The role of language based security in DeFi : The example of the PolyNetwork Attack

Rikke Toft Grabski, s174288 Nathan Paulet, s202574
Jeff Gyldenbrand, s202790 Gary Fougerolle, s211974

December 2021

1 Abstract

The PolyNetwork Cross-Chain platform experienced a 600 million USD hack due to a vulnerability that would escalate the privileges of the attacker putting him as the owner of the entirety of the funds on the network. Most open-source DeFi projects, including the PolyNetwork, are not using automated tests, on their code, with security in mind. This paper propose a number of measures to increase the security of the PolyNetwork and, in general, other DeFi open-source projects, such as introducing automated tests with program analysis on their code.

Keywords— Blockchain, Cross chain, DeFi, Smart contracts, Crypto currency, Language-based security, Ownership analysis

2 Introduction and contributions

Decentralised finance is hotter than ever with the increased use of cryptocurrencies based on block-chains, like Bitcoin, Etherium and many more. This has resulted in many active, open source projects, that are based on the trading of these currencies. Every once in a while there are news of another successful attack, where millions of dollars have been stolen. One such example is the 600 million USD that were stolen from multiple chains via the cross-chain platform PolyNetwork, by using a function that should never have been available to the attacker.

In this paper, we will encourage the use of static program analysis on the very source code of these projects, by analysing the PolyNetwork to uncover the core problem that enabled the attack in the first place, as well as provide a more specific analysis strategy to be used by the platform.

To give the reader a better understanding of the PolyNetwork hack, we will initially explain the basic principles of the architecture behind PolyNetwork smart contracts and cross chain operations. This is explained in section 3. Next, in section 4, we will provide an overview of the extent of the PolyNetwork hack and an insight into the vulnerability that allowed it. Section 5 will dive into possible preventions of such attacks on cross-chain platforms in the future by enforcing language-based security on cross-chain platforms. In section 6, we provide a detailed analysis of the cause behind the attack and identify the main flaws in the PolyNetwork. Finally, in this section, we will present our proposals for implementations that can increase the security of their platform. This report will ultimo provide an conclusion and further work that can be done.

Section	**Nathan**	**Rikke**	**Jeff**	**Gary**
Abstract	0%	0%	100%	0%
Introduction	0%	50%	50%	0%
Required Information	0%	0%	100%	0%
A 600$ Oversight	100%	0%	0%	0%
Enforcing Language based security	0%	100%	0%	0%
Detailed analysis	100%	0%	0%	0%
Conclusions and further work	50%	50%	0%	0%
This list	100%	0%	0%	0%

3 Required Information

3.1 The Architecture of Poly Network Smart Contracts

Poly Network is a platform integrating with different chains and compatible with multiple cryptocurrency. It lets users swap tokens from a digital ledge to another one. [10] To do so, the users are able to use any method on any contract on one ledge to another one.

Why calling a cross-chain method, the platform uses what they call relayers that take the information from the first chain and transfer it to the second chain, while the Poly Network syncs the block header of participating chains.

The Poly Network uses the EthCrossChainManager method to validate a transaction and execute the method in the transaction. Its relayers utilises the Manager's contract, the verifyHeaderAndExecuteTx on Manager to verify the header and check if it has been executed before. They also use the _executeCrossChainTx method to execute the transaction's method.

Each transaction method is supposed to be cryptographically secure since what it does is that it takes the method and arguments and hash it. However, only the four first bytes are used as the MethodID, so we can have a lot of collisions.
[7]

3.2 Cross Chain explained

The chain interoperability described above consists of two main types of operations in the poly network: read- and write. A source chain is the one that initiates a cross chain operation and a destination chain is the target of the operation.[13]

A write-operation will change the destination chain and a read-operation will only return back a status of the destination chain to the source chain, never change it. The cross chain operations can be assets transfers, such as a monetary assets, it could be assets exchanges, contract calls or something else. [1]

For such an operation a relayer is needed. Relayers help transferring the read- or write operations between two chains.
As observed in Figure 1[11], chain A is listening (1) to a specific block (2) on chain B where the relayer helps transferring the status on the destination chain to the source chain.

