TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Nikitchuk Artur IVSB192932

Multichain Non-Fungible Token Minting Application For Security Testing

Bachelor's thesis

Supervisor: Alexander Norta
MSc, PhD

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikitchuk Artur IVSB192932

Turvatestimise eesmärgil kasutatavate asendamatute tokenite ahelsõltumatu müntimisrakendus

Bakalaureusetöö

Juhendaja: Alexander Norta
MSc, PhD

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Artur Nikitchuk

Abstract

Since the birth of bitcoin in 2008, the world has been introduced to a novel concept known as Blockchain, which is reshaping a variety of industries, including but not limited to banking, government, and media. In November 2013, a significant step further was taken in the form of Ethereum. The key idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and decentralised applications. Smart contracts enable creation of decentralised applications where one can create their own arbitrary rules for ownership, transaction formats and state transition functions. Developers, in particular, can specify access permissions for each contract function. Further development introduced Non-Fungible Tokens to the world. The novelty of Non-Fungible Token is that each token is unique and programmable, thus it can be used to identify, prove ownership, and establish the existence of digital assets, as well as serve as a certificate of authenticity.

While a plethora of tools, frameworks & registries have been developed to address vulnerabilities within crypto token smart contracts, there is a gap in available resources regarding security testing of Non-Fungible Token functionality within decentralised applications. This thesis attempts to close a gap in the literature by proposing a working solution for testing the security of NFT functionality integration into applications.

Annotatsioon

Lõputöö on kirjutatud inglise keeles ning sisaldab 7 peatükki, 4 joonist, 3 tabelit. Põhi töö maht (alates sissejuhatusest, lõpetades kokkuvõttega) on 53 lehte.

Alates bitcoini sünnist 2008. aastal on maailm tutvunud uudse kontseptsiooniga, mida nimetatakse plokiahelaks ja mis kujundab ümber erinevaid tööstusharusid, sealhulgas, kuid mitte ainult, pangandust, valitsust ja meediat. 2013. aasta novembris astuti Ethereumi näol oluline samm edasi. Välja pakutud põhiidee oli Turingi täisväärtusliku keele väljatöötamine, mis võimaldab arendada erinevaid programme plokiahela ja detsentraliseeritud rakenduste jaoks. Nutilepingud võimaldavad luua detsentraliseeritud rakendusi, kus saab luua oma reegleid omandiõiguse, tehinguformaatide ja olekute üleminekufunktsioonide kohta. Eelkõige saavad arendajad määrata iga lepingufunktsiooni jaoks juurdepääsuõigused. Edasine areng tõi maailma sisse asendamatuid tokeneid (Non-Fungible Tokens). Non-Fungible Token'i ehk NFT uudsus seisneb selles, et iga token on ainulaadne ja programmeeritav, seega saab seda kasutada digitaalse vara identifitseerimiseks, omandiõiguse tõendamiseks, olemasolu tuvastamiseks ning autentsustunnistusena.

Ehkki regulaarsete krüptotokenide haavatavuste kõrvaldamiseks on välja töötatud hulgaliselt vahendeid, raamistikke ja registreid, on detsentraliseeritud NFT rakenduste turvalisuse testimise osas neid puudu. Käesolevas töös püütakse täita lünk kirjanduses, pakkudes välja toimiva turvatestimise eesmärgil kasutatavate asendamatute tokenite ahelsõltumatu müntimisrakendus.

List of abbreviations and terms

ETH	Ethereum
BTC	Bitcoin
PoW	Proof of Work
PoS	Proof of Stake
NFT	Non-Fungible Token
EIP	Ethereum Improvement Proposal
SWC	Smart Contract Weakness Classification Registry
EVM	Ethereum Virtual Machine
EOA	Externally Owned Address
TOD	Transaction Ordering Dependence
TOCTOU	Time-Of-Check vs Time-Of-Use
DASP	Decentralised Application Security Project
IPFS	The InterPlanetary File System
CID	Content Identifier
DHT	Distributed Hash Table
RSK	Rootstock
JVM	Java Virtual Machine
TCA	Transaction Certificate Authority
CA	Certificate Authorities
VM	Virtual Machine
Dapp	Decentralised Application
PoET	Proof Of Elapsed Time
DSR	Design Science Research

Table of contents

List of figures

List of tables

1. Introduction

Since the development of bitcoin in 2008, the world has been introduced to a novel notion called Blockchain, that is reshaping a variety of industries, including but not limited to banking, government, and media. With the world's largest firms and organisations investing millions in research and development, a variety of innovative technologies such as smart contracts and non-fungible tokens have emerged as a result of experimentation. [1][2][3]. With new emerging technologies, the attack surface for malicious actors has expanded, as a result of which a number of new offensive approaches have been developed [4]. The nature of smart contracts is that once deployed, their source code is made publicly available to anyone interested [4]. This in its turn allows malicious actors to scrutinise and analyse the smart contracts for as long as needed, all while remaining entirely undetected and planning a well-thought-out attack.

Non-Fungible Token (NFT) is a type of crypto currency that is derived by the Ethereum smart contracts. The objective behind NFTs differs from standard crypto currency like Bitcoin by aiming to distinguish each token with distinguishable signs [3]. The uniqueness and programmability of each Non-Fungible Token's property makes it suitable for identification, proof of ownership and proof existence of digital assets as well as serve as a certificate of authenticity [3][5][7].

Currently in the market, most of the sold & used NFTs represent ownerships to digital arts. As of April 2022, current NFT market capitalization is $10,275,018,910.35 with $16,266,350,290.94 all-time sales volume and $42,382,321.34 latest 24 hours sales volume [6]. One of the most popular NFT collections to date is called "CryptoPunks". The current lowest price for a CryptoPunk NFT is 63.95 ETH ($197,712.29 USD according to the price of Ethereum in April 2022) while more expensive ones can cost several millions of dollars [6]. Besides CryptoPunks, there are numerous other collections like "Bored Ape Yacht Club", "Invisible Friends", "Azuki" which follow similar pricing [6]. As a result of outlined factors, malicious actors are increasingly targeting NFT applications and users.

Non-Fungible Token smart contracts have gone through extensive analysis and testing before they were offered to the public [9]. However the availability of testing tools usable during the development and implementation of the NFT-related functionality within applications is currently lacking in variety and availability, resulting in technical security issues related to token minting, token listing and token trading [8], which when exploited can result in substantial financial and reputational losses.

1.1 Thesis Objectives

The aim of this thesis is to fill the identified gap in the security testing process of multichain NFT applications. Based on the identified inefficiencies in currently available NFT minting applications, the paper aims to create a multichain NFT minting application which can be utilised during the security testing of applications that include multichain NFT functionality. Supplemental web application & smart contracts will be created with the aim of facilitating the security testing process. The web application will allow application security testers and researchers to mint NFTs on a number of blockchains on mainnet & testnet networks which can then be utilised during the security testing process. The framework will lower the time spent on development & security testing of the application.

The thesis will provide information about Blockchain, it's relation to Bitcoin & Ethereum, as well as introduce the topic of smart contracts, Non-Fungible Tokens and related security vulnerabilities and considerations. Afterwhich the focus will shift on currently available NFT minting solutions and their analysis. The last sections of this paper will focus on the proposed solution, and it's components.

1.2 Existing Body Of Knowledge

1.2.1 Introduction to Blockchain

Blockchains are tamper-evident and tamper-resistant digital ledgers that are distributed and typically decentralised (i.e., without a central authority like a bank, company, or government). At their most fundamental level, they enable a community of users to record transactions or data in a shared ledger within that community, with the result that no transaction can be modified once it has been published, as long as the blockchain network is operating normally [10]. In 2008, the blockchain concept was integrated with a number of other technologies and computer principles to produce modern cryptocurrencies: electronic cash that is protected by cryptographic processes rather than a central repository or authority [10]. Bitcoin was the first such blockchain-based cryptocurrency [12].

1.2.2 Bitcoin

The use of a blockchain enabled Bitcoin to be deployed in a distributed manner, ensuring that no single user controlled the electronic cash and that there was no single point of failure; this facilitated its adoption. Its key advantage was that it enabled direct transactions between users without requiring the involvement of a third party [10].

The Bitcoin blockchain is a:

- public
- decentralised
- peer-validated
- time-stamped

ledger which is distributed and publicly available to all participants that chronologically registers all validated transactions [11]. Transactions are broadcasted to the Bitcoin network, where they are independently confirmed by peers (also known as nodes). Valid transactions are aggregated into blocks that are cryptographically secured and sequentially interlocked: a chain of blocks [11][12]. Figure 1 illustrates a generic blockchain. A blockchain is usually a collection of data sets composed of a series of data packages (blocks) [10]. Within a blockchain network, cryptographic hash functions are used for many tasks, such as:

- Address derivation
- Creating unique identifiers.
- Securing the block data – a publishing node will hash the block data, creating a digest that will be stored within the block header.
- Securing the block header – a publishing node will hash the block header. If the blockchain network utilises a consensus model, the publishing node will need to

hash the block header with different nonce values until the puzzle requirements have been fulfilled. The current block header's hash digest will be included within the next block's header, where it will secure the current block header data [10].

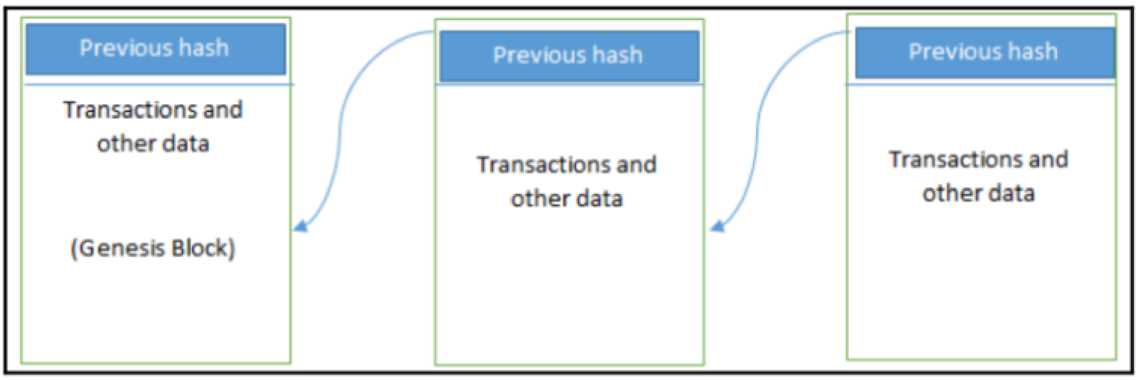

Previous hash
Transactions and other data
(Genesis Block)
Previous hash
Transactions and other data
Previous hash
Transactions and other data

Figure 2. Generic structure of Blockchain [1]

The network can validate blocks using cryptographic techniques. Along with the transactions, each block contains a timestamp, the hash value of the preceding block ("parent"), and a nonce, which is a random number used for hash verification. This approach protects the blockchain's integrity all the way back to the first block ("genesis block"). Because hash values are unique, fraud can be effectively prevented because any change to a block in the chain immediately alters the hash value associated with that block. If the majority of nodes in the network agree on the legitimacy of the transactions included in a block as well as the validity of the block itself via a consensus method, the block can be added to the chain [1][13]

In order for a number of nodes to agree on a transaction and broadcast it to the network, the nodes need to reach an agreement which is also known as consensus. Consensus is a technique used in blockchain to verify the chronological sequence in which requests, transactions (deploy and invoke), and information are executed, changed, or produced [11][14]. The proper sequencing is crucial since it establishes ownership and thus rights and obligations. On a blockchain network, there is no centralised hub or authority that determines the order of transactions, approves transactions, or establishes rules for how nodes communicate. Rather than that, numerous validating "peer" nodes implement the network consensus protocol, and all nodes have access to the information – restricted according to their permission level. As a result, the records are transparent and traceable [11].

Getting a group consensus dynamically requires group-based collaboration. Malicious individuals and flawed processes can disrupt this coordinated consensus. For example, a bad actor may create conflicting messages to cause group members to act in unison, reducing the group's ability to coordinate its activities [10]. When users join a blockchain network, they implicitly accept the system's initial state. The genesis block is the only one that comes pre-configured. Every blockchain network, regardless of the consensus paradigm used, has a published genesis block. Each block must be valid regardless of model, so that it may be independently validated by each blockchain network user [14]. User consensus can be reached by combining the capacity to verify the beginning state with the ability to check each subsequent block. In the majority of blockchain networks, the 'longer' chain is regarded as the proper one and will be adopted due to the amount of work invested in it. [14][15]. This problem is so called the “Byzantine Generals Problem” (BGP) [15].

1.2.3 Proof of Work

A user publishes the next block in the proof of work (PoW) model by being the first to solve a computationally expensive challenge. The solution to this conundrum is the "evidence" that they have worked. The challenge is created in such a way that solving it is complex but validating a solution is simple. This enables all remaining complete nodes to readily validate any proposed subsequent blocks, and any suggested block that does not meet the puzzle is rejected [10].

In Bitcoin, hashing is utilised for 'Proof of Work' (PoW), a process that connects consensus with computational power, thereby influencing the conclusion of consensus. The PoW is what so-called 'miners' do. In a nutshell, mining is a race amongst users to approve transactions. The probability of a user winning a competition is proportional to the amount of computer power he owns. Miners are compensated for their contributions to the process of block verification and building. This technique is the only way for the system to generate new Bitcoins [1][10][11].

