Análise temporal de risco de sistemas computacionais via modelagem de séries de eventos associados a vulnerabilidades

Matheus Martins[1], **Miguel A. Bicudo**[1], **Daniel Menasché**[1], **Leandro P. de Aguiar**[2]

[1]DCC/UFRJ, RJ – Brasil, [2] Siemens Corporate Research, Princeton, NJ – USA

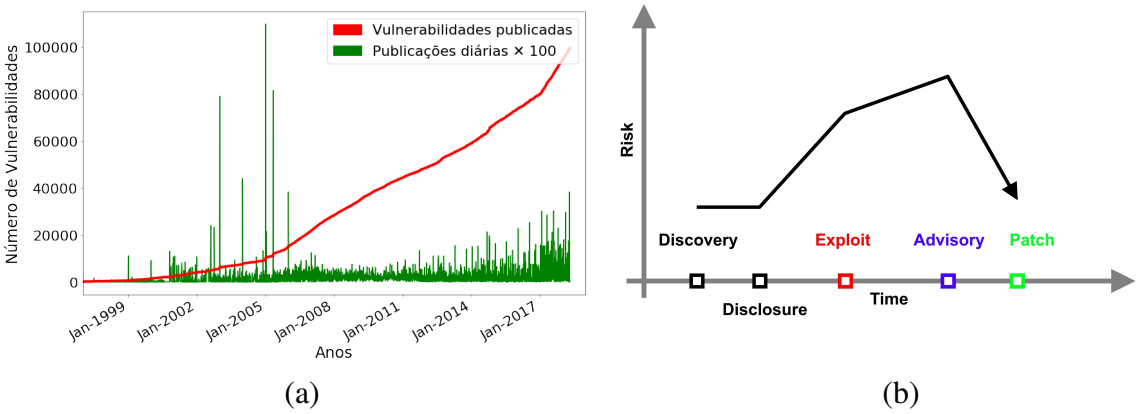
Abstract. *In information security, software and hardware vulnerabilities are increasingly prevalent at the expense of today's technological advances. In this work we present an analysis of time series on vulnerability life cycle searching for trends in information security industry. We also present a machine learning model to predict the occurrence of exploits. The training process was done using approximately 26,000 vulnerability data samples and 132 features, resulting in a model with an initial accuracy of 60% for predicting the first exploit. After adjusting the parameters of the algorithm using grid search, an increase to 67% was achieved using error metrics such as mean absolute error and root mean square error.*

Resumo. *Em segurança da informação, vulnerabilidades de software e hardware são cada vez mais frequentes às custas dos avanços tecnológicos atuais. Neste trabalho apresentamos uma análise de séries temporais sobre o ciclo de vida de vulnerabilidades, buscando as tendências do setor de segurança da informação. Também apresentamos um modelo de aprendizado de máquina para prever a ocorrência de exploits. O processo de treinamento foi feito usando aproximadamente 26.000 amostras de dados de vulnerabilidades e 132 features, resultando num modelo com uma precisão inicial de 60% para prever o primeiro exploit. Depois de ajustar os parâmetros do algoritmo usando grid search, um aumento na acurácia para 67% foi alcançado usando métricas de erro como erro absoluto médio e erro quadrático médio.*

1. Introdução

Ataques cibernéticos são causados por falhas de segurança em sistemas de informação. Essas falhas de segurança são chamadas de vulnerabilidades e são causadas por erros durante o desenvolvimento de módulos de software ou hardware. Quando descobertas, podem ser exploradas por agentes maliciosos comprometendo diversos aspectos de segurança da informação. A Figura 1(a) mostra o crescimento do número de vulnerabilidades, bem como a quantidade de publicações diárias multiplicada por 100. Note que em alguns dias existem centenas de publicações de vulnerabilidades.

Começamos nosso estudo recolhendo dados de vulnerabilidades e seus respectivos históricos de eventos, incluindo: surgimento de *exploits*, publicações de *advisories* por fabricantes e lançamento de *patches*. Um *exploit* é qualquer método capaz de explorar maliciosamente uma vulnerabilidade, seja por código ou apenas uma prova de conceito. Um *advisory* é uma publicação de um boletim de segurança por um fabricante, o que confirma que determinado produto possui uma vulnerabilidade. Um *patch* é uma atualização que pode ser aplicada ao sistema com o intuito de corrigir uma vulnerabilidade, inviabilizando ataques por *exploits* que dependam da vulnerabilidade.



(a) (b)

Figura 1. (a) Acumulado de vulnerabilidades de 1997 até 2018 e (b) ciclo de vida de uma vulnerabilidade.

Objetivos. Usando os eventos históricos de cada vulnerabilidade, criar um modelo capaz de descrever a evolução do risco associado. Para tal, desenvolvemos novas técnicas de parametrização do CVSS [First 2007], que é um modelo de risco padrão da indústria, em função dos eventos recolhidos. CVSS é sigla para *Common Vulnerability Scoring System* [Scarfone and Mell 2009, Mell et al. 2006], e possui diversos parâmetros que por fim se traduzem em uma nota de 0 a 10 medindo o risco da vulnerabilidade. Não é de nosso conhecimento nenhum trabalho anterior que tenha indicado, de forma objetiva, como parametrizar temporalmente o CVSS a partir de dados reais. Adicionalmente, discutimos o uso de aprendizado por máquina para prever o aparecimento de *exploits* e a evolução do risco futuro. *Em suma, visamos compreender, em retrospectiva, como ocorreu a evolução do risco para vulnerabilidades cujos exploits já são de conhecimento geral, para então prever a evolução do risco para novas vulnerabilidades.*

Desafios. Dentre os desafios para alcançar os objetivos propostos, destacamos: *(i) Parametrização do CVSS.* Apesar da existência de um modelo oficial para calcular o risco na medida que eventos acontecem no ciclo de vida de vulnerabilidades, este modelo, chamado de CVSS Temporal, não é parametrizado oficialmente por nenhuma entidade. *(ii) Dados não estruturados.* Coleta de dados de *ground truth* sobre eventos do ciclo de vida de vulnerabilidades. Tais dados costumam estar descentralizados, mal formatados e conflitantes devido a múltiplas fontes de dados. *(iii) Relevância dos atributos.* A identificação das principais *features* nos dados recolhidos que podem influenciar em bons resultados nas técnicas de aprendizado de máquina que abordamos nesse trabalho não é trivial. *(iv) Previsão, visualização.* Os resultados do nosso modelo de aprendizado de máquina envolvem milhares de vulnerabilidades, e a visualização e interpretação dos dados constitui um grande desafio.

