Gerardo Iuliano — Ph.D. Proposal - XL Cycle

Toward Secure-by-design Smart Contract Development

1 Context of the Research

In recent years, blockchain technology has garnered significant attention as a life-changing innovation poised to redefine traditional systems across various industries. This revolutionary technology offers decentralized and secure solutions that enhance transparency and efficiency in transactional processes.

Blockchain and Smart Contracts. A blockchain is an immutable, secure digital ledger shared among multiple nodes, ensuring transparency and data integrity without reliance on a central authority. This decentralized architecture enhances trust and security by preventing single points of failure or control. Smart contracts are self-executing programs hosted on blockchains designed to automate the execution of predefined contractual agreements. Once conditions are met, the terms are automatically enforced, eliminating intermediaries and reducing transaction costs and processing times. This automation ensures agreements are executed reliably and efficiently, enhancing trust between parties.

Applications and Benefits. Smart contracts offer a robust framework for secure, transparent digital transactions across finance, supply chain, healthcare, and more industries. They facilitate innovation and growth by streamlining operations, reducing costs, and opening new avenues for financial products like decentralized finance (DeFi) and non-fungible tokens (NFTs). Their adoption by enterprises enhances supply chain transparency, data integrity, and identity verification, thereby increasing overall investment value. Cryptocurrencies and blockchain applications further democratize access to global financial services, promoting inclusivity and economic empowerment.

Challenges and Security Concerns of Smart Contracts. Despite their many advantages, smart contracts face significant challenges, especially concerning security. There is a lack of established best practices to systematically guide the development of smart contracts with security as a core design principle, making smart contracts vulnerable to weaknesses, coding errors, and vulnerabilities, which can be exploited to cause substantial financial losses and other serious consequences. Existing vulnerability taxonomies that categorize smart contract vulnerabilities are often incomplete or inadequate, eventually confusing developers and hindering effective communication about security risks. Detection tools designed to identify vulnerabilities in smart contracts also pose challenges. Many of these tools suffer from high rates of false positives, wasting developers' time and resources investigating false alarms rather than focusing on genuine security threats. Another significant issue is the need for more precise and comprehensive guidelines for secure smart contract development, leading developers to introduce vulnerabilities due to oversight or lack of awareness inadvertently. Finally, integrated development environments often do not adequately support developers in addressing security concerns, exacerbating the risks associated with human errors and malicious attacks, as developers may struggle to implement robust security measures effectively.

> **◉ Proposal Main Objective**
>
> *This research aims to establish a security-by-design approach for smart contract development aimed at enhancing security and code quality from the initial stages of implementation. My objective includes integrating advanced vulnerability detection tools into the development pipeline of the AstraKode Blockchain (AKB) platform. This initiative seeks to minimize the vulnerabilities associated with smart contracts and facilitate the development of more resilient and secure decentralized applications (DApps).*

2 State of The Art: Vulnerabilities, Tools, and Benchmarks

Vulnerabilities in smart contracts can lead to severe financial losses. Despite extensive research documenting such vulnerabilities, existing solutions often fail to fully address the complexities and risks inherent in smart contract development. The current state of blockchain security underscores the need for improved methods to enhance the robustness and security of decentralized applications, ensuring they are safe and reliable for broader adoption.

Vulnerability Classification. The white literature highlights the need for a comprehensive taxonomy to address the related security risks. Luu et al. (2016) [1] describe various Ethereum smart contract vulnerabilities without additional categorization. They define these vulnerabilities, provide code snippets and examples of attacks, and discuss affected real-world smart contracts. They propose enhancements to Ethereum's operational semantics to address some of these issues. Atzei et al. (2017) [2] classify smart contract vulnerabilities based on their occurrence: in the source code (typically Solidity), at the machine level (bytecode or instruction semantics), or at the blockchain level. This taxonomy includes mappings to real examples of attacks and vulnerable smart contracts. Nevertheless, some inconsistencies exist in classifying specific vulnerabilities. For instance, the "unpredictable state" vulnerability is illustrated with an example commonly seen as transaction order dependency in other works. Problems associated with dynamic libraries are also classified under the same type, although these issues stem from inherently different causes. Dika (2017) [3] extends Atzei et al.'s taxonomy by incorporating additional vulnerabilities and evaluating their criticality levels, providing a more comprehensive overview of potential security issues in smart contracts. The gray literature instead offers the Decentralized Application Security Project Top 10 (DASP Top 10[1]) and Smart Contract Weakness Classification Registry (SWC Registry[2]). The DASP Top 10, initiated by NCC Group in 2018, identifies ten categories of smart contract vulnerabilities without defining them or explaining their selection and ranking criteria. Several studies, e.g., Durieux et al. (2020) [4], leverage such categories but highlight that they are insufficient. The SWC Registry links smart contract vulnerabilities to the Common Weakness Enumeration (CWE) typology by MITRE Corp (2006) [5] and compiles test cases. Currently, the registry includes 36 vulnerabilities, providing descriptions, references, remediation suggestions, and sample Solidity contracts.

