UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

JOYCE QUINTINO ALVES

PERCI: PROCESSO DE VERIFICAÇÃO DE CONTRATOS INTELIGENTES PARA APLICAÇÕES IOT

FORTALEZA

2024

JOYCE QUINTINO ALVES

PERCI: PROCESSO DE VERIFICAÇÃO DE CONTRATOS INTELIGENTES PARA APLICAÇÕES IOT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação.

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientadora: Profa. Dra. Carina Teixeira de Oliveira.

FORTALEZA

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A479p Alves, Joyce Quintino.
PERCI : Processo de Verificação de Contratos Inteligentes para Aplicações IoT / Joyce Quintino Alves. – 2024.
81 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2024.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Profa. Dra. Carina Teixeira de Oliveira.

1. Internet das coisas. 2. Segurança. 3. Blockchain. 4. Contratos inteligentes. I. Título.

CDD 005

JOYCE QUINTINO ALVES

PERCI: PROCESSO DE VERIFICAÇÃO DE CONTRATOS INTELIGENTES PARA APLICAÇÕES IOT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação.

Aprovada em: 30 de Agosto de 2024.

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade (Orientadora)
Universidade Federal do Ceará (UFC)

Profa. Dra. Carina Teixeira de Oliveira (Coorientadora)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dr. Fernando Parente Garcia
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dr. Emanuel Bezerra Rodrigues
Universidade Federal do Ceará (UFC)

Dedico este trabalho aos meus pais, Maura Quintino Jácome e Evando Alves Maia, por acreditarem em mim incondicionalmente e por todo o apoio e amor que sempre me deram.

AGRADECIMENTOS

Aos meus pais, Maura Quintino Jácome e Evando Alves Maia, pelo amor, apoio incondicional e por sempre acreditarem em mim.

Às minhas orientadoras, Profa. Dra. Rossana Maria de Castro Andrade e Profa. Dra. Carina Teixeira de Oliveira, pela orientação e pelo apoio que foram fundamentais para a realização deste trabalho e para o meu crescimento pessoal e profissional.

À Associação Brasileira de Apoio e Desenvolvimento da Arte-Capoeira (Abadá-Capoeira), em especial a José Tadeu Carneiro Cardoso (Mestre Camisa), ao Prof. Dr. Andreyson Calixto de Brito (Prof. Berimbau) e ao aluno César Augusto Gomes Raulino, pelos valiosos ensinamentos e momentos compartilhados, que enriqueceram minha jornada.

Aos professores, Prof. Dr. Mauro Oliveira, Profa. Dra. Raquel Silveira e Prof. Dr. Reinaldo Braga, pelos conselhos, incentivo e por sempre acreditarem em mim.

Aos professores participantes da banca examinadora, Prof. Dr. Fernando Parente Garcia e Prof. Dr. Emanuel Bezerra Rodrigues, pelo tempo, pelas valiosas colaborações e sugestões.

Aos meus amigos, Lailson Azevedo, Fabrício Ferreira, Kamila Lima, Isabely Costa, Belmondo Rodrigues e Alex Felipe, pela amizade, apoio e pelos momentos de descontração que tornaram esta caminhada mais leve.

Ao Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat), por todo apoio.

Aos alunos Sérgio Garcia, Carolina Máximo e Dafne Cavalcante pelas contribuições com o desenvolvimento da aplicação ClimaCor, nas disciplinas de Introdução à Computação Móvel e Ubíqua (CK0264) e Tópicos Avançados em Sistemas Distribuídos (CPK7600), ministradas pela professora Rossana Andrade.

À Universidade Federal do Ceará, pela excelência acadêmica e pelas oportunidades de crescimento profissional que contribuíram para a realização deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

"They will not force us

They will stop degrading us

They will not control us

We will be victorious"

(BELLAMY, Matthew. *Uprising*. *In*: *Muse*. ***The Resistance***. Lago do Como, Itália: Helium 3, Warner Bros, 2009. CD single. Faixa 1.)

RESUMO

Com o crescimento dos dispositivos na Internet das Coisas (IoT), a quantidade de dados trafegando pela rede aumenta e um ambiente confiável torna-se essencial para evitar vulnerabilidades de segurança. Nesse cenário, a *Blockchain* surge como uma tecnologia promissora para aprimorar a segurança na IoT, pois possibilita o registro de dados de forma descentralizada, criptografada e imutável, com o consenso dos participantes da rede. Os contratos inteligentes são programas autoexecutáveis distribuídos em uma *Blockchain*. Nas aplicações de IoT que utilizam *Blockchain*, os contratos inteligentes podem eliminar a necessidade de intermediários, permitindo transferências de dados mais seguras e transparentes entre as partes envolvidas, de forma descentralizada. Contudo, os contratos inteligentes estão sujeitos a falhas de segurança, causadas principalmente por erros de programação e vulnerabilidades presentes no código-fonte podem resultar, por exemplo, em perdas financeiras ou comprometer a integridade dos dados, causando riscos à privacidade e à segurança dos usuários. Diante disso, realizar testes com diferentes abordagens antes da implantação pode expor erros no código do contrato inteligente e reduzir os riscos de segurança. Este trabalho propõe então um processo, denominado PERCI, que define um conjunto de etapas de verificação de contratos inteligentes para aplicações IoT, a fim de detectar vulnerabilidades conhecidas usando a combinação de ferramentas de análises estática e dinâmica, antes da implantação do contrato inteligente. A combinação de análises estática e dinâmica é proposta para melhorar a detecção de vulnerabilidades, proporcionando uma solução mais robusta. Para isso, utiliza-se no processo duas ferramentas de análise estática, *Slither* e *Mythril*, e uma ferramenta de análise dinâmica, *Manticore*. Para avaliar o PERCI, primeiro, demonstrou-se que a combinação das análises de cada ferramenta teve uma detecção mais eficiente de vulnerabilidades, proporcionando uma verificação mais abrangente e precisa do código dos contratos inteligentes. Além disso, este trabalho integrou um contrato inteligente para registrar e autenticar dispositivos na *Blockchain* a uma aplicação IoT que representa condições meteorológicas por meio de cores com uma lâmpada inteligente. A avaliação do processo demonstrou a viabilidade do uso combinado de análises estática e dinâmica na detecção mais eficiente de vulnerabilidades. Por fim, espera-se que esta dissertação contribua para a melhoria da segurança e confiabilidade das aplicações IoT que utilizam *Blockchain*.

Palavras-chave: internet das coisas; segurança; *blockchain*; contratos inteligentes.

ABSTRACT

The growth of devices in the Internet of Things (IoT) has brought an increase in the amount of data flowing through the network. As a consequence of that, a reliable environment has became essential to avoid security vulnerabilities. In this scenario, Blockchain emerges as a promising technology to enhance IoT security, enabling decentralized, encrypted, and immutable data registration with the consensus of network participants. Smart contracts are self-executing programs distributed in a Blockchain. In IoT applications that use Blockchain, smart contracts can eliminate the need for intermediaries, allowing for more secure and transparent data transfers between involved parties in a decentralized manner. However, smart contracts are subject to security flaws, mainly caused by programming errors and vulnerabilities in the source code, which can result in financial losses or compromise data integrity, posing risks to users' privacy and security. Therefore, performing tests with different approaches before deployment can expose errors in the smart contract code and reduce security risks. This work then proposes a process, called PERCI, that defines a set of verification steps for smart contracts in IoT applications to detect known vulnerabilities, using a combination of static and dynamic analysis tools before the contract deployment. The combination of static and dynamic analyses is proposed to improve vulnerability detection, providing a more robust solution. For this, the process uses two static analysis tools, Slither and Mythril, and one dynamic analysis tool, Manticore. PERCI is evaluated, firstly, by demosntrating that the combination of the analyses of each tool resulted in more efficient vulnerability detection, providing a more comprehensive and precise verification of the smart contract code. Additionally, this work integrated a smart contract to register and authenticate devices on the Blockchain with an IoT application that shows weather conditions through colors with a smart lamp. The process evaluation demonstrated the feasibility of using combined static and dynamic analyses for more efficient vulnerability detection. Finally, this dissertation is expected to contribute to improving the security and the reliability of IoT applications that use Blockchain.

Keywords: internet of things; security; blockchain; smart contracts.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AST	Árvore de Sintaxe Abstrata
CID	Confidencialidade, Integridade e Disponibilidade
DAO	*Decentralized Autonomous Organization*
DApps	*Decentralized Applications*
DoS	*Denial of Service*
ETH	*Ether*
EVM	*Ethereum Virtual Machine*
GB	*Gigabytes*
ID	Identificador do experimento
IoT	*Internet of Things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LTS	*Long-Term Support*
NFC	*Near Field Communication*
NFTs	Non-fungible Tokens
OWL	*Web Ontology Language*
PERCI	Processo de Verificação de Contratos Inteligentes para Aplicações IoT
RAM	*Random Access Memory*
RDF	*Resource Description Framework*
RFID	*Radio Frequency Identification*
SAST	*Static Application Security Testing*
SCs	*Smart Contracts*
SI	Segurança da Informação

SUMÁRIO

1 INTRODUÇÃO

Esta dissertação de mestrado tem o objetivo de propor o PERCI, um Processo de vERificação de Contratos Inteligentes para aplicações da Internet das Coisas, a fim de detectar vulnerabilidades conhecidas usando a combinação de ferramentas de análise estática juntamente com análise dinâmica, antes da implantação do contrato inteligente na *Blockchain*.

Neste capítulo é feita uma introdução ao trabalho desenvolvido no mestrado. A Seção 1.1 trata do contexto no qual a presente dissertação está inserida. A Seção 1.2 apresenta a motivação deste trabalho. A Seção 1.3 descreve o objetivo e a metodologia utilizada para desenvolver esta dissertação. Por fim, a Seção 1.4 apresenta a estrutura da dissertação.

1.1 Contextualização

A Internet das Coisas (do inglês, *Internet of Things* (IoT)) é uma rede composta por objetos inteligentes, que são dispositivos embutidos conectados à Internet, capazes de interagir uns com os outros para fornecer serviço (ATZORI *et al.*, 2010) (MINERVA *et al.*, 2015). Ela conecta diferentes dispositivos do nosso cotidiano, como geladeiras, lâmpadas e carros autônomos, promovendo a interconexão de bilhões de dispositivos. De acordo com (MERLINO; ALLEGRA, 2024), estima-se que o número de dispositivos conectados chegará a aproximadamente 29 bilhões até 2030. Com o aumento da quantidade de dispositivos conectados, o volume de dados trafegando pela rede também aumenta, tornando essencial o desenvolvimento de um ambiente IoT seguro para a proteção dos dados dos usuários (INUWA; DAS, 2024).

No entanto, a IoT enfrenta desafios significativos em termos de segurança. Nesse caso, as arquiteturas IoT tradicionais usam um esquema centralizado para interconexão dos dispositivos, deixando-os mais vulneráveis a ameaças, o que compromete a integridade, a privacidade e a confiabilidade dos dados (DEEP *et al.*, 2024). Além disso, limitações como poder computacional e armazenamento também desafiam a segurança na IoT, pois as soluções de segurança existentes para a Internet tradicional podem ser ineficazes nesse contexto. Desta forma, esses dispositivos estão sujeitos a várias ameaças de segurança, tanto de vulnerabilidades genéricas quanto específicas da tecnologia (WAHEED *et al.*, 2020).

A autenticação é um dos requisitos para garantir a confidencialidade, integridade e privacidade dos dados (DEEP *et al.*, 2024), desempenhando um papel fundamental na restrição de acessos a recursos e dados apenas para dispositivos conhecidos e confiáveis (CARDOSO

et al., 2022). No entanto, os métodos de autenticação existentes podem não ser adequados para dispositivos IoT com recursos limitados, devido às suas capacidades computacionais e de armazenamento restritas (ALAJLAN *et al.*, 2023).

Para proporcionar segurança na IoT, *Blockchain* surge como uma tecnologia promissora (HE *et al.*, 2023). A *Blockchain* é uma tecnologia que surgiu a partir da criação do *Bitcoin* por Satoshi Nakamoto em 2008 (NAKAMOTO, 2008), com o intuito de promover a segurança entre troca de ativos financeiros por meio de nós não confiáveis na rede, sem necessidade de uma entidade centralizada. Ela também possui recursos que permitem o registro de dados de forma descentralizada, criptografada, imutável com o consenso dos participantes da rede (BHUTTA *et al.*, 2021). Com isso, a *Blockchain* garante a integridade dos dados e evita a manipulação ou o acesso não autorizado, tornando-se um ambiente confiável (ABIJAUDE *et al.*, 2021).

Nas aplicações de IoT que utilizam *Blockchain*, os Contratos Inteligentes (do inglês, *Smart Contracts* (SCs)) podem ser usados na construção de sistemas de automação e controle dos dispositivos IoT (JÚNIOR; AUGUSTO, 2023). Os SCs são programas com regras definidas, autônomos e autoexecutáveis, capazes de eliminar a necessidade de intermediários e são distribuídos por vários nós em uma *Blockchain*. Isso permite transferências de dados mais seguras e transparentes entre as partes envolvidas de forma descentralizada (ABIJAUDE *et al.*, 2021). Esses SCs também podem ser usados para implementar a lógica de autenticação em aplicações de IoT usando *Blockchain*, possibilitando a execução automática de regras de autenticação sem a necessidade de intermediários (LONE; NAAZ, 2021). O uso dos SCs pode melhorar a confiabilidade e a escalabilidade das aplicações IoT ao estabelecer confiança para dados e processos executados (REJEB *et al.*, 2024).

Outra característica dos SCs é a imutabilidade, ou seja, o código do contrato não pode ser alterado quando implantado na *Blockchain*. Isso torna o processo de testes um desafio significativo, pois qualquer erro ou vulnerabilidade identificada após a implantação não pode ser corrigido diretamente no contrato existente (ZHUANG *et al.*, 2020).

1.2 Motivação

Diversos estudos apontam que, nos últimos anos, diversos ataques foram direcionados a contratos inteligentes na *Blockchain* (GHOSH *et al.*, 2020) (BEGUM *et al.*, 2020) (HE *et al.*, 2023). Em 2010, um atacante explorou uma vulnerabilidade de *Integer Overflow*[1] para

[1] Vulnerabilidade causada por esgotamento da capacidade de armazenamento de uma variável inteira.

criar 184.467 milhões de *bitcoins* em um bloco específico da *Blockchain*. Após cinco horas da descoberta, foi lançada a versão 0.3.1 do *Bitcoin*, corrigindo rapidamente a vulnerabilidade. Em 2012, os servidores da *Bitfloor* e *Bitcoinica*, pertencentes à quarta maior bolsa de dólares do mundo, foram invadidos, resultando no roubo de 12.554 *bitcoins* (BEGUM *et al.*, 2020). Já em 2014, a *Mt. Gox*, que na época era a maior bolsa de *bitcoin* do mundo, anunciou o roubo de 850.000 *bitcoins*, causando um prejuízo estimado em 450 milhões de dólares americanos (HE *et al.*, 2023). Em 2020, a rede do *Poly Network* também foi alvo de um ataque. Em apenas 34 minutos, um *hacker* roubou 302 milhões de dólares, 55.000 *Ethers* e 2.000 *bitcoins*, totalizando um valor de 610 milhões de dólares americanos (GHOSH *et al.*, 2020).

Diante desses relatos, é perceptível que os SCs estão sujeitos a uma variedade de vulnerabilidades, ocasionadas por erros de codificação, falhas na especificação dos requisitos e lacunas de segurança, que podem ser exploradas e comprometer a integridade do sistema (HE *et al.*, 2023). Para garantir a segurança dos Contratos Inteligentes em um sistema, é essencial aplicar boas práticas de desenvolvimento seguro, realizar auditorias do código e avaliar as vulnerabilidades (HE *et al.*, 2023). Realizar testes abrangentes usando diferentes análises antes da implantação pode expor erros no código do contrato inteligente e reduzir os riscos de segurança. Além disso, a detecção de potenciais vulnerabilidades e falhas antes da implantação do SC reduz a necessidade de uma atualização do SC em produção na *Blockchain* (PENG *et al.*, 2021).

Nesse contexto, diversos esforços têm sido realizados pela comunidade acadêmica e indústria com o objetivo de desenvolver ferramentas de análise automatizadas para detectar e/ou eliminar vulnerabilidades em Contratos Inteligentes (KUSHWAHA *et al.*, 2022). Mais especificamente, essas ferramentas são desenvolvidas para executar uma série de testes automatizados a fim de explorar o código do SC em análise, gerando casos de testes na busca de identificar possíveis vulnerabilidades. Conforme será apresentado nos próximos capítulos dessa dissertação, na geração desses casos de testes, as ferramentas podem utilizar diferentes tipos de análises, como estática e dinâmica (KUSHWAHA *et al.*, 2022), para identificar possíveis falhas de segurança no código, auxiliando na identificação de erros antes da implantação, buscando garantir que o código funcione conforme o esperado (DURIEUX *et al.*, 2019).

Apesar das vantagens do uso de ferramentas automatizadas, desenvolver um contrato inteligente seguro não é trivial. Um relatório de 2018 indicou que, na plataforma *Ethereum*[2],

[2] https://ethereum.org/en/

89% das vulnerabilidades identificadas foram provenientes de erros de programação (NIKOLIC *et al.*, 2018). Outro estudo revelou que em quase um milhão de contratos inteligentes, da plataforma *Ethereum*, 34.200 deles foram classificados como vulneráveis e 89% das vulnerabilidades identificadas foram provenientes de erros de programação (DURIEUX *et al.*, 2019).

Sendo assim, a pesquisa conduzida por (DURIEUX *et al.*, 2019) destacou a falta de aprimoramento das ferramentas para lidar com uma variedade mais ampla de vulnerabilidades relacionadas aos Contratos Inteligentes. Embora essas ferramentas sejam capazes de identificar as vulnerabilidades em determinados contextos, elas também enfrentam limitações na detecção de vulnerabilidades mais complexas ou específicas, e podem gerar falsos positivos ou negativos, comprometendo a precisão da análise. Além do mais, utilizar um único tipo de análise pode deixar passar alguma falha de segurança, pois algumas ferramentas podem não identificar determinadas vulnerabilidades que são detectadas por outros tipos de análise (KUSHWAHA *et al.*, 2022). Sendo assim, uma abordagem sugerida para aprimorar a detecção de vulnerabilidades em Contratos Inteligentes é a utilização de uma solução que combine diferentes análises, como estática e dinâmica, buscando melhorar a precisão e reduzindo as chances de falhas de segurança passarem despercebidas (DURIEUX *et al.*, 2019) (KUSHWAHA *et al.*, 2022) (HE *et al.*, 2023) (KHAN; NAMIN, 2024).

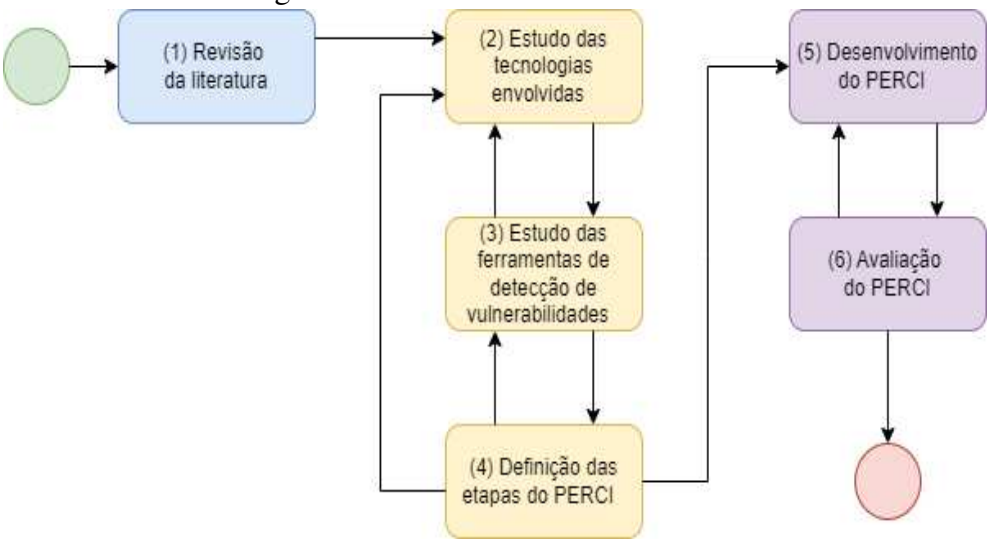
1.3 Objetivo e Metodologia

A fim de contribuir para a identificação de vulnerabilidades em Contratos Inteligentes, esta dissertação tem como objetivo apresentar o PERCI, um Processo de Verificação de Contratos Inteligentes para aplicações IoT. O PERCI possui um conjunto de etapas de verificação de Contratos Inteligentes com o intuito de detectar vulnerabilidades utilizando a combinação de ferramentas de análises estática e dinâmica. A combinação dessas análises é proposta para melhorar a detecção de vulnerabilidades, proporcionando uma solução mais robusta. Para isso, os Contratos Inteligentes são submetidos a ferramentas que seguem abordagens de análise estática e dinâmica em seus testes. Dessa forma, esse processo busca contribuir para a identificação de vulnerabilidades em Contratos Inteligentes antes da fase de implantação na *Blockchain*.

Para alcançar o objetivo desta pesquisa, seguiu-se uma metodologia composta de seis fases ilustrada na Figura 1.

Inicialmente, na FASE (1) foi realizada uma revisão da literatura na área de *Blockchain* com ênfase em contratos inteligentes, assim como abordando a temática de IoT combinada.

Figura 1 – Fases da metodologia.



Fonte: elaborada pelo autor.

Essa revisão possibilitou ter uma visão geral da área e a identificação de pesquisas na mesma temática. Após a realização da revisão da literatura, identificou-se temas relevantes para melhorar a segurança, principalmente da Internet das Coisas utilizando *Blockchain*, o estado atual da detecção das vulnerabilidades em Contratos Inteligentes, e quais ferramentas são usadas.

Na FASE (2) houve o estudo das tecnologias envolvidas para construção desta dissertação. Esse estudo consistiu na apropriação do conhecimento sobre tecnologias como a plataforma *Ethereum*, a linguagem de programação *Solidity* , as ferramentas de desenvolvimento *Truffle* e *Remix*, as ferramentas de análise de Contratos Inteligentes, como *Mythril*, *Slither* e *Manticore*, e a biblioteca *Web3.js*, que permite a interação com a *Blockchain*.

Além dos estudos das tecnologias envolvidas, realizou-se especificamente na FASE (3) um estudo sobre as ferramentas de detecção de vulnerabilidades de Contratos Inteligentes. Esse estudo incluiu a leitura de artigos sobre as ferramentas para entender como funcionam, quais abordagens de teste utilizam, eficiência na detecção e tempo de execução. Alguns trabalhos utilizados durante este estudo relatam as principais características dessas ferramentas.

Em seguida, definiu-se na FASE (4) o Processo de Verificação de Contratos Inteligentes para Aplicações IoT (PERCI), que ao longo da pesquisa teve seus passos analisados e atualizados. Na construção do PERCI, buscou-se refinar todas as etapas para melhorar a precisão e a confiabilidade da detecção de todas as vulnerabilidades presentes no contrato. Para isso, foi necessário revisitar as Fases (2) e (3), conforme ilustram as setas na Figura 1.

Já na FASE (5), o PERCI foi desenvolvido e os detalhes da implementação das

etapas do PERCI é discutido no Capítulo 4. Inicialmente, verificou-se quais etapas do PERCI poderiam ser automatizadas para uma melhor utilização do processo.

Na avaliação do PERCI, na FASE (6), realizou-se testes com um contrato inteligente de controle de acesso para verificar a precisão da detecção das ferramentas *Slither*, *Mythril* e *Manticore*. Para os testes em um ambiente experimental, foi utilizada uma aplicação IoT chamada de ClimaCor, que tem como objetivo transformar a experiência do usuário na descoberta do clima integrando a aplicação a uma lâmpada inteligente, permitindo assim que esta seja usada como um indicador visual das condições meteorológicas atuais. A lâmpada muda de cor em tempo real, oferecendo uma representação visual e intuitiva do clima. As cores específicas são atribuídas a cada condição climática, como ensolarado, nublado, chuva, entre outras, proporcionando uma experiência imediata e compreensível para o usuário. Para avaliar o PERCI, um contrato inteligente adaptado da implementação de (ŠIMUNIĆ, 2018) foi integrado à aplicação ClimaCor para registrar e autenticar a lâmpada na *Blockchain*. No que diz respeito à avaliação final do processo, o contrato inteligente utilizado na aplicação ClimaCor foi submetido aos testes das ferramentas de análise estática, *Slither* e *Mythril*, e de análise dinâmica *Manticore*. A análise com essas ferramentas permitiu avaliar as etapas do PERCI e também verificar a segurança do contrato inteligente, identificando possíveis vulnerabilidades.

1.4 Estrutura da Dissertação

Neste capítulo, foi apresentada uma visão geral da pesquisa desenvolvida nesta dissertação. Foram abordados a contextualização, a motivação para a pesquisa, bem como o objetivo e a metodologia utilizada para alcançá-lo. Além deste **Capítulo 1**, a dissertação está organizada em mais capítulos, descritos a seguir.

No **Capítulo 2** são definidos os principais conceitos que fundamentam esta pesquisa, incluindo Internet das Coisas, Segurança da Informação, *Blockchain*, Análise Estática e Análise Dinâmica, e Contratos Inteligentes.

No **Capítulo 3** são apresentados trabalhos nas temáticas de análises em contratos inteligentes, experimentação com ferramentas de detecção de vulnerabilidades de contratos inteligentes e/ou análises de conjuntos de dados de contratos inteligentes.

Em seguida, o **Capítulo 4** apresenta uma visão geral do PERCI e detalha as etapas definidas para a sua utilização.

O **Capítulo 5** discorre sobre a avaliação do PERCI, abordando os experimentos

conduzidos usando as ferramentas para analisar um contrato inicial com a vulnerabilidade de Reentrância e também o contrato inteligente integrado na aplicação ClimaCor.

Por fim, no **Capítulo 6**, esta dissertação é concluída com o resumo de seus principais resultados e contribuições, limitações e os trabalhos futuros para a evolução do processo proposto.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos fundamentais para compreensão deste trabalho, que são: Internet das Coisas, Segurança da Informação, *Blockchain*, Contratos Inteligentes, Vulnerabilidades de Contratos Inteligentes, Análise Estática, Análise Dinâmica e as ferramentas de detecção de vulnerabilidades.

O capítulo está organizado da seguinte forma: A Seção 2.1 descreve os conceitos principais para o entendimento da Internet das Coisas; A Seção 2.2 apresenta os conceitos de Segurança da Informação; Já a Seção 2.3 apresenta os conceitos sobre *Blockchain*; A Seção 2.4 discorre sobre Contratos Inteligentes; A Seção 2.5 define as Vulnerabilidades de Contratos Inteligentes; A Seção 2.6 descreve os conceitos de Análise Estática e Análise Dinâmica; Por fim, a Seção 2.7 apresenta as Ferramentas de Detecção de Vulnerabilidades.

2.1 Internet das Coisas

A Internet das Coisas ou IoT é um paradigma usado para denominar uma variedade de objetos inteligentes que são capazes de interagir uns com os outros para fornecer serviço (ATZORI *et al.*, 2010). Segundo o IEEE (MINERVA *et al.*, 2015), IoT é uma rede de objetos ligados à Internet. A IoT conecta diferentes dispositivos do nosso cotidiano, como geladeiras, lâmpadas, telefones celulares, sensores, atuadores, entre outros, para coletar e transmitir dados para a rede (ATZORI *et al.*, 2010).

Os dados gerados pelos dispositivos IoT podem informar o estado e a capacidade de trabalho desses dispositivos, além de serem utilizados em benefícios dos usuários, por exemplo, na conexão de pacientes e médicos. Ademais, a IoT pode possibilitar uma agricultura mais eficiente, transporte inteligente, desenvolvimento de casas inteligentes e, consequentemente, proporcionar mais qualidade de vida (FENG, 2019). Para compreender o paradigma da Internet das Coisas, são apresentados a seguir outros conceitos importantes, como os principais elementos dessa tecnologia na Subseção 2.1.1 e os requisitos para as aplicações IoT na Subseção 2.1.2.

2.1.1 Principais elementos da IoT

De acordo com (Al-Fuqaha *et al.*, 2015), a IoT possui seis elementos principais para um bom funcionamento das aplicações: identificação, sensoriamento, comunicação, computação, serviços e semântica. A Figura 2 apresenta uma visão desses elementos principais que compõem

a IoT e cada um deles é definido a seguir.

Figura 2 – Principais elementos da IoT.

Fonte: adaptada de (Al-Fuqaha *et al.*, 2015).

- **Identificação:** É essencial que os objetos tenham uma identificação para facilitar a conexão com outros objetos. Essa identificação é definida através da descrição do serviço e do seu endereço na rede. Alguns métodos de identificação utilizados são *Radio Frequency Identification* (RFID), *Near Field Communication* (NFC) e endereçamento *Internet Protocol* (IP).
- **Sensoriamento:** Trata da coleta dos dados dos objetos IoT conectados na rede e envio destes para um serviço de armazenamento, como banco de dados ou nuvem.
- **Comunicação:** Garante que objetos heterogêneos estejam conectados por meio de protocolos de comunicação utilizados. Alguns exemplos de protocolos são *Bluetooth*, Wi-fi e ZigBee. Tecnologias como RFID e NFC também são usadas para comunicação.
- **Computação:** Unidades de processamento e aplicações de software representam o "cérebro"na IoT. A computação é responsável pelo processamento dos dados coletados. Essa parte pode ser embarcada em microprocessadores e microcontroladores, entre outros. Existem também plataformas desenvolvidas para executar as aplicações IoT, como Arduino.
- **Serviços:** Os serviços podem ser divididos em quatro categorias: serviços de identificação, serviços de agregação, serviços colaborativos e serviços de ubiquidade.
  - Serviços de identificação: são os serviços mais básicos e utilizados por outros tipos de serviços. Esse tipo de serviço transforma os objetos físicos do mundo real em objetos virtuais. Nessa transformação é feita a identificação de cada objeto.
  - Serviços de agregação: trabalham na coleta e resumo de informações oriundos do

sensoriamento para que possam ser processadas e utilizadas pela aplicação IoT.
  - Serviços colaborativos: atuam nos serviços de agregação que utilizam os dados para tomar decisões e retorná-las para as aplicações.
  - Serviços de ubiquidade: possuem as mesmas características de serviços colaborativos, porém podem ser utilizados em qualquer lugar e por qualquer pessoa.
- **Semântica:** Representa a capacidade de extrair conhecimento de forma inteligente por diferentes máquinas e fornecer os serviços necessários. Essa extração de conhecimento refere-se a descoberta e utilização de recursos e informação. Algumas tecnologias que possibilitam isso são o *Resource Description Framework* (RDF) e a *Web Ontology Language* (OWL). Inclui também o reconhecimento e análise de dados para decisão correta.

2.1.2 Requisitos para aplicações IoT

O trabalho (PATEL *et al.*, 2016) lista e define os seguintes requisitos fundamentais para aplicações IoT:

- **Interconectividade:** diferentes objetos podem estar conectados dentro de uma infraestrutura de comunicação.
- **Serviços relacionados aos objetos IoT:** as aplicações IoT podem fornecer serviços como proteção da privacidade e consistência semântica aos objetos que fazem parte da solução.
- **Heterogeneidade:** IoT é composta por dispositivos heterogêneos com diferentes hardware e tecnologias. No entanto, esses dispositivos podem interagir com outros dispositivos ou plataformas de serviço por diferentes redes.
- **Dinamicidade:** o estado dos dispositivos muda de forma dinâmica, por exemplo, conectado ou desconectado, variação da localização, configurações alteradas, entre outros. Além disso, o número de dispositivos pode mudar de forma dinâmica também.
- **Escalabilidade:** considerando que o número de objetos que integram uma solução IoT pode variar, o sistema precisa continuar operacional independentemente do número de objetos conectados.
- **Conectividade:** permite o acesso à rede e a capacidade dos dispositivos consumirem e produzirem dados.
- **Segurança:** as aplicações IoT também precisam se preocupar com a segurança dos dados pessoais e bem-estar físico dos usuários. De acordo com (ATZORI *et al.*, 2010), as aplicações IoT são vulneráveis à diferentes ataques por algumas razões. Primeiro, pela

falta de supervisão dos dispositivos, e assim, torna-se fácil atacá-los fisicamente. Segundo, alguns dispositivos IoT possuem limitações como baixa capacidade de armazenamento, processamento e potência, impedindo o desenvolvimento de mecanismos robustos de segurança para proteção dos dados. Por isso, a segurança ainda é um desafio presente na IoT.

2.2 Segurança da Informação

A Segurança da Informação (SI) é uma área que tem como objetivo manter a proteção, privacidade e a segurança de um conjunto de dados. Ela está diretamente relacionada com proteção de um conjunto de informações, no sentido de preservar o valor que possuem para um indivíduo ou uma organização (STALLINGS, 2015) (ISO/IEC 27000, 2014).

Inicialmente, SI foi composta pelos seguintes princípios básicos: Confidencialidade, Integridade e Disponibilidade (CID). Ao longo dos anos, especialistas adicionaram outros dois princípios para fazer parte da definição da SI, são eles: Autenticidade e Irretratabilidade (STALLINGS, 2015).

Esses princípios da segurança são considerados requisitos fundamentais para o desenvolvimento de aplicações IoT. Na sequência, são apresentados as definições de cada princípio da segurança da informação.

- **Confidencialidade:** trata de evitar a revelação não autorizada de informação. Isto é, garantir que a informação estará acessível apenas para pessoas autorizadas. A criptografia (processo de codificação que faz o embaralhamento dos dados por meio de algoritmos criptográficos que fazem cálculos matemáticos para obter uma informação ilegível) é utilizada para manter a confidencialidade. Segundo (GOODRICH; TAMASSIA, 2013), obter confidencialidade é mais desafiador, pois os computadores estão em todos os lugares e cada um é capaz de executar operações que podem comprometer a confidencialidade.
- **Integridade:** tem o objetivo de proteger a informação contra alterações não autorizadas. Vale ressaltar que o princípio da integridade não garante que os dados não sejam alterados. A garantia é que, se os dados forem alterados sem autorização, a alteração será detectada.
- **Disponibilidade:** garantia que a informação esteja disponível para usuários autorizados. Por exemplo, os usuários legítimos de um sistema devem acessar as informações e recursos do sistema no momento em que desejar.
- **Autenticidade:** garantir de que o originador de uma mensagem seja corretamente identifi-

cado pelo seu destinatário. A verificação de autenticidade é necessária após todo processo de identificação, seja de um usuário para um sistema, de um sistema para um usuário ou de um sistema para outro sistema.

– **Irretratabilidade:** o emissor e o destinatário das informações não podem negar a sua transmissão, recepção ou autoria. Uma forma de garantir esse princípio é o uso de assinatura digital desde que siga um processo oficial de certificação digital.

Em relação aos desafios de segurança, segundo (YAO *et al.*, 2020), uma vez que IoT é composta por dispositivos heterogêneos e os protocolos não são padronizados, os dispositivos IoT estão sujeitos a ameaças específicas inerentes à tecnologia desses dispositivos.

De acordo com (MISHRA; PANDYA, 2021), os dispositivos IoT enfrentam também desafios de segurança relacionados à limitação de recursos, uma vez que existem restrições como custo, potência e tamanho. Por exemplo, os mecanismos tradicionais de segurança utilizados na Internet não são eficazes na IoT, dado que alguns dispositivos tem baixa capacidade de processamento. Isto consequentemente, torna a IoT vulnerável a ataques de segurança. Os recursos energéticos e a localização também são pontos de preocupação para implementação de métodos robustos de segurança na IoT.

2.3 *Blockchain*

A *Blockchain* é uma tecnologia de banco de dados distribuído que surgiu a partir da criação do *Bitcoin* por Satoshi Nakamoto em 2008 (NAKAMOTO, 2008). Essa tecnologia surge com o intuito de promover a segurança entre troca de ativos financeiros por meio de nós não confiáveis na rede, sem necessidade de uma entidade centralizada. Nakamoto introduziu um sistema confiável e imutável, servindo como uma espécie de livro-razão distribuído.

Essa tecnologia possui recursos que permitem o registro de dados de forma descentralizada, criptografada, imutável com o consenso dos participantes da rede usando uma estrutura de dados de pacote conhecida por bloco (BHUTTA *et al.*, 2021). Tais recursos são definidos a seguir:

– **Descentralização:** não tem uma entidade central coordenadora, ou grupo, com poderes para reverter dados, ou alterar regras de consenso.

– **Criptografia:** *Blockchain* possui criptografia de chaves pública e privada. A chave pública é comum a todos os participantes da rede, enquanto que a chave privada é exclusiva de cada membro da rede. Essas chaves juntas desbloqueiam os dados.

- **Imutabilidade:** não é permitido alterar dados depois de registrado na rede.
- **Consenso:** estabelece regras sobre o consentimento dos participantes da rede. O registro de dados somente é possível com o consentimento da maioria dos participantes.

A *Blockchain* também divide-se em três principais tipos: pública, privada e consórcio. Bhutta (BHUTTA *et al.*, 2021) define esses três tipos de *Blockchain*:

- **Pública:** não necessita de permissão para participar desse tipo de *Blockchain*. Todos os participantes têm direitos iguais para participar do processo de consenso, ler, editar e validar.
- **Privada:** diferentemente da pública, na *Blockchain* privada somente participantes autorizados podem fazer parte e manter a rede. Esse tipo de *Blockchain* é considerada mais segura que a pública. No entanto, os participantes não são anônimos.
- **Consórcio:** é também privada, mas destina-se a múltiplas organizações. Apenas participantes convidados e de confiança estão autorizados a participar e manter a rede.

Nos últimos anos, foram desenvolvidas plataformas de *Blockchain*, cada uma com características e funcionalidades específicas (HASAN, 2023). Algumas dessas plataformas são definidas a seguir:

- ***Ethereum*:** é uma plataforma desenvolvida em 2013 por Vitalik Buterin, que possibilita a criação de contratos inteligentes e os *Decentralized Applications* (DApps). Essa plataforma opera com sua própria moeda digital, chamada *Ether* (ETH), que a usa como meio de troca dentro da rede *Ethereum*. No contexto dessa plataforma, chama-se de *gás* a taxa paga na rede em troca do uso do poder computacional da plataforma.
- ***Hyperledger Fabric*:** é uma plataforma privada que possui uma arquitetura modular, contratos inteligentes, serviços de consenso e associação personalizáveis, desenvolvida pela IBM e Digital Asset. Pode suportar protocolos de consenso que não se baseiam em criptomoeda para incentivar mineração ou para promover a execução de contratos inteligentes.
- ***Corda*:** é uma plataforma, desenvolvida pela empresa R3, de contabilidade distribuída e contratos inteligentes voltada para negócios, sendo de código aberto.
- ***Quorum*:** é uma plataforma empresarial baseada no *Ethereum*, projetada para atender às necessidades específicas de empresas e consórcios. Desenvolvida pela JPMorgan Chase.

Uma *Blockchain* é formada por um conjunto de blocos em cadeia e faz parte de uma estrutura organizada como uma rede *peer-to-peer* - P2P, ou seja, não existe uma entidade

centralizadora na rede, são os próprios mineradores ou nós oferecendo o recurso computacional deles para o funcionamento da *Blockchain* (BHUTTA *et al.*, 2021).

As principais partes de um bloco são os registros e o cabeçalho. Os registros são armazenadas no bloco. O cabeçalho de um bloco é composto por alguns campos: *Hash* próprio do bloco, *Hash* do bloco anterior, o *Timestamp*, *Nonce*, Árvore de *merkle* (NAKAMOTO, 2008). Tais campos são definidos a seguir:

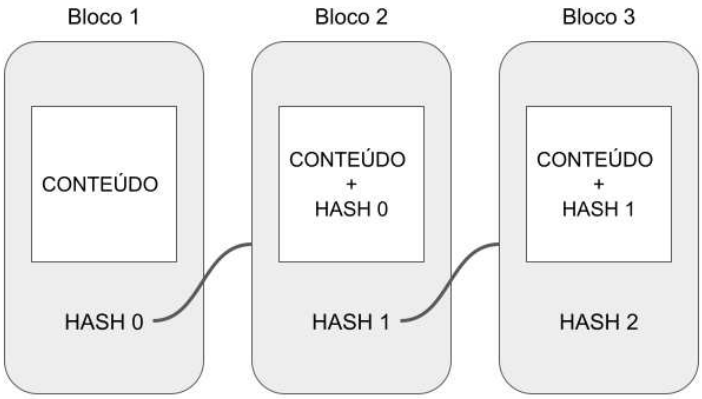
- ***Hash* próprio do bloco:** é o principal identificador do bloco gerado pelo processo de mineração.
- ***Hash* do bloco anterior:** possibilita a ligação com o bloco anterior, criando uma cadeira de blocos sequencial. Em casos de alterações, o bloco seguinte perde a referência e toda cadeia é inválida.
- ***Timestamp*:** tempo em que o bloco foi construído.
- ***Nonce*:** é um valor usado para alterar a saída da função *hash* do cabeçalho durante o processo de mineração.
- **Árvore de *merkle*:** totaliza todos registros de um bloco e gera uma impressão digital de todo o conjunto de operações, permitindo ao utilizador verificar um determinado registro no bloco.

A Figura 3 ilustra um conjunto de blocos em cadeia interligados pelo *hash*. Observa-se que o Bloco 1 não possui o *hash* do anterior, pois este bloco é o primeiro a ser criado e é chamada de *Genesis*. Os demais blocos a partir do *Genesis* possuem sempre o *hash* do bloco anterior (SINGH *et al.*, 2022).

Para registrar os dados no bloco e adicionar na *Blockchain* é necessário um processo de mineração. Para minerar um bloco, os mineradores precisam resolver um problema matemático. Após resolução do problema matemático por um dos mineradores, os demais mineradores precisam validar e entrar em consenso para permitir a adição do novo bloco na *Blockchain*. Depois da rede entrar em consenso, o bloco é adicionado na *Blockchain* e o minerador recebe a recompensa (a moeda digital no caso do *Bitcoin*) (BHUTTA *et al.*, 2021).

O processo de mineração na *Blockchain* é encontrar o *hash* válido para assinar o bloco. A única forma de encontrar o *hash* é por “força bruta", o que custa tempo e energia elétrica. O minerador não pode assinar o bloco com qualquer valor de *hash*, existe o problema matemático para resolver. Esse problema matemático significa encontrar o *hash* dentro de um determinado intervalo. Inicialmente, não se sabe qual *hash* será gerado, então, usa-se o *Nonce* e

Figura 3 – Conjunto de blocos em cadeia.



Fonte: adaptada de (SINGH *et al.*, 2022).

o *Timestamp* para encontrar o *hash* válido. O *Nonce* é alterado até o minerador encontrar o *hash* válido. Somente depois de solucionar o problema matemático, o bloco pode ser adicionado na *Blockchain* (SINGH *et al.*, 2022).

2.4 Contratos Inteligentes

Os Contratos Inteligentes (SCs) são programas autoexecutáveis com regras definidas capazes de controlar determinadas ações programadas sem a necessidade de intermediários, idealizados em 1990 por Nick Szabo (SZABO, 1997). No entanto, devido às limitações tecnológicas da época, os contratos não podiam ser desenvolvidos e implantados. Com o surgimento da tecnologia *Blockchain* em 2008, o desenvolvimento dos contratos ganhou impulso (HE *et al.*, 2023). Os contratos inteligentes são distribuídos por vários nós em uma *Blockchain* e não podem ser alterados devido a característica de imutabilidade da *Blockchain* (ALABA *et al.*, 2023).

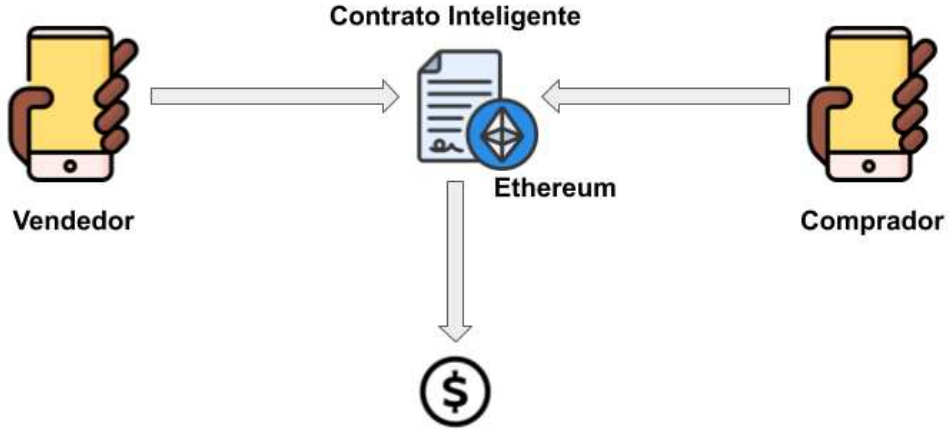
Por meio de regras definidas no código de um contrato inteligente, as cláusulas se executam a partir do momento em que as condições contratuais estabelecidas são atendidas. O principal objetivo dos SCs é permitir um meio mais seguro de trocar dados entre as partes envolvidas de forma descentralizada, sem a necessidade de intermediários ou confiança de terceiros (ABIJAUDE *et al.*, 2021). Aqui estão algumas características fundamentais dos contratos inteligentes:

- **Autoexecução:** os contratos inteligentes são autoexecutáveis, significando que são ativados automaticamente quando as condições predefinidas no código são atendidas.
- **Imutabilidade:** os contratos inteligentes quando implantados na *Blockchain*, o código não pode ser alterado.
- **Descentralização:** eles operam em uma rede descentralizada de nós na *Blockchain*, assim não há necessidade de uma autoridade central para intermediar ou validar as transações.
- **Tokenização:** os contratos inteligentes podem lidar com *tokens*, que representam ativos digitais ou valores. Isso é útil para implementar funcionalidades como a criação e transferência de ativos digitais de maneira segura.
- **Autenticidade e Integridade:** a imutabilidade da *Blockchain* garante a autenticidade e a integridade dos contratos inteligentes. Quando registrado na *Blockchain*, um contrato não pode ser adulterado, proporcionando confiança nas operações realizadas.

Tais contratos são comumente utilizados em ativos financeiros, como em setores de finanças descentralizadas e Non-fungible Tokens (NFTs) (ZHUANG *et al.*, 2020). Eles também podem ser usados para construir sistemas de automação e controle dos dispositivos IoT (ABIJAUDE *et al.*, 2021).

Um exemplo de contrato inteligente é mostrado na Figura 4. Neste exemplo, o contrato inteligente é responsável por definir as regras para uma compra online entre um vendedor e um comprador. Para isso acontecer, o vendedor e o comprador devem fazer parte de uma plataforma *Blockchain*, representada pela *Ethereum* no exemplo da Figura 4.

Figura 4 – Contrato Inteligente para compra online.



Fonte: elaborada pelo autor.

2.5 Vulnerabilidades de Contratos Inteligentes

Uma vulnerabilidade pode ser definida como uma fraqueza ou falha em um sistema, que pode abrir uma porta de entrada e ser explorada por uma ameaça para comprometer a segurança, integridade ou funcionalidade de um sistema. Dado isso, ao integrar contratos inteligentes com uma aplicação IoT podem surgir vulnerabilidades no código das aplicações IoT, comprometendo a segurança, integridade dos dados ou funcionalidade (QIAN *et al.*, 2022).

Esta Seção apresenta as vulnerabilidades comuns dos contratos inteligentes, incluindo a vulnerabilidade de Reentrância, Estouro de Número Inteiro, Negação de Serviço, *Delegatecall*, *tx.origin*, Chamada Não Verificada, *Self Destruct*, Controle de Acesso e Limite de Gás. Tais vulnerabilidades estão descritas a seguir, de acordo com as definições do trabalho de (LIAO *et al.*, 2019; GHALEB *et al.*, 2018; KUSHWAHA *et al.*, 2022; HE *et al.*, 2023):

- **Reentrância (*Reentrancy*):** contratos chamam outros contratos antes de terminar sua própria execução, permitindo que um contrato malicioso execute código em outro contrato, potencialmente alterando estados inesperadamente. De um modo geral, quando um atacante inicia uma operação de transferência, a função de retirada do contrato original chama a função do contrato externo, que aciona a função de retirada do contrato original. O atacante pode utilizar a função de recurso para chamar continuamente o método de levantamento do contrato original, efetuando uma operação de transferência contínua, deixando o contrato original sem saldo. Foi relatada pela primeira vez em 2016 a partir do famoso ataque *Decentralized Autonomous Organization* (DAO) (GHALEB *et al.*, 2018), que causou uma perda de 60 milhões de dólares americanos (LIAO *et al.*, 2019; KUSHWAHA *et al.*, 2022; HE *et al.*, 2023).
- **Estouro de Número Inteiro (*Integer Overflow*):** operações aritméticas podem resultar em números maiores que o máximo representável (overflow). Pode levar a cálculos errôneos ou manipulação de dados de sensores, resultando em ações inadequadas. Depois de o utilizador introduzir o conteúdo e executar o programa, se o resultado final do cálculo exceder o intervalo de armazenamento do tipo de dados, a entrada do contrato pode ser explorada pelo atacante (HE *et al.*, 2023).
- **Negação de Serviço (*Denial of Service* - DoS):** existem três ataques DoS gerais em contratos inteligentes (CHEN *et al.*, 2022). Primeiro, o atacante consome uma grande quantidade de ETH e gás, fazendo com que o contrato fique temporariamente ou permanentemente inoperante. Segundo, a conta do proprietário com o direito de abrir ou

suspender transações é perdida, resultando em um ataque *Denial of Service* (DoS). Em terceiro, como os estados dos contratos inteligentes dependem do resultado da execução da função externa, uma vez que a chamada externa falhe ou seja rejeitada, isso também levará a um ataque DoS (HE *et al.*, 2023).

- ***Delegatecall*:** quando a função de um contrato é acionada para o contrato atual para execução, será gerada uma perigosa *delegatecall()*, também conhecida como chamadas de delegado. O atacante pode usar a chamada delegada para chamar repetidamente a função de inicialização, modificando a si próprio como o proprietário do contrato inteligente (GUPTA; SHUKLA, 2019) (KUSHWAHA *et al.*, 2022).
- ***tx.origin*:** é uma variável global que representa o endereço do usuário externo que iniciou a transação. Essa variável pode ser utilizada para fins de autorização. No entanto, o uso de *tx.origin* pode ser perigoso para o controle de permissões, pois não é possível distinguir entre o usuário que está chamando o contrato inteligente diretamente e o que está fazendo a chamada indireta, através de outro contrato inteligente. Isso possibilita ataques em que um contrato inteligente malicioso pode induzir um usuário a fazer uma transação e, em seguida, redirecioná-la para outro contrato inteligente, enganando o controle de permissões (WOHRER; ZDUN, 2018) (KUSHWAHA *et al.*, 2022).
- **Chamada Não Verificada (*Unchecked Call*):** ocorre devido ao tratamento inadequado de excepções no código *Solidity*. Quando o valor de retorno da execução não é devidamente verificado e não são tomadas medidas adequadas (CHEN *et al.*, 2020) (KUSHWAHA *et al.*, 2022).
- ***Self Destruct*:** é um método que o proprietário utiliza para encerrar o seu contrato, apagar o seu código de bytes e e liberar o armazenamento. Ao utilizar esse tipo de método pode abrir portas para um atacante encerrar um contrato (LI *et al.*, 2020) (KUSHWAHA *et al.*, 2022).
- **Controle de Acesso (*Access Control*):** ocorre quando há autorização ou autenticação inadequada no contrato inteligente. Um atacante pode usar maliciosamente o mesmo para acessar as funções críticas, podendo ocorrer comportamentos inesperados ou inseguros (LI *et al.*, 2020) (KUSHWAHA *et al.*, 2022).
- **Limite de Gás (*Gas Limit*):** Podem existir vários problemas relacionados com o gás, como o envio de uma transação com gás insuficiente, ou até mesmo *loops* dispendiosos de gás presentes no contrato inteligente. O gás é utilizado como uma taxa para executar

as instruções do contrato inteligente na *Blockchain*, uma vez que cada tipo de operação requer um gás diferente, que é cobrado em ETH (KUSHWAHA *et al.*, 2022).

– **Dependência Data/Hora (Timestamp Dependency):** Ocorre quando o contrato inteligente se baseia no valor de data/hora do bloco que é gerado pelo nó que executa o contrato inteligente para executar uma operação. Isto torna-o vulnerável a ataques e suscetível de ser manipulado. Devido à natureza distribuída da *Blockchain*, é inevitável definir a hora exata entre os nós, sincronizá-la e recuperar a hora exata da *Blockchain* (KUSHWAHA *et al.*, 2022).

2.6 Análise Estática e Análise Dinâmica

Existem duas principais abordagens para testar os contratos inteligentes: análise estática e a análise dinâmica (ESQUIVEL *et al.*, 2023).

A análise estática analisa o código-fonte do contrato inteligente sem executá-lo. Esta técnica inspeciona o código em busca por possíveis erros, vulnerabilidades conhecidas e conformidade com padrões de codificação. Ferramentas que utilizam verificações estáticas aplicam regras predefinidas para identificar possíveis falhas. Esse tipo de análise é um teste da estrutura interna do código, em vez de um teste funcional (ESQUIVEL *et al.*, 2023).

Por outro lado, a análise dinâmica adota uma abordagem diferente da estática, precisando da execução do código para examiná-lo no seu estado de funcionamento. Nesse caso, para testar contratos inteligentes, por exemplo, a ferramenta de detecção utiliza-se de uma *Blockchain* de teste. Isso permite identificar falhas de lógica, bugs e vulnerabilidades que podem não ser encontradas na análise estática. Durante os testes dinâmicos, são simuladas condições de entrada e cenários de uso para verificar se o contrato se comporta conforme o esperado e se lida adequadamente com transações, exceções e interações com outros contratos (ESQUIVEL *et al.*, 2023).

2.7 Ferramentas de Detecção de Vulnerabilidades

Na detecção de vulnerabilidades em contratos inteligentes, existem ferramentas de detecção de vulnerabilidades que ajudam aos profissionais de segurança a identificar e corrigir potenciais problemas de segurança (ONTIVERO, 2023), permitindo a segurança e integridade dos contratos inteligentes.

Durante as pesquisas realizadas para esta dissertação, foram encontrados os trabalhos de (DURIEUX *et al.*, 2019), (KUSHWAHA *et al.*, 2022), (IMPERIUS; ALAHMAR, 2022) e (PARIZI *et al.*, 2018) que descrevem as ferramentas de detecção de vulnerabilidades em contratos inteligentes. Especificamente, a pesquisa de (DURIEUX *et al.*, 2019) define critérios de inclusão que auxiliam na seleção de 9 ferramentas de detecção de vulnerabilidades em contratos inteligentes: primeiro, a disponibilidade pública e o suporte a uma *Command Line Interface* (CLI), essenciais para permitir a escalabilidade das análises; segundo, ferramentas de suporte aos contratos *Solidity* como entrada, excluindo aquelas que consideram apenas *bytecode* da *Ethereum Virtual Machine* (EVM); terceiro, as ferramentas que realizam análises apenas com o código-fonte do contrato, sem a necessidade de suítes de teste; por último, ferramentas que identificam vulnerabilidades ou más práticas nos contratos, excluindo aquelas que apenas fornecem artefatos como gráficos de fluxo de controle.

A seguir, são detalhadas as ferramentas selecionadas a partir do estudo de (DURIEUX *et al.*, 2019):

- ***HoneyBadger*:** é uma ferramenta de análise estática de código aberto, desenvolvida em 2019 usando *Python* (TORRES *et al.*, 2019) (KUSHWAHA *et al.*, 2022).
- ***Maian*:** é uma ferramenta de análise dinâmica de código aberto implementada em *Python*, desenvolvida em 2018 (NIKOLIC *et al.*, 2018) (KUSHWAHA *et al.*, 2022).
- ***Osiris*:** É uma ferramenta de análise estática de código aberto implementada em *Python* em 2018 (TORRES *et al.*, 2018) (KUSHWAHA *et al.*, 2022).
- ***Oyente*:** é uma ferramenta de análise estática de código aberto implementada em *Python* em 2016. É uma das primeiras ferramentas criadas (LUU *et al.*, 2016).
- ***Slither*:** é uma ferramenta de análise estática de código aberto baseada em *Python*, para contratos inteligentes *Solidity*, desenvolvida em 2018 (FEIST *et al.*, 2019).
- ***Mythril*:** é uma ferramenta de análise de código aberto implementada na linguagem de programação *Python* em 2017 pela *ConsenSys Ethereum* (SHARMA; SHARMA, 2022).
- ***Securify*:** é uma ferramenta de análise estática para verificação de contratos inteligentes *Ethereum*, desenvolvida em *Java* em 2018 (TSANKOV *et al.*, 2018).
- ***Manticore*:** é uma ferramenta de análise dinâmica de código aberto, desenvolvida em *Python* no ano de 2017 pela *Trail of Bits*. *Manticore*. A arquitetura da *Manticore* é composta por componentes primários e secundários. Os componentes primários são composto pelo módulo de execução da *Ethereum* e o *engine* da ferramenta. Já os secundários são composto

pelo módulo da *Application Programming Interface* (API) e pelo módulo de sistema de eventos (MOSSBERG *et al.*, 2019).

– ***SmartCheck*:** é uma ferramenta de código aberto desenvolvida em *Java* pela *SmartDec* em 2017 de análise estática que busca por padrões de vulnerabilidade e más práticas de programação (TIKHOMIROV *et al.*, 2018).

A Tabela 1 lista outras informações sobre as ferramentas, como ano de criação, última atualização, linguagem de programação usada no seu desenvolvimento e o tipo de análise. Todas as ferramentas listadas encontram-se disponíveis no *GitHub*. Essas ferramentas foram inicialmente planejadas para detectar vulnerabilidades relacionadas a plataforma *Ethereum* (HE *et al.*, 2023). Observa-se que algumas delas receberam atualizações nos últimos anos, como mostra a coluna 'Última atualização'.

Tabela 1 – Ferramentas de detecção de vulnerabilidades de contratos inteligentes.

Ferramenta	**Ano de Criação**	**Última Atualização**	**Linguagem de Programação**	**Tipo de Análise**
HoneyBadger	2019	-	*Python*	Estática
Maian	2018	-	*Python*	Dinâmica
Osiris	2018	-	*Python*	Estática
Oyente	2016	2020	*Python*	Estática
Slither	2018	2024	*Python*	Estática
Mythril	2017	2024	*Python*	Estática
Securify	2018	-	*Java*	Estática
Manticore	2017	2022	*Python*	Dinâmica
SmartCheck	2017	2020	*Java*	Estática

Fonte: elaborada pelo autor.

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados trabalhos nas temáticas de análises em contratos inteligentes, experimentação com ferramentas de detecção de vulnerabilidades de contratos inteligentes e/ou análises de conjuntos de dados de contratos inteligentes. Inicialmente, apresentamos uma visão geral sobre a seleção dos trabalhos relacionados a esta pesquisa (seção 3.1). Em seguida, discutimos o serviço de teste *Fuse* e sua abordagem *black-box* para testar contratos inteligentes (seção 3.2). Exploramos também uma avaliação empírica de ferramentas de análise automatizada utilizando grandes conjuntos de dados de contratos inteligentes (seção 3.3). Além disso, destacamos uma solução baseada em análise estática para detecção de vulnerabilidades utilizando Árvore de Sintaxe Abstrata (AST) (seção 3.4). Posteriormente, discutimos os experimentos com diferentes ferramentas para detecção de vulnerabilidades (seção 3.5), bem como a aplicação de testes metamórficos na detecção de vulnerabilidades (seção 3.6). Por fim, apresentamos uma análise sobre ferramentas *Static Application Security Testing* (SAST) e suas limitações na detecção de vulnerabilidades (seção 3.7), seguida de uma comparação detalhada entre os trabalhos analisados (seção 3.8).

