses.

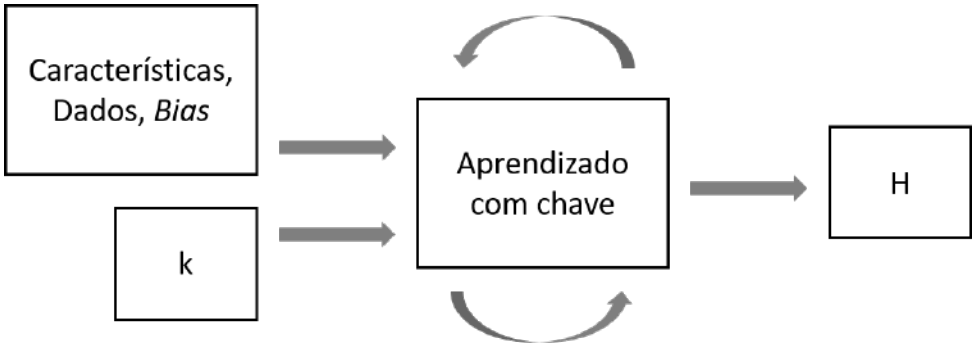

Figura 3 – Aprendizado com chave, baseado em (BERGADANO et al., 2019).

O uso de chaves em um sistema de detecção de anomalias é muito importante quando todos os seus dados de entrada são públicos, como nos sistemas de detecção de desfiguração. Nesses sistemas, os dados de entrada são comumente retirados do próprio domínio que será monitorado e de ataques conhecidos de desfiguração, disponíveis em arquivos da *Internet*. Por outro lado, o uso em excesso da chave aleatória na escolha dos parâmetros do sistema poderá diminuir sua eficiência, caso ocorra uma seleção de entradas com baixo poder preditivo.

2.4 Trabalhos Relacionados

Em KANTI, o autor propõe uma ferramenta de detecção de desfiguração através do cálculo da verificação de soma pelo hash de cada página. O algoritmo proposto é implementado em um navegador próprio, necessitando que seja feita a sua instalação pelo administrador do domínio. Além dessa desvantagem, a ferramenta também não é eficiente para páginas de conteúdo dinâmico.

Para contornar a dificuldade e o custo da instalação e manutenção de ferramentas como em (KANTI, 2011), BARTOLI; DAVANZO; MEDVET propoem um modelo de larga escala de detecção de anomalias, baseado principalmente no monitoramento de medidas estatísticas sobre as páginas. É definida então uma arquitetura dividida em aprendizado e classificação, a partir da extração de determinadas características dos documentos HTML. Apesar do algoritmo proposto ter obtido boa acurácia na detecção de desfigurações em diferentes tipos de conteúdo, as etapas de treino e classificação podem ser computacionalmente difíceis.

Já em HOANG; NGUYEN, o autor propõe uma ferramenta baseada na combinação da detecção de assinaturas e de anomalias. A busca por assinaturas de ataques já conhecidos antes da etapa de classificação evita que as páginas monitoradas sejam sempre enviadas para o classificador, reduzindo os custos computacionais. A ferramenta mostrou-se mais eficiente que outras técnicas baseadas apenas na classificação de anomalias.

Em BERGADANO et al., o autor expõe a necessidade de manter parte do sistema de detecção de anomalias em segredo, já que adversários passivos podem contornar o algoritmo de defesa se tiverem completo conhecimento de todas as suas etapas e forem capazes de reproduzi-lo. Por isso, o autor propõe um sistema de detecção de anomalias em que as suas entradas são escolhidas de forma imprevisível por uma chave secreta, que é gerada aleatoriamente.

Por fim, em BORGOLTE CHRISTOPHER KRUEGEL, os autores se aproximam do problema da desfiguração a partir de um algoritmo de aprendizado de máquina para a classificação nomeado de Meerkat. Ele se diferencia de outros por não necessitar de nenhuma informação prévia além do nome do domínio que será acessado, fazendo com que seja de mais fácil implementação. Analisa-se uma foto do domínio, ao invés do código fonte ou do conteúdo.

Um resumo dos trabalhos citados é ilustrado na Tabela 2.

Tabela 2 – Resumo dos trabalhos relacionados.

	Requer instalação?	É eficiente em páginas dinâmicas?	Requer muito recurso computacional?	É robusta a adversário passivos?
KANTI	x			
BARTOLI; DAVANZO; MEDVET		x	x	
HOANG; NGUYEN		x		
BERGABANO				x
BORGOLTE		x	x	

3 A FERRAMENTA PROPOSTA

A ferramenta proposta está dividida em três módulos principais, mais um módulo de alerta. Os três primeiros módulos compreendem: a busca recursiva em um domínio, a detecção de mudanças no documento web e a classificação da mudança como maliciosa, ou não. O último módulo consiste no envio de alertas por *email*, caso uma desfiguração seja detectada.

A partir de um domínio pré-determinado, a ferramenta fará uma busca recursiva por todas as referências na página e, para a página inicial e cada referência encontrada será realizado um teste para detectar se houve mudanças em seu conteúdo. Quando uma mudança for detectada no documento, ele seguirá para o terceiro módulo, de classificação, onde será determinado se a mudança foi maliciosa. Caso positivo, um alerta será enviado pelo último módulo ao endereço de *email* configurado, para que o administrador seja notificado e possa desfazer o ataque.

A sequência de atuação dos módulos sobre um domínio, desde a sua busca na varredura recursiva até a detecção de mudanças e a classificação delas, é ilustrada na Figura 4.

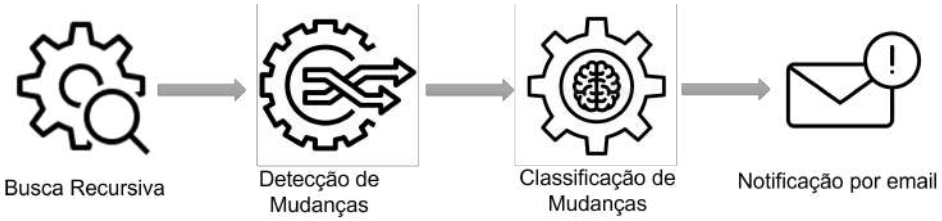

Figura 4 – Sequência de módulos da ferramenta proposta.

A metodologia utilizada em cada etapa, assim como o funcionamento detalhado dos três principais módulos da ferramenta são explicados abaixo. Também são discutidos o sincronismo entre os módulos e o funcionamento do componente de notificação.

3.1 Busca Recursiva

Este módulo consiste na busca recursiva por referências a novas páginas, a partir de um domínio inicial. O objetivo desta etapa é permitir que a ferramenta encontre ataques por desfiguração não apenas na página inicial dada, mas em todas as páginas do domínio.