Also observed, a user (3) performs a cross chain transaction from chain A to chain B. This is done by contacting a *source manager* on the source side, and providing the destination block chain, M_{dst}, and the write operation, Q_{write}. When this is executed, the source manager will log it, and a relayer will retrieve this log.

Now the relayer transfers a transaction on to the destination chain. The transaction calls the destination manager and includes Q_{write} and an inclusion proof, which validates the existence of the log. Then the destination manager validates the write operation and executes it. The relayer makes a proof after the execution which is then

send to the source chain. The source chain then validates this proof and updates the status of the transaction in the log.

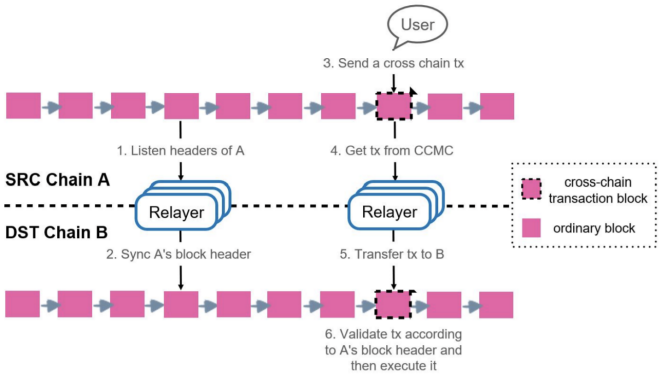

Figure 1: Overview for write operation, PolyNetwork: An Interoperability Protocol for HeterogeneousBlockchains

4 A $600 Million Oversight

The cryptocurrency platform Poly Network was hit by an attack on August 10^{th} 2021. The attacker managed to steal more than $600 millions dollars [3] that he surprisingly returned in the next days.[4]

To make that possible, the attacker abused from one on the Ethereum chain's method called EthCrossChainManager, also called Manager, that allowed him to execute any cross-chain transaction.[7]

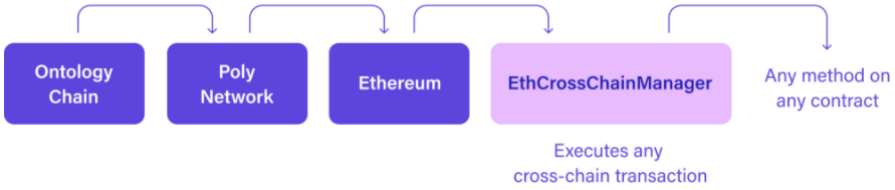

Figure 2: EthCrossChainManager Call, Blog Kraken

This method owns another contract named EthCrossChainData, also called Data, that oversees the entities managing the wallets (they are called the keepers).

The vulnerability was that by executing any transaction with Manager, the attacker could update the list of keepers (entities able to call for big transactions) and then put himself as the only owner of the entirety of funds on the Poly Network (using putCurEpochConPubKeyBytes method) and transfer them to his wallet.

5 Enforcing Language-Based Security in Cross-chain Platforms

Most open source projects are being developed using platforms like GitHub[1] to centrally manage the source code. When someone wishes to contribute to the project it is usually required to create a pull request (or PR) to merge the new or altered code into the code base. This pull request will have to be approved by a number of human reviewers, before it can be merged into the code base.[5]

Human reviews of code can often be lacking from a security perspective, as a reviewer would need an intimate knowledge of the code base, as well as the design, to be able to determine if a given change to the code could potentially compromise the overall security of the solution being developed. And even if the reviewer did have the necessary insight, they would not necessarily have the needed time to ascertain if the PR introduces any vulnerabilities. To help accommodate this, GitHub and others offer the feature of running automated checks and tests, that a PR must succeed, along with the human review, in order to have their PR merged. See, for example Test cases by GitLab[12].

