UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE DISERTAȚIE

Streamlining Blockchain security audits through desktop applications

propusă de

Marius Enache-Stratulat

Sesiunea: Iulie, 2025

Coordonator științific

Conf. Dr. Andrei Arusoaie

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Streamlining Blockchain security audits through desktop applications

Marius Enache-Stratulat

Sesiunea: Iulie, 2025

Coordonator științific

Conf. Dr. Andrei Arusoaie

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Enache-Stratulat Marius** domiciliat în **România, jud. Iași, mun. Iași, strada Amurgului, nr. 10, bl. 258B, et. 4, ap. 14**, născut la data de **31 mai 2001**, identificat prin CNP **5010531046224**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **Securitatea Informatiei**, promoția 2025, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Streamlining Blockchain security audits through desktop applications** elaborată sub îndrumarea domnului **Conf. Dr. Andrei Arusoaie**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: .... Semnătura: ....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Streamlining Blockchain security audits through desktop applications**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Marius Enache-Stratulat**

Data: ....   Semnătura: ....

Contents

Motivation

The Blockchain Technology is relatively new if we consider the release of Bitcoin cryptocurrency blockchain newtork which took place in 2009. However, its foundation was formed decades ago, in 1989, when Leslie Lamport developed the Paxos protocol (to be continued...)

Abstract

The Blockchain, despite its popularity and presence on the internet and in the financial sector, is a relatively new technology and this fact is proved by the large amount of attacks that took place on such networks. This situation raises the need for secure and robust networks that can keep the clients' financial assets safe. Fortunately, there are various solutions to this category of problems, like Slither or Manticore, which take different approaches towards finding vulnerabilities. The aim of our paper is to highlight the strenghts of Slither as a static analysis tool and present a solution that integrates it in a streamlined manner and eases the process of scanning blockchain projects for beginner and intermediate developers.

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Chapter 1

Threats of Blockchain

(TEST)The Blockchain[11] is a technology used in various applications, such as cryptocurrencies, Internet of Things, electronic voting and many others. Its transparent, fully distributed peer to peer and append-only nature are its most notable strengths. The fact that transactions, which may have a different meaning depending on the application they are used for, are transparent and immutable enables the existence of a verifiable ledger with no need of a centralized authority. Bitcoin, the most popular cryptocurrency at this moment, takes advantage of these characteristics, creating a decentralized financial system. Unfortunately, there are also significant shortcomings in terms of security. One of the most notorious instances where these vulnerabilities had been exploited is known as "The DAO", where in 2016, an unknown attacker managed to drain $50 million USD through a *Reentrancy Attack*.

This paper will first focus on the variety of attacks that blockchain networks are vulnerable to, their manner of work and potential effects to its victims. The Quadriga Initiative[8] - a community-based platform - hosts a list of most of the attacks and frauds that involve cryptocurrency exchanges. Among the most well-known attacks, the database of case studies includes instances of Reentrancy Attacks, Replay Attacks and Short Address Attacks.

Reentrancy Attacks[5] can take place whenever the developers of the targeted network do not update the balances before sending the funds. In this case, an attacker could recursively call the withdraw() method and drain all the available funds.

Replay Attacks[12] are specific to the case where a cryptocurrency is forked into two separate currencies. In such a scenario, it is possible to sniff a regular transaction from any of the chains and "replay" it on the other. Processing both transaction causes

2

the user to lose the same amount of assets twice (for example, paying two ether instead of one for the same product or service). This attack was easily possible in the Ethereum blockchain before the implementation of chainID in transactions.

Short Address Attacks[11] represent the exploit of a bug in the EVM – Ethereum Virtual Machine – used to obtain extra tokens on purchases. This category of attacks requires creating an Ethereum walled ending with a 0 byte. The attacker makes a purchase on the address by removing the last digit and causes the contract to try to append the missing byte to the incomplete address. In this case, the contract pays 256 times more tokens than intended.

In terms of prevention and detection of such attacks, there are numerous tools that serve this purpose. As Kaixuan Li et al. describe in their article[9], a significant category of such tools detect vulnerabilities in the code through either static analysis or symbolic execution.