Quando for executado, o algoritmo deverá buscar e percorrer todas as referências que ainda não foram visitadas, incluindo a página inicial, e encaminhá-las para as etapas

seguintes de detecção e classificação. Além disso, o módulo não deverá percorrer páginas de fora do domínio inicial, limitando a busca apenas ao sítio base e suas referências internas.

3.1.1 Metodologia Usada

Para carregar o documento HTML, utilizou-se a biblioteca *Requests* de *Python* (REQUETS, 2021). Inicialmente, a página a ser visitada é carregada através de uma requisição HTTP 1.1, pelo método GET. Em seguida, os dados são extraídos do documento HTML e armazenados em uma estrutura de dados aninhada. Esta etapa é feita de forma eficiente, usando a biblioteca *lxml* (BEHNEL, 2021) de *Python*.

De acordo com a documentação da biblioteca *BeautifulSoup*, que também permite a manipulação de arquivos HTML em *Python*, o armazenamento da estrutura dos documentos HTML ou XML com a biblioteca *lxml* é mais eficiente do que com as outras implementações disponíveis. Isso ocorre pois o *lxml* se baseia nas bibliotecas *libxml2* (VEILLARD, 2021a) e *libxslt* (VEILLARD, 2021b), da linguagem C, enquanto o beautifulsoup é implementado inteiramente em *Python*.

A partir do objeto de dados obtido com a biblioteca *lxml*, é possível percorrer sua estrutura e buscar todos os elementos do tipo “a”, do padrão HTML. Com esses elementos, pode-se obter os próximos endereços a serem visitados, extraindo o valor de seu atributo “href”. Após obter o endereço, é avaliado se ele é válido e se pertence ao domínio desejado. Caso positivo, o endereço é salvo em uma lista, e a função é chamada novamente para visitá-lo e encontrar suas referências de forma recursiva.

3.1.2 Fluxo da Aplicação

O fluxograma de execução do módulo de busca recursiva é ilustrado na Figura 5. Conforme citado na seção anterior, o módulo inicia em um domínio de partida, e busca todas as referências em seu conteúdo. Enquanto houver referências e elas forem válidas, os endereços encontrados serão salvos em uma estrutura de dados e visitados recursivamente pela função.

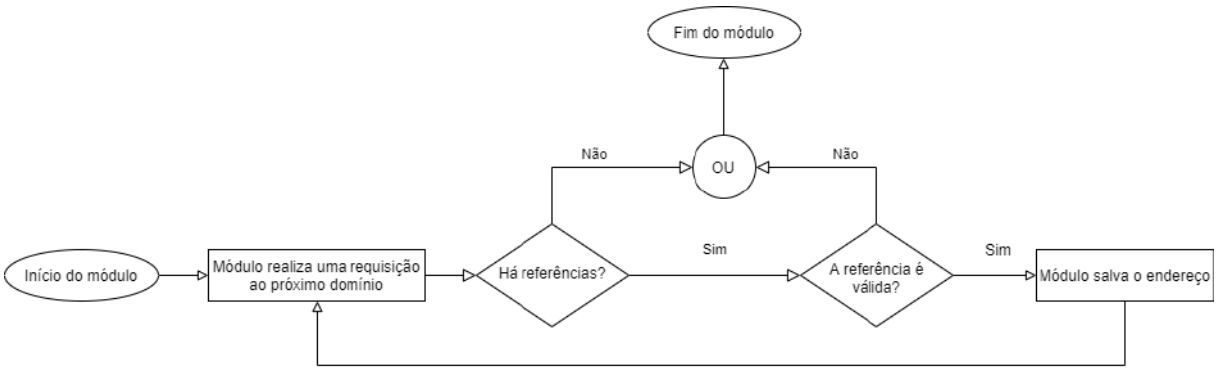

Figura 5 – Fluxograma do módulo de busca recursiva.

3.2 Detecção de Mudanças

Este módulo consiste em uma etapa intermediária na detecção de ataques por desfiguração. Conforme ilustrado na Figura 4, ele é executado após o módulo da busca recursiva, utilizando os resultados obtidos nesta etapa, e antes do módulo de classificação de mudanças, fornecendo os dados necessários para a mesma.

A partir de uma lista de endereços encontrados na busca recursiva, o algoritmo deste módulo deverá percorrer cada item da lista e detectar se houve mudanças em seu conteúdo desde a última execução. Esta etapa servirá como um filtro antes da etapa de classificação. Como o processo de classificação pode ser computacionalmente exaustivo, o filtro permite que apenas as páginas que foram modificadas sejam analisadas na próxima etapa, poupando recursos.

O módulo poderá ser executado repetidamente em intervalos de tempo definidos. A cada execução, os endereços em que forem detectadas mudanças são encaminhados para o próximo módulo, de classificação dessas mudanças.

3.2.1 Metodologia Usada

Dentre as diferentes estratégias que podem ser usadas para a detecção de mudanças em um domínio, como exposto na seção 2.2, escolheu-se a verificação em soma (*checksum*) de algoritmos de *hash*. Nesta ferramenta, o algoritmo selecionado foi o MD5, que será executado utilizando-se a biblioteca *hashlib* (FOUNDATION, 2021b), de *Python*.

Para cada item da lista de endereços obtida na busca, o módulo de detecção receberá como entrada o documento HTML correspondente. Com o documento, o *hash* da página será calculado utilizando-se o método md5, da biblioteca *hashlib*.

3.2.2 Fluxo da Aplicação

O fluxograma de execução do módulo de busca recursiva é ilustrado na Figura 6. O módulo inicia com o primeiro domínio salvo na busca recursiva. Caso seja a primeira execução do código, o valor calculado será armazenado em uma estrutura de dicionário, onde o endereço da página será a chave e o *hash*, o valor desta chave. Isto é feito para todos as referências da lista.

Quando o código for executado novamente, o *hash* será recalculado para cada página encontrada na busca recursiva e comparado com o valor que foi armazenado no dicionário para este endereço na execução anterior. Caso os valores sejam diferentes, uma mudança da página foi detectada, e o endereço será encaminhado para o próximo módulo da ferramenta. Após a comparação, o módulo continuará executando, até analisar o *hash* de todas as páginas salvas na busca recursiva.