3.1 Visão Geral

Os trabalhos relacionados (MEI *et al.*, 2019) e (DURIEUX *et al.*, 2019) foram inicialmente obtidos a partir da revisão sistemática sobre testes de contratos inteligentes para aplicações de *Blockchain*, feita por (IMPERIUS; ALAHMAR, 2022). Nesta revisão sistemática, os autores resumem pesquisas em *Blockchain*, linguagens de desenvolvimento de contratos inteligentes, e processo de desenvolvimento, métodos de teste e ambiente de teste para aplicações de *Blockchain*.

Além disso, buscou-se também outros trabalhos na literatura com o intuito de selecionar pesquisas que envolvem experimentação com contratos inteligentes junto do uso de análise estática e/ou dinâmica. Para encontrar os trabalhos (ESQUIVEL *et al.*, 2023), (HE *et al.*, 2023) e (LI, 2023) utilizou-se o *Google Scholar*.

Todos os trabalhos relacionados ao objetivo desta dissertação são apresentados nas seções seguintes em ordem cronológica de publicação, abrangendo o período de 2019 a 2024.

3.2 *A Fuzz Testing Service for Assuring Smart Contracts*

O trabalho de (MEI *et al.*, 2019) apresenta um serviço de teste chamado *Fuse* para testar códigos de contratos inteligentes. O serviço utiliza uma abordagem *black-box* para gerar casos de teste, na qual o foco está nas entradas e saídas esperadas, nos requisitos funcionais e nas especificações. Os resultados obtidos pelos autores mostram que o serviço *Fuse* detectou vulnerabilidades da *Ethereum* com taxas de verdadeiro positivo de 96-100%. Os autores também disponibilizaram o serviço Fuse como uma ferramenta denominada *ContractFuzzer* (JIANG *et al.*, 2018). Após obter esses resultados, os autores compararam a capacidade de detecção de vulnerabilidades da *ContractFuzzer* com a ferramenta *Oyente* (LUU *et al.*, 2016) e observaram que a ferramenta *ContractFuzzer* detectou cerca de 50% menos casos de verdadeiros positivos (VP). Os autores enfatizam a necessidade de aprimorar o serviço de detecção de vulnerabilidades, uma vez que este identificou apenas 50% das vulnerabilidades em comparação com a ferramenta *Oyente*. Cada vulnerabilidade no serviço Fuse é acompanhada por um conjunto de definições detalhadas (JIANG *et al.*, 2018), que funcionam como um oráculo de teste para possibilitar a detecção. Contudo, é importante destacar que as definições dos oráculos de teste são eficazes apenas para análises offline, o que torna o serviço *Fuse* menos eficiente em determinados casos.

3.3 *Empirical Review of Automated Analysis Tools on 47.587 Ethereum Smart Contracts*

Em (DURIEUX *et al.*, 2019), é apresentado uma avaliação empírica com 9 ferramentas de detecção de contratos inteligentes, usando 2 conjuntos de dados: i) conjunto de dados com 69 contratos inteligentes vulneráveis classificados, que podem ser usados para avaliar a precisão das ferramentas de análise; e ii) conjunto de dados com todos os contratos inteligentes na *Blockchain* do *Ethereum* que possuem código-fonte em Solidity disponível no *Etherscan* (um total de 47.518 contratos). Ambos os conjuntos de dados fazem parte do repositório *SmartBugs*, criado pelos autores para facilitar a integração e comparação entre várias ferramentas de análise e a análise de contratos inteligentes em *Ethereum*. De acordo com os resultados da pesquisa, cerca de 42% das vulnerabilidades do conjunto de dados com os 69 contratos são detectadas por todas as ferramentas, com a ferramenta *Mythril* obtendo a maior precisão (27%). Ao considerar o maior conjunto de dados, observamos que 97% dos contratos são classificados como vulneráveis, sugerindo assim um número considerável de falsos positivos.

3.4 Detecção de Vulnerabilidades em Contratos Inteligentes Utilizando Árvore Sintática Abstrata

A pesquisa feita por (ESQUIVEL *et al.*, 2023) tem como objetivo fornecer documentação sobre a detecção das principais vulnerabilidades em contratos inteligentes em *Solidity* e apresenta uma solução baseada em análise estática, conhecida como AST, chamada AST-Securer. Os resultados do trabalho foram coletados a partir das execuções do ASTSecurer, e utilizaram-se também métricas para a avaliação da confiabilidade e da capacidade de detecção das vulnerabilidades. Para medir o tempo médio de execução da ferramenta na análise de vulnerabilidades, foram analisados 10.480 contratos inteligentes pelo ASTSecurer, resultando um tempo total de análise de 15.250,97 segundos e um tempo médio de execução por contrato de 1,46 segundos. Além disso, realizaram um comparativo do tempo de verificação entre as ferramentas ASTSecurer, *Mythril*, *Oyente*, *Slither* e *Manticore*. Foram utilizados um total de 39 contratos inteligentes para estimar o tempo de execução das ferramentas. Os autores também realizaram um estudo sobre as vulnerabilidades detectadas no conjunto de dados com os 39 contratos e, por meio de uma revisão manual, os contratos inteligentes foram classificados de acordo com as vulnerabilidades encontradas.

3.5 *Detection of Vulnerabilities of Blockchain Smart Contracts*

O trabalho de (HE *et al.*, 2023) realiza experimentos com ferramentas de detecção de vulnerabilidades em contratos inteligentes. Cada ferramenta de detecção é avaliada com base em três requisitos: tipo de vulnerabilidade detectada, precisão da detecção e capacidade de identificar a versão do contrato inteligente. As ferramentas analisadas demonstraram eficiência na detecção de vulnerabilidades como *Reentrancy*, *Timestamp Dependency* e *Integer Overflow*, destacando-se aquelas com menores taxas de falso positivo, como *Vaas*, *Oyente*, *Smartcheck*, e *ContractFuzzer*. Já a capacidade de identificar diferentes versões do contrato inteligente é verificada pelas ferramentas *Oyente*, *Security*, *Slither* e *Mythril*. Apesar do trabalho realizar uma comparação entre diferentes ferramentas de detecção de vulnerabilidades, os autores não apresentam um processo detalhado que possa servir de guia para outros profissionais interessados na análise comparativa dessas ferramentas.

3.6 *Metamorphic Testing for Smart Contract Vulnerabilities Detection*

No trabalho (LI, 2023) é utilizada a técnica de testes metamórficos para detectar vulnerabilidades em contratos inteligentes. Essa técnica de teste é baseada em propriedades e utilizada para atenuar o problema do oráculo dos testes de software. O componente central dos testes metamórficos são as relações metamórficas, que codificam as propriedades necessárias do programa alvo em relação a múltiplas entradas e suas saídas esperadas. Nos experimentos realizados, o trabalho avaliou 67 contratos inteligentes e utilizou as ferramentas *ContractFuzzer*, *Slither* e *Mythril* na detecção de vulnerabilidades de contratos inteligentes. Na avaliação, os autores utilizaram as seguintes métricas: Verdadeiros Positivos (TP), Falsos Negativos (FN) e Falsos Positivos (FP). TP representa o número de contratos que contêm vulnerabilidades corretamente identificadas como vulneráveis pela ferramenta. FN representa o número de contratos vulneráveis não identificados pela ferramenta. FP indica o número de contratos incorretamente declarados como vulneráveis pela ferramenta. Outras métricas consideradas incluíram a Taxa de Verdadeiros Positivos (TPR) e a Taxa de Falsa Descoberta (FDR). Como resultados, a abordagem dos testes metamórficos e a ferramenta *ContractFuzzer* atingiram 100% de TPR. No entanto, a *ContractFuzzer* detectou 29 PFs incorretamente, o que resultou no FDR mais elevado (43,28%) entre as ferramentas. As ferramentas *Slither* e *Mythril* detectaram 30 TPs e atingiram 78,95% de TPR.

3.7 *Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We?*

O estudo de (LI *et al.*, 2024) propõe uma taxonomia atualizada que inclui 45 tipos de vulnerabilidades para contratos inteligentes. Como base nisso, os autores apresentam um *benchmark* com 40 tipos distintos e inclui características de código, padrões de vulnerabilidade e cenários de aplicação. O *benchmark* conta com 8 ferramentas SAST, possui 788 arquivos de contratos inteligentes e 10.394 vulnerabilidades. Os resultados mostram que as ferramentas SAST existentes falham em detectar cerca de 50% das vulnerabilidades no *benchmark* e possuem altos falsos positivos, com precisão não ultrapassando de 10%. Os autores ressaltam que, combinando os resultados de várias ferramentas, a taxa de falsos negativos pode ser reduzida. No entanto, observa-se nos resultados uma dificuldade na detecção de vulnerabilidades, como *Access Control* e *Reentrancy*.

3.8 Comparação entre os trabalhos relacionados

Com base na Tabela 2, observa-se que todos os trabalhos relacionados utilizam a análise estática, como indicado pelos estudos (MEI *et al.*, 2019), (DURIEUX *et al.*, 2019), (ESQUIVEL *et al.*, 2023), (HE *et al.*, 2023), (LI, 2023) e (LI *et al.*, 2024). No entanto, apenas os estudos de (DURIEUX *et al.*, 2019), (ESQUIVEL *et al.*, 2023) e (LI *et al.*, 2024) fazem uso também da análise dinâmica.

O presente trabalho (PERCI) também utiliza tanto a análise estática quanto a dinâmica, mas o seu diferencial reside na forma de detecção das vulnerabilidades, visto que não somente compara os resultados das análises estática e dinâmica, ele utiliza as duas automaticamente em conjunto para a detecção de vulnerabilidades. Além disso, o PERCI incorpora uma aplicação prática para validar as descobertas em um ambiente real, aumentando a aplicabilidade da solução.

Tabela 2 – Características comuns entre os trabalhos.

Trabalho	**Usa Análise Estática**	**Usa Análise Dinâmica**
(MEI *et al.*, 2019)	X	
(DURIEUX *et al.*, 2019)	X	X
(ESQUIVEL *et al.*, 2023)	X	X
(HE *et al.*, 2023)	X	
(LI, 2023)	X	
(LI *et al.*, 2024)	X	X

Fonte: elaborada pelo autor.

A Tabela 3 lista as ferramentas que cada trabalho relacionado utiliza para detectar vulnerabilidades em contratos inteligentes. Pode-se observar que as ferramentas mais utilizadas são *Oyente* em 5 trabalhos, *Mythril* e *Slither* em 4 trabalhos, seguidas por *Manticore* que aparece em 3 trabalhos. No presente trabalho, as ferramentas *Mythril*, *Slither* e *Manticore* também são utilizadas nos experimentos, conforme será apresentado no Capitulo 5.

Tabela 3 – Ferramentas presentes nos experimentos dos trabalhos relacionados.

Trabalho/Ferramenta	***HoneyBadger***	***Maian***	***Manticore***	***Mythril***	***Osiris***	***Oyente***	***Securify***	***Slither***	***SmartCheck***	***ContractFuzzer***
(MEI *et al.*, 2019)						X				
(DURIEUX *et al.*, 2019)	X	X	X	X	X	X	X	X		
(ESQUIVEL *et al.*, 2023)			X	X		X		X		
(HE *et al.*, 2023)						X			X	X
(LI, 2023)				X				X	X	X
(LI *et al.*, 2024)		X	X	X	X	X	X	X	X	
Total	1	2	3	4	2	5	2	4	2	2

Fonte: elaborada pelo autor.

4 PERCI

Este capítulo apresenta o PERCI, um Processo de vERificação de Contratos Inteligentes para aplicações IoT. Primeiro, é apresentada uma visão geral do PERCI para, em seguida, ser realizado um detalhamento de cada etapa do PERCI. Para isso, este capítulo está organizado da seguinte forma: a Seção 4.1 descreve uma visão geral do PERCI e a Seção 4.2 apresenta o detalhamento das etapas do PERCI.

4.1 Visão Geral

O PERCI é um processo de verificação de contratos inteligentes para aplicações IoT. O PERCI possui um conjunto de etapas de verificação de contratos inteligentes com o intuito de detectar vulnerabilidades utilizando a combinação de ferramentas de análises estática e dinâmica. A combinação dessas análises é proposta para melhorar a detecção de vulnerabilidades, proporcionando uma solução mais robusta.

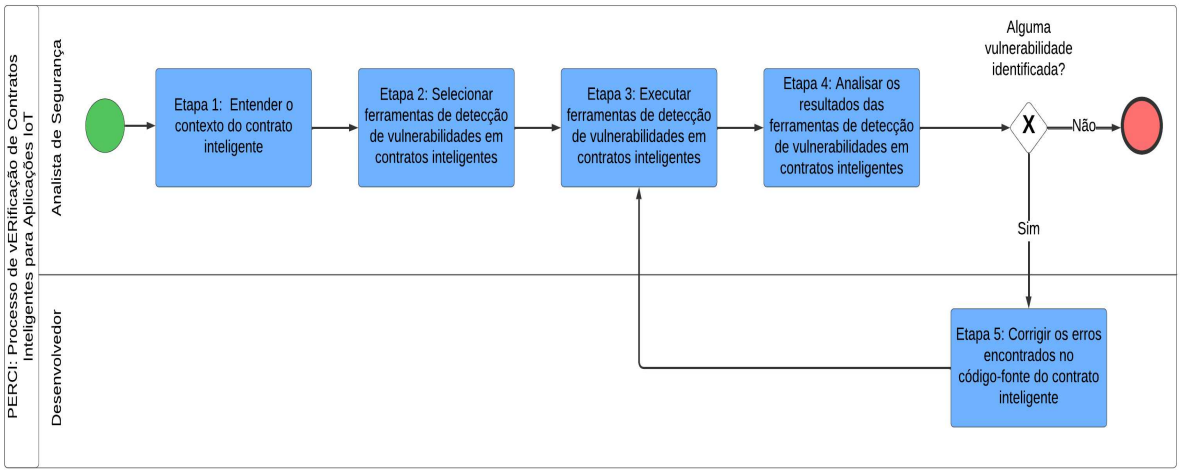
O PERCI foi idealizado a partir das pesquisas de (KUSHWAHA *et al.*, 2022), (HE *et al.*, 2023) e (DURIEUX *et al.*, 2019). O objetivo dessas pesquisas é abordar o estado atual da área de segurança de contratos inteligentes e as ferramentas utilizadas para detectar vulnerabilidades. Além disso, essas pesquisas relatam a importância do uso de um processo com combinação de diferentes tipos de análises na verificação de segurança de contratos inteligentes.

Uma das principais características do PERCI é a flexibilidade em termos do tipo de vulnerabilidade e as ferramentas de detecção de vulnerabilidades em contratos inteligentes. Outra característica do PERCI é a automação das etapas que levam mais tempo, se executadas manualmente, possibilitando a realização dos testes de uma forma mais rápida. Ao utilizar o PERCI, pode-se mitigar os riscos e evitar falhas de segurança que podem comprometer os dados sensíveis ou funcionamento dos dispositivos.

O PERCI pode ser usado como referência por estudantes, pesquisadores e/ou profissionais em geral interessados na área de segurança de contratos inteligentes em aplicações IoT, assim como pode ser utilizado como subsídio para otimizar a tomada de decisão de analistas de segurança e desenvolvedores quanto ao uso de plataformas de *Blockchain*, ferramentas de detecção de contratos inteligentes, linguagens de programação e conjuntos de dados de vulnerabilidades em projetos da área. Dessa forma, o PERCI busca contribuir para a identificação de vulnerabilidades em contratos inteligentes antes da fase de implantação na *Blockchain*.

A Figura 5 ilustra as cinco etapas do PERCI, as quais são detalhadas na sequência do capítulo.

Figura 5 – Etapas do PERCI.



Fonte: elaborada pelo autor.

No PERCI é proposta a atuação de dois perfis: o Analista de Segurança e o Desenvolvedor. Conforme mostra a Figura 5, o Analista de Segurança é o responsável pela execução das quatro primeiras etapas do processo, partindo do entendimento do contrato até a interpretação dos resultados das ferramentas de detecção. O Desenvolvedor, por sua vez, é responsável pela execução da última etapa, ou seja, implementando as correções necessárias, garantindo que o contrato inteligente funcione conforme esperado.

É desejável que os dois perfis possuam um conjunto de expertises teóricas e técnicas para desempenhar com propriedade seus papéis. Por exemplo, inicialmente, é preciso ter conhecimentos de Web (ex: conceitos e diferenças entre Web 2.0 e 3.0) e segurança da informação (ex: controle de acesso, criptografia, gestão de vulnerabilidades, boas práticas de codificação segura, etc). Notoriamente, também são desejáveis conhecimentos conceituais e práticos de funcionamento da *Blockchain*, por exemplo, conhecendo as plataformas mais utilizadas na atualidade, tais como a *Ethereum* (Seção 2.3). Por fim, o último tópico relevante é o de contratos inteligentes (Seção 2.4). Neste caso, ter expertise em uma ou mais linguagens de programação para implementação de contratos inteligentes em plataformas *Blockchain*, tal como a linguagem *Solidity*, que é amplamente usada para escrever contratos inteligentes na *Ethereum*.

4.2 Detalhamento das etapas do PERCI

Esta seção aborda em detalhes as 5 etapas do PERCI: Etapa 1 - Entender o contexto do contrato inteligente (Subseção 4.2.1); Etapa 2 - Selecionar ferramentas de detecção de vulnerabilidades em contratos inteligentes (Subseção 4.2.2); Etapa 3 - Executar ferramentas de detecção de vulnerabilidades em contratos inteligentes (Subseção 4.2.3); Etapa 4 - Analisar os resultados das ferramentas de detecção de vulnerabilidades em contratos inteligentes (Subseção 4.2.4); Etapa 5 - Corrigir os erros encontrados no código do contrato inteligente (Subseção 4.2.5).

4.2.1 Etapa 1: Entender o contexto do contrato inteligente

A primeira etapa do PERCI consiste no entendimento do contexto do contrato inteligente a ser submetido à análise das ferramentas. Nesta etapa, é essencial que o Analista de Segurança, responsável pela execução da etapa, identifique a plataforma *Blockchain* e a linguagem de programação usadas. É importante também que o Analista de Segurança obtenha uma compreensão abrangente da lógica de negócios e funcionalidades do contrato, entendendo o código-fonte do contrato em si, assim como conhecendo os conjuntos de dados de contratos inteligentes já disponíveis. Na sequência desta seção, aprofunda-se sobre estes dois últimos tópicos, respectivamente.

4.2.1.1 Compreensão do contrato inteligente

Na etapa "Compreensão do contrato inteligente", o Analista de Segurança precisa ter acesso ao código-fonte do contrato inteligente para entender a sua estrutura interna, como as variáveis declaradas, funções e chamadas externas por outros contratos inteligentes.

Como forma de demonstrar o uso dos contratos inteligentes no contexto de uma aplicação, são apresentados na sequência um exemplo de uso de contrato com uma aplicação simplificada e outro exemplo de uso de uma aplicação IoT.

4.2.1.2 Aplicação simplificada

Com o intuito de demonstrar a sintaxe simplificada de um contrato inteligente escrito em *Solidity*, o Código-fonte 1 exemplifica um contrato de um leilão digital no qual os participantes podem fazer lances. Se um novo lance superar o lance atual mais alto, o novo lance é aceito, e o lance anterior é reembolsado ao participante que o fez. Esse contrato possui uma

vulnerabilidade de Negação de Serviço (*Denial of Service* - DoS) em um leilão e não deve ser utilizado em aplicações reais devido a essa falha (DURIEUX *et al.*, 2019).

Código-fonte 1 – Exemplo de Contrato *Solidity* de leilão digital com vulnerabilidade de DoS.

```
1  /*
2   * @source: https://github.com/trailofbits/not-so-smart-
       contracts/blob/master/denial_of_service/auction.sol
3   * @author: -
4   * @vulnerable_at_lines: 23
5   */
6  pragma solidity ^0.4.15;
7  //Auction susceptible to DoS attack
8  contract DosAuction {
9    address currentFrontrunner;
10   uint currentBid;
11   //Takes in bid, refunding the frontrunner if they are
       outbid
12   function bid() payable {
13     require(msg.value > currentBid);
14     //If the refund fails, the entire transaction reverts.
15     //Therefore a frontrunner who always fails will win
16     if (currentFrontrunner != 0) {
17       //E.g. if recipients fallback function is just revert
           ()
18       // <yes> <report> DENIAL_OF_SERVICE
19       require(currentFrontrunner.send(currentBid));
20     }
21     currentFrontrunner = msg.sender;
22     currentBid         = msg.value;
23   }
24 }
```

No exemplo apresentado no Código-fonte 1, o contrato inteligente inicia com a diretiva `pragma solidity`, que especifica a versão do compilador *Solidity* a ser utilizada — no caso, a versão 0.4.15. A palavra-chave `contract` é usada para definir um novo contrato denominado `DosAuction`. Entre as variáveis de estado declaradas, temos `address currentFrontrunner`, que armazena o endereço do participante atual, e `uint currentBid`, um número inteiro sem sinal que representa o valor do lance atual. A função `bid()` é definida como `payable`, indicando que pode receber valores em ether. Dentro dessa função, a instrução `require` assegura que o valor enviado na transação `(msg.value)` seja superior ao lance atual `(currentBid)`.

Caso essa condição não seja satisfeita, a transação é revertida. Em seguida, o código verifica se já existe um participante anterior `(currentFrontrunner != 0)`. Se existir, o contrato tenta reembolsar o valor do lance anterior `(currentBid)` ao respectivo endereço `(currentFrontrunner)` utilizando a função `send`. Se o reembolso falhar, a instrução `require` força a reversão de toda a transação. Caso o reembolso seja bem-sucedido, o contrato atualiza o participante atual para `msg.sender` e o valor do lance para `msg.value`.

4.2.1.3 Aplicação IoT

O Código-fonte 2 exemplifica um contrato inteligente mais complexo que o apresentado no Código-fonte 1. Ele permite o registro utilizando o `hash` da chave pública do dispositivo como identificador único, possibilitando a comunicação com os demais dispositivos. Esse contrato é baseado na pesquisa apresentada em (ŠIMUNIĆ, 2018), na qual é proposto um sistema de gerenciamento de dispositivos IoT utilizando contratos inteligentes e a *Blockchain*.

Código-fonte 2 – Contrato *Solidity* para registrar dispositivo IoT.

```
1 pragma solidity ^0.8.0;
2 contract IoTDeviceRegistry {
3     address public admin;
4     struct Device {
5         address owner;
6         bytes32 publicKeyHash;
7         bool isRegistered;
8     }
```

```
9     mapping(bytes32 => Device) public devices;
10    mapping(address => bytes32[]) public ownerDevices;
11    event DeviceRegistered(address indexed owner, bytes32
          indexed publicKeyHash);
12    modifier onlyAdmin() {
13        require(msg.sender == admin, "Only admin can
              perform this action");
14        _;
15    }
16    constructor() {
17        admin = msg.sender;
18    }
19    // Register a new device
20    function registerDevice(bytes32 publicKeyHash) public {
21        require(!devices[publicKeyHash].isRegistered, "
              Device already registered");
22        devices[publicKeyHash] = Device({
23            owner: msg.sender,
24            publicKeyHash: publicKeyHash,
25            isRegistered: true
26        });
27        ownerDevices[msg.sender].push(publicKeyHash);
28        emit DeviceRegistered(msg.sender, publicKeyHash);
29    }
30    // Get devices by owner
31    function getDevicesByOwner(address owner) public view
          returns (bytes32[] memory) {
32        return ownerDevices[owner];
33    }
34    // Change admin
35    function changeAdmin(address newAdmin) public onlyAdmin
36    {
```

```
37          admin = newAdmin;
38      }
39  }
```

A estrutura principal do contrato apresentado no Código-fonte 2 inclui a definição de uma estrutura `struct Device` para armazenar dados sobre cada dispositivo, como o endereço do proprietário (`address owner`), o hash da chave pública (`bytes32 publicKeyHash`) e o status de registro (`bool isRegistered`). Utilizando mapeamentos, o contrato gerencia os dispositivos registrados (`mapping(bytes32 => Device) public devices`) e suas associações com os respectivos proprietários (`mapping(address => bytes32[]) public ownerDevices`).

Um evento `DeviceRegistered` é emitido sempre que um novo dispositivo é registrado. Além disso, o contrato possui um modificador `onlyAdmin` que restringe determinadas operações exclusivamente ao administrador designado. A função `registerDevice` permite o registro de novos dispositivos na rede IoT, garantindo que não estejam previamente cadastrados. Já a função `getDevicesByOwner` retorna uma lista de dispositivos associados a um determinado proprietário.

De forma geral, percebe-se que a programação de contratos inteligentes, como os exemplos apresentados, exige conhecimento de linguagens de programação, como a linguagem *Solidity*, a mais utilizada na plataforma *Ethereum*. Além disso, é fundamental compreender o funcionamento da *Blockchain* e da *Ethereum Virtual Machine* (EVM), especialmente ao se utilizar essa plataforma.

4.2.1.4 Conjuntos de dados de contratos inteligentes

A pesquisa de (DURIEUX *et al.*, 2019) destaca a importância da disponibilização pública de conjuntos de dados para que novos estudos possam reproduzir e comparar seus resultados com trabalhos anteriores. O estudo também relata que, ao desenvolver uma nova ferramenta, muitas vezes é necessário entrar em contato com os autores de soluções existentes e aguardar a liberação dos conjuntos de dados, o que pode dificultar a comparação entre abordagens. Diante desse cenário, para facilitar os estudos e o uso dos contratos inteligentes em um ambiente experimental, a literatura já disponibiliza alguns conjuntos de dados que exploram e classificam uma ampla variedade de vulnerabilidades em contratos inteligentes (DURIEUX *et al.*, 2019; LIU *et al.*, 2021; ANGELO *et al.*, 2023; LIU *et al.*, 2023).

A Tabela 4 sumariza quatro conjuntos de dados, disponíveis na literatura, com vulnerabilidades conhecidas presentes nos contratos inteligentes para uso da comunidade. Cada linha da tabela apresenta um conjunto de dados, a quantidade de contratos inteligentes nele contidos e a respectiva referência.

Tabela 4 – Conjuntos de dados para estudos das vulnerabilidades.

Conjunto de dados	**Qtd. de contratos inteligentes**	**Referência**
SmartBugs Curated	69	(DURIEUX *et al.*, 2019)
SmartBugs Wild	47.587	(LIU *et al.*, 2021)
Skelcodes	248.328	(ANGELO *et al.*, 2023)
Ethereum Smart Contract	12.000	(LIU *et al.*, 2023)

Fonte: elaborada pelo autor.

O primeiro conjunto de dados, o *SmartBugs Curated*, contém 69 contratos inteligentes reais identificados como vulneráveis ou desenvolvidos propositalmente para ilustrar vulnerabilidades específicas. Seu objetivo é fornecer uma base confiável de contratos com vulnerabilidades conhecidas. O segundo conjunto, o *SmartBugs Wild*, contém 47.587 contratos extraídos da *Ethereum*. O terceiro, denominado *Skelcodes*, é um repositório com 248.328 contratos, incluindo uma indicação sobre a disponibilidade do código-fonte no *Etherscan* (uma ferramenta que permite visualizar informações públicas armazenadas na *Ethereum*). Por fim, o conjunto *Ethereum Smart Contract* inclui 12.000 contratos também extraídos da *Ethereum*.

Conforme será mostrado no Capítulo 5, será utilizado para avaliação do PERCI o conjunto de dados *SmartBugs Curated*. A Tabela 5 apresenta informações das categorias de vulnerabilidade dos 69 contratos do *SmartBugs Curated*. Para cada categoria, é apresentada uma descrição, o nível em que o ataque pode ser mitigado e o número de contratos disponíveis dentro dessa categoria.

4.2.2 Etapa 2: Selecionar ferramentas de detecção de vulnerabilidades em contratos inteligentes

Nesta etapa, deve-se realizar a seleção das ferramentas de detecção de vulnerabilidades em contratos inteligentes (Seção 2.7). Conforme fundamentado na Seção 2.6, é recomendado para aplicação deste processo selecionar pelo menos uma ferramenta de análise estática e uma de análise dinâmica dado a característica do PERCI em usar a combinação das duas análises.

A seleção dessas ferramentas pode ser baseada no estudo de (DURIEUX *et al.*, 2019), que define critérios de inclusão que auxiliam nessa seleção. Tais critérios de inclusão para

Tabela 5 – Categorias de vulnerabilidades disponíveis no conjunto de dados *SmartBugs Curated*.

Categoria	**Descrição**	**Nível**	**Contratos**
Controle de Acesso	Falha ao usar modificadores de função ou uso de *tx.origin*	*Solidity*	17
Aritmética	*Overflow/Underflow* de inteiros	*Solidity*	14
Má Aleatoriedade	Minerador malicioso influencia o resultado	*Blockchain*	8
Negação de Serviço	O contrato é sobrecarregado com computações demoradas	*Solidity*	6
Front Running	Duas transações dependentes que invocam o mesmo contrato são incluídas em um bloco	*Blockchain*	4
Reentrância	Chamadas de funções reentrantes fazem o contrato se comportar de maneira inesperada	*Solidity*	7
Endereços Curtos	A própria EVM aceita argumentos com *padding* incorreto	EVM	1
Manipulação de Tempo	O timestamp do bloco é manipulado pelo minerador	*Blockchain*	4
Chamadas Não Verificadas	*call()*, *callcode()*, *delegatecall()* ou *send()* falham e não são verificadas	*Solidity*	5

Fonte: adaptada de (DURIEUX *et al.*, 2019).

essas ferramentas são: primeiro, a disponibilidade pública e o suporte a uma *Command Line Interface* (CLI), essenciais para permitir a escalabilidade das análises; segundo, ferramentas de suporte aos contratos *Solidity* como entrada, excluindo aquelas que consideram apenas *bytecode* da *Ethereum Virtual Machine* (EVM); terceiro, as ferramentas que realizam análises apenas com o código-fonte do contrato, sem a necessidade de suítes de teste; por último, ferramentas que identificam vulnerabilidades ou más práticas nos contratos, excluindo aquelas que apenas fornecem artefatos como gráficos de fluxo de controle.

Além disso, é recomendado analisar e escolher as ferramentas pensando também no ambiente de execução que precisa ser configurado. Por exemplo, é importante verificar as atualizações da ferramenta, últimas versões disponibilizadas, as linguagens que a ferramenta suporta e, por fim, a plataforma *Blockchain* para análise dos contratos. Essa recomendação é devido aos diferentes pacotes necessários para configurar o ambiente de execução para cada ferramenta.

Após essa seleção das ferramentas, é necessário criar o ambiente de experimentação para testar o contrato inteligente no início da Etapa 3.

4.2.3 Etapa 3: Executar ferramentas de detecção de vulnerabilidades em contratos inteligentes

Nesta etapa, o contrato inteligente é submetido às ferramentas de análise para explorar o código-fonte e identificar possíveis vulnerabilidades presentes. Cada ferramenta utiliza uma abordagem diferente para a análise e, ao final, gera um relatório no formato *JavaScript Object Notation* (JSON) apontando cada vulnerabilidade encontrada. Essas ferramentas utilizam das análises estática e dinâmica para encontrar erros de segurança no código. Cada uma dessas análises oferece uma perspectiva única sobre a segurança do contrato inteligente, e o uso combinado delas proporciona uma identificação das possíveis vulnerabilidades de forma mais abrangente.

Ao utilizar ferramentas de detecção de vulnerabilidades de contratos inteligentes, alguns desafios podem surgir, especialmente com relação à instalação e interpretação das saídas geradas por essas ferramentas.

A utilização dessas ferramentas não é um procedimento trivial, especialmente para iniciantes. Cada ferramenta possui suas particularidades, o que pode exigir um conhecimento mais aprofundado que vai além dos conhecimentos sobre *Solidity* e dos tipos de vulnerabilidades que cada ferramenta é capaz de identificar. Além disso, as ferramentas também podem gerar falsos positivos, requerendo uma interpretação cuidadosa dos seus resultados. A instalação e configuração dessas ferramentas pode ser um desafio significativo no início. Ferramentas como a *Mythril*, por exemplo, podem requerer a instalação de dependências específicas e ajustes no ambiente para funcionar corretamente. A *Manticore*, uma ferramentas de análise dinâmica, pode precisar de ajustes na configuração do sistema, as versões das ferramentas e pacotes devem ser compatíveis.

A configuração da saída é outro ponto a considerar. Algumas ferramentas não exportam suas saídas automaticamente após concluir a análise, precisando de uma configuração adicional para exportar seus resultados.

4.2.4 Etapa 4: Analisar os resultados das ferramentas de detecção de vulnerabilidades em contratos inteligentes

Nesta etapa 4, é realizada a verificação e a análise dos resultados obtidos pelas ferramentas na etapa anterior.

Dependendo da configuração do ambiente definido na Etapa 3, os resultados são extraídos das estruturas definidas. Em um ambiente com a configuração manual, os resultados são analisados individualmente a partir dos arquivos JSON gerados. Já em um ambiente com a configuração automatizada, os resultados podem ser analisados conforme uma estrutura de saída criada. A análise deve considerar a criticidade e a frequência das vulnerabilidades encontradas, comparando os resultados de diferentes ferramentas para identificar padrões ou inconsistências. Conforme será apresentado no Capítulo 5, a etapa foi realizada de forma automatizada.

Esses arquivos JSON são então repassados ao Desenvolvedor (Etapa 5), destacando as vulnerabilidades encontradas e fornecendo recomendações para ações corretivas. O objetivo é garantir que os desenvolvedores possam tomar medidas para mitigar os riscos e melhorar a segurança do contrato inteligente.

Após a conclusão desta etapa, na qual as vulnerabilidades em contratos inteligentes são identificadas e analisadas, o processo avança para a Etapa 5, que é crucial para a segurança e a eficácia dos contratos: a implementação de correções no código com base nos problemas identificados.

4.2.5 Etapa 5: Corrigir os erros encontrados no código-fonte do contrato inteligente

Esta etapa envolve a implementação de correções das vulnerabilidades identificadas no código do contrato inteligente. O Desenvolvedor deve abordar cada vulnerabilidade listada nos arquivos JSON gerados na etapa anterior. A correção das vulnerabilidades exige atenção aos detalhes para garantir que as mudanças no código não introduzam novos problemas.

Conforme ilustra a Figura 5, após a implementação das correções, o contrato inteligente retorna à Etapa 3, onde é submetido novamente aos testes pelas ferramentas para assegurar que todas as vulnerabilidades estão resolvidas. Este ciclo de teste e correção se repete até que o contrato inteligente esteja livre de vulnerabilidades conhecidas, garantindo a segurança e a integridade do código antes de seu lançamento final. Cada iteração corrige vulnerabilidades existentes, minimizando os riscos associados aos contratos inteligentes.

5 AVALIAÇÃO DO PERCI

Este capítulo tem como objetivo apresentar a avaliação do PERCI em função de cada etapa do processo apresentada na Seção 4.2. Assim, o capítulo é composto da Seção 5.1, que detalha os contratos inteligentes criados e a aplicação, utilizados na avaliação; da Seção 5.2, que descreve a seleção das ferramentas de detecção de vulnerabilidades; da Seção 5.3, que trata da execução das ferramentas de detecção de vulnerabilidades; da Seção 5.4, que apresenta os resultados obtidos da detecção das ferramentas; e da Seção 5.5, que discute as conclusões conforme os resultados das etapas anteriores do processo.

5.1 Etapa 1: Entender o contexto do contrato inteligente

Esta seção descreve, primeiro, na Subseção 5.1.1, os contratos inteligentes criados para avaliação do PERCI. Em seguida, na Subseção 5.1.2, é apresentada a aplicação IoT ClimaCor.

5.1.1 Contratos inteligentes criados

Para a avaliação do PERCI na Etapa 1, foram criados 3 contratos inteligentes baseados nos contratos disponíveis no conjunto de dados *SmartBugs Curated* (DURIEUX *et al.*, 2019). Esses contratos são escritos na linguagem *Solidity* e usam a versão 4 do compilador. Durante a criação dos contratos para avaliação do PERCI, observou-se incompatibilidades entre as novas versões da *Solidity* e a versão do compilador usado pelo conjunto de dados. Assim, houve a necessidade de atualização para versão 8 do compilador da *Solidity* para utilização dos contratos inteligentes.

Os dois primeiros experimentos usam contratos inteligentes criados com base nos contratos disponíveis no conjunto de dados *SmartBugs Curated* (DURIEUX *et al.*, 2019), ambos da categoria de Reentrância (Tabela 5), que é uma das vulnerabilidades mais populares e utilizadas pelas pesquisas em seus experimentos. Especificamente, o primeiro experimento consiste no uso da função *transfer* no contrato para envio de ETH. Já o segundo experimento faz uso da função *call* no contrato para envio de ETH. O objetivo das configurações específicas desses dois primeiros experimentos é possibilitar a identificação das discrepâncias dos resultados das ferramentas na detecção das vulnerabilidades, pois, mesmo tratando-se de uma vulnerabilidade clássica, as ferramentas demonstram limitações na detecção.

O terceiro experimento usa o mesmo conjunto de dados *SmartBugs Curated*, porém nas categorias Controle de Acesso e Negação de Serviço. Destaca-se que nesse terceiro experimento, diferentemente dos dois anteriores, o contrato foi integrado a uma aplicação IoT denominada ClimaCor, detalhada na sequência na seção 5.1.2. Dada essa integração com essa aplicação, as categorias de Controle de Acesso e Negação de Serviço são escolhidas pelas características de registrar e autenticar presentes no contrato *DeviceRegistry* (Código-fonte 5 no Apêndice 6.4), que é uma adaptação do contrato implementado em (ŠIMUNIĆ, 2018).

A Tabela 6 sumariza os experimentos definidos, informando o Identificador do experimento (ID), a categoria do contrato de acordo com o conjunto de dados *SmartBugs Curated* (DURIEUX *et al.*, 2019), se há ou não o uso de uma aplicação para integrar o contrato inteligente e a justificativa.

Tabela 6 – Experimentos utilizados para avaliação do PERCI.

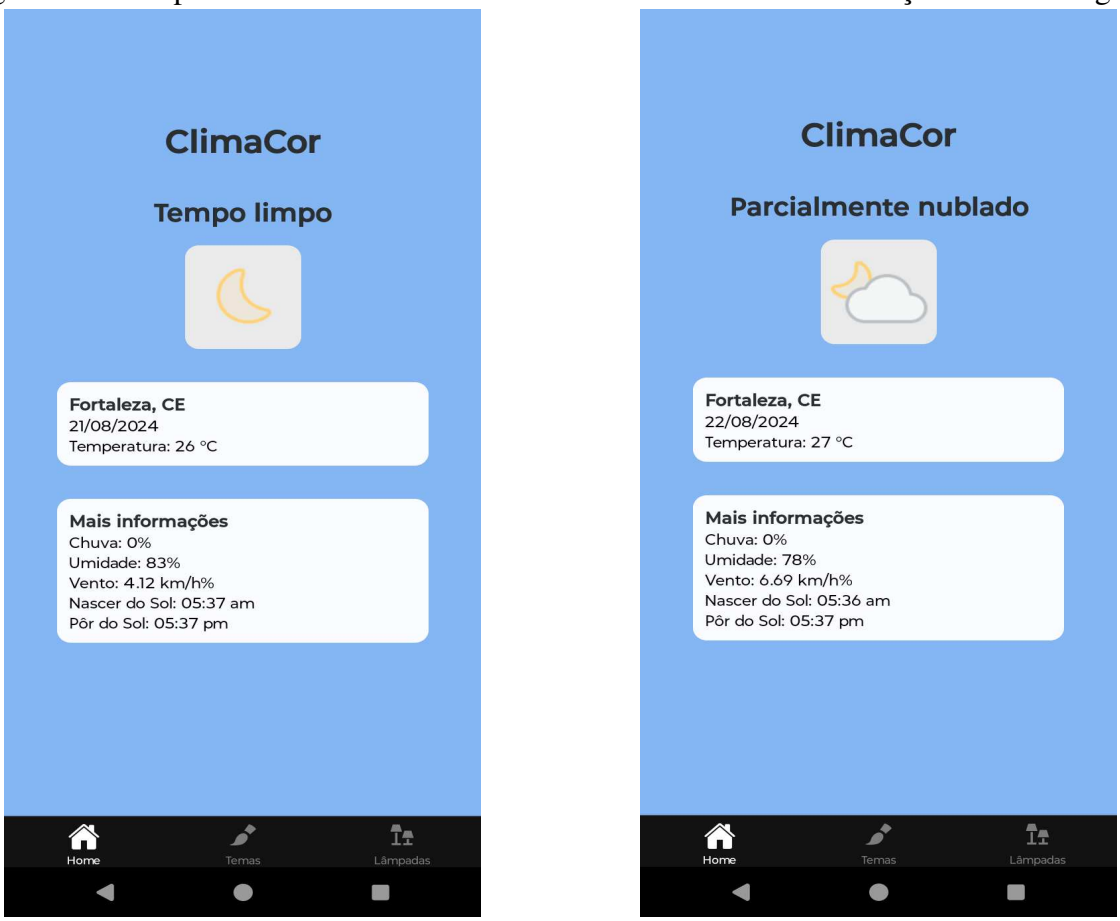
ID	**Categoria do Contrato Criado**	**Utiliza Aplicação**	**Justificativa**
1	Reentrância	Não	A função *transfer* tem limitação de gás. Esse limite impede que o contrato destinatário execute funções recursivamente para manipular o saldo de outro contrato.
2	Reentrância	Não	A função *call* é frequentemente usada em experimentos para simular a vulnerabilidade de *Reentrancy* em contratos inteligentes devido a como ela manipula as transações de envio de ETH.
3	Controle de Acesso e Negação de Serviço	Sim	Foi integrado na aplicação ClimaCor para garantir o uso de um contrato inteligente em um experimento IoT real.

