RNP
ORGANIZAÇÃO SOCIAL DO MCTI

Educação, Pesquisa e Inovação em Rede

Programa P&D do Hackers do Bem

RELATÓRIO DE PROSPECÇÃO

Título do Projeto: GT-CRIVO

Priorização Contextualizada de Vulnerabilidades Orientada a Negócio

Autores: Ítalo Cunha

Data: 17 de março de 2024

1 INTRODUÇÃO

Vulnerabilidades de software podem afetar negativamente o funcionamento de sistemas computacionais, por exemplo, levando à degradação de desempenho, indisponibilidade, corrupção ou vazamento de dados, e uso não autorizado de recursos [56, 60]. Incidentes de segurança trazem bilhões de prejuízo e afetam milhões de usuários todos os anos [4, 47, 54, 54, 66, 183].

A comunidade de segurança reporta, verifica e cataloga novas vulnerabilidades continuamente [138]. Ferramentas de monitoramento proprietárias e de código aberto implementam testes para detectar e identificar um número crescente de vulnerabilidades. Devido à complexidade, dinamicidade e alto nível de integração de sistemas computacionais modernos, a execução destas ferramentas geralmente identifica uma grande quantidade de vulnerabilidades, frequentemente de severidade crítica ou alta [138]. A alta taxa de vulnerabilidades identificadas frequentemente sobrecarrega equipes de desenvolvedores, operadores e mesmo analistas de segurança, inviabilizando que entidades consigam investigar vulnerabilidades conhecidas em tempo hábil, muito menos implementar ou aplicar contramedidas. Esta situação é agravada quando entidades não conseguem alocar recursos humanos para a resolução de vulnerabilidades. Mesmo quando conseguem, alocar os limitados recursos humanos disponíveis de forma eficiente é desafiador face à quantidade de vulnerabilidades [40, 49].

Um ponto positivo neste cenário é que uma fração significativa das vulnerabilidades reportadas possuem mitigações (*patches*) no momento em que são tornadas públicas [138]. Idealmente isto permitiria que equipes de analistas mantivessem sistemas atualizados e seguros contra a maioria das vulnerabilidades. Porém, na prática, equipes de segurança encontram diversos desafios em atingir este objetivo. A manutenção de informações atualizadas sobre vulnerabilidades requer atenção de especialistas, o que leva a várias vulnerabilidades terem informações incompletas ou desatualizadas [60]. A complexidade de vulnerabilidades de segurança aumenta junto da complexidade dos sistemas afetados [83]. A aplicação de atualizações requer acompanhamento por um analista, precisa ser realizada sem comprometer o funcionamento dos sistemas vulneráveis, e pode exigir ajustes quando as atualização não são retro-compatíveis (*backwards compatible*) [110, 156]. Combinados com a grande quantidade de vulnerabilidades detectadas, a realidade é que sistemas continuam vulneráveis mesmo quando atualizações estão disponíveis [40, 49, 143].

Um significativo esforço vem sendo realizado pela comunidade científica, por instituições especializadas, e empresas privadas para tentar remediar esta situação. Nesta revisão bibliográfica focamos em esforços nestas três frentes que visam priorizar vulnerabilidades de software com o objetivo de auxiliar equipes de desenvolvedores e analistas manter sistemas seguros considerando diversas restrições impostas, por exemplo, por disponibilidade de recursos humanos, hardware, software, clientes e usuários.

O conhecimento adquirido na construção deste relatório irá orientar o desenvolvimento de novas soluções para priorização de vulnerabilidades. Neste contexto, temos três ideias que esperamos irão permitir a construção de soluções mais eficazes. Primeiro, vulnerabilidades distintas têm impacto variado dependendo da entidade afetada. Por exemplo, uma bancos e empresas de análise de crédito podem priorizar vulnerabilidades que permitem vazamento de dados e afetam a privacidade de seus clientes ou que comprometam a integridade de bases de dados, enquanto uma empresa de distribuição de conteúdo pode priorizar vulnerabilidades que afetam o desempenho ou disponibilidade do serviço. Segundo, diferentes equipes de funcionários podem solucionar vulnerabilidades de diferentes formas, e com diferentes graus de eficiência. Por exemplo, um time de desenvolvedores pode melhorar a segurança atualizando a pilha de software em uso para versões mais recentes, enquanto um time de administradores de sistema terá maior facilidade em

mitigar vulnerabilidades configurando *firewalls* ou políticas de acesso (ACLs). Por último, contexto externo à vulnerabilidade pode (des)motivar esforços imediatos de remediação. Vulnerabilidades recentes, de fato exploradas, com provas de conceito ou em sistemas críticos devem ser tratadas com prioridade visto a elevada chance de serem alvo de ataques e danos potenciais. Por exemplo, uma vulnerabilidade de *cross-site scripting* (XSS) [69] ou de *SQL injection* [106] requer remediação imediata: Estes ataques são comuns, exploráveis remotamente, amplamente visados e ativamente procurados por sistemas de varreduras (*scanning*). Por outro lado, uma vulnerabilidade que requer acesso de administrador ou acesso à rede local pode receber uma prioridade menor em função da dificuldade de ser explorada.

Nosso objetivo final é empoderar entidades e seus funcionários, direcionando esforços em cibersegurança para que consigam mitigar um número maior de vulnerabilidades mais relevantes, obtendo a máxima melhoria de segurança dentro dos recursos dedicados. Pretendemos gerar contribuições práticas desenvolvendo e empacotando as soluções desenvolvidas para facilitar seu uso e implantação.

2 ESTADO DA ARTE

A comunidade científica e de analistas trabalham ativamente em diversas frentes contra vulnerabilidades. Discutimos a cinco estágios do processo de mitigação de vulnerabilidades de *software*: identificação de vulnerabilidades via análise de código (Seção 2.1), detecção de vulnerabilidades em produção (Seção 2.2), catalogação de vulnerabilidades (Seção 2.3), modelagem de vulnerabilidades (Seção 2.4) e priorização de vulnerabilidades (Seção 2.5). Damos foco especial nas bases de dados que catalogam diversas informações sobre vulnerabilidades e mecanismos de priorização (Seções 2.3 e 2.5).

2.1 Identificação de Vulnerabilidades via Análise de Código

A identificação de vulnerabilidades a partir da análise do código de um programa permite mitigação antes do programa ser distribuído e entrar em operação, contribuindo significativamente para a melhoria da qualidade do *software* e segurança de sistemas [91, 126, 180].

A análise estática de código verifica propriedades de um programa a partir de seu código-fonte, sem executá-lo. Abordagens comuns para análise estática de código incluem:

Verificação por regras [28, 48]. A verificação por regras garante que o código segue um conjunto pré-definido de regras. Regras podem especificar requisitos de segurança como restringir algoritmos criptográficos a versões seguras, verificar a configuração de algoritmos criptográficos e exigir que entradas de usuário ou recebidas pela rede sejam validadas antes de serem usadas. A verificação por regras aborda aspectos específicos do código de forma determinística.

Análise de padrões [74, 78]. A análise de padrões envolve análise de código para identificar padrões e verificar propriedades gerais. Por exemplo, a análise de padrões pode verificar a aderência a melhores práticas e convenções, bem como a utilização de padrões de projeto (*design patterns*) adequados. A análise de padrões toma uma visão mais ampla do código, com resultados potencialmente ambíguos ou incompletos.

Mapeamento de fluxo de dados [26, 114]. O mapeamento de fluxo de dados envolve identificar as fontes dos dados utilizadas pelo programa e construir um grafo do fluxo dos dados através das manipulações, operações e transformações realizadas pelo programa. O grafo permite inferir propriedades sobre as saídas geradas pelo programa, por exemplo, se existe

possibilidade de vazamento de dados sensíveis em alguma saída. A análise de fluxo de dados também permite identificar vulnerabilidades como uso de variáveis não inicializadas. Por último, o mapeamento do fluxo de dados permite identificar quais variáveis dependem de dados recebidos do usuário ou da rede e podem exigir validação.

Execução simbólica [10, 24, 115]. A execução simbólica de código permite inferir propriedades sobre qual código (fluxo de execução) de um programa é executado para cada entradas. A execução simbólica faz esse mapeamento para todos os possíveis fluxos de execução para todas as possíveis entradas, provendo um panorama completo do comportamento de um programa. A execução simbólica permite identificar discrepâncias semânticas entre entradas e o código executado, geralmente devido à presença de erros ou violação de alguma suposição embutida no código.

A análise dinâmica de código verifica propriedades de um programa durante sua execução. A análise dinâmica depende da instrumentação do código, que pode ser provida por um interpretador, máquina virtual, ambiente de execução (*runtime*) ou um compilador. Algumas abordagens para análise dinâmica de código incluem:

Testes com *fuzzing* [111, 152, 153]. Estas técnicas visam testar a execução de um programa com um grande conjunto de entradas. Entradas podem ser geradas aleatoriamente por *fuzzers* caixa-preta [153] ou geradas de forma sistemática por *fuzzers* caixa-branca, que visam minimizar o número de entradas sem sacrificar ou até aumentando a cobertura do código testado. *Fuzzers* caixa-branca podem utilizar outras técnicas de análise de código no processo de geração de entradas, como rastreamento de marcadores [53], modelagem [112], ou execução simbólica [57].

Rastreamento de marcadores (*taint tracking*) [107, 113, 132, 157]. Similar ao mapeamento de fluxo de dados realizado em análise estática, o rastreamento de marcadores utiliza instrumentação de código para rastrear dependências de dados durante a execução de um programa. A análise dinâmica permite melhor cobertura do código de um programa, particularmente de aspectos dinâmicos que não podem ser exercitados estaticamente, e reduz a quantidade de falsos positivos ao reportar sobre observações diretas da execução do código, por exemplo, ao observar vazamentos concretos de dados sensíveis.

Modelagem. A execução dinâmica de código e verificação de propriedades pode tomar várias formas, dependendo principalmente de propriedades conhecidas de um programa que podem ser capturadas por modelos. Por exemplo, a verificação de invariantes [94, 94] permite verificar se propriedades predeterminadas conhecidas que deveriam ser mantidas por um programa (*e.g.,* em um laço) são realmente mantidas durante toda a execução de um programa. De forma geral, modelos podem ser utilizados para verificar propriedades de um programa analisado estaticamente ou executado dinamicamente, por exemplo, através de arcabouços para interpretação abstrata de programas [146, 162]. A violação de um modelo durante a execução dinâmica de um programa permite a identificação de erros, melhorando corretude e reduzindo vulnerabilidades.

Detecção de erros de concorrência [25, 137]. Erros de concorrência como condições de corrida e *deadlocks* em programas paralelos ou sistemas distribuídos são sérias fontes de erros e vulnerabilidades. Sua detecção é complicada pela dificuldade de reproduzir estes erros. Uma abordagem para identificar erros de concorrência é intercalar de forma sistemática o escalonamento de *threads* de um programa [29, 58]. Outra abordagem é instrumentar código para rastrear a ordem de acessos à memória e a recursos com o

objetivo de identificar sequências inválidas de acesso como uso de dados desatualizados ou não inicializados.

Resultados de pesquisa são frequentemente integrados ou disponibilizados como ferramentas. A comunidade científica e de analistas de segurança desenvolve e mantêm dezenas de ferramentas para análise estática e dinâmica de código [15, 61]. Ferramentas de análise de código têm ganhado popularidade e seu uso vem aumentando concomitante à adoção de arcabouços de integração contínua (CI) por diversos projetos de *software* [122, 179]. Apesar dos avanços na análise estática e dinâmica de código, dois desafios limitam sua acurácia e cobertura: (i) a dificuldade de inferir, na análise estática, e exercitar, na análise dinâmica, todos os estados e sequências de eventos possíveis na execução de um programa [29]; e (ii) o alto custo computacional de técnicas avançadas [146]. A quantidade crescente de vulnerabilidades descobertas após a distribuição de *software* indica que as técnicas e ferramentas atuais têm cobertura limitada face à complexidade de sistemas modernos [89, 163], justificando a implantação de sistemas complementares para detecção de vulnerabilidades que discutimos a seguir.