Tools and Benchmarks to Identify Vulnerabilities. Although several state-of-the-art tools for vulnerability detection in smart contracts exist, they need to be improved, particularly regarding the datasets used for training and evaluation. These datasets often focus only on the most common vulnerabilities, neglecting equally critical but less investigated issues. Additionally, the automatic labeling process prevalent in these datasets results in a high incidence of false positives, which can confuse developers and lead to inefficient resource allocation, compromising the effectiveness of security assessments. The inadequate documentation of vulnerability patterns further complicates identifying and mitigating security flaws. Notable tools using these benchmarks include DeeSCVHunter [6], which employs a systematic Deep Learning-based framework to detect smart contract vulnerabilities automatically by identifying complex patterns and anomalies. The Ethereum Security Analysis Framework [7] (ESAF) offers a comprehensive solution for analyzing smart contract vulnerabilities, capable of persistent security monitoring and classic vulnerability analysis. SMARTIAN [8] is an efficient open-source fuzzer that excels at uncovering bugs in real-world smart contracts without requiring access to the source code, being particularly useful for developers addressing vulnerabilities in deployed contracts. ESCORT [9], the first Deep Neural Network-based vulnerability detection framework for Ethereum smart contracts, supports lightweight transfer learning on unseen security vulnerabilities. It leverages neural networks to detect vulnerabilities that traditional analysis methods might miss. Even with these advancements, the effectiveness of these tools is limited by the current benchmarks and insufficient documentation of vulnerability patterns. Therefore, it is crucial to enhance benchmark diversity and accuracy, incorporate less common vulnerabilities, implement stricter manual validation, and improve the documentation of vulnerability patterns. Finally, none of the proposed solutions aim to create a comprehensive framework for developing secure smart contracts by design.

[1] *DASP Top 10:* https://dasp.co/
[2] *SWC Registry*: https://swcregistry.io/

Automated Repair of Smart Contracts. The literature provides several tools to suggest fixing vulnerabilities once identified or even generating a patch. EVMPATCH [10] instantly and automatically patches faulty smart contracts using a bytecode rewriting engine tailored for Ethereum. It converts standard contracts into upgradable ones, focusing on strengthening contracts vulnerable to issues like integer overflows/underflows and access control errors. SGUARD [11] automates smart contract vulnerability detection and fixing with minimal runtime impact. It employs symbolic execution and static analysis to pinpoint vulnerabilities and applies specialized patterns to fix them. Elysium [12] provides a scalable approach to repair smart contracts at the bytecode level automatically. It combines template-based and semantic-based patching by inferring contextual information from the bytecode. It can automatically fix seven types of vulnerabilities, although it is designed for easy extension with new templates and bug-finding tools. SCRepair [13] introduces a gas-aware automated approach to repairing smart contracts. Its search-based method explores various mutations of faulty contracts, considering gas usage through the novel gas dominance relationship. Most of these tools focus on post-deployment scenarios rather than supporting developers during code development, reflecting the prevalent challenge of bytecode-focused operations due to limited access to smart contract source code. Reviewing code during the implementation phases could be highly advantageous because it would allow developers to identify and eliminate vulnerabilities at the source code level, ensuring enhanced security and reliability of smart contracts before deployment.

3 Method

The proposed research aims to implement a security-by-design approach for smart contract development, enhancing security and code quality from the initial stages of implementation. My goal is to integrate advanced vulnerability detection tools and a patch suggestion system directly into the development pipeline of the AstraKode Blockchain (AKB) platform. To achieve this goal, the main goal will be broken down into the following Research Goals (RG):

3.1 RG_1: Systematic Literature Review

Challenge: Current taxonomies are not up-to-date, and no established classification has become a standard. There are numerous tools for detecting vulnerabilities, but it is unclear which implementation is optimal for each vulnerability. Additionally, limited information is available on the benchmarks used for their evaluation.
Objective: Categorize vulnerabilities based on common characteristics and attributes. Develop a structured taxonomy to classify various types of vulnerabilities systematically. Compile a comprehensive list of tools to detect vulnerabilities and analyze the methods and techniques they utilize. Establish mappings between these tools and vulnerabilities and compile a list of benchmarks utilized in the literature to evaluate their effectiveness.
Method: I will follow standard guidelines for conducting a Systematic Literature Review (SLR) such as those by Kitchenham and Charters [14].

