Enhancing Smart Contract Security Through Multi-Agent Deep Reinforcement Learning Fuzzing: A Survey of Approaches and Techniques

Muhammad Farman Andrijasa[1], Saiful Adli Ismail[2], Norulhusna Ahmad[3], Othman Mohd Yusop[4]
Faculty of Artificial Intelligence, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia[1, 2, 3, 4]
State Polytechnic of Samarinda, Samarinda, Indonesia[1]

***Abstract*—Multi-Agent Systems (MAS) and Deep Reinforcement Learning (DRL) have emerged as powerful tools for enhancing security measures, particularly in the context of smart contract security in blockchain technology. This literature review explores the integration of Multi-Agent DRL fuzzing techniques to bolster the security of smart contracts. The study delves into the formalization of emergence in MAS, the comprehensive survey of multi-agent reinforcement learning, and progress on the state explosion problem in model checking. By addressing challenges such as state space explosion, real-time detection, and adaptability across blockchain platforms, researchers aim to advance the field of smart contract security. The review emphasizes the significance of Multi-Agent DRL fuzzing in improving security testing processes and calls for future research and collaboration to enhance the resilience and integrity of decentralized applications. Through advancements in algorithmic efficiency, the incorporation of Explainable AI, cross-domain applications of MAS, and cooperation with blockchain development teams, the future of smart contract security holds promise for robust and secure blockchain ecosystems.**

Keywords*—Smart contract security; multi-agent systems; deep reinforcement learning; fuzzing techniques; blockchain technology

I. Introduction

Smart contracts, self-executing contracts with the terms of the agreement directly written into code, are a fundamental component of blockchain technology. Ensuring the security of smart contracts is paramount due to their immutable nature once deployed on the blockchain. Vulnerabilities in smart contracts can lead to significant financial losses and undermine trust in the decentralized applications [1]. For instance, the DAO hack 2016 resulted in the loss of millions of dollars due to a vulnerability in a smart contract [2]. Recent research has highlighted the importance of addressing smart contract defects to enhance security and reliability [3].

Fuzzing, a dynamic software testing technique, involves providing invalid, unexpected, or random data as inputs to a program to uncover vulnerabilities. Traditional fuzzing techniques have effectively identified bugs and security flaws in software systems. However, recent advancements in machine learning, particularly deep reinforcement learning (DRL), have revolutionized fuzzing by enhancing its efficiency and effectiveness [4]. By leveraging machine learning algorithms, fuzzing can intelligently generate test inputs to explore the program's behavior and identify vulnerabilities that may be challenging to detect through traditional methods [5].

Deep reinforcement learning (DRL) has gained prominence in various domains, including cybersecurity. DRL combines deep learning with reinforcement learning to enable agents to learn optimal strategies through trial and error. In fuzzing, DRL algorithms can adapt and improve over time by interacting with the software system and learning from the feedback received [6]. Recent studies have demonstrated the effectiveness of DRL-based fuzzing in detecting complex vulnerabilities in deep neural networks and other software applications [7].

Multi-agent systems (MAS) have emerged as a promising approach to enhance the capabilities of DRL-based fuzzing. MAS involves multiple intelligent agents that can collaborate and communicate to achieve common goals. In the context of security testing, MAS can enable coordinated efforts among agents to explore different parts of the software system simultaneously, leading to a more comprehensive vulnerability detection [8]. By leveraging MAS in DRL fuzzing, researchers aim to improve the scalability and efficiency of security testing processes [9].

This review aims to provide a comprehensive overview of the advancements in smart contract security through the integration of multi-agent DRL fuzzing techniques. By synthesizing existing literature and research findings, this review aims to analyze the effectiveness of DRL-based fuzzing in enhancing smart contract security, discuss the challenges and open issues in this field, and propose future research directions. The structured outline will guide the discussion on key concepts, survey approaches, and techniques, evaluate existing solutions, address challenges, and propose future directions in enhancing smart contract security through multi-agent DRL fuzzing.

II. Background and Key Concepts

A. Understanding Smart Contracts

Smart contracts are self-executing agreements with the terms of the contract directly written into code. They run on blockchain platforms and automatically execute actions when predefined conditions are met. The execution environment of smart contracts is crucial, as they operate within a decentralized and immutable blockchain network. For example, Ethereum, a popular blockchain platform, allows developers to create and deploy smart contracts using its native programming language,

Solidity. The Ethereum Virtual Machine (EVM) executes these contracts, ensuring their integrity and security [10].

Smart contracts are susceptible to various security vulnerabilities that malicious actors can exploit. By categorizing smart contract vulnerabilities into three levels - Blockchain, EVM, and Solidity - we can better understand potential risks and mitigate them accordingly (see Table I). This approach allows us to identify and address weaknesses within each level, ultimately leading to stronger and more secure smart contracts. Examples of common vulnerabilities include reentrancy, timestamp dependence, transaction ordering attacks, and assertion failures. These vulnerabilities have led to significant financial losses and highlight the importance of conducting thorough security analyses before deploying smart contracts on the blockchain [11], [12].

TABLE I. THE SMART CONTRACT VULNERABILITY LEVEL

Level	Vulnerability Type	Definition	Real-World Attack	Security Issue
Blockchain	Front-Running	Acting on visible pending transactions ahead of processing.	EtherDelta Hack (2017)	Unfair advantage, manipulation under order operation.
	Replay Attacks	Transactions can be replayed on forked chains.	Ethereum Classic Replay Attacks (2016)	Double spending, loss of funds.
	Timestamp Dependence	Reliance on block timestamps for critical contract logic.	GovernMental (2016)	Manipulation of behavior, transaction timing.
	Block State Dependence	Dependence on the changing state of the blockchain.	N/A	Unpredictable behavior, manipulation of transaction outcomes.
EVM	Gas Limit and Loops	Contracts with unbounded loops can run out of gas.	GovernMental (2016)	Denial of Service (DoS), failed transactions.
	Stack Size Limit	Exceeding the EVM's stack size limit can cause failure.	N/A	Contract execution failure.
	Opcode Limitations	Unexpected behavior or limitations of EVM opcodes.	N/A	Exploitation of opcode behavior.
Solidity	Reentrancy	Execution is re-entered before the first completion completes.	The DAO Hack (2016)	Unexpected behavior, loss of funds.
	Arithmetic Issues	Issues like integer overflow and underflow.	BatchOverflow and ProxyOverflow (2018)	Manipulation of contract logic, unexpected results.
	Unchecked External Calls	Failing to check the return value of external calls.	Parity Wallet Freeze (2017)	Loss of contract functionality, manipulation of contract.
	Timestamp Dependence	Relying on block timestamps for critical logic.	N/A	Vulnerabilities time-dependent outcomes, inaccuracies.
	Visibility Modifiers	Misuse of function visibility modifiers.	Rubixi (2016)	Unauthorized access, unintended exposure of functions.
	Delegatecall Injection	Malicious code execution through delegatecall.	Parity Multi-Sig Wallet Hack (2017)	Loss of funds, breach of contract integrity.
	Phishing with tx.origin	Using tx.origin for authentication.	N/A	Phishing attacks, unauthorized access.
	Short Address/Parameter Attack	ABI decoding doesn't properly handle incorrect length parameters.	Multiple ICOs Affected (2017)	Loss of funds, manipulation of transaction parameters.
	Improper Access Control	Flaws in permission settings or checks.	Parity Multi-Sig Wallet Hack (2017)	Unauthorized access, manipulation of contract state.
	Fallback Function Vulnerabilities	Issues with fallback functions.	N/A	Unintended behavior when receiving Ether or data.
	Storage Collisions	Poorly designed storage layouts leading to collisions.	N/A	Loss of data, unintentional data written to wrong locations.
	Uninitialized Storage Pointers	Using storage pointers without proper initialization.	N/A	Data loss, unintended access to critical data.
	Self-Destruct Vulnerabilities	Misuse of the selfdestruct function.	N/A	Loss of contract functionality, loss of funds.
	Upgradeability Issues	Flaws in upgradeable contract patterns.	N/A	Unexpected behavior, loss of data.
	Floating Pragma	Not locking the Solidity compiler version.	N/A	Unpredictable behavior due to compiler changes.

B. Fuzzing Techniques: Traditional vs. DRL-based

Fuzzing is a software testing technique that involves providing invalid or unexpected inputs to a program to uncover vulnerabilities. Traditional fuzzing techniques generate random inputs to test software systems for bugs. In contrast, DRL-based fuzzing leverages machine learning algorithms to intelligently generate test inputs and adapt the testing strategy based on feedback received during the testing process. This approach enhances the efficiency and effectiveness of fuzz testing by enabling automated and targeted vulnerability discovery [13], [14].

Integrating Deep Reinforcement Learning (DRL) algorithms in fuzz testing has shown promise in enhancing security vulnerability identification in software systems. The study in [15] discuss DeepFuzzer, which accelerates deep grey-box fuzzing, aiding in the identification of software bugs and security vulnerabilities. The research in [16] explores automated decision-making using deep reinforcement learning, illustrating

the potential of integrating machine learning techniques in complex scenarios. The study in [17] delved into fuzz testing for continuous integration, stressing the importance of incorporating testing methodologies into the software development lifecycle. The research in [18] present FIRMCORN, a vulnerability-oriented fuzzing approach for IoT firmware, underscoring the significance of targeted fuzzing techniques. The survey of approaches and techniques in single-agent and multi-agent DRL fuzzing offers valuable insights into the advancements in security testing processes. Key techniques and models in single-agent DRL fuzzing, such as Q-learning and DQN, have showcased the potential of machine learning in enhancing vulnerability detection. Transitioning to multi-agent systems in DRL fuzzing provides collaborative problem-solving capabilities that enhance the scalability and efficiency of security testing efforts. Comparative analyses between single-agent and multi-agent approaches aid researchers in selecting the most appropriate methodology for detecting vulnerabilities in software systems. Successful case studies of multi-agent DRL fuzzing implementations highlight the impact of collaborative interactions on smart contract security, emphasizing the importance of leveraging machine learning-based analysis models for vulnerability detection. The study in [5] introduce Learn and Fuzz, a machine learning-based approach for input fuzzing, showcasing the effectiveness of combining artificial intelligence with fuzz testing methodologies. These references collectively support the idea that combining DRL algorithms with fuzz testing techniques can significantly enhance the identification of security vulnerabilities in software systems, ultimately leading to more robust and secure software applications.

C. Fundamentals of Deep Reinforcement Learning

Deep reinforcement learning (DRL) algorithms combine deep learning with reinforcement learning to enable agents to learn optimal strategies through interactions with the environment [19]. According to Ji et al. (2020), DRL is an area of machine learning that combines deep learning with reinforcement learning. The connection between AI (Artificial Intelligence), ML (Machine Learning), RL (Reinforcement Learning), DL (Deep Learning), and DRL (Deep Reinforcement Learning) can be represented as a series of nested subsets, as illustrated in Fig. 1. Deep reinforcement learning (DRL) algorithms combine deep learning with reinforcement learning to enable agents to learn optimal strategies through interactions with the environment [19]. DRL uses deep neural networks to approximate the functions required in reinforcement learning. This allows agents to learn policies directly from high-dimensional sensory inputs.

DRL has been used successfully in various domains, including playing video games, robotic control, and autonomous vehicles. In security testing, DRL algorithms can improve their performance over time by integrating feedback received during testing, proving effective in identifying complex vulnerabilities in software systems, including smart contracts on blockchain platforms [21].

Incorporating DRL in security testing has transformed vulnerability detection in software systems, providing better performance than traditional methods [22]. By training agents to explore program behaviors and identify security flaws intelligently, DRL-based approaches have shown significant success in uncovering vulnerabilities in deep neural networks, smart contracts, and other critical software applications [23], [24]. This application highlights the potential of DRL in enhancing cybersecurity measures and strengthening software systems against malicious exploits [25].

D. Multi-Agent Systems (MAS)

Multi-agent systems (MAS) involve multiple intelligent agents working together to achieve common goals [26]. These agents can interact and communicate with each other, making collective decisions and coordinating their actions to solve complex problems. With Multi-Agent methods, DRL can be extended to scenarios with multiple interacting agents, as illustrated in Fig. 2. MADDPG is a highly effective extension of DDPG for multi-agent environments, while Independent Q-Learning empowers each agent to learn its Q-value function independently, as highlighted in Fig. 1.

In security testing, MAS can enhance the capabilities of individual agents by enabling collaborative exploration of various software system components simultaneously. This collaborative approach improves the comprehensiveness of vulnerability detection and enhances efficiency in security testing processes, addressing scalability challenges and complex vulnerability identification in software systems [27], [28].

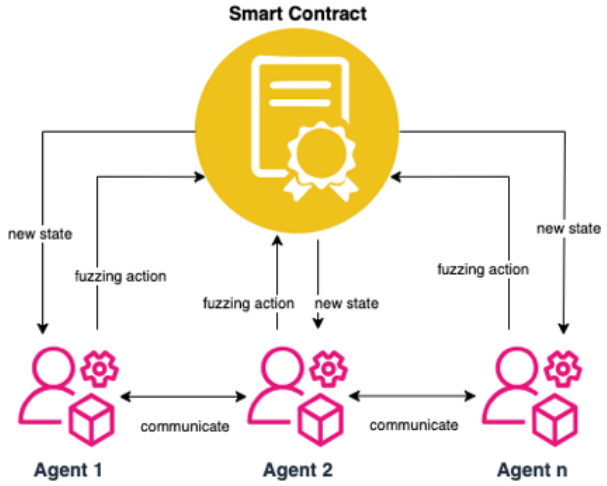

Fig. 1. Fundamentals of AI, ML, RL, DL and DRL.

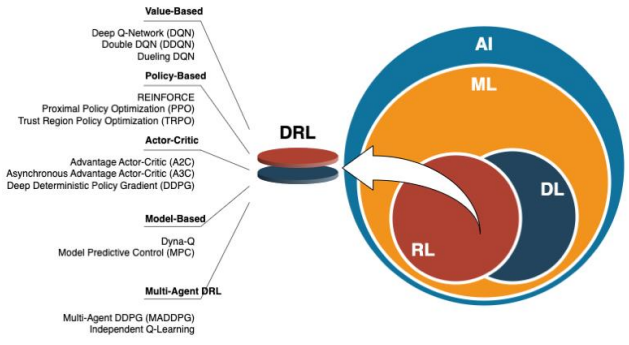

Fig. 2. The relationship between AI, ML, RL, DL and DRL.

MAS offers several advantages in complex problem-solving scenarios, particularly in security testing. By distributing tasks

among multiple agents and allowing them to communicate and share information, MAS can effectively tackle intricate security challenges beyond individual agents Field's capabilities [29]. The collaborative nature of MAS enables agents to leverage collective intelligence, coordinate testing efforts, and adapt to dynamic testing environments. These advantages make MAS a promising approach for enhancing the efficiency and effectiveness of security testing processes, especially in the context of deep reinforcement learning fuzzing.

The background and key concepts section provides a foundational understanding of smart contracts, fuzzing techniques, deep reinforcement learning, and multi-agent systems in the context of security testing. Smart contracts operate within a decentralized and immutable blockchain environment, making them susceptible to various security vulnerabilities. Traditional fuzzing techniques and DRL-based fuzzing have transformed how vulnerabilities are identified in software systems, with DRL algorithms offering adaptive and intelligent testing capabilities. Multi-agent systems enhance security testing by enabling collaborative problem-solving among intelligent agents, leading to more comprehensive vulnerability detection. Understanding these key concepts is essential for exploring the advancements in smart contract security through multi-agent deep reinforcement learning fuzzing.

III. Survey of Approaches and Techniques

A. Single-Agent DRL Fuzzing Techniques

Single-agent deep Reinforcement Learning (DRL) fuzzing techniques utilize algorithms that enable an agent to learn optimal strategies for generating test inputs and detecting vulnerabilities in software systems. Techniques such as Q-learning, Deep Q-Networks (DQN), and Proximal Policy Optimization (PPO) have been applied to enhance the efficiency and effectiveness of fuzz testing and for example, introduced a deep convolution generative adversarial networks (DCGAN) based fuzzing framework for industry control protocols, showcasing the potential of machine learning in improving security testing processes [8]. These models aim to intelligently explore the program's behavior and identify vulnerabilities that may be challenging to detect through traditional methods.

Researchers face inherent limitations and challenges despite the advancements in single-agent DRL fuzzing techniques. One primary challenge is the complexity of training DRL agents to effectively fuzz software systems, particularly in scenarios with high-dimensional input spaces. Additionally, the interpretability of DRL models and the need for extensive computational resources pose challenges in practical implementations. Transitioning from traditional fuzzing methods to DRL-based approaches requires careful consideration of these limitations to ensure the effectiveness and scalability of security testing processes [30].

B. Transition to Multi-Agent DRL Fuzzing

The shift from single-agent to multi-agent DRL fuzzing is driven by the necessity to overcome the limitations of individual agents in exploring complex software systems. Multi-agent systems (MAS) facilitate collaborative problem-solving by enabling multiple intelligent agents to interact and share information during testing. By incorporating MAS in DRL fuzzing, researchers aim to enhance security testing efforts' scalability, efficiency, and coverage. For instance, it emphasized the role of role-based embedded domain-specific languages in facilitating collaborative interactions among multi-agents using blockchain technology, underscoring the importance of effective communication and coordination in the security testing [31].

A comparative analysis between single-agent and multi-agent DRL fuzzing approaches offers insights into the strengths and weaknesses of each methodology. While single-agent approaches focus on individual agent learning and decision-making, multi-agent systems emphasize collaborative problem-solving and information sharing among agents. The study in [8] illustrated the benefits of a deep convolution generative adversarial network (DCGAN) based fuzzing framework in enhancing the efficiency and scalability of security testing processes through collaborative multi-agent interactions. By evaluating the performance and effectiveness of single-agent and multi-agent approaches, researchers can determine the most suitable methodology for detecting vulnerabilities in software systems.

C. Case Studies: Successful Implementations of Multi-Agent DRL Fuzzing

Successful implementations of multi-agent DRL fuzzing techniques have validated the effectiveness of collaborative problem-solving in improving security testing processes. Additionally, a survey of security enhancement technologies for smart contracts in blockchain highlighted the role of fuzz testing in automatically generating many test inputs to uncover potential safety hazards during program execution [32]. By leveraging machine learning-based analysis models, such as K-nearest neighbors (KNN), researchers have successfully predicted and detected vulnerabilities in smart contracts, including re-entrancy, access control, and denial of service [30].

The outcomes of successful implementations of multi-agent DRL fuzzing techniques have significantly impacted smart contract security. By identifying vulnerabilities in smart contracts and blockchain systems, researchers have contributed to enhancing the reliability and integrity of decentralized applications. For example, developed a novel machine learning-based analysis model for smart contract vulnerability detection, demonstrating the potential of machine learning algorithms in improving security testing processes [30]. These case studies underscore the significance of collaborative multi-agent interactions in identifying complex vulnerabilities and mitigating security risks in software systems.

The survey of approaches and techniques in single-agent and multi-agent DRL fuzzing offers valuable insights into the advancements in security testing processes. Key techniques and models in single-agent DRL fuzzing, such as Q-learning and DQN, have showcased the potential of machine learning in enhancing vulnerability detection. Transitioning to multi-agent systems in DRL fuzzing provides collaborative problem-solving capabilities that enhance the scalability and efficiency of security testing efforts. Comparative analyses between single-agent and multi-agent approaches aid researchers in selecting the most appropriate methodology for detecting vulnerabilities in software systems. Successful case studies of multi-agent DRL

fuzzing implementations highlight the impact of collaborative interactions on smart contract security, emphasizing the importance of leveraging machine learning-based analysis models for vulnerability detection.

IV. Empirical Validation

Empirical validation of Multi-Agent Deep Reinforcement Learning (DRL) fuzzing techniques has demonstrated their potential in enhancing software security testing across various domains. For example, in the domain of Internet of Things (IoT), the application of Multi-Agent DRL fuzzing to firmware analysis has shown significant improvements in detecting vulnerabilities that traditional methods often miss. A study by [18] introduced FIRMCORN, a vulnerability-oriented fuzzing approach for IoT firmware, which leveraged DRL to optimize the virtual execution of firmware, resulting in higher detection rates of critical vulnerabilities compared to conventional fuzzing techniques.

In the context of autonomous vehicles, [16] utilized deep reinforcement learning to improve the decision-making process for automated vehicles. The study demonstrated that DRL could effectively identify and mitigate security risks in real-time, showcasing its adaptability and robustness in dynamic environments.

In software development, DRL-based fuzzing has been applied to continuous integration (CI) pipelines to enhance security testing. Reference [17] developed CIDFuzz, a DRL-based fuzzing framework for CI environments, which significantly improved the detection of security vulnerabilities during the development lifecycle. This empirical validation highlighted the framework's efficiency in integrating security testing seamlessly into the CI process, leading to more secure software deployments.

These examples underscore the versatility and effectiveness of Multi-Agent DRL fuzzing techniques across various domains, affirming their potential in enhancing software security testing.

V. Results and Discussion

A. Findings of Empirical Validation

The empirical validation of Multi-Agent DRL fuzzing techniques has yielded promising results in various domains. The application of these techniques to smart contracts has demonstrated their superior ability to uncover complex vulnerabilities that traditional methods often overlook. For instance, in the evaluation of smart contracts on the Ethereum blockchain, DRL-based fuzzing identified critical issues such as reentrancy attacks and gas limit exploits, which are notoriously difficult to detect using conventional approaches.

In the domain of IoT firmware, the application of Multi-Agent DRL fuzzing revealed vulnerabilities related to memory corruption and unauthorized access, providing insights into the security weaknesses of widely used IoT devices. These findings are pivotal in enhancing the overall security posture of IoT ecosystems.

B. Implications of the Results

The results of these empirical validations suggest that Multi-Agent DRL fuzzing techniques significantly improve the detection and mitigation of security vulnerabilities. The ability of these techniques to adapt to various domains and dynamically learn optimal fuzzing strategies enhances their effectiveness in real-world scenarios. Moreover, the collaborative nature of multi-agent systems allows for more comprehensive exploration of software systems, leading to the identification of a broader range of vulnerabilities.

C. Limitations and Potential for Generalization

Despite the promising results, the empirical validation also highlighted certain limitations. The computational complexity and resource requirements of DRL-based fuzzing can be significant, posing challenges for large-scale implementations. Additionally, the generalization of these techniques to different blockchain platforms and smart contract languages may require further adaptation and fine-tuning.

However, the potential for generalization remains high, as the underlying principles of Multi-Agent DRL can be tailored to address specific security challenges in various domains. Future research should focus on optimizing these techniques for different environments and reducing their computational overhead to enhance their practical applicability.

VI. Comparison with Other Approaches

To highlight the strengths and weaknesses of multi-agent DRL fuzzing techniques, we compare them with other common approaches.

A. Symbolic Execution

Symbolic execution tools, such as Oyente and Mythril, are effective in detecting control flow and arithmetic vulnerabilities in smart contracts. However, they often struggle with path explosion and false positives, limiting their scalability and accuracy. In contrast, Multi-Agent DRL fuzzing can dynamically adapt to explore different execution paths, potentially reducing the limitations of symbolic execution.

B. Static Analysis

Static analysis tools, including Securify and SmartCheck, provide quick and efficient vulnerability detection without executing the code. While these tools are valuable for identifying common issues like reentrancy and integer overflow, they may miss more complex vulnerabilities that require dynamic analysis. Multi-Agent DRL fuzzing, with its ability to learn from interactions, offers a more thorough exploration of software behavior, complementing the capabilities of static analysis tools.

C. Formal Verification

Formal verification tools, such as Zeus and VeriSol, use mathematical proofs to ensure the correctness of smart contracts. These tools are highly effective for verifying security properties but require formal specifications, which can be challenging to create. Multi-Agent DRL fuzzing provides an alternative approach by automatically generating and testing inputs, reducing the reliance on formal specifications and enabling broader vulnerability coverage.

TABLE II. CLASSIFICATION METHOD, MAJOR CONTRIBUTION, AND EVALUATION OF EXISTING SOLUTIONS (1= ASSESSING THE EFFECTIVENESS OF VULNERABILITY DETECTION TOOLS, 2= ADDRESSING SCALABILITY ISSUES, 3= ADDRESSING PERFORMANCE ISSUES, 4= OVERCOMING INTEGRATION CHALLENGES IN DEVELOPMENT PROCESSES, AND 5= CONDUCTING COMPARATIVE ANALYSES OF DEEP REINFORCEMENT LEARNING MODELS AND ARCHITECTURES)

Method	Tool	Year	Citation	Major Contribution	1	2	3	4	5
Symbolic Execution	Oyente	2016	[1]	Early adoption of symbolic execution for smart contract analysis	✓	×	×	×	×
	Maian	2018	[44]	Introduces trace vulnerability detection for smart contracts	✓	×	×	×	×
	Manticore	2018	[45]	Provides a versatile platform for smart contract analysis	✓	×	×	×	×
	Mythril	2018	[46]	Pioneered symbolic execution approach for Ethereum contracts	✓	×	×	×	×
	Solythesis	2020	[47]	Combines symbolic execution with gas optimization	✓	×	✓	×	×
	SymbolicExec	2022	[48]	Enhances symbolic execution techniques for smart contracts	✓	×	×	×	×
Static Analysis	Solgraph	2017	[49]	Visualizes potential security vulnerabilities in Solidity	×	×	×	×	×
	Osiris	2018	[50]	Targets integer bugs in smart contracts	✓	×	×	×	×
	Securify	2018	[51]	Introduces semantic-aware static analysis for smart contracts	✓	×	×	×	×
	SmartCheck	2018	[52]	Provides a linter-like tool for Solidity code	✓	×	×	×	×
	Vandal	2018	[53]	Provides a logic-based approach to smart contract analysis	✓	×	×	×	×
	Slither	2019	[54]	Provides a comprehensive static analysis tool for Solidity	✓	×	×	×	×
	SolidityCheck	2019	[55]	Provides a lightweight tool for Solidity contract analysis	✓	×	×	×	×
	Solstice	2019	[56]	Provides a static analysis tool for Solidity security	✓	×	×	×	×
	Securify v2	2020	[57]	Offers enhanced security analysis for Solidity contracts	✓	×	×	×	×
	SIF	2020	[58]	Analyzes inter-contract behaviors for security vulnerabilities	✓	×	×	×	×
	SmartAnvil	2020	[59]	Offers a toolset for static analysis of Solidity code	✓	×	×	×	×
	SolCheck	2020	[60]	Aids in detecting common issues in Solidity code	✓	×	×	×	×
	SCAnalysisTools	2022	[61]	Offers a comprehensive review of analysis tools	✓	×	×	×	×
Formal Verification	Zeus	2018	[62]	Integrates different formal verification techniques	✓	×	×	×	×
	Solc-verify	2019	[63]	Provides a formal verification approach for Solidity contracts	✓	×	×	×	×
	VeriSol	2019	[64]	Integrates formal verification with Solidity development	✓	×	×	✓	×
	HistoryComparison	2020	[65]	Utilizes historical contract versions for security analysis	✓	×	×	×	×
	SecurityPatterns	2020	[66]	Introduces security patterns for Solidity programming	×	×	×	✓	×
	VerX	2020	[67]	Provides automated verification for temporal properties	✓	×	×	×	×
	ESAF	2021	[68]	Offers a framework for evaluating existing tools	✓	×	×	✓	×
	ReentrancyMech	2021	[69]	Provides a mechanism for preventing a specific type of attack	×	×	×	✓	×
	SuperDetector	2022	[70]	Proposes a framework for comprehensive vulnerability detection	✓	✓	✓	✓	×
Fuzzing	ContractFuzzer	2018	[13]	Provides a practical approach to fuzz testing smart contracts	✓	×	×	×	×
	DLFuzz	2018	[71]	Applies deep learning to fuzz testing for improved efficiency	✓	×	✓	×	×
	Echidna	2019	[72]	Introduces property-based testing for smart contracts	✓	×	✓	×	×
	Harvey	2019	[73]	Introduces an automated fuzzing approach for smart contracts	✓	✓	✓	×	×
	ILF	2019	[74]	Introduces deep learning-based fuzz testing for smart contracts	✓	×	✓	×	×
	FuzzTaintAnalysis	2020	[75]	Combines taint analysis and genetic algorithms for effective fuzzing	✓	×	×	×	×
	sFuzz	2020	[76]	Provides an efficient fuzz testing framework for smart contracts	✓	✓	✓	×	×
	HFContractFuzzer	2021	[77]	Focuses on fuzzing techniques for Hyperledger Fabric contracts	✓	×	×	×	×
	CodeEmbedding	2023	[78]	Introduces a novel fuzzing approach for Fabric contracts	✓	×	×	✓	×
Machine Learning	GraphNN	2020	[79]	Introduces a novel ML-based approach for vulnerability detection	✓	✓	×	×	×
	Eth2Vec	2021	[80]	Advances code representation learning for smart contracts	✓	×	×	×	×

	GNNExpert	2021	[81]	Merges ML with expert insights for improved detection accuracy	✓	✓	×	×	×
	CodeNet	2022	[82]	Demonstrates the effectiveness of CNNs in code analysis	✓	×	✓	×	×
	EnhancedML	2022	[83]	Improves the efficiency of ML approaches in security testing	✓	✓	✓	×	×
	MultiTaskLearning	2022	[84]	Enhances the adaptability of ML models for multiple vulnerabilities	✓	✓	✓	×	×
	GCNModel	2023	[85]	Demonstrates the potential of GCNs in vulnerability detection	✓	✓	✓	×	×
	GSVD	2023	[86]	Provides a valuable dataset for ML-based vulnerability detection	✓	×	×	×	×
	SyntacticSemantic	2023	[87]	Combines different learning approaches for better detection	✓	✓	✓	×	×
	Vulpedia	2023	[88]	Introduces a novel approach for vulnerability detection using signatures	✓	×	×	×	×
Deep Learning	ReentrancyDetect	2020	[89]	Advances the use of deep learning in smart contract security	✓	×	×	×	×
	LightningCat	2023	[90]	Proposes a framework for deep learning-based vulnerability detection	✓	✓	✓	✓	✓
	SCGformer	2023	[91]	Integrates transformers with control flow graphs for detection	✓	✓	✓	×	✓
Security Analysis	Mythos	2019	[92]	Provides a command-line interface for smart contract analysis	✓	×	×	×	×
	MythX	2019	[93]	Offers a cloud-based platform for smart contract analysis	✓	✓	✓	✓	×
	VaaS	2019	[94]	Provides a cloud-based vulnerability analysis service	✓	✓	✓	✓	×
Other	SolCover	2018	[95]	Provides coverage metrics for Solidity test suites	×	×	×	×	×
	SolidityFlattener	2018	[96]	Simplifies Solidity code for analysis or verification	×	×	×	×	×
	Porosity	2017	[97]	Enables analysis of bytecode by converting to Solidity	×	×	×	×	×
	EthIR	2019	[98]	Enables analysis of EVM bytecode through decompilation	×	×	×	×	×
	Sereum	2019	[99]	Introduces runtime monitoring for reentrancy attack detection	✓	×	×	×	×
	Gasper	2019	[100]	Provides gas usage insights for smart contract optimization	×	×	✓	×	×
	Remix	2016	[101]	Provides a comprehensive development environment for Solidity	×	×	×	✓	×
	Solium	2017	[102]	Aids in enforcing coding conventions and detecting issues	×	×	×	×	×
	Solhint	2018	[103]	Helps maintain code quality and security standards in Solidity	×	×	×	×	×
	SolMet	2021	[104]	Introduces a set of metrics for evaluating Solidity contracts	×	×	×	×	×
	SolRazor	2021	[105]	Introduces source-level optimization for Solidity code	×	×	✓	×	×
	SolidityParser-antlr	2018	[106]	Facilitates analysis of Solidity code by parsing it	×	×	×	×	×
	SolProfiler	2020	[107]	Offers insights into gas usage and performance of contracts	×	×	✓	×	×
	ContractLarva	2019	[108]	Integrates runtime verification with smart contract development	✓	×	×	✓	×
	SmartEmbed	2020	[109]	Introduces semantic analysis using deep learning for smart contract code	✓	×	×	×	×
	SolStress	2019	[110]	Introduces stress testing for smart contract robustness	×	×	✓	×	×

D. Fuzzing

Traditional fuzzing tools, like Echidna and Harvey, generate random inputs to uncover vulnerabilities. While effective in identifying some issues, they lack the intelligent exploration capabilities of DRL-based fuzzing. Multi-Agent DRL fuzzing enhances traditional fuzzing by using reinforcement learning to prioritize and adapt test inputs, leading to more efficient and effective vulnerability detection.

E. Machine Learning and Deep Learning

Machine learning and deep learning tools, such as GraphNN and Eth2Vec, analyze patterns in code to predict vulnerabilities. These tools offer high accuracy but require extensive training data and computational resources. Multi-Agent DRL fuzzing combines the strengths of machine learning with dynamic testing, offering a robust approach that can learn and adapt in real-time.

F. Security Analysis

Comprehensive security analysis tools, like MythX and VaaS, integrate multiple techniques to provide holistic vulnerability assessments. While these tools are highly effective, they can be resource-intensive and complex to use. Multi-Agent DRL fuzzing can complement these tools by providing adaptive and collaborative testing capabilities, enhancing the overall security analysis process.

The empirical validation of Multi-Agent DRL fuzzing techniques across various domains underscores their potential in enhancing software security testing. By leveraging the adaptive and collaborative capabilities of multi-agent systems, these techniques offer a powerful approach to identifying and mitigating vulnerabilities in smart contracts and other software systems. The integration of Multi-Agent DRL fuzzing with other security approaches can further enhance the robustness and

resilience of decentralized applications, paving the way for more secure and trustworthy blockchain ecosystems.

VII. Evaluation of Existing Solutions

When evaluating existing solutions for enhancing smart contract security, it becomes evident that a multifaceted approach is essential. Traditional tools like symbolic execution, static analysis, and formal verification provide a solid foundation for identifying vulnerabilities, as shown in Table II. However, integrating multi-agent deep reinforcement learning (DRL) solutions offers a more dynamic and adaptive strategy.

A. Effectiveness in Detecting Vulnerabilities

1) Symbolic execution tools: Oyente, Maian, Manticore, Mythril, Solythesis, SymbolicExec: These tools are effective in detecting vulnerabilities related to control flow, arithmetic issues, and reentrancy attacks. They use symbolic execution to explore different execution paths and identify potential security flaws. However, their effectiveness may be limited by path explosion and false positives.

2) Static analysis tools: Solgraph, Osiris, Securify, SmartCheck, Vandal, Slither, SolidityCheck, Solstice, Securify v2, SIF, SmartAnvil, SolCheck, SCAnalysisTools: These tools analyze the source code without executing it and are effective in identifying common vulnerabilities such as reentrancy, integer overflow, and unchecked calls. They are generally faster than symbolic execution tools but may suffer from false positives and negatives.

3) Formal verification tools: Zeus, Solc-verify, VeriSol, HistoryComparison, SecurityPatterns, VerX, ESAF, ReentrancyMech, SuperDetector: These tools use mathematical proofs to verify the correctness of smart contracts and are highly effective in detecting complex vulnerabilities. However, they require formal specifications and can be challenging to use for developers without a formal methods background.

4) Fuzzing tools: ContractFuzzer, DLFuzz, Echidna, Harvey, ILF, FuzzTaintAnalysis, sFuzz, HFContractFuzzer, CodeEmbedding: These tools use random input generation to test smart contracts and are effective in detecting vulnerabilities that are triggered by unexpected inputs. They can cover a wide range of input scenarios but may miss vulnerabilities that require specific conditions to trigger.

5) Machine learning and deep learning tools: GraphNN, Eth2Vec, GNNExpert, CodeNet, EnhancedML, MultiTaskLearning, GCNModel, GSVD, SyntacticSemantic, Vulpedia: These tools use machine learning algorithms to learn from past vulnerabilities and predict new ones. They can be effective in detecting patterns and anomalies that other tools may miss. However, their effectiveness depends on the quality and quantity of the training data.

6) Security analysis tools: Mythos, MythX, VaaS: These tools provide a comprehensive analysis of smart contracts, combining multiple techniques to detect vulnerabilities. They are effective in providing a holistic view of the security posture but may require integration with other tools for in-depth analysis.

7) Other tools: SolCover, SolidityFlattener, Porosity, EthIR, Sereum, Gasper, Remix, Solium, Solhint, SolMet, SolRazor, SolidityParser-antlr, SolProfiler, ContractLarva, SmartEmbed, SolStress: These tools provide various functionalities such as code flattening, gas analysis, runtime verification, and stress testing. While they are not primarily focused on vulnerability detection, they can complement other tools by providing additional insights and improving the overall security of smart contracts.