Despite this, there are still many open source cross-chain projects, like PolyNetwork, that either do not use automated checks at all, or only use it for the sake of checking code quality or other non-security related purpose.

Program analysis are techniques used to predict the behaviours or values of systems that could appear dynamically while it is executed, and can be used to validate software, such that we can ensure that a system is secure, by checking that it cannot behave in an unintended manner[8]. The discipline of program analysis has been around for a long time, and so the idea of using it in a software project is not new.

While searching for resources on using program analysis in DeFi projects, we quickly found that most, if not nearly all, resources only discuss using such analysis for smart contracts - specifically only exploring the security aspect from a user's perspective. One such example is Vandal, which is a framework for analysing the security of Etherium smart contracts[2].

Long story short, people are simply not talking about ensuring a high standard of security on the source code of DeFi projects by using program analysis techniques. It may generally be time consuming and expensive to create program analysis for a system, but if it is a safety-critical system like cross-chains in DeFi are, it should be a no-brainer to invest in it, such that $600 million cannot be stolen again due to a vulnerability in the code.

Our idea for cross-chain platforms is to not only run automatic checks for every PR, as many projects already do, but to add tests based on program analysis to add increased language-based security. In the case of PolyNetwork, they could use a lightweight static analysis, inspired by the information flow analysis. In information flow analysis, usually the flow of information of variables is examined - for example to uncover any undesired flow going from a private or protected variable, into a public

[1] `www.github.com`

variable[9].

$$\text{public} \longrightarrow \text{private} \quad \checkmark$$
$$\text{private} \rightarrow \text{public} \quad \times$$

The vulnerability in the PolyNetwork attack was that exposed functions have access to private functions through ownership. In other words, there was a flow going from a private function to a public one, which is unsafe by design. Specifically, we saw that the exposed function EthCrossChainManager had ownership over private functions, like putCurEpochConPubKeyBytes owned by EthCrossChainData, which was one of the main causes of making the $600 million attack possible.
In other words, we had the following ownership flows:

$$\text{putCurEpochConPubKeyBytes} \longrightarrow \text{EthCrossChainData}$$
$$\text{EthCrossChainData} \longrightarrow \text{EthCrossChainManager}$$

Which resulted in the following, undesired ownership flow, through transitive property:

$$\text{putCurEpochConPubKeyBytes} \longrightarrow \text{EthCrossChainManager}$$

It is possible to make this analysis be far more light-weight by limiting the analysis to ownership flows going into the single functions that PolyNetwork use to expose their contracts for public use, although it will naturally be a less thorough analysis.

Reasons for using this light-weight ownership analysis rather than information analysis on every variable and data file, is in part due to it being far faster by only looking at a subset of the functions (whilst still being able to tell if there are any violations in what is exposed). Besides, being a cross-chain platform, PolyNetwork is unlikely to have mapped out all of the information on the chains it is integrated to, and instead communicates using contracts, meaning an information flow analysis on the source code of PolyNetwork could possibly not find all problem-flows, if the information is part of/owned by another chain. Besides, PolyNetwork already has some idea of what functions they wish to be exposed, considering their quick-fix of using a text file of functions that are white-listed for public use. This means it is only a matter of crafting the analysis and feeding it with the preexisting information of which function that are acceptable, exposed functions.

6 Detailed Analysis - the Cause of the PolyNetwork Attack and Countermeasures

Now that we know the general methods we can apply to this category of system, we need to know about the detailed scenario of the attack to understand how to apply these. Here is a diagram summering the exchanges made over the network:

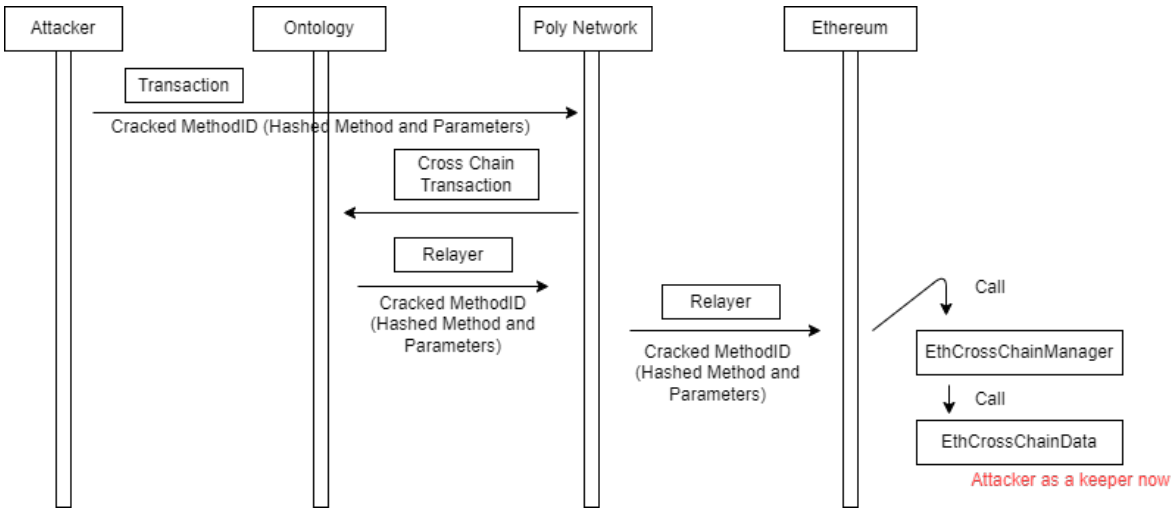

Figure 3: Exchange Diagram other the network

As you can see, the attacker is sending a cross chain transaction between two blockchains (here Ontology and Ethereum). This transaction is composed of a MethodID which represent the first four bytes of the hash result of the called method and its parameters.

It is interpreted by Poly Network which send it between both chains thanks to relayers. Once the transaction arrives at the Ethereum chain, it is interpreted and the EthCrossChainData method is called through the EthCrossChainManager method.

Once this method is called, the attacker is able to add himself to the list of keepers which allow him to transfer big amount of money between wallets.

This is the next part of the attack. He then repeated these steps on multiple chains (Ethereum, Polygon and Binance Smart Chain) to reach the $600 Millions.

Here are the list of assets transferred by the hackers [6]:

Token	Platform	Amount	Transaction Hash
USDC	Ethereum	96,389,444.229984	0x5a8b2152ec7d5538030b53347ac82e263c58fe7455695543055a2356f3ad4998
WBTC	Ethereum	1,032.12483694	0x3f55ff1fa4eb3437afe42f4fea57903e8e663bc3b17cb982f1c8d4c8f03a2083
DAI	Ethereum	673,227.941533113298891801	0xa7c56561bbe9fbd48e2e26306e5bb10d24786504833103d3f023751bbcc8a3d9
UNI	Ethereum	43,023.751365396442021965	0xc917838cc3d1edd871c1800363b4e4a8eaf8da2018e417210407cc53f94cd44e
SHIB	Ethereum	259,737,345,149.519786617235448706	0xe05dcda4f1b779989b0aa2bd3fa262d4e6e13343831cb337c2c5beb2266138f5
renBTC	Ethereum	14.47265047	0xb12681d9e91e69b94960611b227c90af25e5352881907f1deee609b8d5e94d7d
USDT	Ethereum	33,431,197.734821	0x06aca16c483c3e61d5cdf39dc34815c29d6672a77313ec36bf66040c256a7db3
WETH	Ethereum	26,109.060672756730881958	0xc797aa9d4714e00164fcac4975d8f0a231dae6280458d78382bd2ec46ece08e7
FEI	Ethereum	616,082.589988960251715574	0xd8c1f7424593ddba11a0e072b61082bf3d931583cb75f7843fc2a8685d20033a
ETH	Ethereum	26,109.060672756730881958	0x93bacc30f19e46ae40d6a7f38d8a7f8fbc49c979a454dd6d9a4b2577d317636d
ETH	Ethereum	2,857.486346845890372134	0xad7a2c70c958fcd3effbf374d0acf3774a9257577625ae4c838e24b0de17602a
USDC	Binance Smart Chain	87,603,373.774864499503468781	0xd59223a8cd2406cfd0563b16e06482b9a3efecfd896d590a3dba1042697de11a
USDC	Binance Smart Chain	298.940563273249676643	0xea37b3208431f75a8a849fdf13cd357cb64761a848d48a516c3cac5bbd6caaad5
ETH	Binance Smart Chain	26,629.159998706545651647	0x4e57f59395aca4847c4d001db4a980b92aab7676bc0e2d57ee39e83502527d6c
BTCB	Binance Smart Chain	1,023.880948564689526459	0x50105b6d07b4d738cd11b4b8ae16943bed09c7ce724dc8b171c74155dd496c25
BUSD	Binance Smart Chain	32,107,854.114341286723103272	0xd65025a2dd953f529815bd3c669ada635c6001b3cc50e042f9477c7db077b4c9
BNB	Binance Smart Chain	6,613.440488906866981869	0x534966864bda354628d4f1c66db45cbefcdda7433e9576e7664fea01bb05be9a
USDC	Polygon	85,089,610.911661	0x1d260d040f67eb2f3e474418bf85cc50b70101ca2473109fa1bf1e54525a3e01
USDC	Polygon	108.694578	0xfbe66beaadf82cc51a8739f387415da1f638d0654a28a1532c6333feb2857790

Figure 4: List of assets

But how did he got access to this method ?

Since the MethodID is only the first four bytes of the hash gotten from the method and arguments, he simply brute forced it until having a hash with the same first four bytes as the EthCrossChainManager method.

He didn't even have to get the right method name and the right parameters, just to have random characters for both the methods and the parameters leading, through the hash function, to a hash with the same first four bytes. He was then able to call this method without any other restriction.[7]

After the analysis of this attack, we can identify the main flaws in the Poly Network platform that lead to the attack. As we saw earlier, the first problem was the fact that anybody could call any method from a chain (here EthCrossChainManager and EthCrossChainData) without having any verification on whether it should be available to the user to call.

What is normally done for this kind of feature is to use a public/private key pair linked to each user (especially in the context of DeFi and blockchains). This way any user is authenticated and has its own rights. However, since users may not always have a public/private key pair, another solution would be to use an Access Control Policy which could be role based, attribute based, identity based.

The second problem here is a problem of user's inputs. Indeed the user does not need to know the exact name and parameters of these methods to call them. Only providing a similar hash to the one expected allows the method call. As we saw earlier, putting random characters for both the method name and its parameters will work as long as their first four bytes of hash will match.

What should be done here is to sanitise the user input and be sure that the methods called as well as the parameters given are permissible for the user. We could use a whitelist which consists of a list of allowed methods that can be called by the users. We could also use blacklisting but in this case to forbid some characters that could be entered by the user and result in unexpected issues (here unexpected hashes). However, simply checking a text-file in the source code, to check if it is allowed for a user to use it, also does not guarantee that said functions do not use any private functions, and so the white-list fix is insufficient.

The way the methods are accessed is not appropriate as well since using a hash for this is not giving that much of a security level, especially when you use only the first four bytes of it which can be cracked really easily and fast nowadays.

What could be done here is to increase the key space by saving more than the first 4 bytes of the hash and maybe by artificially increasing it more than it should by adding a padding for example to reach the key space we use in today's cryptographic standards.

As a result, what Poly Network could implement the following to increase the security level of their platform and avoid attacks of the same kind in the future is:

- an Access Control policy and mechanism
- limit the methods' accessibility and accesses to avoid flows from high security level to low security level
- an sanitation of the user's inputs (with a whitelist of names and a blacklist of forbidden characters)
- a greater key space for the MethodID (using a bigger part of the result of the hash and using padding if possible)

By doing so, the user will only be able to call known method names (the ones in the whitelist or within the range of access his methods have), with the right parameters, only if he has the right permissions for this action (checked by the access control policy and mechanism).

If a user tries to crack the MethodID the way it was done this will result in:

- first a way longer research to crack it since the key space is bigger (multiplied by two for each bit added to the key space)
- even if the hash is finally cracked, it will not be accepted by the system since the method name and parameters are not in the whitelist or not within its range of access.
- even if these are the right method name and parameters, he won't have the permissions to call it (ensured by the access control policy and mechanism)

As a conclusion, the Poly Network platform's security against this attack would be efficiently increased.

7 Conclusions and further work

From an analysis of the PolyNetwork attack which happened in August 2021, we have proposed a number of measures for the PolyNetwork to take, in order to increase their security and preventing an attack like it from happening again. To summarise, these steps are:

- Implement an Access Control policy and mechanism
- Use automated ownership analysis
- Sanitise user inputs
- Increase the MethodID's key space

We have further motivated the general use of automated analysis of source code in order to increase the language-based security of cross-chain projects, which is often overlooked. Specifically in the case of PolyNetwork and the recent attack, we suggest analysing the flow of ownership of functions, to make sure exposed functions do not have ownership over private functions, as seen in the list above.

Further work could be to formally define the ownership analysis, look into more types analysis that would be essential for DeFi Blockchain projects, as well as implementing them.

References

[1] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. *SoK: Lending Pools in Decentralized Finance*. First Online: 17 September 2021. URL: https://link.springer.com/chapter/10.1007/978-3-662-63958-0_40.

[2] Lexi Brent et al. *Vandal: A Scalable Security Analysis Framework for Smart Contracts*. https://arxiv.org/pdf/1809.03981.pdf. Sept. 2018.

[3] Ryan Browne. *Crypto platform hit by $600 million heist asks hacker to become its chief security advisor*. https://www.cnbc.com/2021/08/17/poly-network-cryptocurrency-hack-latest.html/. Last accessed 2. Nov. 2021. 2021.

[4] Ryan Browne. *Hacker behind $600 million crypto heist returns final slice of stolen funds*. https://www.cnbc.com/2021/08/23/poly-network-hacker-returns-remaining-cryptocurrency.html. Last accessed 2. Nov. 2021. 2021.

[5] *Collaborating With Pull Requests*. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests. Last accessed 18. Dec. 2021.

[6] Community. *Poly Network Exploit*. Last accessed 15. Nov. 2021. URL: https://en.wikipedia.org/wiki/Poly_Network_Exploit.

[7] KrakenFX. *Kraken Blog*. https://blog.kraken.com/post/11078/abusing-smart-contracts-to-steal-600-million-how-the-poly-network-hack-actually-happened/. Last accessed 4. Nov. 2021. Sept. 2021.

[8] Flemming Nielson. *Lecture 1 of Program Analysis 02242*. Technical University of Denmark, DTU Compute. Sept. 2020.

[9] Flemming Nielson and Hanne Riis Nielson. *Language-Based Security. In: Formal Methods*. Springer, Cham., 2019.

[10] *Poly Network Website*. https://poly.networkl. Last accessed 4. Nov. 2021. 2021.

[11] Poly Team. *PolyNetwork: An Interoperability Protocol for Heterogeneous Blockchains*. Last accessed 7. Nov. 2021. 2020. URL: https://www.poly.network/PolyNetwork-whitepaper.pdf.

[12] *Test Cases (Ultimate)*. https://docs.gitlab.com/ee/ci/test_cases/index.html. Last accessed 18. Dec. 2021.

[13] Sam M. Werner et al. *SoK: Decentralized Finance (DeFi)*. Submitted on 21 Jan 2021, last revised 26 Sep 2021. URL: https://arxiv.org/abs/2101.08778.