1.2.4 Proof of Stake

The proof of stake (PoS) paradigm is based on the premise that the more stake a user has in a system, the more likely they are to want it to succeed and the less likely they are to want to subvert it. Stake is sometimes defined as the amount of cryptocurrency that a blockchain network user has invested in the system (through a variety of methods, including locking it via a special transaction type, delivering it to a specific address, or storing it in specialised wallet software). Once a coin is staked, it is generally not spendable. Proof of stake blockchain networks employ a user's stake to determine when new blocks are published. Thus, the likelihood of a blockchain network user publishing a new block is proportional to their stake in relation to the total quantity of staked cryptocurrency in the blockchain network [1][10].

The blockchain based on the Proof of Stake consensus model maintains a list of validators who have made a deposit. Anyone can become a validator by sending a particular form of transaction in which they deposit a specified quantity of their coins in a validator deposit. Each validator has a recognized identity, which is represented by a solid Ethereum address, and the network keeps track of all legitimate validators (those who have reserved funds for validation). Each validator can propose and validate new blocks via a consensus mechanism, which requires the set of validators to wager on the next block and vote in turn. The next block's validator is chosen based on the weight of voting multiplied by the size of each validator's stake. The likelihood of being chosen is proportional to their wager [14].

In a proof-of-work-based blockchain network, miners compete to be the first to solve the proof of work, as the first miner to do so wins the reward for adding the next block to the blockchain. This reward is made up of two components: the block generation reward and transaction fees. In comparison, PoS requires each qualified validator to place a bet on the block in order to qualify as a validator for the block. If the block is successfully attached, all validators will receive a payment corresponding to their stakes. Because there is no prize for block generation, validators (miners) are compensated by sharing the block's transaction fees in addition to the proportion of bets they place on the block. Although validators received small rewards for locking down their state, maintaining nodes, and taking additional precautions to secure their private key, the majority of the cost of reverting transactions comes from penalties that are thousands of times larger than the rewards they received in the interim. Thus, in contrast to proof of work, which provides "security from the benefits of burning computational energy," proof of stake provides "security from the penalties associated with putting up economic value-at-risk." [10][14].

In practice, software handles everything, and users are not required to be aware of these particulars. A key feature of blockchain technology is that no trusted third party is required to provide the system's state - every user within the system can verify the system's integrity. To add a new block to the blockchain, all nodes must eventually reach a consensus; however, some temporary disagreement is permitted. The consensus model for permissionless blockchain networks must work even in the presence of potentially malicious users, as these users may attempt to disrupt or take over the blockchain. It should be noted that in the case of permissioned blockchain networks, legal remedies may be used if a user acts maliciously.
There may be some level of trust between publishing nodes in some blockchain networks, such as permissioned blockchain networks. In this case, a resource-intensive (computation time, investment, etc.) consensus model to determine which participant adds the next block to the chain may not be required. In general, the need for resource usage as a measure of generating trust decreases as the level of trust increases. For some permissioned blockchain implementations, consensus encompasses the entire system of checks and validations from the proposal of a transaction to its final inclusion on a block, rather than just ensuring the validity and authenticity of the blocks [10].

The table below outlines a number of popular consensus models with a key comparison of goals, advantages and disadvantages. The list of consensus models is non-exhaustive, however for the purpose of keeping this paper in-line with the main topic and research questions, some consensus models were left out.

Name	Goals	Advantages	Disadvantages
Proof of	Provide a barrier to publishing blocks in the form of a computationally	Difficult to perform Denial Of Service attacks	Computationally intensive Potential for attack by obtaining enough

work (PoW)	difficult puzzle to solve to enable transactions between untrusted participants.	Open to anyone with hardware to solve the puzzle	computational power (>51%)
Proof of stake (PoS)	Enable less computationally intensive barrier to publishing blocks, but enable transaction between untrusted participants	Less computationally expensive than PoW Open to anyone who wishes to stake cryptocurrencies Stakeholders control the system	Stakeholders control the system Nothing to prevent formation of a pool of stakeholders to create a centralised power. Potential for 51 % attack by obtaining enough financial power.
Delegated PoS	To enable a more efficient consensus model through a 'liquid democracy' where participants vote (using cryptographically signed messages) to elect and revoke the rights of delegates to validate and secure the blockchain	Elected delegates are economically incentivized to remain honest More computationally efficient than PoW	Less node diversity than PoW or pure PoS consensus implementations Greater security risk for node compromise due to constrained set of operating nodes As all delegates are 'known' there may an incentive for block producers to collude and accept bribes, compromising the security of the system
Proof of Authority/ Identity	To create a centralised consensus process to minimise block creation and confirmation rate	Fast confirmation time Allows for dynamic block production rates Can be used in sidechains to blockchain networks which utilise another consensus model	Relies on the assumption that the current validating node has not been compromised Leads to centralised points of failure The reputation of a given node is subject to potential for high tail-risk as it could be compromised at any time.
Round Robin	Provide a system for publishing	Low computational power. Straightforward	Requires a large amount of trust amongst publishing

	blocks amongst approved/trusted publishing nodes	to understand.	nodes.
Proof of Elapsed Time (PoET)	To enable a more economic consensus model for blockchain networks, at the expense of deeper security guarantees associated with PoW.	Less computationally expensive than PoW	Hardware requirement to obtain time. Assumes the hardware clock used to derive time is not compromised

Table 1. Comparison matrix between PoW and PoS consensus models [10]

1.2.5 Ethereum & Smart Contracts

Ethereum was conceptualised by Vitalik Buterin in November 2013. The key idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and decentralised applications [16]. This is in contrast to bitcoin, where the scripting language is very limited and allows basic and necessary operations only.

Ethereum, like Bitcoin, uses the PoW consensus algorithm, which is likewise computationally expensive. To compensate for the cost of solving puzzles done by miners, Ether (ETH) is used instead of coins (like with BTC)[2]. Essentially *gas* serves as an internal price for executing a transaction to overcome the unstable value of ETH. Informally, the total cost of a transaction can be calculated by gas limit × gas price, where gas limit denotes the maximum amount of gas to be used to generate a block and gas price is the cost of a unit of gas (in ETH). Users can pay different amounts of gas to let their transactions be confirmed earlier or later (i.e., large amount of gas resulting in the fast confirmation) [2][16].

Vitalik Buterin published a paper on smart contracts in 2014, explaining the idea and potentiality behind smart contracts. The paper served as a catalyst, significantly increasing public interest in smart contract development, and essentially started an impulse wave, helping to make decentralised applications more available to the wider public.

The goal of Ethereum was to combine and improve on the concepts of scripting and on-chain meta-protocols, allowing developers to create arbitrary consensus-based applications with necessary scalability, standardisation, feature-completeness, ease of development, and interoperability. Ethereum accomplished this by constructing the abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralised applications with their own arbitrary rules for ownership, transaction formats, and state transition functions [16][17].

1.2.6 Smart Contracts

The advancements made by Ethereum introduced a new approach towards smart contracts. Smart contracts can be considered a significant advancement in the field of blockchain technology. Contractual clauses incorporated in smart contracts will be automatically enforced when a specified condition is met [2][16]. Essentially smart contracts are publicly available functions, with programmable conditions and executions. The written code is visible to all users on the blockchain and is transparent to anybody connected to the network. When the conditions are met within the specified time period, the contract is triggered to execute the digital transaction. Because the conditions are cryptographically encrypted, no party has the ability to alter the substance of a contract. Additionally, the immutability of blockchain means that every device connected to the network has a copy of the contract, ensuring a backup copy of the contract [4].

1.2.7 Smart Contract Platforms

Smart contracts have recently been developed on blockchain-based platforms. These platforms provide simple interfaces for developers to create smart contract applications. Many of the existing blockchain platforms are capable of supporting smart contracts. Besides Ethereum, the following are the four most prominent smart contract platforms: Hyperledger Fabric, Corda, Stella, and Rootstock [2].

- Hyperledger Fabric

Hyperledger Fabric is a distributed ledger technology framework for running smart contracts. Unlike Ethereum, which executes code in virtual machines (i.e., EVM), Hyperledger executes code in a Docker container. In comparison to virtual machines (VMs), containers can host smart contract applications at a lower cost without compromising isolation (i.e., applications in one container run on top of the same operating system). Fabric is compatible with most high-level programming languages, including Java and Go (aka Golang). Fabric is Turing complete and uses a key-value pair data model. Because Fabric's blockchain network is meant to serve typical enterprise applications, it is permissioned (private or consortium). Certificate authorities (CAs) are responsible for authorising users to connect to the network. After enrolling, the user must submit a request to the transaction certificate authority for transaction certificates (TCA)[2].

- Corda

In comparison to Ethereum's plethora of applications, Corda is specialised in digital-currency applications. It acts as a distributed ledger platform for storing and processing historical records of digital assets. Corda implements high-level programming languages such as Java and Kotlin6 on top of the Java Virtual Machine (JVM). Meanwhile, Corda is Turing incomplete, which makes verifiability impossible. Additionally, Corda's data model is transaction-based. Corda is primarily used to support private platforms, in which organisations create their own network for the purpose of exchanging digital assets privately. Rather than using blockchains to broadcast globally, Corda makes use of the point-to-point messaging technology. The message receivers and the particular information to be transmitted must be specified by the user [2].

- Stellar

As with Corda, Stellar is a dedicated platform for the development of digital currency apps. Stellar is more straightforward and approachable & supports a variety of programming languages, including Python, JavaScript, Golang, and PHP. Stellar contracts are not Turing complete. As with Fabric, Stellar executes computer code on top of Docker containers, minimising overhead. For instance, the cost of executing a single transaction at Stellar is only $0.0000002, which is virtually negligible. Additionally, the average time required to execute a transaction in Stellar is roughly 5 seconds, compared to 3.5 minutes in Ethereum. As a result, Stellar is an excellent platform for developing digital currency apps. Stellar's data model is account-based [2].

- Rootstock

Rootstock (RSK) runs on top of Bitcoin while supporting faster execution of transactions. RSK can validate the successful completion of a transaction within 20 seconds. RSK is compatible with Ethereum (for example, by utilising Solidity for contract implementation). Additionally, RSK developed its own virtual computers for the purpose of running smart contracts. RSK's data model is account-based, despite the fact that RSK is a public blockchain system. As with Corda and Stellar, RSK was first designed to support primarily digital money applications. RSK has a merit, namely that it is significantly safer than those systems that are not based on blockchains, as it is built on top of Bitcoin [2].

1.2.8 Token standards

Several Ethereum development standards focus heavily on token interfaces. These standards contribute to the continued composability of smart contracts, ensuring that when a new project issues a token, it remains compatible with existing decentralised exchanges [23].

1.2.9 ERC-20 Token Standard

The ERC-20 defines a standard for Fungible Tokens, which have the property of being identical (in type and value) to another Token. For instance, an ERC-20 Token behaves identically to ETH, which means that one Token is and will always be equal to all other Tokens.

Tokens on Ethereum can represent virtually anything, for example:

- reputation points on an online platform
- a character's skills in a game
- lottery tickets
- financial assets such as a company's stock
- a fiat currency such as the US dollar
- an ounce of gold.

By following ERC-20 token standard, the compatibility of the token on the network is ensured [21].

The ERC-20 standard, proposed in November 2015 by Fabian Vogelsteller, is a Token Standard that defines an API for tokens within Smart Contracts. The ERC-20 standard enables the following functions:

- transfer of tokens between accounts
- retrieval of an account's current token balance
- retrieval of the total supply of the token available on the network
- approval of whether an amount of token from an account can be spent by a third-party account.

If a Smart Contract implements the following methods and events, it is referred to as an ERC-20 Token Contract and, once deployed, it is responsible for keeping track of newly created Ethereum tokens [21][22].

1.2.10 ERC-721 Non-Fungible Token Standard

The ERC-721 standard, which was proposed in January 2018 by William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs, is a Non-Fungible Token Standard that defines an API for tokens used in Smart Contracts [9].

A Non-Fungible Token (NFT) is used to uniquely identify an object or person. This type of Token is suited for platforms that sell collectible items, access keys, lottery tickets, and numbered seats for concerts and sporting events, among other things. All NFTs have a unique 256 bits in size integer variable called tokenId. For any ERC-721 Contract, the contract address and tokenId must be globally unique. In a sense, ERC-721 is largely similar to ERC-20, with the former requiring and enforcing the uniqueness of each token, instead of all the tokens being equal [24].

Similar to ERC-20, ERC-721 standard enables the transfer of tokens between accounts, the retrieval of an account's current token balance, the identification of the owner of a specific token, and the total supply of the token available on the network. Apart from these, it also has some additional functionality, such as authorising the transfer of a certain amount of tokens from an account to a third-party account.

If a Smart Contract implements the following methods and events, it is referred to as an ERC-721 Non-Fungible Token Contract and, once deployed, it is responsible for managing the tokens created on Ethereum [9][24].

