MITIGATING SECURITY RISKS IN NFT SMART CONTRACTS

NADIA WOLEK

22017452

BSC IN COMPUTER SCIENCE (CYBER SECURITY)

05/05/2025

SCHOOL OF COMPUTER SCIENCE AND MATHEMATICS
Keele University
Keele
Staffordshire
ST5 5BG

Table Of Contents

Abstract:

The rising popularity of NFTs (Non-Fungible Tokens) has introduced new cybersecurity challenges, especially at the smart contract level. This project explores how common vulnerabilities in NFT smart contracts can be mitigated through secure-by-design principles, with decisions guided by real-world case studies and supporting literature. A custom ERC-721 smart contract was developed in Solidity using OpenZeppelin, implementing security features including role-based access control, reentrancy protection, metadata freezing, supply caps, and URI uniqueness checks. Testing within a simulated Ethereum environment using Remix IDE showcased the effectiveness of these measures in reducing common attack vectors such as unauthorised minting and metadata tampering. Although the contract was not deployed to a live network and did not undergo automated auditing by using tools like Slither, results show that security-focused development can reduce risk. Future work should focus on more comprehensive testing, live deployment, and third-party integrations for results to be able to be generalised and assess the efficacy of the system in realistic conditions.

1. Introduction

Non-Fungible Tokens (NFTs) are tokenised digital assets built on blockchain technology. Unlike fungible cryptocurrencies such as Bitcoin or Ethereum, NFTs are not interchangeable; they contain a unique digital signature, preventing exchange without changes in value (Kaspersky, 2022). Because NFTs are recorded on the blockchain, transactions are immutable, public and verifiable, making them useful for proving ownership of digital content such as art, music and photographs (Coinbase, 2021a).

Evolving from a niche blockchain novelty, the NFT market grew, with an estimated $25 billion in sales in 2021 alone. High-profile sales such as the digital artwork ‘Everydays: The First 5000 Days’ by Beeple sold for $69 million in March 2021, drawing more attention to the space (Cho et al., 2023). These figures show that NFTs are not only a technological advancement, but an important economic event that changed digital ownership and monetisation for artists on a large scale.

Ethereum prominently platformed NFTs due to the flexibility of its smart contracts (Summers, 2023), however other networks such as Solana and Polygon developed their own NFT ecosystems, with various technical trade-offs, such as lower transaction fees (Coinbase, 2021b).

NFTs function through smart contracts, which automatically execute code to define and enforce the behaviour of each token (IBM, 2021). Most Ethereum-based NFTs follow the ERC-721 standard, where each token has a unique identifier and cannot be traded one-to-one on the same contract (OpenZeppelin, n.d.). However, ERC-1155 supports both non-fungible and fungible assets within the same single contract. This

can be applied in gaming environments, where players can both have unique items and a currency (Magic Eden, 2024).

When a smart contract is deployed, it is immutable, and its code cannot be changed. Users send a transaction from their blockchain wallet, executing functions without needing a third-party authorising the transaction (Iberdrola, n.d.). In NFTs, smart contracts handle actions such as the creation of a new token (minting), transferring tokens between addresses and enforcing royalties on secondary markets. In addition to this, smart contracts link off-chain metadata to an NFT using uniform resource identifiers (URIs), which point to external storage that holds files that are too large to store on-chain (Srivastava, 2024).

Online marketplaces such as OpenSea, Blur and Magic Eden enable the trade and sale of NFTs by interacting directly with smart contracts (Coinbase, 2021b). These platforms enable users to easily mint, buy, sell and auction their NFTs through a user interface, simplifying the normally complex blockchain transactions. Marketplaces also deploy their own smart contracts process payments, manage listings, and automate creator royalties. However, the support for royalties has diminished over time; platforms have implemented zero-royalty models, which threatens the artists' financial sustainability (Holcomb, 2025).

Alongside the rise of NFTs, concerns over the cybersecurity risks have grown. The popularity and value of NFTs have made them a target for attackers, with smart contracts being vulnerable to exploitation. In July 2022, the Omni NFT liquidity protocol suffered a hack in which an attacker took advantage of a Reentrancy bug to steal $1.4 million worth of cryptocurrency (Immunefi, 2023). Reentrancy attacks happen when malicious code calls back into a vulnerable contract before its previous

execution is complete, allowing attackers to repeatedly withdraw funds (Nightingale, 2024). OpenSea, the largest NFT marketplace was also targeted; a phishing attack in February 2022 caused $1.7 million in stolen tokens after users unknowingly signed malicious transactions (Behnke, 2022c). These incidents highlight the importance of maintaining proper contract logic, implementing safeguards and user education before deployment.

To explore how cybersecurity risks in NFTs can be mitigated through smart contract architecture, I developed an Ethereum-based NFT contract using Solidity and the OpenZeppelin framework. The contract follows the ERC-721 standard and implements security-focused features such as role-based access control, reentrancy protection, metadata freezing and URI duplication prevention. These features address common, real-world vulnerabilities, including unauthorised minting, metadata manipulation and role escalation. For example, building with potential reentrancy in mind prevents the kind of exploit used in the 2022 Omni attack. Through building and analysing this contract, I aim to demonstrate how secure-by-design principles and libraries like OpenZeppelin can reduce the attack surface in NFT smart contract development. Insights gained from reviewing literature and real-world case studies directly influenced the design choices in my smart contract implementation.

2. Literature Review:

2.1 Types of NFT vulnerabilities

NFTs are susceptible to a range of vulnerabilities across multiple layers of their architecture. One of the primary concerns is smart contract flaws, as NFTs implemented via Ethereum's ERC-721, or ERC-1155 standards rely heavily on the integrity of the contract code. Bugs or malicious logic within an NFT's smart contract can be exploited to bypass ownership rules or mint tokens without permission. For example, a sleepminting attack exploits a smart contract bug to falsify ownership records by minting an NFT directly to a legitimate wallet and then reclaiming it without the owner's consent (Guidi and Michienzi, 2023). Such exploits stem from the same kinds of issues seen in other blockchain contracts, like reentrancy or poor logic.

Off-chain metadata vulnerabilities also pose significant risks; NFT tokens often reference metadata, such as images, via a URI that points to external storage. If this link is not properly secured, attackers can manipulate it to change its content (Guidi and Michienzi, 2023). Since smart contracts are publicly accessible and cannot control the off-chain data they reference, attackers have a centralised point of failure they can target and have the opportunity to duplicate tokens (Salem , 2024). Inauthentic NFTs are rampant, with 80% of free mint listings on OpenSea being inauthentic duplications of popular collections (Pearson, 2022). As a result, buyers may struggle to distinguish between authentic and inauthentic tokens, undermining the idea of NFTs being used as proof of ownership.

NFTs are also exposed to application-layer vulnerabilities within online marketplaces that facilitate their trade. Marketplaces like OpenSea, Blur and Magic Eden handle

listings and auctions using their own backend. If these systems are not secured properly, attackers can exploit issues in how listings are stored or how transactions are processed. In January 2022, a logic bug between the frontend and backend on OpenSea allowed previously thought to be delisted NFTs to be sold at a significantly lower price than they were worth, resulting in financial losses for users (Behnke, 2022b). Marketplaces also enable manipulative behaviours such as price inflation through shill looping, where groups trade NFTs among themselves repeatedly to spoof demand and a higher floor price for a certain token (Ankit Gangwal et al., 2023). Unsuspecting buyers purchase these inflated tokens, while insiders sell off assets, leaving buyers with devalued tokens. The decentralised nature of NFTs and their marketplaces typically allow perpetrators to remain anonymous, making it more difficult to trace and prevent pump-and-dump schemes. This highlights the need for stronger preventative measures at the application layer such as stricter listing rules and improved detection of false inflation of assets.

In summary, NFT vulnerabilities can be grouped into flaws within code on chain, like smart contracts, off-chain storage and metadata, and the marketplace. Understanding these weaknesses is important; they emphasise exactly where more security measures need to be implemented.

2.2 Smart Contract and Blockchain Security Risks:

Because NFT contracts run on blockchain platforms such as Ethereum, they inherit the security risks associated with smart contracts and decentralised ledgers. Many documented Ethereum vulnerabilities are directly applicable to NFTs- Reentrancy attacks being an example. In these attacks, malicious code repeatedly calls back into

the victim contract before its previous execution is complete. NFT contracts are vulnerable to reentrancy and can be exploited in similar ways, with attackers repeatedly calling certain functions, such as token transfers, before the contract has properly recorded the last transaction (Chhipa, 2024).

Other common smart contract vulnerabilities include integer overflows/ underflows, unchecked exceptions and transaction-order dependence (TOD). TOD is a type of front-running where attackers monitor the mempool and submit their own transaction with a higher gas fee to be processed before the original transaction (Kadenzipfel, 2019). Within an NFT network, this could allow an attacker to buy a token just before another user's transaction is confirmed, giving them an unfair advantage and potentially securing assets at the expense of legitimate users. An example of this was seen in CryptoKitties, a blockchain-based game where users collect and breed digital cats. Attackers could monitor and front-run the giveBirth() function to complete another user's transaction in order to claim ether rewards meant for other users (Eskandari, Moosavi and Clark, 2020). This could also potentially be exploited by an automated process, such as a bot that is instructed to constantly monitor the mempool for pending transactions. This highlights how even minor aspects of an NFT project can be targeted for profit, especially when incentives are involved.

Outside of code bugs, NFT contracts can be affected by denial-of-service (DDoS) attacks. In January 2022, LooksRare suffered a DDoS attack soon after they launched their new NFT marketplace, preventing users from connecting their wallets, listing and buying NFTs (Adejumo, 2022). Ultimately, this led to financial losses during a critical period of time, negatively affecting the reputation of the platform. Some NFT projects also use upgradable proxy contracts that allow future updates, however, if the upgrade system is not implemented correctly, attackers can potentially take control of

the contract and change its logic. This is referred to as a “risky mutable proxy” (Wang, 2024).

Private keys can be compromised by attackers, either through phishing or malware, leading to a complete loss of assets. Multiple large-scale NFT thefts have occurred through social engineering. Notably, a phishing attack against BAYC, a popular NFT collection, led to official social media accounts being compromised. Illegitimate mint links were posted and when unsuspecting users connected their wallet, their assets were transferred to the attacker. Although the fraudulent posts were removed promptly, the attacker managed to seize $3 million worth in assets (Bîzgă, 2022). Similarly, a misconfiguration in how a marketplace verifies digital signatures could let an attacker forge approvals and transfer NFTs they don’t own. The blockchain network itself is a risk; if the network suffers a 51% attack, where users control over 50% of the network’s mining hashrate, they could reverse transactions or change ownership records (Ye et al., 2018). Although a 51% attack is less of a concern for Ethereum after it changed its model to proof of stake as it’s extremely difficult and expensive to control 51% of staked Ethereum, the risk is still relevant for NFTs hosted on Proof of Work blockchains (Saxo, 2023).

To summarise, NFT smart contracts have the same issues their underlying blockchain networks have, which are further exacerbated by operational security risks, such as platform breaches and phishing attacks. To successfully build a secure NFT contract, you need to address both the logic, but also the broader security challenges of the blockchain it operates within.

2.3 Case Studies of NFT Exploits:

OpenSea fell victim to an exploit where attackers purchased valuable NFTs for significantly below their market prices. Hackers discovered that if an NFT owner had previously listed their token for sale and then removed or changed the listing, OpenSea's APIs did not prevent access to the old data. In January 2022, attackers used this to their advantage, and purchased NFTs at outdated prices, and immediately sold them at the current market price, earning them an approximate profit of over 300 ETH (Brandom, 2022). This incident demonstrates a marketplace-level vulnerability; the smart contracts holding the NFTs functioned as intended, however, the off-chain listing system had security flaws which led to financial loss. This emphasises how indirect weaknesses, like API logic, can have a significant impact on the security of NFT assets.

Omni, an NFT liquidity protocol that allowed users to borrow cryptocurrency using their NFTs as collateral, had multiple reentrancy vulnerabilities within their smart contracts. These were then exploited by an attacker in July 2022, resulting in $1.4 million in losses. The attacker used a flash loan to deposit a large number of NFTs, then took advantage of the reentrancy vulnerabilities to withdraw the same collateral multiple times before the contract could update its records (Immunefi, 2023). This could've been entirely prevented by introducing a nonReentrant modifier, for example, using OpenZeppelin's ReentrancyGuard, which blocks the repeated calls to the same function during a transaction (OpenZeppelin, n.d.). While this mitigation is widely used, it may not fully protect contracts that involve multiple external calls. Even correct use of reentrancy guards can be bypassed if developers do not account for interactions between other contracts. Ultimately, relying on a single form of protection is insufficient; contracts need to be tested and audited thoroughly to address a wider range of potential risks.

NFT rug pulls are planned scams where creators launch and promote a collection, and inflate its value, only to abandon the project after collecting money from buyers. The NFT often becomes worthless afterwards, leaving buyers with financial losses. An example of this is the Eternal Beings collection, promoted by American rapper Lil Uzi Vert (Das et al., 2022). After buyers invested, all promotional material was deleted from social media pages, causing a sharp drop in value. Instead of taking advantage of smart contract logic or blockchain vulnerabilities, rug pulls take advantage of trust within a community. Recent studies show how common these scams are, with 253 confirmed NFT rug pulls and an algorithm identifying 7,487 likely rug-pulled projects (Huang et al., 2023). While smart contracts can include safeguards to reduce the chance of this happening- such as token locking and multi-signature withdrawals, the main issue is fraud. This highlights the importance of educating buyers and encouraging researching investments, as well as implementing contractual safeguards that make it more difficult for project teams to abandon projects with investor funds.

2.4 Security Frameworks and Mitigation Strategies

Given the number of vulnerabilities, researchers and developers have proposed multiple frameworks and best practices to mitigate NFT security risks. A common theme in the literature is the importance of avoiding known weaknesses, such as insecure access control, unlimited minting and reentrancy, by regularly auditing code and using standard libraries like OpenZeppelin. These approaches are deemed as important in the prevention of many common vulnerabilities.

Instead of writing an NFT smart contract entirely from scratch, it is common practice for developers to use libraries as a template. OpenZeppelin is the industry standard, offering implementations of ERC-721 and other standards that have been audited by

blockchain security experts (Raka Widhi Antoro, 2024). Using OpenZeppelin significantly lowers the chance of vulnerabilities within projects by introducing standardisation, which minimises the attack surface by ensuring developers follow patterns that have been well-tested and proven to be secure. However, using OpenZeppelin's prewritten components without properly tailoring them to the project can introduce risks, such as misconfigured access control, missing functionality, and logic errors. (Onesafe, 2025).

Access control in NFT smart contracts ensures that only authorised wallets can perform important functions, such as minting, burning or changing metadata. Many past vulnerabilities, such as unlimited minting or the 'Public Burn' vulnerability which allows any wallet to burn tokens, stem from missing or misconfigured access control (Wang, 2024). These risks can be mitigated using 'require' statements, function modifiers, or access control tools provided by OpenZeppelin, such as 'Ownable' and 'AccessControl' (Chakraborty, 2023). To summarise, role management should be built into the contract from the beginning, with clearly defined roles that can be revoked at a later point. Following secure design principles like least privilege and separation of roles reduces the risk of unauthorised access.

NFTs rely on off-chain data, which becomes an attack surface if not properly secured. A common solution is to use decentralised storage methods. IPFS (InterPlanetry File System) is favoured over traditional cloud solutions because it is immutable- content cannot be altered, preserving the value and integrity of an NFT (Igiakong, 2022a). IPFS works by linking a token's URI to a cryptographic hash of its content. If an attacker changes the file, the hash no longer matches, alerting users to tampering (Borgesi, 2024). Some NFT platforms offer a metadata freeze feature, where once an NFT is minted, the creator can no longer modify the metadata, preventing any future

alterations (Igiakong, 2022b). To combat duplication, NFT platforms are implementing a stricter verification process for creators to prevent the copying of metadata. Techniques such as requiring digital signatures on off-chain metadata are being pushed, giving buyers a greater sense of security in a token's authenticity. By combining these methods, developers can significantly reduce security risks by minimising potential attack surfaces associated with off-chain data.

3. Methodology and Implementation

In order to validate and demonstrate the security concepts identified in the literature review and case studies, I developed and tested a custom Ethereum-based NFT smart contract. Implementing a real smart contract allowed me to apply real-world mitigation strategies and evaluate their effectiveness in a controlled, virtual environment. Instead of only discussing vulnerabilities, I chose to build a working proof-of-concept that incorporates strategies against security risks outlined in the research.

3.1 Tools and Environment

Solidity was used as the smart contract language for my implementation due to its integration with the Ethereum blockchain and support for the ERC-721 standard. The contract was written and compiled using the Remix IDE, a web-based environment with a built-in Ethereum Virtual Machine (EVM). Using the EVM gave me the opportunity to test contract behaviour by simulating transactions from multiple wallets, without having to connect to a live network or use real funds. This allowed me to attempt to bypass the security features I had implemented, such as access control, reentrancy guards, and metadata restrictions, and confirm that the contract responded correctly to prevent unauthorised behaviour.

I used OpenZeppelin libraries to support my implementation as they are regarded as an industry standard for secure smart contract development. Relying on the OpenZeppelin framework helped me make my contract more standardised, reducing the attack surface by following community-audited components, instead of re-writing important functionality (Onesafe, 2025).

ERC721 was used to implement token logic and ERC721URIStorage added support to assign, store and update token metadata through URIs. For role-based permissions, I used AccessControl, which allowed me to clearly separate and define roles for minting and metadata freezing. Reentrancy was secured using ReentrancyGuard, which prevents repeated function calls in a single transaction by using nonReentrant. Lastly, I included Pausable, which allows an administrator wallet to disable important functions such as minting in an emergency scenario.

3.2 Role-Based Access Control

To prevent unauthorised users from escalating their privileges and gaining access to functions restricted to specific roles, the contract I designed uses OpenZeppelin's AccessControl system to enforce role-based permissions. Two roles are defined: DEFAULT_ADMIN_ROLE, which is automatically given to the wallet address that deploys the contract, and MINTER_ROLE, which is needed to mint new tokens or to freeze metadata. By checking roles with the onlyRole modifier, only authorised wallets with these roles are able to perform actions like minting and modifying metadata state. This was influenced by real-world examples, where errors in access control implementations are a common root cause of exploits, such as unauthorised minting and token burning (Wang, 2024).

The choice to define separate roles was made to follow the principle of least privilege, which limits permissions to only what is necessary to perform an action. For example, a wallet with permission to mint tokens should not also be able to pause the contract like an administrator. Separating roles helps minimise the risk of a compromised wallet controlling an entire contract.

```
// mint NFTs (only minter, only if not paused, and metadata not frozen)
function mint(address to, string memory uri)    infinite gas
    public
    onlyRole(MINTER_ROLE)
    whenNotPaused
    nonReentrant
```

Figure 1: Mint function with access control enforcement using onlyRole(MINTER_ROLE). Only the wallets with MINTER_ROLE assigned can mint new tokens. The function also includes whenNotPaused and nonReentrant to prevent minting during paused states and protect against reentrancy.

In order to test that access control was implemented correctly, I attempted to mint a new token by calling the mint() function from a wallet which had not been assigned the MINTER_ROLE. When trying this, the contract rejected the transaction as intended, returning an AccessControlUnauthorisedAccount error. This confirmed that only wallets with the correct role can call restricted functions such as minting new tokens.

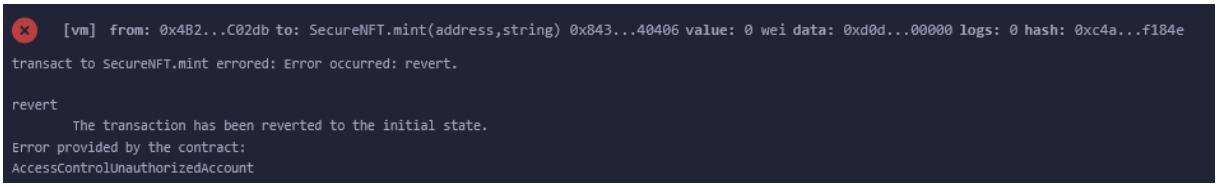


Figure 2: A failed minting attempt by a wallet without MINTER_ROLE, showing access control was successfully enforced.

3.3 Reentrancy Protection

Although less common in NFT contracts compared to other Ethereum-based applications, reentrancy vulnerabilities still need to be accounted for. Reentrancy, where a contract is interrupted and called again before completing the transaction, can allow attackers to exploit logic errors and perform the same action multiple times

before the contract's state is updated (Nightingale, 2024). In NFT contracts, an attacker can use this to potentially bypass max supply, and mint tokens with the same metadata URI or exceed the supply cap.

To prevent this, I used OpenZeppelin's ReentrancyGuard and applied it to the mint() function. This ensures that once the function is called, it cannot be re-entered until the transaction is complete. Even though the minting process does not involve sending Ether, it still changes the contract's state and works with external metadata links (URIs), so I felt that it was important to include reentrancy protection as a precautionary measure to make my implementation future-proof.

As shown in Figure 1, the mint() function includes nonReentrant alongside access controls.

3.4 Metadata Freezing and URI Protection

NFTs use URIs to link to off-chain metadata, meaning that it is important that these links remain immutable as soon as a token is minted. As mentioned in section 2.1, attackers exploit mutable metadata to duplicate NFTs to make illegitimate collections (Guidi and Michienzi, 2023). To address this vulnerability, I implemented both metadata freezing, and a URI duplication check.

The freezeMetadata() function allows authorised wallets to lock metadata after a token is minted, preventing any future changes permanently. This helps maintain authenticity and the value of a token. Once freezeMetadata is called on a token, the metadataFrozen mapping ensures that any attemps to change the associated URI will be blocked.

```
// freeze metadata for a token
function freezeMetadata(uint256 tokenId) public onlyRole(MINTER_ROLE) {    infinite gas
    require(_existsPublic(tokenId), "Token does not exist"); // safe check
    metadataFrozen[tokenId] = true; // lock metadata
    emit MetadataFrozen(tokenId); // log freeze
}
```

Figure 3: freezeMetadata function that locks metadata and saves it in the transaction log.

To test this, I minted a token with an initial metadata URI and then called the metadata freeze function for that token. After freezing, attempts to call the updateTokenURI failed as intended. This confirms that the immutability feature works, with URIs being unable to be changed permanently when metadata is frozen.

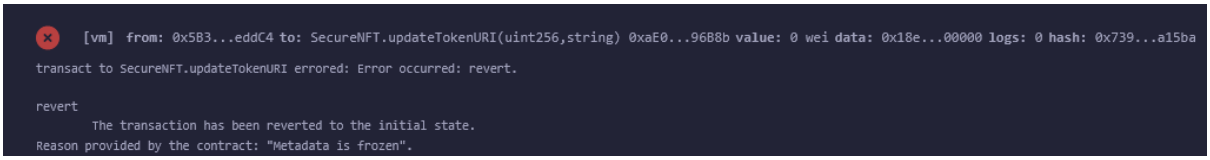


Figure 4: Token URI set to frozen. Calls to change the URI returned with error stating that metadata is frozen.

The contract also enforces unique metadata URIs for each token to prevent duplication of NFT content. The implementation maps all URIs that have been assigned to minted tokens and then checks whether a given URI has already been used before allowing a new mint. While having NFTs with the same metadata is not a vulnerability from a security perspective, it diminishes the value of an NFT and violates the principle of NFTs being non-fungible.

```
mapping(string => bool) private usedURIs; // tracks used metadata URIs
```

Figure 5: URI duplication protection using mapping to store previously used metadata URIs.

Testing confirmed that duplicate URIs cannot be used during minting, with attempts triggering an error stating metadata is already in use.

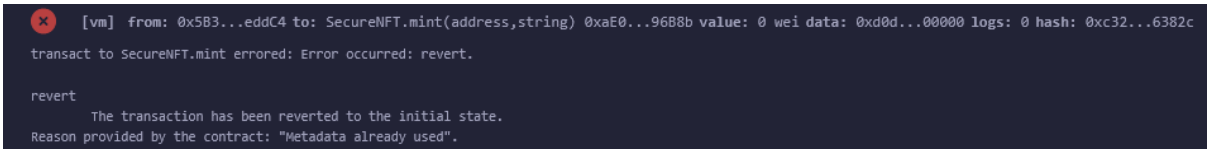


Figure 6: "Metadata already used" error after attempting to mint an NFT with the same URI as previous tokens.

3.5 Supply Cap Enforcement

Supply Caps are limits placed on token creation that define how many tokens can exist within a contract. Supply caps are important in NFT culture; they increase value of tokens through built-in scarcity (Cole, 2024). If supply cap logic is flawed, it can lead to over minting, potentially devaluing a collection and causing financial losses for buyers.

To mitigate this, I implemented a hard supply cap (MAX_SUPPLY) which checks the current total supply before allowing new mints. Each call to mint() checks whether the tokenIdCounter (total minted tokens) is less than the total supply. If the cap is reached, an error is displayed showing the max supply has been reached.

```
require(tokenIdCounter < MAX_SUPPLY, "Max supply reached");
```

Figure 7: Supply cap enforcement in the mint() function. Checks whether tokenIdCounter is less than the maximum allowed MAX_SUPPLY.

Testing confirmed the supply cap logic worked as intended; when attempting to mint more tokens than the defined limit, the transaction is rejected, and an error is returned.

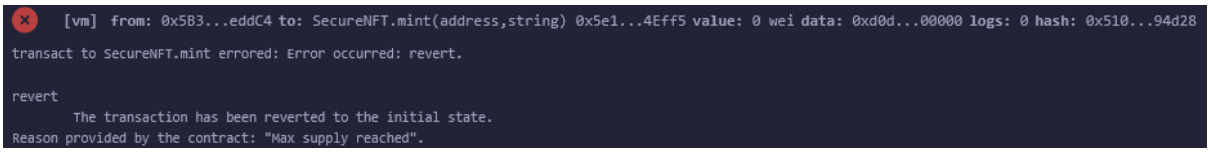


Figure 8: Failed minting attempt after exceeding MAX_SUPPLY. “Max supply reached” error message returned, confirming logic is enforced.