2.2 Detecção de Vulnerabilidades via Monitoramento

A comunidade científica e de analistas de segurança trabalha continuamente no desenvolvimento de novas ferramentas para monitoramento e descoberta de vulnerabilidades de segurança [14, 43, 51, 117]. Este esforço é essencial para entender como detectar vulnerabilidades em programas e sistemas em uso, ponto de partida para esforços de mitigação e suporte a operação continuada destes sistemas. De forma complementar, a pesquisa em métodos de monitoramento e deteção permite antecipar e desenvolver contramedidas contra ataques perpetrados por agentes maliciosos.

O monitoramento e levantamento de informações para fins de exploração de vulnerabilidades pode considerar diversos alvos [117]: indivíduos, perfis e contas de usuários em um sistema, organizações, informações sobre infraestrutura física como configurações de *hardware* e topologia de rede, bem como programas e serviços executando em um dispositivo. Apesar de estarem fora do escopo deste relatório, é importante ressaltar que a combinação de diversas fontes de informação é utilizada para realizar sofisticados ataques [5, 64]; exemplos incluem o ataque cibernético contra a malha elétrica da Ucrânia [47] e o cibergolpe contra o Banco de Bangladesh [66].

No restante desta seção focamos no monitoramento e descobrimento de vulnerabilidades de *software* que podem ser exploradas remotamente através de uma rede, uma classe de vulnerabilidades de alto risco devido à facilidade de serem exploradas. Agrupamos as diversas técnicas existentes para levantar informações e identificar vulnerabilidades em dispositivos em duas classes:

Varredura de rede (*scanning*) [13, 20, 41, 167]. A varredura de rede é uma técnica para identificar dispositivos ativos em uma rede e aplicações acessíveis pela rede. Varreduras de rede podem utilizar ou combinar pacotes ICMP, TCP e UDP com diferentes propriedades (e.g., portas e *flags* TCP) para induzir uma resposta de dispositivos ativos [1, 45]. Técnicas de varredura também podem estabelecer conexões e comunicar com uma aplicação, através de protocolos específicos utilizados pela aplicação, para obter informações adicionais [12, 44]. Por exemplo, este tipo de varredura frequentemente visa obter *banners*, uma resposta de um serviço que frequentemente permite identificar a aplicação e a versão escutando em uma porta de rede. *Malwares* implementam diversos mecanismos de varredura de rede para maximizar a quantidade de dispositivos vulneráveis identificados [1, 8, 102].

***Fingerprinting* [21, 109, 128, 130, 131].** Esta técnica permite identificar propriedades de um dispositivo ou aplicação, como modelo, sistema operacional, ou versão de um *software* (mesmo quando uma aplicação não envia um *banner*). Isto é feito através de observações do

comportamento do dispositivo ou aplicação alvo em cenários específicos. As observações podem ser coletadas por sondagem ativa ou através da observação passiva de tráfego de rede por longos períodos. Estas observações permitem construir uma assinatura do tráfego de uma aplicação ou dispositivos, permitindo compará-lo com um conjunto de assinaturas conhecidas. Por exemplo, o sistema operacional de um dispositivo pode ser inferido observando como ele inicializa o campo ID de pacotes IP [79], como trata pacotes com a opção de *Record Route* [135], ou qual algoritmo de controle de congestionamento ele utiliza no protocolo TCP [18].

A seguir discutimos quatro dimensões que perpassam o projeto e execução de soluções de monitoramento de rede para detecção de vulnerabilidades:

Eficiência e escalabilidade. Uma barreira ao monitoramento de vulnerabilidades de rede é o tamanho da Internet. Monitorar bilhões de dispositivos conectados à Internet requer quantidade significativa de recursos de rede; primariamente banda de rede, mas a quantidade de pacotes frequentemente pode colocar carga significativa em roteadores [19, 36]. Para agravar esta situação, a dinamicidade dos dispositivos implica a necessidade de monitoramento constante para manter uma visão atualizada do estado da rede. Diversos esforços na literatura visam reduzir o número de sondas necessárias para monitoramento de rede [19, 96, 170, 177] e a velocidade com a qual a sondagem pode ser realizada [44, 45].

Furtividade. Algumas técnicas de sondagem e *malware* utilizam mecanismos sofisticados para realizar varreduras de rede, coletando informações por longos períodos sem serem detectados [145, 148]. Por exemplo, estas técnicas podem minimizar o número de pacotes enviados, distribuir o processo de sondagem no tempo e no espaço (entre diversos dispositivos alvo), falsificar o endereço IP de origem (*spoofing*) para mascarar a fonte do ataque, ou utilizar técnicas de sondagem modificadas para contornar dispositivos de monitoramento [37].

Seleção de alvos. A seleção de alvos para uma varredura de rede envolve diversas decisões. Uma varredura pode ser vertical, focando em múltiplas portas de rede de um único dispositivo para identificar todas as aplicações em execução, ou horizontal, focando em uma porta de vários dispositivos para identificar dispositivos executando uma aplicação específica [13]. Para varreduras horizontais, os endereços IPs podem ser obtidos de uma lista pré-construída (*hitlist*) ou a partir de prefixos IP (e.g., controlados por uma rede ou instituição alvo), os endereços podem ser sondados de forma exaustiva ou seletiva, e a ordem de sondagem pode ser sequencial, aleatória, ou seguindo alguma priorização específica [2]. Para varreduras verticais, todas as portas podem ser sondadas, ou um subconjunto de portas alvo pode ser considerado [1]. Varreduras híbridas combinam propriedades de varreduras verticais e horizontais, cobrindo múltiplas portas de diversos dispositivos.

Abordagem de medição. A varredura de rede pode ser realizada de forma passiva, observando o tráfego de rede, ou de forma ativa, enviando pacotes de sondagem para dispositivos na rede [19, 96, 170, 177]. Abordagens ativas incorrem custo adicional de medição, mas permitem escolha dos alvos e das técnicas de sondagem. Abordagens passivas não incorrem custo adicional de medição, são mais difíceis de detectar, mas são aplicáveis apenas aos dispositivos e aplicações enviando tráfego através da rede monitorada [65, 174, 175].

2.3 Bases de Dados Públicas

Nesta seção discutimos as diversas bases de dados sobre vulnerabilidades de *software* disponíveis para analistas de segurança. Na medida do possível agregamos diversas bases de dados por tipo ou similaridade.

Catalogação de vulnerabilidades. A catalogação de vulnerabilidades é um esforço coordenado contínuo para coletar, verificar e manter informações sobre vulnerabilidades de *software*. O CVE (*Common Vulnerabilities and Exposures*) [103] é o mais amplo destes esforços, catalogando vulnerabilidades reportadas por desenvolvedores, analistas, pesquisadores e fabricantes. O CVE permite identificar vulnerabilidades de forma única, facilitando a gerência de vulnerabilidades. O CVE é o conjunto de dados mais frequentemente utilizado em pesquisas sobre vulnerabilidades [83]. Outras bases de vulnerabilidades existem, como o NVD (*National Vulnerability Database*), mantido pelo NIST (*National Institute of Standards and Technology*), o descontinuado OSVDB (*Open Source Vulnerability Database*), bem como as bases da Snyk e VulDB.[1]

O CVD (*Coordinated Vulnerability Disclosure*)[2] descreve o processo de divulgação de vulnerabilidades e visa garantir que vulnerabilidades sejam divulgadas de forma responsável, minimizando o risco de exploração por agentes maliciosos. As etapas do processo incluem a coleta e análise de dados sobre vulnerabilidade, notificação de desenvolvedores e fabricantes responsáveis, implementação e aplicação de medidas corretivas, bem como a divulgação simultânea por fabricantes, prestadores de serviço, distribuidores de *software* e pelo CVE.

Ranqueamento de vulnerabilidades. Diversas bases de dados ranqueiam vulnerabilidades de acordo com critérios específicos. Por exemplo, o CVSS (*Common Vulnerability Scoring System*) [104] é um sistema de ranqueamento de vulnerabilidades que visa padronizar a avaliação de vulnerabilidades em função de propriedades como vetor de ataque, complexidade de exploração e a necessidade de interação de um usuário. O EPSS (*Exploit Prediction Scoring System*) [73] é um sistema de ranqueamento de vulnerabilidades que visa prever a probabilidade de uma vulnerabilidade ser explorada em um futuro próximo utilizando dados complementares, por exemplo, monitoramento de discussão de uma vulnerabilidade em redes sociais como Twitter. O catálogo KEV (*Known Exploited Vulnerabilities*) é uma lista de vulnerabilidades para as quais há comprovação de exploração em ataques reais [90].

Metadados sobre vulnerabilidades. O CVE está fortemente acoplado a outras bases de dados que complementam informações sobre vulnerabilidades. Em particular, o *Common Weakness Enumeration* (CWE) [101] identifica a causa raiz de uma vulnerabilidade, como validação incorreta de entrada ou escrita fora dos limites de um *buffer*. O CWE tem aplicação no projeto de mitigações e contramedidas para vulnerabilidades e auxilia no direcionamento de esforços de desenvolvimento para prevenir a introdução de causas comuns de vulnerabilidades. CVEs também são frequentemente relacionados a CPEs (*Common Platform Enumeration*) [27], que identificam produtos e versões de *software* afetadas por uma vulnerabilidade. O CPE é útil para analistas de segurança que precisam identificar quais sistemas são afetados por uma vulnerabilidade.

Provas de conceitos e *exploits*. Bases de dados de *exploits* são repositórios de códigos que exercitam e demonstram a exploração de uma vulnerabilidade. Estas bases de dados são frequentemente utilizadas para comprovar a existência da vulnerabilidade, detectar sua presença em sistemas em operação e para avaliar a eficácia de contramedidas. Exemplos

[1] https://nvd.nist.gov, https://security.snyk.io e https://vuldb.com

[2] https://www.cisa.gov/coordinated-vulnerability-disclosure-process

de bases de dados de *exploits* incluem Exploit-DB, 0day.today, Packet Storm e Metasploit.[3] Uma limitação destas bases de dados é que elas disponibilizam provas de conceito, mas *exploits* utilizados para realizar ataques reais são muito mais complexos e raramente obtidos para estudo pela comunidade de segurança [71, 121].

Informações de rede. Diversas informações de rede são públicas e podem prover informações complementares relacionadas a vulnerabilidades. Por exemplo, registros de numeração coletam e disponibilizam informações sobre detentores de prefixos IP e nomes de domínio através dos protocolos WHOIS e RDAP [80].[4] Estes dados podem ser úteis para identificar, por exemplo, qual organização é responsável por um dispositivo afetado por uma vulnerabilidade. Registros de DNS e RDNS (resolução reversa de DNS), como coletados pela Rapid7,[5] podem ser utilizados para inferir o operador [98], localização [97], ou função de um dispositivo.

Varreduras de rede. Diversas ferramentas permitem a realização de varreduras de rede, e alguns serviços e entidades disponibilizam resultados de varreduras publicamente. Por exemplo, a Rapid7 disponibiliza coletas de varreduras TCP e UDP para portas comumente utilizadas para todo o espaço de endereçamento IPv4,[6] varreduras de scans para várias portas utilizadas por servidores HTTP e HTTPS,[7] e os certificados SSL recebidos de servidores HTTPS e executando outros protocolos.[8] Serviços como Shodan e Censys disponibilizam resultados dados similares para consulta [11, 44]. Estes dados podem ser utilizados para identificar dispositivos ativos em uma rede, identificar aplicações e versões de *software* em execução, e identificar vulnerabilidades conhecidas em dispositivos.