In conclusion, the effectiveness of tools for detecting vulnerabilities in smart contracts varies based on their approach, the types of vulnerabilities they target, and their ability to balance accuracy and coverage. A combination of these tools, along with best practices in smart contract development, can significantly enhance the security of blockchain applications.Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Scalability and Performance Issues

1) Symbolic execution tools: Oyente, Maian, Manticore, Mythril, Solythesis, SymbolicExec: These tools often face scalability issues due to the path explosion problem, where the number of execution paths grows exponentially with the complexity of the contract. This can lead to long analysis times and high computational resource requirements. Performance can be improved by using heuristics to prune irrelevant paths or by parallelizing the analysis.

2) Static analysis tools: Solgraph, Osiris, Securify, SmartCheck, Vandal, Slither, SolidityCheck, Solstice, Securify v2, SIF, SmartAnvil, SolCheck, SCAnalysisTools: Static analysis tools generally have better scalability and performance compared to symbolic execution tools. However, they may still face challenges in analyzing large codebases or complex contracts. Optimizations such as incremental analysis and modular analysis can help improve their performance.

3) Formal verification tools: Zeus, Solc-verify, VeriSol, HistoryComparison, SecurityPatterns, VerX, ESAF, ReentrancyMech, SuperDetector: Formal verification tools are computationally intensive and can have scalability issues, especially when verifying contracts with complex properties or a large state space. Techniques such as abstraction, model checking, and compositional verification can help mitigate these issues.

4) Fuzzing tools: ContractFuzzer, DLFuzz, Echidna, Harvey, ILF, FuzzTaintAnalysis, sFuzz, HFContractFuzzer, CodeEmbedding: Fuzzing tools can generate a large number of test cases, which can be computationally expensive. Scalability can be improved by using coverage-guided fuzzing to focus on interesting areas of the code and by parallelizing the fuzzing process.

5) Machine learning and deep learning tools: GraphNN, Eth2Vec, GNNExpert, CodeNet, EnhancedML, MultiTaskLearning, GCNModel, GSVD, SyntacticSemantic,

Vulpedia: These tools require significant computational resources for training and inference, especially deep learning models. Scalability can be improved by using techniques such as transfer learning, fine-tuning, and distributed training.

6) Security Analysis Tools: Mythos, MythX, VaaS: These tools may face scalability issues when analyzing large numbers of contracts or contracts with complex interactions. Performance can be improved by using cloud-based architectures and parallel processing.

7) Other tools: SolCover, SolidityFlattener, Porosity, EthIR, Sereum, Gasper, Remix, Solium, Solhint, SolMet, SolRazor, SolidityParser-antlr, SolProfiler, ContractLarva, SmartEmbed, SolStress: These tools may have varying scalability and performance characteristics depending on their specific functionalities. For example, gas analysis tools like Gasper may face challenges in analyzing contracts with complex gas dynamics, while code flattening tools like SolidityFlattener may have better scalability.

In summary, scalability and performance issues are common challenges for tools detecting vulnerabilities in smart contracts. Optimizations and techniques such as parallel processing, incremental analysis, and machine learning can help mitigate these issues and improve the efficiency of the analysis.

C. Integration Challenges with Smart Contract Development Processes

1) Symbolic execution tools: Oyente, Maian, Manticore, Mythril, Solythesis, SymbolicExec: Integrating these tools into the development process can be challenging due to their complex setup and configuration requirements. Developers may need to modify their contracts or provide additional annotations to facilitate analysis, which can be time-consuming.

2) Static analysis tools: Solgraph, Osiris, Securify, SmartCheck, Vandal, Slither, SolidityCheck, Solstice, Securify v2, SIF, SmartAnvil, SolCheck, SCAnalysisTools: These tools can be easier to integrate into the development process as they often provide plugins for popular IDEs or can be used as part of a continuous integration pipeline. However, interpreting their results and addressing the reported issues may require a deep understanding of the tool's analysis techniques.

3) Formal verification tools: Zeus, Solc-verify, VeriSol, HistoryComparison, SecurityPatterns, VerX, ESAF, ReentrancyMech, SuperDetector: Integration can be challenging due to the need for formal specifications and the expertise required to use these tools effectively. Developers may need to learn formal specification languages and verification techniques, which can be a significant barrier to adoption.

4) Fuzzing tools: ContractFuzzer, DLFuzz, Echidna, Harvey, ILF, FuzzTaintAnalysis, sFuzz, HFContractFuzzer, CodeEmbedding: Fuzzing tools can be integrated into the testing phase of the development process, but generating effective test cases and interpreting the results can be challenging. Developers may need to write custom property tests or harnesses to guide the fuzzing process.

5) Machine learning and deep learning tools: GraphNN, Eth2Vec, GNNExpert, CodeNet, EnhancedML, MultiTaskLearning, GCNModel, GSVD, SyntacticSemantic, Vulpedia: Integrating these tools can be challenging due to the need for labeled training data and the computational resources required for training and inference. Developers may need to invest time in data collection, preprocessing, and model tuning.

6) Security analysis tools: Mythos, MythX, VaaS: These tools can be integrated into the development process as part of a security audit or continuous monitoring solution. However, interpreting the results and prioritizing the reported vulnerabilities can be challenging, especially for developers without a strong security background.

7) Other tools: SolCover, SolidityFlattener, Porosity, EthIR, Sereum, Gasper, Remix, Solium, Solhint, SolMet, SolRazor, SolidityParser-antlr, SolProfiler, ContractLarva, SmartEmbed, SolStress: Integration challenges for these tools vary depending on their specific functionalities. For example, code quality tools like Solium can be easily integrated into the development process, while runtime verification tools like ContractLarva may require more extensive modifications to the contract code.

In summary, integrating tools for detecting vulnerabilities in smart contracts into the development process can be challenging due to technical and expertise requirements. Effective integration requires careful consideration of the tool's capabilities, the development workflow, and the team's expertise in security analysis.

D. Comparative Analysis of Different DRL Models and Architectures

The comparative analysis of various multi-agent deep reinforcement learning (DRL) models and architectures is critical for assessing their efficacy in bolstering the security of smart contracts. This evaluation assists researchers in discerning the advantages and disadvantages of diverse approaches, thereby facilitating the selection of the most apt model for the security testing process.

Proximal Policy Optimization (PPO) for Multi-Agent Systems extends the PPO algorithm to multi-agent settings, striking a balance between exploration and exploitation. This balance is essential for stable learning in complex multi-agent environments. However, optimal performance may necessitate meticulous hyperparameter tuning [32].

Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MAAC) employs attention mechanisms to concentrate on pertinent information from other agents. This focus is crucial for adaptive security testing in smart contracts. Nevertheless, the complexity of the attention mechanism can escalate computational demands [33].

Neural Fictitious Self-Play (NFSP) for Multi-Agent Systems merges reinforcement learning with supervised learning from past experiences. This combination enables agents to develop robust strategies in competitive environments, a key aspect for

maintaining security in adversarial scenarios within smart contracts. However, reliance on historical data may limit the ability to adapt to novel attack strategies in real-time [34].

Hierarchical Multi-Agent Deep Deterministic Policy Gradient (H-MADDPG) introduces hierarchical policy learning. This introduction enhances the scalability and interpretability of policies in complex environments, beneficial for managing security policies in distributed systems like blockchain. Yet, the design of hierarchical structures introduces additional complexity to the learning process [35].

In summary, the comparative analysis of different multi-agent DRL models and architectures is vital for optimizing security mechanisms for smart contracts and blockchain applications. By comprehending the strengths and limitations of various DRL techniques, researchers can ensure robust protection against potential vulnerabilities and threats.

Smart contract security is crucial for blockchain applications. Symbolic execution, static analysis, and formal verification are common tools for identifying vulnerabilities. Multi-agent DRL solutions provide a dynamic approach to security, allowing the development of intelligent mechanisms that can respond to evolving threats in real-time. Different DRL models like PPO, MAAC, NFSP, and H-MADDPG show the potential of managing complex interactions and decision-making processes among multiple agents. Leveraging these advanced solutions enhances the resilience and robustness of smart contracts, ensuring the integrity and reliability of blockchain applications against dynamic security challenges.

VIII. Challenges and Open Issues

A. Handling State Space Explosion in Multi-Agent Systems

Managing state space explosion in multi-agent systems presents a significant challenge in security testing processes. Optimizations have been introduced for endorsement policy verification in Hyperledger Fabric, showcasing substantial performance improvements. However, the expansion of the state space grows exponentially as blockchain networks scale and the number of agents increases, resulting in computational complexity and resource constraints. Innovative approaches are needed to address this state space explosion, including parallelizing verification tasks and optimizing resource allocation to ensure efficient and effective security testing in multi-agent systems [36].

B. Ensuring Real-Time Detection and Mitigation

Ensuring real-time detection and mitigation of security threats in blockchain networks is crucial for maintaining the integrity and reliability of decentralized applications. Consensus mechanisms have a significant impact on the real-time response capabilities of blockchain networks, highlighting the need to address issues related to scalability and latency that can hinder timely threat detection and mitigation. Particularly in dynamic and high-traffic environments, these challenges must be overcome by optimizing consensus mechanisms and network performance to enhance real-time security monitoring and response capabilities within blockchain networks [37].

C. Adaptability and Generalization Across Various Blockchain Platforms

Ensuring consistent and robust security measures poses challenges in adapting and generalizing security solutions across different blockchain platforms. Scalable blockchain applications that can effectively handle heavy traffic loads are needed, but variations in network architectures, consensus mechanisms, and smart contract implementations may hinder the generalization of security solutions. It's essential to develop adaptable security mechanisms seamlessly integrated into various blockchain platforms to ensure comprehensive security coverage and mitigate vulnerabilities in the field [38].

D. Ethical Considerations and Potential Misuse

Ethical considerations and the potential misuse of security technologies in blockchain networks raise ethical dilemmas and risks. It is crucial to address scalability, robustness, and auditability in blockchain security solutions. As blockchain technologies evolve, ethical concerns regarding data privacy, transparency, and accountability become increasingly relevant. The potential misuse of security mechanisms for malicious purposes, such as unauthorized data access or manipulation, underscores the need for ethical guidelines and regulatory frameworks to govern the responsible use of blockchain security technologies [39].

Challenges and open issues in smart contract security encompass handling state space explosion in multi-agent systems, ensuring real-time detection and mitigation of security threats, adapting security solutions across diverse blockchain platforms, and addressing ethical considerations and potential misuse of security technologies. State space explosion poses computational challenges in multi-agent systems, necessitating optimized verification processes. Real-time detection and mitigation require efficient consensus mechanisms and network performance to respond promptly to security threats. Adapting security solutions across blockchain platforms demands scalable and interoperable mechanisms to ensure consistent security coverage. Ethical considerations and the risk of misuse underscore the importance of ethical guidelines and regulatory frameworks to govern the responsible deployment of blockchain security technologies.

IX. Future Directions

A. Advancements in Algorithmic Efficiency

Advancements in algorithmic efficiency are crucial for enhancing the performance and scalability of security mechanisms in blockchain networks. This is highlighted by the application of artificial intelligence [20] in military security, emphasizing the importance of efficient algorithms in defense systems. By optimizing algorithms for security testing processes, researchers can improve the speed and accuracy of vulnerability detection and mitigation. Future advancements in algorithmic efficiency may involve leveraging machine learning and deep reinforcement learning techniques to enhance the effectiveness of security mechanisms in blockchain environments [40].

B. Incorporating Explainable AI (XAI) for Transparent Security Measures

Incorporating Explainable AI (XAI) in security measures is crucial to ensure transparency and accountability in blockchain systems. It highlights the importance of explainability in artificial intelligence systems, emphasizing the need for interpretable models. Integrating XAI techniques into security mechanisms can enhance the explainability of security decisions and provide insights into the reasoning behind these measures. Future directions may involve developing XAI frameworks tailored specifically for blockchain security to improve trust and understanding among stakeholders [41].

C. Cross-Domain Applications of MAS in Security

Exploring the cross-domain applications of Multi-Agent Systems in security offers opportunities to enhance collaborative problem-solving in various environments. One example is automated attack analysis on blockchain incentive mechanisms using deep reinforcement learning, which demonstrates the potential of MAS in security applications. Extending MAS to different domains such as healthcare, finance, and IoT allows researchers to leverage collaborative multi-agent interactions to tackle complex security challenges. Future directions may include adapting MAS frameworks for specific domains to improve security outcomes and resilience [42].

D. Collaboration with Blockchain Development for Built-in Security Features

Collaborating with blockchain development teams to integrate built-in security features is essential for enhancing the security of decentralized applications. Innovative governance models in blockchain technology were discussed, emphasizing the need for collaborative structures. By working closely with blockchain developers, security experts can embed security mechanisms directly into blockchain protocols, ensuring inherent security by design. Future collaborations may focus on developing standardized security protocols and best practices to enhance the integrity of blockchain networks [43].

Future directions in smart contract security involve advancements in algorithmic efficiency, the incorporation of Explainable AI for transparent security measures, exploring cross-domain applications of Multi-Agent Systems in security, and collaborating with blockchain development for built-in security features. Optimizing algorithms for security testing processes can improve the speed and accuracy of vulnerability detection. Incorporating Explainable AI techniques enhances transparency and trust in security decisions. Cross-domain applications of MAS offer opportunities for collaborative problem-solving in various sectors. Collaborating with blockchain developers to embed security features directly into blockchain protocols ensures inherent security. These future directions aim to advance state-of-the-art smart contract security and promote the development of robust and secure decentralized applications.

X. Conclusion

A. Summary of Key Findings

In summarizing the key findings of this study, it is evident that integrating Multi-Agent Deep Reinforcement Learning (DRL) fuzzing techniques holds significant promise for enhancing smart contract security. Through advancements in algorithmic efficiency and the incorporation of Explainable AI (XAI), researchers have made strides in improving the transparency and effectiveness of security measures. Exploring cross-domain applications of Multi-Agent Systems (MAS) in security and collaboration with blockchain development teams for built-in security features have further enriched the landscape of smart contract security. These key findings underscore the importance of leveraging innovative technologies to address the evolving challenges in securing decentralized applications.

B. The Significance of Multi-Agent DRL Fuzzing in Enhancing Smart Contract Security

The significance of Multi-Agent DRL fuzzing in enhancing smart contract security lies in its ability to revolutionize security testing processes. By leveraging collaborative problem-solving among intelligent agents, Multi-Agent Systems enhance the scalability and efficiency of security testing efforts. Integrating deep reinforcement learning techniques enables agents to learn optimal strategies for vulnerability detection, improving the overall security posture of smart contracts. Multi-agent DRL fuzzing represents a paradigm shift in security testing methodologies, offering a robust and adaptive approach to identifying and mitigating vulnerabilities in blockchain systems.

C. Call to Action for Future Research and Collaboration

As we look towards the future of smart contract security, a call to action for future research and collaboration is essential. Researchers are encouraged to explore advancements in algorithmic efficiency, transparency through Explainable AI, and the application of MAS in diverse security domains. Collaboration with blockchain development teams to embed built-in security features directly into protocols is crucial for ensuring inherent security by design. By fostering interdisciplinary collaborations and innovative research initiatives, the field of smart contract security can continue to evolve, addressing emerging challenges and enhancing the resilience of decentralized applications.

In conclusion, the future of smart contract security hinges on integrating Multi-Agent DRL fuzzing techniques, which offer a collaborative and adaptive approach to security testing. By embracing advancements in algorithmic efficiency, transparency through Explainable AI, and cross-domain applications of MAS, researchers can pave the way for robust and secure decentralized applications. A call to action for future research and collaboration underscores the importance of continuous innovation and interdisciplinary cooperation in addressing the evolving challenges of smart contract security. Through these efforts, the field can advance towards a more secure and resilient blockchain ecosystem.

ACKNOWLEDGMENT

This work was supported/funded by the Ministry of Higher Education under Fundamental Research Grant Scheme (FRGS/1/2021/ICT07/UTM/02/2).

REFERENCES

[1] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 254–269.

[2] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in International conference on principles of security and trust, Springer, 2017, pp. 164–186.

[3] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on ethereum," vol. 48, no. 1, pp. 327–345, 2022, [Online]. Available: https://doi.org/10.1109/tse.2020.2989002

[4] Y. Wang, P. Jia, L. Liu, C. Huang, and Z. Liu, "A systematic review of fuzzing based on machine learning techniques," vol. 15, no. 8, pp. e0237749–e0237749, 2020, [Online]. Available: https://doi.org/10.1371/journal.pone.0237749

[5] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: Machine learning for input fuzzing," in 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2017, pp. 50–59.

[6] T. Nguyen and V. J. Reddi, "Deep Reinforcement Learning for Cyber Security," Inst. Electr. Electron. Eng., vol. 34, no. 8, pp. 3779–3795, 2023, doi: 10.1109/tnnls.2021.3121870.

[7] A. Ye, L. Wang, L. Zhao, and J. Ke, "¡i¿Ex¡/I¿ ¡sup¿2¡/Sup¿ : Monte carlo tree Search-based test inputs prioritization for fuzzing deep neural networks," vol. 37, no. 12, pp. 11966–11984, 2022, [Online]. Available: https://doi.org/10.1002/int.23072

[8] W. Lv, J. Xiong, J. Shi, Y. Huang, and S. Qin, "A deep convolution generative adversarial networks based fuzzing framework for industry control protocols," vol. 32, no. 2, pp. 441–457, 2020, [Online]. Available: https://doi.org/10.1007/s10845-020-01584-z

[9] M. Lin, Y. Zeng, T. Wu, Q. Wang, L. Fang, and S. Guo, "GSA-Fuzz: Optimize seed mutation with gravitational search algorithm," vol. 2022, pp. 1–17, 2022, [Online]. Available: https://doi.org/10.1155/2022/1505842

[10] A. Mukhtarova and N. I. Lesnova, "Smart contracts in international trade in services in the field of intellectual property," 2019, [Online]. Available: https://doi.org/10.2991/iscde-19.2019.100

[11] Y. Zhuang, B. Wang, J. Sun, H. Liu, S. Yang, and Q. Da, "Deep learning-based program-wide binary code similarity for smart contracts," vol. 74, no. 1, pp. 1011–1024, 2023, [Online]. Available: https://doi.org/10.32604/cmc.2023.028058

[12] P. Praitheeshan, L. Pan, J. Yu, J. K. Liu, and R. Doss, "Security analysis methods on ethereum smart contract vulnerabilities: A survey," 2019, [Online]. Available: https://arxiv.org/abs/1908.08605

[13] B. Jiang, Y. Liu, and W. K. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2018, pp. 259–269.

[14] L. Brent et al., "Vandal: A Scalable Security Analysis Framework for Smart Contracts," pp. 1–28, 2018, [Online]. Available: http://arxiv.org/abs/1809.03981

[15] J. Liang et al., "DeepFuzzer: Accelerated Deep Greybox Fuzzing," Ieee Trans. Dependable Secur. Comput., 2020, doi: 10.1109/tdsc.2019.2961339.

[16] Y. Ye, X. Zhang, and J. Sun, "Automated Vehicle's Behavior Decision Making Using Deep Reinforcement Learning and High-Fidelity Simulation Environment," Transp. Res. Part C Emerg. Technol., 2019, doi: 10.1016/j.trc.2019.08.011.

[17] [J. Zhang, Z. Cui, X. Chen, H. Yang, L. Zheng, and J. Liu, "CIDFuzz: Fuzz Testing for Continuous Integration," Iet Softw., 2023, doi: 10.1049/sfw2.12125.

[18] Z. Gui, S. Y. R. Hui, F. Kang, and X. Xiong, "FIRMCORN: Vulnerability-Oriented Fuzzing of IoT Firmware via Optimized Virtual Execution," Ieee Access, 2020, doi: 10.1109/access.2020.2973043.

[19] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[20] H. Ji, O. Alfarraj, and A. Tolba, "Artificial Intelligence-Empowered Edge of Vehicles: Architecture, Enabling Technologies, and Applications," IEEE Access, vol. 8, pp. 61020–61034, 2020, doi: 10.1109/ACCESS.2020.2983609.

[21] H. Yin and S. J. Pan, "Knowledge transfer for deep reinforcement learning with hierarchical experience replay," vol. 31, no. 1, 2017, [Online]. Available: https://doi.org/10.1609/aaai.v31i1.10733

[22] P. Andersen, M. Goodwin, and O. Granmo, "Towards a deep reinforcement learning approach for tower line wars." pp. 101–114, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-71078-5_8

[23] Z. Liang, D. Feng, and X. Qu, "Deep reinforcement learning based three-dimensional path tracking control of an underwater robot," vol. 2456, no. 1, p. 12031, 2023, [Online]. Available: https://doi.org/10.1088/1742-6596/2456/1/012031

[24] C. El Mazgualdi, T. Masrour, I. El Hassani, and A. Khdoudi, "A deep reinforcement learning (DRL) decision model for heating process parameters identification in automotive glass manufacturing." pp. 77–87, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-51186-9_6

[25] M. Chen, A. Joseph, M. Kumhof, X. Pan, R. Shi, and X. Zhou, "Deep reinforcement learning in a monetary model," 2021, [Online]. Available: https://arxiv.org/abs/2104.09368

[26] H. Mouratidis, P. Giorgini, and G. A. Manson, "Modelling secure multiagent systems," 2003, [Online]. Available: https://doi.org/10.1145/860575.860713

[27] W. Y. Wang, J. Li, and X. He, "Deep reinforcement learning for NLP," 2018, [Online]. Available: https://doi.org/10.18653/v1/p18-5007

[28] K. Yang, "Using DQN and double DQN to play flappy bird." pp. 1166–1174, 2022. [Online]. Available: https://doi.org/10.2991/978-94-6463-010-7_120

[29] E. Korkmaz, "Deep reinforcement learning policies learn shared adversarial features across MDPs," vol. 36, no. 7, pp. 7229–7238, 2022, [Online]. Available: https://doi.org/10.1609/aaai.v36i7.20684

[30] Y. Xu, G. Hu, L. You, and C. Cao, "A Novel Machine Learning-Based Analysis Model for Smart Contract Vulnerability," Secur. Commun. Networks, vol. 2021, 2021, doi: 10.1155/2021/5798033.

[31] O. Oruç, "Role-based embedded domain-specific language for collaborative multi-agent systems through blockchain technology," 2021, [Online]. Available: https://doi.org/10.5121/csit.2021.110501

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv Prepr. arXiv1707.06347, 2017.

[33] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," pp. 2961–2970, 2018, [Online]. Available: http://proceedings.mlr.press/v97/iqbal19a/iqbal19a.pdf

[34] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," arXiv Prepr. arXiv1603.01121, 2016, [Online]. Available: https://arxiv.org/pdf/1603.01121.pdf

[35] H. Tang et al., "Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction," arXiv (Cornell Univ., 2018, doi: https://doi.org/10.48550/arxiv.1809.09332.

[36] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," 2018, [Online]. Available: https://doi.org/10.1109/mascots.2018.00034

[37] W. Wang et al., "A survey on consensus mechanisms and mining strategy management in blockchain networks," vol. 7, pp. 22328–22370, 2019, [Online]. Available: https://doi.org/10.1109/access.2019.2896108

[38] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the internet of things: A comprehensive survey," vol. 21, no. 2, pp. 1676–1717, 2019, [Online]. Available: https://doi.org/10.1109/comst.2018.2886932

[39] Q. Nasir, I. Qasse, M. A. Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," vol. 2018, pp. 1–14, 2018, [Online]. Available: https://doi.org/10.1155/2018/3976093

[40] U. S. Gaire, "Application of artificial intelligence in the military: An overview," vol. 4, no. 01, pp. 161–174, 2023, [Online]. Available: https://doi.org/10.3126/unityj.v4i01.52237

[41] M. Khan and J. Vice, "Toward accountable and explainable artificial intelligence part one: Theory and examples," 2022, [Online]. Available: https://doi.org/10.36227/techrxiv.19102085

[42] C. Hou et al., "SquirRL: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning," 2021, [Online]. Available: https://doi.org/10.14722/ndss.2021.24188

[43] H. Zhao and R. Xu, "An innovative mechanism of blockchain technology on joint governance model," vol. 1, no. 2, 2021, [Online]. Available: https://doi.org/10.37965/jait.2020.0038

[44] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 653–663.

[45] M. Mossberg et al., "Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts," in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1186–1189.

[46] J. Muñoz and H. D. Macedo, "Mythril: A framework for bug hunting on the Ethereum blockchain," arXiv Prepr. arXiv1811.03959, 2018.

[47] Y. Feng, E. Torlak, and R. Bodik, "Solythesis: Detecting and Avoiding Solidity Re-Entrancy Attacks," in 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 874–887.

[48] Q. Liu, L. Wang, and Y. Shen, "A Symbolic Execution Approach for Smart Contract Vulnerability Detection," IEEE Trans. Dependable Secur. Comput., 2022.

[49] R. Revere, "Solgraph." 2017. [Online]. Available: https://github.com/raineorshine/solgraph

[50] C. F. Torres and M. Steichen, "Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts," in Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 664–676.

[51] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 67–82.

[52] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2018, pp. 9–16.

[53] L. Brent et al., "Vandal: A scalable security analysis framework for smart contracts," 2018, [Online]. Available: https://arxiv.org/abs/1809.03981

[54] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," Proc. - 2019 IEEE/ACM 2nd Int. Work. Emerg. Trends Softw. Eng. Blockchain, WETSEB 2019, pp. 8–15, 2019, doi: 10.1109/WETSEB.2019.00008.

[55] Y. Zhang, X. Xu, Y. Liu, Q. Zhang, and L. Liu, "SolidityCheck: Quickly Detecting Problems in Smart Contracts through Regular Expressions," J. Syst. Softw., vol. 158, p. 110391, 2019.

[56] Z. Zhou, L. Rui, and J. Wu, "Solstice: A Framework for Analyzing Solidity Smart Contracts," in 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), 2019, pp. 252–259.

[57] P. Tsankov, A. Dan, A. Permenev, D. Drachsler-Cohen, A. Gervais, and M. Vechev, "Securify v2: A Practical Security Analysis Tool for Ethereum Smart Contracts," in Proceedings of the ACM Conference on Computer and Communications Security, 2020, pp. 127–134.

[58] Y. Zhou, R. Wang, Z. Li, X. Luo, T. Wu, and K. Ren, "SIF: A Framework for Solidity Contract Instrumentation and Analysis," in Proceedings of the 35th Annual Computer Security Applications Conference, 2020, pp.

[59] I. Grishchenko, M. Maffei, and C. Schneidewind, "SmartAnvil: Open-source tool suite for smart contract analysis," in International Conference on Principles of Security and Trust, 2020, pp. 53–76.

[60] B. Alpern, M. Bozga, P. Habermehl, R. Iosif, and J. Sifakis, "SolCheck: A Tool for the Static Analysis of Solidity Smart Contracts," in 2020 IEEE 20th International Symposium on Network Computing and Applications (NCA), 2020, pp. 1–4.

[61] P. Kushwaha, A. Shukla, and S. Sharma, "A Systematic Review of Ethereum Smart Contract Analysis Tools," IEEE Access, vol. 10, pp. 13311–13331, 2022.

[62] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts.," in Ndss, 2018, pp. 1–12.

[63] Á. Hajdu and D. Jovanovic, "solc-verify: A Modular Verifier for Solidity Smart Contracts," in 2019 Formal Methods in Computer Aided Design (FMCAD), 2019, pp. 1–5.