3.6 Pause Functionality

Pause functionality is designed to give administrators a way to temporarily suspend important contract operations. In a paused state, functions like minting, and optionally token transfers, will not execute. This acts as a safeguard to allow for an emergency response if a vulnerability is discovered or if an attack is detected. Administrators can pause the contract promptly while finding a solution, potentially limiting the impact of an attack.

Although this mitigates risk, it is criticised for impeding on the principle of decentralisation; there is centralised authority which can control the contract’s behaviour (Behnke, 2022d).

In my implementation, I used OpenZeppelin’s Pausable contract. This adds both pause() and unpause() functions, which can only be called by a wallet assigned the DEFAULT_ADMIN_ROLE. The mint() function is protected with a whenNotPaused check, which prevents token creation while the contract is paused.

```
// emergency stop functions (pause/unpause contract)
function pause() public onlyRole(DEFAULT_ADMIN_ROLE) {    infinite gas
    _pause(); // stop minting
    emit ContractPaused(msg.sender); // log pause
}

function unpause() public onlyRole(DEFAULT_ADMIN_ROLE) {    infinite gas
    _unpause(); // resume minting
    emit ContractUnpaused(msg.sender); // log unpause
}
```

Figure 9: pause() and unpause() functions protected by onlyRole(DEFAULT_ADMIN_ROLE). An administrator can temporarily disable contract functions like minting.

To test if I had successfully implemented both the pause() and unpause() functions, I first called the pause() function from the administrator wallet. This successfully paused the contract. When I attempted to mint a new token, the transaction failed, returning an error stating that the contract was paused. This confirmed that the mint() function correctly follows the paused logic. Once I called the unpause() function, the mint() function returns to working as intended.

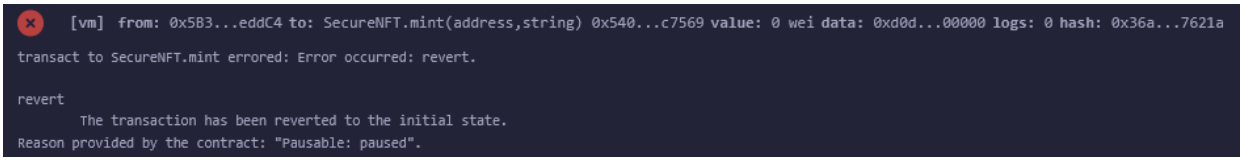


Figure 10: Pausable error returned after attempting to mint a new token while the contract is in a paused state.

3.7 Burn Functionality

The burn function allows token holders to destroy an NFT they own, permanently removing it from the supply. Similarly to supply caps, they are important in the culture, with burning adding scarcity, thus adding value (CCI, 2024).

In my implementation, the burn function only allows the owner of a token to burn, or a wallet approved for that specific token by the holder. If these conditions are met, the NFT is successfully removed from the blockchain.

```
// burn an NFT
function burn(uint256 tokenId) public {    infinite gas
    address owner = ownerOf(tokenId); // get current owner
    require(
        msg.sender == owner || // must be owner
        getApproved(tokenId) == msg.sender || // or approved for this token
        isApprovedForAll(owner, msg.sender), // or approved for all tokens
        "Not owner or approved"
    );
    _burn(tokenId); // destroy token
    emit TokenBurned(tokenId); // log burn
}
```

Figure 11: Burn() function implementation. Only the token owner, an approved address or wallet can call this function. If authorised, the NFT is permanently destroyed.

To test this, I attempted to burn a token from an unauthorised wallet, which returned an error as intended. Afterwards, I burned the same token using the correct wallet, and the transaction succeeded. This confirms that the access control and burning functionalities were implemented correctly.

```
[vm] from: 0xAb8...35cb2 to: SecureNFT.burn(uint256) 0x540...c7569 value: 0 wei data: 0x429...00000 logs: 0 hash: 0xbfd...e6b51
transact to SecureNFT.burn errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Not owner or approved".
```

Figure 12: Burn not completing from an unauthorised wallet. Error returns with “Not owner or approved”

```
transact to SecureNFT.burn pending ...

[vm] from: 0x5B3...eddC4 to: SecureNFT.burn(uint256) 0x540...c7569 value: 0 wei data: 0x429...00000 logs: 2 hash: 0xc3e...c33dd
status                          0x1 Transaction mined and execution succeed
```

Figure 13: Burn successful from owner wallet.

3.8 Conclusion

I chose to implement the security measures instead of solely discussing them and reviewing literature, to gain a better understanding of how they work in practice. Each design decision in the contract, from using reentrancy guards to freezing metadata after minting, was influenced by real-world case studies, and by vulnerabilities commonly highlighted in literature. By using development tools that are widely used and recognised as industry standards, I was able to create a secure NFT smart contract with full functionality that follows best security practices. The tests I conducted show that these security features behave as intended and help reduce multiple major attack surfaces in NFT smart contracts.

While I was able to demonstrate how to mitigate common vulnerabilities can be mitigated through design, the contract was only tested in a simulated environment. This is useful during development, but it does not reflect the behaviour of a live blockchain network.

4. Evaluation

4.1 Effectiveness of Implementation Tools

All testing was conducted within the Remix IDE using its Ethereum Virtual Machine. This made development and testing straightforward, without needing to install third-party frameworks such as Hardhat or Ganache to deploy my contract, nor use real wallets or deploy with actual funds. Instead, I was able to simulate a blockchain environment using virtual wallet addresses. However, this limited the realism of the testing environment. Remix provides limited control over the blockchain configuration, such as gas fluctuations or transaction ordering, unless third-party plugins are used (Tutorialspoint, 2019). Remix IDE simplifies gas estimation and instead returns “Infinite Gas” when execution costs are unpredictable, rather than simulating realistic gas fluctuations as seen on live blockchains (Singh, 2022).

This is a significant limitation of my methodology. Without deploying the contract on a live network, I could not evaluate how the contract would behave in real-world conditions, including unpredictable user behaviour or edge cases involving external contracts (Phung, 2024). In addition to this, testing was limited to a single user. This means the evaluation of the contract’s security is based solely on my usage, rather than accounting for individual differences in user behaviour. Because of this, a narrow testing scope bias is introduced, as the evaluation was limited to a single user and environment. Factors such as the wallet interfaces users choose, network latency and browser extensions can all influence on how reliable the security mitigations are.

In addition to this, I did not use automated audit tools such as Slither. Slither is a Solidity static analysis framework, used to audit smart contracts and detect vulnerabilities (Stepanov, 2023). As a result, the contract was only manually

reviewed, introducing possible human error and limits the scope of the security evaluation.

4.2 Effectiveness of Implemented Security Features

The security features implemented in my smart contract were directly informed by vulnerabilities documented in literature and case studies of real-world NFT exploits. Each function was selected to address a specific threat classification, including unauthorised minting, metadata tampering and reentrancy. Although there are limitations of using simulated testing environments, the results of my testing provide insight into how mitigation strategies defend against common security threats in NFT smart contracts.

4.2.1 Role-Based Access Control

The role-based access control system was successful in limiting function execution to authorised users, with separate roles being assigned to follow the principle of least privilege. During testing, wallets without the required roles were blocked from performing higher privilege actions such as minting or freezing metadata. In a practical deployment, this would likely mitigate real-world issues such as unauthorised minting and privilege escalation and public burning. These vulnerabilities are often identified in NFT projects where access control is misconfigured (Wang, 2024).

By clearly separating roles using OpenZeppelin's AccessControl, the contract avoided the single point of failure seen in simpler Ownable designs (OpenZeppelin, 2017). Contracts that rely on Ownable often have a single privileged wallet; if the deployer is compromised, the entire contract can be impacted (Behnke, 2022a). This could lead to unlimited authorised token creation and loss of user trust. Compromised owner

wallets have frequently been the primary vector in NFT-related rug pulls (Huang et al., 2023).

By building around this using AccessControl, the contract is more resilient against compromised wallets. However, a shortcoming is that in a live blockchain environment, AccessControl may incur higher gas fees in comparison to Ownable. In addition to this, its complexity introduces a risk for human error, where administrators can mishandle role assignment during deployment, such as forgetting to revoke privileges after minting ends. Ultimately leaving an attack surface which enables attackers to mint unauthorised tokens, highlighting the importance of careful role management.

4.2.2 Reentrancy Protection

Although reentrancy vulnerabilities are more commonly exploited in contracts other than NFTs, they still pose a significant risk when external calls or Ether handling are involved. The inclusion of OpenZeppelin's ReentrancyGuard within the mint() function was a preventative measure against these attacks. In testing, the nonReentrant modifier successfully blocked repeated calls to the function within the same transactions, confirming the contract behaved properly. Although minting in my implementation does not involve sending Ether, it interacts with metadata, which could create indirect reentrancy opportunities in more complex systems.