Monitoramento de redes sociais. Diversos trabalhos demonstram a utilidade de utilizar dados de redes sociais como Twitter e Reddit, bem como informações de sites da Dark Web e fórums de discussão para diversas inferências sobre vulnerabilidades. Por exemplo, a discussão de vulnerabilidades nestes meios é fortemente correlacionado à exploração das mesmas na Internet [6, 7]. Apesar de bases de dados com informações de redes sociais e fórums de discussão não estarem publicamente disponíveis, as informações *são* públicas e podem ser capturadas por sistemas de coleta. Um problema com estas bases de dados é que podem estar sujeitas a informações falsas ou manipuladas, o que exige validação ou uso cuidadoso em mecanismos de inferência.

A lista acima captura diferentes tipos de bases de dados, mas reforçamos que a construção de uma base exaustiva é impossível. Outras bases de dados podem estar disponíveis, ou podem ser coletadas ativamente com objetivos específicos por analistas ou pesquisadores. Por exemplo, o site Kaggle armazena diversas bases de dados, centenas delas relacionadas diretamente ou indiretamente a segurança de sistemas computacionais.[9] Discussões no GitHub frequentemente revelam informação relacionadas a vulnerabilidades antes mesmo destas serem públicadas em bases como o CVE [84].

2.3.1 Desafios. A criação e manutenção de bases de dados sobre vulnerabilidades, como as discutidas nesta seção, requer atenção e esforço constantes de peritos em diversas áreas, como

[3] https://www.exploit-db.com, https://0day.today, https://packetstormsecurity.com e https://www.metasploit.com
[4] https://irr.net/registry e https://www.caida.org/catalog/datasets/as-organizations
[5] https://opendata.rapid7.com/sonar.rdns_v2
[6] https://opendata.rapid7.com/sonar.tcp e https://opendata.rapid7.com/sonar.udp
[7] https://opendata.rapid7.com/sonar.http e https://opendata.rapid7.com/sonar.https
[8] https://opendata.rapid7.com/sonar.ssl e https://opendata.rapid7.com/sonar.moressl
[9] https://www.kaggle.com/datasets?search=security

compiladores, engenharia de software, redes de computadores, sistemas operacionais e arquitetura de computadores. Uma consequência da dificuldade de manter bases de dados sobre vulnerabilidades é a existência de inconsistências ou incompletude dos dados [9, 42, 67], o que dificulta a relação entre múltiplas bases de dados [42, 52, 171]. Esta limitação motiva esforços para identificar e padronizar termos técnicos em descrições de vulnerabilidades, como produtos, fabricantes, versões, entidades e causa raiz. Alguns trabalhos na literatura propõem abordagens com estes objetivos, por exemplo, utilizando processamento de texto via técnicas de aprendizado profundo [62, 151, 168]. Na próxima seção discutimos esforços de pesquisa na caracterização e previsão de propriedades de vulnerabilidades, com o potencial de aliviar a pressão sobre peritos na manutenção destas bases de dados.

Entidades podem submeter uma aplicação para se tornarem um CVE *Numbering Authority* (CNA). CNAs são responsáveis por atribuir identificadores CVE para vulnerabilidades descobertas em produtos e serviços específicos, levando a um aumento do número de vulnerabilidades catalogadas. CNAs incluem fabricantes de equipamentos, desenvolvedores de *software*, provedores de serviços, e organizações de segurança. Preocupações da comunidade incluem um aumento de CVEs sem impacto em segurança; por exemplo, o kernel do Linux adota uma política inclusiva[10] e gera CVEs mesmo quando não há comprovação de implicações de vulnerabilidade. Este aumento de CVEs complica e motiva a priorização de vulnerabilidades.

2.4 Modelagem de Vulnerabilidades

A priorização de vulnerabilidades requer um entendimento de suas características e propriedades. Diversos trabalhos propuseram o uso de diferentes técnicas como análise estática e dinâmica de código, sistemas inteligentes baseados em regras, processamento de linguagem natural, aprendizado de máquina e aprendizado profundo para prever propriedades de vulnerabilidades a partir de dados públicos. Estes modelos são úteis para acelerar o processo de caracterização de vulnerabilidades cada vez mais complexas, provendo entradas mais precisas para outras ferramentas, arcabouços e analistas de segurança, em particular para a priorização de vulnerabilidades. Nesta seção discutimos trabalhos que visam modelar propriedades de vulnerabilidades de software para fins de previsão e caracterização, que agrupamos em três classes que discutimos a seguir.

2.4.1 Propriedades de Vulnerabilidades. Diversos trabalhos tratam de prever propriedades de vulnerabilidades a partir de sua descrição e propriedades externas. Dentre as propriedades de vulnerabilidades frequentemente estudadas estão as métricas de confidencialidade, integridade e disponibilidade definidas pelo CVSS. Trabalhos podem focar em prever uma propriedade específica do CVSS como impacto em confidencialidade ou integridade [31, 46, 77, 87, 158, 167, 172], ou em prever várias propriedades conjuntamente [55, 59, 140]. A previsão de uma única métrica geralmente obtém precisão maior do que modelos que preveem várias propriedades e pode ser realizada com modelos mais simples. Porém, esta abordagem implica na execução de múltiplos modelos quando é necessário prever diferentes propriedades de uma vulnerabilidade, o que pode levar a inconsistências entre os modelos utilizados e acúmulo de erros. Nestes cenários, os modelos que preveem várias propriedades são preferíveis.

Outra frente de pesquisa é modelar a categoria de uma vulnerabilidade, por exemplo, se uma vulnerabilidade é causada por estouro de *buffer*, *cross-site scripting*, ou injeção SQL. Identificar a categoria de uma vulnerabilidade permite inferir causas, potenciais impactos, e estratégias de mitigação. A abordagem mais comum nessa frente é tentar prever a classificação CWE de uma

[10] https://docs.kernel.org/process/cve.html

vulnerabilidade [38, 119, 182], o que é desafiador devido à classificação hierárquica e em grão fino do CWE, que tem mais de 900 categorias. Alguns trabalhos contornam este desafio definindo tipos diferentes de vulnerabilidades em grão mais grosso que o CWE [105, 120].

Por fim, alguns trabalhos vão além das métricas clássicas do CVSS e buscam prever, por exemplo, se uma vulnerabilidade impacta aplicações Web [118]; os níveis de privilégio necessários para explorar uma vulnerabilidade em função dos diferentes sistemas operacionais onde pode se manifestar [3]; se uma vulnerabilidade é explorável apenas com acesso local ao dispositivo, via uma rede local, ou remotamente [33]; ou a consequência prática de uma vulnerabilidade, como acesso ao dispositivo, alteração de dados, ou negação de serviço [33].

2.4.2 Probabilidade de Exploração ao Longo do Tempo. Diversos trabalhos propõem modelos para prever se e quando uma probabilidade será explorada na prática. Essas informações têm aplicação direta na priorização de vulnerabilidades e direcionamento de esforços de analistas de segurança. Em particular, uma vulnerabilidade com alta probabilidade de ser explorada em um curto intervalo de tempo exige uma resposta imediata de equipes de segurança [85].

A probabilidade de uma vulnerabilidade ser explorada pode ser inferida a partir de sua descrição e metadados públicos [50, 72, 155]. Informações adicionais obtidas através de análise estática e dinâmica do código, por exemplo, fazendo análise de chamadas de sistema [178]; propriedades extraídas de executáveis [173]; ou falhas durante a execução do programa (*crashes*) [159] permitem a construção de modelos mais precisos. Neste contexto, o treino de modelos utilizando transferência de aprendizado obtido de vulnerabilidades conhecidas e bem caracterizadas para novas vulnerabilidades permite prever mais precisamente a probabilidade da nova vulnerabilidade ser explorada [176].

Outra frente de pesquisa é a modelagem do perfil temporal de exploração de uma vulnerabilidade, por exemplo, determinar quão rapidamente e por quanto tempo uma vulnerabilidade será explorada [22, 32]. Uma fração significativa de vulnerabilidades são exploradas antes de um CVSS ser assinalado; nestes cenários o monitoramento de redes sociais permite antecipar a exploração de uma vulnerabilidade e preparar analistas de segurança [32].

2.4.3 Esforço de Mitigação de Vulnerabilidades. Alguns trabalhos na literatura tentam prever a complexidade de se mitigar uma vulnerabilidade considerando sua descrição e propriedades do código [181]. Esta informação é útil na alocação de recursos de analistas de segurança, e de especial interesse para o nosso projeto considerando limitações de equipes sobrecarregadas. O trabalho desenvolvido por Othmane *et al.* [16, 17] utilizando bases de código e vulnerabilidades da empresa SAP[11] modela o tempo para a correção (*patching*) de vulnerabilidades por desenvolvedores. Os autores identificam 65 propriedades potencialmente relacionadas ao esforço de correção, e identificam que análise do código e dos componentes envolvidos são mais informativos do que a descrição da vulnerabilidade para a previsão do esforço de correção.

2.5 Priorização de Vulnerabilidades

A priorização de vulnerabilidades é uma área de pesquisa mais recente que as anteriores e não tão estudada. Uma possível explicação para isso é que apenas agora as limitações de equipes de segurança em lidar com a enorme quantidade de vulnerabilidades detectadas estão ficando claras [129].

Os primeiros trabalhos de priorização de vulnerabilidades propuseram ranqueamentos alternativos para substituir o CVSS, por exemplo, ponderando de forma diferente as propriedades de

[11] https://www.sap.com

vulnerabilidades [141]ou focando em um tipo específico de aplicação [116]. Estes sistemas não trazem muitos benefícios além do CVSS, pois geralmente não recebem atenção de analistas e nem integram informações complementares. Mais recentemente, o EPSS [73] provê uma nova métrica que integra informações adicionais e estima a probabilidade de uma vulnerabilidade ser explorada, o que, ao contrário de outras métricas de ranqueamento, tem significado concreto. Apesar de suas limitações, o CVSS ainda é utilizado em soluções para priorização de vulnerabilidades [125, 161], possivelmente impulsionado por sua ampla disponibilidade.

Trabalhos subsequentes propuseram avaliar o risco associado a vulnerabilidades de *software* estimando o possível impacto em objetivos de negócio como operação ou perda de receita [100, 160]. Estas abordagens, porém, ignoram outros danos resultantes de vulnerabilidades, como vazamento de dados, o que pode levar a uma discrepância entre o impacto estimado e o impacto real. Uma alternativa a esta abordagem proposta literatura é a estimativa de risco baseado na análise de vulnerabilidades em um dispositivo. Um desafio é estimar a importância ou relevância de um dispositivo, propriedades que dependem do contexto onde está implantado, das aplicações que executa, dos dados que armazena, dentre outros fatores. A risco relativo a um dispositivo precisa ser avaliado caso-a-caso [75]. Esta avaliação é dificultada quando informações incompletas estão disponíveis sobre um dispositivo [133].

O SSVC [142] introduz uma árvore de decisão que pode ser utilizada por entidades para identificar a severidade de uma vulnerabilidade e como proceder, considerando o contexto da entidade. Por exemplo, o SSVC considera o impacto técnico e de segurança causado por uma vulnerabilidade, e retorna recomendações de tratamento da vulnerabilidade. As recomendações variam dependendo do destinatário, por exemplo, os desenvolvedores da mitigação ou administradores de sistema que irão implantá-la. As ações recomendadas podem estar entre *deferir*, *agendar*, *resolver em paralelo* e *imediato*. O documento definindo o SSVC [144] levanta como pontos que podem ser melhorados no futuro a inclusão de mais etapas no processo de decisão, incluindo informações adicionais como formação dos analistas e refinamento da árvore de decisão.