3.2 RG_2: Tools Infrastructure

Challenge: Tools are validated in disparate environments using different benchmarks, making it challenging to compare them effectively to determine the most suitable for specific needs. No standardized infrastructure allows for evaluating tools under consistent conditions.
Objective: Develop an infrastructure enabling parallel execution of tools in a uniform environment leveraging a standardized benchmark for all tools.
Method: I will combine containerization, e.g., Docker[3] and orchestration, e.g., Kubernetes[4], to developer a container-based platform to execute multiple tools simultaneously.

[3] *Docker is an open platform for developing, shipping, and running applications.*
[4] *Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services.*

3.3 RG_3: Fix Suggestion with LLM

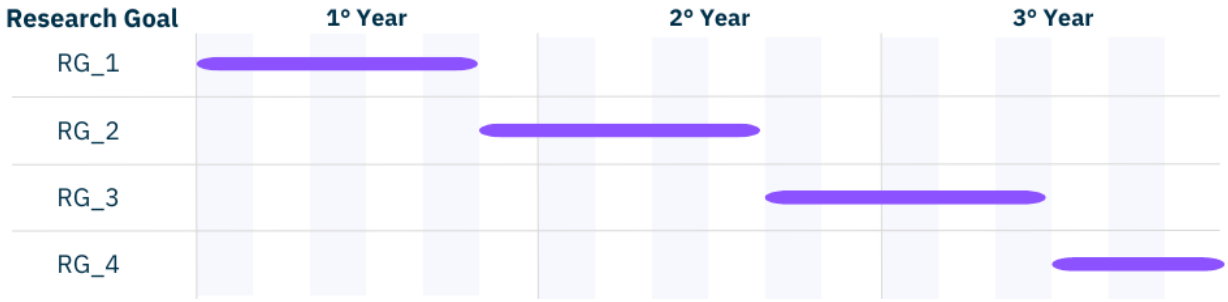
Challenge: Existing tools are not developed to provide support during development and suggest patches. Large Language Models (LLM) are capable of generating vulnerability-free code versions. However, LLM alone cannot effectively identify vulnerabilities due to the broad spectrum of potential issues.
Objective: Use the infrastructure developed in RG_2 to identify vulnerabilities, which will be repaired by LLM by providing in input the smart contract and information concerning the vulnerability.
Method: I will leverage the infrastructure developed in RG_2 to detect vulnerabilities in smart contracts. Subsequently, I will employ an LLM like GPT[5] to generate vulnerability-free smart contracts.

3.4 RG_4: Pipeline Integration

Challenge: AKB features a no-code development approach in its pipeline. Like many other companies, it relies on professionals to verify the correctness of smart contracts. However, auditing is often expensive and error-prone.
Objective: Integrate a chatbot into AKB's pipeline. The chatbot should identify vulnerabilities in the early stages of smart contract development and provide a vulnerability-free version of the contract.
Method: I will integrate the outcomes from RG_3 into a chatbot that will be incorporated into AKB's pipeline.

By achieving these research goals, I aim to create a robust, secure, and efficient smart contract development environment that ensures high code quality and minimizes security risks. The research schedule is shown in Fig. 1. I will spend 10 months on each of the first 3 RGs and 6 months on the fourth RG.

Figure 1: Research Goals Schedule



3.5 Cross-cutting Goals

Alongside the primary objectives, the cross-cutting goals of publishing results, promoting open science, and facilitating technology transfer enhance the research's impact and reach.

Publication of Results. The proposed research encompasses the realms of Software Engineering, Security, and Artificial Intelligence. The attained results will be published in relevant general-purpose venues and venues about the specific themes of the proposal. The targeted venues will be, but not limited to, IEEE Transactions on Software Engineering (TSE), ACM Transactions on Software Engineering and Methodology (TOSEM), Empirical Software Engineering (EMSE), Journal of Systems and Software (JSS), the International Conference on Software Engineering (ICSE), the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), the IEEE International Conference on Software Testing (ICST). Their rankings are listed respectively by Scimago Journal Rank, with values ranging from Q1 to Q2, and Core Ranking, with values ranging from A to A*.