The Omni exploit case study influenced my decision to include a ReentrancyGuard, where reentrancy was used to bypass withdrawal limits, leading to significant financial loss (Immunefi, 2023). Although my contract is not vulnerable in the same way, proactively incorporating ReentrancyGuard improves the contract's scalability. If

in the future the contract was to be extended to support features such as paid minting or marketplace integration, there would be safeguards already in place.

However, the nonReentrant modifier only protects against reentrancy within a single function that it is applied to. It does not safeguard against interactions between multiple functions or contracts. For example, external integration with a third-party marketplace, reentrancy could occur through the external call, especially if the contract fails to follow secure execution order of contract logic. As a result, contracts must include multiple safeguards alongside ReentrancyGuard.

4.2.3 Metadata Freezing and URI Protection

The metadata freezing and URI uniqueness mechanisms were effective in maintaining token integrity and authenticity. Freezing successfully blocked modification attempts, and duplication checks enforced non-fungibility. However, despite correct implementation, relying on off-chain solutions inherently has limitations. If the linked metadata is taken offline, the NFT effectively loses its content. While IPFS links a token URI to a cryptographic hash, which can alert users of tampering, it does not prevent the data from becoming unavailable. A smart contract cannot manage off-chain resources, making this an unavoidable limitation in current NFT architecture (Borgesi, 2024).

4.2.4 Supply Cap Enforcement

The supply cap was successfully enforced, preventing minting beyond the defined limit. This mechanism is important in NFT projects, where scarcity directly influences market value (Cole, 2024). A comparable example is seen in Bitcoin, where its fixed supply contributed to price inflation as it reaches its limit (Maina, 2024). However,

the supply cap is immutable once it is deployed, which makes it less flexible to future project changes.

4.2.5 Pause Functionality

The pause mechanism functioned as intended; when triggered by a wallet with appropriate roles, important functions like mint() return an error. This gives administrators more control and allows them to respond faster when vulnerabilities are detected.

Although this mitigates risk, it is criticised for impeding the principle of decentralisation; a central authority has control over the contract's behaviour (Behnke, 2022d). If an admin wallet is compromised, the pause function can be misused, for example, to manipulate market conditions. This relates to the rise of NFT rug pull scams, where trading can be potentially halted at peak valuation to maximise profit (Huang et al., 2023).

4.2.6 Burn Functionality

The burn function worked as intended, permitting only token owners or approved wallets to destroy NFTs. This safeguarded against the 'Public Burn' vulnerability, which allows any wallet to burn tokens, as seen in poorly configured contracts (Wang, 2024).

While burning does not directly improve security, it influences token scarcity and value. Similarly to the pause mechanism, it is a legitimate feature that can be exploited in NFT rug pull scams to inflate market prices by creating artificial scarcity, causing financial losses (Huang et al., 2023).

5. Future Work

While the implemented contract demonstrates how secure design choices can mitigate common vulnerabilities in NFT smart contracts, there are several areas where future implementations could address additional limitations, including broader testing and external integrations. Most notably, the smart contract was developed and tested exclusively within a simulated environment using Remix IDE. While this was convenient during development, it did not reflect real-world blockchain behaviour. Future work should include testing functionality on a live Ethereum network. Using tools such as the Sepolia testnet allows testing to occur on a separate leger, while still simulating harsh network conditions (Alchemy, 2023). This accounts for Remix IDE not being able to simulate variable gas costs, providing more accurate testing and transaction ordering. This provides a stronger insight into the performance of security features under real conditions, and potential vulnerabilities which may arise from this different environment.

In addition, the evaluation was based solely on interactions from a single user, which introduces a narrow testing bias. Future work should include a wide range of real-world users to evaluate how different wallet interfaces, devices, network conditions or browser extensions might affect contract interactions. Gathering user feedback can also be beneficial in discovering potential edge cases, informing future design decisions.

Automated security audit tools such as Slither should be integrated into development to reduce reliance on manual review. Slither is a static analysis framework that detects smart contract vulnerabilities and provides insights into code structure through control flow and inheritance graphs (Feist, Grieco and Groce, 2019). Incorporating tools like

Slither decreases the chance of vulnerabilities being unintentionally overlooked and aligns with industry-standard security practices.

While this project mainly focused on contract level threats, future work can expand on this by investigating how external contract integrations, such as NFT marketplaces, can impact overall security. Integrations can potentially introduce additional vulnerabilities, including reentrancy through external calls, and logic mismatches. Testing the contract alongside external contracts provides a more comprehensive overview of the vulnerabilities present and how integration may impact the reliability of NFT systems.

Role-based access controls were effective in preventing access to privileged functions, however additional safeguards can still be added. For example, requiring multi-signature authorisation for critical functions prevents a wallet becoming a single point of failure. This could be applied when pausing a contract; two administrator wallets need to approve a signature in order to change the state of the contract. This distributes control between multiple parties and reduces the risk of a compromised wallet disrupting a contract.

6. Conclusion

This project aimed to explore how common NFT smart contracts can be mitigated through secure-by-design principles and practical implementation. By reviewing literature, analysing real-world case studies, and building a functional Ethereum-based contract using Solidity and OpenZeppelin, I was able to demonstrate how preventative security strategies can be implemented into the contract's architecture. Features such as role-based access control, reentrancy guards, metadata freezing and supply caps were implemented in response to documented threats, including unauthorised minting and reentrancy exploits.

The evaluation showed that the features implemented behaved as intended under testing conditions, reducing the attack surface. However, testing was limited to a simulated environment, conducted by a single user, which restricts the generalisability of the results. The lack of automated auditing tools also introduces the possibility of overlooked vulnerabilities.

Despite these limitations, the project successfully showed that proactive smart contract design can reduce common vulnerabilities in NFT systems. The results highlight how secure-by-design principles can be applied in practice, showing that defensive coding can significantly improve a contract's resilience against documented vulnerabilities.

Future work should focus on testing in live environments, incorporating audits, and investigating integration with external platforms to gain insight into more complex attack vectors. Ultimately, this project contributes to the efforts in blockchain development to make NFT systems more secure against evolving security threats.

7. Appendices

Appendix A: Smart Contract Code

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;



// NFT standard with metadata

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

// access control (roles)

import "@openzeppelin/contracts/access/AccessControl.sol";

// reentrancy protection

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

// pause/unpause contract

import "@openzeppelin/contracts/security/Pausable.sol";



// contract with security features and access control

contract SecureNFT is ERC721URIStorage, AccessControl, ReentrancyGuard,
Pausable {
```

```
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE"); //
secure hashed role
    uint256 private tokenIdCounter; // token ID tracker
    uint256 public constant MAX_SUPPLY = 5; // hard cap on NFTs

    mapping(uint256 => bool) public metadataFrozen; // tracks which NFTs are frozen
    mapping(string => bool) private usedURIs; // tracks used metadata URIs

    // custom event logs
    event TokenMinted(address to, uint256 tokenId, string uri);
    event TokenBurned(uint256 tokenId);
    event MetadataFrozen(uint256 tokenId);
    event ContractPaused(address admin);
    event ContractUnpaused(address admin);

    constructor() ERC721("SecureNFT", "SNFT") {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender); // deployer is admin
        _grantRole(MINTER_ROLE, msg.sender); // deployer is allowed to mint
    }
```

```
    // mint NFTs (only minter, only if not paused, and metadata not frozen)
    function mint(address to, string memory uri)
        public
        onlyRole(MINTER_ROLE)
        whenNotPaused
        nonReentrant
    {
        require(tokenIdCounter < MAX_SUPPLY, "Max supply reached"); // check if supply cap is reached
        require(!metadataFrozen[tokenIdCounter], "Metadata is frozen"); // prevent minting if metadata is locked
        require(!usedURIs[uri], "Metadata already used"); // prevent duplicate metadata

        uint256 tokenId = tokenIdCounter;
        _safeMint(to, tokenId); // mint token safely
        _setTokenURI(tokenId, uri); // set metadata link
        usedURIs[uri] = true; // mark URI as used
        emit TokenMinted(to, tokenId, uri); // log mint
        tokenIdCounter++; // increment counter
    }
```

```
// freeze metadata for a token
function freezeMetadata(uint256 tokenId) public onlyRole(MINTER_ROLE) {
    require(_existsPublic(tokenId), "Token does not exist"); // safe check
    metadataFrozen[tokenId] = true; // lock metadata
    emit MetadataFrozen(tokenId); // log freeze
}

// burn an NFT
function burn(uint256 tokenId) public {
    address owner = ownerOf(tokenId); // get current owner
    require(
        msg.sender == owner || // must be owner
        getApproved(tokenId) == msg.sender || // or approved for this token
        isApprovedForAll(owner, msg.sender), // or approved for all tokens
        "Not owner or approved"
    );
    _burn(tokenId); // destroy token
    emit TokenBurned(tokenId); // log burn
}
```

```
// emergency stop functions (pause/unpause contract)
function pause() public onlyRole(DEFAULT_ADMIN_ROLE) {
    _pause(); // stop minting
    emit ContractPaused(msg.sender); // log pause
}

function unpause() public onlyRole(DEFAULT_ADMIN_ROLE) {
    _unpause(); // resume minting
    emit ContractUnpaused(msg.sender); // log unpause
}

// check if a token exists
function _existsPublic(uint256 tokenId) internal view returns (bool) {
    try this.ownerOf(tokenId) returns (address) {
        return true;
    } catch {
        return false;
    }
}
```

```
// testing functions for metadata freezing

    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal
override {

        require(!metadataFrozen[tokenId], "Metadata is frozen");

        super._setTokenURI(tokenId, _tokenURI);

    }



    function updateTokenURI(uint256 tokenId, string memory newURI) public
onlyRole(MINTER_ROLE) {

    require(_existsPublic(tokenId), "Token does not exist");

    _setTokenURI(tokenId, newURI);

}



    // support multiple inherited interfaces

    function supportsInterface(bytes4 interfaceId)

        public

        view

        override(AccessControl, ERC721URIStorage)

        returns (bool)
```

```
    {
        return super.supportsInterface(interfaceId);
    }
}
```

Appendix 2: Project Plan Template

UG Project Plan

CSC-30014

Project Overview and Description

Student Name: Nadia Wolek

Student Username: X8D36

Student Number: 22017452

Degree Title: Computer Science (Cyber Security)

Supervisor Name: Dr Aisha Junejo
Project Title: Mitigating Security Risks in NFTs

Please provide a brief Project Description:

- Explain how NFTs are used -> digital assets on chain, created/ bought/ sold through smart contracts, public transactions and ownership
- Explain security issues NFTs have as a blockchain based asset -> smart contract vulnerabilities including reentrancy attacks + inadequate code quality (e.g. floating pragma), social engineering attacks.
- Demonstrate how common security issues can be mitigated (reentracy attacks -> using a ReentracyGuard, using proper version control + access control)
- Code a secure NFT contract that successfully addresses common security issues (access control -> role-based permissions, data handling -> metadata protected from manipulation, control flow -> prevent re-entrancy attacks, etc.)
- Emerging technology and its impact on security (e.g. machine learning)

What are the aims and objectives of the Project?

- Explore key vulnerabilities in NFTs and provide ways they can be mitigated
- Develop an NFT Smart Contract that addresses the common issues mentioned
- Test the contract and show potential vulnerabilities I found
- Evaluate emerging technologies such as machine learning and their potential impact on security of NFTs -> both positively and negatively
- Use the findings of my own development to suggest potential solutions

Please provide a brief overview of the key literature related to the Project:

- Multiple studies on smart contracts and vulnerabilities, but not necessarily addressing the development process and how it can be prevented
- Studies on how NFTs can be used and utilised in different scenarios
- Research on different blockchains and which chains are more efficient for NFT deployment

Project Processes and Methods

Please provide a brief overview of the Methodology to be used in the Project (inc. an overview of best practice within the Methodology):

- Qualitative methodology
- Case study analysis
- Reviewing current literature

Will any special Data Collection Methods will be employed (e.g card sorts, questionnaires, simulations, …)?

/

Briefly describe how you will ensure your project is in line with BCS Project Guidelines (BSc Computer Science Single Honours Students only)?

- Following ethical guidelines
- Document all research
- All articles used will be referenced
- Clearly explained reasoning behind decisions made in code

Time and Resource Planning

Will Standard Departmental Hardware be used? YES/NO

If NO please outline the Hardware/Materials to be used:

/

Will Software which is already available in department be used? YES/NO

If NO please outline the Software to be used including how any necessary licences will be obtained:

/

Will the project require any Programming? YES/NO

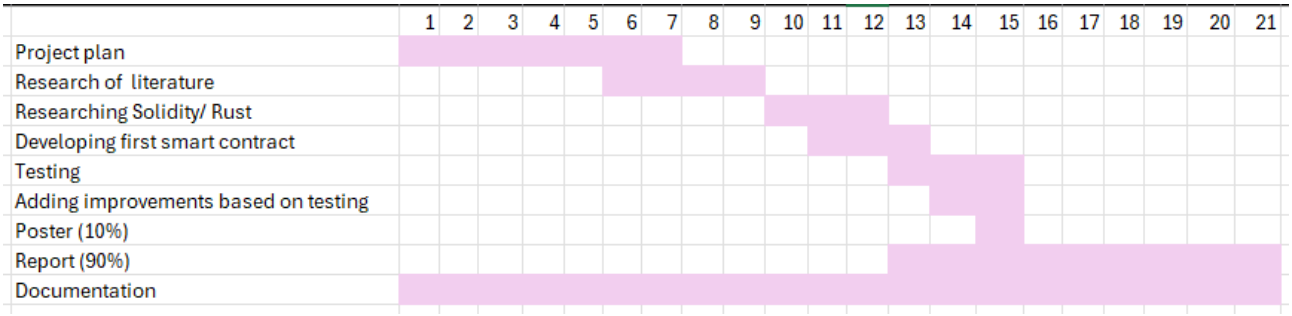
If YES please list the (potential) Programming Languages to be used (including any IDEs and Libraries you may make use of):

- Solidity
- Rust

Table of Risks (*if non Standard Hardware and/or Software to be used please also include backup options/ contingency plans here)*:

Risk-id Description	Probability/Likelihood of occurring	Best practice prevention measures	Remedy
Unable to access blockchain resources	**Low**	**Find tools needed early, research different types of tools**	**Look for alternative platforms**
Security risks in code	**Medium**	**Research best practices for security, review latest security issues**	**Ask for feedback, update code frequently,**
Bugs occurring whilst coding smart contract	**High**	**Research how to use libraries beforehand, research solidity**	**Ask for feedback, debug**
Plagiarism due to relying on existing code online	**Low**	**Ensure I use code bases online as guidance only, if sections are used reference it.**	**Reference, rewrite sections of code**

Gantt Chart (must include milestones and deliverables):



References

Please include a list of References used in this Plan (using Harvard reference style):

CoinFantasy (2024). *Remilia Founder Alleges Hack: Ether, NFTs Transferred*. [online] Medium. Available at: https://medium.com/invest-gaming-journal/remilia-founder-alleges-hack-ether-nfts-transferred-9fbbb1a283cc [Accessed 4 Nov. 2024].

Dixit, R.R. and Gokulapriya R (2023). A Comprehensive Investigation of Blockchain Technology's Role in Cyber Security. [online] doi:https://doi.org/10.1109/icaccs57279.2023.10113026.

Gagliardoni, T. (2021). *The Poly Network Hack Explained*. [online] Kudelski Security Research. Available at: https://research.kudelskisecurity.com/2021/08/12/the-poly-network-hack-explained/.
Gupta, N.A., Bansal, M., Sharma, S., Mehrotra, D. and Kakkar, M. (2023). Detection of Vulnerabilities in Blockchain Smart Contracts: A Review. [online] doi:https://doi.org/10.1109/cictn57981.2023.10140767.

Lage, O. and Saiz-Santos, M. (2021). Blockchain and the Decentralisation of the Cybersecurity Industry. doi:https://doi.org/10.6036/10188.
Qiu, M., Liu, X., Qi, Y., Zhao, H. and Liu, M. (2020). AI Enhanced Blockchain (II). [online] doi:https://doi.org/10.1109/smartblock52591.2020.00034.

Rot, A. and Blaicke, B. (2019). *Blockchain's Future Role in Cybersecurity. Analysis of Defensive and Offensive Potential Leveraging Blockchain-Based Platforms | IEEE Conference Publication |*

IEEE Xplore. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/document/8779855.

Salman, T., Zolanvari, M., Erbad, A., Jain, R. and Samaka, M. (2019). Security Services Using Blockchains: A State of the Art Survey. *IEEE Communications Surveys & Tutorials*, 21(1), pp.858–880. doi:https://doi.org/10.1109/comst.2018.2863956.

Wylde, V., Rawindaran, N., Lawrence, J., Balasubramanian, R., Prakash, E., Jayal, A., Khan, I., Hewage, C. and Platts, J. (2022). Cybersecurity, Data Privacy and Blockchain: A Review. *SN Computer Science*, [online] 3(2). Available at: https://link.springer.com/article/10.1007/s42979-022-01020-4.

Submission Date:

PLEASE NOTE THAT SHOULD YOUR PROJECT UNDERGO ANY MAJOR CHANGES FOLLOWING THE SUBMISSION OF THIS PLAN YOU ARE EXPECTED TO SUBMIT AN UPDATED PLAN WHICH ACCURATELY REFLECTS YOUR PROJECT.

Appendix 3: GDPR Checklist

Computer Science CSC-30014 Project GDPR and Ethics Checklist 2024-25

STUDENT NAME: **NADIA WOLEK**

STUDENT NUMBER & E-MAIL: X8D36@STUDENTS.KEELE.AC.UK

PROJECT TITLE: **Mitigating Security Risks in NFTs**

SUPERVISOR NAME: **Dr Aisha Junejo**

Complete all the questions below and electronically sign and date at the end. Ask your supervisor to check the responses and to also electronically sign and date below. You should submit this completed checklist to the KLE drop-box provided. Please retain your own copy of the completed checklist as all end of project reports/dissertations must provide a copy of this completed checklist in the Appendices.

If a Computer Science Final-Year Project Ethical Review Application Form has also had to be completed a copy of the final ethical approval certificate must also be included with your Project Plan document in your final report/dissertation Appendices.

GDPR Check

Does your project involve the use or collection of "personal data" for which permission will not have been explicitly granted? (Before answering, please read and carefully consider the GDPR Check Guidance at the end of this form).	No/Yes

Ethics Check

Will your project involve the use of human participants or capturing human data? (Before answering, please read and carefully consider the Significant Ethical Concern Checklist, below). **Please Note**: software evaluation by any other person counts as human participation. If you ask people their opinions about software or use their data in any way, you need to seek ethical approval first. **If you answered "Yes" to this question you must discuss your plans with your supervisor and, by end of week 12 of Semester 1, complete the Computer Science Final-Year Project Ethical Review Application Form and prepare a Participant Information Sheet and Consent Form using the templates that can be found at the end of that application form.**	No/Yes

Significant Ethical Concern Checklist

Could the project expose the participants or the project student to images and/or information that they might find distressing (e.g. pictures or descriptions of injuries, symptomatic health conditions, or atrocities, or pictures or descriptions of tumours or cancerous cells, or creatures in distress)?	No/Yes
Does the project involve deception of the participants?	No/Yes
Could the project uncover information about identifiable individuals that could cause embarrassment or distress to one or more of those individuals (e.g. evidence of illegal or unethical behaviour, such as fraud or illegal drug use or a personal revelation)?	No/Yes
Could the project cause pain, discomfort or risk to the participants and/or the project student?	No/Yes
Will the project involve participants who are vulnerable in any way? (e.g. participants who are under 18, or who are mentally or physically impaired, or participants who may feel under pressure to participate.)	No/Yes
Does the project involve recall or discussion of personal or sensitive memories?	No/Yes
Does the project involve a significant risk of participants later regretting taking part?	No/Yes
Does the project involve procedures which are likely to provoke interpersonal or inter-group conflict?	No/Yes

If the answer to any of the Significant Ethical Concern Checklist questions is "Yes" (or you think "Maybe") you must discuss your project and aims with your supervisor and with the CSC-30014 Module Co-ordinator to assess whether an appropriate level of ethical scrutiny might be required via the completion of the *Faculty of Natural Sciences (non-Psychology) Research Ethics Application Form*.

GDPR Check Guidance:

Personal data includes any and all of: names, addresses, emails, phone numbers, bank details, employment details, IP addresses, date of birth, medical or health data, images, video or audio recordings.

If you answered "Yes" you must not proceed with your project.

It is illegal under European GDPR legislation to make use of personal data without explicit permission. Discuss your project with your supervisor and revise your plans to ensure you do not risk illegal use of personal data.

Note. Even if personal data is publicly available on the Internet, it must not be used without permission. (Also note that you cannot contact individuals to request permission to use their on-line data without prior ethical approval to do so).

It is strongly recommended that you either:

1. use non-personal data for your project, or
2. use existing, well-established, publicly available databases or data repositories, for example: https://archive.ics.uci.edu/ml/index.php, https://physionet.org/ or https://www.kaggle.com/ etc. (A list of acceptable data repositories is maintained on the CSC-30014 KLE pages.) You might also see, for example, https://blog.scrapinghub.com/web-scraping-gdpr-compliance-guide) for further information.

I confirm that the responses are correct and that the project, as proposed, is GDPR compliant and that ethical approval will be sought if required and that any work requiring ethical approval will not take place unless ethical approval has been granted. If, during the course of the project work, any of the information supplied on this checklist changes substantially a new checklist will need to be completed and then brought to the attention of the School's Ethics Advisory team.

Signed (Student) Nadia Wolek	Date: 16/10/24

I confirm that I have read the form and that the project, as proposed, is GDPR compliant and that, to the best of my knowledge, the ethical information is correct.

Signed (Supervisor) Dr Aisha Junejo	Date: 17/10/24

Electronically typed signatures and dates *are* acceptable.

8. References:

Adejumo, O. (2022). *New NFT Marketplace LooksRare Launches, Suffers DDoS Attack*. [online] Yahoo Finance. Available at: https://finance.yahoo.com/news/nft-marketplace-looksrare-launches-suffers-082959106.html.

Alchemy (2023). *What is the Sepolia testnet?* [online] www.alchemy.com. Available at: https://www.alchemy.com/overviews/sepolia-testnet.

Ankit Gangwal, Apoorva Thirupathi, Alessandro Brighente and Conti, M. (2023). A First Look at Shill Looping in NFT Ecosystem. [online] pp.1–6. doi:https://doi.org/10.1109/wifs58808.2023.10374954.

Behnke, R. (2022a). *Designing Secure Access Control For Smart Contracts*. [online] Halborn. Available at: https://www.halborn.com/blog/post/designing-secure-access-control-for-smart-contracts.

Behnke, R. (2022b). *Explained: The OpenSea NFT Listing Vulnerability*. [online] Halborn.com. Available at: https://www.halborn.com/blog/post/explained-the-opensea-nft-listing-vulnerability [Accessed 3 May 2025].

Behnke, R. (2022c). *Explained: The OpenSea Phishing Hack (February 2022)*. [online] Halborn.com. Available at: https://www.halborn.com/blog/post/explained-the-opensea-phishing-hack-february-2022 [Accessed 3 May 2025].

Behnke, R. (2022d). *How to Pause a Smart Contract*. [online] Halborn.com. Available at: https://www.halborn.com/blog/post/how-to-pause-a-smart-contract [Accessed 4 May 2025].

Bîzgă, A. (2022). *Instagram Hacker Steals Millions Worth of Bored Ape Yacht Club NFTs*. [online] Hot for Security. Available at: https://www.bitdefender.com/en-

gb/blog/hotforsecurity/instagram-hacker-steals-millions-worth-of-bored-ape-yacht-club-nfts [Accessed 3 May 2025].

Borgesi, R. (2024). *The importance of IPFS when creating NFTs | Coinmonks*. [online] Medium. Available at: https://medium.com/coinmonks/exploring-the-ipfs-protocol-and-its-integration-with-nfts-a8e0e7c5acf2#9776 [Accessed 4 May 2025].

Brandom, R. (2022). *$1.7 million in NFTs stolen in apparent phishing attack on OpenSea users*. [online] The Verge. Available at: https://www.theverge.com/2022/2/20/22943228/opensea-phishing-hack-smart-contract-bug-stolen-nft.

CCI (2024). *What is NFT Burning?* [online] Crypto Council for Innovation. Available at: https://cryptoforinnovation.org/what-is-nft-burning/ [Accessed 4 May 2025].

Chakraborty, D. (2023). *How to Set Access Control for Smart Contracts*. [online] Hackernoon.com. Available at: https://hackernoon.com/how-to-set-access-control-for-smart-contracts [Accessed 4 May 2025].

Chhipa, M. (2024). *OpenZeppelin Reentrancy Guard: A Quickstart Guide - Mayank Chhipa - Medium*. [online] Medium. Available at: https://medium.com/@mayankchhipa007/openzeppelin-reentrancy-guard-a-quickstart-guide-7f5e41ee388f [Accessed 3 May 2025].

Cho, E., Jensen, G., Yoo, Y., Aniket Mahanti and Kim, J.-K. (2023). Characterizing the Initial and Subsequent NFT Sales Market Dynamics: Perspectives from Boom and Slump Periods. *IEEE Access*, 12, pp.1–1. doi:https://doi.org/10.1109/access.2023.3333897.

Coinbase (2021a). *How do you buy an NFT?* [online] @coinbase. Available at: https://www.coinbase.com/en-gb/learn/crypto-basics/how-to-buy-nft [Accessed 2 May 2025].

Coinbase (2021b). *What is a non-fungible token (NFT)?* [online] @coinbase. Available at: https://www.coinbase.com/en-gb/learn/crypto-basics/what-are-nfts.

Cole, J. (2024). *Understanding Tokenomics in Crypto: A Comprehensive Supply Cap Analysis - BlockApps Inc.* [online] BlockApps Inc. Available at: https://blockapps.net/blog/understanding-tokenomics-in-crypto-a-comprehensive-supply-cap-analysis/ [Accessed 4 May 2025].

Das, D., Bose, P., Ruaro, N., Kruegel, C. and Vigna, G. (2022). *Understanding Security Issues in the NFT Ecosystem*. [online] Available at: https://sites.cs.ucsb.edu/~vigna/publications/2022_CCS_StudyNFT.pdf [Accessed 4 May 2025].

De Best, R. (2024). *NFT - statistics & facts*. [online] Statista. Available at: https://www.statista.com/topics/8513/nft/#topicOverview.

Eskandari, S., Moosavi, S. and Clark, J. (2020). SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. *Financial Cryptography and Data Security*, pp.170–189. doi:https://doi.org/10.1007/978-3-030-43725-1_13.

Feist, J., Grieco, G. and Groce, A. (2019). *Slither: A Static Analysis Framework for Smart Contracts*. [online] IEEE Xplore. doi:https://doi.org/10.1109/WETSEB.2019.00008.

Guidi, B. and Michienzi, A. (2023). Delving NFT vulnerabilities, a sleepminting prevention system. *Multimedia Tools and Applications*, 82. doi:https://doi.org/10.1007/s11042-023-16087-1.

Holcomb, K. (2025). THE PARADOXICAL SOLUTION TO ENFORCE RESALE ROYALTIES THE PARADOXICAL SOLUTION TO ENFORCE RESALE ROYALTIES AND KEEP THE NFT MARKET DECENTRALIZED AND KEEP THE NFT MARKET DECENTRALIZED. *Technology & Arts Washington Journal of Law*, [online] 20(1), pp.1–8. Available at: https://digitalcommons.law.uw.edu/cgi/viewcontent.cgi?params=/context/wjlta/article/1349/&path_info=3__The_Paradoxical_Solution_To_Enforce_Resale_Royalties_And_Keep_The_NFT_Market_Decentralized__Holcomb.pdf [Accessed 3 May 2025].

Huang, J., He, N., Ma, K., Xiao, J. and Wang, H. (2023). *A Deep Dive into NFT Rug Pulls*. [online] Available at: https://arxiv.org/abs/2305.06108 [Accessed 4 May 2025].

Iberdrola (n.d.). *Smart contracts: contracts to formalise agreements in the digital age*. [online] Iberdrola. Available at: https://www.iberdrola.com/innovation/smart-contracts.

IBM (2021). *Smart contracts on Blockchain*. [online] Ibm.com. Available at: https://www.ibm.com/think/topics/smart-contracts.

Igiakong, G. (2022a). *Host NFTs on IPFS Instead of Cloud Services*. [online] Cloudinary Blog. Available at: https://cloudinary.com/blog/why-nfts-and-ipfs-are-a-great-pair [Accessed 4 May 2025].

Igiakong, G. (2022b). *Should I freeze my NFT Metadata? - Coinmonks - Medium*. [online] Medium. Available at: https://medium.com/coinmonks/should-i-freeze-my-nft-metadata-d50d015f2780 [Accessed 4 May 2025].

Immunefi (2023). *Hack Analysis: Omni Protocol, July 2022*. [online] Immunefi. Available at: https://medium.com/immunefi/hack-analysis-omni-protocol-july-2022-2d35091a0109.

Kadenzipfel (2019). *smart-contract-vulnerabilities/vulnerabilities/transaction-ordering-dependence.md at master · kadenzipfel/smart-contract-vulnerabilities*. [online] GitHub. Available at: https://github.com/kadenzipfel/smart-contract-vulnerabilities/blob/master/vulnerabilities/transaction-ordering-dependence.md [Accessed 3 May 2025].

Kaspersky (2022). *NFTs Definition & Explanation*. [online] /. Available at: https://www.kaspersky.co.uk/resource-center/definitions/what-is-an-nft [Accessed 3 May 2025].

Magic Eden (2024). *Understanding ERC-721 and ERC-1155 Tokens | Magic Eden Help Center*. [online] Magiceden.io. Available at: https://help.magiceden.io/en/articles/8975489-understanding-erc-721-and-erc-1155-tokens.

Maina, K. (2024). *Bitcoin Nears 21 Million Supply Cap as 19.8 Million Coins Mined - Brave New Coin*. [online] Brave New Coin. Available at: https://bravenewcoin.com/insights/bitcoin-nears-21-million-supply-cap-as-19-8-million-coins-mined [Accessed 4 May 2025].

Nightingale, C. (2024). *The Full Guide on Reentrancy Attacks in Solidity Smart Contracts*. [online] Cyfrin.io. Available at: https://www.cyfrin.io/blog/what-is-a-reentrancy-attack-solidity-smart-contracts.

Onesafe (2025). *OpenZeppelin: Making Blockchain Safer, One Contract at a Time*. [online] Onesafe.io. Available at: https://www.onesafe.io/blog/how-does-openzeppelin-transform-blockchain-security [Accessed 4 May 2025].

OpenZeppelin (2017). *Access Control - OpenZeppelin Docs*. [online] Openzeppelin.com. Available at: https://docs.openzeppelin.com/contracts/5.x/access-control [Accessed 4 May 2025].

OpenZeppelin (n.d.). *ERC721 - OpenZeppelin Docs*. [online] docs.openzeppelin.com. Available at: https://docs.openzeppelin.com/contracts/3.x/erc721.

OpenZeppelin (n.d.). *Security - OpenZeppelin Docs*. [online] docs.openzeppelin.com. Available at: https://docs.openzeppelin.com/contracts/4.x/api/security.

Pearson, J. (2022). *More Than 80% of NFTs Created for Free on OpenSea Are Fraud or Spam, Company Says*. [online] VICE. Available at: https://www.vice.com/en/article/more-than-80-of-nfts-created-for-free-on-opensea-are-fraud-or-spam-company-says/ [Accessed 3 May 2025].

Phung, T. (2024). *Smart Contracts Common Attack Vectors and Solutions* . [online] DEV Community. Available at: https://dev.to/truongpx396/smart-contracts-common-attack-vectors-and-solutions-244g [Accessed 4 May 2025].

Raka Widhi Antoro (2024). *Why Using OpenZeppelin in Smart Contracts Is Essential*. [online] DEV Community. Available at: https://dev.to/zororaka/why-using-openzeppelin-in-smart-contracts-is-essential-2gi4 [Accessed 4 May 2025].

Salem , H. (2024). *Hidden Risks: The Centralization of NFT Metadata and What It Means for the Market*. [online] Arxiv.org. Available at: https://arxiv.org/html/2408.13281v1.

Saxo (2023). *One year later: How proof of stake has changed Ethereum*. [online] Home.saxo. Available at: https://www.home.saxo/en-gb/content/articles/cryptocurrencies/one-year-later---how-proof-of-stake-has-changed-ethereum-21092023.

Singh, A. (2022). *Codedamn*. [online] Codedamn News. Available at: https://codedamn.com/news/blockchain/how-to-estimate-gas-prices-of-a-function-in-remix-ide [Accessed 4 May 2025].

Srivastava, S. (2024). *What Is Token URI and How It Works*. [online] Tatum. Available at: https://tatum.io/blog/understanding-token-uri [Accessed 3 May 2025].

Stepanov, M. (2023). *How to use Slither for auditing smart contracts*. [online] HackenProof Blog. Available at: https://hackenproof.com/blog/for-hackers/how-to-use-slither-for-auditing-smart-contracts.

Summers, S. (2023). *Ethereum and NFTs: A Comprehensive Guide for Beginners and Experts*. [online] nft now. Available at: https://nftnow.com/guides/ethereum-and-nfts-a-comprehensive-guide-for-beginners-and-experts/.

Tutorialspoint (2019). *Limitations of Remix in Ethereum*. [online] Tutorialspoint.com. Available at: https://www.tutorialspoint.com/ethereum/ethereum_limitations_of_remix.htm [Accessed 4 May 2025].

Verma, R., Chandrawanshi, K., Soni, G., Jain, G., Nigam, S. and Jain, N. (2024). Unveiling Security Vulnerabilities in NFTs: A Comprehensive Risk Assessment. *2024 IEEE 4th International Conference on ICT in Business Industry & Government (ICTBIG)*, pp.1–6. doi:https://doi.org/10.1109/ictbig64922.2024.10911117.

Wang, X. (2024). *AI-Based Vulnerability Analysis of NFT Smart Contracts*. [online] Available at: https://arxiv.org/html/2504.16113 [Accessed 3 May 2025].

Ye, C., Li, G., Cai, H., Gu, Y. and Fukuda, A. (2018). Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting. *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. doi:https://doi.org/10.1109/dsa.2018.00015.