LLM como "Copiloto Virtual de Segurança": Apoio à Geração de Requisitos, Evil User Stories e Casos de Testes

André Carvalho
Instituto Metrópole Digital, Universidade Federal do Rio Grande do Norte
Natal, Brasil
andre.carvalho@ufrn.br

Eiji Adachi
Instituto Metrópole Digital, Universidade Federal do Rio Grande do Norte
Natal, Brasil
eijiadachi@imd.ufrn.br

Resumo

Este trabalho propõe o uso de Large Language Models (LLMs) como "copiloto virtual de segurança" para apoiar equipes de desenvolvimento e profissionais de segurança ao longo do ciclo de desenvolvimento de software. A proposta parte de um contexto em que especialistas em segurança frequentemente atuam de forma sobrecarregada, enquanto equipes de desenvolvimento possuem baixa proficiência na identificação de ameaças e na definição de requisitos de segurança. Nesse cenário, os LLMs são utilizados como mecanismos de apoio ao raciocínio e à tomada de decisão, e não como substitutos de especialistas humanos. O trabalho apresenta um fluxo assistido por LLMs integrado ao processo de desenvolvimento já existente nas organizações, apoiando a identificação de ameaças, a elicitação de requisitos de segurança, a geração de *evil user stories* e a construção de cenários conceituais de testes de segurança a partir de requisitos funcionais, regras de negócio e histórias de usuário. A abordagem combina práticas inspiradas nos Security Touchpoints de Gary McGraw com técnicas de Threat Modeling baseadas em STRIDE, utilizando LLMs como suporte contextual para ampliar a percepção de riscos e vulnerabilidades durante o desenvolvimento. O artefato proposto é estruturado como um conjunto de microsserviços integráveis ao processo de desenvolvimento, buscando incorporar práticas de segurança de forma contínua, acessível e escalável em ambientes com recursos limitados. Espera-se investigar o potencial do uso de LLMs como assistentes de segurança para apoiar a geração progressiva de artefatos de segurança rastreáveis e potencialmente mais completos, contribuindo para ampliar a identificação de ameaças e mecanismos de mitigação ao longo do SSDLC e para a evolução de práticas DevSecOps apoiadas por LLMs.

Palavras-chave

Segurança de Software, Engenharia de Segurança, Large Language Models, Threat Modeling, STRIDE, DevSecOps

1 Introdução

A crescente dependência de sistemas digitais em processos críticos tornou a segurança de software um requisito essencial de qualidade, e não apenas uma preocupação adicional do desenvolvimento. A segurança da informação é tradicionalmente definida pela preservação da confidencialidade, integridade e disponibilidade da informação [5].

O aumento da frequência e da sofisticação dos ataques cibernéticos tem ampliado a necessidade de adoção de práticas sistemáticas de segurança ao longo do ciclo de desenvolvimento de software [1, 9]. Nesse contexto, o *Secure Software Development Life Cycle* (SSDLC) busca integrar atividades de segurança às diferentes fases do desenvolvimento, desde a elaboração de requisitos até manutenção e evolução do sistema [4, 20]. Além disso, práticas alinhadas ao conceito de *shift-left security* têm enfatizado a importância da identificação de riscos o mais cedo no ciclo de desenvolvimento, antecipando a identificação de vulnerabilidades [20].

Entre as práticas associadas ao SSDLC, destaca-se a de *Threat Modeling*, considerada uma abordagem relevante para identificação sistemática de ameaças e riscos ainda nas fases de design, mas podendo e devendo ser aplicada a todo fluxo de desenvolvimento [10]. Entretanto, conforme discutido por [20] e [19], a integração efetiva de atividades de segurança em ambientes ágeis ainda enfrenta obstáculos significativos. A necessidade de conhecimento especializado, combinada à pressão por entregas rápidas e documentação reduzida, frequentemente dificulta a adoção consistente de práticas de segurança por equipes de desenvolvimento e *Quality Assurance* (QA).

Além disso, embora existam estratégias voltadas à incorporação de segurança em ambientes ágeis — como *Security Champions*, *security stories* e *Protection Poker* — tais abordagens ainda dependem fortemente da disponibilidade de especialistas em segurança [20]. Mesmo em organizações de maior porte, esses profissionais frequentemente atuam de forma consultiva e precisam atender simultaneamente múltiplas equipes e projetos, dificultando sua participação contínua ao longo do SSDLC. Em equipes com baixa proficiência em segurança ou com recursos limitados, esse cenário torna-se ainda mais evidente, fazendo com que atividades importantes como o *Threat Modeling*, definição de requisitos de segurança e elaboração de cenários de testes, sejam frequentemente negligenciadas ou tratadas como um *afterthought*, contribuindo para a permanência de falhas de lógica e design em sistemas colocados em produção [18].

Recentemente, *Large Language Models* (LLMs) têm demonstrado potencial para apoiar tarefas complexas relacionadas à engenharia de software e segurança, incluindo interpretação de documentação técnica, geração de código seguro e apoio à modelagem de ameaças [1, 9]. Entretanto, ainda existem poucas evidências sobre abordagens estruturadas capazes de utilizar LLMs para apoiar, de forma encadeada, a geração de artefatos de segurança derivados de requisitos funcionais, especialmente em contextos de baixa proficiência em segurança.

Diante desse cenário, este trabalho investiga o uso de LLMs como mecanismo de apoio a especialistas e equipes com baixa proficiência em segurança, buscando ampliar a escala de aplicação de práticas do SSDLC. Especificamente, propõe-se uma abordagem baseada em um fluxo encadeado de geração de artefatos de segurança, no

qual requisitos funcionais são progressivamente transformados em requisitos de segurança, *evil user stories* e cenários conceituais de testes orientados por ameaças. A abordagem utiliza a metodologia STRIDE (*Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* e *Elevation of Privilege*) como mecanismo estruturante de identificação de ameaças e busca apoiar a tomada de decisão.

Além de investigar a viabilidade da arquitetura proposta, busca-se compreender em que medida o fluxo encadeado favorece a geração de artefatos de segurança rastreáveis e com maior cobertura em relação às vulnerabilidades representadas nos requisitos funcionais analisados.

Para orientar esta investigação, este trabalho busca responder às questões de pesquisa apresentadas na Tabela 1.

Tabela 1: Questões de pesquisa

QP1 – Como LLMs orientadas pela metodologia STRIDE podem apoiar a geração sistemática de requisitos de segurança?
QP2 – Em que medida uma abordagem encadeada baseada em LLMs favorece a geração de artefatos de segurança completos, rastreáveis e potencialmente úteis para identificação de vulnerabilidades?

2 Fundamentação e Trabalhos Relacionados

2.1 Secure Software Development Life Cycle (SSDLC)

O *Secure Software Development Life Cycle* (SSDLC) consiste na incorporação sistemática de práticas de segurança ao longo do ciclo de desenvolvimento de software, buscando reduzir riscos e vulnerabilidades desde as fases iniciais do projeto [4]. Modelos amplamente utilizados, como o *Microsoft Secure Development Lifecycle* (SDL), o *OWASP Software Assurance Maturity Model* (SAMM) e os *Software Security Touchpoints* de McGraw, enfatizam atividades como levantamento de requisitos de segurança, modelagem de ameaças e testes orientados a riscos [3, 8].

Dentre esses modelos, os *Touchpoints* de McGraw apresentam especial relevância para esta pesquisa por enfatizarem a integração de segurança ao processo de construção do software, e não apenas controles periféricos [8]. Em particular, três atividades são centrais ao presente trabalho: definição de requisitos de segurança, análise de risco arquitetural (*Threat Modeling*) e testes de segurança baseados em riscos. A literatura aponta que a ausência dessas práticas, especialmente em equipes com baixa maturidade em segurança, favorece a introdução de vulnerabilidades ainda nas fases iniciais do desenvolvimento [7, 20].

2.2 Threat Modeling e Artefatos de Segurança

No contexto do SSDLC, o *Threat Modeling* é reconhecido como uma abordagem estruturada para identificação antecipada de ameaças e definição de mecanismos de mitigação [10, 17]. Entre os modelos existentes, o STRIDE destaca-se por categorizar ameaças em seis classes principais: *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* e *Elevation of Privilege*, permitindo sistematizar a análise de riscos durante o design da aplicação [12, 17].

A identificação dessas ameaças subsidia diretamente a geração de artefatos de segurança, como requisitos de segurança, *misuse cases*, *evil user stories* e cenários de testes orientados a vulnerabilidades [6, 11]. Os *misuse cases* descrevem comportamentos abusivos realizados por agentes maliciosos que exploram funcionalidades legítimas do sistema. Em contextos ágeis, esse raciocínio pode ser representado por meio de *evil user stories*, histórias formuladas sob a perspectiva de um agente malicioso que descrevem um objetivo abusivo, a forma como uma funcionalidade pode ser explorada e o impacto esperado sobre o sistema. Neste trabalho, as *evil user stories* são geradas a partir dos requisitos de segurança e utilizadas como artefatos intermediários para orientar a elaboração de cenários conceituais de testes de segurança. Em ambientes ágeis, especialmente aqueles com baixa proficiência em segurança, tais artefatos auxiliam a traduzir riscos abstratos em mecanismos concretos de mitigação e validação [2, 14]. Entretanto, sua elaboração ainda depende significativamente de conhecimento especializado, tornando sua adoção inconsistente em equipes com recursos limitados [20].

Além disso, taxonomias amplamente adotadas na indústria, como *Common Weakness Enumeration* (CWE) e *Common Attack Pattern Enumeration and Classification* (CAPEC), oferecem mecanismos estruturados para representar fraquezas, padrões de ataque e critérios de verificação de segurança, reduzindo ambiguidades e ampliando a rastreabilidade dos artefatos produzidos [13].

2.3 LLMs Aplicados à Segurança

Avanços recentes em LLMs têm impulsionado sua aplicação em atividades relacionadas à Engenharia de Software, incluindo geração de documentação, suporte à implementação e automação de testes. No domínio da segurança, estudos recentes indicam potencial para apoiar tarefas como priorização de vulnerabilidades, interpretação de documentação técnica, geração de recomendações de segurança e apoio à modelagem de ameaças [1, 9].