Principais Contribuições Entre as principais contribuições deste trabalho, podemos destacar: *(i)* ***Dataset.*** Composto por eventos associados às vulnerabilidades, incluindo datas e fases de seu ciclo de vida. A organização e montagem do dataset se deu em várias etapas, e se baseou na mesclagem entre várias fontes a partir de dados em comum, como o identificador da vulnerabilidade. *(ii)* ***Parametrização do CVSS.*** Heurística de parametrização do *CVSS Temporal* a partir de uma sequência de eventos de surgi-

mento de *exploits*, *patches* e *advisories*, para gerar uma série temporal de riscos. *(iii)* ***Análise de séries temporais.*** Com o objetivo de mostrar tendências na área de segurança da informação. Dentre as análises possíveis, buscamos descobrir as correlações entre características de vulnerabilidades e as datas dos eventos do seu ciclo de vida. *(iv)* ***Modelo de previsão de exploits e CVSS.*** Modelo baseado em aprendizado de máquina para previsão de aparecimento de *exploits* que é capaz sugerir o tempo até a ocorrência de um *exploit* e de medir o risco associado. Vislumbramos que esse modelo possa ser usado para dar suporte à decisão em gerência de *patches*.

Estrutura do artigo. Na seção 1 apresentamos a motivação, problemática, objetivos e contribuições. Na seção 2 discutiremos a metodologia aplicada. Na seção 3 apresentamos os as análises e resultados. Na seção 4 mostraremos os principais trabalhos relacionados. Na seção 5 concluímos o trabalho com observações e comentários e na seção 6 discutiremos as limitações que nosso trabalho possui.

2. Metodologia

Esse trabalho faz uso de resultados empíricos para uma análise ampla de dados históricos de eventos de segurança da informação. Foram recolhidos dados temporais históricos de eventos que marcam as fases do ciclo de vida de vulnerabilidades: *exploits*, *patches* e *advisories*. Com a coleção de dados sobre eventos, foram feitas: (i) a parametrização do *CVSS temporal* (Seção 2.1), (ii) as análises de séries temporais de eventos no ciclo de vida de vulnerabilidades (Seção 3.1), e (iii) a criação de um modelo de previsão de aparecimento de *exploits* (Seções 2.3 e 3.2).

Buscou-se as associações entre vulnerabilidades e as diversas fases que compõem seu ciclo de vida. Estas associações foram realizadas através do identificador de vulnerabilidades *Common Vulnerabilities and Exposures Identifier (CVE-ID)* [MITRE 2018]. Por exemplo, CVE-2012-0001 corresponde à primeira vulnerabilidade divulgada em 2012. Foram recolhidos aproximadamente 60 mil eventos para 21 mil vulnerabilidades. Os dados de CVEs foram recolhidos no *National Vulnerability Database* [NIST 2018] que é um repositório do governo dos EUA de dados de gerenciamento de vulnerabilidades com mais de 90 mil vulnerabilidades registradas. Já os dados de *exploits* foram recolhidos no *Exploit Database* [Security 2018]. O *Exploit Database* é um banco de dados público sobre *exploits*, atualizado regularmente e contando em janeiro de 2019 com mais de 39 mil *exploits*.

Toda CVE possui uma pontuação de severidade chamado *Common Vulnerability Scoring System (CVSS)* que fornece uma maneira de capturar as principais características de uma vulnerabilidade e produzir uma pontuação numérica que reflita sua gravidade. O valor do CVSS é quantificado de 0 até 10, sendo que essa pontuação possui uma representação qualitativa de risco: baixa, média, alta e crítica. A Figura 1(b) mostra as principais fases do ciclo de vida de uma vulnerabilidade em um exemplo genérico de ocorrência desses eventos numa série temporal, indicando a respectiva evolução de risco.

A descoberta de um vulnerabilidade é categorizada como o primeiro evento possível numa linha temporal do ciclo de vida de uma vulnerabilidade. Por exemplo, não é possível existir um *exploit* ou *patch* sem que antes uma vulnerabilidade seja descoberta. Também é importante notar que a data de descoberta é diferente da data de divulgação. A descoberta é marcada pela primeira data reportada de uma vulnerabilidade reconhecida

como sendo um risco de segurança enquanto a divulgação é a primeira data em que uma vulnerabilidade é revelada ao público. Note que a ocorrência desses eventos não influencia o risco. Abaixo é descrito em detalhes os principais eventos na linha temporal que são responsáveis pela variação do risco:

Eventos de *exploit*: representam o aparecimento de métodos ou programas que são capazes de explorar a vulnerabilidade em questão. *Exploits* podem ter diversos tipos de maturidade: uma simples prova de conceito ou até mesmo um *exploit* funcional e automatizado. Quando um evento de *exploit* acontece, é esperado que o risco aumente. Os dados de *exploits* foram recolhidos no *Exploit Database* [Security 2018].

Eventos de *advisory*: marcam o aparecimento de relatórios e boletins de segurança por fabricantes que possuem relação com o produto afetado pela vulnerabilidade. Quando um evento de *advisory* acontece, o risco aumenta. Estes dados foram recolhidos no *National Vulnerability Database*.

Eventos de *patch*: marcam o surgimento de correções para vulnerabilidades, e geralmente representam o fim de um ciclo de vida de uma vulnerabilidade. Quando um evento de *patch* acontece, o risco diminui. Estes dados também foram recolhidos no *National Vulnerability Database*.

2.1. Parametrização do CVSS Temporal

O CVSS possui um grupo de métricas temporais, chamado de CVSS Temporal, para cálculo do risco no decorrer do tempo na medida em que eventos de segurança acontecem. Ele tem como objetivo refletir os eventos ocorridos no nível de risco da vulnerabilidade. Três desses eventos que o CVSS Temporal captura são: confirmação dos detalhes técnicos de uma vulnerabilidade, o status de remediação da vulnerabilidade e a disponibilidade de código ou técnicas de exploração. Apesar da existência desse modelo, o CVSS Temporal é considerado opcional e não é parametrizado pelas organizações responsáveis pelo CVSS. Um dos grandes objetivos desse trabalho é a parametrização desse modelo usando heurísticas para o mapeamento de ocorrências desses eventos a partir de dados históricos. Abaixo são descritas as submétricas do CVSS Temporal, seus possíveis estados e as respectivas heurísticas para mapeamentos de eventos propostas nesse trabalho.