Open Science and Technology Transfer. The proposed research will adhere to Open Science, making all findings freely available to enhance accessibility, transparency, and efficiency in scientific research. The research will also adopt the Technology Transfer methodology to make technology accessible to the market, ensuring that discoveries benefit a wider audience.

[5] *GPT is a type of artificial intelligence model developed by OpenAI.*

4 Expected Impact

The research impact of the proposed solutions could be substantial and multifaceted.

Boosting Blockchain Trust through Smart Contract Security. The proposed research will increase stakeholders' confidence in adopting DApps. As smart contract security improves, more industries may adopt blockchain technology, accelerating innovation and integration across finance, supply chain, and healthcare sectors. Enhanced security will also boost trust among end users and regulators, which is crucial for mainstream blockchain adoption. AKB will be able to promote a trustworthy blockchain ecosystem by ensuring secure smart contracts, encouraging broader industry implementation, and leading to a robust technology framework.

Reduction in Financial Losses. Smart contracts often manage significant financial transactions. This research could help prevent substantial financial losses due to hacking or other security breaches by minimizing vulnerabilities and potential exploits, benefiting both individual users and organizations.

Enhanced Development Practices and Standardization. Establishing a security-by-design approach will provide developers with a structured methodology that promotes best practices in smart contract development and elevates the overall code quality, making the development process more efficient and less error-prone. The methodology could also be standardized to increase the adoption of such practices across the industry, fostering a more cohesive and secure blockchain ecosystem. AKB could streamline development practices, reduce vulnerabilities, and contribute to a higher security standard in blockchain applications.

Educational Impact. The research findings could be used to educate developers, helping them understand the importance of security in smart contract development. Based on the research, training programs and educational materials could be developed, contributing to a more knowledgeable and skilled developer community.

Overall, this research has the potential to create a more secure, reliable, and widely accepted blockchain ecosystem, fostering innovation and trust in decentralized applications. By establishing a robust security-by-design approach for smart contract development, the project aims to revolutionize blockchain technology. Enhanced security and reliability through advanced vulnerability detection and formal verification techniques will build greater trust and transparency among users and regulators. Integrating these improvements into the AKB platform will set new industry standards, paving the way for secure, efficient, and widely adopted blockchain applications.

References

[1] Loi Luu et al. *Making Smart Contracts Smarter*. Cryptology ePrint Archive, Paper 2016/633. 2016.

[2] Nicola Atzei et al. "A Survey of Attacks on Ethereum Smart Contracts (SoK)". In: 2017.

[3] Ardit Dika. "Ethereum Smart Contracts: Security Vulnerabilities and Security Tools". In: 2017.

[4] Thomas Durieux et al. "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts". In: ICSE '20.

[5] MITRE Corp. *Common Weakness Enumeration (CWE): A Community-Developed List of Software Weakness Types*. 2006.

[6] Xingxin Yu et al. "DeeSCVHunter: A Deep Learning-Based Framework for Smart Contract Vulnerability Detection". In: 2021.

[7] Antonio López Vivar et al. "A security framework for Ethereum smart contracts". In: *Computer Communications* (2021).

[8] Jaeseung Choi et al. "SMARTIAN: Enhancing Smart Contract Fuzzing with Static and Dynamic Data-Flow Analyses". In: 2021.

[9] Oliver Lutz et al. *ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning*. 2021. arXiv: 2103.12607 [cs.CR].

[10] Michael Rodler et al. "EVMPatch: Timely and Automated Patching of Ethereum Smart Contracts". In: Aug. 2021.

[11] Tai D. Nguyen et al. "SGUARD: Towards Fixing Vulnerable Smart Contracts Automatically". In: 2021.

[12] Christof F. Torres et al. "Elysium: Context-Aware Bytecode-Level Patching to Automatically Heal Vulnerable Smart Contracts". In: RAID '22. New York, NY, USA: Association for Computing Machinery.

[13] Xiao Liang Yu et al. "Smart Contract Repair". In: *ACM Trans. Softw. Eng. Methodol.* (2020). ISSN: 1049-331X.

[14] Barbara Kitchenham and Stuart Charters. "Guidelines for performing Systematic Literature Reviews in Software Engineering". In: 2 (Jan. 2007).