Technology	Tool	Analysis Level	Stars
	Securify2		529
Static Analysis	Slither	Source Code	4 500
	SmartCheck		315
	Manticore		3 500
Symbolic Execution	Osiris	Bytecode	50
	Oyente		1 300

Table 1.1: List of the main Static Analysis and Symbolic Execution tools, adapted from Kaixuan Li et al.[9]

According to Table 1.1, Slither had the highest scores on GitHub among the Static Analysis tools, while Manticore had the highest scores among the Symbolic Execution tools. This metric shows the interest expressed by the open source community towards these projects.

Generally, Static Analysis tools[4] are automatic methods that determine the runtime properties of a program without running it. In Slither's[7] case, the analysis' purpose is to detect security vulnerabilities within the smart contract and to point out potential optimizations, which result in a lower gas consumption, an essential aspect in this field. As such, this category of security tools should be considered the opposite of Dynamic Analysis tools, which rely on running the target program and can also monitor it's performance among other aspects.

On the other hand, Symbolic Execution tools[1] take a similar approach of analyzing the given program without running the code, based solely on its bytecode. Esentially, Manticore[10] uses the bytecode of a smart contract to map out all the execution paths of the program. Afterwards, all the paths get explored and the tool determines what kind of inputs determine each possible outcome. Two of the main components of Manticore are the Core Engine and the Event System. The Core Engine is responsible with operating and managing the state of a program at a certain point in the execution. The Execution Module has the purpose to emulate a system that runs the target program, with a CPU, system memory and an operating system like Linux.

```
int foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }
    assert(x - y != 0);
}
```

Figure 1.1: Example of code to analyze. From Roberto B et al.[1]

The figure above shows a simple example of function that a Symbolic Execution tool can operate on. A tool like Manticore can trace all the possible paths the execution may go at run-time. The assertion at the end is the possible point of failure for this function, in case x and y are equal.
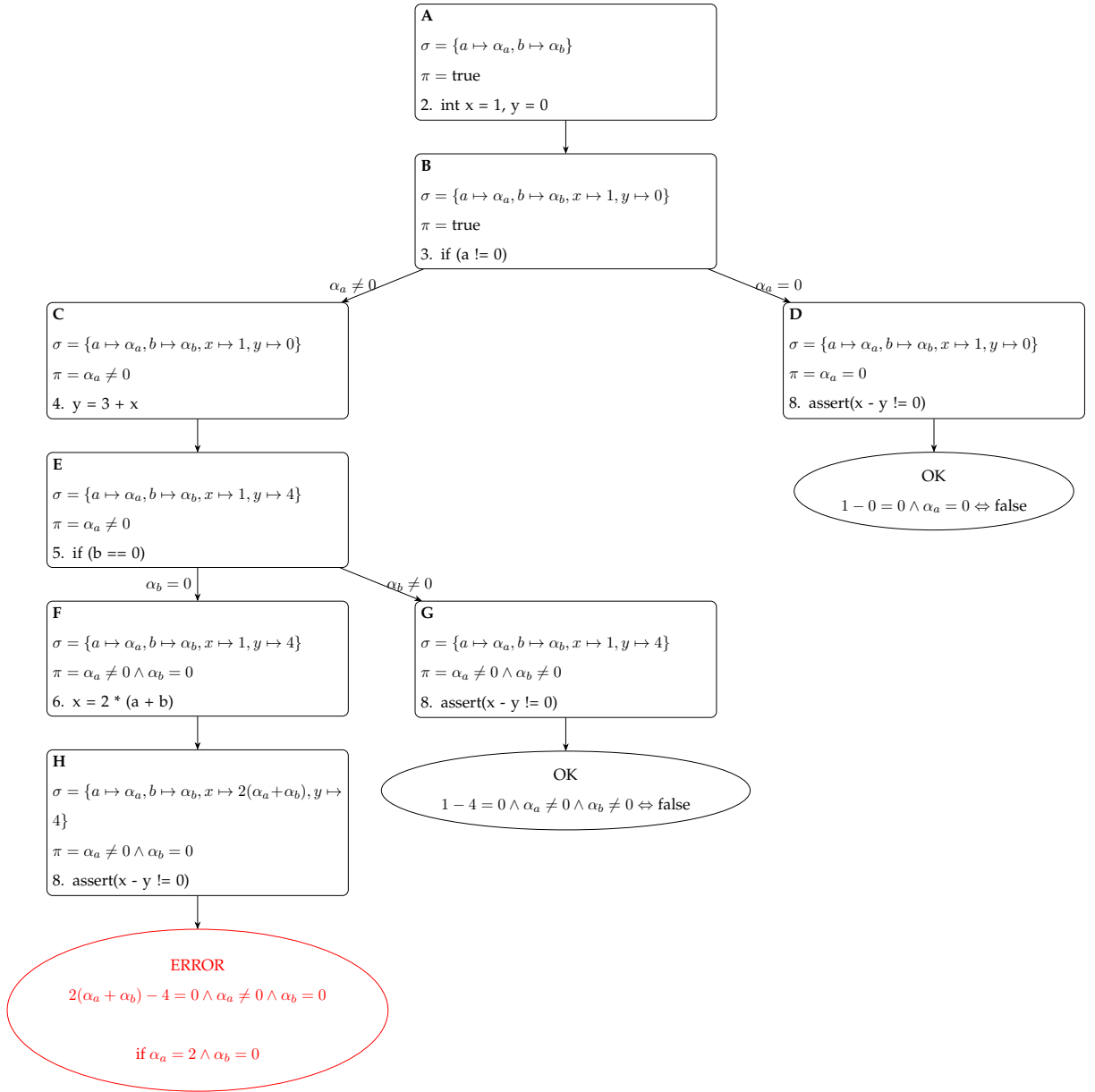
A
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b\}$
$\pi = $ true
2. int x = 1, y = 0