1. Exploitability: Essa métrica mede o estado atual de técnicas de exploração de uma determinada vulnerabilidade, categorizando a maturidade das mesmas. Por exemplo, a existência de provas de conceito ou códigos de *exploits* maliciosos funcionais é capturada por meio desta métrica. A disponibilidade pública de um programa malicioso aumenta o número de possíveis ataques e, consequentemente, aumenta a gravidade da vulnerabilidade. Estados possíveis: (i) *Unproven*: Nenhum *exploit* está disponível, ou a exploração da vulnerabilidade é inteiramente teórica; (ii) *Proof-of-Concept*: Um código de *exploit* em forma de prova de conceito ou uma demonstração não prática de ataque para a maioria dos sistemas está disponível. O código ou técnica não é funcional em todas as situações e pode exigir modificações substanciais por um invasor habilidoso; (iii) *Functional*: O código de um *exploit* funcional está disponível. O código funciona na maioria das situações em que a vulnerabilidade existe; (iv) *High*: A vulnerabilidade é explorável por um código de *exploit* autônomo que seja móvel e funcional, ou não é necessário nenhum *exploit* (acionamento manual) e os detalhes estão amplamente disponíveis para o público. O código funciona em qualquer situação ou está sendo usado ativamente por meio de um

agente autônomo móvel (como um *worm* ou vírus).

Heurística de parametrização. Para mapear os eventos acima, aplicamos a seguinte heurística: categorizamos a maturidade do *exploit* de acordo com a quantidade de *exploits* encontrados para aquela vulnerabilidade. Ou seja, o primeiro *exploit* reportado é mapeado como *Proof-of-Concept*, o segundo *Functional* e, a partir do terceiro, *High*. Trata-se de uma heurística simples para o mapeamento dos eventos de aparecimento de *exploits*. A heurística baseia-se na ideia de que os *hackers* têm motivação para criar novos *exploits* caso haja espaço para melhorias nos *exploits* já existentes. Cabe destacar que a criação de um novo *exploit* é uma tarefa de alta complexidade. O surgimento de um *exploit* funcional para o público geral (e.g., que possa ser usado por *script kiddies*) indica que os códigos maliciosos pre-existentes eram, possivelmente, apenas provas de conceito (*proof of concept*).

2. Remediation Level: O nível de remediação de uma vulnerabilidade é um fator importante para priorização. Uma vulnerabilidade geralmente não possui um *patch* no momento de sua publicação. Soluções alternativas ou *hotfixes* podem oferecer remediação provisória até que um *patch* oficial ou atualização seja emitida. Cada um desses estágios ajusta a pontuação temporal para baixo, refletindo a urgência decrescente à medida que a remediação se torna final. Estados possíveis: (i) *Official Fix*: Uma solução completa do fabricante que corrige a vulnerabilidade está disponível. O fabricante emitiu um *patch* oficial ou uma atualização está disponível; (ii) *Temporary Fix*: Há uma remediação oficial disponível, porém temporária. Isso inclui instâncias em que o fornecedor emite um *hotfix*, uma ferramenta ou uma solução temporária; (iii) *Workaround*: Existe uma solução não-oficial e não-comercial disponível. Em alguns casos, os usuários da tecnologia afetada criarão um *patch* próprio ou fornecerão etapas para contornar ou atenuar a vulnerabilidade; (iv) *Unavailable*: Não há solução disponível ou é impossível aplicá-la.

Heurística de parametrização. Para mapear os eventos acima, aplicamos a seguinte heurística: Quando existe um link na tabela de histórico de vulnerabilidades do NVD que está classificado como *patch*, recolhemos sua data e seu evento é mapeado como *Official Fix*. Se outro link de *patch* é encontrado, o primeiro é mapeado como *Temporary Fix* e o segundo como *Official Fix*. A motivação aqui é similar a aquela descrita na seção acima. Além disso, quando são encontrados links classificados como *third party advisory*, o estado correspondente é mapeado como *Workaround*.

3. Report Confidence: Essa métrica mede o grau de confiança na existência da vulnerabilidade e a credibilidade dos detalhes técnicos conhecidos. Às vezes, apenas a existência de vulnerabilidades é divulgada, mas sem detalhes específicos. A vulnerabilidade pode mais tarde ser corroborada e depois confirmada através do reconhecimento pelo autor ou fabricante da tecnologia afetada. Quanto mais uma vulnerabilidade for validada pelo fornecedor ou por outras fontes confiáveis, maior será a pontuação. Estados possíveis: (i) *Unconfirmed*: Existe uma única fonte não confirmada ou possivelmente vários relatórios conflitantes. Há pouca confiança na validade dos relatórios. Um exemplo é um boato que surge entre fóruns de hackers e mercados negros de *exploits*; (ii) *Uncorroborated*: Existem várias fontes não oficiais, possivelmente incluindo empresas de segurança independentes ou organizações de pesquisa. Neste ponto, pode haver detalhes técnicos conflitantes ou alguma outra ambiguidade remanescente; (iii) *Confirmed*: A vulnerabilidade foi reconhecida pelo fornecedor ou autor da tecnologia afetada. A vul-

nerabilidade também pode ser confirmada quando a sua existência é confirmada a partir de um evento externo, como publicação de código de exploração funcional ou de prova de conceito ou exploração generalizada.

Heurística de parametrização. Para mapear os eventos acima, aplicamos a seguinte heurística: Quando existe um link na tabela de histórico de vulnerabilidades do NVD que está classificado como *patch*, recolhemos sua data e seu evento é mapeado como *Confirmed*. Além disso, quando é encontrado links classificados como *third party advisory*, seu estado é mapeado para *Uncorroborated*. Note que esse mapeamento é bem parecido com o mapeamento para *remediation level* visto que a divulgação de um *patch* oficial também confirma a existência da vulnerabilidade pelo fabricante.

2.2. Estudos de caso: *Heartbleed* e *Wannacry*

A Figura 2 mostram dois exemplos de *ground truth* com todos os eventos mapeados, suas respectivas datas relativas à data de divulgação e o seu risco associado dado pelo CVSS Temporal. A Figura 2(a) mostra a evolução o risco da vulnerabilidade conhecida como *Heartbleed* enquanto que a Figura 2(b) mostra a evolução o risco da vulnerabilidade conhecida como *Wannacry*. Ambas as vulnerabilidades foram amplamente repercutidas na mídia. Observe que, apesar de terem existido ataques reais para essas vulnerabilidades, o modelo do CVSS Temporal não prevê a ocorrência desse evento importante no ciclo de vida de vulnerabilidades.

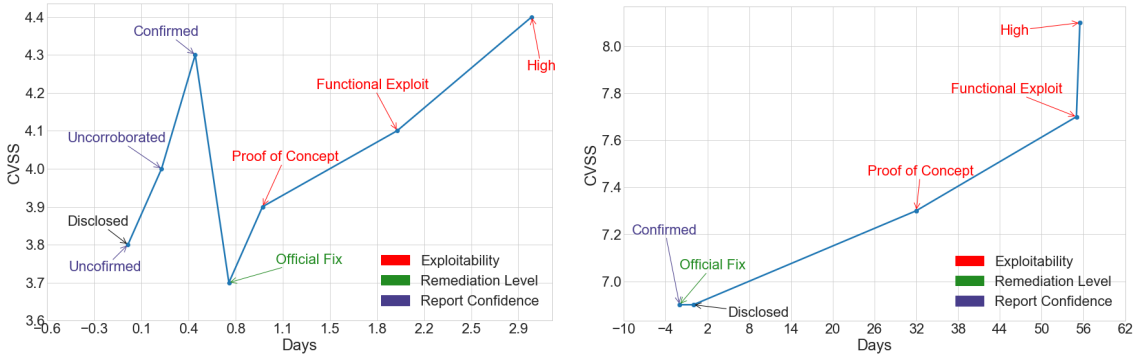


Figura 2. Eventos de segurança da informação mapeados no CVSS Temporal para a vulnerabilidade *Heartbleed* e *Wannacry*

2.3. Previsão de aparecimento de *exploits*

Nossa abordagem busca treinar modelos baseados em aprendizado de máquina usando as características das CVEs para que estes modelos sejam capazes de predizer, com um certo nível de confiança, um intervalo de tempo até que um *exploit* apareça, dado que uma vulnerabilidade foi divulgada. O algoritmo utilizado foi o *Extreme Gradient Boosting* [Chen et al. 2015]. O *XGBoost* é uma biblioteca de otimização de *Gradient Boosting* otimizada e projetada para ser altamente eficiente, flexível e portátil. Ela implementa algoritmos de aprendizado de máquina sob a estrutura do *Gradient Boosting*.

Dentre os modelos avaliados, os melhores foram o de redes neurais pelo Keras [Keras 2019] e o *gradient boosting* pelo XGBoost. Entre esses modelos, o XGBoost

possui características atraentes como a facilidade de explicar o resultado, bastando olhar os nós das árvores de decisão geradas.

O ajuste fino dos hiperparâmetros foi feito seguindo recomendações de especialistas em XGBoost [Spark 2017], que são: (i) usar *grid-search* para algumas combinações de hiperparâmetros, que em suma consta de força bruta, escolhe-se uma resolução e se divide o espaço de combinações dos hiperparâmetros na resolução escolhida, (ii) usar a validação cruzada do XGBoost, que no caso foi feita com sete subconjuntos, sendo seis para treinamento e um para validação, o que permite evitar *overfitting*.

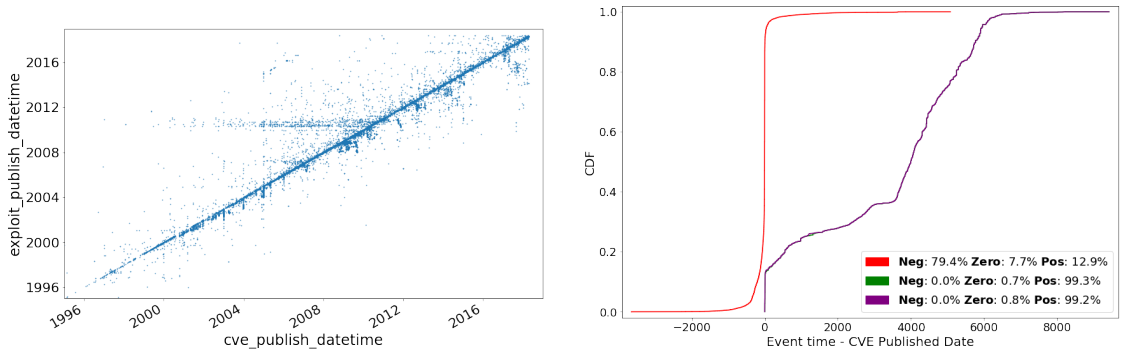
A previsão do risco foi implementada como a solução de um problema de classificação, onde cada associação entre um *exploit* e uma vulnerabilidade (CVE) corresponde a uma amostra de nossa métrica alvo, denotada por *weaponization time*. Esta métrica representa o dia do aparecimento de um *exploit*, assumindo o dia da divulgação da vulnerabilidade como sendo o dia zero. Por exemplo, se uma CVE foi divulgada 24 de Setembro de 2014 e um *exploit* para essa vulnerabilidade apareceu no dia 23 de Setembro de 2014, então seu *weaponization time* é -1.

Para entender nossa abordagem de pesquisa na criação deste modelo, é preciso conhecer as variáveis que compõe nosso dados, ou seja, as *features* que foram usadas no processo treinamento do algoritmo. Dentre as características das CVEs que usamos para o treino destacamos: (i) Data de divulgação da vulnerabilidade; (ii) Data de geração do CVSS; (iii) Data da última modificação dos históricos de eventos; (iv) Common Vulnerability Scoring System (CVSS) que é a métrica de severidade da vulnerabilidade; (v) Os componentes do CVSS que representam características sobre impacto e exploração; (vi) Produto afetado pela vulnerabilidade; (vii) Fabricantes do produto afetado.

O objetivo principal para nosso modelo de aprendizado de máquina é determinar, dada uma vulnerabilidade que acabou de ser divulgada, qual será seu *weaponization time*, ou seja, o aparecimento do primeiro *exploit*. A classificação se baseia em 4 classes pré definidas de *weaponization time*. As classes foram divididas da seguinte forma: (i) Classe 1: amostras com valor de *weaponization time* menor ou igual a -31; (ii) Classe 2: amostras com valor de *weaponization time* entre -30 e -4; (iii) Classe 3: amostras com valor de *weaponization time* entre -3 e 0; (iv) Classe 4: amostras com valor de *weaponization time* maior que 0. Buscamos gerar classes balanceadas, com relação ao número de instâncias em cada uma, mas respeitando a semântica do problema (e.g., o dia zero, ou *zero day*, precisa estar na fronteira de uma classe).

Foi treinado o algoritmo com as *features* descritas e, após isso, com objetivo de aumentar a acurácia do modelo, foi feito o *tunning* de alguns parâmetros do XGBoost. Existem parâmetros que adicionam restrições à arquitetura das árvores e podem ser usados para controlar sua complexidade, o que pode tornar o modelo mais ou menos suscetível a *overfitting*. Também existem parâmetros que controlam a amostragem do conjunto de dados em cada iteração do algoritmo, a fim de construir uma árvore com dados ligeiramente diferentes em cada etapa, o que pode também tornar o modelo menos propenso a *overfitting*. A seguir, descrevemos os parâmetros do XGBoost que foram otimizados pelo nosso *tunning*:

Max Depth: Profundidade máxima de uma árvore. Se a profundidade máxima for 5, será permitido interações de até 5 variáveis juntas. Cada nível duplica a complexidade



(a) (b)

Figura 3. Gráfico de dispersão entre publicação de *exploits* e vulnerabilidades

e o tempo de execução. Aumentar esse valor tornará o modelo mais complexo e mais propenso a *overfitting*.

Min Child Weight: O peso mínimo (ou o número de amostras, se todas as amostras tiverem um peso de 1) necessários para criar um novo nó de uma árvore. Um *Min child weight* menor permite que o algoritmo crie filhos que correspondam a menos amostras, permitindo árvores mais complexas, porém tornando o modelo mais propenso a *overfitting*.

Row Sample: Taxa de amostragem de linhas na instância de treinamento. Um valor de 0.5, por exemplo, significa que o XGBoost amostra aleatoriamente metade dos dados de treinamento antes do crescimento das árvores, visando evitar *overfitting*. Um *row sampling* ocorrerá uma vez em cada iteração do algoritmo. Ele pode ajudar a lidar com *outliers* no conjunto de dados pois, a cada rodada, estes *outliers* têm maior chance de não acabar no conjunto de treinamento.

Column Sample: Taxa de amostragem de colunas ao construir cada árvore. O *column sampling* ocorre uma vez para cada árvore construída e este pode ajudar a evitar *overfitting*.

Learning Rate: Taxa de aprendizado, indicando o quão agressivo será o algoritmo ao buscar corrigir erros das árvores anteriores ao construir novas árvores.

Na Seção 3.2 apresentamos os resultados do ajuste dos parâmetros acima, no cenário de interesse.

3. Análise dos resultados

Nesta seção aplicaremos experimentos de análise de dados para descobrir tendências e importantes características que possam ter alguma significância no aparecimento de *exploits*. A Figura 3(a) mostra o gráfico de dispersão das datas de divulgação da vulnerabilidade e sua respectiva data de publicação de *exploit*. Boa parte da população de pontos se distribui próximo da diagonal, que representa o dia 0 relativo a data de publicação da vulnerabilidade. Mas é interessante notar que entre os anos de 2009 até 2012 foram publicados diversos *exploits* para vulnerabilidades antigas. Isso pode ter acontecido por conta de algum grupo de hackers que resolveu publicar *exploits* para vulnerabilidades antigas de produtos que possivelmente ainda estavam vulneráveis e sem uma correção.

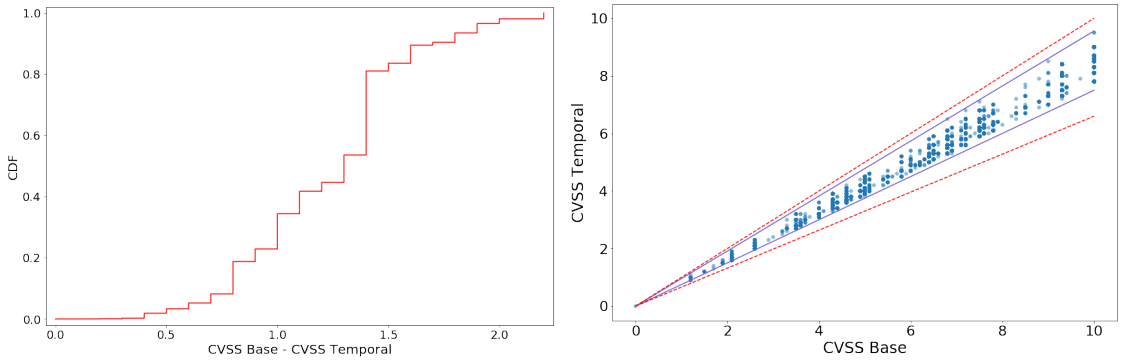
3.1. Análise das Séries Temporais

Nesta seção, nossos objetivos são avaliar como (i) o risco varia para as vulnerabilidades conhecidas, fazendo uso das heurísticas propostas e (ii) os tempos de surgimento de exploits e patches se relacionam com o risco. A Figura 3(b) mostra a CDF dos tempos de aparecimento de eventos de *exploit*, *patch* e *advisory*. O tempo é calculado pela diferença em dias da data de um evento e a data de publicação de sua vulnerabilidade. Para a curva vermelha é possível observar que mais de 80% dos *exploits* é lançado antes do dia zero, ou seja, antes da publicação da CVE. A curva roxa e a curva verde estão sobrepostas porque a descrição de ocorrência dos eventos é muito parecida. Logo, sua respectivas heurísticas são semelhantes. Note que em torno de 18% das publicações de *patch* e *advisory* acontecem próximas à data de publicação da vulnerabilidade. Note também que não existem valores negativos para eventos de *patch* e *advisory*. Isso acontece porque o NVD inclui links de eventos somente depois que uma vulnerabilidade é publicada.

A Figura 4 (a) mostra a CDF da diferença entre seu CVSS Base e o CVSS temporal final para toda a população das séries geradas. É importante notar que o CVSS Base é o valor fixo do risco associado a uma vulnerabilidade onde este valor corresponde a características inerentes da vulnerabilidade, enquanto que o CVSS temporal varia com o tempo de acordo com o aparecimento de eventos no ciclo de vida de vulnerabilidades. O CVSS temporal final representa o valor do risco depois da ocorrência de todos esses eventos, ou seja, o valor do risco associado ao seu último evento ocorrido em ordem cronológica. O gráfico nos mostra que existe boa parte da população que não chega ao risco máximo. É possível verificar isso observando o excedente da diferença entre o CVSS base e o CVSS temporal. Esse comportamento acontece porque para muitas vulnerabilidades existem *patches* que fazem seu risco diminuir.

A Figura 4 (b) é um gráfico de dispersão em que o eixo x apresenta o valor do CVSS de base e o eixo y apresenta o valor do CVSS temporal final resultante de uma vulnerabilidade. Podemos observar por este gráfico que as vulnerabilidades variam entre 74% e 100% do valor de base do CVSS, o que se deve à forma como a heurística trata os eventos nas séries temporais. O que acontece é que quando um *Official-Fix* surge, ou seja, um *patch* oficial, dois parâmetros são alterados ao mesmo tempo, pois foi considerado que este evento também é mapeado com uma confirmação oficial sobre a vulnerabilidade, ou seja, o parâmetro *Report Confidence* vai para *Confirmed*. O valor mínimo real permitido pela métrica CVSS é de 66% do valor de base, e está marcado no gráfico como uma linha vermelha. A linha azul por sua vez é a marca de 74% do CVSS base com as heurísticas aplicadas deste trabalho.

As Figuras 5(a) e 5(b) são gráficos de dispersão em que o eixo x apresenta o tempo relativo do evento de *exploit* e o eixo y é o tempo relativo do evento de *patch*. As cores dos pontos são o valor do CVSS base. Na primeira, considera-se toda a população de vulnerabilidades. No segunda, considera-se a população que sejam do fabricante Microsoft. Podemos notar que na figura 5 as vulnerabilidades de maior risco obtém um *patch* de forma geral mais próximas às datas de divulgação, ao passo que, os de menos risco podem receber *patches* muito depois da sua data de divulgação.



(a) (b)

Figura 4. CDF da diferença CVSS temporal final e o CVSS Base

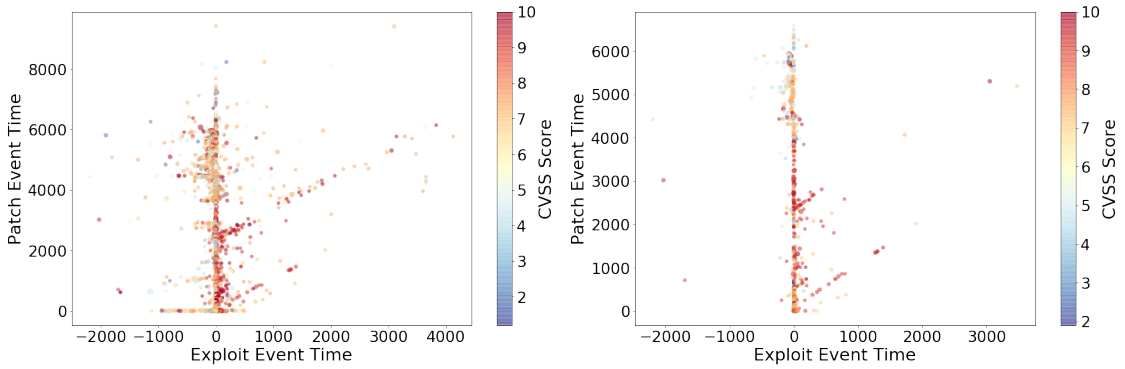


Figura 5. Gráfico de dispersão das datas de evento de *patch* e datas de evento de *exploits* para toda população e para microsoft

3.2. Análise do Modelo de Previsão de *Exploits*

Nesta seção, nossos objetivos são (i) avaliar o desempenho do modelo de previsão do tempo de surgimento de exploits e (ii) avaliar a sensibilidade do desempenho com relação a diferentes hiperparâmetros do modelo.

A Figura 6 mostra algumas métricas obtidas para o ajuste dos hiperparâmetros *max depth* e *min child weight*. O ajuste foi feito com uma busca usando as combinações de vários valores possíveis desses hiperparâmetros. Este tipo de análise pode nos mostrar onde o modelo pode estar sofrendo *overfitting*. Esta Figura foi dividida em três linhas e duas colunas. Na primeira coluna, subfiguras (a) e (b), estão as métricas de validação e treino usando o *mean absolute error (MAE)*. De acordo com (a) essa métrica apresenta um resultado insatisfatório, pois permite interpretar que o valor de *max depth* pode ser aumentado indefinidamente que o modelo sempre vai melhorar. Entretanto essa interpretação não pode ser verdadeira pois este hiperparâmetro aumenta a complexidade das árvores criadas, consequentemente, aumentando as chances de ocorrência de *overfitting*. Por isso, avaliamos outra métrica de erro: *root mean square error* (RMSE), cujos gráficos estão na segunda coluna, subfiguras (c) e (d). Observe que (c) mostra que o modelo estava em processo de *overfitting* e que existe um ponto ótimo, que está nos pontos azuis mais escuros. Na terceira coluna, temos a (e) medida de tempo do treino e o (f) número de árvores de decisão usadas pelo algoritmo para cada iteração do *grid search*. É interessante notar

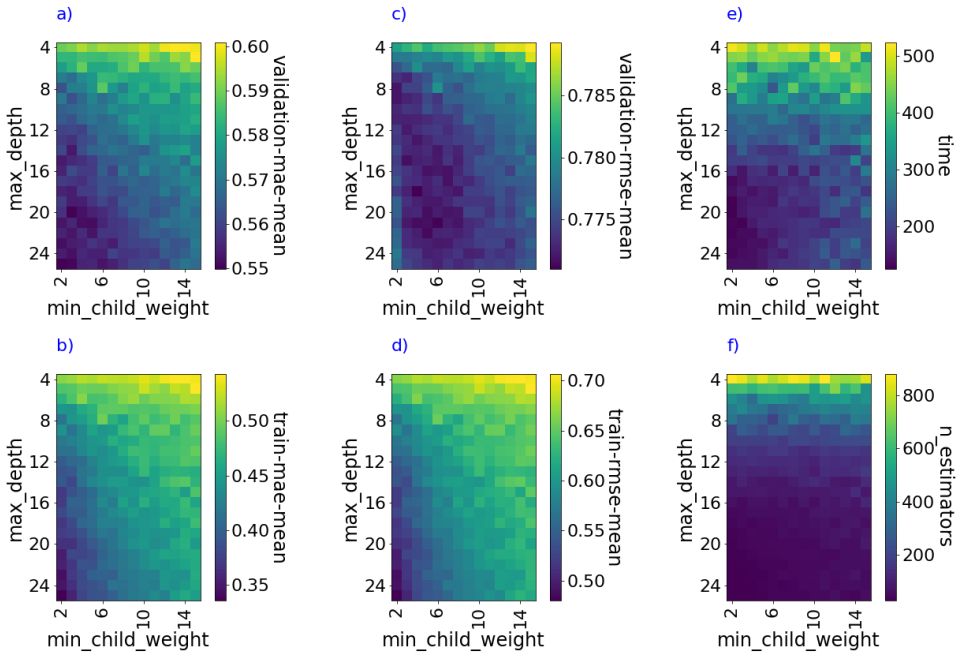


Figura 6. Gráfico dos hiperparâmetros de aprendizado em forma de mapa de calor

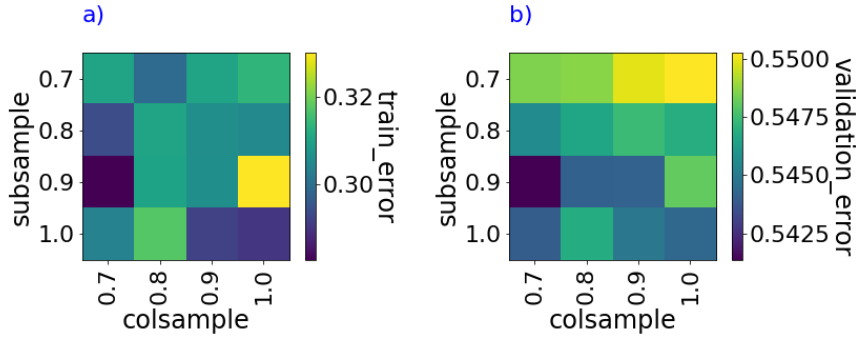


Figura 7. Mapa de calor para os hiperparâmetros de row sample e column sample

que quanto menos profundas são as árvores (*max depth*), mais o treinamento demora e, ao mesmo tempo, um maior número de árvores são necessárias para o treinamento chegar a um ponto ótimo. No gráfico do tempo também pode-se observar que quando a árvore se torna mais profunda, o hiperparâmetro *min child weight* começa a ter um peso maior no tempo de treinamento.

A Figura 7 mostra algumas métricas obtidas para o ajuste dos hiperparâmetros *row samples* e *column samples*. Para estes hiperparâmetros foram feitos apenas os gráficos de (a) treino e (b) validação usando *mean absolute error* (MAE). Ambos os hiperparâmetros podem variar de 0.0 até 1.0, pois representam frações conjunto de amostras (linhas) e *features* (colunas) dos dados de entrada em cada nó da árvore de decisão. Essa aleatorização serve para evitar *overfiting*.

A Figura 8 mostra a matriz confusão do modelo gerado para os dados de validação. Essa matriz considera os valores verdadeiros no eixo vertical e os valores obtidos pelo classificador no eixo horizontal. A diagonal representa os acertos em que a classe real e a

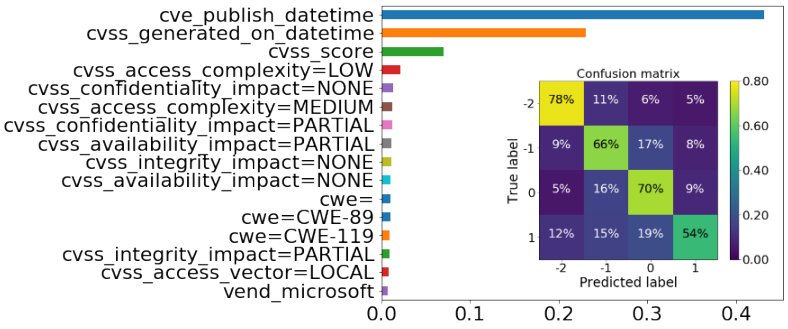


Figura 8. Gráfico de importância das *features* usadas no treinamento

classe prevista pelo algoritmo são iguais. Fora da diagonal tem-se os erros do modelo, em que a classe real difere da classe indicada pelo modelo. Como é possível ver, a previsão de classe das vulnerabilidades é bem mais precisa para classes 1, 2 e 3 que representam os casos em que o *exploit* ocorreu antes da data de divulgação da mesma. Isso quer dizer que o algoritmo consegue prever bem, com aproximadamente 70% de chance de acerto, se uma vulnerabilidade já possui pelo menos um *exploit* na data de sua divulgação. Por outro lado, para aquelas classificadas na classe 4, que representam amostras de *exploits* que vêm depois da divulgação da vulnerabilidade, a chance de acerto é de 55%. Ou seja, com 55% de chance o *exploit* virá de fato após a data de divulgação, mas com 45% de chance o *exploit* já existe.

Também nessa figura, encontram-se as importâncias das *features* mais relevantes de acordo com o modelo do XGBoost. Isso representa quais delas possuem maior influência sobre a previsão das classes. É possível observar que as *features* que mais importam são as datas de publicação da vulnerabilidade, e de geração do CVSS, que é quando os parâmetros do CVSS foram preenchidos. O escore CVSS é o terceiro atributo que possui alguma relevância. Depois disso, praticamente todos os atributos são pouco significantes para a previsão, includindo fabricantes, produtos e outros.

4. Trabalhos Relacionados

Estudos sobre o ciclo de vida de vulnerabilidades Os autores de [Frei et al. 2006] propõem uma base teórica sobre as fases do ciclo de vida de vulnerabilidades de segurança da informação detalhando uma análise empírica e sistemática de vulnerabilidades de segurança em grande escala. Eles também fornecem ferramentas para medir dinâmicas de segurança e quantificam disponibilidade de *exploits* e de *patch* para modelar a exposição ao risco e para apoiar decisões de negócios. Outros estudos [Shahzad et al. 2012] também propõem trabalhos similares focando em fabricantes de produtos vulneráveis e sua capacidade de resposta na aplicação de correções de vulnerabilidades. Em [Bilge and Dumitras 2012], os autores fazem uso de propriedades do ciclo de vida de vulnerabilidades para identificar automaticamente ataques partir de dados coletados em campo que registram quando códigos binários benignos e malignos são baixados em mais de 11 milhões de hosts reais.

Estudos sobre análises de métricas de risco Trabalhos que mostram estudos sobre métricas de risco de vulnerabilidades também são essenciais para este trabalho. Neste artigo [Scarfone and Mell 2009] os autores propõem uma especificação para medir

a gravidade relativa das vulnerabilidades de software chamada *Common Vulnerability Scoring System* (CVSS). Este artigo analisa a eficácia da versão 2 do CVSS se baseando em experimentos usando a pontuação da versão 1 e da versão 2 do CVSS a um grande conjunto de vulnerabilidades recentes. Há também artigos que criticam a metodologia do CVSS [Petraityte et al. 2018] afirmando que a fórmula do CVSS deve ser ajustada para refletir com mais precisão os riscos associados às vulnerabilidades de aplicativos móveis. Os autores exploram a aplicabilidade deste modelo aplicando-o em diferentes vulnerabilidades endereçando o desafio de gerenciamento de risco de vulnerabilidades de software em dispositivos móveis.

Previsão de aparecimento de *exploits* Com relação a previsão de ocorrência de *exploits*, pode se dizer que os trabalhos atualmente estão em estágios iniciais. Trabalhos como em [Bozorgi et al. 2010] descrevem abordagens complementares para avaliação de vulnerabilidades usando ferramentas de mineração de dados e aprendizado de máquina, onde o modelo descrito por eles pode classificar vulnerabilidades significativamente melhor que o atual sistema de escore de severidade [Mell et al. 2006]. Também em [Wang et al. 2008] os autores propuseram uma métrica probabilística de ataques baseada em grafos e mostraram que essa métrica tem uma interpretação intuitiva e significativa, que é útil na tomada de decisões no mundo real. Também existem trabalhos que propõem modelos para predição de aparecimento de *exploits* usando dados de redes sociais [Sabottke et al. 2015].

5. Limitações e discussão

Nesta seção apresentaremos e discutiremos três limitações relevantes ao nosso trabalho: (i) **Validação dos dados.** É importante reconhecer que nem sempre os dados temporais coletados sobre vulnerabilidades e seus diversos eventos são representativos da realidade. A comunidade na área de segurança ofensiva e defensiva costuma não publicar seus seus dados por conta de diversos fatores: processos internos dentro de uma empresa, sigilo, espionagem e até mesmo uso mal intencionado sem divulgação alguma. Trata-se de um problema inerente aos nossos dados e que estamos investigando novas fontes de dados como mercados negros de *exploits* e fóruns sobre *hacking*. (ii) **CVSS.** A representação de risco de uma vulnerabilidade também é bem genérica e, muitas vezes, criticada por pesquisadores da comunidade por não capturar diversas outras características específicas de vulnerabilidades e eventos importantes que não mapeados no CVSS temporal como um real acontecimento de um ataque cibernético. Por conta disso, é possível que a métrica de risco do CVSS não represente a realidade de risco daquela vulnerabilidade. (iii) **Classes do classificador.** As quatro classes consideradas em nosso problema de classificação levaram em conta um *tradeoff* entre gerar classes relativamente balanceadas com relação ao número de amostras e a semântica das mesmas. Trabalho futuro consiste em considerar mais classes, comparando os resultados contra aqueles obtidos via regressão.

6. Conclusão

Este trabalho, que é vastamente baseado em estudos empíricos, mostrou diversas análises sobre o ciclo de vida de vulnerabilidades. A construção das séries temporais ajuda a entender como os diversos eventos do ciclo de vida de uma vulnerabilidade são distribuídos e como seu risco se comporta com o tempo. Foi mostrado que a parametrização do CVSS temporal é possível, apesar de ainda existirem limitações. Também foi apresentado neste

trabalho um modelo de aprendizado de máquina para previsão de aparecimento de *exploits* usando o algoritmo chamado *Extreme Gradient Boosting*. O modelo criado possui uma acurácia de 67% e vem sendo usado em um protótipo para gerência de risco de vulnerabilidades.

Referências

Bilge, L. and Dumitras, T. (2012). Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM.

Bozorgi, M., Saul, L. K., Savage, S., and Voelker, G. M. (2010). Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *SIGKDD*, pages 105–114.

Chen, T., He, T., Benesty, M., et al. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4.

First (2007). Common vulnerability scoring system. `https://www.first.org/cvss/v2/guide`. [Online; accessed 01-May-2019].

Frei, S., May, M., Fiedler, U., and Plattner, B. (2006). Large-scale vulnerability analysis. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 131–138. ACM.

Keras (2019). The python deep learning library. [Online; accessed 13-May-2019].

Mell, P., Scarfone, K., and Romanosky, S. (2006). Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6):85–89.

MITRE (2018). Common vulnerabilities and exposures (cve). [Online; accessed 01-May-2019].

NIST (2018). National vulnerability database. [Online; accessed 01-May-2019].

Petraityte, M., Dehghantanha, A., and Epiphaniou, G. (2018). A model for android and ios applications risk calculation: Cvss analysis and enhancement using case-control studies. *Cyber Threat Intelligence*, pages 219–237.

Sabottke, C., Suciu, O., and Dumitras, T. (2015). Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *USENIX Security Symposium*, pages 1041–1056.

Scarfone, K. and Mell, P. (2009). An analysis of cvss version 2 vulnerability scoring. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 516–525. IEEE Computer Society.

Security, O. (2018). Exploit database. [Online; accessed 01-May-2019].

Shahzad, M., Shafiq, M. Z., and Liu, A. X. (2012). A large scale exploratory analysis of software vulnerability life cycles. In *ICSE*, pages 771–781. IEEE.

Spark, C. (2017). Hyperparameter tuning. `https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f`.

Wang, L., Islam, T., Long, T., Singhal, A., and Jajodia, S. (2008). An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer.