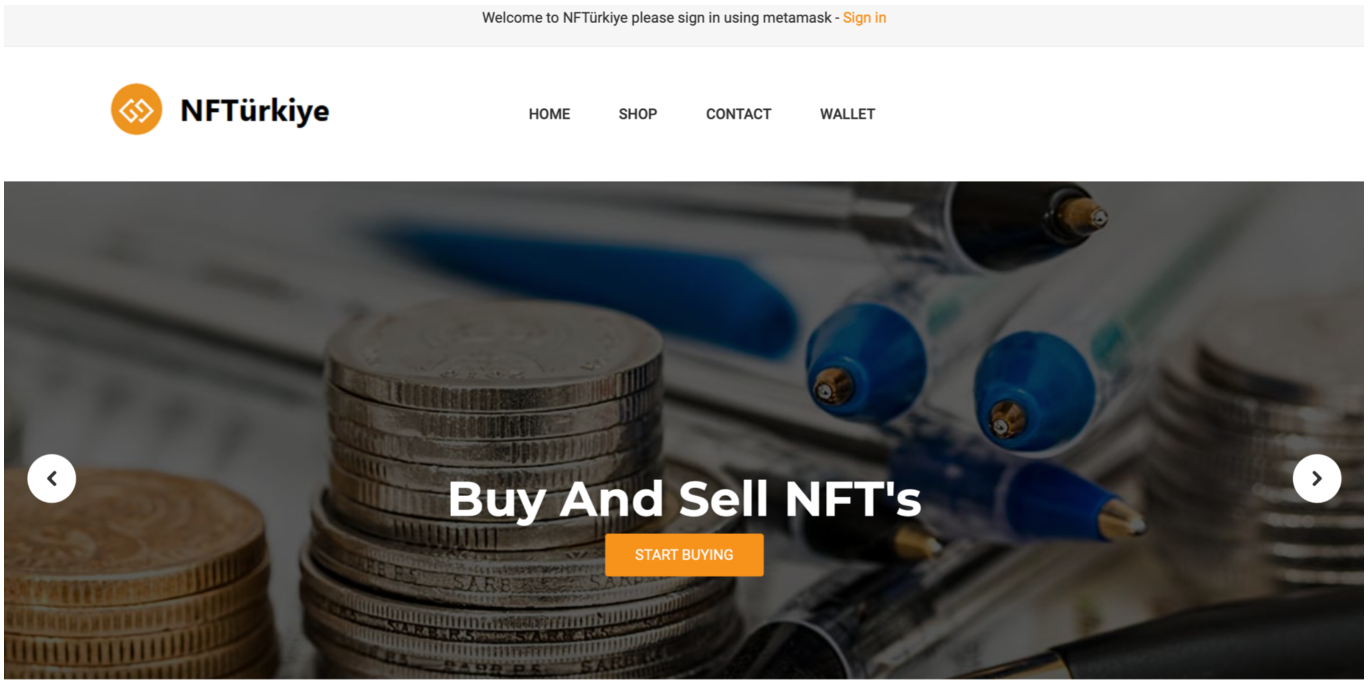
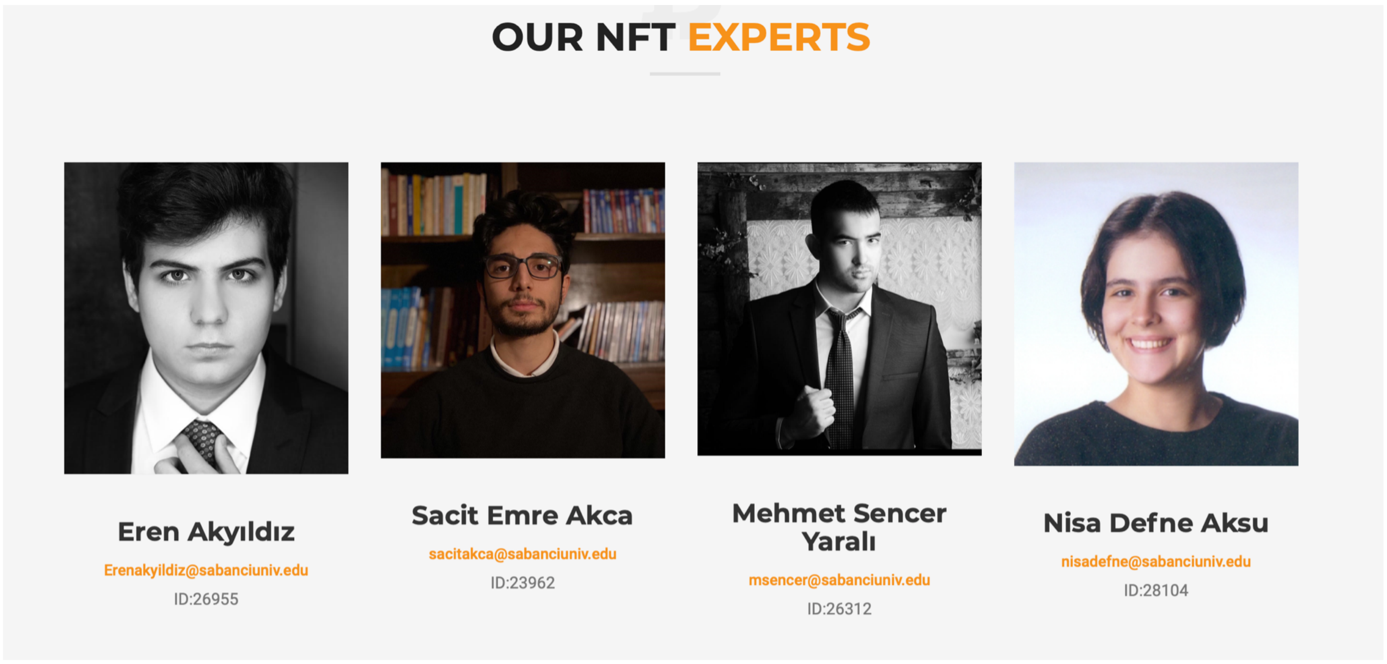
CS 48001 PROJECT FINAL REPORT



Welcome to NFTürkiye please sign in using metamask - Sign in

Buy And Sell NFT's



OUR NFT EXPERTS

Eren Akyıldız	Sacit Emre Akca	Mehmet Sencer Yaralı	Nisa Defne Aksu
Erenakyildiz@sabanciuniv.edu	sacitakca@sabanciuniv.edu	msencer@sabanciuniv.edu	nisadefne@sabanciuniv.edu
ID:26955	ID:23962	ID:26312	ID:28104

Group: Sacit Emre Akca, Eren Akyıldız, Nisa Defne Aksu, Mehmet Sencer Yaralı

WEBSITE

RECOMMENDED: Video explanation of everything done by Eren Akyıldız :

https://www.youtube.com/watch?v=G-QKqwC9X8Y

Pages that are not done by Eren Akyıldız: Contact.php, Add_to_db.php, Delete_from_db.php.

Everything else by Eren Akyıldız

Wallet.php: Wallet is supposed to store all NFT's that are owned by a user, the wallet firstly checks every possible NFT that is on the database to see if it can match a NFT on there with the metamask address of the user, this is very practical since there is no login required other than metamask, so the privacy of the user is secure, since nobody knows who owns that address, the user can choose the NFT they desire and they will be forwarded to the Sell_page.php of that NFT.

Sell_page.php: Sell page is for selling the NFT owned by the user, on this page the user can start an auction with a specific starting price, they can put their nft on the regular sale page by entering a price. This functions require the approval of the user then the specific function to run, so running a task takes a long time since block creation on ropsten is slow (better than btc at least) There is also a button for swap, but it does not work, since i could not implement it in time.

Shop.php: Shop page has 4 features, Swap(does not work on website), Auction, Regular sale and lottery. The first page you go to when pressed to shop is the regular sale page, the user can press any NFT they desire to buy them, if they press on one of them they are forwarded to that NFT's page on single_shop.php.

Single_shop.php: Single shop allows the user to see the description, name and price of the NFT, the user can buy this NFT by pressing the buy button and metamask will alert them for the transaction.

Auction.php: Auction page is similar to shop, the difference is that it shows maximum bids instead of prices, and if a token is clicked you will be forwarded to single_auction.php instead of single_shop.php.

Single_auction.php: Single auction is almost identical to single shop, the difference is you can not buy the token but you can bid on the token, you can see how much the last bid is, and also you can see the end time of this auction, if you try to enter this page and the auction has ended, you will be forwarded back to the auction.php.

Lottery.php: Lottery is very similar to single_shop and Single_auction, it holds one NFT and it allows you to buy Tickets for this lottery instead of offering a token.

Security.html: Just a page that talks about our security features, and is not complete. I wrote it 2-3 months ago.

**Also there are some unused swap pages.

DATABASE PART by Mehmet Sencer Yaralı

Add_to_db: This page is used from the wallet when an owner of a NFT puts his/her NFT for sale. This php page takes 2 request parameters for addresses of the NFT and the user, then after some checks, if everything is alright with the addresses, it inserts these addresses to the database "address_nft", which is the database we use for the NFTs that are for sale.

Delete_from_db:
This page is used for 2 purposes:

1) When an owner of a NFT is changed because of a regular sale. This page updates the database "userNFTs" with the new owner's address and deletes that nft from "address_nft", because it is no longer for sale. "address_nft" is the database we use for the nfts that are for sale, "userNFTs" is the database we use for the whitelisted NFTs that are allowed to be sold on our website.

2) When an owner of a NFT wants to cancel his/her listing. If this page is used for this purpose, it will only delete the NFT from the database "address_nft", it won't have an effect on the database "userNFTs" since ownership of the NFT is not changed.

SOLIDITY FUNCTIONS

Contract name: approvedTokenCreator

Contract location: approver.sol

By Eren Akyıldız

Video explanation of everything done by Eren Akyıldız:

https://www.youtube.com/watch?v=G-QKqwC9X8Y

1) Approver function works like a governance that only has one voter, me, the tokens that can be transferred, auctioned, or swapped have to be approved by this contract, this is done by hard coding this contracts address in our NFT contract, by doing this kind of security we are preventing attackers from ever being able to request a transfer with an approved NFT.

Weakness: The attacker may create their own NFT and start any function they desire, but a regular user of our website will not be able to swap their tokens with them. Since our website's database only holds the tokens that we have created.

Contract Name: newNFT

Contract Location: NFT.sol

By All Of Us

Eren’s Part 1: Approver

Before anything i want to talk about the approver function in the NFT contract, to be able to run any transactions, this function has to be run, this function takes approved token addresses from the approver function from approver.sol, and sets them as approved NFTs.

Eren’s Part 2: Regular sale

The regular sale is done by 3 functions on this contract; setprice, approve and transferNft;

Setprice: Setprice function sets the price for this token, it checks if the token is owned by the sender, if it is it sets a price for this token.

Approve: approve works as a way to exchange the token with ether, when approved, a token can be sent from this contract to another person. This can be done better with an escrow feature but I wanted it to look simpler for the users.

transferNFT: This function firstly checks the validity of the token, if this token is a token that has been created by us, it continues, afterwards it checks if the seller has approved this tokens sale, if not, this function reverts, afterwards it checks if enough eth has been sent to this contract, if it has enough ether it will send the ether to the NFT' s current owner, and send the NFT to the user who interacted with the contract.

Eren's Part 3: Auction

The Auction is done with 3 functions; bid, Auction, EndAuction.

Auction: This function starts the auction, Firstly it checks if the token that is being auctioned is created by us, afterwards it checks if the tokens owner is trying to start this auction, afterwards it checks if the auction is already sent for an auction or not, then it checks the price, the price must be larger than a number for good bidding purposes, owner before auction is set to the auction starter, the NFT is sent from owner to the contract for escrow, auctionended is set to false signaling the other functions to be able to start, and some functions such as transferNFT to not be able to start, maxbid is set to the owners desired bid, current winner is set to the owner so if no one wants the token, it can be sent back to them and is not lost in the contract, and lastly auction end time is set for 1 day from the creation of this block.

Bid: This function is for bidding on an auction it requires the user to send more than the maximum bid at the time + admins desired amount (for fair bidding in my case it is 0,002 ETH) , then it checks if the auction has ended or not, if it has ended, it does not allow anyone to bid anymore. After the checks the old maximum bid is transferred back to the owner (this does not create any problems since the auction starter will get all the money anyways, better explained at video timestamp : 19.00 - 19.45) maxbid is set to the current amount of bid sent to the auction, and current winner is set to the senders address.

AuctionEnd: This function is for ending the auction, it will only check the timestamp, if the time is not over yet, it will just return true (this is for bid part, to avoid bidding on a ended auction), if it has ended, the contract will send the token to current winner, and maxbid - start bid in the contract to the auction starter IF the current winner is not the auction starter meaning that there is no bidder, if that is the case, the auction starter only gets their token back and no money.

SWAP PART & CONTACT PAGE OF WEBSITE by Nisa Defne Aksu

1. **"Swap"** function can be used only by the NFT owner of tokenid number 1 of the contract. Swap proposal collection period starts automatically. This status is kept in a boolean variable.

Ownership is changed from real owner to the swap proposal collection period starting contract, i.e. this contract temporarily. This simulates an escrow role till the end of swapping progress. In order to start a swap proposal collection period, the NFT owner must approve our contract transferring tokens on the owner's behalf. To do this, there is another special function "approve". The owner must run this function and must enter our contract address and token id as the parameter before using the "swap" function.

2. **"SwapRequest"** function can be used by swap proposers. These proposers are the owners of other NFTs in other contracts. As the parameter, they must enter their NFTs contract address. First, this function checks the ownership of the proposed NFT and the current user of this function. Proposed NFTs contract is checked, whether it is created by us or not. It must be an approved contract. Also the swap proposal collection period must be still on. If all these requirements are fulfilled then the proposer's address and NFT's address are stored in a struct for the next phase. Of course there can be any number of proposers. There is no limit. Same proposer can propose different NFTs too. Same approving rule which we mentioned above is valid for all the proposers. They must use the "approve" function in their contracts to approve swap proposal collection period starting contract address to be able to transfer tokens on the owner's behalf. "SwapRequest" function changes the ownership from proposer's address to the swap proposal collection period starting contract address.

3. **"SwapAgree"** function can be used only by the NFT owner of token id number 1 of the contract, i.e. "swap starter". As the parameter, the owner enters the proposal id which isselected for swapping. There must be at least 1 proposal. Entered proposal id must be an existing proposal. It mustn't be withdrawn. Message sender must be the owner before the swap proposal collection period. If all these requirements are fulfilled then this function changes the owners of the NFTs. Hence the swap proposal collection starting contract is the owner of both NFTs, it can exchange the real owners, as the new owners. Swap proposal collection period ended automatically. Other parameters are reset to defaults. Owner of the NFT can start a new swap collection period and swap the current NFT with another NFT proposal, which is still existing in the proposal list.

4. **"swapWithdraw"** function can be used by the proposers. As the parameter, they enter theirproposal's id. This id is checked against the original ownership. Before swapping has been completed or afterwards, whenever the original owner decides to withdraw the proposal, ifit is not selected for swapping, there is no restriction to cancel it.

** While writing the contact page, I changed the html code of the website we received as a template and set our personal information to appear on the contact page as a slideshow. The photo with our pictures, ID's and names on the first page of the report is a screenshot of that part.

LOTTERY PART - Functions by Sacit Akca

We decided to give away some tokens with a lottery-based approach.

Function Raffle:

The main function that starts the lottery is the “Raffle” function. It takes the address of the token that the owner wants to give away with a lottery, the price of one ticket in wei as parameters. It checks the owner of the token and if there is an ongoing raffle for the token. Thereafter, it does the required initializations for the raffle, such as setting up variables for ticket price and lottery end time.

Function buyticket:

Function for buying tickets, users can specify the amount of the tickets that they want to buy with the parameter “ticketsWanted”. First, it checks if there is an ongoing raffle and the value sent to the function is enough for buying the number of tickets wanted. If the buying operation is successful it pushes the address of the buyer by the number of tickets purchased to an array defined as “raffleArray”. If there are fewer tickets remaining than the user wants, it gives all remaining tickets to the user. In both cases, the user’s unspent ethers will be refunded.

Function Raffle_end:

The function that is used for checking if the raffle has reached its end time by comparing the current block’s timestamp with the defined end time of the raffle. If the raffle has reached its end time, this function randomly picks an index of the “raffleArray” and sends the token to that address.

Function generateRandomNumber:

One line code that generates a 256bit random number. It’s used while picking the winner of the raffle. We found a way to generate truly secure random numbers by using Chain Link Services, but since there is a service price for every random number request done, we didn’t implement Chain Link method in this project and instead we used a non-truly random, random number generator.

INFORMAL SECURITY ANALYSIS

Includes all the security precautions that were taken while creating the project in total. Contracts, website usability and user safety were considered while doing this project. Necessary “require” statements and contracts were added as well.

1- Regular sale:

* Regular sale had some security issues. I will start with the approval problem, the transfer requires the user to approve their token sale, but this can be undone, to prevent this from happening we have used a require().

* The regular sale had a second problem, upon sale, the price would not change, so i set the price to a very large value, (you need about %90 ether in existence), when the user sets the value, then it can be used normally (the website approves and starts sale one after another).

2- Auctions problems:

* I actually did not think that auction had any security issues, but there are some major problems with auction, we need to use a withdraw function instead of sending money back to old bidder with a bidding function, since it can softlock and make auction unusable.(By writing a contract that reverts upon receiving a payment, the attacker can stop people from bidding on the auction, a very nice idea that I did not think of.)

* Also there was another problem with bid, the auction starter used to receive all the money he set before auction upon the first bid, and the auction ending function tried to send the auction starter the maxbid, which the contract did not have(since it already sent some of the money back), this was a security issue, since if we wanted to take a cut from the auctions, we had to have some money in the contract, the attacker could start an auction, and then wait for it to end, after some time (when there is enough money in the contract) the attacker can end the auction, and get the amount they won + the amount they initially started the bid for.

3- Swap problems:

* Swap does not have the same problem as auction, it has a withdraw function, also all the NFT’s that are sent to the swap are by us (this is by default, since the only way to request a swap is to send an NFT that is approved by us), so the attacker can not just write a contract from 0 and try to attack swapAgree function (swapAgree has a part where the NFT’ s that are not agreed upon are sent back to their original users, if the contracts were not approved, this can be manipulated just like the auctions bid() function, making it unable to send back any NFT that were sent to this contract.) .

* There was another problem, that was fixed, the problem was that an attacker can use their own minted NFT’ s to request a swap. This was a big problem, since we can not hardcode what

the approved NFT addresses are (since they have to be created first, removing the possibility of us knowing the addresses before creation), this problem led to me creating a approval contract and a function that uses that approval contract to set the approved NFT's, so i hardcoded the approval contracts address into the NFT contract, after creating the NFT's I simply added their addresses to the approved contracts list, and by calling the approval function, this issue was fixed.

* Swap() function has a weakness, even if the owner does not approve their token, they can start the swap process, setting themselves as the owner, even though the token is not even in the contract, this can be solved by simply adding a requirement of approval before everything else.

* Add "buy now price" to listings (could be a good idea).

* Swap multiple tokens in one transaction.

* Measure gas usage.

* Control concurrency between "**swapAgree**" and "**swapWithdraw**" functions. If "**swapWithdraw**" is running, "**swapAgree**" must wait. Race conditions can be stopped using a mutex control.

* Exception handling must be done better.

* Set a total limit for swap requests against gas limit reached, unexpected throw, unexpected kill, access control breached problems.

* Validate arguments before its use against buffer overflow attacks.

* Instead of using delete function, use pop function to control swappers array increasing uncontrolled against denial of service attacks.

* Add new functions like "**swapCancel**" for swap proposal collection period.

4- Lottery Problems:

*Lottery uses a random number (which is not securely random) function to pick the winner and in order to generate that random number, it hashes the block stamp and applies mod operation on it. Since the block stamp is known, the winning number can be computed easily. Therefore this is not a secure implementation for deciding the winner. Alternative and secure way to generate random numbers is using oracle services. Chain Link is one of those services, the reason why we didn't use Chain Link in this project is because each of a random number generation request consumes 0.1 LINK token.

FORMAL SECURITY ANALYSIS:

Tool used for formal security analysis: **SOLIDITY STATIC ANALYSIS for remix IDE**

The formal security analysis showed us that our code has 147 potential vulnerabilities including problems with gas and economy.

SECURITY:

1- Checks-Effects-Interactions This is not a vulnerability for approve(), Auction(), auctionEnd(), Raffle() and swapWithdraw() function, but it is a vulnerability for TransferNFT(), Bid(), Swap().

Any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed.

From:
https://docs.soliditylang.org/en/v0.8.0/security-considerations.html#re-entrancy

2-Inline assembly, this is caused by the NFT contract, not our additions. When writing a library this is useful, but in our case it should be removed.

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. You should only use it for tasks that need it, and only if you are confident with using it.

From :
https://docs.soliditylang.org/en/v0.8.0/assembly.html

3- Block timestamp creates a security risk, since miners can influence the time the txn is mined we used this in our lottery, which makes it very vulnerable to attacks.

Do not rely on `block.timestamp` or `blockhash` as a source of randomness, unless you know what you are doing.

From:
https://docs.soliditylang.org/en/v0.8.0/units-and-global-variables.html?highlight=block.timestamp#block-and-transaction-properties

GAS & ECONOMY:

1- In retrieving data from the approver contract, we are using loops. Loops should be avoided since they may run indefinitely and will cause gas problems, the biggest problem is that if gas exceeds the block gas limit, that function will never be able to run, (so if we add a lot of NFT's to our approver, the function will not be able to run after a certain amount of NFT's).

2- The gas cost also increases when we try to allocate memory to variables, this makes our contract very expensive, since we use memory allocation a lot. (Creating different contracts and including functions in NFT's is good for this actually, since almost every variable will only have 1 value, unlike a transfer contract, swap contract, auction contract, all of these contracts will require a lot of storage since they must use big data structures for transfers such as mapping, dynamic arrays.) .

3-Using "this" to call functions on our contract makes no sense, we should have just called the contract, it just consumes more gas and not needed.

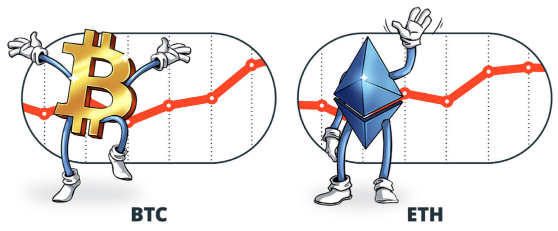
4- Deleting dynamic arrays, Deleting dynamic arrays may also cause the gas limit to exceed the blocks gas limit, if the array is too large. Also deleting will cause the array to have a gap, which could have been used for something else. Using a mapping is a way better approach to this problem.

`delete a` assigns the initial value for the type to `a`. I.e. for integers it is equivalent to `a = 0`, but it can also be used on arrays, where it assigns a dynamic array of length zero or a static array of the same length with all elements set to their initial value. `delete a[x]` deletes the item at index `x` of the array and leaves all other elements and the length of the array untouched. This especially means that it leaves a gap in the array. If you plan to remove items, a mapping is probably a better choice.

From:
https://docs.soliditylang.org/en/v0.8.0/types.html#delete

MISCELLANEOUS:

-Similar variable names
-Guard conditions: Use assert() instead of require() for approvedTokenCreator’s onlyContract modifier, NFT’s setprice() function, TransferNFT() function Bid() function, Auction() function.
-Do not use delete on an array, since it leaves a gap(i don't know why this was also considered miscellaneous, since we talked about it in gas and economy part.)



THANK YOU VERY MUCH FOR YOUR TIME AND ATTENTION