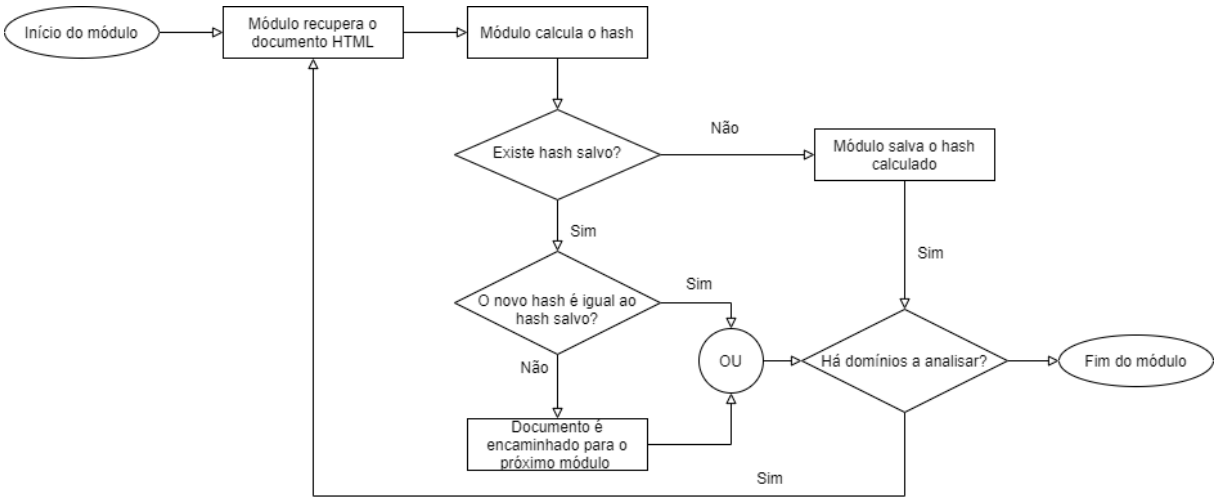

Figura 6 – Fluxograma do módulo de detecção de mudanças.

3.3 Classificação de Mudanças

Este módulo consiste na classificação das mudanças encontradas no módulo anterior como maliciosas, ou não. O módulo irá percorrer e avaliar todas as páginas encontradas na busca recursiva que sofreram mudanças, resultadas dos módulos anteriores. A partir dessas entradas, ele irá detectar as páginas em que as mudanças sofridas podem ser classificadas como uma desfiguração.

Assim, após a execução do módulo de detecção de mudanças, apenas as páginas cujo conteúdo foi modificado são encaminhadas para o módulo de classificação, seguindo o fluxo ilustrado na Figura 4. Essas páginas são então avaliadas de acordo com o método proposto para detectar se houve uma desfiguração. Ao final da classificação, o módulo de

notificação será acionado sempre que houver alguma desfiguração detectada. A metodologia utilizada é explicada abaixo.

3.3.1 Metodologia Usada

Entre as estratégias encontradas na literatura para detectar desfigurações, foram escolhidas algumas regras para a detecção de padrões e assinaturas de pichação. A primeira delas é a busca por assinaturas já utilizadas em ataques anteriores, a partir de uma lista construída manualmente de padrões conhecidos.

Para a construção da lista, foram extraídas 23 assinaturas do arquivo de domínios desfigurados, disponíveis em (ZONEH, 2021). As assinaturas foram encontradas em domínios públicos brasileiros, entre setembro e julho de 2021. A detecção das assinaturas em uma página é feita através da busca de expressões regulares no documento, implementada com a biblioteca *re* (FOUNDATION, 2021d) de *Python*.

Além disso, foi implementada a busca por mudanças bruscas na estrutura das páginas, em que o corpo do documento HTML consiste em apenas uma imagem, que por sua vez apresenta uma assinatura de pichação. Para tanto, o documento HTML da página é armazenado em uma estrutura de dados aninhada através da biblioteca *lxml* (BEHNEL, 2021) de *Python*. Nessa estrutura, é avaliado se a página contém apenas uma *tag* do tipo "img"em seu corpo, e se esta imagem possui uma das assinaturas da lista, classificando nesse caso uma desfiguração.

Para extrair o conteúdo textual da imagem, escolheu-se a ferramenta de reconhecimento ótico de caracteres (OCR) *Tesseract*, desenvolvida pela Google. Essa ferramenta está disponível em *Python* pela biblioteca *pytesseract* (PYPI, 2021). Assim, para cada *tag* do tipo "img"encontrada no documento HTML que satisfaça o teste de mudança estrutural, a imagem é baixada com uma requisição da biblioteca *requests* e seu conteúdo é extraído pela ferramenta de OCR. Em seguida, o texto extraído é analisado e, caso contenha uma das assinaturas listadas, é detectada uma desfiguração, acionando o módulo seguinte.

Assim como discutido no módulo da busca recursiva, o uso da biblioteca *lxml* permite a leitura e armazenamento da árvore do documento HTML de forma eficiente em uma estrutura de dados. Assim, é possível realizar o teste de mudanças bruscas na página usando a sintaxe *XPath* e o método *xpath* dessa biblioteca.

3.3.2 Fluxo da Aplicação

O fluxograma de execução do módulo de classificação de mudanças é ilustrado na Figura 7. O módulo inicia com o carregamento da lista de assinaturas previamente configurada. Ele recebe como entrada o documento HTML de uma página encontrada e encaminhada pelos dois primeiros módulos.

Com a lista de assinaturas e a página a ser analisada, o módulo realiza então a busca por padrões de desfiguração no documento: a presença de uma das assinaturas da lista e a estruturação do corpo do documento HTML em uma única imagem que também contenha uma assinatura. Para as páginas avaliadas, caso um dos padrões seja encontrado, o módulo de notificação é acionado, e o módulo de classificação termina de executar.

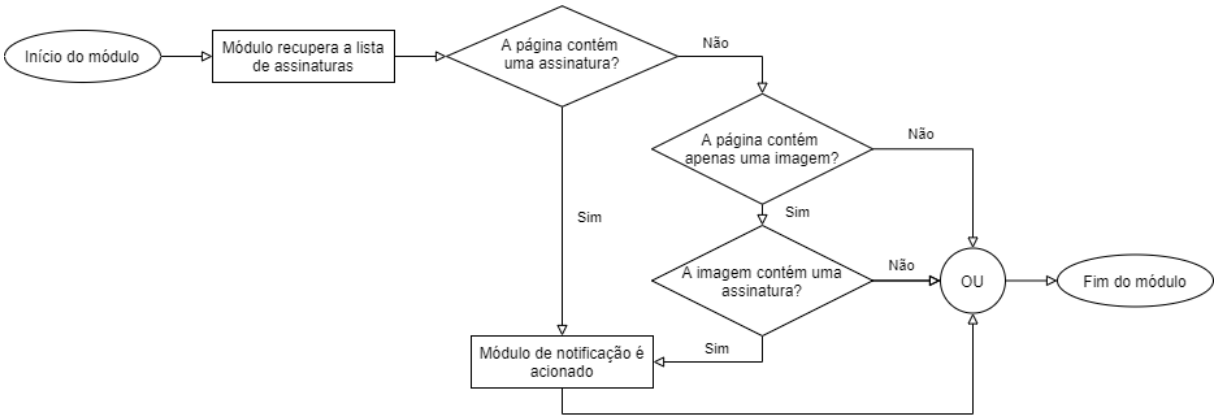

Figura 7 – Fluxograma do módulo de classificação de mudanças.

3.4 Notificação por Email

O módulo de notificação da ferramenta é acionado pelo módulo de classificação, sempre que este último detecta ao menos uma desfiguração. Para todas as desfigurações detectadas, o componente envia uma mensagem de alerta por email, para que o administrador da página seja notificado e possa atuar na correção do ataque.

A partir dos endereços de destinatário e remetente previamente configurados, é criada uma sessão segura com o servidor do gmail usando o protocolo SMTP, através das bibliotecas *smtplib* (FOUNDATION, 2021e) e *ssl* (FOUNDATION, 2021f), *buit-in* de *Python*. Em seguida, a conexão é criptografada e protegida pelo protocolo TLS, utilizando a *smtplib*. Após o *login* na sessão com as credenciais fornecidas na configuração, uma mensagem do tipo MIME com o alerta de desfiguração e os endereços desfigurados é enviada.

3.5 Paralelismo

A implementação de *threads* (FOUNDATION, 2021g) foi escolhida para gerar o paralelismo entre os módulos, de forma que eles possam ser executados simultaneamente, independente do término do módulo anterior. Para isso, as *threads* deverão ser capazes de se comunicar, transferindo informações entre os módulos e evitando *race conditions*, ou

condições de corrida. Condições de corrida são resultados inesperados que podem gerar erros na aplicação por depender da sequencia ou sincronia de outros eventos, sendo nesse caso, duas *threads* diferentes acessando a mesma informação. Para evitar isso, devemos usar objetos considerados seguros para implementação em *threads*, como é o caso de filas em *python*.

Optou-se, então por um padrão de *thread* produtor e consumidor com filas, onde uma *thread* produtora, alimenta uma fila e uma ou mais *threads* consumidoras consomem as informações dessa fila. Para sinalizar o fim de uma thread para as demais, um objeto chamado de sentinela é adicionado ao final da fila. Ao detectar o objeto sentinela, a *thread* consumidora reconhece que a fila chegou ao fim e que a *thread* produtora não populará mais a fila, podendo finalizar sua execução.

Uma outra alternativa possível de ser utilizada seria o *async.io* (FOUNDATION, 2021a) uma biblioteca para escrever código concorrente, utilizando co-rotinas no lugar de *threads*, sendo ideal para estruturas e sistemas *IO-bound*, isto é com período de espera por entrada e saída de uma operação.

4 IMPLEMENTAÇÃO E TESTES DA FERRAMENTA

Neste capítulo, será detalhada a implementação da ferramenta proposta, sua estrutura de arquivos e as principais funções executadas. Além disso, são ilustrados exemplos de funcionamento da aplicação e o resultados dos testes, de acordo com o fluxo proposto anteriormente.

4.1 Estrutura de Arquivos

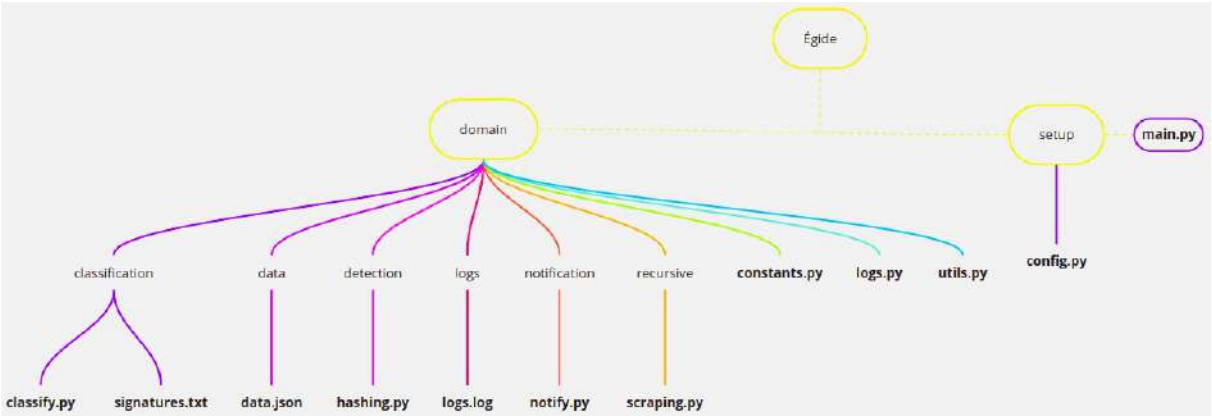

Figura 8 – Estrutura de arquivos

Para a implementação de todos os módulos e a documentação, o código foi estruturado nos arquivos ilustrados na Figura 8.

- *main.py*: Responsável pela a execução do fluxo inteiro da aplicação, iniciando as threads de todos os outros módulos.
- *domain/recursive/scraping.py*: Arquivo com as funções responsáveis por executar o fluxo definido para o módulo recursivo.
- *domain/detection/hashing.py*: Arquivo com as funções responsáveis por executar o fluxo definido para o módulo de detecção.
- *domain/classification/classify.py*: Arquivo com as funções responsáveis por executar o fluxo definido para o módulo de classificação, utilizando *signatures.txt* como arquivo com banco de assinaturas utilizadas nas análises.
- *domain/notification/notify.py*: Arquivo com as funções responsáveis por realizar a notificação para o usuário, feita através de um e-mail, previsamente configurado.

- *domain/logs/logs.log*: Arquivo onde é armazenada toda a documentação do passo a passo das execuções do processo como um todo.
- *domain/data/data.json*: Arquivo onde estão armazenados todos caminhos dentro de cada domínio e seus respectivos *hashes*, para comparação.
- *domain/constants.py*: Contém constantes utilizadas em diversos lugares na aplicação
- *domain/logs.py*: Arquivo que contém a configuração inicial dos *logs* para a documentação do processo em todos os módulos, feita utilizando a biblioteca de *logging* (FOUNDATION, 2021c).
- *domain/utils.py*: Arquivo com um módulo de funções auxiliares úteis principalmente a manipulação de arquivos, leitura e escrita. Onde encontra-se a *thread* de escrita de arquivos.
- *data.json*: Arquivo onde estão armazenados todos caminhos dentro de cada domínio e seus respectivos hashes, para comparação
- *setup/config.py*: Arquivo que contém todas as configurações da aplicação que deverão ser feitas pelo usuário, especificadas em 4.2.1.

4.2 Funcionamento da Aplicação

4.2.1 Estrutura do Código

O caminho a ser executado pelo código seguirá os fluxos propostos no Capítulo 3. Para tanto, na implementação e testes dos módulos, a função principal do arquivo *main.py* foi dividida no desenvolvimento descrito abaixo. Essa estrutura é ilustrada na Figura 9

```
if __name__ == '__main__':
    configure_logs()
    data = load_data()
    urls_queue = Queue()
    write_queue = Queue()
    hits_queue = Queue()
    with concurrent.futures.ThreadPoolExecutor(max_workers=max_threads) as executor:
        executor.submit(compare_hashes_thread, urls_queue, hits_queue, data, len(bases))
        executor.submit(classify_thread, hits_queue, write_queue)
        executor.submit(write_data_thread, write_queue)
        if rerun_recursion:
            for (base, base_domain) in zip(bases, bases_domains):
                executor.submit(recursive_get_urls_in_domain, base, urls_queue, base_domain)

        else:
            executor.submit(fill_urls_queue_thread, data, urls_queue)
```

Figura 9 – Função principal do código

- Em *configure_logs*, o processo de configuração inicial dos logs de documentação é iniciado e é configurado para *domain/logs/logs.py*. Esse arquivo vai conter a documentação com o tempo e o passo a passo de cada uma das etapas e execuções do processo.

- Em *load_data*, a configuração anterior do arquivo *data/data.json* é carregada em memória no formato de dicionário;

- As 3 filas *urls_queue*, *write_queue* e *hits_queue*, utilizadas em todo o processo, são iniciadas;

- A fila *urls_queue* é de comunicação entre o módulo recursivo e o módulo de detecção. Os sítios encontrados no módulo recursivo, bem como a requisição feita para os mesmos, são inseridas nessa fila no módulo recursivo, que é consumida pelo módulo de detecção, aproveitando a requisição e fazendo o *hash* para cada sítio encontrado.

- A fila *hits_queue* é de comunicação entre o módulo de detecção e o módulo de classificação. No módulo de detecção , para todos os sítios que foram encontradas diferenças em *hash*, os sítios, seus respectivos *hashes* e a requisição são inseridos na fila *hits_queue*. O módulo de classificação então consome essa fila, classificando as mudanças em desfiguração ou não;

- A fila *write_queue* é de comunicação entre o módulo de classificação e a *thread* de escrita dos dados em memória. No módulo de classificação, todas as mudanças que tiveram diferença em *hash* e não foram classificadas como desfiguração são enviadas para a fila *write_queue*. A fila é então consumida pela *thread* de escrita dos dados em memória e a partir de o acúmulo de um número configurável de informação, gradualmente atualiza o arquivo em *data/data.json*

- Em *recursive_get_urls_in_domain(base, urls_queue, base_domain)*, será chamado o módulo de busca recursiva para encontrar todas os sítios. Esse passo depende de *base* e *base_domain*, que são parâmetros que devem ser definidos de antemão. O parâmetro *base* é um dos sítios iniciais utilizados para a busca recursiva. O parâmetro *base_domain* é um dos domínios utilizado para restringir o limite da busca recursiva, de modo a buscar mudanças apenas dentro do mesmo domínio. *urls_queue* é a fila usada para comunicação entre os módulos explicada em sequência;

- Em *compare_hashes_thread(urls_queue, hits_queue, data, recursive_threads_count)*, será chamado o módulo de detecção de mudanças, onde serão encontrados os *hashes* para cada sítio presente em *urls_queue*, encontrado na busca recursiva. Em caso de mudanças no *hash*, o sítio, a requisição e o novo *hash* são enviados para a fila *hits_queue*;

- Em *classify_thread(hits_queue, write_queue)*, os sítios recuperados de *hits_queue*, em que houveram mudanças, são analisados quanto a presença de assinaturas, mudança brusca na estrutura da página e reconhecimento de imagens conforme exposto no capítulo 3. Para os casos em que não foram detectados nenhuma desfiguração, as informações são enviadas para *write_queue*, para serem posteriormente atualizadas;
- Em *write_data_thread(write_queue)*, as informações são recuperadas de *write_queue* e acumuladas até um número pré-estabelecido, com 100 sítios como padrão. Os dados são então escritos novamente no arquivo *data/data.json*. Ao terminar todo o processo, as informações restantes também são escritas no arquivo.
- *fill_urls_queue_thread(data,urls_queue)* ocorre quando se opta nas configurações por não fazer o módulo recursivo. Os dados são recuperados de *data.json*, são feitas as requisições para os sítios presentes no arquivo e o sítio e a requisição são colocadas na fila *urls_queue* que é consumidas pelo módulo de detecção.

Para cada url em *bases*, uma *thread* do módulo recursivo é criada, paralelizando as recursões de cada domínio. Caso o número de *threads* criado ultrapasse o estabelecido em *max_threads*, espera-se uma das *threads* anterioras ser finalizada antes de se criar a próxima.

Configurações de execução do processo a serem configuradas no arquivo do *setup/config.py*:

- *recursive.base*: *Urls* pelas quais irão se começar a busca recursiva.
- *recursive.domain*: Lista de domínios aos quais serão restritas, respectivamente, a busca iniciada pelas *urls* na base.
- *recursive.max_depth*: Número que limita a profundidade da recursão. Utilizado em testes para ver o inicio e o fim do processo. Caso não configurado, ou configurado como *None*, não haverá limite, e será percorrido todos sítios válidos contidos no domínio.
- *max_threads*: Número que limita a quantidade máxima de *threads* que vão estar em execução simultaneamente.
- *load_previous_data*: Booleano (*True/False*). Decide se utiliza o arquivo *data.json* previamente carregado ou analisa todos os domínios novamente.
- *rerun_recursion*: Booleano (*True/False*). Decide se o módulo recursivo será executado novamente ou apenas a detecção e classificação baseadas nos dados salvos em *data.json*
- *tesseract_path*: Caminho para o executável do programa *Tesseract* necessário para o módulo de classificação.

- *notification.from_email*: Endereço de *email* do qual será enviado o alerta.
- *notification.password*: Senha para apps gerada para o email do qual será enviado o alerta.
- *notification.to_email*: Endereço de *email* para o qual será enviado o alerta.

4.2.2 Exemplos e testes

Para demonstrar o fluxo de funcionamento da aplicação implementado, foram executados alguns testes englobando todos os módulos e testando algumas métricas de eficiência do processo.

Para os testes a seguir, serão utilizadas as seguintes configurações:

- *recursive.bases*: *"https://webscraper.io/test-sites/e-commerce/static"*.
- *recursive.domains*: *'webscraper.io'*
- *recursive.max_depth*: 100
- *load_previous_data*: True
- *rerun_recursion*: True
- *tesseract_path*: Caminho para o executável do programa *Tesseract*
- *notification.from_email*: Endereço de *email* do qual será enviado o alerta.
- *notification.password*: Senha para aplicações gerada para o email do qual será enviado o alerta.
- *notification.to_email*: Endereço de *email* para o qual será enviado o alerta

O código foi executado 3 vezes para as configurações acima. O sítio *Webscraper.io* (WEBSCRAPER, 2021) utilizado contém um conjunto de domínios especializados em testes, estáticos e dinâmicos, onde se pode analisar com constância alguns dados e métricas de performance dos módulos.

Na primeira execução, como não havia nenhum dado ainda preenchido no arquivo de *data.json*, foram encontradas 100 diferenças de *hash*, e o módulo de classificação foi executado em todas as 100 vezes. Não foi detectada nenhuma desfiguração, as informações foram salvas no arquivo de *data.json* e o processo demorou 53 segundos.

A segunda execução, em seguida, com arquivo de *data.json* já preenchido, encontrou 11 diferenças de *hash*, executando o módulo de classificação 11 vezes. Não foi detectada

nenhuma desfiguração, e as diferenças foram salvas novamente no arquivo de *data.json*. O processo demorou 51 segundos

Na terceira execução, *rerun_recursion* foi configurada para *False*. As informações foram carregadas diretamente do arquivo de data.json e foram encontradas agora 10 diferenças de *hash*, executando 10 vezes o módulo de classificação. Não foi detectada nenhuma desfiguração, e as informações foram salvas no arquivo de *data.json* novamente. Esse processo demorou 55 segundos.

```
--------------------------
LOG START
AUXILIAR: Previous data loaded
RECURSIVE: Limiting max_depth: 100
DETECTION: Calculating Hashes

RECURSIVE: Finding recursive urls

CLASSIFICATION: Classifying hash differences
CLASSIFICATION: Getting regex signatures

RECURSIVE: https://webscraper.io/test-sites/e-commerce/static
DETECTION: hashing https://webscraper.io/test-sites/e-commerce/static
DETECTION: Hash difference for https://webscraper.io/test-sites/e-commerce/static: c8bb159ca039beb11808fb784108c6f4
CLASSIFICATION: No defacement detected for https://webscraper.io/test-sites/e-commerce/static

DETECTION: new_hash: 3b9dcf9b9effb22c1cd4ded9a1350486
RECURSIVE: https://webscraper.io/
DETECTION: hashing https://webscraper.io/
RECURSIVE: https://webscraper.io/cloud-scraper
DETECTION: hashing https://webscraper.io/cloud-scraper
```

Figura 10 – Início dos logs da segunda execução

```
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/scheduler
RECURSIVE: https://webscraper.io/documentation/web-scraper-cloud/data-export
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/data-export
RECURSIVE: https://webscraper.io/documentation/web-scraper-cloud/parser
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/parser
RECURSIVE: https://webscraper.io/documentation/web-scraper-cloud/parser/replace-text
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/parser/replace-text
RECURSIVE: https://webscraper.io/documentation/web-scraper-cloud/parser/regex-match
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/parser/regex-match
RECURSIVE: https://webscraper.io/documentation/web-scraper-cloud/parser/append-and-prepend-text
DETECTION: hashing https://webscraper.io/documentation/web-scraper-cloud/parser/append-and-prepend-text
RECURSIVE: Finished recursion

DETECTION: 11 differences found
CLASSIFICATION: Classification finished
AUXILIAR: Writing remaining Data to data.json
DETECTION: Finished hashing

CLASSIFICATION: Defaced Urls:
[]
AUXILIAR: All data saved successfully

LOG END
---------------------------
```

Figura 11 – Final dos logs da segunda execução

Esse processo das 3 execuções foi repetido 10 vezes, e os números se mantiveram semelhantes e consistentes durante todas as execuções. Observando esses testes, percebe-se que mesmo não fazendo a recursão por todos os domínios no módulo recursivo na terceira

execução, ainda é necessário fazer uma requisição para recuperar as informações de cada um desses sítios, e aí se encontra o principal gargalo da aplicação. A velocidade para encontrar as referências em uma página é desconsiderável frente ao tempo de se realizar uma requisição.

Também foram feitos testes de execução em exemplos reais de domínios que apresentavam uma desfiguração. Após realizar uma pesquisa por sítios desfigurados em tempo real, foram encontrados diversos exemplos em que se pôde testar a aplicação. Um deles foi analisado pela ferramenta com a configuração abaixo:

- *recursive.bases*: *"http://www.sicca.ima.mg.gov.br/index.html"*.
- *recursive.domains*: *None*
- *recursive.max_depth*: None

A *url* especificada apresentava a estrutura presente na figura 12

Figura 12 – Sítio desfigurado

Ao executar a aplicação com a configuração especificada, encontramos o *log* de execução da figura 13.

```
--------------------------
LOG START
AUXILIAR: Previous data loaded
RECURSIVE: Finding recursive urls

DETECTION: Calculating Hashes

CLASSIFICATION: Classifying hash differences
CLASSIFICATION: Getting regex signatures

RECURSIVE: http://www.sicca.ima.mg.gov.br/index.html
DETECTION: hashing http://www.sicca.ima.mg.gov.br/index.html
RECURSIVE: Finished recursion

DETECTION: Hash difference for http://www.sicca.ima.mg.gov.br/index.html: None
DETECTION: new_hash: a9d47bcb03e8ee5c2f8568e9ac30561d
DETECTION: 1 differences found
DETECTION: Finished hashing

CLASSIFICATION: Defacement Detected for http://www.sicca.ima.mg.gov.br/index.html!

NOTIFICATION: defacement email sent to: alessandramalizia@hotmail.com

CLASSIFICATION: Classification finished
AUXILIAR: Writing remaining Data to data.json
CLASSIFICATION: Defaced Urls:
['http://www.sicca.ima.mg.gov.br/index.html']
AUXILIAR: All data saved successfully

LOG END
--------------------------
```

Figura 13 – Logs da execução para o sítio desfigurado

Percebe-se que houve a detecção da desfiguração do domínio e que um email foi enviado notificando tal desfiguração. Assim como o exemplo acima, foi possível testar em diversos outros sítios, além de em imagens disponibilizadas no arquivo do *ZoneH* (ZONEH, 2021). Das dezenas de desfigurações com assinaturas analizadas, duas desfigurações com imagens de assinaturas de fonte confusas e sem nitidez não foram detectadas.

Analisando o segundo módulo, a verificação por *hash* não é eficiente para sítios que tenham informações dinâmicas, isto é, em constante mudança. Ainda assim, é capaz de diminuir consideravelmente o número de análises a serem feitas posteriormente pelo módulo de classificação, já que apenas as páginas que sofreram alguma mudança serão avaliadas, evitando uma sobrecarga e aumentando a eficiência do sistema.

Em termos de eficiência, a manipulação dos dados dentro do código é feita através de dicionários em *Python*. Essas estruturas são eficientes, por usar como implementação interna tabelas *hash*, garantindo complexidade de ordem constante (*O(1)*) para acessar as chaves. O armazenamento em um arquivo *json* garante uma boa estrutura e legibilidade dos dados armazenados. Por outro lado, o uso de arquivos para o armazenamento de endereços é pouco escalável em memória para grandes quantidades de entradas, fazendo-se

necessário o uso de um banco de dados nesse caso.

Já no módulo de classificação, verificou-se que nem todos os exemplos de páginas desfiguradas foram detectados. A escolha das assinaturas, o teste sobre a estrutura da página e o conjunto de dados usado para o treinamento da ferramenta de OCR impõem limitações na tarefa da classificação. Além disso, como a ferramenta é baseada na análise do conteúdo de texto e de imagem de documentos HTML, ela não será capaz de detectar a inserção de scripts maliciosos na página, sendo necessária a implementação futura de outras estratégias de classificação. Melhorias futuras na estrutura de *threads* para permitir múltiplas threads para cada módulo também podem ser implementadas, agilizando ainda mais a aplicação.

Em comparação com os trabalhos relacionados discutidos no Capítulo 2, o Égide não necessita de instalação no servidor do domínio, nem utiliza métodos computacionalmente exaustivos de classificação. Por outro lado, não apresenta proteção contra adversários passivos e irá detectar mudanças frequentes nas páginas dinâmicas. Vale destacar que, neste último caso, a ferramenta não irá detectar um falso positivo para desfiguração, aumentando apenas o número de endereços detectados no módulo de detecção e encaminhados para o módulo de classificação. Neste sentido, faz-se necessário a futura priorização dos endereços a serem analisados, atribuindo baixa prioridade às páginas dinâmicas.

5 CONCLUSÃO

O presente trabalho teve como finalidade implementar uma ferramenta capaz de detectar ataques de desfiguração em domínios selecionados. Essa ferramenta se faz importante por atender aos interesses nacionais na defesa do espaço cibernético brasileiro e pelo grande número de ataques cibernéticos registrados nos últimos anos. Além disso, ela pode ser usada por organizações no meio civil e militar, para minimizar os danos da ocorrência de pichações em seus domínios.

Nesta monografia, foram detalhadas a modelagem e a implementação do sistema Égide proposto, dividido em três módulos principais: a busca recursiva por referências, a detecção e a classificação de mudanças. Também são descritos a execução e os testes realizados para avaliar o funcionamento da ferramenta, explicitando as métricas utilizadas.

Nos testes, foi possível verificar a integração entre os módulos, incluindo seu sincronismo e o paralelismo planejados. Foram verificados a capacidade do sistema de percorrer um domínio recursivamente, encontrar as referências no domínio e detectar desfigurações de exemplos extraídos da *Internet*. Assim, o trabalho cumpriu os objetivos propostos, dentro do escopo deste projeto de fim de curso.

Como trabalhos futuros, propõem-se mudanças no arquivo de dados da ferramenta, utilizado para manter a lista dos endereços a serem visitados e seus respectivos valores de *hash*. Entre elas está a criação de um banco de dados, permitindo maior escalabilidade de armazenamento. No banco, os endereços podem ser armazenados juntamente com um atributo de prioridade, de forma que páginas com maior prioridade, como a página principal de um domínio, sejam analisadas com maior frequência do que páginas menos relevantes. De maneira semelhante, páginas dinâmicas que apresentarem mudanças não maliciosas com muita frequência podem ser atribuídas a um nível mais baixo de prioridade.

A partir do banco de dados proposto, são sugeridas novas métricas de classificação de desfigurações para o terceiro módulo. Com o armazenamento de um histórico de características sobre cada página HTML, como a quantidade de *tags* de cada tipo e a máxima árvore comum entre as observações, discutidas no Capítulo 2, seria possível inserir novos testes de desfiguração. Além disso, pode-se incluir novos padrões de pichação nas assinaturas utilizadas pelo terceiro módulo e aprimorar a ferramenta de OCR utilizada, através do pré-processamento das imagens e do treinamento da ferramenta com outros conjuntos de dados.

Ainda como trabalhos futuros, escalando a aplicação, pode-se implementar um número de *threads* dinâmico para os módulos, de acordo com a necessidade. Essa melhoria seria específicamente útil para o módulo de recursão, onde são feitas as requisições, que são

o maior gargalo da aplicação. A implementação geraria a necessidade de um monitoramento e compartilhamento de informações entre essas *threads* do módulo recursivo: informações de quais domínios já foram visitados, para evitar uma recursão em *loop*. Tal implementação também poderia ser feita com o uso da biblioteca *async.io*, que se encaixa perfeitamente nesse caso, como descrito em 3.5.

REFERÊNCIAS

BAI, X.; GU, W.; CHELLAPPAN, S.; WANG, X.; XUAN, D.; MA, B. Pas: Predicate-based authentication services against powerful passive adversaries. In: *2008 Annual Computer Security Applications Conference (ACSAC)*. [s.l.]: IEEE, 2008. p. 433–442. 15 mai. de 2021. Disponível em: <https://ieeexplore.ieee.org/abstract/document/4721578>.

BARTOLI, A.; DAVANZO, G.; MEDVET, E. A framework for large-scale detection of web site defacements. *ACM Trans. Internet Techn.*, v. 10, 2010. 15 mai. de 2021. Disponível em: <https://medvet.inginf.units.it/publications/2010-j-bdm-framework/>.

BEHNEL, S. *lxml - XML and HTML with Python*. 2021. 12 ago. de 2021. Disponível em: <https://lxml.de/>.

BERGADANO, F.; CARRETTO, F.; COGNO, F.; RAGNO, D. Defacement detection with passive adversaries. *Algorithms*, v. 12, p. 150, 2019. 15 mai. de 2021. Disponível em: <https://www.mdpi.com/1999-4893/12/8/150>.

BERGADANO FABIO CARRETO, F. C. F.; RAGNO, D. A conceptual approach to detect webdefacement through artificial intelligence. *International Journal of Advanced Computer Technology*, v. 3, n. 6, 2014. 15 mai. de 2021. Disponível em: <https://www.ijact.org/ijactold/volume3issue6/IJ0360061.pdf>.

BLOCKCHAIN, C. *Testing Different Image Hash Functions*. 2020. 15 mai. de 2021. Disponível em: <https://content-blockchain.org/research/testing-different-image-hash-functions/>.

BORGOLTE CHRISTOPHER KRUEGEL, G. V. K. Meerkat: Detecting website defacements through image-based object recognition. *USENIX Security Symposium 2015*, 2015. 15 mai. de 2021. Disponível em: <https://sites.cs.ucsb.edu/~chris/research/doc/usenix15_meerkat.pdf>.

BORGOLTE, K.; KRUEGEL, C.; VIGNA, G. Delta: Automatic Identification of Unknown Web-based Infection Campaigns. In: GLIGOR, V. D.; YUNG, M. (Ed.). *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery (ACM). p. 109–120. ISBN 978-1-4503-2477-9. 19 mai. de 2021. Disponível em: <http://dx.doi.org/10.1145/2508859.2516725>.

BRASIL. Decreto nº 6.703, de 18 de dezembro de 2008. aprova a estratégia nacional de defesa, e dá outras providências. 2008. 15 mai. de 2021. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/decreto/d6703.htm>.

CONSORTIUM, W. W. W. *W3C DOM 4.1*. 2020. 15 mai. de 2021. Disponível em: <https://www.w3.org/TR/dom41/>.

COOKS, A.; OLIVER, M. S. Curtailing web defacement using a read-only strategy. In: *Proceedings of the Fourth Annual Information Security South Africa Conference*. [s.l.]: [s.n.]], 2014. 15 mai. de 2021. Disponível em: <http://mo.co.za/open/deface.pdf>.

CRUZ, J. V. V. da. *O Futuro da Segurança Cibernética no Brasil: Papéis e Responsabilidades*. 16 p. Monografia (Curso de Aperfeiçoamento Militar) — Escola de Formação Complementar do Exército, Rio de Janeiro, 2020.

DUTRA, A. M. C. Introdução à guerra cibernética: a necessidade de um despertar brasileiro para o assunto. In: *Simpósio de Guerra Eletrônica*. [s.l.]: [s.n.]], 2007. 15 mai. de 2021. Disponível em: <https://www.sigeold.ita.br/anais/IXSIGE/Artigos/GE_39.pdf>.

FOUNDATION, P. S. *asyncio — Asynchronous I/O*. 2021. 02 out. de 2021. Disponível em: <https://docs.python.org/3/library/asyncio.html>.

FOUNDATION, P. S. *hashlib — Secure hashes and message digests*. 2021. 20 jul. de 2021. Disponível em: <https://docs.python.org/3/library/hashlib.html>.

FOUNDATION, P. S. *logging - Logging facility for Python*. 2021. 02 out. de 2021. Disponível em: <https://docs.python.org/3/library/logging.html>.

FOUNDATION, P. S. *re - Regular Expression Operations*. 2021. 27 ago. de 2021. Disponível em: <https://docs.python.org/3/library/re.html>.

FOUNDATION, P. S. *smtplib — SMTP protocol client*. 2021. 20 set. de 2021. Disponível em: <https://docs.python.org/3/library/smtplib.html>.

FOUNDATION, P. S. *ssl — TLS/SSL wrapper for socket objects*. 2021. 20 set. de 2021. Disponível em: <https://docs.python.org/3/library/ssl.html#ssl-security>.

FOUNDATION, P. S. *threading — Thread-based parallelism*. 2021. 02 out. de 2021. Disponível em: <https://docs.python.org/3/library/threading.html>.

HACKERTARGET. *Extract Links From Page*. 2021. 18 mai. de 2021. Disponível em: <https://hackertarget.com/extract-links/>.

HOANG, D.; NGUYEN, T. Detecting website defacements based on machine learning techniques and attack signatures. *Computers*, v. 8, p. 1, 2019. 15 mai. de 2021. Disponível em: <https://www.mdpi.com/2073-431X/8/2/35>.

JUNIOR, R. M. *Segurança e Defesa do Espaço Cibernético Brasileiro*. 1. ed. [s.l.]]: Cubzac, 2010. 476 p.

KANTI, T. Implementing a web browser with web defacement detection techniques. *World of Computer Science and Information Technology Journal (WCSIT)*, v. 1, n. 7, p. 307–310, 2011. 17 mai. de 2021. Disponível em: <http://wcsit.org/media/pub/2011/aug/ImplementingaWebBrowserwithWebDefacementDetectionTechniques.pdf>.

KERRISK, M. *Linux manual page*. 2020. 15 mai. de 2021. Disponível em: <https://man7.org/linux/man-pages/man1/diff.1.html>.

KUMAR, C. *10 Website Defacement Monitoring Tools for Better Security*. 2020. 15 mai. de 2021. Disponível em: <https://geekflare.com/website-defacement-monitoring/>.

LABS, F. *Threat Intelligence Insider*. 2020. 15 mai. de 2021. Disponível em: <https://www.fortiguardthreatinsider.com/pt/bulletin/Q4-2020>.

MACHADO, M. Entre o controle e o ativismo hacker: a ação política dos anonymous brasil. *História, Ciências, Saúde-Manguinhos*, v. 22, p. 1531–1549, 2015. 15 mai. de 2021. Disponível em: <https://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-59702015001001531>.

OWASP. *DirBuster Package Description*. 2021. 18 mai. de 2021. Disponível em: <https://tools.kali.org/web-applications/dirbuster>.

PARKS, R.; DUGGAN, D. Principles of cyberwarfare. *Security and Privacy, IEEE*, v. 9, p. 30 – 35, 2011. 15 mai. de 2021. Disponível em: <https://revista.egn.mar.mil.br/index.php/revistadaegn/article/download/797/pdf_1>.

PYPI. *Python-tesseract is a python wrapper for Google's Tesseract-OCR*. 2021. 25 set. de 2021. Disponível em: <https://pypi.org/project/pytesseract/>.

REQUETS, P. *Requests: HTTP for Humans*. 2021. 15 jul. de 2021. Disponível em: <https://docs.python-requests.org/en/master/>.

SCHMITT, A. M. e Sandro Cordeiro e Hermes Mello e M. Inovação em defesa cibernética brasileira no contexto sul-americano. *Revista Política Hoje*, v. 23, n. 2, p. 87–104, 2015. ISSN 0104-7094. 15 mai. de 2021. Disponível em: <https://periodicos.ufpe.br/revistas/politicahoje/article/view/3743>.

SHANNON, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, XXVII, n. 3, p. 379–423, 1948. 17 mai. de 2021. Disponível em: <https://ieeexplore.ieee.org/document/6773024>.

VEILLARD, D. *The XML C parser and toolkit of Gnome*. 2021. 12 ago. de 2021. Disponível em: <http://xmlsoft.org/>.

VEILLARD, D. *The XSLT C library for GNOME*. 2021. 12 ago. de 2021. Disponível em: <http://xmlsoft.org/XSLT/>.

WEBSCRAPER. *Test Sites*. 2021. 20 jul. de 2021. Disponível em: <https://webscraper.io/test-sites>.

ZONEH. *Yearly Attacks Statistics*. 2020. 15 mai. de 2021. Disponível em: <http://www.zone-h.org/stats/ymd>.

ZONEH. *Archive*. 2021. 20 out. de 2021. Disponível em: <http://www.zone-h.org/archive/filter=1/special=1>.