B
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 0\}$
$\pi = $ true
3. if (a != 0)

$\alpha_a \neq 0$ $\alpha_a = 0$

C
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 0\}$
$\pi = \alpha_a \neq 0$
4. y = 3 + x

D
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 0\}$
$\pi = \alpha_a = 0$
8. assert(x - y != 0)

OK
$1 - 0 = 0 \wedge \alpha_a = 0 \Leftrightarrow $ false

E
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 4\}$
$\pi = \alpha_a \neq 0$
5. if (b == 0)

$\alpha_b = 0$ $\alpha_b \neq 0$

F
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 4\}$
$\pi = \alpha_a \neq 0 \wedge \alpha_b = 0$
6. x = 2 * (a + b)

G
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 1, y \mapsto 4\}$
$\pi = \alpha_a \neq 0 \wedge \alpha_b \neq 0$
8. assert(x - y != 0)

OK
$1 - 4 = 0 \wedge \alpha_a \neq 0 \wedge \alpha_b \neq 0 \Leftrightarrow $ false

H
$\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 2(\alpha_a + \alpha_b), y \mapsto 4\}$
$\pi = \alpha_a \neq 0 \wedge \alpha_b = 0$
8. assert(x - y != 0)

ERROR
$2(\alpha_a + \alpha_b) - 4 = 0 \wedge \alpha_a \neq 0 \wedge \alpha_b = 0$

if $\alpha_a = 2 \wedge \alpha_b = 0$

Figure 1.2: Example of symbolic execution tree of function `foobar`. Each state shows the symbolic store σ, the path constraint π, and the current statement. From Roberto B. et al[1]

As shown in this graph, σ contains all the symbols stored in the context of the function, π is a statement that represents the path constraint and below it is the line of code that gets excecuted at a given time. State *B* is the first one to fork in two hypothetical states, based on the value of the first parameter, as shown by π in the states *C* and D. If a == 0, the function skips to the final assert and returns with no failure. Otherwise, the program would continue with state E and the following ones. Finally, a Symbolic Execution tool can reveal that the function will throw an error in case a == 2 and b == 0.

Tool	# Success	# Failure		
		# Timeout	# Compilation	# Total
Slither	767 (97.34%)	2	19	21
Manticore	112 (14.21%)	626	50	676

Table 1.2: Success rate and failures summary of Slither and Manticore. Adapted from Kaixuan Li et al.[9]

Finally, to address Manticore's performance, as shown above, its exhaustive approach causes it to fail most of the tests due to timeout. Even if we were to ignore the timeout fails, Slither is visibly more consistent, having less compilation failures. The main flaw and reason of Manticore's high timeout rate may be the fact that it has to process large amounts of possible paths in order to find the ones where a security flaw is exploited. As such, Slither is the more consistent and better performing in comparison to Manticore.

1.1 Slither

As such, the main focus of this thesis will be Slither[7], a static analysis tool catered to smart contracts written in Solidity and designed to offer granular information about smart contract code. This tool specializes in vulnerability detection, but also manages to point out potential optimizations that lower the gas consumption of a smart contract. In order to facilitate the usage of this tool, Slither leverages printers to help the user better understand the contract's functionality and structure. This is done through representations such as the control flow graph or the inheritance graph or summaries of the contract and the issues detected in its code.

In opposition to dynamic analysis[3] tools, which rely on executing the contract's code to find security weaknesses, static analysis tools detect vulnerabilities without running the smart contract. Such approaches include parsing and decompiling the EVM bytecode and translating it into semantic facts. These semantic facts are then matched with predefined patterns meant to reveal common issues.

Slither[7] proceeds in a similar manner by taking the Solidity Abstract Syntax Tree generated by the compiler and extracting information such as the control flow graph and transforms the code given into SlithIR, its internal representation language. After these two initialization stages, the actual code analysis takes place. This stage includes

identifying the reads and writes of variables, modeling the protection of each function and the data dependency analysis.

This tool[7] helps auditors detect various vulnerabilities and potential security threats, such as reentrancy attacks or uninitialized storage variables. Its aproach of quickly transalating and analyzing large codebases makes it significantly faster than symbolic execution engines, as shown Table 1.2, a great feature for fast development workflows.

Furthermore, its set of detectors are able to find obvious and subtle vulnerabilities alike while providing useful educational resources to help developers fix and prevent future attacks. All these factors make Slither a great solution for quick and automated daily audit processes, preventing many critical security vulnerabilities.

Another essential topic in blockchain development, beside security concerns, is optimization, the techniques used to lower the "gas" consumption. Slither includes a set of detectors specialized in finding ways to optimize code[6] from different perspectives. They are able to find issues such as dead code, as in methods or code branches that are never called or executed, unused state variables or variables that should be declared constant to save gas.

As to how it works[7], Slither uses a multi stage procedure to parse and process the codebase of a blockchain project. First, it uses the Solidity compiler to generate the Abstract Syntax Tree of the contract, from which it recovers important information: inheritance graph, control flow graph and the list of expressions. Next, the contracts get translated to SlithIR, the internal representation language. The following step is the actual code analysis, in which Slither processes data dependencies, read/write variables and protected functions to output detected vulnerabilies and optimization opportunities.
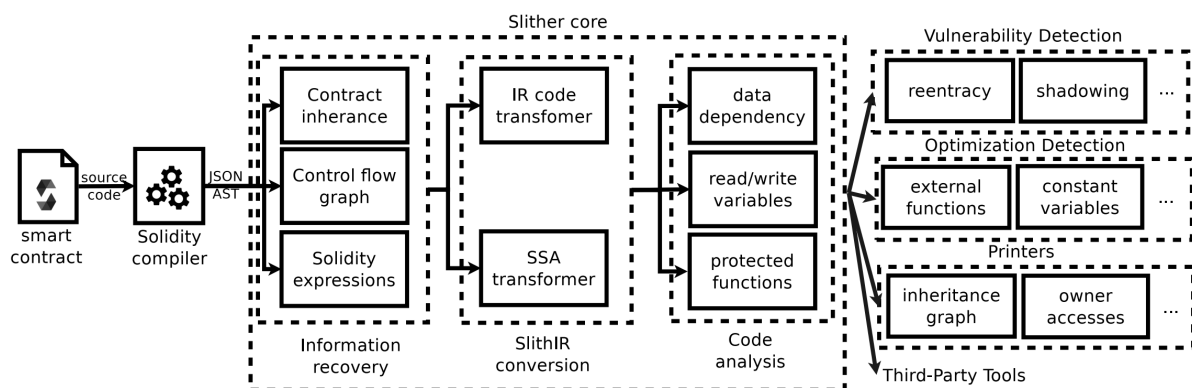

Figure 1.3: Slither overview. From Josselin F. et al.[7]

Chapter 2

Slither's performance analysis

This chapter will be dedicated to assessing Slither's performance in terms of vulnerability detection. The metrics used to determine how well security scanning tools work are: detection rate (how often a tool can correctly flag a vulnerability), analysis speed and attacks coverage (the variety of attacks that a tool can prevent).

		Slither	Securify	SmartCheck	Solhint
	False Positive Rate	10.9%	25%	73.6%	91.3%
Accuracy	Flagged contracts	112	8	793	81
	Detections per contract	3.17	2.12	10.22	2.16
Performance	Execution time	0.79 ± 1	41.4 ± 46.3	10.9 ± 7.14	0.95 ± 0.35
	Time out rate	0%	20.4%	4%	0%
Robustness	Failure rate	0.1%	11.2%	10.22%	1.2%
Reentrancy examples	DAO	Yes	No	Yes	No
	Spankchain	Yes	No	No	No

Table 2.1: Performance comparison between Slither, Securify, SmartCheck and Solhint. From Josselin F. et al.[7]

The table above covers the first two performance criteria: the detection rate and analysis speed. The experiment was done by analyzing 1000 contracts focusing only on reentrancy detectors. Slither's results are visibly better than the ones obtained by its counterparts. Firstly, it's false positive rate is far lower, at just 10.9%, while Securify falls in the second place with 25%. Secondly, it's average execution time is also significantly lower than those of Securify and SmartCheck. In short, Slither is by far faster, more accurate and more consistent than the other three tools. Another important aspect is the time out rate of 0% for Slither and Solhint, which further supports Slither's

consistency in the detection of reentrancy attacks. Also, while this results do not conclude Slither's attack coverage, we can see it is the only tool that managed to detect attacks similar to the ones that targeted DAO or Spankchain, which are notorious for the amount of funds stolen and its effect on people's perception towards blockchain's security.

Vulnerabilities	Read	TP	TN	FP	FN
Re-entrancy	29/31	28	0	0	1
Access Control	18/18	14	0	0	3
Arithmetic	15/15	0	0	5	0
Unchecked LLC	26/26	26	0	0	3
Total		88	68	0	5	7

Table 2.2: Slither's coverage of security attacks. Adapted from Senan B. [2]

This table shows how Slither can detect four types of vulnerabilities: Reentrancy, Access Control, Arithmetic and Unchecked LLC and covers 4 cases: True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN). Out of the 90 test cases given, 88 passed the reading phase, 68 had got correctly flagged as vulnerable, five ended up as False Positives and seven as False Negatives. That equates to a rate of 77.27% for True Positive, 5.68% for False Positive and 7.95% for False Negative. Therefore, while in almost 6% of cases the user is falsely lead to believe a section of his code is vulnerable, a case that has no impact on the project's security, the rate at which Slither misses a vulnerability is almost 8%. In particular, Slither missed 3.45% of reentrancy attacks, 16.67% of Access Contol attacks, none of the Arithmetic attacks and 11.54% of Unchecked LLC attacks. As such, Slither may leave the analyzed project open to Access Control and Unchecked LLC attacks to a higher degree.

2.1   Titlul secțiunii 1

A diam sollicitudin tempor id eu nisl. Hac habitasse platea dictumst vestibulum. Integer enim neque volutpat ac tincidunt. Facilisi nullam vehicula ipsum a arcu cursus vitae congue. Vel turpis nunc eget lorem. Vestibulum mattis ullamcorper velit sed ullamcorper morbi tincidunt ornare. Nunc sed blandit libero volutpat. Sit amet luctus venenatis lectus magna fringilla urna porttitor. Hac habitasse platea dictumst quisque

sagittis purus. Sed faucibus turpis in eu mi bibendum neque egestas. Vel orci porta non pulvinar neque laoreet suspendisse interdum consectetur. Erat nam at lectus urna duis convallis convallis tellus id. Tristique sollicitudin nibh sit amet commodo nulla facilisi nullam vehicula. Etiam dignissim diam quis enim lobortis scelerisque. Nunc congue nisi vitae suscipit tellus mauris a diam maecenas. Lacus viverra vitae congue eu consequat ac felis donec. Mauris sit amet massa vitae tortor condimentum. Mauris augue neque gravida in. Lorem ipsum dolor sit amet. Arcu dui vivamus arcu felis bibendum ut tristique et.

2.2   Titlul secțiunii 2

Sit amet mauris commodo quis imperdiet massa tincidunt nunc pulvinar. Ligula ullamcorper malesuada proin libero nunc consequat interdum. Mauris a diam maecenas sed enim ut. Ut sem nulla pharetra diam sit amet nisl suscipit adipiscing. Leo duis ut diam quam nulla. Neque ornare aenean euismod elementum. Vitae sapien pellentesque habitant morbi tristique senectus. Lectus magna fringilla urna porttitor rhoncus dolor purus non enim. Egestas sed sed risus pretium quam vulputate dignissim suspendisse in. At quis risus sed vulputate odio ut enim. Hac habitasse platea dictumst quisque sagittis. Lectus vestibulum mattis ullamcorper velit sed. Massa vitae tortor condimentum lacinia quis vel eros donec ac. Vulputate dignissim suspendisse in est ante. Sed faucibus turpis in eu mi bibendum neque. Enim eu turpis egestas pretium aenean pharetra magna. Tellus mauris a diam maecenas.

2.3   Titlul secțiunii 3

Faucibus ornare suspendisse sed nisi lacus sed. Mi in nulla posuere sollicitudin aliquam ultrices. Lacus suspendisse faucibus interdum posuere lorem ipsum dolor sit amet. Odio tempor orci dapibus ultrices in iaculis nunc sed augue. Congue eu consequat ac felis donec et odio. Enim ut sem viverra aliquet eget sit amet. Sit amet consectetur adipiscing elit duis tristique sollicitudin. Quis blandit turpis cursus in. Cras fermentum odio eu feugiat pretium nibh ipsum consequat nisl. Non curabitur gravida arcu ac tortor dignissim convallis aenean. Porta non pulvinar neque laoreet suspendisse interdum consectetur libero id. Lacus viverra vitae congue eu consequat ac felis. Vulputate dignissim suspendisse in est ante in nibh mauris. Amet mauris

commodo quis imperdiet massa. Varius sit amet mattis vulputate enim nulla aliquet. Pellentesque diam volutpat commodo sed egestas egestas. Amet est placerat in egestas erat imperdiet sed euismod. Scelerisque varius morbi enim nunc faucibus a pellentesque sit. Ut sem viverra aliquet eget sit amet tellus cras. Sem integer vitae justo eget magna fermentum iaculis eu.

Chapter 3

SmartScan

Amet venenatis urna cursus eget. Quam vulputate dignissim suspendisse in est ante. Proin nibh nisl condimentum id. Egestas maecenas pharetra convallis posuere morbi. Risus viverra adipiscing at in. Vulputate eu scelerisque felis imperdiet. Cras adipiscing enim eu turpis egestas pretium aenean pharetra. In aliquam sem fringilla ut morbi tincidunt augue. Montes nascetur ridiculus mus mauris. Viverra accumsan in nisl nisi scelerisque eu ultrices vitae. In nibh mauris cursus mattis molestie a iaculis. Interdum consectetur libero id faucibus nisl tincidunt eget. Gravida in fermentum et sollicitudin ac orci. Suscipit adipiscing bibendum est ultricies. Etiam non quam lacus suspendisse. Leo urna molestie at elementum eu facilisis sed odio morbi. Egestas congue quisque egestas diam in arcu cursus. Amet consectetur adipiscing elit ut aliquam purus.

3.1   Titlul secțiunii 1

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer

feugiat scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

3.2 Titlul secțiunii 2

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volutpat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliography

[1] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 2018.

[2] Senan Behan. Solidity smart contract testing with static analysis tools. 2022.

[3] Lyubka Dencheva. Comparative analysis of static application security testing (sast) and dynamic application security testing (dast) by using open-source web application penetration testing tools. 2022.

[4] Pär Emanuelsson and Ulf Nilsson. A comparative study of industrial static analysis tools. 2008.

[5] Noama F. and Manar H. Reentrancy vulnerability identification in ethereum smart contracts. 2021.

[6] Josselin Feist, Gustavo Grieco, and Alex Groce. Slither github repository. `https://github.com/crytic/slither`, 2018.

[7] Josselin Feist, Gustavo Grieco, and Alex Groce. Slither: A static analysis framework for smart contracts. 2019.

[8] Quadriga Initiative. Quadriga initiative. `https://quadrigainitiative.com/indexjay.php`. Accessed: 2024-05-24.

[9] Kaixuan Li et al. Static application security testing (sast) tools for smart contracts: How far are we? 2024.

[10] Mark Mo, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, and Artem Dinaburg. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. 2019.

[11] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. Exploring the attack surface of blockchain: A systematic overview. 2019.

[12] Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, and George Danezis. Replay attacks and defenses against cross shard consensus in sharded distributed ledgers. 2020.