Entretanto, limitações importantes permanecem, incluindo alucinações, inconsistências entre execuções e elevada sensibilidade à qualidade dos *prompts* [9]. Para reduzir esses problemas, a literatura tem investigado estratégias como *prompt engineering*, arquiteturas baseadas em *Retrieval-Augmented Generation* (RAG), utilização de bases externas de conhecimento e validação humana, buscando aumentar a consistência e a confiabilidade das respostas produzidas pelos modelos [1, 9].

Assim, embora os LLMs apresentem potencial para apoiar atividades de segurança ao longo do *SSDLC*, seu uso ainda demanda mecanismos que reduzam respostas não fundamentadas e preservem a rastreabilidade das recomendações produzidas. Neste trabalho, os LLMs são tratados como mecanismos de apoio cognitivo, buscando ampliar a capacidade operacional de especialistas em segurança e de equipes com baixa proficiência, sem substituir a tomada de decisão humana.

2.4 Trabalhos Relacionados

Estudos recentes têm proposto arquiteturas especializadas baseadas em LLMs para apoiar diferentes atividades do SSDLC. O trabalho de Bedoya et al. [1] explora a utilização de LLMs em práticas de

DevSecOps e *Security Chaos Engineering*, demonstrando potencial na geração de recomendações e explicações relacionadas à segurança. Já o *ThreatLens* [16] propõe uma arquitetura multiagente baseada em LLMs voltada à automação de *Threat Modeling* e geração de planos de testes para verificação de segurança em hardware.

De forma semelhante, o *ThreatFinderAI* [21] investiga o uso de LLMs para automação da modelagem de ameaças em sistemas que integram IA, utilizando grafos de ataque e bases de conhecimento para apoiar a descoberta de vulnerabilidades. Mais recentemente, Sabetta et al. [15] propuseram uma arquitetura baseada em RAG para geração de requisitos de segurança fundamentados em OWASP, utilizando um mecanismo de verificação de cobertura e geração compensatória para reduzir omissões durante a elicitação de requisitos.

Embora relevantes, essas abordagens concentram-se predominantemente em atividades específicas do SSDLC, como geração de requisitos de segurança ou automação de *Threat Modeling*. Diferentemente dessas abordagens, este trabalho investiga uma arquitetura encadeada baseada em LLMs que integra, em um único fluxo, a geração de requisitos de segurança, *evil user stories* e cenários conceituais de testes a partir de requisitos funcionais, utilizando *STRIDE* como mecanismo estruturante. Além disso, a proposta posiciona os LLMs como mecanismos de apoio cognitivo ao SSDLC, buscando ampliar a capacidade operacional de especialistas e equipes com baixa proficiência em segurança.

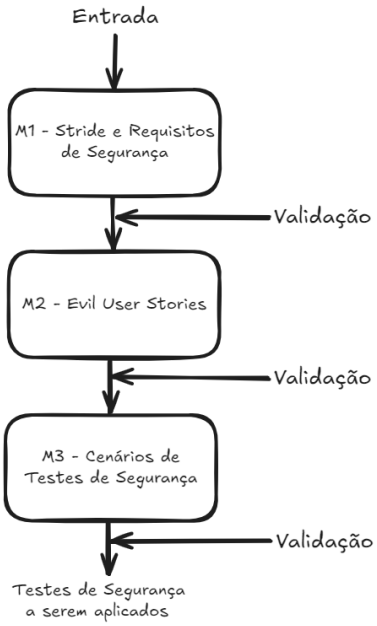

Figura 1: Organização da solução proposta
Fonte: Elaborado pelo autor

3 Metodologia

Esta pesquisa adota uma abordagem exploratória apoiada pela construção de um protótipo arquitetural e por uma avaliação experimental preliminar utilizando aplicações deliberadamente vulneráveis. A proposta investiga a viabilidade de um fluxo encadeado baseado em LLMs para geração progressiva de artefatos de segurança a partir de requisitos funcionais, utilizando *Threat Modeling* orientado por STRIDE e bases consolidadas de conhecimento em segurança. Considerando a natureza exploratória do estudo, o objetivo desta etapa não é comparar diferentes modelos ou estratégias de *prompting*, mas investigar a viabilidade do fluxo encadeado proposto e obter evidências iniciais sobre sua capacidade de produzir artefatos de segurança rastreáveis a partir de requisitos funcionais.

3.1 Arquitetura Proposta

A arquitetura proposta organiza a geração de artefatos de segurança em um fluxo encadeado, conforme apresentado na figura 1, permitindo a transformação progressiva de requisitos funcionais em requisitos de segurança, *evil user stories* e cenários conceituais de testes. O fluxo utiliza LLMs como mecanismo de apoio cognitivo, buscando ampliar a escala de aplicação de práticas do SSDLC sem substituir a tomada de decisão humana [1, 9].

A proposta incorpora uma estratégia baseada em *Retrieval-Augmented Generation* (RAG), fundamentada em bases de conhecimento derivadas de CWE, CAPEC, ATT&CK e D3FEND, buscando reduzir alucinações e ampliar a rastreabilidade dos artefatos produzidos. Similarmente ao mecanismo proposto por Sabetta et al. [15], a arquitetura utiliza grounding por conhecimento externo, porém estendendo sua aplicação além da geração de requisitos de segurança para um fluxo multiartefato.

Além do mecanismo de recuperação de conhecimento, a arquitetura incorpora estratégias de *few-shot learning* e um algoritmo automático de compensação. Inicialmente, são gerados artefatos orientados por STRIDE a partir dos requisitos funcionais e do contexto recuperado pelo RAG. Em seguida, um mecanismo compensatório realiza nova inferência para complementar ameaças ou mecanismos de mitigação potencialmente omitidos. Essa estratégia busca reduzir erros por omissão, problema frequentemente observado em LLMs aplicadas à engenharia de requisitos de segurança [9, 15].

3.2 Configuração Experimental

A implementação da arquitetura utilizou a OpenAI como provedora de LLM, empregando o modelo *gpt-4o-mini*, acessado por meio da API da OpenAI. A escolha desse modelo considerou sua disponibilidade durante o desenvolvimento da ferramenta e as recomendações da documentação oficial da OpenAI, que o caracteriza como um modelo de uso geral com bom equilíbrio entre desempenho e custo para diferentes tarefas.

Todas as respostas foram solicitadas exclusivamente em formato JSON estruturado, permitindo sua validação e processamento automático pelos microserviços da arquitetura. As chamadas utilizaram limite máximo de 8000 tokens de saída, enquanto a temperatura não foi configurada explicitamente, mantendo o comportamento padrão da API utilizada (*temperature*=1.0).

A arquitetura emprega uma estratégia de *few-shot learning* composta por dois exemplos específicos para cada microserviço (M1, M2 e M3). Ressalta-se, entretanto, que este trabalho não compara diferentes estratégias de *prompting*, uma vez que seu objetivo é investigar a viabilidade da arquitetura proposta.

Quando habilitado, o mecanismo de *Retrieval-Augmented Generation* (RAG) utiliza o modelo de embeddings *text-embedding-3-small* e o banco vetorial ChromaDB para recuperação de conhecimento proveniente das bases CWE, CAPEC, ATT&CK e D3FEND. Os experimentos utilizaram recuperação dos dez documentos mais relevantes (*top-k*=10), fornecendo *grounding* para apoiar a geração dos artefatos de segurança.

3.3 Preparação do Conjunto de Avaliação

A avaliação da arquitetura proposta foi conduzida utilizando casos de uso representativos derivados das aplicações vulneráveis *OWASP Juice Shop* e *OWASP crAPI*, amplamente empregadas como ambientes educacionais para segurança ofensiva e validação de vulnerabilidades. Foi feita a escolha dessas aplicações devido à disponibilidade de documentação estruturada dos desafios (*challenges*), presença de vulnerabilidades conhecidas e diversidade de cenários relacionados ao *OWASP Top 10*.

Para construção do conjunto de entrada na pipeline, foi realizada uma curadoria manual dos casos de uso e requisitos funcionais relacionados aos desafios presentes em ambas as aplicações a partir da documentação oficial. A elaboração dos requisitos ocorreu individualmente para cada *challenge*, buscando reconstruir os fluxos esperados da aplicação com base em documentação oficial e artefatos disponibilizados pelos projetos.

A curadoria contemplou 10 casos de uso do *OWASP Juice Shop* e 5 casos de uso do *OWASP crAPI*, selecionados a partir dos fluxos funcionais documentados pelas aplicações e relacionados aos *challenges* considerados na avaliação. Para cada caso de uso, foram elaborados requisitos funcionais individualmente, posteriormente utilizados como entrada do primeiro microserviço da arquitetura, responsável pela geração inicial de requisitos de segurança orientados por STRIDE.

No caso do *OWASP Juice Shop*, a curadoria considerou a documentação oficial dos desafios, a documentação do projeto e inspeção dos componentes relevantes da aplicação disponibilizados pelo repositório oficial. Para o *OWASP crAPI*, a curadoria utilizou a documentação oficial do projeto, descrição dos desafios, especificação OpenAPI, documentação do fluxo esperado (*happy path*) e artefatos disponibilizados no repositório oficial.

3.4 Mapeamento de Vulnerabilidades e Definição do Ground Truth

Como estratégia de avaliação, foi estabelecida uma referência entre os desafios documentados pelas aplicações e possíveis vulnerabilidades associadas representadas na taxonomia *Common Weakness Enumeration* (CWE). Diferentemente de abordagens automatizadas de classificação, o processo de associação foi conduzido manualmente pelo pesquisador, por meio da comparação entre a descrição textual dos *challenges* e as definições presentes na base oficial do CWE.

Ao todo, foram construídos conjuntos curados contendo 18 associações entre *challenges* e CWEs para o *OWASP Juice Shop* e 15 associações para o *OWASP crAPI*. Esse mapeamento buscou reduzir ambiguidades semânticas e ampliar a rastreabilidade entre vulnerabilidades conhecidas e os artefatos de segurança gerados pela arquitetura proposta.

Para definição do *ground truth*, foram considerados exclusivamente o nome da vulnerabilidade e sua categoria conforme documentados oficialmente pelas aplicações avaliadas. Dessa forma, o processo de comparação adotou como referência os comportamentos vulneráveis explicitamente descritos pelos próprios projetos.

3.5 Avaliação de Cobertura dos Artefatos

A partir dos requisitos funcionais curados e das referências estabelecidas entre *challenges* e CWEs, os artefatos produzidos pela arquitetura — requisitos de segurança, *evil user stories* e cenários conceituais de testes — foram comparados ao conjunto de vulnerabilidades esperadas de cada aplicação.

