Security in Ethereum GSN

Kanodia Ritwik
Nanyang Technological University
Singapore
ritwik002@e.ntu.edu.sg

Fazili Numair
Nanyang Technological University
Singapore
numair001@e.ntu.edu.sg

Gupta Jay
Nanyang Technological University
Singapore
jay002@e.ntu.edu.sg

Abstract- When a user in the Ethereum network sends transactions, it incurs a gas fee which goes as a reward to miner for mining the transaction. This gas fee can typically be only paid using the Ethereum networks native token ETH. This poses a serious usability problem. All token holders who own tokens other than ETH, like ECR20, need to bear the additional over head of acquiring ETH only to pay for the gas fee even if they are paying for the transaction using other tokens. This also increases the barrier of entry into the network as acquiring ETH can prove to be a daunting task given the compliance and regulatory requirements like KYC specially for new users. In this paper, we present how to implement a Gas Station Network. It is a network implemented with the help of OpenGSN framework which provides mediator contracts that can send the transaction on the behalf of the user such that they do not need ETH. The first part of the paper offers an in-depth analysis of the various components followed by implementation details. The second part of the paper addresses possible security vulnerabilities in this framework. It discusses some of the attacks that might be launched by the attackers in the Gas Station network, the impact it can have and clearly lays out if and how they can be mitigated.

Keywords—Ethereum, security, contracts, gasless, relayer, paymaster, attacks, mitigations

I. Introduction

The Ethereum blockchain is essentially a transaction-based state machine [1]. When a transaction is fired in the network, it is first checked for validity. If it is valid, it transitions the state of the Ethereum state machine.

For a transaction to be considered valid, it must go through a validation process known as mining. Mining is when a group of nodes expends their resources to compute a mathematical proof of validity known as proof of work and create a block of valid transactions. This computation occurs because a transaction incurs a gas fee [2] which must be paid using the Ethereum Network's native token ETH which presents a usability problem. Ethereum supports a wide range of tokens and other digital assets. However, nodes in the network must own ETH tokens to pay for the gas fee which leads to the additional overhead of passing KYC and acquiring ETH before users can use a decentralized application. This also adds a barrier to entry to the network and motivates the following question: how can we enable users to transact in the Ethereum Network without paying the gas fee in ETH?

In this paper, we take the first step to explore answers to this question. If the user does not want to pay the gas fee in ETH, some alternatives must be introduced to pay using tokens other than ETH or subsidize the gas fee by smart contracts to reduce the entry barrier to the network. Ideally, any of these solutions should work as long as the sender is not forced to acquire ETH. This would increase the flexibility of the network.

Formally, we seek to explore the OpenGSN [2] framework which allows senders to send transactions without having to pay for the gas fee in ETH by introducing intermediate third-party contracts and servers that pay and send the transactions on the user's behalf. These transactions are referred to as gasless transactions [2]. The first half of this paper offers an in-depth analysis of the framework and the implementation details of the Gas Station Network in our project.

These third-party contracts operate on the existing Ethereum network, increase usability, and reduce the entry barrier to the network. Users can directly send transactions without having to own ETH. However, introducing this new service bring its own set of security concerns. A blockchain network is as secure as its weakest link and if these new components are vulnerable to attacks, the entire Gas Station Network is at risk. These are addressed in the second half of this paper where potential security issues and attacks on the GSN network are identified, analyzed, and how these can be mitigated.

II. Gas Station Network

A gas station network is a network that allows users to execute transactions without having to pay the gas fee in ETH. In this paper, we discuss the GSN implementation by the OpenGSN framework [2] which provides helpful tools and methods to implement such an infrastructure. The project corresponding to this paper implemented an end-to-end pipeline to enable users to execute transactions using smart contracts without requiring ETH.

A. Components

These are a few of the components including contracts and servers which are introduced to the Blockchain network to enable users to send gasless transactions. This section lays out the terminology and a brief description of their functionality.