[64] S. K. Lahiri et al., "VeriSol: A verifier for Solidity smart contracts," in International Symposium on Formal Methods, 2019, pp. 596–602.

[65] T. Chen, "A Comparative Analysis of Historical Versions of Ethereum Smart Contracts for Security Issues," in 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2020, pp. 1–4.

[66] A. N'Da, B. A. Kaba, T. F. Bissyandé, and J. Klein, "Security Patterns for Smart Contract Programming in Solidity," IEEE Access, vol. 8, pp. 222957–222967, 2020.

[67] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "VerX: Safety Verification of Smart Contracts," in 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 1661–1677.

[68] A. López Vivar, A. L. Sandoval Orozco, and L. J. García Villalba, "A security framework for Ethereum smart contracts," Comput. Commun., vol. 172, pp. 119–129, Apr. 2021, doi: 10.1016/J.COMCOM.2021.03.008.

[69] A. Alkhalifah, A. Ng, P. A. Watters, and A. S. M. Kayes, "A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks," Front. Comput. Sci., vol. 3, no. February, pp. 1–15, 2021, doi: 10.3389/fcomp.2021.598780.

[70] H.-N. Dai, L. Wang, Y. Zhang, and Q. Liu, "SuperDetector: A Framework for Detecting Smart Contract Vulnerabilities," IEEE Trans. Netw. Sci. Eng., vol. 9, no. 1, pp. 13–25, 2022.

[71] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "DLFuzz: Differential Fuzzing Testing of Deep Learning Systems," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 739–753.

[72] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: effective, usable, and fast fuzzing for smart contracts," in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, in ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, pp. 557–560. doi: 10.1145/3395363.3404366.

[73] V. Wüstholz and M. Christakis, "Harvey: A Greybox Fuzzer for Smart Contracts," arXiv Prepr. arXiv1905.06944, 2019.

[74] J. He, M. Balunović, N. Ambroladze, P. Tsankov, and M. Vechev, "Learning to fuzz from symbolic execution with application to smart contracts," Proc. ACM Conf. Comput. Commun. Secur., pp. 531–548, 2019, doi: 10.1145/3319535.3363230.

[75] L. Wei, Q. Liu, and Y. Shen, "FuzzTaintAnalysis: Fuzzing Smart Contracts Based on Taint Analysis and Genetic Algorithms," in 2020 IEEE International Conference on Blockchain (Blockchain), 2020, pp. 359–366.

[76] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "Sfuzz: An efficient adaptive fuzzer for solidity smart contracts," Proc. - Int. Conf. Softw. Eng., pp. 778–788, 2020, doi: 10.1145/3377811.3380334.

[77] S. Ding, Y. Zhang, T. Ban, Q. Liu, and Y. Shen, "HFContractFuzzer: Fuzzing Hyperledger Fabric Smart Contracts for Vulnerability Detection," in 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021, pp. 1–3.

[78] M. Xu, L. Wang, and Q. Liu, "CodeEmbedding: A Novel Approach for Vulnerability Detection in Fabric Smart Contracts," IEEE Trans. Netw. Sci. Eng., vol. 10, no. 2, pp. 1103–1114, 2023.

[79] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Graph Neural Networks for Smart Contract Vulnerability Detection," in IEEE Access, 2020, pp. 57510–57520.

[80] K. Ashizawa, S. Hara, and J. Sakuma, "Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum," in Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 1116–1130.

[81] Q. Liu, Y. Zhuang, and Y. Shen, "GNNExpert: A System for Combining Graph Neural Networks with Expert Knowledge for Vulnerability Detection," Expert Syst. Appl., vol. 168, p. 114444, 2021.

[82] S.-H. Hwang, S.-H. Lee, and J.-H. Lee, "CodeNet: A Code-Targeted Convolutional Neural Network for Smart Contract Vulnerability Detection," Appl. Sci., vol. 12, no. 4, p. 2104, 2022.

[83] E. Sosu, P. Zavarsky, and B. Swar, "Enhanced Machine Learning Techniques for Automated Vulnerability Detection in Smart Contracts," Comput. Secur., vol. 115, p. 102669, 2022.

[84] L. Huang, Q. Liu, Y. Zhuang, and Q. He, "A Multi-Task Learning Approach for Vulnerability Detection in Smart Contracts," Comput. Secur., vol. 112, p. 102510, 2022.

[85] L. Wang, Q. Liu, and Y. Shen, "A Graph Convolutional Network Model for Vulnerability Detection in Smart Contracts," IEEE Trans. Netw. Sci. Eng., vol. 10, no. 1, pp. 305–316, 2023.

[86] Y. Shen, L. Wang, and Q. Liu, "GSVD: A Common Vulnerability Dataset for Smart Contracts on BSC and Polygon," J. Netw. Comput. Appl., vol. 204, p. 103390, 2023.

[87] L. Han, Y. Zhang, Y. Li, Y. Zhao, and Q. Liu, "A Fusion Learning Model for Smart Contract Vulnerability Detection," IEEE Trans. Netw. Sci. Eng., 2023.

[88] M. Li, Q. Liu, and L. Wang, "Vulpedia: Detecting Smart Contract Vulnerabilities Using Abstract Vulnerable Signatures," IEEE Trans. Inf. Forensics Secur., vol. 18, pp. 1540–1551, 2023.

[89] P. Qian, Y. Zhuang, W. Shi, and Q. He, "Deep Learning for Reentrancy Detection in Ethereum Smart Contracts," IEEE Access, vol. 8, pp. 148145–148155, 2020.

[90] X. Tang, Q. Liu, and L. Wang, "LightningCat: A Deep Learning Framework for Smart Contract Vulnerability Detection," Inf. Sci. (Ny)., vol. 610, pp. 419–433, 2023.

[91] Y. Gong, Q. Liu, and L. Wang, "SCGformer: A Transformer-Based Model for Vulnerability Detection in Smart Contracts," Inf. Sci. (Ny)., vol. 611, pp. 304–317, 2023.

[92] C. Diligence, "Mythos: Security Analysis Tool for Ethereum Smart Contracts." 2019. [Online]. Available: https://github.com/ConsenSys/mythos-cli

[93] Consensys, "MythX: Smart contract security service for Ethereum." Accessed: Mar. 05, 2024. [Online]. Available: https://mythx.io/

[94] V. Contributors, "Vulnerability Analysis as a Service (VaaS) for Ethereum Smart Contracts." 2019. [Online]. Available: https://github.com/VaaS/smart-contract-audit

[95] S. C. Contributors, "Solidity Coverage." Accessed: Mar. 05, 2024. [Online]. Available: https://github.com/sc-forks/solidity-coverage

[96] N. Labs, "Solidity Flattener." Accessed: Mar. 05, 2024. [Online]. Available: https://github.com/nomiclabs/truffle-flattener

[97] A. Suciu, R. Toderean, and M. Buhu, "Porosity: A Decompiler For Blockchain-Based Smart Contracts Bytecode," in 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017, pp. 50–57.

[98] E. Albert, P. Gordillo, A. Rubio, and I. Sergey, "EthIR: A Framework for High-Level Analysis of Ethereum Bytecode," in International Symposium on Automated Technology for Verification and Analysis, 2019, pp. 513–520.

[99] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks," 26th Annual Network and Distributed System Security Symposium, NDSS 2019. 2019. doi: 10.14722/ndss.2019.23413.

[100] S.-M. Chen, T.-F. Tsai, and R.-H. Lai, "Gasper: Analyzing the Energy Consumption of Mobile Offloading in Ethereum," in 2019 IEEE 20th International Conference on Mobile Data Management (MDM), 2019, pp. 227–232.

[101] E. Foundation, "Remix IDE." 2016. [Online]. Available: https://remix.ethereum.org

[102] L. Duarte, "Solium: Analyzing the Security of Smart Contracts," in 2017 IEEE 39th Sarnoff Symposium, 2017, pp. 1–5.

[103] Protofire, "Solhint." 2018. [Online]. Available: https://github.com/protofire/solhint

[104] H. Horta, M. Ribeiro, and R. Medeiros, "SolMet: A Metric Suite for Solidity Smart Contracts," in 2021 IEEE/ACM 1st International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2021, pp. 16–22.

[105] H. Wang and P. Mueller, "SolRazor: Combining Source-Level Optimizations with Correctness Proofs for Smart Contracts," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 1282–1293.

[106] F. Bond, "Solidity Parser Antlr." 2018. [Online]. Available: https://github.com/federicobond/solidity-parser-antlr

[107] H. Liu, Z. Yao, F. Xiong, P. He, Z. Zhang, and Q. Deng, "SolProfiler: Profiling the Performance and Gas Costs of Smart Contracts Based on Ethereum," in 2020 IEEE International Conference on Services Computing (SCC), 2020, pp. 49–56.

[108] J. Ellul and G. J. Pace, "ContractLarva: A Runtime Verification Framework for Smart Contracts," in 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 5–6.

[109] Z. Gao, "When Deep Learning Meets Smart Contracts," Proc. - 2020 35th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2020, no. i, pp. 1400–1402, 2020, doi: 10.1145/3324884.3418918.

[110] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "SolStress: A Tool to Stress Test the Resilience of Solidity Smart Contracts," in 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 9–10.