Inspirada na estratégia de cobertura proposta por Sabetta et al. [15], a análise utilizou duas perspectivas complementares de cobertura:

- **Cobertura Direta**: percentual de *challenges* do benchmark cuja vulnerabilidade correspondente foi explicitamente recuperada pelos artefatos produzidos pela arquitetura. Essa métrica considera a correspondência entre os artefatos gerados e os desafios efetivamente presentes nas aplicações avaliadas.
- **Cobertura Útil**: percentual de vulnerabilidades representadas pelas CWEs curadas do benchmark que apresentaram correspondência com CWEs sugeridas pela arquitetura, mesmo quando não havia correspondência direta com o *challenge* principal avaliado. Essa métrica busca capturar a capacidade da abordagem em recuperar fraquezas semanticamente relevantes e mecanismos potencialmente úteis à mitigação das vulnerabilidades avaliadas.

3.5.1 Ameaças à Validade. A avaliação conduzida apresenta limitações inerentes ao experimento adotado.

Este trabalho utiliza o modelo *GPT-4o-mini* como instância de LLM para demonstrar a viabilidade da arquitetura proposta. Não foi objetivo desta pesquisa comparar diferentes modelos fundacionais ou avaliar o impacto da escolha do LLM na qualidade dos artefatos produzidos. A comparação entre diferentes modelos fundacionais constitui uma oportunidade para trabalhos futuros.

Além disso, o desempenho observado pode variar conforme o modelo de LLM empregado, sua versão e eventuais atualizações do provedor. Embora a arquitetura utilize RAG, respostas estruturadas e mecanismo compensatório de geração para reduzir inconsistências e omissões, esses mecanismos não eliminam completamente a possibilidade de alucinações ou respostas não fundamentadas. Assim, a proposta deve ser compreendida como ferramenta de apoio à engenharia de segurança, e não como mecanismo de substituição da análise humana.

Embora o *OWASP Juice Shop* e o *OWASP crAPI* representem ambientes amplamente utilizados para treinamento e validação de vulnerabilidades, essas aplicações são deliberadamente vulneráveis, podendo não refletir a complexidade de sistemas corporativos reais.

Além disso, embora a associação entre *challenges* e CWEs tenha sido conduzida manualmente com base na documentação oficial dos projetos e nas descrições públicas do CWE, o processo ainda envolve interpretação e julgamento do pesquisador, especialmente em cenários nos quais uma mesma vulnerabilidade pode apresentar relações com outros CWEs relacionados.

Os resultados também dependem da qualidade, abrangência e atualização das bases de conhecimento utilizadas pelo mecanismo RAG (CWE, CAPEC, ATT&CK e D3FEND). Alterações nessas bases ou a utilização de outras fontes de conhecimento podem influenciar os artefatos produzidos.

Por fim, a métrica de cobertura útil envolve um julgamento contextual acerca da relevância dos artefatos produzidos para mitigação dos desafios avaliados. Assim, embora a abordagem permita avaliar utilidade prática além da simples correspondência direta de vulnerabilidades, diferentes avaliadores podem interpretar níveis diferentes de contribuição dos artefatos gerados.

4 Discussão e Resultados

A avaliação da arquitetura proposta foi conduzida utilizando as aplicações vulneráveis *OWASP Juice Shop* e *OWASP crAPI*, considerando os conjuntos curados de requisitos funcionais, associações entre *challenges* e CWEs e o *ground truth* definidos na metodologia.

A avaliação foi estruturada para responder às questões de pesquisa por meio da análise da capacidade da arquitetura em recuperar vulnerabilidades relevantes (QP1) e produzir artefatos de segurança rastreáveis e potencialmente úteis ao longo do fluxo encadeado (QP2). Para isso, foram utilizadas as métricas de cobertura direta e cobertura útil, que oferecem perspectivas complementares sobre os artefatos gerados. Os resultados foram analisados sob duas perspectivas complementares: (i) **cobertura direta**, baseada no *match* entre os artefatos produzidos e os *challenges* efetivamente presentes no benchmark; e (ii) **cobertura útil**, baseada na recuperação de vulnerabilidades semanticamente relacionadas por meio das associações curadas entre *challenges* e CWEs, permitindo identificar artefatos potencialmente relevantes mesmo quando não havia correspondência direta com o desafio principal avaliado.

Apesar disso, para *OWASP Juice Shop*, observou-se uma cobertura direta de 11,61% (13 cobertos ao final de 112 challenges mapeados no ground truth considerado). Apesar de relativamente baixa, a cobertura direta observada deve ser interpretada considerando o experimento adotado e objetivo da avaliação. Embora o benchmark do *OWASP Juice Shop* possua 112 *challenges*, a arquitetura proposta não recebeu como entrada a aplicação completa, mas apenas requisitos funcionais reconstruídos manualmente para um subconjunto de dez casos de uso representativos. Dessa forma, a cobertura direta não busca medir a identificação de todas as vulnerabilidades existentes no benchmark, mas sim a capacidade da abordagem em recuperar vulnerabilidades relevantes a partir do contexto funcional disponível. Assim, vulnerabilidades associadas a funcionalidades não contempladas pelos requisitos fornecidos não poderiam ser inferidas pela arquitetura, o que limita naturalmente a cobertura observada.

Observou-se também um comportamento distinto entre os microserviços ao longo do pipeline. O primeiro microserviço (M1), responsável pela geração de requisitos de segurança orientados por STRIDE, apresentou cobertura útil de 55,56% (20 artefatos com relacionadas a CWEs curadas de 36 artefatos gerados no total), indicando capacidade relevante de recuperação de vulnerabilidades relacionadas ao benchmark a partir dos requisitos funcionais fornecidos. Esse comportamento é compatível com estudos como o de Sabetta et al. [15], que também investigam a geração de requisitos de segurança apoiada por LLMs e RAG, reforçando o potencial desse tipo de abordagem para apoiar a elicitação inicial de requisitos de segurança.

Em contrapartida, o segundo microserviço (M2), responsável pela geração de *evil user stories*, apresentou cobertura útil inferior, de 18,00% (9 de 50), sugerindo maior sensibilidade à interpretação semântica e possível perda parcial de contexto ao transformar requisitos de segurança em cenários de abuso. Esse comportamento também pode ser explicado pela natureza das *evil user stories*. Diferentemente dos requisitos de segurança, que preservam explicitamente mecanismos de mitigação associados a vulnerabilidades específicas, as *evil user stories* reorganizam essas informações em narrativas centradas nos objetivos e comportamentos do atacante. Esse nível adicional de abstração tende a reduzir a correspondência explícita com as CWEs utilizadas como referência na avaliação, embora mantenha informações relevantes para atividades de modelagem de ameaças e elaboração de cenários de testes.

O terceiro microserviço (M3) apresentou cobertura útil de 53,57% (15 de 28), indicando recuperação consistente de vulnerabilidades relacionadas e preservação parcial da rastreabilidade.

Os resultados do *Juice Shop* sugerem que, embora a transformação intermediária em *evil user stories* introduza perda relativa de alinhamento com o benchmark, o pipeline tende a recuperar parte desse contexto durante a geração dos cenários de teste. Esse comportamento pode indicar que artefatos mais operacionais, como testes de segurança, favorecem a recuperação de vulnerabilidades semanticamente próximas às representadas nos *challenges*, especialmente em cenários amplamente documentados nas bases utilizadas pelo mecanismo de RAG. Além disso, a disponibilidade de documentação estruturada exerce influência positiva sobre a qualidade dos artefatos produzidos. No caso do OWASP Juice Shop, a reconstrução dos requisitos funcionais foi apoiada por documentação detalhada dos desafios, do projeto e da implementação da aplicação, favorecendo a contextualização fornecida à arquitetura. Esse resultado indica que abordagens baseadas em LLM podem beneficiar-se significativamente de informações de entrada bem estruturadas.

Em projetos reais, entretanto, a documentação frequentemente é incompleta, desatualizada ou inexistente. Nesses cenários, a aplicação da arquitetura poderá depender da utilização de fontes complementares de informação, como histórias de usuário, documentação arquitetural, especificações de APIs, código-fonte ou conhecimento de especialistas de domínio para reconstrução do contexto funcional. Dessa forma, além de evidenciar o potencial da arquitetura proposta, os resultados reforçam a importância da adoção de boas práticas de documentação ao longo do SSDLC, uma vez que a qualidade dos artefatos de entrada influencia diretamente a consistência dos artefatos de segurança gerados.

Na avaliação conduzida com o *OWASP crAPI*, observou-se cobertura direta de 38,89% (7/18), considerando correspondência explícita entre os artefatos gerados e os *challenges* avaliados.

A análise por microserviço no *crAPI* apresentou comportamento progressivo ao longo do pipeline. O M1 apresentou cobertura útil de 10,71% (3/28), indicando menor capacidade inicial de recuperação de vulnerabilidades relacionadas ao benchmark em um contexto predominantemente orientado a APIs. O M2 apresentou aumento para 24,00% (12/50), sugerindo que a representação das ameaças em formato de *evil user stories* favoreceu parcialmente a recuperação de

cenários abusivos semanticamente próximos aos desafios avaliados. O melhor desempenho foi observado no M3, com cobertura útil de 46,43% (13/28), indicando maior proximidade entre os cenários conceituais de testes gerados e as vulnerabilidades efetivamente presentes na aplicação.

Os resultados observados, apresentados em resumo na Tabela 2, evidenciam comportamentos distintos entre as métricas de cobertura direta e cobertura útil. Enquanto a cobertura direta mede a recuperação explícita das vulnerabilidades representadas pelos challenges dos benchmarks, a cobertura útil busca capturar a capacidade da arquitetura em recuperar fraquezas semanticamente relacionadas por meio das CWEs associadas. Nesse sentido, embora a cobertura direta tenha sido limitada, especialmente no *OWASP Juice Shop*, a cobertura útil permaneceu significativamente superior, indicando que a abordagem foi capaz de identificar mecanismos de segurança potencialmente relevantes mesmo quando não havia correspondência exata com o desafio avaliado. Além disso, em ambas as aplicações observou-se um padrão no qual os cenários conceituais de testes (M3) tenderam a apresentar maior proximidade com as vulnerabilidades presentes nos benchmarks.

Diferentemente de abordagens como *ThreatLens* [16] e *ThreatFinderAI* [21], que concentram-se em atividades específicas de Threat Modeling, os resultados sugerem que a geração progressiva de múltiplos artefatos permite preservar parte do contexto ao longo do fluxo, ainda que com perdas intermediárias observadas em M2.

As diferenças observadas entre os benchmarks podem estar relacionadas às características das aplicações avaliadas. O *OWASP Juice Shop* reúne um conjunto amplo e heterogêneo de desafios, muitos dos quais dependem de funcionalidades que não foram contempladas pelos requisitos funcionais utilizados como entrada da arquitetura. Em contraste, o *OWASP crAPI* possui um conjunto mais restrito de cenários, favorecendo maior correspondência entre os requisitos analisados e as vulnerabilidades avaliadas. Devido a isso, entende-se que enquanto o *Juice Shop* apresentou maior recuperação de vulnerabilidades no estágio inicial (M1), o *crAPI* demonstrou crescimento progressivo ao longo do pipeline, culminando no melhor desempenho no estágio de testes. Além disso, esses resultados indicam que a cobertura obtida depende não apenas da arquitetura proposta, mas também da abrangência e representatividade dos artefatos utilizados como entrada.

É importante ressaltar que a arquitetura proposta não busca substituir especialistas em segurança nem eliminar a necessidade de validação humana. Seu objetivo consiste em estruturar e sistematizar a elicitação inicial de artefatos de segurança, reduzindo omissões e oferecendo apoio a equipes com baixa proficiência em segurança. Dessa forma, mesmo que os artefatos produzidos demandem revisão posterior, espera-se favorecer a incorporação de práticas de segurança em contextos onde tais atividades frequentemente não são realizadas ou são executadas de forma limitada.

Como toda abordagem baseada em LLM, entretanto, ainda existe a possibilidade de respostas inconsistentes ou não fundamentadas. A utilização de RAG, bases consolidadas de conhecimento em segurança, respostas estruturadas em formato JSON e mecanismos compensatórios de geração busca reduzir esse risco e aumentar a rastreabilidade dos artefatos produzidos, mas não o elimina completamente. Assim, os resultados gerados pela arquitetura devem ser compreendidos como artefatos de apoio ao processo de engenharia de segurança, passíveis de revisão e refinamento pelos envolvidos no desenvolvimento.

Tabela 2: Resumo da avaliação das aplicações utilizadas

Aplicação	Desafios com CWEs	C. Direta	C. Útil
OWASP Juice Shop	18	11,61%	58,82%
OWASP crAPI	15	38,89%	40,00%

5 Conclusão

Este trabalho investigou o uso de LLMs como mecanismo de apoio, um copiloto, à geração progressiva de artefatos de segurança no contexto do SSDLC. A proposta explorou uma arquitetura encadeada orientada por STRIDE, combinando estratégias de RAG, *few-shot learning* e um mecanismo automático de compensação apoiado por bases especializadas de conhecimento em segurança.

Os resultados preliminares obtidos a partir dos benchmarks *OWASP Juice Shop* e *OWASP crAPI* indicam potencial da abordagem para apoiar a geração de requisitos de segurança, *evil user stories* e cenários conceituais de testes a partir de requisitos funcionais e descrições de cenários vulneráveis. Embora a cobertura observada ainda apresente limitações, os resultados sugerem capacidade da arquitetura em recuperar ameaças relevantes, produzir artefatos potencialmente úteis, preservando parcialmente a rastreabilidade entre requisitos de segurança, evil user stories e cenários conceituais de testes.

A principal contribuição desta pesquisa reside na investigação de uma abordagem multiartefato baseada em LLMs para apoio ao SSDLC, diferenciando-se de trabalhos anteriores que se concentram predominantemente em tarefas isoladas, como modelagem de ameaças ou geração de requisitos de segurança [15, 16, 21]. Além disso, o trabalho reforça uma perspectiva de uso de LLMs como suporte cognitivo a especialistas e equipes com baixa maturidade em segurança, buscando ampliar a escala de aplicação de práticas seguras sem substituir a tomada de decisão humana.

Como trabalhos futuros, pretende-se ampliar a avaliação utilizando outros sistemas, refinar o mecanismo automático de compensação, investigar critérios mais objetivos para mensuração de cobertura útil e conduzir estudos envolvendo especialistas em segurança e equipes de desenvolvimento, buscando compreender o impacto da abordagem na autonomia, esforço cognitivo e qualidade dos artefatos produzidos.

Disponibilidade de Artefatos

Os artefatos utilizados neste estudo, incluindo fluxos de *prompts*, conjuntos curados de requisitos funcionais, mapeamentos entre *challenge* e *CWE*, scripts de avaliação e resultados experimentais, estão disponíveis no seguinte repositório: https://anonymous.4open.science/r/virtual-security-copilot-article/

Referências

[1] Martin Bedoya, Sara Palacios, Daniel Díaz-López, Estefania Laverde, and Pantaleone Nespoli. 2024. Enhancing DevSecOps practice with Large Language Models and Security Chaos Engineering. *International Journal of Information Security* 23, 6 (Dec. 2024), 3765–3788. doi:10.1007/s10207-024-00909-w

[2] Maya Daneva and Chong Wang. 2018. Security Requirements Engineering in the Agile Era: How Does it Work in Practice?. In *2018 IEEE 1st International Workshop on Quality Requirements in Agile Projects (QuaRAP)*. IEEE, Banff, AB, 10–13. doi:10.1109/QuaRAP.2018.00008

[3] Michael Howard and Steve Lipner. 2006. *The Security Development Lifecycle*. Microsoft Press.

[4] ISO/IEC. 2011. *Information technology – Security techniques – Application security – Part 1: Overview and concepts*. Technical Report ISO/IEC 27034-1:2011. International Organization for Standardization and International Electrotechnical Commission, Geneva, CH. https://www.iso.org/standard/44378.html

[5] ISO/IEC. 2018. *Information technology – Security techniques – Information security management systems – Overview and vocabulary*. Technical Report ISO/IEC 27000:2018. International Organization for Standardization and International Electrotechnical Commission, Geneva, CH. https://www.iso.org/standard/73906.html

[6] Samer Khamaiseh and Dianxiang Xu. 2017. Software Security Testing via Misuse Case Modeling. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. IEEE, Orlando, FL, 534–541. doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2017.98

[7] Rafiq Ahmad Khan, Siffat Ullah Khan, Habib Ullah Khan, and Muhammad Ilyas. 2022. Systematic Literature Review on Security Risks and its Practices in Secure Software Development. *IEEE Access* 10 (2022), 5456–5481. doi:10.1109/ACCESS.2022.3140181

[8] G. McGraw. 2006. *Software Security: Building Security in*. Addison-Wesley. https://books.google.com.br/books?id=1oz6swEACAAJ

[9] Seyedeh Leili Mirtaheri, Narges Movahed, Reza Shahbazian, Valerio Pascucci, and Andrea Pugliese. 2025. Cybersecurity in the age of generative AI: A systematic taxonomy of AI-powered vulnerability assessment and risk management. *Future Generation Computer Systems* 175 (Feb. 2025), 108107. doi:10.1016/j.future.2025.108107

[10] Demircioğu Murat, Ufuk Berkan, and Işikli Ali. 2024. An Overview of Secure by Design: Enhancing Systems Security through Systems Security Engineering and Threat Modeling. In *2024 17th International Conference on Information Security and Cryptology (ISCTürkiye)*. IEEE, Ankara, Turkiye, 1–6. doi:10.1109/ISCTrkiye64784.2024.10779293

[11] Suvda Myagmar, Adam J Lee, and William Yurcik. 2005. Threat Modeling as a Basis for Security Requirements. (2005).

[12] S Nagasundari, Pratham Manja, Priyansh Mathur, and Prasad B. Honnavalli. 2025. Extensive Review of Threat Models for DevSecOps. *IEEE Access* 13 (2025), 45252–45271. doi:10.1109/ACCESS.2025.3547932

[13] OWASP Foundation. 2024. OWASP Application Security Verification Standard (ASVS) Project. https://owasp.org/www-project-application-security-verification-standard/.

[14] Bernhard Peischl, Michael Felderer, and Armin Beer. 2016. Testing Security Requirements with Non-experts: Approaches and Empirical Investigations. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Vienna, Austria, 254–261. doi:10.1109/QRS.2016.37

[15] Giuseppe Sabetta, Alfonso Cannavale, Andrea De Lucia, and Fabio Palomba. 2026. Enhancing Security Requirements Coverage via RAG and Automated Feedback Loops. In *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*. IEEE, Limassol, Cyprus, 285–292. doi:10.1109/SANER-C67878.2026.00045

[16] Dipayan Saha, Hasan Al Shaikh, Shams Tarek, and Farimah Farahmandi. 2025. Special Session: ThreatLens: LLM-guided Threat Modeling and Test Plan Generation for Hardware Security Verification. In *2025 IEEE 43rd VLSI Test Symposium (VTS)*. IEEE, Tempe, AZ, USA, 1–5. doi:10.1109/VTS65138.2025.11022932

[17] Adam Shostack. 2014. *Threat modeling: designing for security*. John Wiley and Sons, Indianapolis, IN. OCLC: 874161580.

[18] I. Tarandach and M.J. Coles. 2020. *Threat Modeling: A Practical Guide for Development Teams*. O'Reilly Media, Incorporated.

[19] Arpit Thool and Chris Brown. 2024. Securing Agile: Assessing the Impact of Security Activities on Agile Development. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24)*. Association for Computing Machinery, New York, NY, USA, 668–678. doi:10.1145/3661167.3661280

[20] Yolanda Valdés-Rodríguez, Jorge Hochstetter-Diez, Mauricio Diéguez-Rebolledo, Ana Bustamante-Mora, and Rodrigo Cadena-Martínez. 2024. Analysis of Strategies for the Integration of Security Practices in Agile Software Development: A Sustainable SME Approach. *IEEE Access* 12 (2024), 35204–35230. doi:10.1109/ACCESS.2024.3372385 Conference Name: IEEE Access.

[21] Jan Von Der Assen, Alberto Huertas, Jamo Sharif, Chao Feng, Gérôme Bovet, and Burkhard Stiller. 2024. ThreatFinderAI: Automated Threat Modeling Applied to LLM System Integration. In *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, Prague, Czech Republic, 1–3. doi:10.23919/CNSM62983.2024.10814632