Client or Sender – An external account which may hold tokens but does not pay ETH for transactions.
Relay Recipient – The recipient contract that receives the transaction from the user.
Relayer – The middleman between the sender and the recipient that enables the gasless transaction by paying for the gas.
Paymaster – It is the contract that verifies the user transaction based on some business logic and compensates the relayer for the transaction.
Relay Hub – The Hub that mediates all the calls between the different contracts.
Forwarder – A component that validates the users request by checking the signature and nonce before forwarding it to the recipient.

B. Implementation

The architecture of the Gas Station Network implemented for the project corresponding to this paper is shown in Figure 1.

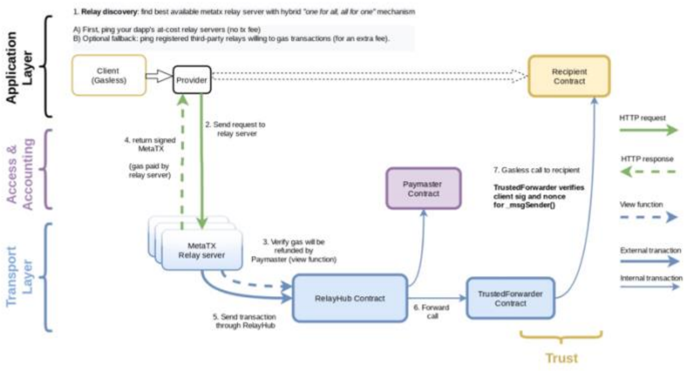
1. Relay discovery: find best available metatx relay server with hybrid "one for all, all for one" mechanism
A) First, ping your dapp's at-cost relay servers (no tx fee)
B) Optional fallback: ping registered third-party relays willing to gas transactions (for an extra fee).
Application Layer
Access & Accounting
Transport Layer
Client (Gasless)
Provider
Recipient Contract
2. Send request to relay server
4. return signed MetaTX
(gas paid by relay server)
Paymaster Contract
7. Gasless call to recipient
TrustedForwarder verifies client sig and nonce for _msgSender()
MetaTX Relay server
3. Verify gas will be refunded by Paymaster (view function)
RelayHub Contract
6. Forward call
TrustedForwarder Contract
5. Send transaction through RelayHub
Trust
HTTP request
HTTP response
View function
External tranaction
Internal transaction

Figure 1: OpenGSN Architecture [2]

In the OpenGSN framework, when a client wants to send a gasless transaction to a recipient contract, both operating in the application layer, it sends a signed message (a.k.a meta transaction) to the Relay Server which verifies the contents of this message with the paymaster contract. Once the transaction is verified, The Relay Server signs and returns the meta transaction to the user and sends the corresponding transaction to the relay hub by paying the gas fee. The Relay Hub then uses a Forwarder contract to forward this transaction to the recipient contract. The paymaster validates the transaction using some custom-defined business logic (whether to pay for the transaction) and transfers the tokens from user to recipient contract. The paymaster is also responsible for compensating the gas fee paid by the Relay Server. The RelayHub uses a Forwarder contract to send the transaction further to the Recipient contract. This Forwarder contract verifies the signature, checks if the nonce in the transaction is unique, and forwards it to the recipient. The RelayHub is responsible for bringing these components together which does not require any trust assumptions among the participating entities.

The merits of the OpenGSN framework are that it provides and abstracts the implementation of the RelayHub and the associated smart contracts. While ideally, each recipient contract should have its relay server, for ease of implementation, the project uses one of the relay servers hosted by OpenGSN.

The project provides the implementation for a simple recipient contract that provides two methods for invocation. One is the noCommisionTx() method which charges the user a single Token for execution while the gas fee is subsidized by the paymaster. The other is the tokenCommisionTx() method, which charges one Token for execution and another one as the gas fee. These methods are for demonstrating the two possible settings of the GSN Network. The contract should inherit the BaseRelayRecipient base class from the OpenGSN framework to be able to receive relayed transactions.

The project provides two implementations of the paymaster which can again choose depending on the use case. They are No Fee Paymaster and Token Fee paymaster. As the name implies, the No Fee Paymaster is invoked when the transaction is subsidized as no commission is charged by the paymaster. The paymaster contract only checks if it has enough allowance to draw the required number of tokens for the transaction from the sender's account to the recipient contract. The Token Fee paymaster on the other hand has the same business logic but it also charges the client's account using a flat fee in the token currency as a substitute for the ETH it must pay for the gas. These components, while highly beneficial have security vulnerabilities and are discussed in the subsequent half of this paper which analyses plausible attacks on the Gas Station Network.

III. Security in Gas Station Network

With time, new security attacks are emerging which are sophisticated and can cause significant damages. These attacks can overwhelm the Blockchain infrastructure at different levels of the P2P Network, Consensus, Runtime, and the application layers respectively. The network layer is susceptible to eclipse attacks where the adversary takes control of all the immediate neighbors of a node and controls its entire view of the system or the Sybil attack where one adversary assumes multiple identities to influence the network. In the consensus layer, selfish mining is prevalent where the miner holds back mined blocks and discreetly mines the next one. Another popular attack is the 51% attack, where malicious miners hold 51% of the effective mining power in the blockchain, preventing specific transactions to be mined or reversing older transactions by forking. Smart contracts too may be vulnerable to some security Vulnerabilities as in the case of the Reentrancy bug, where a malicious contract may call back into the calling contract and take over the control flow.

When we discuss gasless transactions between nodes or contracts, we are mainly dealing with the application layer and the Transport layer. Thus, the scope of the paper is limited to security in those two layers. The Gas Station Network introduces new contracts and servers to the network such as the Relayer and the Paymaster. The benefits of adding these new parties to the network come with its own set of possible security issues. Since these contracts are offering a service, there are different Denial of Service attacks that may come up between the parties. The exchange of financial assets, for example, the Relayer paying the gas fee and the paymaster compensating the Relayer or the user sending tokens as payment for the gas fee also invite financial attacks to the network.

A. Denial of Service attacks

A Denial-of-Service attack is an attack meant to shut down a server or network [3], making it inaccessible to its intended user. It may be accomplished by flooding the service provider with unwanted traffic or sending information that triggers the crash. It may also be that the service provider may selectively not offer its services to specific nodes. For example, in a typical blockchain network, the miners may refuse to mine the transactions of a particular node. It may be incentivized either by some financial or crypto economics condition. In the Gas Station Network, the Relay and Paymaster are the main service providers. They may themselves launch DoS attacks or be suspectable to flooding, rendering them ineffective. The following section delineates the possible DoS attacks in the Gas station network and how can they be mitigated.

(i) Relay launches a DoS attack on Sender

The user sends a transaction to the relayer. The relayer is then supposed to sign and return the transaction to the sender immediately and put the transaction in to the block chain. In this pipeline, the Relayer may launch a Denial-of-service attack against the user. It may refuse to sign and return the transaction to the user. The user can detect such an attack if it does not receive a signed transaction from the Relayer it sent the meta transaction to within a few seconds. They can simply switch to a different Relayer immediately. At most, the user will just lose out on a few seconds. In our code, we can also implement a local array storage where the sender contract can store a list of such relay address in its private storage and avoid sending transactions to them in the future. However, it is important to note that blacklisting only works if there is a way to prevent Sybil attacks or the attacker may just create new addresses. In the OpenGSN framework, sybil attacks is avoided by making the relay include a stake in the relay hub during the registration process. This stake is also used to penalize the relay for bad behavior and is only returned when the relay unregisters.

It may also be the case that the Relayer signs and returns the meta transaction to the user but publishes a different transaction to the blockchain with the same nonce. In this scenario, the user will have a signed transaction from the Relayer with nonce N and get a mined transaction from the blockchain with the same nonce N. If this happens, the user will immediately detect a DoS attack. Since it is a uniquely attributable fault of the Relayer address the user sent the original transaction to, a penalization scheme can be devised. This project inherits a Penalizer contract which is provided by OpenGSN. If which has a Penalize function to check if the two transactions are signed by the same Relayer and if their nonce are the same. If the condition is met, the Relay is unregistered and its stake in the RelayHub is confiscated and shared with the reporting party thus incentivizing anyone to report offending relays.

(ii) Paymaster launches a DoS attack on Relayer

As outlined earlier, in the project implemented corresponding to the paper, the Relay server pays the gas fee temporarily for the user and is later refunded by the paymaster contract. If the paymaster contract turns out to be malicious, it can verify transaction such that the relay pays the transaction fee for them but then not refund the relay. As a result, the relay will suffer a small loss. If this happens, the relay will know that the paymaster has acted inconsistently. If the paymasters acts inconsistently gain in the future, the relays can blacklist the offending paymaster. Sybil attack is practically unlikely as registering a paymaster requires a payment of a stake to the Relay Hub and can be expensive.

(iii) Attacker tries to reduce the availability of relays

The attacker may register multiple unreliable relay servers with the Relay Hub to reduce the availability of the relay servers. The user may think that the relays are launching a DoS against them. If the unreliable relays form a significant portion of all the relay servers, it is more likely the users meta transactions will end up with unreliable relays, causing a few second delay each time. This attack is however only theoretically possible. The GSN is designed as such that there are many relay servers. Therefore, to reduce the network availability of the network, the attacker will have to register a significant number of unreliable relays. This Sybil attack is again not practical as it will be expensive because when a relay is registered, it needs to pay a stake to the Relay Hub. If a relay server acts unreliable multiple times, users can also blacklist the server.

B. Financial Attacks

Financial attacks are attacks which are mainly targeted at financial assets. Double spending through 51% attack may be a typical example of a financial attack. In the Open Gas Station Network, financial assets are being moved between the different parties. The Relay pays the gas fee for the user, the paymaster compensates it, the user sends tokens to paymaster etc. This opens possibilities of financial attacks where one contract may try to deplete the funds or steal the assets of others. Some of the probable financial attacks on the OpenGSN network are discussed below.

(i) Paymaster exploits users Token Allowance

Allowance is a permission the sender gives to a smart contract to spend tokens from its account. In the OpenGSN network, in our implementation, the sender gives an allowance to the paymaster smart contract. The paymaster smart contract can use this allowance to compensate for the gas fee paid in ETH or to forward the senders token to the recipient smart contract. This can be a possible security issue especially if the paymaster contract turns malicious. It may then spend the senders token arbitrarily and it will be difficult to fix this. The effect of this attack becomes worse if the sender sets an unlimited allowance for the paymaster. While it may make the senders job easier as it will not have to send multiple transaction to change the allowance, the paymaster might exploit it and spend all the tokens in the sender's wallet. One way to mitigate this attack is to set small allowances. The incentive for paymaster contracts to turn malicious will be less and even if they do turn malicious, the tokens lost will be few.

(ii) User tries to empty Dapp by registering its own Relay

Dapps may choose not to charge the user for the gas fee to incentivize transactions and reduce the barrier to entry for new users. However, a malicious sender may abuse this by registering its own relay and sending meaningless transactions to deplete the paymaster. The networks paymaster may soon lose all its token and not be able to compensate the gas fee for meaningful transactions. To avoid such a situation, a whitelist paymaster can be implemented which validates and allows only verified user addresses to relay their transactions. Despite this, our network may still be vulnerable to sybil attacks as the malicious sender may register with more than one address. We can modify our code to require a captcha during user creation or to login with any verified third-party accounts which can limit the rate of such attacks. Alternatively, the best solution would be to require the sender to buy some credits in their account in the Dapp. This way we can use the Token Fee Paymaster implemented in our project to charge the user for the gas. Since the user himself is now paying for gas, this attack is longer feasible.

(iii) Attacker attempts to replay a relayed transaction

An attacker may try to relay the same transaction more than once by sending the same meta transaction to the relay server multiple times. If the gas fee is subsidized by the recepient contract then it will have to pay the gas fee for the same transaction more than once. This attack is mitigated by the RelayHub in the OpenGSN framework. A nonce is included in the transaction. RelayHub also maintains a nonce for each sender. Transactions with bad nonce get reverted by RelayHub to make sure each transaction is relayed once.

C. Mitigations

The paper already mentions about mitigation of individual attacks in the sections above. In this section we aim to highlight and summaries a few characteristics of the OpenGSN framework and other measures that can go a long way in mitigating attacks general and greatly improve the security of the Gas Station Network.

(i) Compulsory stake for Paymaster and Relay Server in the Relay Hub
The OpenGSN framework compels paymaster and relay servers to deposit a stake in to the RelayHub during the registeration process. Doing this protects the network from a variety of Sybil attacks as it is more difficult and expensive now for attackers to create multiple identities and registers relays or paymasters. The security can be further enhanced by implementing captchas which can restrict the use of bots for Sybil attacks. This stake can also be used to penalize and deincentivize the components from misbehaving.

(ii) Mutual Distrust Model
The OpenGSN network is designed in such a way that none of the components in the network have to trust each other. All components inclusing the clients, relay servers and paymaster is connected using the Relay Hub. It takes care a number of things like making sure the client finds a suitable relay server, relay servers do not censor transactions and the Paymaster refunds the relay server for verified transaction. This ensures that the recipient is not exposed to potential security threats and the overall network is less susceptible to Denial Of Service and Financial attacks.

(iii) Nonce for each transaction
This is not something new but still worth mentioning. A nonce is included in the Transaction so that it can be verified. The RelayHub uses this nonce to check that each transaction is only relayed once. This nonce also helps detect denial of service attacks by the relay and if a user is trying to double spend a transaction

IV. Conclusion

The Gas Station network successfully increases the usability of Blockchain Networks like Ethereum and reduces the barrier of entry by allowing users to execute transactions without needing the native tokens. OpenGSN provides an excellent framework to design the infrastructure of such networks allowing users to send gasless transactions by abstracting some of the components. In our project, we provide implementation for recipient contract with different methods and paymaster contract with two different business logic that can be used depending on the business use case. The added benefit of usability does come with its share of security issues. The paper laid out a few vulnerabilities of the Gas Station Network and how it can lead to different attacks. However, upon analysis we find that given the robust characteristics of the framework, these attacks are very difficult to scale in a practical scenario. The impact of these attacks is limited in terms on time and financial assets. The paper also discusses how the project can improved in the future to further minimize the security threats, mainly in terms of Denial of Service and Financial attacks.

References

[1] Kasireddy, P. (2021). Retrieved from Medium: http://www.easygoing.pflog.eu/32_blockchain_P2P/ethereum_blockchain.pdf

[2] OpenGSN. (n.d.). *OpenGSN Documentation*. Retrieved from https://docs.opengsn.org/

[3] paloaltonetworks. (n.d.). *CYBERPEDIA*. Retrieved from https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos

[4] *SoFi Learn*. (2021, February 25). Retrieved from How Safe is Blockchain? Blockchain Security Guide: https://www.sofi.com/learn/content/blockchain-security/

[5] IBM. (n.d.). *What is Blockchain Security?* Retrieved from IBM:https://www.ibm.com/sg-en/topics/blockchain-security

[6] (2020, November). Retrieved from GitHub: https://github.com/opengsn/gsn-protocol/blob/master/gsn-protocol.md