Fonte: elaborada pelo autor.

5.1.2 Aplicação IoT ClimaCor

O ClimaCor é uma aplicação IoT que tem como objetivo transformar a experiência do usuário integrando-se à uma lâmpada inteligente, permitindo que esta seja usada como um indicador visual das condições meteorológicas atuais. A lâmpada muda de cor em tempo real, oferecendo uma representação visual e intuitiva do clima. As cores específicas são atribuídas a cada condição climática, como ensolarado, nublado, chuva, entre outras, proporcionando uma experiência imediata e compreensível para o usuário. A Figura 6 ilustra telas da aplicação ClimaCor em diferentes condições meteorológicas.

Figura 6 – Exemplos de telas do ClimaCor com diferentes dados de condições meteorológicas.



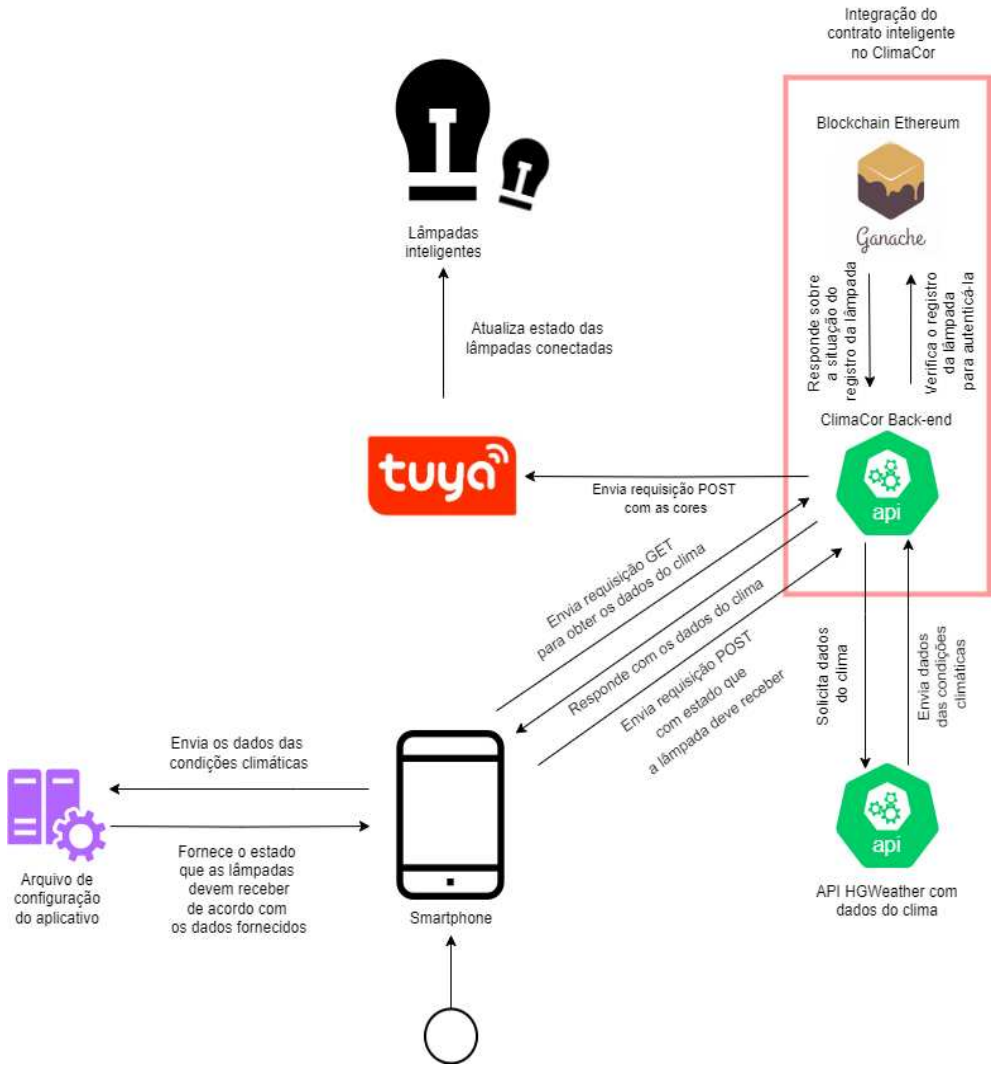
(a) Tempo limpo (b) Parcialmente nublado

Fonte: elaborada pelo autor.

Já a Figura 7 ilustra o fluxo de funcionamento do ClimaCor. Inicialmente, o aplicativo faz uma requisição GET para obter os dados meteorológicos da API *HGWeather*[1], que fornece informações detalhadas sobre as condições climáticas. Em seguida, esses dados são processados pelo aplicativo e enviados para um arquivo de configuração interno, que especifica os estados que as lâmpadas inteligentes devem atingir com base nas condições climáticas atuais. Após a atualização desse arquivo de configuração com os dados climáticos mais recentes, o aplicativo envia as instruções sobre o estado que a lâmpada conectada deve assumir, resultando na atualização do estado da lâmpada conforme as instruções recebidas. Após isso, verifica-se se a lâmpada está registrada na *Blockchain*. Caso afirmativo, uma requisição POST é enviada contendo as cores para a API da *Tuya*[2], que é responsável por atualizar o estado da lâmpada. Caso contrário, o usuário é notificado de que a lâmpada não está registrada, e, portanto, não é possível atualizar seu estado.

1 https://hgbrasil.com/status/weather
2 https://www.tuya.com/

Figura 7 – Fluxo de funcionamento do ClimaCor com destaque para a integração do contrato inteligente.

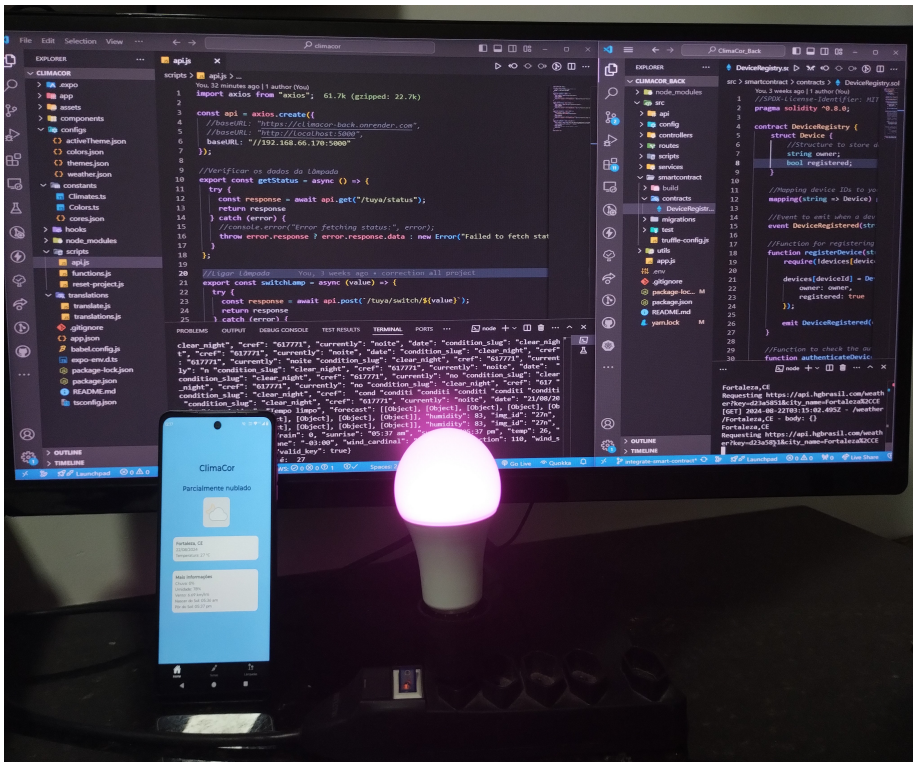


Fonte: elaborada pelo autor.

Neste trabalho, conforme mencionado na seção anterior, o ClimaCor foi integrado ao contrato inteligente desenvolvido para o experimento 3, sendo capaz de registrar e autenticar a lâmpada na *Blockchain Ethereum*. A Figura 8 ilustra o *testbed* montado para o experimento 3. Para fazer a integração, foram necessárias implementações adicionais no código do ClimaCor, como a criação de um módulo para representar o contrato inteligente, ilustrada na parte destacada em vermelho na Figura 7. Na criação desse módulo, fez-se uso do *Truffle* para criação do projeto do contrato inteligente dentro da estrutura de pastas do ClimaCor. Em seguida, realizou-se a compilação do contrato para garantir a execução correta do código. Após a compilação, fez-se a migração do contrato na *Blockchain* local através do *Ganache*[3].

[3] *Blockchain* para desenvolvimento *Ethereum* usada para testar contratos inteligentes.

Figura 8 – *Testbed* do ClimaCor.



Fonte: elaborada pelo autor.

No contexto da integração do ClimaCor com a *Blockchain*, foi realizada uma análise de segurança no contrato inteligente de registro e autenticação das lâmpadas. Durante essa análise, uma vulnerabilidade crítica foi identificada no controle de acesso, especificamente no uso do parâmetro *tx.origin* na função *authenticateDevice*. O uso inadequado de *tx.origin* torna o contrato suscetível a ataques de *phishing* de contrato, onde um atacante pode criar um contrato malicioso para enganar o contrato legítimo, utilizando-se do endereço *tx.origin* para se passar por um proprietário legítimo de um dispositivo registrado. Para ilustrar essa vulnerabilidade, foi realizada a simulação de um ataque controlado, em que um contrato malicioso foi utilizado para manipular a autenticação de um dispositivo, permitindo que o atacante acessasse funções restritas do contrato como se fosse o verdadeiro proprietário. O vídeo de demonstração desse ataque pode ser consultado no Apêndice 6.4, para melhor compreensão do impacto dessa falha na segurança da aplicação.

5.2 Etapa 2: Selecionar ferramentas de detecção de vulnerabilidades em contratos inteligentes

Esta seção especifica a escolha das ferramentas utilizadas nos experimentos para avaliação do PERCI. Para tal seleção, realizou-se a leitura dos trabalhos (DURIEUX *et al.*, 2019) e (KUSHWAHA *et al.*, 2022), que fazem uma revisão da literatura e lista as principais ferramentas existentes para detecção de vulnerabilidades em contratos inteligentes.

As ferramentas selecionadas nesta etapa seguem a classificação entre análise estática e dinâmica, conforme definida em (KUSHWAHA *et al.*, 2022). Com base nisso, a seleção das ferramentas mais utilizadas, conforme mostrado na Tabela 3, foi realizada utilizando os critérios de inclusão definidos por (DURIEUX *et al.*, 2019).

A Tabela 7 apresenta as ferramentas exploradas no estudo de (DURIEUX *et al.*, 2019) de acordo com os critérios de inclusão: primeiro, a disponibilidade pública e o suporte a uma *Command Line Interface* (CLI), essenciais para permitir a escalabilidade das análises; segundo, ferramentas de suporte aos contratos *Solidity* como entrada, excluindo aquelas que consideram apenas *bytecode* da *Ethereum Virtual Machine* (EVM); terceiro, as ferramentas que realizam análises apenas com o código-fonte do contrato, sem a necessidade de suítes de teste; por último, ferramentas que identificam vulnerabilidades ou más práticas nos contratos, excluindo aquelas que apenas fornecem artefatos como gráficos de fluxo de controle.

Tabela 7 – Ferramentas selecionadas no estudo (DURIEUX *et al.*, 2019) para análise de contratos inteligentes.

Ferramentas	**URL**
HoneyBadger	*https://github.com/christoftorres/HoneyBadger*
Maian	*https://github.com/MAIAN-tool/MAIAN*
Manticore	*https://github.com/trailofbits/manticore*
Mythril	*https://github.com/ConsenSys/mythril*
Osiris	*https://github.com/christoftorres/Osiris*
Oyente	*https://github.com/enzymefinance/oyente*
Securify	*https://github.com/eth-sri/securify*
Slither	*https://github.com/crytic/slither*
Smartcheck	*https://github.com/smartdec/smartcheck*

Fonte: adaptada de (DURIEUX *et al.*, 2019).

Inicialmente, fez-se tentativas de instalação das nove ferramentas listadas por (DURIEUX *et al.*, 2019) e foi possível observar alguns pontos de limitações para instalação e configuração. É importante ressaltar que a maioria das ferramentas são compatíveis com Linux e possuem pacotes com versões específicas para serem executadas. Por exemplo, as ferramentas *HoneyBadger*, *Osiris* e *Oyente* usam *Python* ainda na versão 2. A ferramenta *Securify* tem

sua versão depreciada. A ferramenta *Smartcheck* também possui versão depreciada desde 2020, a análise pode funcionar incorretamente para as versões do *Solidity* a partir da 0.6.0 e sua versão Web que estava disponível online anteriormente foi encerrada. Já a ferramenta *Maian* tem problemas com a atualização da biblioteca *Web3.js*.

Após essa seleção inicial das ferramentas usando os critérios de (DURIEUX *et al.*, 2019) e as observações durante as tentativas de instalação das ferramentas, buscou-se filtrar as ferramentas com configurações compatíveis entre si. Diante disso, foram verificadas as versões da linguagem *Python* suportadas por cada ferramenta. Outra observação a considerar no momento da seleção foi a versão da distribuição Linux. Para esse caso, verificou-se a compatibilidade das ferramentas com a distribuição do Ubuntu 22.04.3 *Long-Term Support* (LTS).

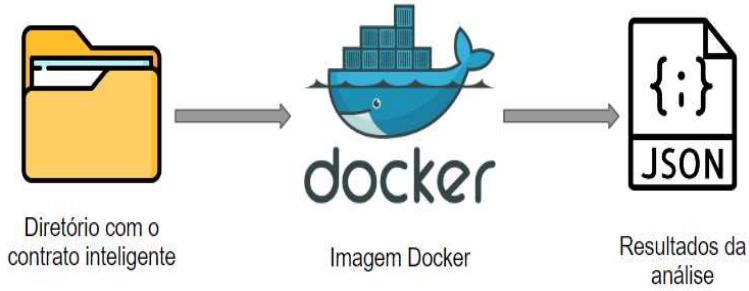
Ao final dessas verificações, foi possível selecionar as ferramentas de detecção de vulnerabilidades de análise estática *Mythril*, *Slither* e de análise dinâmica *Manticore* para fazerem parte da avaliação do PERCI.

5.3 Etapa 3: Executar ferramentas de detecção de vulnerabilidades em contratos inteligentes

Inicialmente, houve a instalação manual das ferramentas de análise estática *Mythril* e *Slither*, e de análise dinâmica *Manticore*. Porém, observou-se a possibilidade de criar um ambiente de configuração de tais ferramentas para deixar a etapa automatizada.

Conforme ilustra a Figura 9, para a criação do ambiente de execução das ferramentas, que inclui sua instalação com suas dependências, foi criada uma imagem *Docker* (Código-fonte 3 no Apêndice 6.4) com as configurações necessárias para instalar os pacotes e as ferramentas de detecção de vulnerabilidades.

Figura 9 – Visão geral de execução das ferramentas com a imagem *Docker*.



Fonte: elaborada pelo autor.

Além disso, implementou-se um *script* de execução das ferramentas para automatizar a criação e configuração do ambiente de execução das ferramentas (Código-fonte 4 no Apêndice 6.4). No *script* criado, existe uma ordem de execução das ferramentas, primeiro são executadas as ferramentas de análise estática, e em seguida, a ferramenta de análise dinâmica.

O ambiente utilizado para os experimentos nos contratos consiste de uma máquina com sistema operacional Linux com a distribuição Ubuntu 22.04.3 LTS, processador *Core* i7, 12 *Gigabytes* (GB) de *Random Access Memory* (RAM), *Python* 3, *Pip* 3 e o compilador do *Solidity* na versão 8.

5.4 Etapa 4: Analisar os resultados da detecção das ferramentas

Esta seção apresenta os resultados organizados em função dos três experimentos descritos na Etapa 1.

5.4.1 Resultados para o Experimento 1

O resultado esperado para o primeiro experimento é que nenhuma das ferramentas detecte vulnerabilidades. No entanto, conforme sumariza a Tabela 8, as ferramentas *Slither* e *Manticore* detectaram, erroneamente, uma vulnerabilidade no contrato inteligente do experimento 1 (falso positivo), enquanto a ferramenta *Mythril*, corretamente, não apontou nenhuma vulnerabilidade de Reentrância.

Tabela 8 – Ferramentas que detectaram a vulnerabilidade de Reentrância no experimento 1.

Vulnerabilidade	***Mythril***	***Slither***	***Manticore***
Reentrância com função *transfer()*	-	X	X

Fonte: elaborada pelo autor.

A ferramenta *Mythril* não detectou a vulnerabilidade de Reentrância com o uso da função *transfer*. Essa não detecção da vulnerabilidade ocorre porque a ferramenta, internamente, é projetada para não considerar cenários vulneráveis com uso da função *transfer*. Isso se deve ao limite de gás (conceito apresentado na Seção 2.3) aplicado durante as transações, o qual impede que contratos maliciosos realizem muitas operações, incluindo chamadas subsequentes a funções do contrato analisado. Dessa forma, a vulnerabilidade de Reentrância torna-se inexistente para a ferramenta *Mythril*.

Por outro lado, a ferramenta *Slither* detectou a vulnerabilidade de Reentrância no mesmo contrato, pois não controla o comportamento específico relacionado ao consumo de gás nas transações realizadas durante os testes de detecção de vulnerabilidades.

A ferramenta *Manticore* também indicou, de forma equivocada, a presença da vulnerabilidade de Reentrância no contrato, gerando casos de teste que simulam diferentes cenários de ataque. Durante esses testes, foi possível explorar a suposta vulnerabilidade até esgotar o saldo da conta proprietária.

A discrepância na detecção da vulnerabilidade de Reentrância entre as ferramentas reforça a importância de se utilizar múltiplas abordagens e ferramentas de análise no processo de verificação da segurança de contratos inteligentes.

5.4.2 Resultados para o Experimento 2

O resultado esperado para o segundo experimento é a detecção de vulnerabilidades por todas as ferramentas. Ao analisar o contrato inteligente do experimento 2, a Tabela 9 mostra que as ferramentas *Mythril*, *Slither* e *Manticore* conseguiram detectar a vulnerabilidade de Reentrância.

Tabela 9 – Ferramentas que detectaram a vulnerabilidade de Reentrância no experimento 2.

Vulnerabilidade	***Mythril***	***Slither***	***Manticore***
Reentrância com função *call()*	X	X	X

Fonte: elaborada pelo autor.

A ferramenta *Mythril*, neste experimento, conseguiu detectar a vulnerabilidade de Reentrância, sendo capaz de analisar corretamente o fluxo de execução e identificar os possíveis riscos associados ao uso de *call()*.

Da mesma forma, a ferramenta *Slither* também identificou os potenciais riscos relacionados ao uso da função *call()* no contrato inteligente.

Já a ferramenta *Manticore*, por meio de sua abordagem de geração simbólica de testes, não apenas detectou a vulnerabilidade de Reentrância, como também gerou múltiplos casos de teste que confirmaram sua exploração, simulando cenários em que o saldo do contrato é completamente esgotado por um ataque de Reentrância.

Esses resultados reforçam a importância de utilizar uma combinação de ferramentas para uma análise de segurança mais robusta e abrangente, garantindo que vulnerabilidades sejam identificadas e mitigadas antes da implantação do contrato inteligente.

5.4.3 Resultados para o Experimento 3

Conforme detalhado na Seção 5.1, o terceiro experimento foi realizado com base em um *testbed* IoT, utilizando a aplicação ClimaCor. O resultado esperado para o experimento é analisar a capacidade de detecção de cada ferramenta de acordo com as vulnerabilidades pertencentes as categorias do conjunto de dados *SmartBugs Curated* (DURIEUX *et al.*, 2019). Assim, o resultado esperado para o terceiro experimento é a detecção de todas as vulnerabilidades por todas as ferramentas.

A Tabela 10 mostra a classificação das vulnerabilidades presentes no contrato *DeviceRegistry* (Código-fonte 6 no Apêndice 6.4), que foi implementado a partir de uma adaptação da implementação feita por (ŠIMUNIĆ, 2018). A maioria das vulnerabilidades é da categoria Controle de Acesso, exceto uma delas, o Limite de gás, que faz parte da categoria Negação de Serviço. As vulnerabilidades presentes na tabela foram fundamentadas na Seção 2.5.

Tabela 10 – Ferramentas que detectaram vulnerabilidades no experimento 3.

Vulnerabilidade	**Mythril**	**Slither**	**Manticore**	**Categoria do Contrato**
Acesso não autorizado	-	X	X	Controle de Acesso
Uso incorreto de *tx.origin*	-	X	-	Controle de Acesso
Uso inseguro de *delegatecall*	X	X	-	Controle de Acesso
Uso incorreto de *selfdestruct*	X	X	-	Controle de Acesso
Limite de gás	-	-	X	Negação de Serviço

Fonte: elaborada pelo autor.

A ferramenta *Mythril* foi capaz de identificar tanto o uso inseguro de *delegatecall* quanto o uso incorreto de *selfdestruct*. No entanto, não conseguiu detectar o uso inadequado de *tx.origin*, o que pode indicar uma limitação em sua análise de controle de acesso baseado em variáveis globais.

Por sua vez, a ferramenta *Slither* identificou todas as vulnerabilidades da categoria de Controle de Acesso, incluindo o uso incorreto de *delegatecall*, *selfdestruct*, *tx.origin* e o acesso não autorizado. Isso demonstra sua eficácia na análise de funções potencialmente inseguras relacionadas ao controle de acesso.

Já a ferramenta *Manticore*, com sua abordagem de execução simbólica, destacou-se na identificação da vulnerabilidade associada ao limite de gás. Além disso, também conseguiu detectar a vulnerabilidade de acesso não autorizado. A *Manticore* não apenas identificou essas falhas, como também simulou cenários em que o contrato falha devido ao esgotamento de gás.

Esses resultados reforçam a importância de usar uma combinação de ferramentas para realizar uma análise de segurança abrangente de contratos inteligentes. A ferramenta *Mythril* complementou a análise detectando certos padrões de uso inseguro. Já a ferramenta *Slither* se mostrou eficiente na detecção de vulnerabilidades relacionadas ao controle de acesso. A ferramenta *Manticore* identificou vulnerabilidades associadas ao gerenciamento de gás e controle de acesso. Observa-se que utilizando essas ferramentas em conjunto, é possível obter uma visão mais completa das potenciais falhas de segurança antes da implantação do contrato inteligente.

5.5 Etapa 5: Corrigir os erros encontrados no código-fonte dos contrato inteligente

Ao final dos experimentos com o contrato *DeviceRegistry* (Código-fonte 6 no Apêndice 6.4), o Desenvolvedor recebeu 3 arquivos com os resultados gerados por cada ferramenta separadamente, além de um arquivo contendo a combinação dos resultados das três ferramentas.

Após a implementação das correções pelo Desenvolvedor, o contrato inteligente retornou à Etapa 3, na qual foi submetido novamente às ferramentas *Mythril*, *Slither* e *Manticore*, com o objetivo de assegurar que todas as vulnerabilidades detectadas anteriormente foram devidamente corrigidas.

Este ciclo de teste e correção é repetido até que o contrato inteligente esteja livre de vulnerabilidades conhecidas, garantindo a segurança e a integridade do código antes de seu lançamento final na *Blockchain*.

6 CONCLUSÃO

Esta dissertação de mestrado apresentou o PERCI, um processo para verificar a segurança de contratos inteligentes para aplicações IoT, com foco na detecção de vulnerabilidades por meio da combinação de ferramentas de análise estática juntamente com análise dinâmica. Este processo foi definido com o objetivo de mitigar os riscos de segurança associados a contratos inteligentes, fornecendo uma abordagem mais robusta para a identificação de falhas de segurança.

O presente capítulo apresenta a conclusão deste trabalho, organizado da seguinte forma: A Seção 6.1 fornece uma visão geral do texto; a Seção 6.2 resume os resultados obtidos durante o desenvolvimento desta pesquisa; a Seção 6.3 discute as limitações do PERCI e, por fim, a Seção 6.4 encerra o capítulo com sugestões para trabalhos futuros.

6.1 Visão Geral

Esta dissertação de mestrado abordou a análise de segurança de contratos inteligentes, com ênfase na verificação de vulnerabilidades para mitigar falhas de segurança presentes no código. A pesquisa utilizou ferramentas para detecção de vulnerabilidades de contratos inteligentes, incluindo análises estática e dinâmica, para propor o PERCI, um processo de verificação de contratos inteligentes para aplicações IoT.

O trabalho enfatiza a importância da verificação rigorosa de vulnerabilidades para evitar falhas em aplicações IoT. A pesquisa resultou no PERCI, que combina análises estática e dinâmica para fortalecer a detecção de vulnerabilidades. Ferramentas como *Slither*, *Mythril* e *Manticore* foram utilizadas na validação do processo, demonstrando a eficiência do PERCI na identificação de vulnerabilidades em contratos inteligentes antes de sua implantação na *Blockchain*.

Para avaliar o PERCI, primeiro, demonstrou-se que a combinação das análises de cada ferramenta resultou em uma detecção mais eficiente de vulnerabilidades, proporcionando uma verificação mais abrangente do código dos contratos inteligentes. Além disso, este trabalho integrou um contrato inteligente para registrar e autenticar dispositivos na *Blockchain* a uma aplicação IoT que representa condições meteorológicas por meio de uma lâmpada inteligente que exibe cores. A avaliação do processo demonstrou a viabilidade do uso combinado das análises estática e dinâmica na detecção mais eficiente de vulnerabilidades. Por fim, espera-se que esta dissertação contribua para a melhoria da segurança das aplicações IoT que utilizam *Blockchain*.

6.2 Resultados

Esta seção aborda os principais resultados obtidos durante a realização desta pesquisa, como o PERCI e os trabalhos publicados durante esta pesquisa.

O PERCI é um processo de verificação de contratos inteligentes para aplicações IoT. O PERCI possui um conjunto de etapas de verificação de Contratos Inteligentes com o intuito de detectar vulnerabilidades utilizando a combinação de ferramentas de análises estática e dinâmica. A combinação dessas análises é proposta para melhorar a detecção de vulnerabilidades, proporcionando uma solução mais robusta. Verificou-se durante os experimentos realizados no Capítulo 5 que as abordagens definidas em cada etapa do Capítulo 4 foram importantes para o Analista de Segurança e o Desenvolvedor desempenharem seus papéis na utilização do PERCI.

Além do PERCI, a Tabela 11 apresenta os artigos submetidos e aceitos durante o desenvolvimento desta pesquisa. O primeiro artigo refere-se ao mapeamento sistemático realizado no início desta pesquisa, fundamentado nos preceitos metodológicos propostos por (KITCHENHAM; BRERETON, 2013). O segundo artigo, submetido e aceito no ADVANCEs 2024, apresenta as primeiras definições do PERCI. Por fim, o terceiro artigo, aceito na *IEEE Latin American Conference on Internet of Things* 2025, descreve os resultados do PERCI obtidos ao longo desta pesquisa.

Tabela 11 – Artigos publicados.

Título	**Autores**	**Conferência**
Blockchain Security in the Internet of Things: Literature Review	Joyce Quintino; Carina T. de Oliveira; Rossana M. C. Andrade	ADVANCEs 2023
Smart Contract verification process focusing on IoT applications	Joyce Quintino; Carina T. de Oliveira; Rossana M. C. Andrade	ADVANCEs 2024
PERCI: Smart Contract verification process for IoT applications	Joyce Quintino; Carina Oliveira; Rossana Andrade	*IEEE Latin American Conference on Internet of Things* 2025

Fonte: elaborada pelo autor.

6.3 Limitações

Entre as limitações identificadas, destaca-se o tempo de execução do processo PERCI, que pode ser elevado dependendo do tamanho e complexidade dos contratos analisados. Outra limitação importante a ser considerada é a dependência da qualidade das ferramentas de detecção

de vulnerabilidades utilizadas, sua eficiência final está atrelada à capacidade dessas ferramentas de identificar corretamente as vulnerabilidades.

6.4 Trabalhos Futuros

Esta seção aborda os principais pontos de evolução da presente pesquisa e também discute possíveis direcionamentos para trabalhos futuros. Como sugestão para pesquisas futuras, são propostos os seguintes direcionamentos:

- **Aplicação do PERCI em diferentes contextos**. Neste trabalho, utilizou-se a aplicação IoT ClimaCor para testar o contrato inteligente utilizado na avaliação do PERCI. Para testar a capacidade de generalização é interessante explorar a aplicação do PERCI em contextos além do IoT.
- **Criação de um guia para seleção de ferramentas de análise**. Durante o desenvolvimento do PERCI, foram utilizadas ferramentas de análise estática e dinâmica para a identificação de vulnerabilidades em contratos inteligentes. Para facilitar o uso dessas ferramentas em outros contextos, torna-se interessante desenvolver um guia prático que auxilie pesquisadores e desenvolvedores na seleção das ferramentas mais adequadas, considerando fatores como linguagem do contrato, desempenho e integração com diferentes plataformas de *Blockchain*. Tal guia poderia incluir estudos comparativos detalhados baseados em métricas como precisão, escalabilidade e facilidade de uso, fundamentados nos trabalhos de (DURIEUX *et al.*, 2019) e (FERREIRA *et al.*, 2020).
- **Desenvolvimento de mecanismo de validação de falsos-positivos**. Outra linha de investigação promissora é investigar o desenvolvimento de um mecanismo que auxilie na validação de falsos-positivos identificados pelo PERCI, melhorando a precisão dos resultados. Uma possibilidade é utilizar um conjunto de dados com falsos-positivos conhecidos para que seja possível consultar a precisão das detecções das vulnerabilidades. No uso de métricas para validação, os trabalhos de (PARIZI *et al.*, 2018) e (DURIEUX *et al.*, 2019) podem servir de base para a validação de falsos-positivos no PERCI.
- **Avaliação da escalabilidade do PERCI**. Recomenda-se avaliar a escalabilidade do processo de identificação de vulnerabilidades, aumentando a quantidade de contratos inteligentes analisados para verificar o desempenho do PERCI em cenários com maior volume de dados. Para isso, os conjuntos de dados citados nos trabalhos de (DURIEUX *et al.*, 2019), (FERREIRA *et al.*, 2020), (ANGELO; SALZER, 2023) e (ANGELO *et al.*, 2023)

podem auxiliar na realização dessa avaliação.

– **Integração com técnicas de Aprendizagem de Máquina**. É relevante também para esta pesquisa explorar a integração de técnicas de Aprendizagem de Máquina com as análises estática e dinâmica na etapa de execução das ferramentas do PERCI. Durante o desenvolvimento deste trabalho, observou-se que as ferramentas de detecção de vulnerabilidades baseiam-se em padrões previamente conhecidos, o que limita sua capacidade de identificar novas vulnerabilidades. Quando surge um novo padrão de vulnerabilidade, essas ferramentas podem apresentar limitações em sua detecção. Incorporar Aprendizagem de Máquina no PERCI pode resultar em uma abordagem mais robusta para garantir a segurança de contratos inteligentes nas plataformas de *Blockchain*. Os trabalhos de (XU *et al.*, 2021) e (KIANI; SHENG, 2024) podem ajudar nessa integração.

REFERÊNCIAS

ABIJAUDE, J.; GREVE, F.; SOBREIRA, P. Blockchain e contratos inteligentes para aplicações em iot, uma abordagem prática. In: **Jornada de Atualização em Informática 2021**. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação, 2021. p. 149–197. ISBN 9786587003573. Disponível em: https://books-sol.sbc.org.br/index.php/sbc/catalog/book/67.

Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2347–2376, 2015.

ALABA, F. A.; SULAIMON, H. A.; MARISA, M. I.; NAJEEM, O. Smart Contracts Security Application and Challenges: A Review. **Cloud Computing and Data Science**, v. 5, n. 1, p. 15–41, set. 2023. Accessed: 2024-08-06. Disponível em: https://ojs.wiserpub.com/index.php/CCDS/article/view/3271.

ALAJLAN, R.; ALHUMAM, N.; FRIKHA, M. Cybersecurity for blockchain-based iot systems: A review. **Applied Sciences**, v. 13, n. 13, 2023. ISSN 2076-3417. Disponível em: https://www.mdpi.com/2076-3417/13/13/7432.

ANGELO, M. D.; SALZER, G. Consolidation of ground truth sets for weakness detection in smart contracts. In: SPRINGER. **International Conference on Financial Cryptography and Data Security**. [*S. l.*], 2023. p. 439–455.

ANGELO, M. di; DURIEUX, T.; FERREIRA, J. F.; SALZER, G. **Evolution of Automated Weakness Detection in Ethereum Bytecode: a Comprehensive Study**. 2023. Disponível em: https://arxiv.org/abs/2303.10517.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. **Computer Networks**, Elsevier, v. 54, n. 15, p. 2787–2805, oct 2010. ISSN 13891286.

BEGUM, A.; TAREQ, A.; SULTANA, M.; SOHEL, M.; RAHMAN, T.; SARWAR, A. Blockchain attacks analysis and a model to solve double spending attack. **International Journal of Machine Learning and Computing**, v. 10, n. 2, p. 352–357, 2020.

BHUTTA, M. N. M.; KHWAJA, A. A.; NADEEM, A.; AHMAD, H. F.; KHAN, M. K.; HANIF, M. A.; SONG, H.; ALSHAMARI, M.; CAO, Y. A survey on blockchain technology: Evolution, architecture and security. **IEEE Access**, v. 9, p. 61048–61073, 2021.

CARDOSO, A. L.; ROTONDARO, B.; PENHA, L.; ENDLER, M.; CONCEIÇÃO, A. F. da; SILVA, F. J. da Silva e. Gerenciamento descentralizado de identidades para cidades inteligentes baseado na tecnologia blockchain. In: **Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação, 2022. p. 57–70. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/sbseg/article/view/21658.

CHEN, H.; PENDLETON, M.; NJILLA, L.; XU, S. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 3, jun 2020. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3391195.

CHEN, X.; HU, S.; LI, Y.; YUE, D.; DOU, C.; DING, L. Co-estimation of state and fdi attacks and attack compensation control for multi-area load frequency control systems under fdi and dos attacks. **IEEE Transactions on Smart Grid**, v. 13, n. 3, p. 2357–2368, 2022.

DEEP, A.; PERRUSQUíA, A.; ALJABURI, L.; AL-RUBAYE, S.; GUO, W. A novel distributed authentication of blockchain technology integration in iot services. **IEEE Access**, v. 12, p. 9550–9562, 2024.

DURIEUX, T.; FERREIRA, J. F.; ABREU, R.; CRUZ, P. Empirical review of automated analysis tools on 47.587 ethereum smart contracts. **CoRR**, abs/1910.10601, 2019. Disponível em: http://arxiv.org/abs/1910.10601.

ESQUIVEL, E.; CAMPOS, J.; MENDONÇA, R.; VIEIRA, A.; NACIF, J. Detecção de vulnerabilidades em contratos inteligentes utilizando arvore sintática abstrata. In: **Anais do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação, 2023. p. 335–348. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/sbseg/article/view/27217.

FEIST, J.; GREICO, G.; GROCE, A. Slither: a static analysis framework for smart contracts. In: **Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain**. IEEE Press, 2019. (WETSEB '19), p. 8–15. Disponível em: https://doi.org/10.1109/WETSEB.2019.00008.

FENG, P. Big data analysis of e-commerce based on the internet of things. In: **2019 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)**. [*S. l.*: *s. n.*], 2019. p. 345–347.

FERREIRA, J. F.; CRUZ, P.; DURIEUX, T.; ABREU, R. Smartbugs: A framework to analyze solidity smart contracts. In: **Proceedings of the 35th IEEE/ACM international conference on automated software engineering**. [*S. l.*: *s. n.*], 2020. p. 1349–1352.

GHALEB, B.; AL-DUBAI, A.; EKONOMOU, E.; QASEM, M.; ROMDHANI, I.; MACKENZIE, L. Addressing the dao insider attack in rpl's internet of things networks. **IEEE Communications Letters**, IEEE, v. 23, n. 1, p. 68–71, 2018.

GHOSH, A.; GUPTA, S.; DUA, A.; KUMAR, N. Security of cryptocurrencies in blockchain technology: State-of-art, challenges and future prospects. **Journal of Network and Computer Applications**, v. 163, 04 2020.

GOODRICH, M. T.; TAMASSIA, R. **Introdução à segurança de computadores**. [*S. l.*]: Bookman, 2013. ISBN 978-85-407-0192-2.

GUPTA, B. C.; SHUKLA, S. K. A study of inequality in the ethereum smart contract ecosystem. In: **2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)**. [*S. l.*: *s. n.*], 2019. p. 441–449.

HASAN, Q. O. M. **Machine Learning Based Framework for Smart Contract Vulnerability Detection in Ethereum Blockchain**. Tese (Doutorado) – Rochester Institute of Technology, 2023.

HE, D.; WU, R.; LI, X.; CHAN, S.; GUIZANI, M. Detection of vulnerabilities of blockchain smart contracts. **IEEE Internet of Things Journal**, v. 10, n. 14, p. 12178–12185, 2023.

IMPERIUS, N. P.; ALAHMAR, A. D. Systematic mapping of testing smart contracts for blockchain applications. **IEEE Access**, v. 10, p. 112845–112857, 2022.

INUWA, M. M.; DAS, R. A comparative analysis of various machine learning methods for anomaly detection in cyber attacks on iot networks. **Internet of Things**, v. 26, p. 101162, 2024. ISSN 2542-6605. Disponível em: https://www.sciencedirect.com/science/article/pii/S2542660524001033.

ISO/IEC 27000. **ISO/IEC 27000 - Information technology — Security techniques — Information security management systems — Overview and vocabulary**. [*S. l.*], 2014.

JIANG, B.; LIU, Y.; CHAN, W. K. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: **Proceedings of the 33rd ACM/IEEE international conference on automated software engineering**. New York, NY, USA: Association for Computing Machinery, 2018. (ASE '18), p. 259—-269. ISBN 9781450359375. Disponível em: https://doi.org/10.1145/3238147.3238177.

JÚNIOR, F.; AUGUSTO, T. A utilização do blockchain para internet das coisas: Um mapeamento sistemático. **Revista Contemporânea**, v. 3, n. 10, p. 18433–18448, Oct. 2023. Disponível em: https://ojs.revistacontemporanea.com/ojs/index.php/home/article/view/1793.

KHAN, Z. A.; NAMIN, A. S. A survey of vulnerability detection techniques by smart contract tools. **IEEE Access**, v. 12, p. 70870–70910, 2024.

KIANI, R.; SHENG, V. S. Ethereum smart contract vulnerability detection and machine learning-driven solutions: A systematic literature review. **Electronics**, MDPI, v. 13, n. 12, p. 2295, 2024.

KITCHENHAM, B.; BRERETON, P. A systematic review of systematic review process research in software engineering. **Information and Software Technology**, v. 55, n. 12, p. 2049–2075, 2013. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584913001560.

KUSHWAHA, S. S.; JOSHI, S.; SINGH, D.; KAUR, M.; LEE, H.-N. Ethereum smart contract analysis tools: A systematic review. **IEEE Access**, v. 10, p. 57037–57062, 2022.

LI, J. Metamorphic testing for smart contract vulnerabilities detection. **ArXiv**, abs/2303.03179, 2023. Disponível em: https://api.semanticscholar.org/CorpusID:257365957.

LI, K.; XUE, Y.; CHEN, S.; LIU, H.; SUN, K.; HU, M.; WANG, H.; LIU, Y.; CHEN, Y. Static application security testing (sast) tools for smart contracts: How far are we? **arXiv preprint arXiv:2404.18186**, 2024.

LI, X.; JIANG, P.; CHEN, T.; LUO, X.; WEN, Q. A survey on the security of blockchain systems. **Future Generation Computer Systems**, v. 107, p. 841–853, 2020. ISSN 0167-739X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167739X17318332.

LIAO, J.-W.; TSAI, T.-T.; HE, C.-K.; TIEN, C.-W. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In: IEEE. **2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)**. [*S. l.*], 2019. p. 458–465.

LIU, Z.; QIAN, P.; WANG, X.; ZHUANG, Y.; QIU, L.; WANG, X. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, 2021.

LIU, Z.; QIAN, P.; YANG, J.; LIU, L.; XU, X.; HE, Q.; ZHANG, X. Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting. **arXiv preprint arXiv:2301.03943**, 2023.

LONE, A. H.; NAAZ, R. Applicability of blockchain smart contracts in securing internet and iot: A systematic literature review. **Computer Science Review**, v. 39, p. 100360, 2021. ISSN 1574-0137. Disponível em: https://www.sciencedirect.com/science/article/pii/S1574013720304603.

LUU, L.; CHU, D.-H.; OLICKEL, H.; SAXENA, P.; HOBOR, A. Making Smart Contracts Smarter. In: **Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)**. New York, NY, USA: Association for Computing Machinery, 2016. p. 254–269. Disponível em: https://doi.org/10.1145/2976749.2978309.

MEI, X.; ASHRAF, I.; JIANG, B.; CHAN, W. A Fuzz Testing Service for Assuring Smart Contracts. In: **Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)**. [*S. l.*: *s. n.*], 2019. p. 544–545.

MERLINO, V.; ALLEGRA, D. Energy-based approach for attack detection in iot devices: A survey. **Internet of Things**, p. 101306, 2024. ISSN 2542-6605. Disponível em: https://www.sciencedirect.com/science/article/pii/S2542660524002476.

MINERVA, R.; BIRU, A.; ROTONDI, D. Towards a definition of the internet of things (iot). **IEEE Internet Initiative**, IEEE, v. 1, n. 1, p. 1–86, 2015.

MISHRA, N.; PANDYA, S. Internet of things applications, security challenges, attacks, intrusion detection, and future visions: A systematic review. **IEEE Access**, v. 9, p. 59353–59377, 2021.

MOSSBERG, M.; MANZANO, F.; HENNENFENT, E.; GROCE, A.; GRIECO, G.; FEIST, J.; BRUNSON, T.; DINABURG, A. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In: **2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [*S. l.*: *s. n.*], 2019. p. 1186–1189.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. 2008.

NIKOLIC, I.; KOLLURI, A.; SERGEY, I.; SAXENA, P.; HOBOR, A. Finding the greedy, prodigal, and suicidal contracts at scale. In: **Proceedings of the 34th annual computer security applications conference**. [*S. n.*], 2018. p. 653–663. Disponível em: http://arxiv.org/abs/1802.06038.

ONTIVERO, F. I. L. **Um estudo comparativo de ferramentas de auditoria em contratos inteligentes**. Dissertação (Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação)) – Universidade Federal de Uberlândia (UFU), Uberlândia, 2023. Disponível em: https://repositorio.ufu.br/handle/123456789/37299. Acesso em: 3 fev. 2023.

PARIZI, R. M.; DEHGHANTANHA, A.; CHOO, K. R.; SINGH, A. Empirical vulnerability analysis of automated smart contracts security testing on blockchains. **CoRR**, abs/1809.02702, 2018. Disponível em: http://arxiv.org/abs/1809.02702.

PATEL, K. K.; PATEL, S. M.; SCHOLAR, P. G. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. **International Journal of Engineering Science and Computing**, 2016. Disponível em: http://ijesc.org/.

PENG, K.; LI, M.; HUANG, H.; WANG, C.; WAN, S.; CHOO, K.-K. R. Security challenges and opportunities for smart contracts in internet of things: A survey. **IEEE Internet of Things Journal**, v. 8, n. 15, p. 12004–12020, 2021.

QIAN, P.; LIU, Z.; HE, Q.; HUANG, B.; TIAN, D.; WANG, X. Smart contract vulnerability detection technique: A survey. **arXiv preprint arXiv:2209.05872**, 2022.

REJEB, A.; REJEB, K.; APPOLLONI, A.; JAGTAP, S.; IRANMANESH, M.; ALGHAMDI, S.; ALHASAWI, Y.; KAYIKCI, Y. Unleashing the power of internet of things and blockchain: A comprehensive analysis and future directions. **Internet of Things and Cyber-Physical Systems**, v. 4, p. 1–18, 2024. ISSN 2667-3452. Disponível em: https://www.sciencedirect.com/science/article/pii/S2667345223000366.

SHARMA, N.; SHARMA, S. A survey of mythril, a smart contract security analysis tool for evm bytecode. v. 13, p. 51003–51010, 12 2022.

ŠIMUNIĆ, S. **Upotreba blockchain tehnologije za registraciju i upravljanje IoT uređajima**. Bachelor's Thesis – University of Rijeka, Faculty of Engineering, Department of Computer Engineering, Section of Software Engineering, Rijeka, Croatia, September 2018. Access restricted to students and staff of home institution. Disponível em: urn:nbn:hr:190:464395.

SINGH, A.; KUMAR, G.; SAHA, R.; CONTI, M.; ALAZAB, M.; THOMAS, R. A survey and taxonomy of consensus protocols for blockchains. **Journal of Systems Architecture**, v. 127, p. 102503, 2022. ISSN 1383-7621. Disponível em: https://www.sciencedirect.com/science/article/pii/S1383762122000777.

STALLINGS, W. **Criptografia e segurança de redes: princípios e práticas**. [*S. l.*]: Pearson, 2015. ISBN 978-85-430-1450-0.

SZABO, N. Formalizing and securing relationships on public networks. **First Monday**, v. 2, n. 9, Sep. 1997. Disponível em: https://firstmonday.org/ojs/index.php/fm/article/view/548.

TIKHOMIROV, S.; VOSKRESENSKAYA, E.; IVANITSKIY, I.; TAKHAVIEV, R.; MARCHENKO, E.; ALEXANDROV, Y. Smartcheck: Static analysis of ethereum smart contracts. In: **2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)**. [*S. l.*: *s. n.*], 2018. p. 9–16.

TORRES, C.; SCHüTTE, J.; STATE, R. **Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts**. 2018.

TORRES, C. F.; STEICHEN, M.; STATE, R. The art of the scam: demystifying honeypots in ethereum smart contracts. In: **Proceedings of the 28th USENIX Conference on Security Symposium**. USA: USENIX Association, 2019. (SEC'19), p. 1591–1607. ISBN 9781939133069.

TSANKOV, P.; DAN, A.; DRACHSLER-COHEN, D.; GERVAIS, A.; BüNZLI, F.; VECHEV, M. Securify: Practical security analysis of smart contracts. In: **Proceedings of the 2018 ACM**

SIGSAC Conference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery, 2018. (CCS '18), p. 67–82. ISBN 9781450356930. Disponível em: https://doi.org/10.1145/3243734.3243780.

WAHEED, N.; HE, X.; IKRAM, M.; USMAN, M.; HASHMI, S. S.; USMAN, M. Security and privacy in iot using machine learning and blockchain: Threats and countermeasures. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 6, dec 2020. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3417987.

WOHRER, M.; ZDUN, U. Design patterns for smart contracts in the ethereum ecosystem. In: **2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)**. [*S. l.*: *s. n.*], 2018. p. 1513–1520.

XU, Y.; HU, G.; YOU, L.; CAO, C. A novel machine learning-based analysis model for smart contract vulnerability. **Security and Communication Networks**, Wiley Online Library, v. 2021, n. 1, p. 5798033, 2021.

YAO, C.; LIU, Y.; WEI, X.; WANG, G.; GAO, F. Backscatter technologies and the future of internet of things: Challenges and opportunities. **Intelligent and Converged Networks**, v. 1, n. 2, p. 170–180, 2020.

ZHUANG, Y.; LIU, Z.; QIAN, P.; LIU, Q.; WANG, X.; HE, Q. Smart contract vulnerability detection using graph neural network. In: BESSIERE, C. (Ed.). **Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20**. International Joint Conferences on Artificial Intelligence Organization, 2020. p. 3283–3290. Main track. Disponível em: https://doi.org/10.24963/ijcai.2020/454.

APÊNDICE A - CÓDIGO-FONTE DO PERCI

Este apêndice apresenta o código-fonte utilizado para desenvolver o Processo de Verificação de Contratos Inteligentes (Código-fonte 3 e Código-fonte 4) e também para a sua avaliação (Código-fonte 5 e Código-fonte 6). Todos os códigos estão disponíveis no seguinte repositório: https://github.com/JoyceQuintino/masterResearch/tree/staging/tools

Código-fonte 3 – Dockerfile para criar a imagem com as configurações das ferramentas de detecção de vulnerabilidades de contratos inteligentes:

```
# syntax=docker/dockerfile:1.6

###
### Smart Contract Analyser Toolbox
###
FROM ubuntu:jammy AS toolbox

# Add common tools
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install
    -y --no-install-recommends \
     bash-completion \
     curl \
     git \
     build-essential \
     python3-dev \
     python3-pip \
     python3-venv \
     sudo \
     vim \
     unzip \
     wget \
     jq \
     && rm -rf /var/lib/apt/lists/*

# Install Slither
RUN pip3 install --no-cache-dir slither-analyzer

# Install Mythril with its necessary dependencies
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install
```

```
    -y --no-install-recommends \
29      libssl-dev \
30      zlib1g-dev \
31      && rm -rf /var/lib/apt/lists/*
32
33  # Install Mythril
34  RUN pip3 install --no-cache-dir mythril
35
36  # Install Node.js and npm
37  RUN curl -fsSL https://deb.nodesource.com/setup_16.x | bash -
38  RUN apt-get install -y nodejs
39
40  # Install protobuf version
41  RUN pip3 install protobuf==3.20.3
42
43  # Instalar Manticore
44  RUN pip3 install manticore
45  RUN pip3 install --upgrade manticore
46
47  # Add a user with passwordless sudo
48  RUN useradd -m ethsec && \
49      usermod -aG sudo ethsec && \
50      echo 'ethsec ALL=(ALL) NOPASSWD: ALL' >> /etc/sudoers
51
52  ##### user-level setup follows
53  ##### Things should be installed in $HOME from now on
54  USER ethsec
55  WORKDIR /home/ethsec
56  ENV HOME="/home/ethsec"
57  ENV PATH="${PATH}:${HOME}/.local/bin"
58
59  # Install python tools for user
60  RUN pip3 install --no-cache-dir --user pyevmasm solc-select crytic-
    compile
61
62  # Install one solc release from each branch and select the latest
    version as the default
63  RUN solc-select install 0.4.26 0.5.17 0.6.12 0.7.6 latest && solc-
    select use latest
```

```
64
65 # Clone useful repositories
66 RUN git clone --depth 1 https://github.com/crytic/building-secure-
       contracts.git
67
68 # Configure MOTD
69 COPY --chown=root:root motd /etc/motd
70 RUN echo 'cat /etc/motd' >> ~/.bashrc
71
72 # Create and set permissions for /share directory
73 USER root
74 RUN mkdir /share && chown -R ethsec:ethsec /share && chmod -R 777 /
       share
75
76 USER ethsec
77
78 # Verify installations
79 RUN slither --version && \
80     myth version && \
81     manticore --version
82
83 CMD ["/bin/bash"]
```

Código-fonte 4 – Script bash para executar e exportar os resultados das ferramentas de detecção de vulnerabilidades de contratos inteligentes:

```
1  #!/bin/bash
2
3  # Define the working directory
4  WORK_DIR="/share"
5
6  # Check that the argument has been provided
7  if [ $# -eq 0 ]; then
8      echo "Usage: $0 <contract_filename>"
9      exit 1
10 fi
11
12 # The first argument is the name of the contract file
```

```
13 CONTRACT_FILENAME="$1"
14 CONTRACT_PATH="${WORK_DIR}/${CONTRACT_FILENAME}"
15
16 # Check if the file exists
17 if [ ! -f "$CONTRACT_PATH" ]; then
18     echo "File '$CONTRACT_FILENAME' not found in $WORK_DIR"
19     exit 1
20 fi
21
22 # Remove previous output files
23 rm -f ${WORK_DIR}/slither_report.json
24 rm -f ${WORK_DIR}/mythril_report.json
25 rm -f ${WORK_DIR}/combined_report.json
26
27 # Run Slither to analyse and check that the JSON file is valid
28 slither ${CONTRACT_PATH} --json ${WORK_DIR}/slither_report.json
29
30 # Check if Slither found any issues
31 if [ $(jq length ${WORK_DIR}/slither_report.json) -eq 0 ]; then
32     echo "The analysis was completed successfully. No issues were
        detected by Slither."
33     exit 0
34 fi
35
36 # Validate Slither JSON output
37 if ! jq empty ${WORK_DIR}/slither_report.json > /dev/null 2>&1; then
38     echo "Error: JSON generated by Slither is invalid or has not been
         generated correctly."
39     cat ${WORK_DIR}/slither_report.json
40     exit 1
41 fi
42
43 # Run Mythril to analyse and check that the JSON file is valid
44 myth analyze ${CONTRACT_PATH} -o jsonv2 > ${WORK_DIR}/mythril_report.
    json
45
46 # Check if Mythril found any issues
47 if [ ! -s "${WORK_DIR}/mythril_report.json" ]; then
48     echo "The analysis was completed successfully. No issues were
```

```
        detected by Mythril."
49      echo '{"issues": []}' > ${WORK_DIR}/mythril_report.json
50  fi
51
52  # Validate Mythril JSON output
53  if ! jq empty ${WORK_DIR}/mythril_report.json > /dev/null 2>&1; then
54      echo "Error: JSON generated by Mythril is invalid or has not been
            generated correctly."
55      echo '{"issues": []}' > ${WORK_DIR}/mythril_report.json
56      cat ${WORK_DIR}/mythril_report.json
57      exit 1
58  fi
59
60  jq -s '{
61      slither: (
62          if .[0].results.detectors | length > 0 then
63              .[0].results.detectors | map(. as $slitherIssue | {
64                  check: $slitherIssue.check,
65                  description: $slitherIssue.description,
66                  elements: $slitherIssue.elements,
67                  sourceMap: $slitherIssue.sourceMap,
68              })
69          else
70              "No issues were detected by Slither."
71          end
72      ),
73
74      mythril: (
75          if .[1] | length > 0 then
76              .[1] | map(. as $mythrilIssue | {
77      code: $mythrilIssue.code,
78                  description: $mythrilIssue.description,
79      function: $mythrilIssue.function,
80                  severity: $mythrilIssue.severity,
81                  sourceMap: $mythrilIssue.sourceMap,
82      max_gas_used: $mythrilIssue.max_gas_used,
83            min_gas_used: $mythrilIssue.min_gas_used,
84              })
85          else
```

```
            "No issues were detected by Mythril."
        end
    )
}' ${WORK_DIR}/slither_report.json ${WORK_DIR}/mythril_report.json >
    ${WORK_DIR}/combined_report.json

echo "Combined report created at ${WORK_DIR}/combined_report.json"
```

Código-fonte 5 – Contrato Inteligente para cadastrar e autenticar dispositivos IoT na Blockchain:

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DeviceRegistry {
    struct Device {
        //Structure to store device information
        string owner;
        bool registered;
    }

    //Mapping device IDs to your information
    mapping(string => Device) private devices;

    //Event to emit when a device is registered
    event DeviceRegistered(string deviceId);

    //Function for registering a new device
    function registerDevice(string memory deviceId, string memory
        owner) public {
        require(!devices[deviceId].registered, "Device is already
            registered");

        devices[deviceId] = Device({
            owner: owner,
            registered: true
        });

        emit DeviceRegistered(deviceId);
```

```
27      }
28
29      //Function to check the authentication of a device
30      function authenticateDevice(string memory deviceId, string memory
            owner) public view returns (bool) {
31          //Add this in arguments function: bytes32 messageHash, bytes
                memory signature
32
33          require(devices[deviceId].registered, "Device is not
                registered");
34
35          //Checks that the recovered address is the same as that of
                the device owner
36          return
37              keccak256(abi.encodePacked(devices[deviceId].owner)) ==
38              keccak256(abi.encodePacked(owner));
39      }
40
41      //Function to check if a device is registered
42      function isDeviceRegistered(string memory deviceId) public view
            returns (bool) {
43          return devices[deviceId].registered;
44      }
45  }
```

Código-fonte 6 – Contrato Inteligente para cadastrar e autenticar dispositivos IoT na Blockchain com vulnerabilidades:

```
1   //SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.0;
3
4   contract DeviceRegistry {
5       struct Device {
6           // Structure to store device information
7           string owner;
8           bool registered;
9       }
10
```

```
11      // Mapping device IDs to your information
12      mapping(string => Device) private devices;
13      address private owner;
14
15      // Event to emit when a device is registered
16      event DeviceRegistered(string deviceId);
17
18      constructor() {
19          owner = msg.sender;
20      }
21
22      // Function for registering a new device
23      function registerDevice(string memory deviceId, string memory
            ownerName) public {
24          // Vulnerability: No access control
25          require(!devices[deviceId].registered, "Device is already
                registered");
26
27          devices[deviceId] = Device({
28              owner: ownerName,
29              registered: true
30          });
31
32          emit DeviceRegistered(deviceId);
33      }
34
35      // Function to check the authentication of a device
36      function authenticateDevice(string memory deviceId, string memory
            ownerName) public view returns (bool) {
37          // Vulnerability: Using tx.origin for access control
38          require(tx.origin == owner, "Not authorized");
39          require(devices[deviceId].registered, "Device is not
                registered");
40
41          return
42              keccak256(abi.encodePacked(devices[deviceId].owner)) ==
43              keccak256(abi.encodePacked(ownerName));
44      }
45
```

```
46      // Vulnerability: Unprotected critical function
47      function changeOwner(address newOwner) public {
48          // Vulnerability: No access control on this function
49          owner = newOwner;
50      }
51
52      // Vulnerability: Potential for denial of service
53      address[] public addressList;
54
55      function addAddresses(uint numberOfAddresses) public {
56          // Vulnerability: Gas limit issue
57          for (uint i = 0; i < numberOfAddresses; i++) {
58              addressList.push(msg.sender);
59          }
60      }
61
62      function clearAddresses() public {
63          // Vulnerability: Gas limit issue
64          delete addressList;
65      }
66
67      // Vulnerability: Unsafe delegatecall
68      function forwardCall(address callee, bytes memory _data) public {
69          (bool success, ) = callee.delegatecall(_data);
70          require(success, "Delegatecall failed");
71      }
72
73      // Vulnerability: Selfdestruct function accessible to anyone
74      function destroyContract() public {
75          // Vulnerability: No access control, can be called by anyone
76          selfdestruct(payable(msg.sender));
77      }
78
79      // Function to check if a device is registered
80      function isDeviceRegistered(string memory deviceId) public view
            returns (bool) {
81          return devices[deviceId].registered;
82      }
83  }
```

APÊNDICE B - VÍDEO EXPLICATIVO: ATAQUE AO CONTROLE DE ACESSO DO CONTRATO INTELIGENTE

O vídeo ilustra a simulação de um ataque controlado ao contrato inteligente responsável pelo registro de dispositivos na *Blockchain* e encontra-se disponível no seguinte repositório: https://github.com/JoyceQuintino/ClimaCor_Back/tree/vulnerability_test/video. O ataque explora uma vulnerabilidade no controle de acesso devido ao uso inadequado do parâmetro *tx.origin*.