1.2.11 Non-Fungible Tokens

A Non-Fungible Token (NFT) is an ownership record stored on a blockchain (such as the Ethereum blockchain). While digital items, such as pictures and videos, are the most common assets traded as NFTs, the sale of physical assets, e.g., postal stamps, gold, real estate, physical artwork, etc., is also steadily gaining popularity. An NFT is the cryptocurrency equivalent of a traditional proof-of-purchase document, such as a paper invoice or an electronic receipt. NFTs are attractive for a variety of reasons, including their verifiability and trustless transfer. Verifiability refers to the fact that sales are recorded as blockchain transactions, which enables ownership tracking. Additionally, the NFT concept enables the trading of digital assets between mutually distrusting parties, as both the cryptocurrency payment and asset transfer occur atomically in a single transaction [8].

1.3 Research Gap

The identified gap in academic literature shows absence of materials regarding NFT vulnerabilities and NFT application security testing, as well as identified lack of available tools which supplement the security testing process of NFT applications.

1.4 Research Methodology and Research Questions

This paper follows the design science research methodology (DSR). DSR is a problem-solving paradigm that aims to advance human knowledge through the creation of novel solutions which are also called artefacts. DSR's mission is to advance technology and science knowledge bases by developing solutions that solve real-world problems and improve the environment in which they are instantiated. DSR produces both newly designed solutions and design knowledge (DK), which provides a more complete understanding of why the artefacts enhance (or, disrupt) the relevant application contexts via design theories [48]. The artefact in the context of this paper is a developed NFT minting application, which allows security testers, bug bounty hunters and researchers mint NFTs for a number of blockchains which then can be utilised during the security testing procedure.

Below in figure 1 is presented the core concept of Design Science Research framework which explains how Environment, IS Research and Knowledge base are connected. Figure 1 also provides a visualisation of DSR framework utilisation within the scope of this paper.

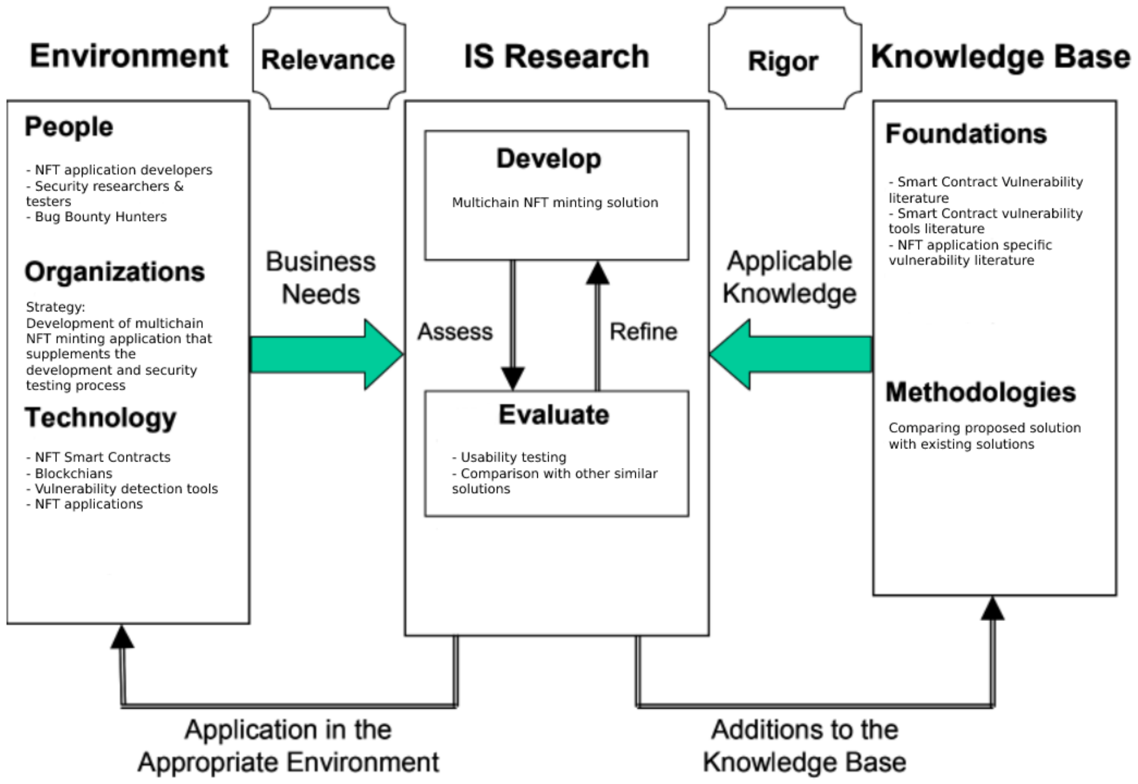

Environment
Relevance
IS Research
Rigor
Knowledge Base
People
- NFT application developers
- Security researchers & testers
- Bug Bounty Hunters
Organizations
Strategy: Development of multichain NFT minting application that supplements the development and security testing process
Technology
- NFT Smart Contracts
- Blockchians
- Vulnerability detection tools
- NFT applications
Business Needs
Develop
Multichain NFT minting solution
Assess
Refine
Evaluate
- Usability testing
- Comparison with other similar solutions
Applicable Knowledge
Foundations
- Smart Contract Vulnerability literature
- Smart Contract vulnerability tools literature
- NFT application specific vulnerability literature
Methodologies
Comparing proposed solution with existing solutions
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1. Design Science Research conceptual framework [48]

The main research question this paper aims to answer: **How to streamline secure implementation of NFT functionalities within multichain applications?**
In order to sufficiently answer the main research question, 3 additional research questions have been formulated:

- **Research question 1:**
  Q: What are common security vulnerabilities related to incorrect implementation of NFT smart contracts & NFT functions into web3 applications?

A: Answered by reviewing and analysing existing materials around NFT smart contract and NFT applications vulnerabilities.

- **Research question 2:**
  Q: What are the features of currently existing NFT minting solutions?
  A: Answered by reviewing and evaluating existing NFT minting platforms.

- **Research question 3:**
  Q: How can the security testing process of NFT functionalities within web3 applications be improved?
  A: Answered by developing a supplemental application for creating testing NFTs for a number of different Blockchains.

1.5 Thesis Structure

The structure of the thesis is as follows:

- Chapter 2 sets the preliminaries and describes the running case
- Chapter 3 provides information about NFT smart contract & application vulnerabilities and remediation, answering research question 1.
- Chapter 4 provides an evaluation of existing NFT minting applications, answering research question 2.
- Chapter 5 develops the multichain NFT minting application, answering research question 3.
- Chapter 6 evaluates achieved results, as well as discusses the limitations of the solution as well as future work possibilities
- Chapter 7 concludes the paper

2. Presuppositions

The following sections explain the running case and provide information about the multichain NFT minting application.

2.1 Running case

Recently NFTs have generated significant traction in scientific, industrial, commercial and a number of other industries. Because each Non-Fungible Token is unique and programmable, it can be used to identify, prove ownership, and establish the existence of digital assets, as well as serve as a certificate of authenticity [3][5][7][25].
Currently on the market, the majority of sold and used NFTs represent digital arts ownerships. As of April 2022, the current market capitalization of NFT is $10,275,018,910.35, with an all-time sales volume of $16,266,350,290.94 and a recent 24-hour sales volume of $42,382,321.34 [6].

A spike in popularity can partially be attributed to a rise in applications that involve NFTs. CryptoPunks, one of the first NFTs on Ethereum, created over 10,000 collectible images and aided in the mainstream adoption of the ERC-721 standard. CryptoKitties put NFTs on notice by introducing gamification mechanics in 2017. Participants fiercely competed for the rare collectibles, with the highest bid exceeding 999 ETH. Another noteworthy example is NBA Top Shot, an NFT trading platform for the purchase and sale of digital short videos of NBA moments. Thousands of NBA fans from around the world have amassed over $600 million in sales [3][28].

Through NFTs, buyers can now acquire ownership of tokenized creative works in any format (image, animation, video, or music). Additionally, it is pointless to circulate counterfeit or replica versions of that specific work of art, as any user can trace the work of art back to its rightful owner, confirming its authenticity and ownership. As a result, buyers of NFTs are protected from falling for sellers who falsely claim to sell authentic art. Previously, digital art had to be authenticated by an expert, which was time consuming and costly for both collectors and sellers. Authenticity is ensured with an NFT. Additionally, because artists can sell their work directly through an NFT platform, there is no need for an intermediary art gallery [28].

In the mainstream NFT projects, a cryptographic "hash" as the identifier, instead of a copy of the file, will be tagged with the token and then recorded on the blockchain to save the gas consumption. Several NFT projects integrate their system with a specialised file storage system such as IPFS in which IPFS addresses allow users to find a piece of content so long as someone somewhere on the IPFS network is hosting it. The data may become unavailable if the asset is stored on IPFS and the only node storing it is disconnected from the network [3].

The InterPlanetary File System (IPFS) is a permissionless, distributed, peer-to-peer file system. Anyone can become a member of the IPFS overlay network. A unique immutable address, also known as a content identifier (CID), is assigned to each data item d. This address is the hash of the file's content d. As a result, when the content of a file changes, the CID also changes. A file's content is first divided into blocks. All storage elements, including a directory, its files, and the blocks contained within those files, are stored in a directed acyclic graph structure called a Merkle DAG. IPFS uses a distributed hash table (DHT) distributed across the network's nodes to store provider records that identify peers that store the requested content. To obtain a data item d, a node searches the DHT for providers and then requests d from the members [8].

Apart from selling digital art, NFT technology use is expanding into new industries. Examples include:

- using Blockchain-based transaction systems for community energy infrastructure, where energy assets are implemented as NFTs [26]
- Secure anti-counterfeiting pharmaceuticals supply chain system using composable NFTs [29]
- Using NFTs for patents and intellectual property assets [27]
- Real estate tokenization [30]
- Credit Scoring [38]

With a strong growth potential, comes an increased reliance and necessity for secure implementation, operation and usage of NFT technologies within applications.

Due to their popularity and the amount of money flowing into the NFT market, NFT applications are constantly being targeted by malicious actors. Prominent examples of major hacks include:

- The Axie Infinity $625 million hack, where the attacker was able to use hacked private security keys to compromise the network nodes [31]

- TreasureDAO hack where approximately $1.4 million was stolen due to a smart contract bug [32]
- Critical flaw in OpenSea NFT marketplace, which allowed hijacking and stealing crypto accounts through the use of malicious NFTs [33]
- Another bug in OpenSea which allowed hackers to buy NFTs at a heavily discounted price [34]

With millions of dollars being lost & with interest in NFTs growing, the importance and need for proper NFT applications security is rapidly increasing. One of the key components ensuring application security is security testing.

2.2 Multichain NFT minting application

To simplify and facilitate the NFT application security testing, the thesis proposes a new application, which can be utilised during the security testing procedure. The application will take several inputs from the user:

1. Image(s) to be minted
2. ERC type (721 or 1155)
3. Recipients address
4. Network

In order for the presented applications use-case to be as all-encompassing as possible, the application will offer to mint NFTs for a number of chains:

1. Ethereum (ETH)
    a. mainnet
    b. Ropsten testnet
2. Binance Smart Chain (BSC)
    a. mainnet
    b. testnet
3. Solana (SOL)
    a. mainnet
    b. devnet
    c. testnet
4. Avalanche (AVAX)
    a. mainnet

    b. testnet
5. Polygon (MATIC)
    a. mainnet
    b. testnet
6. Fantom (FTM)
    a. mainnet
    b. testnet

The minted NFTs can then be used for security and application functionality testing.

3. Non-Fungible Tokens applications security

This section discusses various types of vulnerabilities that are related to Smart Contract development. For the showcasing and examples of the vulnerabilities, Solidity language is used. Measures for remediations are discussed as well.

3.1 Introduction

The section answers research question 1:
What are common security vulnerabilities related to incorrect implementation of NFT smart contracts & NFT functions into web3 applications?
For vulnerabilities related to smart contracts, the chosen language is Solidity. For the overview of the vulnerabilities the DASP TOP 10 taxonomy is used.
For vulnerabilities relating specifically to NFT, academic literature is used as the source of information.

The security of an NFT application can be divided into two main parts:

1. Smart Contract security
    a. The first component of NFT security is the protection of the application's smart contracts. This code is stored on the blockchain and is publicly accessible. If this code contains security vulnerabilities, they can be exploited by anyone who discovers them. These contracts contain the sensitive data and assets that must be safeguarded. The security of Smart

Contracts can be enhanced by employing formal verification techniques to establish that the code does what it is supposed to do.

2. NFT application security
    a. The second part of NFT security is the security of the application itself. This includes the server infrastructure, resilient functional operability

3.2 Smart Contracts

3.2.1 Smart Contracts

Smart contracts enable creation of decentralised applications where one can create their own arbitrary rules for ownership, transaction formats and state transition functions. Developers, in particular, can specify access permissions for each contract function. When a condition in a smart contract is satisfied, the triggered statement will automatically and predictably perform the relevant function [16].

Using smart contracts, a number of decentralised applications have been built, also referred to as DApps. DApps are open-source applications based on the Ethereum blockchain where a consensus is maintained between the user and programmer during the development process. The source code is available for examination and the application is stored in the blockchain to ensure trust and transparency [4]. Smart contracts being open-source introduce new kinds of vulnerabilities. If developers do not introduce proper access- and permission-controls, malicious actors are able to interact with the smart contracts and exploit it according to their objectives [18].

3.2.2 Smart Contract Platforms

Smart contracts have recently been developed on blockchain-based platforms. These platforms provide simple interfaces for developers to create smart contract applications. Many of the existing blockchain platforms are capable of supporting smart contracts. Besides Ethereum, the following are the four most prominent smart contract platforms: Hyperledger Fabric, Corda, Stella, and Rootstock [2].

- Hyperledger Fabric

Hyperledger Fabric is a distributed ledger technology framework for running smart contracts. Unlike Ethereum, which executes code in virtual machines (i.e., EVM), Hyperledger executes code in a Docker container. In comparison to virtual machines (VMs), containers can host smart contract applications at a lower cost without compromising isolation (i.e., applications in one container run on top of the same operating system). Fabric is compatible with most high-level programming languages, including Java and Go (aka Golang). Fabric is Turing complete and uses a key-value pair data model. Because Fabric's blockchain network is meant to serve typical enterprise applications, it is permissioned (private or consortium). Certificate authorities (CAs) are responsible for authorising users to connect to the network. After enrolling, the user must submit a request to the transaction certificate authority for transaction certificates (TCA)[2].

- Corda

In comparison to Ethereum's plethora of applications, Corda is specialised in digital-currency applications. It acts as a distributed ledger platform for storing and processing historical records of digital assets. Corda implements high-level programming languages such as Java and Kotlin6 on top of the Java Virtual Machine (JVM). Meanwhile, Corda is Turing incomplete, which makes verifiability impossible. Additionally, Corda's data model is transaction-based. Corda is primarily used to support private platforms, in which organisations create their own network for the purpose of exchanging digital assets privately. Rather than using blockchains to broadcast globally, Corda makes use of the point-to-point messaging technology. The message receivers and the particular information to be transmitted must be specified by the user [2].

- Stellar

As with Corda, Stellar is a dedicated platform for the development of digital currency apps. Stellar is more straightforward and approachable & supports a variety of programming languages, including Python, JavaScript, Golang, and PHP. Stellar contracts are not Turing complete. As with Fabric, Stellar executes computer code on top of Docker containers, minimising overhead. For instance, the cost of executing a single transaction at Stellar is only $0.0000002, which is virtually negligible. Additionally, the average time required to execute a transaction in Stellar is roughly 5 seconds, compared to 3.5 minutes in Ethereum. As a result, Stellar is an excellent platform for developing digital currency apps. Stellar's data model is account-based [2].

- Rootstock

Rootstock (RSK) runs on top of Bitcoin while supporting faster execution of transactions. RSK can validate the successful completion of a transaction within 20 seconds. RSK is compatible with Ethereum (for example, by utilising Solidity for contract implementation). Additionally, RSK developed its own virtual computers for the purpose of running smart contracts. RSK's data model is account-based, despite the fact that RSK is a public blockchain system. As with Corda and Stellar, RSK was first designed to support primarily digital money applications. RSK has a merit, namely that it is significantly safer than those systems that are not based on blockchains, as it is built on top of Bitcoin [2].

3.3 Smart Contract Vulnerabilities

Extensive research has been done around smart contract security and related vulnerabilities [2][4][19][20][35]. One prominent example is Decentralised Application Security Project (or DASP) Top 10 taxonomy, which aims to provide a high-level overview of most wide-spread smart contract vulnerabilities by using a similar approach employed by OWASP Top 10.

3.3.1 Reentrancy Attacks

This attack method is capable of completely destroying the contract or stealing vital information. Reentrancy can occur when a function makes an outer call to another contract. The exploit enables an attacker to call the main function recursively, resulting in an unintended loop that is repeated numerous times. For example, if a vulnerable contract contains a revoke function, the contract may call the revoke function repeatedly in order to drain the contract's available balance. Single function Reentrancy attacks and cross-function Reentrancy attacks are two different types that can be exploited by the attackers. The exploitation allows the attacker to use external calls to execute the desired tasks.

The first version of this bug that was discovered involved functions that could be called repeatedly before the function's initial execution completed. This may result in destructive interactions between the function's various invocations.

```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
// At this point, the caller's code is executed, and can call
withdrawBalance again
    require(success);
    userBalances[msg.sender] = 0;
}
```

Listing 1. Solidity code susceptible to single function Reentrancy attack

Due to the fact that the user's balance is not reset to zero until the function is completed, the second (and subsequent) invocations will continue to succeed and will withdraw the balance repeatedly.

In the example given, the best way to avoid this attack is to ensure that you do not call an external function until you have completed all necessary internal work:

```
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0;
    (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
    require(success);
}
```

Listing 2. Solidity code with applied fix for single function Reentrancy attack

Additionally, an attacker may be able to conduct a similar attack by utilising two distinct functions that share the same state. This is also known as Cross-function Reentrancy attack.

```
mapping (address => uint) private userBalances;
function transfer(address to, uint amount) {
    if (userBalances[msg.sender] >= amount) {
       userBalances[to] += amount;
       userBalances[msg.sender] -= amount;
    }
}
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
// At this point, the caller's code is executed, and can call
transfer()
    require(success);
    userBalances[msg.sender] = 0;
}
```
Listing 3. Solidity code vulnerable to cross function Reentrancy attack

When the attacker's code is executed on the external call in withdrawBalance, they call `transfer()`. Due to the fact that their balance has not yet been reset to zero, they can transfer the tokens despite the fact that they have already received the withdrawal.

3.3.2 Flawed Access Control

Typically, a smart contract's functionality is accessed via its public or external functions. While insecure visibility settings make it relatively easy for attackers to gain access to a contract's private values or logic, access control bypasses can be more subtle. These vulnerabilities can occur when contracts use the deprecated `tx.origin` instead of `msg.sender` to validate callers or when large authorization logic is handled via lengthy `require` function. In the example below, an `onlyowner` identifier is defined, but not used.

```
contract Unprotected{
    address private owner;
    modifier onlyowner {
        require(msg.sender==owner);
        _;
    }
    function changeOwner_vulnerbale(address _newOwner)
        public
    {
       owner = _newOwner;
    }
    function changeOwner_fixed(address _newOwner)
        public
        onlyowner
    {
       owner = _newOwner;
    }
}
```
Listing 4. Example 1 of vulnerable and fixed access control function [36]

A frequently used pattern for granting special privileges, is to determine the smart contract's owner by the address that initialises it. Unfortunately, due to the nature of Blockchain, anyone can call the initialization function — even after it has already been called. Permitting anyone to acquire ownership of a contract and withdraw its funds.

```
function initContract() public {
      owner = msg.sender;
}
```
Listing 5. Solidity code example with a flawed access control logic

The flaw in the example in Listing 5 is that the function `initContract()` does not perform verification if the contract had already been initialised

3.3.3 Arithmetic Issues

Also known as integer overflow and integer underflow attacks. This vulnerability is relatively straightforward to exploit and occurs when transactions accept unauthorised input data or value. Overflows in smart contracts occur most frequently when more value is provided than the maximum value. Because the contracts are primarily written in Solidity, which supports up to 256-bit numbers, an increment of one would result in an overflow.

```
function withdraw(uint _amount) {
      require(balances[msg.sender] - _amount > 0);
      msg.sender.transfer(_amount);
      balances[msg.sender] -= _amount;
}
```

Listing 6. Solidity code with underflow vulnerability

The `withdraw()` function of a smart contract allows to retrieve ether donated to the contract as long as the contract's balance remains positive following the operation. An attacker makes an attempt to withdraw more money than he or she currently has. The result of the `withdraw()` function check is always positive, allowing the attacker to withdraw more than is permitted. The resulting balance underflows and increases by an order of magnitude.

3.3.4 Unchecked Return Values For Low Level Calls

The low-level functions `call()`, `callcode()`, `delegatecall()`, and `send()` are some of the more advanced features of Solidity. Their error-handling behaviour is quite distinct from that of other Solidity functions, as they do not propagate (or bubble up) and do not result in a complete reversion of the current execution. Rather than that, they will return a false boolean value and the code will continue to run. This can catch developers off guard and, if the return value of such low-level calls is not checked, can result in failed opens and other undesirable outcomes.

```
function withdraw(uint256 _amount) public {
      require(balances[msg.sender] >= _amount);
      balances[msg.sender] -= _amount;
      etherLeft -= _amount;
      msg.sender.send(_amount);
}
```

Listing 7. Vulnerable code, where the result of `send(_amount);` is not checked

3.3.5 Denial of Service

This type of vulnerability includes such gas limit reached, unexpected throw, unexpected kill, access control breached While other types of applications can eventually recover from denial of service attacks, smart contracts can be taken offline permanently by just one of these attacks. Denials of service can occur in a variety of ways, including maliciously behaving as the recipient of a transaction, artificially increasing the amount of gas required to compute a function, abusing access controls to gain access to private components of smart contracts, exploiting mixups and negligence, and so on.

```
// INSECURE
contract Auction {
    address currentLeader;
    uint highestBid;

    function bid() payable {
        require(msg.value > highestBid);

        require(currentLeader.send(highestBid)); // Refund the old
leader, if it fails then revert

        currentLeader = msg.sender;
        highestBid = msg.value;
    }
}
```

Listing 8. Solidity code example with unexpected revert DoS vulnerability [37]

If an attacker bids using a smart contract that includes a fallback function that cancels any payment, the attacker has the ability to win any auction. When it attempts to refund the previous leader, it reverts if the refund is unsuccessful. This means that a malicious bidder can rise to the top position while ensuring that refunds to their address will always fail. This way, they can prevent others from calling the `bid()` function, ensuring that they remain the leader indefinitely [37]

Another instance is when a contract iterates over an array in order to pay users (e.g., supporters in a crowdfunding contract). It's natural to want to ensure that each payment is successful. Otherwise, one should revert. The issue is that if one call fails, the entire payout system is reverted, which means the loop will never complete. Nobody is paid because a single address forces an error [37].

```
address[] private refundAddresses;
mapping (address => uint) public refunds;

// bad
function refundAll() public {
    for(uint x; x < refundAddresses.length; x++) { // arbitrary
length iteration based on how many addresses participated
require(refundAddresses[x].send(refunds[refundAddresses[x]])) //
doubly bad, now a single failure on send will hold up all funds
    }
}
```
Listing 9. Code example with unexpected revert and Gas Limit DoS vulnerability

By paying everyone at once, there is a risk of exceeding the block gas limit. This can create complications even in the absence of a deliberate attack. However, this is particularly dangerous if an attacker is able to manipulate the amount of gas required. In the preceding example, the attacker could add a large number of addresses, each of which would require a very small refund. As a result, the gas cost of refunding each of the attacker's addresses may exceed the gas limit, effectively blocking the refund transaction.

3.3.6 Bad Randomness

While Solidity provides functions and variables that allow for the access of apparently unpredictable values, they are typically more public than they appear or are subject to miner influence. Due to the predictability of these sources of randomness, malicious users can generally replicate them and attack the function based on their unpredictability.

Example:

- A smart contract makes use of the block number to generate random numbers for a game.
- A malicious contract is created by an attacker that checks to see if the current block number is a winner. If that is the case, it will call the first smart contract in order to win; because the call will be part of the same transaction, the block number on both contracts will remain the same.
- The attacker needs only to invoke her malicious contract until it is victorious.

In the example below, `block.blockhash` is being used to generate a random number. This hash is unknown if the `blockNumber` is set to the current `block.number` (for obvious reasons), and is thus set to 0. In the case where the `blockNumber` is set to more than 256 blocks in the past, it will always be zero. Finally, if it is set to a previous block number that is not too old, another smart contract can access the same number and call the game contract as part of the same transaction.

```
function play() public payable {
      require(msg.value >= 1 ether);
      if (block.blockhash(blockNumber) % 2 == 0) {
            msg.sender.transfer(this.balance);
      }
}
```

Listing 10. Example of contract which has bad random generation mechanism

3.3.7 Front Running

Also known as time-of-check vs time-of-use (TOCTOU), race condition, transaction ordering dependence (TOD). Due to the fact that miners are always compensated in the form of gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Due to the public nature of the Ethereum blockchain, anyone can view the contents of other people's pending transactions. This means that if a user discloses the solution to a puzzle or other valuable secret, a malicious user can steal the solution and replicate it with higher fees in order to preempt the original solution. Without caution on the part of smart contract developers, this situation can result in practical and devastating front-running attacks.
Example:

- A smart contract makes available an RSA key (N = prime1 x prime2).
- A call to its public function `submitSolution()` with the correct prime1 and prime2 parameters rewards the caller.
- Alice factors the RSA number successfully and submits a solution.
- Someone on the network notices Alice's pending transaction (which contains the solution) and submits it with a higher gas price.
- Due to the higher fee paid, the second transaction is prioritised by miners. The attacker is the winner of the prize.

3.3.8 Time Manipulation

Contracts rely on the current time for a variety of reasons, from locking a token sale to unlocking funds at a specified time for a game. This is typically accomplished using `block.timestamp` or its alias `now` in Solidity. However, that value originates from the miners. Because the miner of a transaction has some leeway in reporting the time of mining, good smart contracts will avoid relying heavily on the advertised time. Notably, as discussed in the Bad Randomness vulnerability overview, `block.timestamp` is occasionally (mis)used in the generation of random numbers.

The following function accepts only calls received on or after a specified date. Due to the fact that miners have some control over their block's timestamp, they can attempt to mine a block containing their transaction with a block timestamp set in the future. If it is close enough, the transaction will be accepted by the network, and the miner will receive ether before any other player attempts to win the game.

```
function play() public {
      require(now > 1521763200 && neverPlayed == true);
      neverPlayed = false;
      msg.sender.transfer(1500 ether);
}
```

Listing 11. Vulnerable Solidity code due to incorrectly implemented reliance on timestamp

3.3.9 Short Addresses Attack

Also known as or related to off-chain issues, client vulnerabilities. Short address attacks are an unintended consequence of the EVM accepting improperly padded arguments. Attackers can take advantage of this by using specially crafted addresses to cause insecure clients to incorrectly encode arguments prior to including them in transactions. While this vulnerability has not been exploited in the wild, it is an excellent demonstration of the issues that can arise when clients interact with the Ethereum blockchain. Other off-chain issues exist, the most significant of which is the Ethereum ecosystem's strong reliance on specific Javascript front ends, browser plugins, and public nodes.

Example:

- A trading function is included in an exchange API that accepts a recipient address and a transaction amount.

- The API then uses padded arguments to interact with the smart contract `transfer(address _to, uint256 _amount)` function: it prepends the address (which is expected to be 20 bytes long) with 12 zero bytes to make it 32 bytes long.
- Bob (`0x3bdde1e9fbaef2579dd63e2abbf0be445ab93f00`) requests 20 tokens from Alice. He maliciously provides her with an address that has been truncated to eliminate the trailing zeroes.
- Alice accesses the exchange API using Bob's shorter 19-byte address (`0x3bdde1e9fbaef2579dd63e2abbf0be445ab93f`).
- The API appends 12 zero bytes to the address, making it 31 bytes rather than 32 bytes. Effectively stealing one byte from the `_amount` argument that follows.
- At some point, the EVM that executes the smart contract's code will notice that the data is not properly padded and will append the missing byte to the end of the `_amount` argument. Effectively transferring 256 times the number of tokens anticipated.

An effective remediation of such vulnerability would include performing additional validations, to check if the provided address is 20 bytes long:

```
function isAddress(address toTest) internal constant returns (bool)
{
    require(bytes(toTest).length == 20);
}
```

Listing 12. Solidity code with address length verification

3.3.10 Unknown Unknowns

Solidity, the primary language for developing smart contracts, has not yet reached a stable version, and the ecosystem's tools remain experimental. Some of the most damaging smart contract vulnerabilities surprised everyone, and there is no reason to believe that another will not be equally surprising or destructive. Although methods for formally verifying smart contracts are not mature, they appear to hold great promise as a way to move beyond today's precarious status quo. As new classes of vulnerabilities are discovered, developers must remain vigilant, and new tools must be developed to discover them before the bad guys do.

3.4 Prevention And Remediation Of Smart Contract Vulnerabilities

The list of issues mentioned previously includes only the most common vulnerabilities found in smart contracts, and manually resolving such issues is frequently difficult, especially when the application or protocol contains a sizable amount of code. Fortunately, a variety of tools & standards are available to assist in the prevention, detection and remediation of smart contract vulnerabilities.
This section will include information about most widely used security analysis tools for identifying vulnerabilities in smart contracts, as well as include information about Smart Contract Weakness Classification Registry and secure development libraries.

3.4.1 Static and dynamic analysis

Static and dynamic analysis tools perform automated security scans of the smart contracts attempting to identify vulnerabilities. Below are presented 5 most popular smart contract security scanning tools as of April 2022.

- Slither

Slither can be used to automate a variety of tasks, including vulnerability detection, optimization detection, code comprehension, and assisted code review. The security analysis is initiated via a multi-stage procedure. From the contract source code, the Solidity compiler generates a Solidity Abstract Syntax Tree, which is used as an input to Slither. Slither obtains critical contract information during the initial stage, such as the inheritance graph and control-flow graph. The following stage entails converting the entire codebase to SlithIR. The following stage performs the task of code analysis by computing a list of predefined analyses [4][41].

- MythX

Primary objective of MythX is to assist DApp developers in developing smart contracts in order to ensure a more secure platform. MythX does not fulfil the requirements on its own; it is used in conjunction with development tools such as Truffle and Remix. It is not limited to developers working on the Ethereum platform; developers working on Tron, Vechain, Quorum, Roostock, and a few other EVM-based platforms can also use these security tools to identify bugs in smart contracts. MythX analyses smart contract code in three stages. To begin, it requires developers to submit their code; second, it activates a comprehensive suite of analysis techniques; and finally, it generates an analysis report indicating whether any errors exist [4][39].

- Mythril

Mythril is a security tool that analyzes smart contracts written by Solidity. Mythril, an open-source tool, takes advantage of the symbolic execution technique in order to determine the errors in code. The examination of security flaws involves executing smart contract bytecode in a custom built EVM. Mythril goes through four major working stages to accomplish its security analysis. When a flaw in a program is discovered, the input transactions are analysed to determine the possible reasons. This security method helps to deduce the main cause of the program vulnerability, and also mitigate exploitation. If a developer produces the source code of the contract, Mythril is able to locate the bugs within the code [4][40].

- Manticore

Manticore's primary functions include tracing inputs that terminate a program, logging implementation at the instruction level, and providing access to its analysis engine via a Python API. It includes a dynamic symbolic execution capability for analysing binaries and Ethereum smart contracts. Manticore's architecture is composed of three primary components: the Core Engine, Native Execution Modules, and Ethereum Execution Modules. Secondary attributes include the Satisfiability Modulo Theories module, the Event System, and the API [4][43].

- Securify

Securify 2.0 is a security scanner for Ethereum smart contracts supported by the Ethereum Foundation and ChainSecurity. The core research behind Securify was conducted at the Secure, Reliable, and Intelligent Systems Lab at ETH Zurich. Securify performs security analysis on EVM bytecode. Contracts written in Solidity are also accepted as inputs; however, the code must be compiled to EVM bytecode in order to effect the security process. When a security violation occurs, Securify generates a command that causes the pattern of the violation to match. Similarly, it generates a warning when the violation and compliance pattern do not match. Securify's security analysis technique is unique in comparison to other tools such as Mythril. While

Mythril symbolically enumerates the distinct paths of a contract, Securify analyzes each path of the smart contract using static analysis [4][42].

3.4.2 Smart Contract Weakness Classification

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the EIP-1470 proposed scheme for classifying weaknesses in smart contracts. It adheres loosely to the terminology and structure of the Common Weakness Enumeration (CWE) while incorporating a wide variety of weakness variants unique to smart contracts [45]. The SWC registry presents the reader with a description of the vulnerability, as well as with examples of vulnerable solidity code combined with proper remediation methods [44].

3.4.3 Development libraries

Development libraries allow developers to use modular and reusable contracts within their applications. One of the most prominent, and currently most popular examples is OpenZeppelin which is used by companies like Ethereum Foundation and Coinbase [46]. OpenZeppelin allows developers to utilise pre-built smart contracts, which follow top security patterns and practises, while offering easy-to-use robust code. Offered contracts are open source, allowing security researchers and developers to scrutinise and look for vulnerabilities [47].

3.5 NFT Application vulnerabilities

Most of the currently popular NFT applications are NFT marketplaces, where users can create, buy & sell various art forms by utilising Non-Fungible Tokens as a proof of ownership. Though more advanced NFT-related applications are still yet to achieve widespread adoption and usage, their core backbone functionality will still need to remain similar, due to the intended use, limitations and standards of Non-Fungible Token smart contracts. The sections below will provide information about core NFT application functionalities, as well as give a number of security and precautionary advice

3.5.1 Token minting/issuing

Creation of NFTs, also known as *minting*. Examples include: issuing a proof of digital identity, proof of asset ownership and more.

Advised safety precautions:

- Verifiability of token contracts
    - Prior to minting NFTs, the source code of the underlying token contract should ideally be made available for public inspection to ensure that they are neither malicious nor flawed.
- Prevention of token metadata tampering
    - Utilisation and enforcement of IPFS for token metadata hosting [8]

3.5.2 User authentication

Authentication of users, either via web3 interface using a supported web3 browser add-on like Metamask, or via usual login & password form.

Advised safety precautions:

- Mandatory 2 Factor Authentication
- Identity verification using Know-Your-Customer procedure [8]

3.5.3 Token listing and/or transacting of tokens

Especially relevant to various types of NFT applications where tokens are used as a proof of ownership and the intended functionality allows for transactions or transfers to take place. The application should provide an ability for secure transactions of NFTs.

Advised safety precautions:

- Principle of least privilege

When listing an NFT for sale or preparing for the participation in a transaction, the NFT platform ideally assumes control of the token so that, upon completion of a sale, it can transfer ownership of the NFT to the buyer. In order to accomplish this, the platform must be **either the owner** of the NFT (i.e., the current owner must transfer the asset to an escrow account during listing), **a controller** where an Ethereum account can manage that specific NFT on behalf of the owner, or **an operator** where an Ethereum account can manage all the NFTs in that collection. The escrow model is risky because all assets being traded on the platform are held in a single escrow contract/wallet managed by the platform. Therefore, the security of all assets within a marketplace is contingent upon the security of the escrow contract or the external account that manages such a contract.

This design violates the principle of least privilege significantly. Therefore, a flaw in the contract or the disclosure of the private key of the external account could compromise the security of all NFTs stored. Adopting a proxy contract deployed by the platform, which then becomes the controller or operator of the entire NFT collection, would be a safer alternative. According to the terms of the contract, the platform can only transfer an NFT after it has been sold and the required amount has been paid to the seller. This safeguards the NFT token even in the event of a marketplace hack. If the private key of a seller (owner of an NFT) is compromised, it can only compromise the security of that particular NFT or collection, as opposed to all NFTs in the escrow model [8].

- Sufficient transparency

Asset ownership records for NFTs should be stored on the blockchain to enable public verification. In a decentralised environment, an NFT transaction is managed by the platform's contract, which invokes the token transfer contract to transfer the token from transaction participant 1 to participant 2. On the blockchain, every transaction and its associated transfer should be visible. Each transaction should include the following information, among others:

- The address of the participant 1 (the current owner)
- The address of the participant 2 (the new owner)
- The time of the transfer of ownership
- Incase of a sale of the NFT, the price at which the NFT was sold

The ERC-721 `ownerOf()` API, which returns the current owner of a token, has simplified the process of determining ownership. In conjunction with the API, the sales records allow one to reconstruct the precise sales and ownership history of an NFT. If sales records and transactions are stored off chain, however, it becomes impossible to verify any trades and the ownership history of an NFT. In addition, a malicious NFT platform can exploit this fact to create fictitious sales records in order to inflate trading activity and volume. Off-chain records are susceptible to modification, censorship, and deletion if the NFTM database is unavailable [8].

Faulty implementation of NFT functionality can lead to issues with authenticity, integrity, confidentiality & availability among a number of other things [3]. Table 2 presents an aggregated overview of potential vulnerabilities, as well as proposes possible solutions to the mentioned issues using STRIDE threat and risk evaluation, which covers security aspects of a system: authenticity, integrity, non-repudiability, availability and access control [3].

STRIDE	Security Issues	Solutions
Spoofing (Authenticity)	● An attacker may exploit authentication vulnerabilities ● An attacker may steal a user's private key.	● A formal verification on the smart contract. ● Using the cold wallet to prevent the private key leakage.
Tampering (Integrity)	● The data stored outside the blockchain may be manipulated.	● Sending both the original data and hash data to the NFT buyer when trading NFTs.
Repudiation (Non-repudiability)	● The hash data may bind with an attacker's address.	● Using a multi-signature contract partly.
Information disclosure (Confidentiality)	● An attacker can easily exploit the hash and transaction to link a particular NFT buyer or seller.	● Using privacy-preserving smart contracts instead of smart contracts to protect the user's privacy.
Denial of service (Availability)	● The NFT data may become unavailable if the asset is stored outside the blockchain.	● Using the hybrid blockchain architecture with a weak consensus algorithm.
Elevation of privilege (Authorization)	● A poorly designed smart contract may make NFTs lose such properties.	● A formal verification on the smart contracts.

Table 2. Potential Security Issues and Corresponding Solutions of NFTs [3]

When the implemented functionality has flawed logic & proper validation is missing, malicious actors can leverage gaps in functional logic to attempt to exploit the application without using any technical vulnerabilities.

One of the potential ways for simplifying the remediation of such vulnerabilities is by utilising code visualisation which will aid in detecting faulty EVM bytecode, smart contracts, and their control flow graphs.

An list of currently popular visualisation solutions is presented below:

- Solgraph - Creates a DOT graph that illustrates the function control flow of a Solidity contract and highlights potential security flaws [49]
- ethereum-graph-debugger - A graphical debugger for the EVM. The entire program's control flow graph is displayed [50]
- Solidity Visual Auditor - This extension enhances Visual Studio Code with security-focused syntax and semantic highlighting, a detailed class outline, and advanced Solidity code insights [52].

When verifying NFT applications functional requirements, one key part is manual testing of the NFT application/solution to ensure that the functional flow is working as intended. Mentioned visualisation tools can help with outlining more precise flows that need to be verified.

3.6 Conclusion

This section answers the research question 1: **What are common security vulnerabilities related to incorrect implementation of NFT smart contracts & NFT functions into web3 applications?**

The question was answered by employing the DASP TOP 10 taxonomy and SWC registry. The common smart contract vulnerabilities are: (i) Reentrancy attack, (ii) Access Control, (iii) vulnerabilities related to arithmetics, (iv) Unchecked Return Values for low level calls, (v) Denial of Service, (vi) Bad Randomness, (vii) Front Running, (viii) Time Manipulation, (ix) Short Address attack and (x) unknown unknowns. The most common NFT application vulnerabilities include issues that relate to token minting/issuing, user authentication and token transacting.

Employed approach has its limitations, by mainly employing the DASP 10 taxonomy, which only covers a smaller subset, yet most popular types of attacks. The variety of vulnerabilities is vastly greater than presented in this paper. One more limitation factor is the fact that the vulnerable code examples are presented in Solidity, which is not applicable in other blockchains such as Solana and require their own code examples.

4. Assessment of Currently Available Applications For Creating Non-Fungible Testing Tokens

This section will present an overview of currently available minting NFT applications, afterwhich an evaluation will be made based on a number parameters which are necessary for an efficient security testing process.

4.1 Introduction

This section will answer research question 2:

What are the features of currently existing NFT minting solutions?

This section will evaluate 10 available NFT minting services & tools. Will explain why it is efficient to use NFT minting solutions instead of minting NFTs manually when testing multi chain NFT applications, as well as evaluate currently offered NFT minting solutions and analyse the platform's usability during a security testing process.

4.2 Minting NFTs Using Smart Contracts

When testing NFT applications, having test NFTs to utilise during the security testing procedure is necessary. One way to mint NFTs is by utilising smart contracts directly, however a strong drawback is the necessary preliminary system setup and steps that need to be taken before the NFT can be seen and used.

For example for minting an NFT using on Ethereum blockchain it is necessary to [53]:

1. Install web3 library on the system
2. Create a mint-nft.js script
3. Retrive the contract ABI
4. Configure the metadata for the NFT using IPFS

5. Create an instance of the NFT smart contract
6. Edit the .env file
7. Create the transaction
8. Sign the transaction

To mint an NFT on Avalanche network [54]:

1. Create the NFT Family using `avm.createNFTAsset method` which is a method of the X-Chain's API
2. Retrieve UTXOs for NFT
3. Mint the Asset
4. Send the NFT

For different blockchains respective steps necessary for manually minting an NFT are different, required preliminary setup varies and in general the process consumes time, which could be spent on the security testing. From this comes a need for ready NFT minting solutions which allow to mint NFTs directly using a GUI.

4.3 Evaluation Of Currently Available NFT Minting Solutions

Currently there are a number of NFT minting applications available. To analyse their applicability during a security testing process of a multichain NFT application, a number of NFT minting solutions were analysed based on the following criteria:

1. **IPFS image upload support**
   a. Does the NFT minting application upload NFTs directly to IPFS or does the application store them in the off chain environment?
   b. Ideally the application should automatically upload the image to IPFS to simulate real world environment and create NFTs securely
2. **Required number of steps & actions needed before the actual mint of the NFT takes place**
   a. Ideally steps like authentication, registration etc should be kept to a minimum or be not present at all, to maximise the simplicity of the NFT minting when NFTs are intended to be used during security testing, and not for sale.
3. **Supported mainnet networks**

   a. How many mainnet networks are supported?
   b. Ideally, the application should support a number of mainnet chains to support the “multichain” statement.
4. **Supported testnet networks**
   a. How many testnet networks are supported?
   b. Ideally the application should support a number of testnet chains to be usable during the testing phase of the multichain NFT application
5. **NFT minting transaction data providing**
   a. Does the application provide data about the transaction hash?
   b. Ideally, supplemental transaction data should be provided for the manual verification of the minted NFT
6. **Supported NFT types**
   a. Does the application support both ERC721 & ERC1155 token types?
   b. Considering the changing nature of the blockchain industry, both token types should be supported in order for the application to have a wide range of applicability

Upon initial analysis of the available NFT minting applications, it was found that most applications were over-fixated on NFT art sales, with the corresponding functionality being tailored towards artists, limiting any other potential use cases for the minted NFTs.

The selected platforms with NFT minting functionality for the comparison were chosen based on their current popularity according to the number of total page visits. An additional selection criteria was the intended use. Authors of this paper tried their best to find NFT minting solutions which are not focused on the sale of art, however only one such application was found.

1. Opensea.io
   a. 78.7 million total visits [55]
2. Rarible.com
   a. 3 million total visits [58]
3. Mintable.app
   a. 1.5 million total visits [56]
4. Mintnft.today

a. 10.6 thousand total visits[57]

The result of the analysis is as follows: applications Opensea, Rarible & Mintable are tailored towards NFT art creators, meaning that offered functionality requires registration, setting the price of the NFT, requires providing social media data for verification, navigation through a number of pages before the NFTs can be minted. In the case of Mintable, the images might falsely be not allowed to be minted, due to copyright considerations. During testing, a randomly generated image was falsely marked as being popular. Additionally, authentication and linking the applications account to a wallet is necessary before it is possible to mint an NFT. Exception here is Mintnft platform, which allows for a direct minting of the NFT, the drawback being that only mainnet Polygon network is supported. Applications do allow for the direct upload of NFTs to the IPFS with the exception of Mintable.app, which requires the user to choose if the NFT metadata will be stored on IPFS, or stored off chain. The amount of supported mainnet networks is kept to a minimum, with Opensea and Rarible supporting the most mainnet networks (Ethereum, Klaydn, Polygon, Solana for Opensea & Ethereum, Flow, Tezos, Polygon for Rarible). Support for testnet is minimal, with applications offering either 1 testnet network Ethereum Rinkeby or none. Considering the results of the evaluation, it can be concluded that current platforms are unfitting for security testing needs.

4.4 Conclusion

This chapter answers the second research question: **What are the features of currently existing NFT minting solutions?**

It can be concluded that existing NFT minting solutions do not fit the functional needs when it comes to the process of security testing. Existing NFT minting solutions are heavily geared towards NFT markets, and namely selling the NFTs. The functionality that currently surrounds the NFT minting applications has great limitations in the form of (i) GUI (e.g. users need to mint NFTs one by one, not always the IPFS file upload is automatic, minting more than 1 NFT requires filling out each request manually etc), additionally (ii) lack of support for minting testnet NFTs takes away the platforms ability to be utilised during the security testing of external applications where minted NFTs are needed to verify the correct implementation of NFT-oriented functionality within another application. (iii) The variety of offered blockchains is also heavily limited, requiring the security tester to jump from one application to another to mint a number of NFTs for different blockchains.

The limitations found in this section serve as a basis for the functionality requirements developed in the multichain NFT minting application, the topic of which gets expanded in the following sections.

5. Multichain NFT Minting Application For Security Testing

5.1 Introduction

This section will go into details about what NFT minting solution requirements need to be met, in order for the solution to be suitable for the security and application testing of multichain NFT applications. Following that, an outline and logical structure of the multichain NFT minter will be presented, as well as the multichain NFT minting solution itself.

5.2 Multi-chain NFT minting application requirements

For an NFT minting application to be suitable for usage during security testing process, the following criteria should be met:

1. Supports a number of currently most active mainnet chains
2. Supports respective testnet chains
3. Automatic image upload to IPFS
4. Support for bulk image upload
5. Allows for specification of address to which mint the NFTs to
6. Supports ERC271 & ERC1155
7. Relevant NFT minting transaction data is presented upon successful minting of the NFTs

5.3 Selection of Chains for the application

Due to each Blockchain having its limitations, new chains emerge to improve the issues found in the older chains. For example Solana was developed in 2017, with intention to provide high performance Blockchain by utilising a zero cost Foreign Function Interface, which offers a higher transaction speed of more than 50000 transactions per second (TPS) (compared to ETH 12-15 TPS), while offering significantly lower transaction costs and fees [51].

Currently the number of available Blockchains is rising quickly with more Blockchains being available to the public everyday. It is worth noting that a lot of those chains are a direct fork (also known as clone) of other chains, with minor (if any) adjustments which are aimed only to be used as a marketing gimmick. By analysing crypto currency market screeners data like DefiLlama and crypto currency exchanges like Binance, Coinbase and KuCoin, a list of most popular chains can be combined by analysing the:

- Number of protocols developed on the chain
- Total Value Locked (TVL)

Number of protocols indicates how actively the community is building applications which utilise the chain. Total Value Locked indicates how large the circulating amount of USD value is circulating inside the protocol. Aforementioned data points can hence be used as an indicator of the popularity of the chain.

By utilising previously mentioned logic, the most popular chains as of May 2022 can be concluded as:

- Ethereum (ETH)
    - 479 protocols developed
    - $111 billion TVL
- Binance Smart Chain (BSC)
    - 380 protocols developed
    - $12.16 billion TVL
- Avalanche (AVAX)
    - 206 protocols developed
    - $9.5 billion TVL
- Solana (SOL)
    - 63 protocols developed

    - $6.11 billion TVL
- Fantom (FTM)
    - 222 protocols developed
    - $4.13 billion TVL
- Polygon (MATIC)
    - 249 protocols developed
    - $3.6 billion TVL

As a result of this analysis, the currently popular chains can be concluded, hence they need to be supported within the multichain NFT minting solution as well as their respective testnet chains.

5.4 Bulk Upload & Automatic Image Upload to IPFS

When an NFT is created and linked to a digital file residing on another system, the manner in which the data are linked is crucial. There are several reasons why conventional HTTP links are not optimal.

Anyone can retrieve the contents of *my-nft.jpeg* using an HTTP address such as *https://cloud-bucket.provider.com/my-nft.jpeg* as long as the server is operational. The contents of *my-nft.jpeg* cannot be guaranteed to be the same as they were when the NFT was created. The server owner can easily replace *my-nft.jpeg* with a different image at any time, thereby altering the NFT's meaning.

This problem can be solved by utilising IPFS and its Content Addressing functionality. Adding data to IPFS generates a content identifier (CID) that is directly derived from the data itself and provides a link to the data within the IPFS network. Because a CID can only ever refer to a single piece of content, we know that no one can replace or modify it without breaking the link.

Using the CID, anyone can retrieve a copy of the data from the IPFS network so long as at least one copy exists, even if the original provider has vanished. Thus, CIDs are ideal for NFT storage. Simply inserting the CID into an *ipfs:/* URI such as *ipfs:/bafybeidlkqhddsjrdue7y3dy27pu5d7ydyemcls4z24zlyemcls4z24zlyek3we7vqvam/nft-image.png* creates an immutable link from the blockchain to our token's data.

Obviously, there may be circumstances in which it might be necessary to modify the metadata for a published NFT. In such a case it is only needed to add support to the smart contract for updating a token's URI after it has been issued. This will allow the user to change the URI to a new IPFS URI while leaving the previous version in the blockchain's transaction history [59].

For the solution to have an efficient and effective use case, the application should support bulk image upload, which will then consequently be minted to the provided wallet address. Additionally, the application should have an ability to automatically upload the metadata of the NFT to the IPFS.

5.5 Multichain NFT minting application components

For an application to function accordingly to the requirements, the following suite of components is used:

- Server
  - AWS EC2 for hosting the server
  - Nginx to serve the web server
- Frontend
  - HTML Bootstrap v5.1 for visually appealing frontend
  - Javascript for sending data from frontend to backend
- ETH NFT minting
  - Pinata [60] for file upload for Ethereum Blockchain
  - Infura [62] as a Web3 backend
- Solana NFT minting
  - Metaplex Candy Machine v2 [61] for file upload for Solana Blockchain

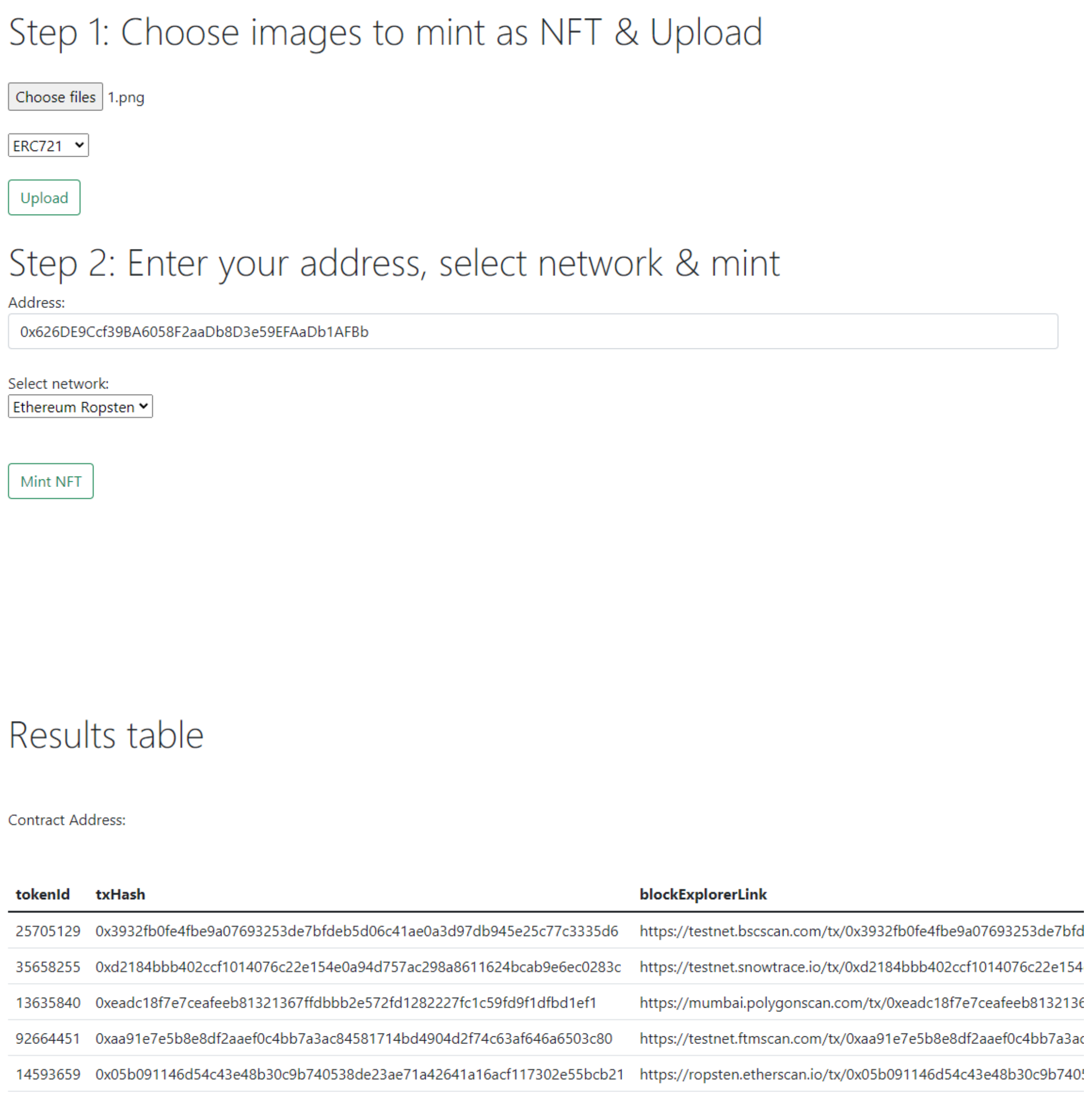
For each of the supported Blockchain, their respective Blockchain scanners are used for retrieval of the token minting data.

Chain	Mainnet Explorer link	Testnet Explorer link
BSC	bscscan.com	testnet.bscscan.com

Avalanche	snowtrace.io	testnet.snowtrace.io
Polygon	polygonscan.com	mumbai.polygonscan.com
Fantom	ftmscan.com	testnet.ftmscan.com
Ethereum	etherscan.io	ropsten.etherscan.io
Solana	solscan.io	solscan.io/?cluster=testnet solscan.io/?cluster=devnet

Table 3. List of chains mapped to their respective mainnet and testnet chain explorers

The application can be seen on the image below:


Step 1: Choose images to mint as NFT & Upload
Choose files
Upload
Step 2: Enter your address, select network & mint
Mint NFT
Results table

1.png

ERC721

Address: 0x626DE9Ccf39BA6058F2aaDb8D3e59EFAaDb1AFBb

Select network: Ethereum Ropsten

Contract Address:

tokenId	txHash	blockExplorerLink
25705129	0x3932fb0fe4fbe9a07693253de7bfdeb5d06c41ae0a3d97db945e25c77c3335d6	https://testnet.bscscan.com/tx/0x3932fb0fe4fbe9a07693253de7bfd
35658255	0xd2184bbb402ccf1014076c22e154e0a94d757ac298a8611624bcab9e6ec0283c	https://testnet.snowtrace.io/tx/0xd2184bbb402ccf1014076c22e154
13635840	0xeadc18f7e7ceafeeb81321367ffdbbb2e572fd1282227fc1c59fd9f1dfbd1ef1	https://mumbai.polygonscan.com/tx/0xeadc18f7e7ceafeeb813213
92664451	0xaa91e7e5b8e8df2aaef0c4bb7a3ac84581714bd4904d2f74c63af646a6503c80	https://testnet.ftmscan.com/tx/0xaa91e7e5b8e8df2aaef0c4bb7a3a
14593659	0x05b091146d54c43e48b30c9b740538de23ae71a42641a16acf117302e55bcb21	https://ropsten.etherscan.io/tx/0x05b091146d54c43e48b30c9b740

Figure 3. Screenshot of the multichain NFT minting application

Application first asks the user to choose the image files which need to be minted, after which the user can select the necessary token type (ERC721 or ERC1155).
The application can be used from this address: http://3.141.225.202/
Code can be found here: https://github.com/arniki/NFT-Minter

5.6 Conclusion

This section answers the research question 3: **How can the security testing process of NFT functionalities within web3 applications be improved?**

The question gets answered by creating a supplemental multichain NFT minting application, where the usability is designed around ease of use during security testing. Additional smart contracts were deployed which perform the NFT minting operation.
The application supports multiple image uploads at once, offers a choice between the ERC721 and ERC1155 token types, and supports a number of currently most active & popular blockchains according to Total Value Locked in USD & developed protocols which employ the chain. The application automatically uploads the files to the IPFS and returns to the user relevant transaction data for verification purposes in the form of blockchain explorer URLs.

The current limitations of this solution is that the minting procedure can take place only for one address at the time. The plan is to add functionality for minting to a number of addresses concurrently.

6. Evaluation

This section of the thesis will present an evaluation of the created artefact - Multichain NFT minting application.

6.1 Introduction

This section will evaluate the artefact according to the hierarchy of criteria for IS artefact evaluation. The following methodology presented in figure 4 is used for the evaluation methodology.

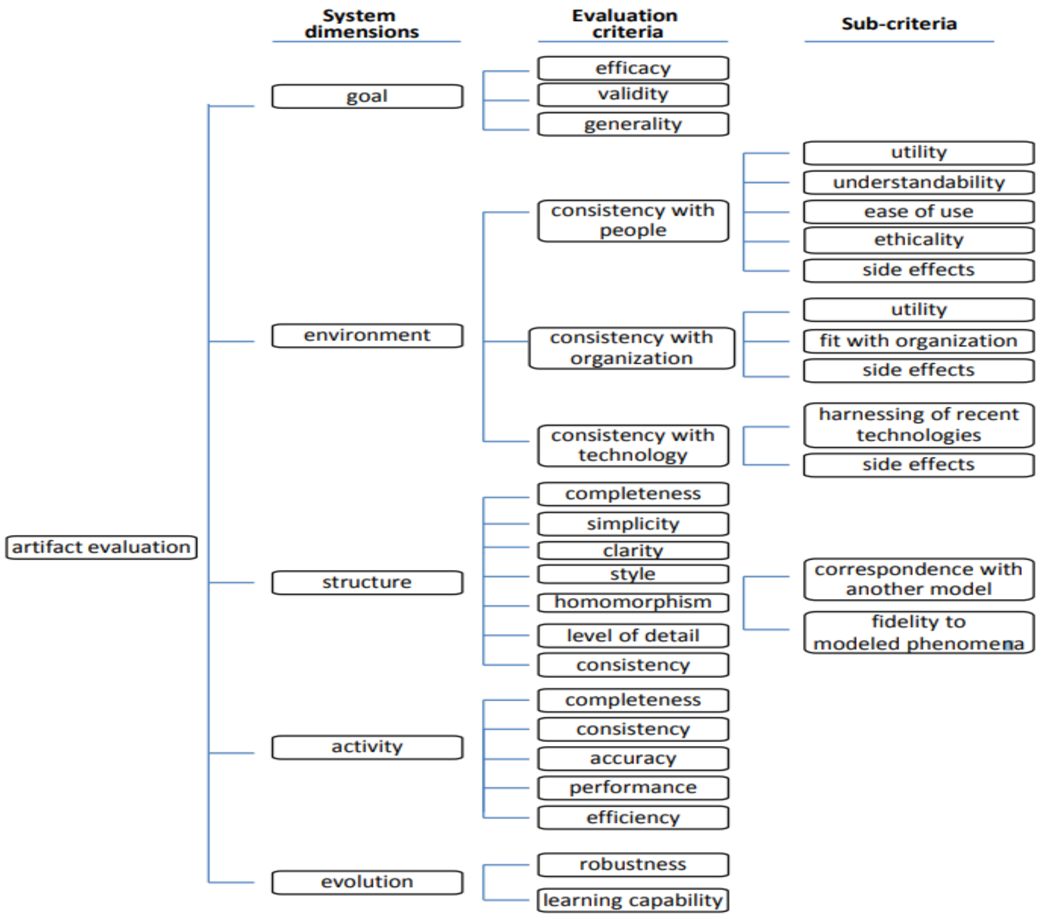


Figure 4. Hierarchy of criteria for IS artefact evaluation [63]

6.2 Evaluation of the Multichain NFT minting solution

The initial requirements set in Chapter 5 of this thesis were as follows:

1. Supports a number of currently most active mainnet chains
2. Supports respective testnet chains
3. Automatic image upload to IPFS
4. Support for bulk image upload
5. Allows for specification of address to which mint the NFTs to
6. Supports ERC271 & ERC1155

7. Relevant NFT minting transaction data is presented upon successful minting of the NFTs

Points 1 & 2 were completed by supporting mainnet and testnet version of the chains for the following list of Blockchains: Ethereum (ETH), Binance Smart Chain (BSC), Solana (SOL), Avalanche (AVAX), Polygon (MATIC), Fantom (FTM). The developed backend functionality automatically uploads the files to IPFS and supports bulk image upload, completing points 3 & 4. The GUI of the application allows for the recipients address & token type specification, fulfilling the points 5 & 6. Upon successful minting of the NFT, the respective block explorer links for each chain where the NFT was minted get returned to the user to be used for verification and further analysis, fulfilling the point number 7.

6.3 Discussion

Goal of this artefact was to improve the spectrum of available usable tools when it comes to the security testing of the multichain NFT applications. By presenting a usable multichain NFT minting solution which supports (i) a number of currently popular mainnet blockchains and (ii) a number of respective testnets, the efficiency and efficacy of the security testing process is improved. An example can be seen from the non-blockchain security testing industry e.g. web application security testing: a wide variety of available tools that help/supplement the security testing process resulted in growth of the security in online e-commerce.

The initial idea of the solution had its main aim towards the web3 applications security testers, however as the research and thesis itself progressed, it was found that such tools are currently completely unavailable even for regular (non-security) NFT application testing, solidifying the idea and development of the artefact even further. The solution was designed with two main points in mind:

- Easy to use
- Easy to maintain

By offering easy-to-understand solution, the utility horizon for such a solution expands, requiring organisations and individual users very minimal effort when it comes to using & maintaining the solution.

The GUI of the application was kept minimalistic, while still providing necessary functionality and information. This result was based on one of the points discovered during the analysis of the existing NFT minting applications - currently offered NFT minting solutions are overpacked with unnecessary features like registration, NFT listing on the NFT market etc. Such features are not needed during the security testing process and hinder rather than help.

Currently the only maintenance required for the solution to function properly, is refilling the mainnet and testnet wallet with funds required for paying the fees caused by NFT minting operations on the Blockchain. This drawback however is easily mitigated by freely available testnet token faucets, which offer users testnet tokens free of charge.

7. Conclusion

The main research question (MRQ) posed by the thesis was: **How to streamline secure implementation of NFT functionalities within multichain applications?**
The MRQ was subdivided into three sub research questions (SRQ):

- **SRQ1:** What are common security vulnerabilities related to incorrect implementation of NFT smart contracts & NFT functions into web3 applications?
  A: Answered by reviewing and analysing existing materials around NFT smart contract and NFT applications vulnerabilities.

- **SRQ2:** What are the features of currently existing NFT minting solutions?
  A: Answered by reviewing and evaluating existing NFT minting platforms.

- **SRQ3:** How can the security testing process of NFT functionalities within web3 applications be improved?
  A: Answered by developing a supplemental application for creating testing NFTs for a number of different Blockchains.

7.1 Summary

The paper first introduced topics about Bitcoin, Ethereum, smart contracts, consensus algorithms & token types to provide an overview of the existing state of the art body of knowledge. The paper then provided an overview of most widespread smart contract vulnerabilities using the DASP TOP 10 taxonomy for smart contracts vulnerabilities, as well as introduced the SWC vulnerabilities registry to provide further understanding of existing vulnerabilities related to smart contracts. The focus then shifts to more NFT application specific vulnerabilities that are found not only in smart contracts via bugs/vulnerabilities, but also via flawed implementation of operational logic This answers the sub research question 1. The paper then explains why minting NFTs is necessary for the security testing of NFT applications, as well as evaluates a number of currently available NFT minting solutions according to criteria necessary for security testing of the applications, to outline their applicability during a security testing process, answering the sub research question number 2. Based on data collected by answering the SRQ2, the criteria for the new NFT minting solution have been created, to ensure high applicability of the application during the security testing process. Within the scope of SRQ3 a new NFT minting solution was developed, which supports a number of currently most active Blockchains, as well as has the GUI optimised for security testing.

By answering SRQ1 + SRQ2 + SRQ3, we get an answer to our main research question. The process of secure implementation of NFT functionalities within multichain applications can be improved by utilising the developed artefact.

7.2 Limitations

One of the biggest limitations endured during the research phase is the severe lack of academic resources related to NFT security, which drastically hindered the research process. Additional challenge was finding academic literature regarding smart contract vulnerabilities, which are not in Solidity or oriented towards Ethereum blockchain.

One of the main current limitations of the solution is the fact that the NFTs can only be minted to one address at a time. This is considered a limitation due to the fact that during functionality and security testing of the multichain NFT applications, it might be necessary to mint NFTs to a bigger number of different addresses.

Another limitation of the designed solution is the fact that as the Blockchain industry and market grows, new token types and blockchains will emerge, due to which the solution requires continuous developmental support. Additionally the initial setup can be relatively time consuming, due to a number of 3rd party services employed for the operation of the solution (e.g. API key for each of the blockchain's explorers etc).

7.3 Open issues

Based on the endured limitations, the current open issues include the necessity of more academic research into NFT technologies as a whole & with a focus on the security aspect of the applications, development and vulnerabilities.

7.4 Further Work

Further work of the current solution includes addressing the previously mentioned limitations. Specifically, (i) adding the supporting functionality for minting NFTs to multiple addresses concurrently, this can be achieved by changing the operational logic and employing multithreading, (ii) adding more blockchains, to ensure the applicability of the solution is as all-encompassing as possible towards the testing of the multichain applications and (iii), potentially adding a simplified way to automatically fund testnet and mainnet funds when the balance of the minting addresses gets low.

References

1. Mastering Blockchain - Imran Bashir - Google Books. (2017). Packt Publishing Ltd.
2. Zheng, Z., Xie, S., Dai, H.-N., Chen, W., Chen, X., Weng, J., & Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. Future Generation Computer Systems, 105, 475–491. https://doi.org/10.1016/j.future.2019.12.019

3. Wang, Q., Li, R., Wang, Q., & Chen, S. (2021). Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges. ArXiv. https://doi.org/10.48550/arxiv.2105.07447
4. Sayeed, S., Marco-Gisbert, H., & Caira, T. (2020). Smart contract: attacks and protections. IEEE Access : Practical Innovations, Open Solutions, 8, 24416–24427. https://doi.org/10.1109/ACCESS.2020.2970495
5. Ante, L. (2021a). Non-fungible token (NFT) markets on the Ethereum blockchain: Temporal development, cointegration and interrelations. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3904683
6. CoinMarketCap. (n.d.). CoinMarketCap Non-Fungible Token market statistics. CoinMarketCap NFT Statistics. Retrieved April 14, 2022, from https://coinmarketcap.com/nft/
7. Ante, L. (2021b). The non-fungible token (NFT) market and its relationship with Bitcoin and Ethereum. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3861106
8. Das, D., Bose, P., Ruaro, N., Kruegel, C., & Vigna, G. (2021). Understanding Security Issues in the NFT Ecosystem. ArXiv. https://doi.org/10.48550/arxiv.2111.08893
9. Entriken, W., Shirley, D., Evans, J., & Sachs, N. (2018, January). EIP-721: Non-Fungible Token Standard. Ethereum Improvement Proposals, No. 721. https://eips.ethereum.org/EIPS/eip-721
10. Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2018). Blockchain technology overview. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8202
11. Aste, T., Tasca, P., & Di Matteo, T. (2017). Blockchain technologies: the foreseeable impact on society and industry. Computer, 50(9), 18–28. https://doi.org/10.1109/MC.2017.3571064
12. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review.
13. Nofer, M., Gomber, P., Hinz, O., & Schiereck, D. (2017). Blockchain. Business & Information Systems Engineering, 59(3), 183–187. https://doi.org/10.1007/s12599-017-0467-3
14. Zhang, R., Xue, R., & Liu, L. (2019). Security and privacy on blockchain. ACM Computing Surveys, 52(3), 1–34. https://doi.org/10.1145/3316481

15. Vujičić, D., Jagodić, D., & Ranđić, S. (2018). Blockchain technology, bitcoin, and Ethereum: A brief overview.
16. Buterin, V. (2014). A next-generation smart contract and decentralized application platform. White Paper.
17. Tikhomirov, S. (2018). Ethereum: state of knowledge and research perspectives. In A. Imine, J. M. Fernandez, J.-Y. Marion, L. Logrippo, & J. Garcia-Alfaro (Eds.), Foundations and practice of security (Vol. 10723, pp. 206–221). Springer International Publishing. https://doi.org/10.1007/978-3-319-75650-9_14
18. Chen, H., Pendleton, M., Njilla, L., & Xu, S. (2020). A survey on ethereum systems security. ACM Computing Surveys, 53(3), 1–43. https://doi.org/10.1145/3391195
19. Su, L., Shen, X., Du, X., Liao, X., Wang, X. F., & Xing, L. (n.d.). Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications.
20. Lin, I. C., & Liao, T. C. (2017). A survey of blockchain security issues and challenges. Int. J. Netw. Secur.
21. ERC-20 Token Standard | ethereum.org. (n.d.). Retrieved April 22, 2022, from https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
22. Vogelsteller, F., & Buterin, V. (2015, November 20). EIP-20: Token Standard. Ethereum Improvement Proposals. https://eips.ethereum.org/EIPS/eip-20
23. TOKEN STANDARDS. (2021, October 19). TOKEN STANDARDS. https://ethereum.org/en/developers/docs/standards/tokens/
24. ERC-721 Non-Fungible Token Standard | ethereum.org. (n.d.). Retrieved April 22, 2022, from https://ethereum.org/en/developers/docs/standards/tokens/erc-721/
25. Chohan, U. W. (2021). Non-Fungible Tokens: Blockchains, Scarcity, and Value. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3822743
26. Karandikar, N., Chakravorty, A., & Rong, C. (2021). Blockchain Based Transaction System with Fungible and Non-Fungible Tokens for a Community-Based Energy Infrastructure. Sensors (Basel, Switzerland), 21(11). https://doi.org/10.3390/s21113822
27. Bamakan, S. M. H., Nezhadsistani, N., Bodaghi, O., & Qu, Q. (2022). Patents and intellectual property assets as non-fungible tokens; key technologies and

challenges. Scientific Reports, 12(1), 2178. https://doi.org/10.1038/s41598-022-05920-6

28. Houser, K., & Holden, J. T. (n.d.). Navigating the Non-Fungible Token. Utah Law Review.
29. Omar, A. S., & Basir, O. (2020). Secure Anti-Counterfeiting Pharmaceuticals Supply Chain System Using Composable Non-Fungible Tokens. In Y. Maleh, M. Shojafar, M. Alazab, & I. Romdhani (Eds.), Blockchain for cybersecurity and privacy: architectures, challenges, and applications (pp. 243–259). CRC Press. https://doi.org/10.1201/9780429324932-14
30. Real Estate NFT Marketplace | Real Estate NFT Development | Real estate NFT Marketplace Development | NFT Marketplace For Real Estate | NFT Real estate Platform | NFT Real estate Marketplace | How Real Estate NFT Works| NFT Real Estate | NFT for Real Estate | Virtual Land NFT | Property NFT. (n.d.). Retrieved April 22, 2022, from https://www.blockchainappfactory.com/nft-for-realestate
31. Lessons From the Nifty Gateway NFT Heist: Not Your Keys... - CoinDesk. (n.d.). Retrieved April 22, 2022, from https://www.coindesk.com/tech/2021/03/17/lessons-from-the-nifty-gateway-nft-heist-not-your-keys-not-your-art/
32. Hackers Exploit Arbitrum-based Marketplace Treasure: Over 100 NFTs Stolen. (n.d.). Retrieved April 22, 2022, from https://cryptopotato.com/hackes-exploit-arbitrum-based-marketplace-treasure-over-100-nfts-stolen/
33. Check Point Software Prevents Theft of Crypto Wallets on OpenSea, the World's Largest NFT Marketplace - Check Point Software. (n.d.). Retrieved April 22, 2022, from https://blog.checkpoint.com/2021/10/13/check-point-software-prevents-theft-of-crypto-wallets-on-opensea-the-worlds-largest-nft-marketplace/
34. OpenSea Bug Allows Attackers to Get Massive Discount on Popular NFTs. (n.d.). Retrieved April 22, 2022, from https://www.coindesk.com/tech/2022/01/24/opensea-bug-allows-attacker-to-get-massive-discount-on-popular-nfts/
35. DASP - TOP 10. (n.d.). Retrieved April 23, 2022, from https://dasp.co/index.html

36. not-so-smart-contracts/unprotected_function at master · crytic/not-so-smart-contracts. (n.d.). Retrieved April 23, 2022, from https://github.com/crytic/not-so-smart-contracts/tree/master/unprotected_function
37. Denial of Service - Ethereum Smart Contract Best Practices. (n.d.). Retrieved April 23, 2022, from https://consensys.github.io/smart-contract-best-practices/attacks/denial-of-service/
38. CreDA Protocol Whitepaper - CreDA protocol International. (n.d.). Retrieved April 23, 2022, from https://creda-app.gitbook.io/creda-protocol/introduction/creda-protocol-whitepaper
39. MythX: Smart contract security service for Ethereum. (n.d.). Retrieved April 24, 2022, from https://mythx.io/
40. ConsenSys/mythril: Security analysis tool for EVM bytecode. Supports smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. (n.d.). Retrieved April 24, 2022, from https://github.com/ConsenSys/mythril
41. crytic/slither: Static Analyzer for Solidity. (n.d.). Retrieved April 24, 2022, from https://github.com/crytic/slither
42. eth-sri/securify2: Securify v2.0. (n.d.). Retrieved April 24, 2022, from https://github.com/eth-sri/securify2
43. trailofbits/manticore: Symbolic execution tool. (n.d.). Retrieved April 24, 2022, from https://github.com/trailofbits/manticore
44. Overview · Smart Contract Weakness Classification and Test Cases. (n.d.). Retrieved April 24, 2022, from https://swcregistry.io/
45. SmartContractSecurity/SWC-registry: Smart Contract Weakness Classification and Test Cases. (n.d.). Retrieved April 24, 2022, from https://github.com/SmartContractSecurity/SWC-registry
46. OpenZeppelin. (n.d.). Retrieved April 24, 2022, from https://openzeppelin.com/
47. OpenZeppelin | Contracts. (n.d.). Retrieved April 24, 2022, from https://openzeppelin.com/contracts/
48. Hevner, March, Park, & Ram. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75. https://doi.org/10.2307/25148625

49. raineorshine/solgraph: Visualize Solidity control flow for smart contract security analysis. ⇆. (n.d.). Retrieved May 3, 2022, from https://github.com/raineorshine/solgraph
50. fergarrui/ethereum-graph-debugger: Ethereum solidity graph plain debugger. To have the whole picture when debugging. (n.d.). Retrieved May 3, 2022, from https://github.com/fergarrui/ethereum-graph-debugger
51. Yakovenko, A. (2018). Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*.
52. Solidity Visual Developer - Visual Studio Marketplace. (n.d.). Retrieved May 3, 2022, from https://marketplace.visualstudio.com/items?itemName=tintinweb.solidity-visual-auditor
53. How to Mint an NFT (Part 2/3 of NFT Tutorial Series) | ethereum.org. (n.d.). Retrieved May 12, 2022, from https://ethereum.org/en/developers/tutorials/how-to-mint-an-nft/
54. Create an NFT (Part 1) | Avalanche Docs. (n.d.). Retrieved May 12, 2022, from https://docs.avax.network/dapps/smart-digital-assets/creating-a-nft-part-1/
55. opensea.io Traffic Analytics & Market Share | Similarweb. (n.d.). Retrieved May 12, 2022, from https://www.similarweb.com/website/opensea.io/#overview
56. mintable.app Traffic Analytics & Market Share | Similarweb. (n.d.). Retrieved May 12, 2022, from https://www.similarweb.com/website/mintable.app/#overview
57. mintnft.today Traffic Analytics & Market Share | Similarweb. (n.d.). Retrieved May 12, 2022, from https://www.similarweb.com/website/mintnft.today/#overview
58. rarible.com Traffic Analytics & Market Share | Similarweb. (n.d.). Retrieved May 12, 2022, from https://www.similarweb.com/website/rarible.com/#overview
59. Mint an NFT with IPFS | IPFS Docs. (n.d.). Retrieved May 12, 2022, from https://docs.ipfs.io/how-to/mint-nfts-with-ipfs/
60. Pinata | Your Home for NFT Media. (n.d.). Retrieved May 12, 2022, from https://www.pinata.cloud/
61. Introduction | Metaplex Docs. (n.d.). Retrieved May 12, 2022, from https://docs.metaplex.com/candy-machine-v1/introduction

62. Ethereum API | IPFS API & Gateway | ETH Nodes as a Service | Infura. (n.d.). Retrieved May 12, 2022, from https://infura.io/

63. Prat, N., Comyn-Wattiau, I., & Akoka, J. (2014). Artifact Evaluation in Information Systems Design-Science Research-a Holistic View. PACIS.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.