Abordagens mais recentes combinam informações sofisticadas para realizar priorização de vulnerabilidades de forma mais precisa para diferentes entidades. Exemplos incluem o levantamento do impacto que diferentes tipos de falhas, como roubo de dados ou execução de código malicioso, podem ter em uma empresa [129]. Este levantamento pode ser realizado através de estudos de caso, onde analistas e gerentes respondem a questionários relativos a diferentes cenários de falha. Outros trabalhos propõem considerar as contramedidas disponíveis ou que podem ser adotadas por uma empresa, por exemplo, em função da capacitação da equipe de analistas ou serviços pagos [154]. Outras soluções similares propõem prever o tempo necessário para o desenvolvimento e aplicação de contramedidas contra vulnerabilidades no processo de priorização [134].

Por fim, dois desafios identificados na priorização de vulnerabilidades são a necessidade de reduzir ou controlar o nível de subjetividade das prioridades associadas a vulnerabilidades [147]. Por fim, contratar analistas de segurança ou soluções comerciais de monitoramento de vulnerabilidades implica em um custo elevado [124].

2.6 Tendências Emergentes

Nesta seção discutimos tendências emergentes que identificamos na literatura, com foco especial para aspectos relevantes à priorização de vulnerabilidades.

2.6.1 Aprendizado de Máquina e Inteligência Artificial. O avanço das técnicas de aprendizado de máquina e recente adoção de aprendizado profundo perpassa todos os contextos analisados relativos ao gerenciamento de vulnerabilidades de *software*: identificação e monitoramento de

vulnerabilidades; construção de modelos de previsão e caracterização de vulnerabilidades; padronização e enriquecimento de dados; e modelos de priorização. Um efeito desta tendência é a extensão e aprimoramento de sistemas baseados em abordagens clássicas como árvores de decisão e regras [35, 136, 150] por sistemas que utilizam propriedades (*features*) e relações identificadas por modelos de aprendizado de máquina ou soluções de inteligência artificial [56, 89, 95]. Soluções recentes utilizam redes neurais em grafos (GNNs) [30, 31], redes neurais profundas (DNNs) [91, 180], redes neurais recorrentes com atenção (LSTMs) [59, 123], *Extreme Gradient Boosting* (XGBoost) [164], dentre outras técnicas. Diversos trabalhos comparam técnicas clássicas de aprendizado de máquina com técnicas mais modernas e comprovam o ganho de precisão obtido com a evolução das metodologias [9, 63, 82, 164, 182].

O uso de inteligência artificial permite a extração automática de propriedades em múltiplos níveis de abstração [88] e com maior generalidade [93]. Esta tendência tem o potencial de remover um gargalo importante no combate a vulnerabilidades de *software*: a alta demanda por uma quantidade escassa de especialistas para a caracterização e entendimento de vulnerabilidades [127]. A inteligência artificial pode reduzir a incidência de erros quando comparado a humanos ou permitir a identificação de propriedades que não seriam consideradas nem por especialistas [92].

Um aspecto negativo de técnicas de aprendizado profundo é a interpretabilidade limitada, o que pode comprometer a utilidade de modelos para analistas, e potenciais tendencias (*biases*) intrínsecas, difíceis de identificar e corrigir.

2.6.2 Agregação de Dados. Diversos trabalhos demonstram ganhos de precisão combinando dados de diferentes fontes. Por exemplo, alguns trabalhos encontraram que mensagens em redes sociais como Twitter, Reddit e Dark Web estão relacionados com alterações de código no GitHub [68]; diversas vulnerabilidades são publicadas no Twitter antes mesmo de serem inseridas em bases de vulnerabilidades [30, 121]; dados obtidos de falhas de execução de programas *crashes* são úteis em preditores da probabilidade de que uma falha será explorada [159, 173]. Ataques sofisticados como o ataque cibernético contra a malha elétrica da Ucrânia [47] e o cibergolpe contra o Banco de Bangladesh [66] dependem da captura de dados elaborados sobre os alvos.

Soluções de segurança utilizando reconhecimento de padrões e aprendizado de máquina combinando múltiplas bases de dados proveem alternativas para a automação e melhoria de eficiência para a identificação, modelagem e priorização de vulnerabilidades [35, 56, 95]. Os algoritmos de aprendizado de máquina conseguem aprender padrões latentes e abstratos de código vulnerável, de forma mais geral do que sistemas baseados em regras. Por exemplo, redes neuronais e aprendizado profundo conseguem definir propriedades (*features*) de forma automática [165]. Propriedades definidas por técnicas de aprendizado de máquina conseguem extrair informações mais representativas e ricas, melhor sumarizando múltiplas bases de dados [92]. Além disso, técnicas de aprendizado profundo identificam métricas complexas e não intuitivas que provavelmente não seriam definidas por analistas e peritos, mas que mesmo assim contribuem para a modelagem e priorização de vulnerabilidades [88, 93].

Usar bases de dados complementares é geralmente positivo, mas o acúmulo de dados pode ser danoso caso os dados fiquem desatualizados. Há uma demanda constante por analistas e peritos na criação e manutenção dos dados. Outro desafio na manutenção de bases de dados é a deriva de terminologia que ocorre ao longo do tempo. Novas vulnerabilidades podem ser descritas por terminologia diferente de vulnerabilidades similares identificadas anteriormente [87]. Dados de uma mesma vulnerabilidade podem estar disponíveis em múltiplas bases de dados, mas identificar e desambiguar dados de uma vulnerabilidade em diferentes bases é desafiador devido à ausência de uma chave global [42]. Por fim, o agregamento de múltiplas bases de dados incorre

eu aumento de custo e complexidade de processamento e armazenamento de dados. Na direção de aliviar estes problemas, alguns trabalhos visam selecionar o menor número de propriedades (*feature selection*) suficientes para treinar modelos precisos [81] ou aplicar técnicas de redução de dimensionalidade [99] para reduzir a quantidade de dados e CPU para treino e execução de modelos.

2.6.3 Aumento de Vulnerabilidade em Dispositivos Industriais e de IoT. Diversas fontes relatam um aumento significativo do número de vulnerabilidades em dispositivos industriais e de IoT [110, 139, 183]. Vulnerabilidades em dispositivos industriais e de IoT são difíceis de mitigar devido a diversos fatores práticos como a impossibilidade ou dificuldade de aplicação de atualizações; indisponibilidade de atualizações; dificuldade de acesso físico a equipamentos remotos; alto custo e risco associados ao desligamento e reinicialização para instalação de uma atualização [4, 156, 166]. Além disso, dispositivos IoT frequentemente têm baixo custo, e sua segurança é frequentemente colocada em segundo plano. Nem mesmo dispositivos IoT destinados a garantir a segurança física de usuários, como travas elétricas e câmeras de vigilância, estão livres de vulnerabilidades [23, 34, 110]. Este problema é agravado devido ao grande número de dispositivos IoT, que podem ser utilizados para montar ataques massivos de negação de serviço [39, 70, 76].

2.6.4 Importância da Atuação de Analistas de Segurança. Apesar de ferramentas estarem utilizando cada vez mais dados (Seção 2.3) e modelos de previsão e aprendizado cada vez mais sofisticados (Seção 2.4), diversos trabalhos reforçam a importância de informações obtidas de especialistas e da aplicação de inteligência humana para guiar o desenvolvimento e uso de diferentes soluções no processo de combate a vulnerabilidades [163]. Por exemplo, análise de dados em redes sociais como o Twitter, Reddit, fóruns de discussão e Dark Web permitem prever um aumento na probabilidade de exploração de vulnerabilidades [121] ou quando uma vulnerabilidade está sendo visada para ataque [32, 108]; validar e eficácia de *exploits* públicos [149]; e estimar atividade em repositórios de código aberto [68]; construir bases de dados para treino de modelos supervisionados [169]. Diversas vulnerabilidades são publicadas no Twitter ou discutidas no GitHub antes de serem adicionadas a bases de vulnerabilidades [30, 84, 121]. Por fim, apesar de todo o esforço de pesquisa na modelagem e previsão de propriedades de vulnerabilidades, a manutenção de bases de dados do CVE e cálculo do CVSS continua sendo realizada manualmente por peritos.

3 ANÁLISE DE SOLUÇÕES EXISTENTES

Nesta seção discutimos ferramentas de código aberto para identificação e gerenciamento de vulnerabilidades de *software* (Seção 3.1) bem como soluções de gerenciamento e priorização de vulnerabilidades (Seção 3.2).

3.1 Ferramentas de Código Aberto

Nesta seção discutimos ferramentas de código aberto para levantamento de informações relacionadas a vulnerabilidades no contexto de uma entidade. Estas ferramentas são úteis para identificar e caracterizar vulnerabilidades em sistemas computacionais.

Captura de tráfego de rede. Ferramentas como tcpdump, WinDump, Wireshark, nfdump[12] filtram, capturam e analisam pacotes de rede. Estas ferramentas proveem dados de baixo nível que são úteis para análises novas não disponíveis em outros sistemas.

[12]https://www.tcpdump.org, https://www.winpcap.org, https://www.wireshark.org e https://github.com/phaag/nfdump

Sistemas de detecção de intrusão de rede (NIDS). Ferramentas como Nagios, Snort, Suricata e Zeek[13] estendem ferramentas de captura de tráfego da rede com sistemas de verificação de regras ou detecção de anomalias para identificar tráfego malicioso ou dispositivos com comportamento suspeito em uma rede.

Sistemas de detecção de intrução de dispositivo (HIDS). Ferramentas como OSSEC e Wazuh[14] monitoram processos, tráfego de rede, arquivos e diretórios para identificar mudanças não autorizadas e comportamento anômalo.

Sistemas de varredura de rede. Ferramentas como Masscan, Zmap, ZDNS e Zgrab[15] realizam varreduras de portas e serviços em dispositivos de rede para identificar portas abertas e coletar informações de serviços em execução. Ferramentas como Nmap, OpenVAS e Nuclei[16] realizam varreduras mais profundas, realizando sequências de requisições seguindo protocolos de aplicação e um conjunto de regras pré-definidas. Estas ferramentas analisam respostas recebidas para identificar propriedades de dispositivos e aplicações, incluindo a presença de vulnerabilidades de *software*. Outras ferramentas fazem varreduras de aplicações e protocolos específicos, como o OWASP ZAP, Burp e Wapiti,[17] que focam em aplicações Web; o SQLmap,[18] que foca em vulnerabilidades de *software* relacionadas a bancos de dados; o Akto,[19] que foca em vulnerabilidades em APIs Web.

Sistemas de verificação e teste de configuração. Alguns sistemas tratam de testar a configuração de um software. Este tipo de ferramenta é útil na verificação de sistemas complexos. Exemplos de ferramentas incluem o BloodHound,[20], que enumera possíveis ataques contra o sistema de gerência de identidade e acesso Active Directory; o ScoutSuite,[21] que verifica a configuração de serviços em núvem; e o Lynis,[22] que verifica a configuração de sistemas operacionais.

Descobrimento de informações. Diversas ferramentas coletam informações adicionais sobre uma rede, o que é particularmente útil para identificar a superfície de vulnerabilidade de uma rede ou entidade. Exemplos de aplicações incluem o netdiscover,[23] que utiliza análise passiva e sondagem ativa de pacotes ARP para descobrir dispositivos em uma rede; o bettercap,[24] que detecta dispositivos realizando varreduras WiFi, Bluetooth e IP; o Subfinder e o sublist3r,[25] que descobrem subdomínios de um domínio; bem como o Amass e o theHarverster,[26] que descobrem informações sobre domínios a partir de dados públicos e motores de busca.

