Threat Copilot: Um sistema de recomendação para a modelagem de ameaças

Yuri Feitosa Negócio[1], **Nilton da Trindade Herthel Jr.**[1], **Juliana Dantas Medeiros**[1]

[1]Instituto Federal da Paraíba (IFPB)

{negocio.yuri, nilton.trindade}@academico.ifpb.edu.br

juliana.medeiros@ifpb.edu.br

Abstract. *Secure software development processes aim to ensure that products can operate effectively even in the face of attacks. One relevant activity in a secure development lifecycle is identifying security flaws proactively through threat modeling. Various threat modeling methods have been proposed in both industry and academic research. Despite this, integrating this activity into development teams has not been straightforward. This paper introduces a tool named "Threat Copilot", which is a knowledge-based recommendation system. Its purpose is to identify threats by comparing them to pre-existing threat models within an organization. Preliminary results indicate that the tool can be useful in facilitating threat elicitation.*

Resumo. *Os processos de desenvolvimento seguro de software buscam garantir que a construção do software resulte em produtos capazes de funcionar adequadamente mesmo diante de ataques. Uma das atividades mais relevantes em um processo de desenvolvimento seguro é a identificação antecipada das falhas de segurança através da modelagem de ameaças. Diversos métodos de modelagem de ameaças foram propostos tanto na indústria como na academia, mesmo assim a adoção desta atividade ainda não tem sido trivial pelos times de desenvolvimento. Este trabalho apresenta a ferramenta Threat Copilot, um sistema de recomendação baseado em conhecimento que identifica ameaças com base na similaridade com modelos de ameaças pré-existentes em uma organização. Resultados preliminares indicam que a ferramenta pode ser útil em facilitar a elicitação de ameaças.*

1. Introdução

Com a expansão da transformação digital dos governos e da sociedade, temos uma maior demanda para o desenvolvimento e adoção em curto prazo dos sistemas de informação para a execução de atividades críticas. Ao mesmo tempo, também é notório o aumento da exploração de falhas de segurança nos sistemas de informação por agentes maliciosos[BeyondTrust 2023]. Em busca de tornar a construção e o software mais seguros, diversas metodologias de desenvolvimento seguro foram propostas na academia e indústria [Kudriavtseva and Gadyatskaya 2022]. Estes métodos buscam, através da adoção de práticas e ferramentas de segurança durante as etapas do ciclo de vida de desenvolvimento, reduzir a exposição do software a agentes maliciosos. Uma das práticas comuns nos métodos é a modelagem de ameaças, que

tem sido qualificada como uma das mais importantes atividades na busca por desenvolver software seguro [Bernsmed et al. 2022] [Xiong and Lagerström 2019]. Seu uso não é restrito a sistemas web, encontrando sucesso em áreas como internet das coisas [Casola et al. 2019], veículos autônomos [Ghosh et al. 2023] e cidades inteligentes [Tok and Chattopadhyay 2023].

Um método de modelagem de ameaças é uma abordagem para criação de uma abstração de um sistema de software para identificar as habilidades e objetivos de um atacante e gerar uma lista de ameaças possíveis [Shull 2016]. A partir das ameaças identificadas, é possível criar requisitos de segurança que sejam atendidos através da adoção de controles, reduzindo assim a exposição ao risco.

A modelagem de ameaças ainda é uma disciplina que possui baixo nível de maturidade nos termos de pesquisa, suporte de ferramentas e abordagens práticas [Yskout et al. 2020]. Estas dificuldades se somam aos desafios do seu uso em projetos que adotam metodologias ágeis, é comumente ignorada quando os times de desenvolvimento adotam metodologias ágeis [Bernsmed et al. 2022].

A maioria do trabalho executado na modelagem de ameaças ainda é feito manualmente [Xiong and Lagerström 2019]. Neste contexto, este artigo apresenta a ferramenta *Threat Copilot*, que possui como objetivo facilitar a adoção da modelagem de ameaças por times ágeis de desenvolvimento em organizações. Para isso, a ferramenta armazena e integra os modelos de ameaças com os repositórios de produtos, de ativos e as bases de dados de conhecimento sobre fraquezas (CWE)[1] e ataques (CAPEC)[2]. A ferramenta permite a automação da fase de identificação de ameaças através de um sistema de recomendação baseado em conhecimento. Nesta função, quando o usuário submete um novo design de um produto de software, a ferramenta identifica, nos modelos de ameaças já armazenados, a similaridade das interações, dos elementos e dos termos e sugere as ameaças mais prováveis para o novo modelo. Desta forma, facilita a criação do modelo de ameaças por desenvolvedores sem um completo aprofundando em segurança da informação.