Uma enorme gama de ferramentas comerciais estão disponíveis para realização de análise de segurança. Muitas das ferramentas acima possuem versões comerciais que oferecem funcionalidades

[13]https://www.nagios.org, https://www.snort.org, https://suricata.io e https://zeek.org
[14]https://www.ossec.net e https://wazuh.com
[15]https://github.com/robertdavidgraham/masscan e https://zmap.io
[16]https://nmap.org, https://www.openvas.org e https://projectdiscovery.io/nuclei
[17]https://www.zaproxy.org, https://portswigger.net/burp e https://wapiti-scanner.github.io
[18]https://sqlmap.org
[19]https://www.akto.io
[20]https://github.com/BloodHoundAD/BloodHound
[21]https://github.com/nccgroup/ScoutSuite
[22]https://cisofy.com/lynis
[23]https://github.com/netdiscover-scanner/netdiscover
[24]https://www.bettercap.org
[25]https://github.com/projectdiscovery/subfinder e https://github.com/aboul3la/Sublist3r
[26]https://github.com/owasp-amass/amass e https://github.com/laramies/theharvester

adicionais, como um conjunto mais completo de regras para detecção de vulnerabilidades, implementação de varreduras complementares, suporte técnico, atualizações automáticas e integração com outras ferramentas.

3.2 Soluções para Gerência e Priorização de Vulnerabilidades

Diversas empresas e projetos de código aberto proveem serviços e ferramentas para o monitoramento de vulnerabilidades. Nesta seção focamos em serviços de inteligência contra ameaças (*threat intelligence*), discutindo funcionalidades comuns e abordagens de implementação.

Soluções de código aberto. Diversos projetos de *software* livre disponibilizam sistemas para gerência de vulnerabilidades. Exemplos incluem o DefectDojo, ArcherySec, reNgine, Osmedeus, IVRE, Faraday e Crossfeed.[27] Estas soluções utilizam uma arquitetura modular (*plugins*), onde cada módulo é responsável por realizar uma tarefa específica, como a varredura de vulnerabilidades em um sistema. Os módulos em geral executam ferramentas de código aberto, como as descritas na Seção 3.1, processam a saída da ferramenta para converter em um formato padrão, armazenam as informações em um banco de dados, e proveem uma interface Web para visualização e inspeção das vulnerabilidades. Estes sistemas são úteis para organizar e consolidar informações sobre vulnerabilidades. Porém, atualmente nenhuma das soluções de código aberto que investigamos tem capacidade de priorizar vulnerabilidades.

As abordagens de implementação destas ferramentas são diversas. Por exemplo, o DefectDojo agrega resultados de múltiplas ferramentas de monitoramento e armazena as vulnerabilidades encontradas em um banco de dados utilizando uma estrutura compartilhada (chamada Finding) que possui uma multitude de campos, a maioria deles pertinentes a um pequeno subconjunto dos módulos disponíveis, levando a desperdício de recursos. Outros sistemas utilizam estruturas específicas para cada ferramenta de monitoramento, o que evita desperdício de recursos, mas implica maior complexidade para manutenção e extensão.

Soluções comerciais. A priorização de vulnerabilidades é um desafio complexo que requer a combinação de informações de diversas fontes e que pode ser abordada de diferentes formas. Não surpreendentemente, inúmeras ferramentas comerciais existem para detecção e gerência de vulnerabilidades. Este é um mercado em crescimento, com uma taxa de crescimento anual entre 2024–2028 estimada em 10.56%.[28] Muitas empresas apresentam soluções comerciais para a detecção, gerenciamento e priorização de vulnerabilidades, também chamadas de inteligência contra ameaças (*threat intelligence*). Soluções comerciais são desenvolvidas por empresas de grande porte, como o IBM X-Force, Cisco Vulnerability Management, AWS GuardDuty e AT&T OSSIM.[29] *Startups* de médio e pequeno porte

[27] https://www.defectdojo.com, https://www.archerysec.com, https://rengine.wiki/, https://www.osmedeus.org, https://ivre.rocks, https://faradaysec.com e https://docs.crossfeed.cyber.dhs.gov/

[28] https://www.statista.com/outlook/tmo/cybersecurity/worldwide

[29] https://www.ibm.com/x-force, https://www.cisco.com/c/en/us/products/security, https://aws.amazon.com/guardduty e https://cybersecurity.att.com/products/ossim

como Tenable, Greenbone, Rapid7, Snyk, Symantec DeepSight[30] e várias outras[31] também oferecem soluções comerciais para gerência de vulnerabilidades.
Diversas soluções comerciais são ofertadas por empresas que mantém ferramentas e sistemas de código aberto. Exemplos incluem a Greenbone, que desenvolve o OWASP ZAP, a Tenable, que desenvolve o Nexus, e a Rapid7, que desenvolve o Metasploit. Estas empresas oferecem serviços de monitoramento e gerência de vulnerabilidades utilizando versões estendidas das ferramentas, geralmente capaz de detectar mais vulnerabilidades e com funcionalidades de gerência adicionais. Outras empresas proveem serviços de gerência de vulnerabilidade desenvolvidos sobre versões livres, por exemplo, o DefectDojo e Faraday possuem versões comerciais com funcionalidades adicionais como execução a partir de nuvem (em vez de auto-hospedado), integrações com sistemas de tíquete, controle de acesso, atualizações automáticas e agendamento de varredura não disponíveis nas versões de código aberto.

Abaixo discutimos três dimensões onde soluções de gerenciamento de vulnerabilidades apresentam diferentes abordagens e funcionalidades. No contexto comercial, estas diferenças são utilizadas para posicionar a solução desenvolvida no mercado relativo aos concorrentes. Infelizmente, uma comparação aprofundada de soluções comerciais é impossível, pois o acesso às ferramentas é restrito a clientes. Revisamos as informações disponíveis nas páginas destas soluções, mas as descrições nas páginas são vagas. A seguir apresentamos diferenças gerais das soluções comerciais e de código aberto avaliadas.

Gerência de vulnerabilidades. Muitas soluções incluem um sistema de gerência implementado como uma interface Web onde vulnerabilidades são registradas e acompanhadas até a resolução. As funcionalidades disponíveis no sistema de gerência varia entre aplicações. O ArcherySec, por exemplo, busca identificar vulnerabilidades encontradas por diferentes ferramentas comparando o campo de descrição. Esta é uma abordagem simples que pode levar a falsos positivos e falsos negativos, mas permite a consolidação de informações de sobre uma vulnerabilidade obtida de diferentes fontes. Soluções permitem o agrupamento de vulnerabilidades em diferentes granularidades, como dispositivos, máquinas virtuais, contêineres e aplicações. Por fim, soluções podem ter diferentes funcionalidades para acompanhar o processo de resolução de uma vulnerabilidade, por exemplo, armazenando anotações capturando informações sobre medidas tomadas por cada analista.

Pesquisa em detecção de vulnerabilidades. Algumas empresas como Trendmicro, Tenable e Fortinet mantém equipes de analistas que trabalham efetivamente no descobrimento de novas vulnerabilidades (*zero-day exploits*) para estender suas ferramentas. O esforço em pesquisar por novas vulnerabilidades também é realizado por pesquisadores e analistas independentes, que frequentemente contribuem extensões para soluções de monitoramento de código aberto.

Priorização de vulnerabilidades. Diversas soluções de gerenciamento permitem a priorização de vulnerabilidades. Abordagens incluem priorização manual pelo analista, priorização manual pela empresa em soluções comerciais (*security advisories*) e priorização utilizando métricas de severidade como o (CVSS) ou a probabilidade de exploração (EPSS).

[30]https://www.tenable.com, https://www.greenbone.net, https://www.rapid7.com, https://snyk.io, https://docs.broadcom.com/doc/deepsight-intelligence-ds

[31]https://www.recordedfuture.com, https://vulners.com, https://www.fortiguard.com, https://www.openwall.com, https://www.skyboxsecurity.com, https://www.trendmicro.com, https://www.flexera.com/products/software-vulnerability-manager, https://www.secureworks.com, https://www.reversinglabs.com, https://immunityinc.com e https://www.runzero.com, https://www.intruder.io

Como identificado na Seção 2.6, algumas soluções comerciais já disponibilizam assistentes de inteligência artificial. Considerando que não temos acesso às soluções propostas, é difícil avaliar as capacidades e eficácia destes assistentes. No momento, nenhum dos projetos de código aberto estudado apresentam assistentes de inteligência artificial.
Soluções comerciais possuem serviços personalizados, que podem incluir a priorização de vulnerabilidades baseada em informações sobre uma entidade cliente. Porém, nosso estudo das soluções existentes não levantou nenhuma solução automática para priorização de vulnerabilidades considerando o contexto da empresa como pretendemos desenvolver.

Nesta seção não discutimos inúmeras ferramentas e serviços que abordam outros desafios relacionados a segurança. Por exemplo, a Proofpoint e a NetCraft[32] oferecem soluções de segurança para identificar ameaças de *phishing* e *malware* em e-mails, documentos e sítios Web. A Cloudflare, a Akamai[33] e vários provedores de computação em núvem oferecem serviços de proteção contra ataques *DDoS* e *firewall* de aplicação. O BuiltWith[34] identifica tecnologias utilizadas em sítios Web. Tripwire e CrowdStrike[35] monitoram dispositivos para detectar comportamento anônomalo ou mudanças não autorizadas. O Ghidra[36] é uma ferramenta para engenharia reversa de código e realização de análises estáticas e dinâmicas. Estas e outras soluções são complementares aos serviços de inteligência discutidos nesta seção e contribuem para a segurança de sistemas computacionais, mas estão fora do escopo deste relatório.

4 CONSIDERAÇÕES FINAIS

Neste relatório apresentamos uma revisão bibliográfica sobre gerenciamento de vulnerabilidades de software. Trabalhos anteriores já propuseram soluções para priorização de vulnerabilidades, por exemplo, identificando quais vulnerabilidades têm maior probabilidade de serem exploradas. A partir da revisão realizada, identificamos oportunidades para aprimorar técnicas existentes de classificação de vulnerabilidades considerando metadados sobre a instituição monitorada, como: área de atuação, os sistemas que opera, os dados que armazena, o perfil e expectativa dos clientes, bem como a experiência e formação dos funcionários (desenvolvedores, operadores e analistas de segurança). Pretendemos também utilizar metadados sobre as vulnerabilidades e os sistemas monitorados obtidos de bases públicas. Avaliaremos e aprimoraremos os algoritmos de priorização desenvolvidos no projeto em cenários realistas através da colaboração com a startup parceira, utilizando dados reais e validando os resultados com os times das instituições auditadas.

A priorização de vulnerabilidades tem aplicação direta em diversos contextos. Em particular, a priorização de vulnerabilidades é uma fonte adicional de informação para operadores, com potencial de reduzir custos operacionais e permitir realocação estratégica de recursos humanos. Argumentamos que estes são objetivos concretos e práticos, que podem impulsionar a ideação de novos produtos, práticas, métodos, processos ou serviços na área de segurança.

4.1 Implicações e Recomendações Estratégicas

Considerando os desafios e oportunidades discutidos, apresentamos as seguintes recomendações para o desenvolvimento de soluções para priorização de vulnerabilidades.

[32]https://www.proofpoint.com e https://www.netcraft.com
[33]https://www.cloudflare.com e https://www.akamai.com
[34]https://builtwith.com
[35]https://www.tripwire.com e https://www.crowdstrike.com
[36]https://ghidra-sre.org/

Agregação de dados e combinação ferramentas. Projetos devem combinar diversas fontes de dados públicas sobre vulnerabilidades disponíveis, como as bases do CVE, CWE, CPE, CVSS, EPSS, Exploit DB, Metasploit, e Canvas Exploitation Framework. Estas bases de dados podem ser complementadas com dados de rede. No contexto de auditoria de uma instituição ou empresa, melhor cobertura e informações complementares podem ser obtidas através da combinação de diversas ferramentas de código aberto, como OWASP ZAP, OpenVAS, Wapiti, Nuclei, Akto, Nmap e SQLmap, bem como serviços de monitoramento como Shodan e Censys.

Curadoria de dados. A agregação de bases de dados adicionais e utilização de mais ferramentas provê mais oportunidades para extração de informação útil. Porém, extrair informações úteis requer padronização, normalização e verificação dos dados bem como identificação de relacionamentos entre as bases. Sugerimos o desenvolvimento de técnicas robustas para encontrar equivalências e desambiguar vulnerabilidades encontradas por diferentes ferramentas, potencialmente utilizando modelos de inteligência artificial que possuem eficácia superior para os dados ruidosos sobre vulnerabilidades.

Transferência de aprendizado. Uma abordagem já utilizada em alguns trabalhos que consideramos promissora para este projeto é o treino de modelos em dados públicos de grande escala, como a Wikipedia ou Projeto Gutemberg. Os modelos treinados nestes conjuntos podem ser refinados posteriormente para uso em contextos de segurança utilizando técnicas de transferência de aprendizado [176]. Com isso podemos contornar o problema do desbalanceamento e escassez de dados em bases de dados sobre vulnerabilidades como relatórios gerados por ferramentas de varredura como OWASP e OpenVAS.

Integração do contexto de entidades e analistas. As vulnerabilidades que mais requerem esforços de analistas nem sempre são as vulnerabilidades com maior severidade ou classificadas como mais importantes em bases públicas como o CVSS e ranques "top 10" [86]. Soluções de segurança devem integrar conhecimento específico do contexto de uma entidade e seus analistas. Por exemplo, soluções devem considerar a formação e experiência de analistas e sua proficiência em lidar com diferentes tipos de vulnerabilidades, bem como a relevância de uma vulnerabilidade no contexto de uma entidade. Acreditamos que com estas informações é possível adequar melhor a priorização de vulnerabilidades para cada entidade e equipe.

Avaliação em cenários realistas. A avaliação de soluções de segurança é essencial para quantificar sua eficácia. Porém, avaliar soluções de segurança em cenários realistas é desafiador. A maioria dos estudos revisados avaliaram soluções desconsiderando diversos fatores relevantes para a implantação das soluções propostas em ambientes de produção. Para trabalhos futuros, esforços devem ser colocados na avaliação realista, preferencialmente de sistemas em operação, com vulnerabilidades reais, e acessíveis por possíveis adversários.

REFERÊNCIAS

[1] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proc. ACM IMC*, 2006.

[2] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha. Cyber Deception: Virtual Networks to Defend Insider Reconnaissance . In *Proc. ACM CCS International Workshop on Managing Insider Security Threats*, 2016.

[3] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, and E. I. Tatli. Automated Generation of Attack Graphs Using NVD. In *Conference on Data and Application Security and Privacy*, 2018.

[4] M. Alanazi, A. Mahmood, and M. J. M. Chowdhury. Scada Vulnerabilities and Attacks: A Review of the State-of-the-Art and Open Issues. *Computers and Security*, 125:103028, 2023.

[5] S. M. Albladi and G. R. Weir. User Characteristics that Influence Judgment of Social Engineering Attacks in Social Networks. *Human-centric Computing and Information Sciences*, 8(1):1–24, 2018.

[6] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian. Proactive Identification of Exploits in the Wild through Vulnerability Mentions Online. In *International Conference on Cyber Conflict*, 2017.

[7] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian. *Patch Before Exploited: An Approach to Identify Targeted Software Vulnerabilities*. 2019.

[8] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai Botnet. In *Proc. USENIX Security*, 2017.

[9] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen. Cleaning the NVD: Comprehensive Quality Assessment, Improvements, and Analyses. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4255–4269, 2021.

[10] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, and D. Brumley. Automatic Exploit Generation. *Commun. ACM*, 57(2):74–84, 2014.

[11] M. Bada and I. Pete. An Exploration of the Cybercrime Ecosystem around Shodan. In *International Conference on Internet of Things: Systems, Management and Security*, 2020.

[12] P. Bajpai, A. K. Sood, and R. J. Enbody. The art of Mapping IoT Devices in Networks. *Network Security*, 2018(4):8–15, 2018.

[13] R. J. Barnett and B. Irwin. Towards a Taxonomy of Network Scanning Techniques. In *Proc. Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, 2008.

[14] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In *Proc. IEEE Symposium on Security and Privacy*, 2010.

[15] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman. Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software. In *IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016.

[16] L. Ben Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, and A. D. Brucker. Time for Addressing Software Security Issues: Prediction Models and Impacting Factors. *Data Science and Engineering*, 2:107–124, 2017.

[17] L. Ben Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, A. D. Brucker, and P. Miseldine. Factors Impacting the Effort Required to Fix Security Vulnerabilities: An Industrial Case Study. In *International Conference on Information Security*, 2015.

[18] R. Beverly. A Robust Classifier for Passive TCP/IP Fingerprinting. In *Proc. PAM*, 2004.

[19] R. Beverly, A. Berger, and G. Xie. Primitives for Active Internet Topology Mapping: Toward High-Frequency Characterization. In *Proc. ACM IMC*, 2010.

[20] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Surveying Port Scans and Their Detection Methodologies. *The Computer Journal*, 54(10):1565–1581, 2011.

[21] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke. Fingerprinting Software-Defined Networks. In *Proc. IEEE ICNP*, 2015.

[22] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2010.

[23] C. Caballero-Gil, R. Álvarez, C. Herná ndez Goya, and J. Molina-Gil. Research on Smart-Locks Cybersecurity and Vulnerabilities. *Wireless Networks*, pages 1–13, 2023.

[24] C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of High-Coverage tests for complex systems programs. In *Proc. USENIX OSDI*, 2008.

[25] Y. Cai, H. Yun, J. Wang, L. Qiao, and J. Palsberg. Sound and Efficient Concurrency Bug Prediction. In *Proc. ACM European Software Engineering Conference*, 2021.

[26] W. Chang, B. Streiff, and C. Lin. Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis. In *Proc. ACM CCS*, 2008.

[27] B. A. Cheikes, B. A. Cheikes, K. A. Kent, and D. Waltermire. *Common Platform Enumeration: Naming Specification Version 2.3*. US Department of Commerce, National Institute of Standards and Technology, 2011.

[28] D. Chen, Y.-d. Zhang, W. Wei, S.-x. Wang, R.-b. Huang, X.-l. Li, B.-b. Qu, and S. Jiang. Efficient Vulnerability Detection Based on an Optimized Rule-Checking Static Analysis Technique. *Frontiers of Information Technology*

and Electronic Engineering, 18(3):332–345, 2017.

[29] H. Chen, S. Guo, Y. Xue, Y. Sui, C. Zhang, Y. Li, H. Wang, and Y. Liu. MUZZ: Thread-Aware Grey-Box Fuzzing for Effective Bug Hunting in Multithreaded Programs. In *Proc. USENIX Security*, 2020.

[30] H. Chen, J. Liu, R. Liu, N. Park, and V. Subrahmanian. VASE: A Twitter-Based Vulnerability Analysis and Score Engine. In *International Conference on Data Mining*, 2019.

[31] H. Chen, J. Liu, R. Liu, N. Park, and V. Subrahmanian. VEST: A System for Vulnerability Exploit Scoring and Timing. In *International Joint Conference on Artificial Intelligence*, 2019.

[32] H. Chen, R. Liu, N. Park, and V. Subrahmanian. Using Twitter to Predict When Vulnerabilities Will Be Exploited. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2019.

[33] Z. Chen, Y. Zhang, and Z. Chen. A Categorization Framework for Common Computer Vulnerabilities and Exposures. *The Computer Journal*, 53(5):551–580, 2010.

[34] A. Costin. Security of CCTV and Video Surveillance Systems: Threats, Vulnerabilities, Attacks, and Mitigations. In *Proc. International Workshop on Trustworthy Embedded Devices*, 2016.

[35] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang. Data-Driven Cyber Security in Perspective—Intelligent Traffic Analysis. *IEEE Transactions on Cybernetics*, 50(7):3081–3093, 2020.

[36] I. Cunha, R. Teixeira, D. Veitch, and C. Diot. DTRACK: A System to Predict and Track Internet Path Changes. *IEEE/ACM Trans. Netw.*, 22(4):1025–1038, 2014.

[37] A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescapé. Analysis of a /0 Stealth Scan from a Botnet. In *Proc. ACM IMC*, 2012.

[38] S. S. Das, E. Serra, M. Halappanavar, A. Pothen, and E. Al-Shaer. V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. In *International Conference on Data Science and Advanced Analytics*, 2021.

[39] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi. Analysis of DDoS-Capable IoT Malwares. In *Proc. Federated Conference on Computer Science and Information Systems*, 2017.

[40] S. de Smale, R. van Dijk, X. Bouwman, J. van der Ham, and M. van Eeten. No One Drinks From the Firehose: How Organizations Filter and Prioritize Vulnerability Information. In *IEEE Symposium on Security and Privacy*, 2023.

[41] M. De Vivo, E. Carrasco, G. Isern, and G. O. De Vivo. A Review of Port Scanning Techniques. *SIGCOMM Comput. Commun. Rev.*, 29(2):41–48, 1999.

[42] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports. 2019.

[43] A. Doupé, M. Cova, and G. Vigna. Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In *International Conference on Detection of Intrusions and Malware , and Vulnerability Assessment*, 2010.

[44] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A Search Engine Backed By Internet-Wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 542–553, 2015.

[45] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. 2013.

[46] C. Elbaz, L. Rilling, and C. Morin. Fighting N-Day Vulnerabilities with Automated CVSS Vector Prediction at Disclosure. In *International Conference on Availability, Reliability and Security*, 2020.

[47] Electricity Information Sharing and Analysis Center. Analysis of the Cyber Attack on the Ukrainian Power Grid. Technical report, SANS Industrial Control System, 2016.

[48] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. *SIGOPS Oper. Syst. Rev.*, 35(5):57–72, 2001.

[49] A. Ethembabaoglu, R. van Wegberg, Y. Zhauniarovich, and M. van Eeten. The unpatchables: Why municipalities persist in running vulnerable hosts. 2024.

[50] Y. Fang, Y. Liu, C. Huang, and L. Liu. FastEmbed: Predicting Vulnerability Exploitation Possibility Based on Ensemble Machine Learning Algorithm. *PLoS One*, 15(2), 2020.

[51] S. Figueroa-Lorenzo, J. Añorga, and S. Arrizabalaga. A Survey of IIoT Protocols: A Measure of Vulnerability Risk Analysis Based on CVSS. *ACM Computing Surveys*, 53(2):1–53, 2020.

[52] J. R. Finkel, T. Grenager, and C. D. Manning. Incorporating Non-Local Information into Information Extraction Systems by Gibbs Sampling. In *Nnual Meeting of the Association for Computational Linguistics*, pages 363–370, 2005.