O restante deste artigo está organizado da seguinte forma: A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a arquitetura da ferramenta e o o sistema de recomendação. A Seção 4 apresenta uma breve prova de conceito do uso da ferramenta. Por fim, a Seção 5 apresenta as Conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Em [Granata and Rak 2023] é apresentada uma revisão sistemática da literatura sobre técnicas de automação de modelagem de ameaças. A partir de uma análise sistemática das pesquisas acadêmicas nos repositórios de publicação, foram selecionadas quatro ferramentas de código aberto para comparação: Microsoft TMT[3], OWASP Threat Dragon[4], SLA-generator [5] e PyTM[6]. A Microsoft TMT utiliza o relacionamento entre dois elementos de um modelo de ameaças para determinar a lista de ameaças. A OWASP Threat

[1]https://cwe.mitre.org/
[2]https://capec.mitre.org/
[3]https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling
[4]https://owasp.org/www-project-threat-dragon/
[5]https://github.com/DanieleGranata94/SlaGenerator
[6]https://github.com/izar/pytm

Dragon utiliza o tipo de elemento e um alguns parâmetros para sugerir as ameaças. A SLA-generator utiliza o tipo do elemento associado com o protocolo de comunicação para determinar a lista de ameaças. Por fim, a PyTM permite a descrição do modelo de ameaças na linguagem de programação Python e a lista de ameaças é determinada apenas pelo tipo de elemento existente na descrição.

A ferramenta *Threat Copilot* difere das propostas apresentadas na RSL no fato de que a geração da lista de ameaças ocorre com base na experiência prévia de modelos gerados pelos analistas de segurança da organização, funcionando como um sistema de recomendação. Outra diferença relevante é que o catálogo de ameaças e de termos de classificação dos ativos são dinâmicos. Desta forma, permite que o modelo de ameaça tenha sua lista de ameaças atualizada quando novos modelos forem inseridos.

Os sistemas de recomendação podem ser utilizados em contextos diferentes dos inicialmente projetados. Segundo [Aggarwal et al. 2016] os tipos básicos de sistemas de recomendação são: de filtragem colaborativa; baseado em conteúdo; baseado em conhecimento; e híbridos. A *Threat Copilot* se classifica como um sistema de recomendação baseado em conhecimento de propósito específico para a engenharia de software *(Recommender Systems for Software Engineering - RSSE)*. [Elkamel et al. 2016] descreve um RSSE no qual auxilia a fase de design de projetos de software através da sugestão de de classes UML já existentes durante a construção de diagramas de classes UML. A partir da criação de uma classe no diagrama sob construção, o sistema de recomendação pesquisa em sua base de dados a similaridade do nome, atributos e operações da classe criada com as classes UML pré-existentes, sugerindo em sequência de classes comumente utilizadas em composição com a classe criada. A *Threat Copilot* se assemelha no objetivo. Entretanto, além do nome, ela utiliza para similaridade um vocabulário de termos para classificar de forma hierárquica os elementos do design do modelo de ameaças.

3. A Ferramenta Threat Copilot

Nesta seção é apresentado o funcionamento da ferramenta como um sistema de recomendação e a arquitetura da ferramenta. A Figura 1 ilustra o funcionamento da ferramenta como parte de um ciclo de modelagem de ameaças. Inicialmente, na fase *(Application Design)* o time de desenvolvimento de produto realiza o desenho do escopo do software da modelagem de ameaça utilizando uma ferramenta de diagramação. O resultado desta fase é um arquivo no formato específico da ferramenta.

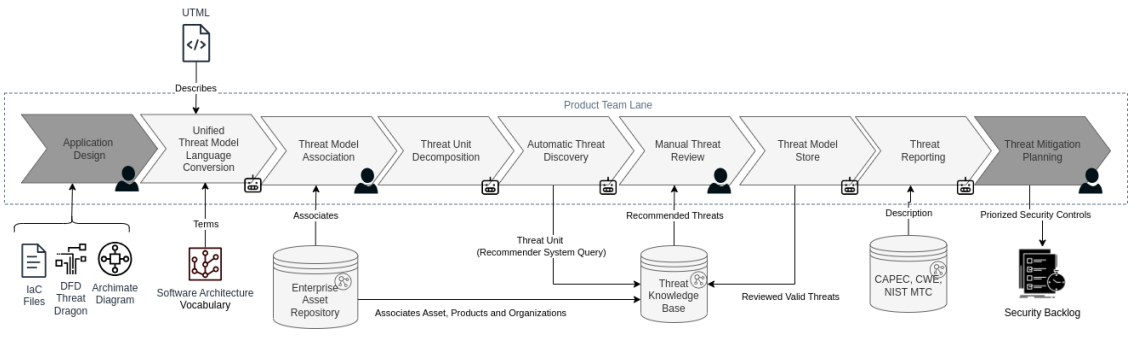

Figura 1. Funcionamento da Ferramenta

Na fase *(UTML Conversion)* o arquivo gerado pela fase anterior será convertido

para uma linguagem de modelo de ameaças unificada - UTML. Essa conversão transforma o modelo específico em um modelo genérico no formato YAML que pode ser interpretado pela ferramenta. A Figura 2 detalha um exemplo básico de descrição com UTML.

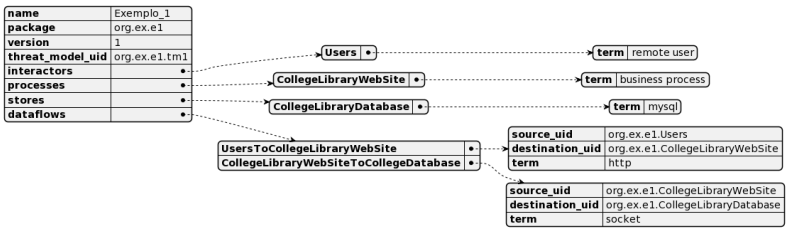

Figura 2. Descrição de um Modelo de Ameças com UTML

Neste exemplo de descrição temos um diagrama de fluxo de dados (DFD) que descreve um sistema básico web no qual o ator *(Users)* classificado como *(remote user)* possui um fluxo de dados *(UsersToCollegeLibraryWebSite)* classificado como (http) conectado ao processo *(CollegeLibraryWebSite)*, que por sua vez possui um fluxo de dados *(CollegeLibraryWebSiteToCollegeDatabase)* classificado como *(socket)* com o armazenamento *(CollegeLibraryDatabase)* classificado como *(mysql)*.

Após a conversão e inserção do modelo de ameaças na ferramenta, o *Threat Copilot* realiza a ligação entre os termos classificadores descritos nos elementos do modelo e os termos existentes no vocabulário da ferramenta. O vocabulário é dividido em um por elemento e define uma hierarquia recursiva sobre os termos. A Figura 3 descreve o modelo conceitual adotado para representar a hierarquia de termos.

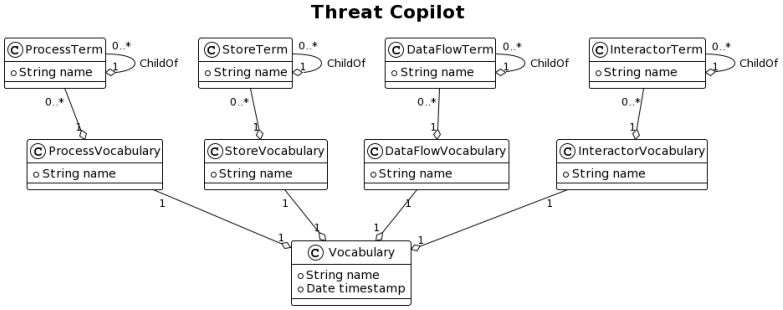

Figura 3. Modelo Conceitual do Vocabulário

A partir da adoção da hierarquia de termos será possível classificar a similaridade entre um termo e outro considerando a posição do termo na árvore do vocabulário. Esta definição aumenta a precisão e extensibilidade para a identificação das ameaças.

Na fase *(Threat Model Association)* é realizada a ligação entre os ativos *(Asset)* de infraestrutura da organização com os elementos do modelo de ameaças. Desta forma, tornando possível a rastreabilidade até os ativos, mesmo a partir de abstrações descritas nos modelos e não gerenciados pela organização. Por exemplo, na descrição da Figura 2 o processo *(CollegeLibraryWebSite)* poderia estar associado a um conjunto de ativos de configuração, como por exemplo, servidores *(srv1.local, srv2.local, etc.)*. Esta ligação entre o modelo de ameaças e o CMDB (*Configuration Management Database*) de uma organização permite que a partir de um determinado item de configuração seja possível

identificar quais ameaças ele está associado. Essa navegação é permitida pelo armazenamento na forma de grafo das informações no banco de dados. O resultado do modelo de ameaças pode ser visto no modelo conceitual apresentado na Figura 4.

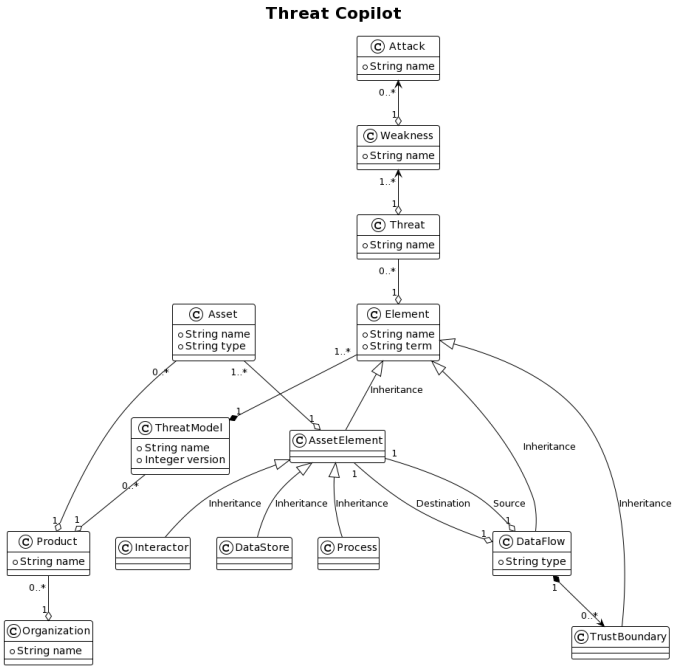

Figura 4. Modelo Conceitual

Além dos elementos já descritos, na Figura 4 temos os conceitos de Organização e Produtos. Estes permitem que modelos entre diferentes organizações possam ser reutilizados, pois eles serão consultados de forma única pelo sistema de recomendação.

Encerrada a fase anterior, temos a Fase *(Threat Unit Decomposition)* que irá recortar o modelo de ameaças em unidades menores *(threat units)* passíveis de verificação de similaridade. Uma unidade de ameaça é composta do nome do elemento, do tipo de elemento *(Interactor, Process, Store)*, da lista de fluxos de dados de entrada e da lista de fluxo de dados de saída.

Na Fase *(Automatic Threat Discovery)* todas as unidades de ameaças são enviadas como consulta à base de conhecimento *(Threat Knowledge Base)* da ferramenta para identificar unidades de ameaças similares. A ferramenta compara com todas as unidades de ameaças existentes na base de conhecimento, consolidando em um ranking ao final.

Para calcular a similaridade entre duas unidades de ameaças U_i e U_j a equação 1 é utilizada. Nela são considerados a soma com pesos da similaridade do nome, dos termos e entre termos dos fluxos de dados de entrada e de saída.

$$sim(U_i, U_j) = \frac{p_{nome} * sim_n(U_i, U_j)) + p_{termo} * sim_t(U_i, U_j) + p_{es} * sim_{es}(U_i, U_j))}{p_{nome} + p_{termo} + p_{es}} \quad (1)$$

Para calcular a similaridade entre os nomes das unidades de ameaças $sim_n(U_i, U_j)$ a equação (2) utiliza o algoritmo de similaridade de Jaccard com um filtro prévio de *stemming* para redução ao radical da palavra. Para calcular a similaridade entre dois termos

em um o vocabulário de termos é utilizada a equação (3). Ela transforma t_1 e t_2 em vetores preenchidos com 0s ou 1s de acordo com caminho percorrido até a localização do termo. Estes vetores são comparados utilizando a função de similaridade cosseno com pesos, estes pesos foram atribuídos de forma crescente até o valor do último nível da árvore, sendo assim, quanto maior o caminho compartilhado na árvore de vocabulário maior a similaridade entre os termos.

$$sim_n(n_1, n_2) = jaccard(stemming_{filter}(n_1), stemming_{filter}(n_2)) \quad (2)$$

$$sim_t(t_1, t_2, pesos) = cosine_{weighted}(t_1, t_2, pesos) \quad (3)$$

Para calcular a similaridade do fluxo de dados dos termos de entrada e saída de uma unidade de ameaça é utilizada a equação (4). Nesta equação temos que $termos_{entrada}(U_j) = te_{i1}, te_{i2}, ..., te_{in}$ e $termos_{saida}(U_j) = ts_{i1}, ts_{i2}, ..., ts_{im}$ representam o conjunto dos termos dos fluxos de entrada e saída da unidade. As funções sim_e e sim_s calculam o maior valor de similaridade de um termo com um conjunto de termos que da unidade que está sendo comparada. As variáveis n e m representam os números dos termos de entrada e saída.

$$sim_{es}(U_i, U_j) = \frac{\sum_{k=1}^{n} \frac{sim_e(te_{ik}, termos_{entrada}(U_j))}{n} + \sum_{w=1}^{m} \frac{sim_s(ts_{uw}, termos_{saida}(U_j))}{m}}{2} \quad (4)$$

Quanto maior a similaridade entre unidades de ameaças, maior a probabilidade de uma ameaça associada ao elemento principal existir no modelo sob avaliação. Neste sentido, o sistema de recomendação sugere de forma ordenada as ameaças mais prováveis para o modelo sob avaliação. Após a sugestão, na fase *(Manual Threat Review)* o analista aprova ou não ameaças sugeridas e, se necessário, realiza a inclusão de novas ameaças associadas aos elementos do modelo de ameaças.

Na fase *(Threat Model Store)* o novo modelo de ameaças é armazenado na base de conhecimentos e pode ser utilizado em futuros modelos. Após esta fase o relatório final de ameaças pode ser gerado e disponibilizado para as partes interessadas. O relatório além do detalhamento do modelo terá as descrições das ameaças associadas as bases de conhecimento da indústria como a CWE e CAPEC. Ao final temos a fase externa *(Threat Mitigation Planning)* cujo resultado final será o backlog de segurança que será atendido pelo time de desenvolvimento.

A ferramenta foi desenvolvida em Python utilizando o framework Flask 2.3[7]. Nela são utilizadas duas bases de dados: a SQLite[8] para os serviços de gestão de identidade; e a outra uma base dados orientada a grafos Neo4J[9] para o armazenamento dos dados.

4. Prova de Conceito

Para a demonstração preliminar da eficácia da ferramenta foram convertidos dois modelos de ameaças reais de uma organização. O modelo de ameaças do produto X foi descrito pelo time de segurança de aplicações que identificou ao total treze ameaças. O modelo foi convertido para UTML, armazenado no *Threat Copilot* e as ameaças foram associadas

[7]https://flask.palletsprojects.com/en/2.3.x/
[8]https://www.sqlite.org/
[9]https://neo4j.com/

com os elementos do DFD (processos, armazenamento e atores). Ao final deste processo o modelo resultou em nove unidades de ameaças pesquisáveis que serão utilizadas para sugerir ameaças aos novos modelos.

O segundo modelo de ameaças do produto Y foi construído por um analista de segurança que manualmente identificou quatro ameaças ao total. O objetivo para a prova de conceito foi utilizar este modelo, sem a definição inicial das ameaças identificadas pelo analista, para que o *Threat Copilot* recomende as ameaças mais prováveis de serem reutilizadas com base no modelo de ranqueamento de similaridade com as unidades de ameaças definidas no produto X. O modelo de ameaças foi convertido para a UTML e resultou em doze unidades de ameaças. Estas unidades de ameaças foram utilizadas para serem comparadas com as nove unidades de ameaças do produto X. Ressalta-se que cada unidade de ameaça pode ter uma ou mais ameaças associadas e o conjunto dessas ameaças será o resultado observável da recomendação. Ao final do processamento a lista total de recomendações das ameaças foi ranqueada utilizando o limite inferior de similaridade de 0.55. A Tabela 1 apresenta o resultado destas recomendações.

Nome	Elemento (DFD)	Ameaça(s)	Similaridade
APIGateway	Processo	1,2,3,4,6,9	1.0
AppServer-A	Processo	4,5,6,7	0.8
AppServer-B	Processo	1,2,4,5,6,7	0.75
AppServer-B	Processo	10	0.726
FileServer-B	Processo	10	0.64
PDFFile-A	Armazenamento	13	0.64
AppServer-A	Processo	1,2	0.58

Tabela 1. Recomendações das Ameaças

Como o primeiro modelo de ameaças em seu design também possui um *API Gateway* com as mesmas características do segundo modelo a similaridade alcançou seu valor máximo. As demais ameaças foram sendo indicadas para os demais elementos que compõe o segundo modelo de ameaças. Um ponto relevante é que a ameaça de número treze referente a falsificação de documento digital, além de não ser indicada pelo analista de segurança, representa um risco significativo para o produto do segundo modelo de ameaças. Inversamente, a ameaça 10 do processo *FileServer-B* embora bem ranqueada ela não existe para o modelo avaliado. O que sugere a necessidade de validar os pesos e o modelo com um dataset de modelos de ameaças mais extenso.

5. Considerações Finais

Para demonstracão o código-fonte, documentação e instrução de instalação estão disponibilizadas no repositório da *Threat Copilot* [10]. A demonstração da ferramenta será realizada através de um ambiente disponibilizado na nuvem pelos próprios autores. A execução da ferramenta será feita da seguinte forma: (a) apresentação da descrição da UTML e vocabulários; (b) inclusão do novo modelo de ameaças; (c) apresentação das recomendações obtidas.

[10]https://github.com/yurix/threatcopilot

Neste artigo apresentamos a ferramenta *Threat Copilot* que visa facilitar a adoção da modelagem de ameaças por times de desenvolvimento através da reutilização de conhecimento de modelos anteriores. A ferramenta ainda está sob desenvolvimento, e como trabalhos futuros temos: (a) expandir o dataset de modelos de ameaças; (b) evoluir para um vocabulário mais representativo; (c) evoluir a usabilidade no seu uso; (d) melhorar a acurácia do recomendador, considerando as decisões unárias de aplicabilidade ou não de uma ameaça; e (e) avaliar o uso da ferramenta utilizando um modelo de aceitação tecnológica TAM [Davis et al. 1989].

Referências

Aggarwal, C. C. et al. (2016). *Recommender systems*, volume 1. Springer.

Bernsmed, K., Cruzes, D. S., Jaatun, M. G., and Iovan, M. (2022). Adopting threat modelling in agile software development projects. *Journal of Systems and Software*, 183:111090.

BeyondTrust (2023). Microsoft vulnerabilities report 2023. Technical report, BeyondTrust. Disponível em: https://www.beyondtrust.com/resources/whitepapers/microsoft-vulnerability-report. Acesso em 15/06/2023.

Casola, V., De Benedictis, A., Rak, M., and Villano, U. (2019). Toward the automation of threat modeling and risk assessment in iot systems. *Internet of Things*, 7:100056. B1.

Davis, F. D. et al. (1989). Technology acceptance model: Tam. *Al-Suqri, MN, Al-Aufi, AS: Information Seeking Behavior and Technology Adoption*, pages 205–219.

Elkamel, A., Gzara, M., and Ben-Abdallah, H. (2016). An uml class recommender system for software design. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8.

Ghosh, S., Zaboli, A., Hong, J., and Kwon, J. (2023). An integrated approach of threat analysis for autonomous vehicles perception system. *IEEE Access*, 11:14752–14777.

Granata, D. and Rak, M. (2023). Systematic analysis of automated threat modelling techniques: Comparison of open-source tools. *Software Quality Journal*, pages 1–37.

Kudriavtseva, A. and Gadyatskaya, O. (2022). Secure software development methodologies: A multivocal literature review. (arXiv:2211.16987).

Shull, F. (2016). Evaluation of competing threat modeling methodologies. Technical report, Software Engineering Institute - Carnegie Mellon University.

Tok, Y. C. and Chattopadhyay, S. (2023). Identifying threats, cybercrime and digital forensic opportunities in smart city infrastructure via threat modeling. *Forensic Science International: Digital Investigation*, 45:301540.

Xiong, W. and Lagerström, R. (2019). Threat modeling – a systematic literature review. *Computers Security*, 84:53–69.

Yskout, K., Heyman, T., Van Landuyt, D., Sion, L., Wuyts, K., and Joosen, W. (2020). Threat modeling: from infancy to maturity. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, page 9–12, Seoul South Korea. ACM.