[53] V. Ganesh, T. Leek, and M. Rinard. Taint-Based Directed Whitebox Fuzzing. In *Proc. IEEE International Conference on Software Engineering*, 2009.
[54] Gartner. Gartner Forecasts Worldwide Information Security Spending to Exceed $124 Billion in 2019, 2018. https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts-worldwide-information-security-spending-to-exceed-124-billion-in-2019.
[55] M. Gawron, F. Cheng, and C. Meinel. Automatic vulnerability classification using machine learning. In *International Conference Risks and Security of Internet and Systems*, 2018.
[56] S. M. Ghaffarian and H. R. Shahriari. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Computing Surveys*, 50(4), 2017.
[57] P. Godefroid, M. Y. Levin, D. A. Molnar, et al. Automated Whitebox Fuzz Testing. In *Proc. NDSS*, 2008.
[58] S. Gong, D. Altinbüken, P. Fonseca, and P. Maniatis. Snowboard: Finding Kernel Concurrency Bugs Through Systematic Inter-Thread Communication Analysis. In *Proc. ACM SOSP*, 2021.
[59] X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han. Joint Prediction of Multiple Vulnerability Characteristics through Multi-Task Learning. In *International Conference on Engineering of Complex Computer Systems*, 2019.
[60] D. Gonzalez, H. Hastings, and M. Mirakhorli. Automated Characterization of Software Vulnerabilities. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019.
[61] A. Gosain and G. Sharma. A Survey of Dynamic Program Analysis Techniques and Tools. In *International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA)*, 2015.
[62] H. Guo, Z. Xing, and X. Li. Predicting Missing Information of Key Aspects in Vulnerability Reports, 2020.
[63] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In *International Conference on Software Maintenance and Evolution*, 2017.
[64] N. A. Hassan and R. Hijazi. Technical Footprinting. *Open Source Intelligence Methods and Tools: A Practical Guide to Online Intelligence*, pages 313–339, 2018.
[65] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch. Spoki: Unveiling a New Wave of Scanners Through a Reactive Network Telescope. pages 431–448, 2022.
[66] J. A. Hill. SWIFT Bank Heists and Article 4A. *Journal of Consumer and Commercial Law*, 22(1):25–30, 2018.
[67] D. Hommersom, A. Sabetta, B. Coppola, D. D. Nucci, and D. A. Tamburri. Automated Mapping of Vulnerability Advisories onto Their Fix Commits in Open Source Repositories. *ACM Transactions on Software Engineering and Methodology*, 2024.
[68] S. Horawalavithana, A. Bhattacharjee, R. Liu, N. Choudhury, L. O. Hall, and A. Iamnitchi. Mentions of Security Vulnerabilities on Reddit, Twitter and GitHub . In *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019.
[69] I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro. Current State of Research on Cross-Site Scripting (XSS)—A Systematic Literature Review. *Information and Software Technology*, 58:170–186, 2015.
[70] M. Injadat, A. Moubayed, and A. Shami. Detecting Botnet Attacks in IoT Environments: An Optimized Machine Learning Approach. *Proc. International Conference on Microelectronics*, 2020.
[71] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker. Improving Vulnerability Remediation through Better Exploit Prediction. *Journal of Cybersecurity*, 6(1):tyaa015, 2020.
[72] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker. Improving Vulnerability Remediation through Better Exploit Prediction. *Journal of Cybersecurity*, 6(1), 2020.
[73] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman. Exploit Prediction Scoring System (EPSS). *Digital Threats: Research and Practice*, 2(3):1–17, 2021.
[74] J. Jang, A. Agrawal, and D. Brumley. ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions. In *Proc. IEEE Symposium on Security and Privacy*, 2012.
[75] B. Jerman-Blažič et al. Towards a Standard Approach for Quantifying an ICT Security Investment. *Computer Standards & Interfaces*, 30(4):216–222, 2008.
[76] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng. Flowguard: An Intelligent Edge Defense Mechanism Against IoT DDoS Attacks. *IEEE Internet of Things Journal*, 7(10):9552–9562, 2020.
[77] Y. Jiang and Y. Atif. An Approach To Discover and Assess Vulnerability Severity Automatically in Cyber-Physical Systems. In *International Conference on Security of Information and Networks*, 2020.
[78] S. Kim, S. Woo, H. Lee, and H. Oh. VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery. In *Proc. IEEE Symposium on Security and Privacy*, 2017.
[79] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[80] T. Krenc and A. Feldmann. BGP Prefix Delegations: A Deep Dive. In *Proc. ACM IMC*, 2016.
[81] P. Kudjo, J. Chen, S. Mensah, R. Amankwah, and C. Kudjo. The Effect of Bellwether Analysis on Software Vulnerability Severity Prediction Models. *Software Quality Journal*, 28:1–34, 12 2020.
[82] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent Convolutional Neural Networks for Text Classification. 2015.
[83] T. H. Le, H. Chen, and M. A. Babar. A Survey on Data-driven Software Vulnerability Assessment and Prioritization. *ACM Computing Surveys*, 55(5):1–39, 2021.
[84] T. H. M. Le and M. A. Babar. On the Use of Fine-Grained Vulnerable Code Statements for Software Vulnerability Assessment Models. In *International Conference on Mining Software Repositories*, 2022.
[85] T. H. M. Le, H. Chen, and M. A. Babar. A Survey on Data-driven Software Vulnerability Assessment and Prioritization. *ACM Comput. Surv.*, 55(5), 2022.
[86] T. H. M. Le, R. Croft, D. Hin, and M. A. Babar. A Large-scale Study of Security Vulnerability Support on Developer Q&A Websites. In *International Conference on Evaluation and Assessment in Software Engineering*, 2021.
[87] T. H. M. Le, B. Sabir, and M. A. Babar. Automated Software Vulnerability Assessment with Concept Drift. In *International Conference on Mining Software Repositories*, 2019.
[88] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521:436–444, 2015.
[89] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In *Proc. NDSS*, 2018.
[90] J. Lim, Y. L. Lau, L. K. Ming Chan, J. M. Tristan Paul Goo, H. Zhang, Z. Zhang, and H. Guo. CVE Records of Known Exploited Vulnerabilities. In *International Conference on Computer and Communication Systems*, 2023.
[91] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang. Software Vulnerability Detection Using Deep Neural Networks: A Survey. *Proceedings of the IEEE*, 108(10):1825–1848, 2020.
[92] G. Lin, J. Zhang, W. Luo, L. Pan, O. De Vel, P. Montague, and Y. Xiang. Software Vulnerability Discovery via Learning Multi-Domain Knowledge Bases. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2469–2485, 2021.
[93] G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel, and P. Montague. Cross-Project Transfer Representation Learning for Vulnerable Function Discovery. *IEEE Transactions on Industrial Informatics*, 14(7):3289–3297, 2018.
[94] Y. Lin, Y. Zhang, S. Chen, F. Song, X. Xie, X. Li, and L. Sun. Inferring Loop Invariants for Multi-Path Loops. In *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2021.
[95] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang. Detecting and Preventing Cyber Insider Threats: A Survey. *IEEE Communications Surveys and Tutorials*, 20(2):1397–1417, 2018.
[96] M. Luckie, R. Beverly, W. Brinkmeyer, and k. claffy. Speedtrap: Internet-Scale IPv6 Alias Resolution. In *Proc. ACM IMC*, 2013.
[97] M. Luckie, B. Huffaker, A. Marder, Z. Bischof, M. Fletcher, and K. Claffy. Learning to Extract Geographic Information from Internet Router Hostnames. In *Proc. ACM CoNEXT*, 2021.
[98] M. Luckie, A. Marder, M. Fletcher, B. Huffaker, and K. Claffy. Learning to Extract and Use ASNs in Hostnames. In *Proc. ACM IMC*, New York, NY, USA, 2020.
[99] R. Malhotra and Vidushi. Severity Prediction of Software Vulnerabilities Using Textual Data . In *International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications*, 2021.
[100] A. Manna, A. Sengupta, and C. Mazumdar. A Quantitative Methodology for Business Process-Based Data Privacy Risk Computation. *Advanced Computing and Systems for Security: Volume Ten*, pages 17–33, 2020.
[101] R. A. Martin and S. Barnum. Common Weakness Enumeration (CWE) Status Update. *ACM SIGAda Ada Letters*, 28(1):88–91, 2008.
[102] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. Chaves, Ítalo Cunha, D. Guedes, and W. M. Jr. The Evolution of Bashlite and Mirai IoT Botnets. In *Proc. IEEE ISCC*, 2018.
[103] P. Mell and T. Grance. Use of the Common Vulnerabilities and Exposures (CVE) Vulnerability Naming Scheme. *NIST Special Publication*, 800:51, 2002.
[104] P. Mell, K. Scarfone, and S. Romanosky. Common Vulnerability Scoring System. *IEEE Security and Privacy*, 4(6):85–89, 2006.
[105] S. M. Naim, A. P. Boedihardjo, and M. S. Hossain. A Scalable Model for Tracking Topical Evolution in Large Document Collections. In *International Conference on Big Data*, 2017.
[106] M. Nasereddin, A. Al-Khamaiseh, M. Qasaimeh, and R. Al-Qassas. A Systematic Review af Detection and Prevention Techniques of SQL Injection Attacks. *Information Security Journal: A Global Perspective*, 32(4):252–265, 2023.

[107] J. Newsome and D. X. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proc. ISOC NDSS*, 2005.

[108] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian. Darknet and Deepnet Mining for Proactive Cybersecurity Threat Intelligence. In *Conference on Intelligence and Security Informatics*, 2016.

[109] R. Owens and W. Wang. Non-Interactive OS Fingerprinting Through Memory De-Duplication Technique in Virtual Machines. In *Proc. International Performance Computing and Communications Conference*, 2011.

[110] R. Panesar, M. Neve, and D. Rogers. The State of Vulnerability Disclosure Policy (VDP) Usage in Global Consumer IoT. Technical report, IoT Security Foundation, 2023.

[111] J. Pewny, F. Schuster, L. Bernhard, T. Holz, and C. Rossow. Leveraging Semantic Signatures for Bug Search in Binary Programs. In *Computer Security Applications Conference*, 2014.

[112] V.-T. Pham, M. Böhme, and A. Roychoudhury. Model-Based Whitebox Fuzzing for Program Binaries. In *Proc. IEEE/ACM International Conference on Automated Software Engineering*, 2016.

[113] G. Portokalidis, A. Slowinska, and H. Bos. Argos: An Emulator for Fingerprinting Zero-Day Attacks for Advertised Honeypots with Automatic Signature Generation. In *Proc. ACM EuroSys*, 2006.

[114] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. In *IEEE/ACM International Symposium on Microarchitecture*, 2006.

[115] D. A. Ramos and D. Engler. Under-Constrained Symbolic Execution: Correctness Checking for Real Code. In *USENIX Security*, 2015.

[116] K. R. M. Rao and D. Pant. Security Risk Assessment of Geospatial Weather Information System (GWIS): An Owasp Based Approach. *International Journal of Computer Science and Information Security*, 8(5):208–218, 2010.

[117] S. Roy, N. Sharmin, J. C. Acosta, C. Kiekintveld, and A. Laszka. Survey and Taxonomy of Adversarial Reconnaissance Techniques. *ACM Computing Surveys*, 55(6):1–38, 2022.

[118] J. Ruohonen. Classifying web exploits with topic modeling. In *International Workshop on Database and Expert Systems Applications*, pages 93–97, 2017.

[119] J. Ruohonen and V. Leppänen. Toward Validation of Textual Information Retrieval Techniques for Software Weaknesses. In *International Workshop on Database and Expert Systems Applications*, 2018.

[120] E. R. Russo, A. Di Sorbo, C. A. Visaggio, and G. Canfora. Summarizing Vulnerabilities' Descriptions to Support Experts During Vulnerability Assessment Activities. *Journal of Systems and Software*, 156:84–99, 2019.

[121] C. Sabottke, O. Suciu, and T. Dumitraş. Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits. 2015.

[122] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan. Lessons from Building Static Analysis Tools at Google. *Communications of the ACM*, 61(4):58–66, 2018.

[123] S. E. Sahin and A. Tosun. A Conceptual Replication on Predicting the Severity of Software Vulnerabilities. In *International Conference on Evaluation and Assessment in Software Engineering*, 2019.

[124] A. Santos-Olmo, L. E. Sánchez, D. G. Rosado, M. A. Serrano, C. Blanco, H. Mouratidis, and E. Fernández-Medina. Towards an Integrated Risk Analysis Security Framework According To a Systematic Analysis of Existing Proposals. *Frontiers of Computer Science*, 18(3):183808, 2024.

[125] B. J. Santoso, R. M. Ijtihadie, and G. N. S. Aryawan. Vulnerability Data Assessment and Management Based on Passive Scanning Method and CVSS. In *2023 14th International Conference on Information & Communication Technology and System*, pages 325–330. IEEE, 2023.

[126] A. O. A. Semasaba, W. Zheng, X. Wu, and S. A. Agyemang. Literature Survey of Deep Learning-based Bulnerability Analysis on Source Code. *IET Software*, 14(6):654–664, 2020.

[127] C. D. Sestili, W. Snavely, and N. M. V. Houdnos. Towards Security Defect Prediction with Ai. *ArXiv*, abs/1808.09897, 2018.

[128] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen. Network Reconnaissance. *Network Security*, 2008(11):12–16, 2008.

[129] A. Shameli-Sendi. An Efficient Security Data-Driven Approach for Implementing Risk Assessment. *Journal of Information Security and Applications*, 54:102593, 2020.

[130] Z. Shamsi, D. B. Cline, and D. Loguinov. Faulds: A Non-Parametric Iterative Classifier for Internet-Wide OS Fingerprinting. *IEEE/ACM Transactions on Networking*, 29(5):2339–2352, 2021.

[131] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov. Hershel: Single-Packet OS Fingerprinting. *IEEE/ACM Transactions on Networking*, 24(4):2196–2209, 2015.

[132] D. She, Y. Chen, A. Shah, B. Ray, and S. Jana. Neutaint: Efficient Dynamic Taint Analysis with Neural Networks. In *Proc. IEEE Symposium on Security and Privacy*, 2020.

[133] P. Shedden, W. Smith, and A. Ahmad. Information Security Risk Assessment: Towards a Business Practice Perspective. Technical report, School of Computer and Information Science, Edith Cowan University, 2010.

[134] Z. Sheng, B. Yu, C. Liang, and Y. Zhang. VPnet: A Vulnerability Prioritization Approach Using Pointer Network and Deep Reinforcement Learning. In *International Conference on Digital Forensics and Cyber Crime*, pages 307–325. Springer, 2022.

[135] R. Sherwood, A. Bender, and N. Spring. DisCarte: a Disjunctive Internet Cartographer. In *Proc. ACM SIGCOMM*, 2008.

[136] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2010.

[137] D. Silva, L. Teixeira, and M. d'Amorim. Shake it! Detecting Flaky Tests Caused by Concurrency with Shaker. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020.

[138] V. Smyth. Software Vulnerability Management: How Intelligence Helps Reduce the Risk. *Network Security*, 2017(3):10–12, 2017.

[139] SonicWall. Cyber Threat Report. Technical report, 2024.

[140] G. Spanos and L. Angelis. A Multi-Target Approach to Estimate Software Vulnerability Characteristics and Severity Scores. *Journal of Systems and Software*, 146:152–166, 2018.

[141] G. Spanos, A. Sioziou, and L. Angelis. WIVSS: A New Methodology for Scoring Information Systems Vulnerabilities. In *Panhellenic Conference on Informatics*, pages 83–90, 2013.

[142] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick. Time to Change the CVSS? *IEEE Security and Privacy*, 19(2):74–78, 2021.

[143] J. M. Spring. An Analysis of How Many Undiscovered Vulnerabilities Remain In Information Systems. *Computers & Security*, 131:103191, 2023.

[144] J. M. Spring, E. Hatleback, A. D. Householder, A. Manion, and D. Shick. Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization. Technical report, Software Engineering Institute, Carnegie Mellon University, 2019.

[145] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical Automated Detection of Stealthy Portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[146] B. Stein, B.-Y. E. Chang, and M. Sridharan. Demanded Abstract Interpretation. In *Proc. ACM PLDI*, 2021.

[147] G. Stergiopoulos, D. Gritzalis, and V. Kouktzoglou. Using Formal Distributions for Threat Likelihood Estimation In Cloud-Enabled IT Risk Assessment. *Computer Networks*, 134:23–45, 2018.

[148] B. Stojanović, K. Hofer-Schmitz, and U. Kleb. Apt Datasets and Attack Modeling for Automated Detection Methods: A Review. *Computers and Security*, 92:101734, 2020.

[149] O. Suciu, C. Nelson, Z. Lyu, T. Bao, and T. Dumitras. Expected Exploitability: Predicting the Development of Functional Vulnerability Exploits. 2022.

[150] S. Sukhbaatar, J. Weston, and R. Fergus. End-to-End Memory Networks. *Advances in Neural Information Processing Systems*, 28, 2015.

[151] J. Sun, Z. Xing, H. Guo, D. Ye, X. Li, X. Xu, and L. Zhu. Generating Informative CVE Description From ExploitDB Posts by Extractive Summarization, 2021.

[152] M. Sutton, A. Greene, and P. Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Pearson Education, 2007.

[153] A. Takanen, J. D. Demott, C. Miller, and A. Kettunen. *Fuzzing for Software Security Testing and Quality Assurance*. Artech House, 2018.

[154] S. Tamjidi and A. Shameli-Sendi. Intelligence in Security Countermeasures Selection. *Journal of Computer Virology and Hacking Techniques*, 19(1):137–148, 2023.

[155] N. Tavabi, P. Goyal, M. Almukaynizi, P. Shakarian, and K. Lerman. DarkEmbed: Exploit Prediction with Neural Language Models. In *AAAI Conference on Artificial Intelligence*, 2018.

[156] R. J. Thomas, J. Gardiner, T. Chothia, E. Samanis, J. Perrett, and A. Rashid. Catch Me If You Can: An In-Depth Study of CVE Discovery Time and Inconsistencies for Managing Risks in Critical Infrastructures. In *Proc. Joint Workshop on CPS and IoT Security and Privacy*, 2020.

[157] Z. Tian, C. Sun, D. Zeng, and G. Tan. podft: On Accelerating Dynamic Taint Analysis with Precise Path Optimization. In *NDSS Workshop on Binary Analysis Research*, 2023.

[158] D. Toloudis, G. Spanos, and L. Angelis. Associating the Severity of Vulnerabilities with their Description . In *Advanced Information Systems Engineering Workshops*, 2016.

[159] S. Tripathi, G. Grieco, and S. Rawat. Exniffer: Learning to Prioritize Crashes by Assessing the Exploitability from Memory Dump. In *Asia-Pacific Software Engineering Conference*, 2017.

[160] Á. J. Varela-Vaca, L. Parody, R. M. Gasca, and M. T. Gómez-López. Automatic Verification and Diagnosis of Security Risk Assessments in Business Process Models. *IEEE Access*, 7:26448–26465, 2019.

[161] V. Vasilyev, A. Kirillova, A. Vulfin, and A. Nikonov. Cybersecurity Risk Assessment Based on Cognitive Attack Vector Modeling With CVSS Score. In *International Conference on Information Technology and Nanotechnology*, 2021.

[162] F. Védrine, M. Jacquemin, N. Kosmatov, and J. Signoles. Runtime Abstract Interpretation for Numerical Accuracy and Robustness. In *Verification, Model Checking, and Abstract Interpretation*, 2021.

[163] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. In *Proc. IEEE Symposium on Security and Privacy*, 2018.

[164] P. Wang, Y. Zhou, B. Sun, and W. Zhang. Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost. In *International Conference on Advanced Computational Intelligence*, pages 72–77, 2019.

[165] S. Wang, T. Liu, and L. Tan. Automatically Learning Semantic Features for Defect Prediction. In *International Conference on Software Engineering*, 2016.

[166] Y. Wang, J. Shen, J. Lin, and R. Lou. Staged Method of Code Similarity Analysis for Firmware Vulnerability Detection. *IEEE Access*, 7:14171–14185, 2019.

[167] T. Wen, Y. Zhang, Y. Dong, and G. Yang. A Novel Automatic Severity Vulnerability Assessment Framework. *J. Commun.*, 10(5):320–329, 2015.

[168] E. Wåreus and M. Hell. Automated CPE Labeling of CVE Summaries with Machine Learning. In *International Conference on Detection of Intrusions, Malware, and Vulnerability Assessment*, 2020.

[169] X. Wu, W. Zheng, X. Xia, and D. Lo. Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction. *IEEE Transactions on Software Engineering*, 48(7):2541–2556, 2022.

[170] T. Xia, G. Qu, S. Hariri, and M. Yousif. An Efficient Network Intrusion Detection Method Based on Information Theory and Genetic Algorithm. In *Proc. IEEE International Performance, Computing, and Communications Conference*, 2005.

[171] H. Xiao, Z. Xing, X. Li, and H. Guo. Embedding and Predicting Software Security Entity Relationships: A Knowledge Graph Based Approach. In *International Conference on Neural Information Processing*, pages 50–63, 2019.

[172] Y. Yamamoto, D. Miyamoto, and M. Nakayama. Text-Mining Approach for Estimating Vulnerability Score. In *International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, 2015.

[173] G. Yan, J. Lu, Z. Shu, and Y. Kucuk. ExploitMeter: Combining Fuzzing with Machine Learning for Automated Evaluation of Software Exploitability. In *IEEE Symposium on Privacy-Aware Computing*, 2017.

[174] G. Yao, J. Bi, and A. V. Vasilakos. Passive IP Traceback: Disclosing the Locations of IP Spoofers From Path Backscatter. *IEEE Transactions on Information Forensics and Security*, 10(3):471–484, 2015.

[175] G. Yao, J. Bi, and Z. Zhou. Passive IP Traceback: Capturing the Origin of Anonymous Traffic Through Network Telescopes. In *Proc. ACM SIGCOMM*, 2010.

[176] J. Yin, M. Tang, J. Cao, and H. Wang. Apply Transfer Learning to Cybersecurity: Predicting Exploitability of Vulnerabilities by Description. *Knowledge-Based Systems*, 210:106529, 2020.

[177] W. You, X. Wang, S. Ma, J. Huang, X. Zhang, X. Wang, and B. Liang. Profuzzer: On-The-Fly Input Type Probing for Better Zero-Day Vulnerability Discovery. In *Proc. IEEE Symposium on Security and Privacy*, pages 769–786. IEEE, 2019.

[178] A. A. Younis and Y. K. Malaiya. Using Software Structure to Predict Vulnerability Exploitation Potential. In *International Conference on Software Security and Reliability-Companion*, 2014.

[179] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta. How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. In *IEEE/ACM International Conference on Mining Software Repositories*, 2017.

[180] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang. Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey. *IEEE Access*, 8:197158–197172, 2020.

[181] H. Zhang, L. Gong, and S. Versteeg. Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects. In *International Conference on Software Engineering*, 2013.

[182] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin. μVulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2224–2236, 2019.
[183] Zscaler ThreatLabz. Enterprise IoT and OT Threat Report. Technical report, 2023.


HACKERS DO BEM
Softex
SENAI
RNP
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO