Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Exploit Detection in the Evolution of DApps

Master's Thesis


Jean Marc Müller

muelleje@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich


**Supervisors:**
Benjamin Estermann, Prof. Dr. Ye Wang
Prof. Dr. Roger Wattenhofer

August 30, 2024

# Acknowledgements

The author would like to thank Benjamin Estermann and Ye Wang for the very pleasant cooperation during the research of this thesis, their insight in interpreting results, their support in solving problems and the careful proofreading of this report.

# Abstract

The rapid proliferation of Decentralized Finance (DeFi) on the Ethereum blockchain has introduced significant vulnerabilities in smart contracts, leading to an increase in exploit transactions that result in substantial financial losses, totalling 3.24 billion USD from DeFi related incidents between April 2018 and April 2022. This study investigates the detection of such exploit transactions on vulnerable smart contracts. It aims to reproduce, validate and extend the DEFIER approach, initially proposed by Su et al. In this process we gain fundamental insight into the evolution of Decentralized Applications (DApps), but also encounter substantial obstacles. The dataset from Su et al. covers the years 2016 to 2018. We find that the more recent exploits in the data by Zhou et al. from 2018 to 2022 are conducted via multiple intermediary smart contracts; highlighting the fact that the structure of DApps has grown more complex, and attacks have adapted to exploit vulnerabilities not only in a DApp's main smart contract, but also those called internally. Furthermore, the amount of data DEFIER collects for analysis of an incident on average has grown 20-fold, emphasizing the vast increase in transaction volume and consequently adoption of blockchain technology. Significant challenges arise in the context of transaction clustering, and the DEFIER model appears inadequate in adapting to this new setting. Eventually, it raises the question, if the concepts on which DEFIER is based are still valid today.

# Contents

Chapter 1

# Introduction

This study is an investigation into the detection of exploit transactions on vulnerable smart contracts of DeFi Dapps. In the research of existing work, to extend and build upon, the 2021 paper by Su et al. called 'Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications' [1] is discovered. It presents an approach for automated attack detection, termed DEFIER, which is base on the concept that such attacks may vary in their specific implementation, but generally undergo a four-stage lifecycle of preparation, exploitation, propagation and completion. The paper reports promising results, however, it is based on an older dataset of transactions between 2016 and 2018. Consequently, before attempting to refine or improve this model, it must be validated on a modern dataset. For this purpose we use the dataset from the 2023 paper 'SoK: Decentralized Finance (DeFi) Attacks' by Zhou et al. [2] containing DeFi exploit incidents between 2018 and 2022.

Chapter 2

# Background

## 2.1 Ethereum and Smart Contracts

The Ethereum blockchain, a decentralized open-source peer-to-peer network, is the second largest blockchain after Bitcoin by market capitalization [3]. While Bitcoin was primarily designed as a digital currency, Ethereum fundamentally differentiates itself from that by functioning as both a distributed computing system and an operating system with scripting capabilities. For this purpose it supports two types of accounts; Externally Owned Accounts (EOAs), which are controlled by users or external servers through private keys, and Contract Accounts, which are governed by smart contracts. These are essentially programs than run and reside on the Ethereum blockchain and facilitate automated agreements without intermediaries. [4, 1].

Ethereum tracks the state of every account and records changes resulting from transactions. A transaction is a signed data package that transfers value or data between accounts. There are three types of transactions, Ether transfers, contract calls, and contract creations. Contract creation transactions deploy smart contracts by including bytecode in the transaction, which contains the creation code, runtime code, and swarm code. These contracts are then executed by the Ethereum Virtual Machine (EVM), during which a contract can communicate with EOAs and other contracts. Each transaction produces a receipt that records the transaction's outcome, including involved account addresses and execution status. These are written onto Ethereum's cryptographically secured ledger, with every node on the network maintaining a copy [4, 1].

More detailed information than just the transaction's outcome can be obtained through tracing APIs that generate EVM traces, also known as transaction execution traces. By re-executing transactions using the historical state of the blockchain, they provide a record of the sequence of operations made during the transaction, such as external contract calls or internal function calls [5, 1].

## 2.2 Decentralized Applications and Vulnerabilities

Decentralized applications (DApps) are public applications that run on peer-to-peer networks such as Ethereum. They use smart contracts as their backend, which encode the core logic of the application and manage critical data, while users interact with the frontend through decentralized or centralized interfaces. Unlike traditional applications hosted on centralized servers, the fact that DApps operate on the blockchain makes them transparent and resistant to censorship. Having their code publicly viewable means that anyone can ensure the DApp will execute the way it claims. However, it also gives rise to the opportunity to find vulnerabilities, flaws in the code, which can be exploited, leading to significant financial losses [4, 1].

This poses a problem in particular for DApps in the Decentralized Finance (DeFi) space, where the peak total value locked (TVL) across all blockchains has surpassed 253 billion USD in December 2021, with 145 billion USD (57% TVL) on the Ethereum blockchain. These protocols for exchanges, lending, cross-chain bridging, etc. have grown rapidly since the beginning of 2020. Unfortunately, the complexity and interconnectivity of these systems have expanded the attack surface, contributing to a total loss of 3.24 billion USD from DeFi related incidents between April 2018 and April 2022 [2].

Consequently it would be extremely beneficial for all honest actors as well as the projects maintaining DApps, to have a reliable way of detecting such adversarial transactions which exploit vulnerabilities in smart contracts. In the best case this detection would happen while the transaction is still in the Ethereum memory pool, waiting for execution, so it can be prevented. The next best scenario would be to detect an exploit immediately after execution in order to invoke an emergency halt of the DApp to stop the exploit from being repeated. A very promising approach for automated attack detection is found in the 2021 paper 'Evil Under the Sun: Understanding and Discovering Attacks on Ethereum Decentralized Applications' by Su et al.[1] (henceforth referred to as 'Evil Under the Sun'). Their approach is termed 'DEFIER' and is derived from a detailed study on the structure of such attacks. The detection is based on similarities in transaction execution traces rather than simple transaction parameters such as 'Ether sent', 'Transaction Fee', etc. The next two Sections 2.3 and 2.4 give a detailed explanation on the approach.

## 2.3 The Four-Stage Attack Lifecycle

In 'Evil Under the Sun' the authors manually reconstruct and analyse 42 DApp attack incidents on 25 victim Dapps from technical blogs, news posts and annual security reports from blockchain security companies. They find that all these attacks, although varying in specific implementation, follow the same general

pattern, which they term the 'four-stage attack lifecycle'. This pattern consists of four sequential stages, namely the attack preparation, exploitation, attack propagation and mission completion. Figure 2.1 visualizes this sequence [1].

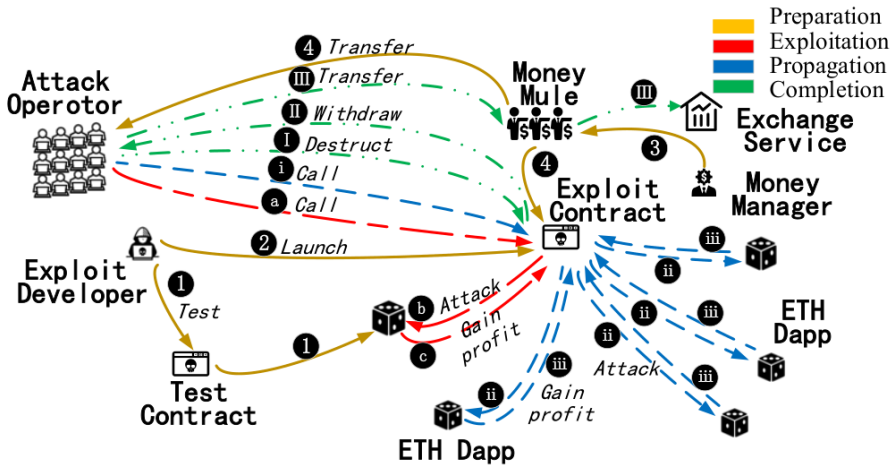


Figure 2.1: Visualization of a four-stage attack lifecycle: *1-4* depict the attack preparation stage, *a-c* depict the exploitation stage, *i-iii* depict the attack propagation stage and *I-III* depict the mission completion stage [1].

### Attack Preparation Stage

Before actually launching an attack on the vulnerable smart contract of a DApp via an exploit contract (*2*), exploit developers begin by testing and refining their exploit code over multiple transaction (*1*). In further preparation, money managers transfer the funds necessary to execute the attack, such as gas fees and other possible costs, into the exploit contract (*3*). In an effort to conceal their EOAs these funds are channelled through money mules (*4*) [1].

### Exploitation Stage

In the main exploitation stage the exploit contract is called by one or multiple attack operators from different EOAs (*a*). The exploit contract then executes the attack on the vulnerable smart contract of the victim DApp (*b*) and gains the corresponding profit (*c*) [1].

### Attack Propagation Stage

In the propagation stage the attacker aims to exploit further DApps that exhibit the same or a similar vulnerability in their smart contract. To this end they either

reuse or adapt the original exploit contract through updates. Then they proceed in the same fashion as in the exploitation stage, where the exploit contract is called by attack operators (*i*) to execute the attack on the other DApp or DApps (*ii*) and gain further profit (*iii*) [1].

#### Mission Completion Stage

Finally, in the mission completion stage 'selfdestruct' is called on the exploit contract by one of the attack operators (*I*) and consequently the entire attack profit stored in the contract are withdrawn to that EOA (*II*). From there the profit is transferred to an exchange server in order to be withdrawn (*III*). Once again it is channelled through multiple money mules in an effort to obscure the origin of the funds [1].

## 2.4 DEFIER

Based on the insights of the four-stage attack lifecycle described in the previous Section 2.3 Su et al. create a methodology termed DEFIER to identify new attacks, including zero-day attacks on new victim DApps. It is based on the concept that even though specific operations may vary, for attacks targeting different vulnerabilities, the sequential procedure of the lifecycle including high-level behaviour per attack stage remains the same. DEFIER aims to learn these patterns from transactions and their execution traces to identify new exploits and determine their attack stage. It consists of two components, Preprocessing and Sequence-based Classification [1].

### 2.4.1 Preprocessing

Preprocessing identifies transactions directly or indirectly related to a DApp and clusters them based on their execution traces and invocation times [1].

#### Data Collection

The data collection starts with scraping all DApp transactions, that means all transactions interacting with the target DApp directly, or indirectly via an intermediary smart contract; more precisely, all normal and internal transactions that have the target DApp's vulnerable smart contract in their 'to' field [1]. We refer to these transactions as level 0, visualized in Figure 2.2.

However, the goal is to discover all EOAs' operations and intents towards the target DApp, and specifically the intent to reuse an attack vector on a different DApp, i.e. propagation stage, cannot be found by only considering transactions

that interact with the target DApp. Therefore, in a second step all semantically-similar transactions are scraped. First, starting from the previously collected DApp transactions, all unique EOAs that have initiated a normal transaction on level 0 are gathered. Next, all unique smart contracts that have initiated an internal transaction on level 0 are considered, and all unique EOAs that have interacted with these smart contracts are added to the set. Specifically, that means all unique EOAs that have initiated a normal transaction with the corresponding smart contract in the 'to' field. We refer to these transactions and EOAs as level 1. The gathering of this EOA set is visualized in Figure 2.2. For each EOA in this set all of its so far scraped transactions on level 0 and level 1 are considered, and every transaction from that same EOA that lies within $\pm$ 1 day of one of the already scraped transactions is scraped preliminary, but only kept if it does not exceed a similarity threshold of 3. In particular, given a scraped transaction $tx_{\text{scraped}}$ and another transaction $tx_{\text{prelim}}$ from the same EOA that lies within $\pm$ 1 day of $tx_{\text{scraped}}$, then $tx_{\text{prelim}}$ is only kept if the distance between transaction graphs (TG) is less than 3 [1].

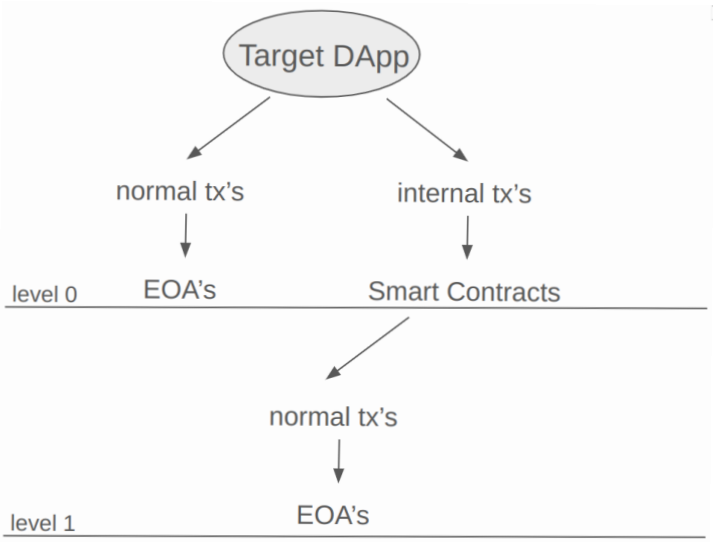


Figure 2.2: Visualization of the gathering of the set of unique EOAs. First all normal and internal transactions that have the Target DApp's vulnerable smart contract in their 'to' field are scraped. These constitute transactions on *level 0*, also referred to as the DApp transactions. All unique EOAs that have initiated a normal level 0 transaction are added to the set. Then for each smart contract, that has initiated an internal level 0 transaction, all normal transactions are scraped. These constitute transactions on *level 1*. All unique EOAs that have initiated a normal level 1 transaction are added to the set.

The TG distance $D(g_1, g_2)$ between two transaction graphs $g_1$ and $g_2$ is a measure of structural similarity and closeness of execution time. It is based on graph

representations of the transactions' execution traces and is defined as follows:

$$D(g_1, g_2) = \alpha \min_{[o_1,o_k]\in O(g_1,g_2)} \sum_{i=1}^{k} c(o_i) + \beta\Delta t \tag{2.1}$$

where $O(g_1, g_2)$ represents the set of necessary graph edits, i.e. insertions, deletions or substitutions of vertices or edges, to transform $g_1$ into $g_2$, and $c(o_i)$ gives the cost for edit $o_i$. $\Delta t$ is the time difference of execution in units of hours and $\alpha$ and $\beta$ are weights. This implementation sets the values as $\alpha = 0.9$, $\beta = 0.1$ and $c() = 1$ [1].

#### Clustering

As described in Section 2.3 the operational intent during one of the stages of an attack lifecycle is usually reflected in a sequence of multiple transactions from different EOAs. To capture this, all scraped normal transactions are clustered according to the similarity of their execution trace graphs as well as their temporal proximity. Accordingly, this constitutes a between-graph clustering problem which is solved via a $k$-means algorithm based on the TG distance as defined in Equation 2.1. This results in multiple clusters with some representing one of the stages of the attack lifecycle, and many others that are unrelated to any sort of attack [1].

### 2.4.2 Sequence-based Classification

Sequence-based Classification reconstructs potential attack footprints by modelling transaction sequences as vectors and classifies them to determine whether they are part of an attack, and if so, identify the stage of the exploit [1].

#### Sequence Representation

The basis for this classification is the sequence representation which is created in the following way. First, all transactions within a cluster are ordered by their timestamp to turn the transaction cluster into a transaction sequence. Next, the transaction sequence is transformed into a vector sequence, where each element is a graph embedding of the execution trace graph of the corresponding transaction. For every element, in order to highlight the EOA's intent and filter out irrelevant transaction operations an attention mechanism is applied which assigns weights to different parts of the transaction as well as an embedding representation of the EOA and the DApp, and combines them into a new vector. The weights are learned by a long short term memory (LSTM) model. Finally, all combined vectors are fed into another bidirectional LSTM neural network, which is trained

on a labeled dataset with the classifier, to capture relationships and dependencies across time and output a final single vector to represent the entire sequence [1].

#### Sequence Classification

The output vectors from the bidirectional LSTM model, representing entire transaction sequences, are classified by a multilayer percetpron (MLP) classifier that predicts the probability that a sequence corresponds to a certain attack stage. The entire classification module is trained together on a labeled dataset through stochastic gradient descent [1].

### 2.4.3 New Attack Discoveries

Su et al. collect the smart contract addresses corresponding to 104 potential target DApps and inspect them with their DEFIER model. The preprocessing results in 342'224 transaction clusters, 100'081 of which are labeled with an attack stage and belong to 85 victim DApps. Randomly selected 4% of these clusters are checked manually, and they find that 3'671 of those 4'003 clusters are indeed related to attack incidents, including 75 zero-day victim DApps [1].

## 2.5 A Modern Dataset

As described in the previous Section 2.4.3 the results of the DEFIER model on unseen data make the approach appear very promising and a good starting point for further investigation. The main caveat in this context is that the unseen data as well as the training data, from which also the insights of the four-stage attack lifecycle stem, only include transactions covering the years 2016 to 2018 [1].

In the context of the very rapid progress in the entire blockchain space, it can be argued that this data is quite out of date in the year 2024. Ethereum's first production-ready version, called Homestead, launched in March 2016 and DeFi was still a new and emerging concept in 2018 [6].

Consequently, the first step in any investigation in this direction is to validate the effectiveness of the DEFIER model on a more modern dataset. For this purpose we consider the attack dataset on DeFi DApps from the 2023 paper 'SoK: Decentralized Finance (DeFi) Attacks' by Zhou et al. [2] (henceforth referred to as 'SoK'). This includes data on exploit incidents from April 2018 to April 2022 [2].

The 'Evil Under the Sun' paper provides a google drive link[1] at which the annotated dataset as well as an implementation of DEFIER is supposed to be available.

[1] https://drive.google.com/drive/folders/1cdD1gHNbWIS228QXmeUReougSL_k1kvf?usp=sharing

However, the link only leads to an empty folder. Also, the first and second author of the paper are contacted by email, unfortunately however, without response. Therefore, an implementation of DEFIER must be reverse engineered from its description in the paper.

Chapter 3

# Data Collection

This chapter focuses on the data collection process, which is the first step in reproducing the DEFIER approach. First, the technical method is explained, detailing how individual task are achieved and also highlighting differences to the method explained in the 'Evil Under the Sun' paper. Afterwards, some unexpected findings are presented and discussed.

## 3.1 Technical Method

### 3.1.1 Dataset and Incident Sorting

In total, the 'SoK' dataset includes 200 attack incidents on DeFi DApps spanning a period from April 2018 to April 2022 [2]. However, this dataset actually includes incidents on the Ethereum blockchain as well as the Binance Smart Chain (BSC). After excluding the 96 BSC related incidents, that leaves 104 incidents on Ethereum.

Since these incident are collected from differnet sources, such as academic papers, audit reports and real-world incidents [2], they exhibit a variety of incident causes and sometimes are missing crucial information for further processing. Therefore, 14 incidents are excluded, because they mostly consist of a description, but do not state the actual address of the DApp's vulnerable smart contract. Additionally, since the focus of this study are attack incidents that result from vulnerabilities in smart contracts, further 28 incidents are excluded as they are resulting from other causes, such as compromised private keys or operator backdoors.

Lastly, there are 6 incidents that each involve more than one vulnerable smart contract. This constitutes an additional layer of complexity which the original DEFIER approach is not designed to handle. Therefore, they are also excluded and left as a problem to address, once a working reproduction of the approach is achieved, and a baseline on the new dataset is found. This leaves 56 incidents from the Ethereum blockchain under investigation.

The dataset is structured in the form of individual .json files, one for each incident.

For every incident the block number and the address of the DApp's vulnerable smart contract are extracted, as well as the known adversarial EOAs (one or multiple) and known adversarial transactions, i.e. attack transactions.

### 3.1.2 DApp Transaction Scraping

This section describes the scraping of the DApp transactions as defined in Section 2.4.1, namely, the normal transactions, i.e. direct interactions with the vulnerable smart contract, and the internal transactions, i.e. interactions via an intermediary smart contract. These constitute transactions on level 0, as depicted in Figure 2.2.

#### Normal Transaction Scraping

Normal transactions by definition originate from EOAs ('from' field). They are scraped via the Etherscan API [7], namely the 'txlist' action with a specified Ethereum address, that returns a list of the normal transactions in which the specified address is involved. When this is called on a contract address, such as a DApp's vulnerable smart contract, it consequently returns only transactions where that contract address appears in the 'to' field. Therefore, they constitute all direct interactions between an EOA and the vulnerable smart contract. This is the same API as the one used in 'Evil Under the Sun' [1].

The only problem arising in this context is the fact that a single API call can only return a maximum of 10'000 transactions. Since there are contract addresses with more than 10'000 associated normal transactions, the first approach to a solution is to simply keep making API calls on these contract addresses with an increasing value for the 'offset' parameter until all transactions are fetched. However, it is discovered that there are many occasions, especially when scraping normal transactions in the context of semantically-similar transactions, as described in the following section, where the contract address in question has many hundred thousand or even millions of associated normal transactions.

Therefore, to keep the data load from exceeding manageable levels, we make by default two API calls for every contract address. The first one sets the block number associated with the incident as the 'startblock' parameter and 'sort' to ascending, thereby fetching up to 10'000 transactions (if available) including and after the incident block. The second call sets the block number as 'endblock' and 'sort' to descending, thereby fetching up to 10'000 transactions (if available) including and before the incident block.

#### Internal Transaction Scraping

Internal transactions by definition originate from contract accounts ('from' field). In the 'Evil Under the Sun' paper these are also scraped via the Etherscan API [7, 1], now with the 'txlistinternal' action, again with a specified Ethereum address. This action returns a list of the internal transactions in which the specified address is involved. When this is called on a contract address, such as the DApps vulnerable smart contract, it consequently returns both, transactions where that contract address appears in the 'to' field as well as those where it appears in the 'from' field. The latter are disregarded, and the remaining transactions constitute all interactions, where a smart contract functions as an intermediary for the interaction of an EOA with the vulnerable smart contract.

We tried to use the Etherscan API, but encountered the following problem. It only fetches high-level internal transactions, however, more complex internal transactions that occur as part of multi-step contract interactions are neglected. These more complex internal transactions are those that are only shown on the Etherscan website for a contract address under the 'Internal Transactions' tab when 'Advanced Mode' is activated [8].

The effective problem arises for certain known adversarial transactions that are located on level 1, as depicted in Figure 2.2, meaning they are found by scraping normal transactions for smart contracts that are previously scraped on level 0. Some of these known adversarial transactions are only found from those complex internal transactions. Therefore, it is essential that they are included in the scraping of internal transactions on level 0.

We solved this problem by using the Blockscout API [9] instead, namely the 'txlistinternal' action, which essentially functions in the same way as the Etherscan API, but includes the complex internal transactions in its response.

The Blockscout API also only returns a maximum of 10'000 transactions per API call. Consequently, it is treated the same way as the Etherscan API and by default there are two API calls made per contract address, where the first fetches up to 10'000 transactions after and the second one up to 10'000 transactions before the block number of the incident.

### 3.1.3 Semantically-Similar Transaction Scraping

This section describes the scarping of semantically-similar transactions as defined in Section 2.4.1. First the transactions through which EOAs have indirectly interacted with the DApp, via an intermediary smart contract, are scraped. These constitute transactions on level 1, as depicted in Figure 2.2. Then, for all EOAs that have initiated transactions on level 0 or level 1, additional transactions with similar transaction execution traces or executed in close temporal proximity are gathered.

### EOAs of Indirect Interaction

To scrape the transactions of EOAs that have indirectly interacted with the DApp, the smart contracts that have initiated the internal transactions on level 0 are considered. For each of these smart contracts their normal transactions are scraped as described in the previous Section 3.1.2.

Specifically, for each smart contract that has interacted with the target DApp via an internal transaction, this fetches all normal transactions, originating from an EOA, that have interacted with that smart contract. Consequently, these are all transactions initiated by an EOA which used that smart contract as an intermediary to interact with the target DApp, as well as all other transactions where an EOA interacted with that smart contract for any other reason. These transaction constitute the normal transaction on level 1.

### Additional Transactions

The additional transactions scraped at this stage essentially provide the basis for detecting the propagation stage of the attack lifecycle, as defined in Section 2.3. To find transactions that are potentially part of the propagation stage, the idea is to consider all EOAs that have initiated normal transactions on level 0 or level 1, i.e. all EOAs that might be involved in the original attack.

To this end, all normal transactions on level 0 and level 1 are considered. For each transaction the 'from' EOA and block number is extracted, and all normal transactions from the same EOA within the block number $\pm$ 6640 are scraped. This corresponds to an interval of $\pm$ 1 day from the original transaction as specified in the 'Evil Under the Sun' paper [1]. However, only those transactions are kept, that have a TG distance, as defined in Equation 2.1, to the original transaction of less or equal to 3, again as specified in the paper [1].

In order to calculate the TG distances between the original transaction and every single additional transaction, the transaction execution traces for all these transactions have to be scraped first. This is done via the Infura API [10], namely the 'trace_transaction' method, which returns for an individual transaction a list of the traces in the order called by the transaction.

For every transaction the list of its traces is then stored as a directed graph in a NetworkX object [11]. Next, these graph representations of transactions are pair-wise fed to the 'GraphEditDistance.compare()' function of the GMatch4py library, with all edit costs set to 1, in order to calculate the graph edit distance between the pair of graphs [12]. Afterwards, the TG distance is calculated as the weighted sum of the graph edit distance and invocation time difference according to Equation 2.1.

These proceedings follow the outline in the 'Evil Under the Sun' paper, includ-

ing the used libraries [1]. The only major difference is the API used for the transaction trace scraping. The paper uses the Bloxy API [13] for that purpose. However, since no API key for the Bloxy API could be obtained, the Infura API is used instead.

The calculation of this large number of TG distances is a very time consuming process, mainly due to the fact that a correspondingly large number of transaction execution traces needs to be scrape. Unfortunately, rather than being able to scrape in bulk like for the normal and internal transactions, this can only be done for each transaction individually, which in turn necessitates a large number of API calls.

## 3.2 Findings

### 3.2.1 A Recursive Process

During the data collection process we made an unexpected discovery, specifically in the part before scraping the additional transactions, as described in the previous Section 3.1.3. In the 'Evil Under the Sun' paper the adversarial transactions, i.e. attack transactions, as initiated by an EOA are reported to be found either on level 0, i.e. direct interaction with the target DApp, or on level 1, i.e. interacting with the target DApp via an intermediary smart contract, as described in Section 2.3 in the main exploitation stage [1].

As mentioned previously in Section 3.1.1 our incidents under investigation from the 'SoK' dataset include corresponding known adversarial transactions. These adversarial transactions are tracked during the data collection process and at each level, when scraping the level's normal transactions, we check whether the adversarial transactions are among them. What we find is that the majority of incidents have their adversarial transactions actually located at level 2 with some even beyond that (more details follow in the next Section 3.2.2). This means that there is more than one intermediary smart contract between the attacking EOA and the targeted vulnerable smart contract.

On an operational level, this does not necessarily complicate the data collection, however, it does make the process recursive. Specifically, if the adversarial transaction is found on level 0 or level 1 as in 'Evil Under the Sun', this means there are two API calls to scrape the normal transactions from the target DApp and two API calls to scrape the internal transactions from the target DApp, as described in Section 3.1.2. Afterwards, we loop over all unique smart contracts that have initiated internal transactions on level 0, with two API calls for each to scrape their normal transactions. This gives us all normal transactions on level 0 and level 1. On the other hand if the adversarial transaction is located on level 2, that means not only do we need to loop over the unique smart contracts again

with two API calls each to also scrape their internal transactions as well, but for every one of those smart contracts we receive another corresponding set of unique smart contracts from their internal transactions. We then need to loop over every smart contract in all of these sets to find their normal transactions. This process is visualized in Figure 3.1.

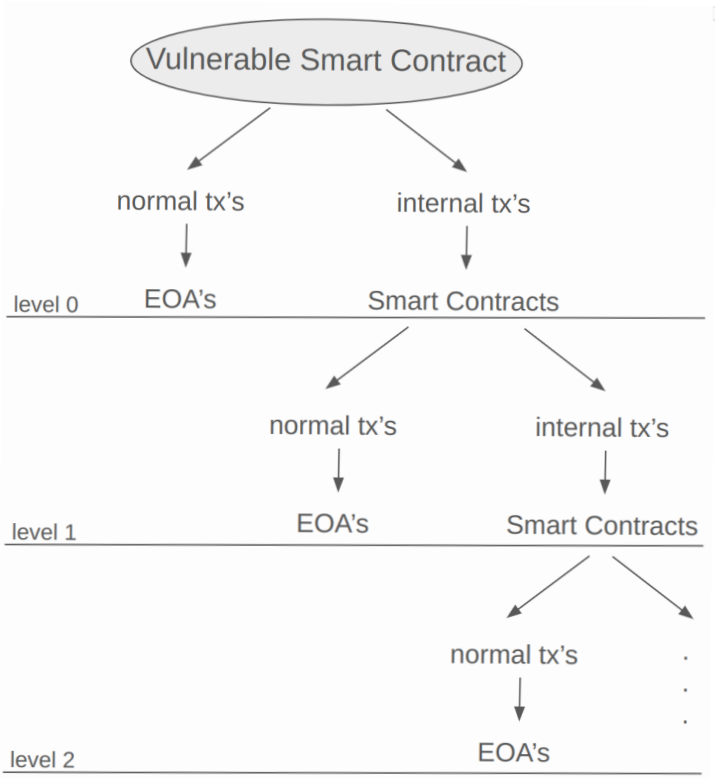


Figure 3.1: Visualization of the recursive process of data collection for incidents with adversarial transactions located on level 2 or higher. First all normal and internal transactions of the target DApp's vulnerable smart contract are scraped. These transactions constitute *level 0*, also referred to as the DApp transactions. Then, we loop over all unique smart contracts that have initiated internal transactions on level 0 and scrape all their normal and internal transactions. These transactions constitute *level 1*. After that, the process becomes recursive, as for every smart contract found on level 0 we receive another set of internal transactions on level 1 initiated by a corresponding set of unique smart contracts. We then need to loop over every smart contract in all of these sets to scrape their normal and internal transactions, which will then constitute *level 2*. This will consequently leads to an exponential growth with increasing level, both in the number of necessary API calls as well as the total number of scraped transactions.

If we assume an average number of associated normal and internal transactions for a smart contract, this process constitutes an exponential growth with increasing levels, both in the number of necessary API calls as well as the sheer number of scraped transactions. Consequently, for incidents with adversarial transactions located on level 2 or higher, the data collection process becomes extremely time consuming. First, because the number of API calls grows exponentially with every level, but after the adversarial transactions are found, there is still the next step of scraping the additional transactions. As described in the previous Section 3.1.3, in that step we need to loop over every normal transaction across all levels, where that number of normal transactions has also grown exponentially with every level. Additionally, in that loop we receive another preliminary set of normal transactions for every transaction in the loop, and then need to loop over every preliminary transaction in all of these sets to individually scrape all corresponding transaction execution traces in order to calculate the TG distances and decide whether a preliminary transaction is kept as an additional transaction or disregarded.

Furthermore, on a qualitative level, this discovery also gives interesting insight into the evolution of DApps, specifically in regards to their structure and complexity. While all victim DApps in the incidents from the 'Evil Under the Sun' dataset exhibit a vulnerability in their main smart contract, which is the smart contract that other accounts directly interact with, when interacting with the DApp. These findings on the more recent 'SoK' dataset show that the structure of interacting smart contracts that constitute a DApp has since grown more complex; to such a degree, that the majority of incidents in the new dataset actually exploit vulnerabilities in smart contracts that are not the main interaction point of the DApp, but are called by the DApp internally.

### 3.2.2 Statistics on Collected Data

Tables 3.1 and 3.2 give an overview of the number of normal transactions for each exploit incident, i.e. those used for clustering in Section 4, that are collected up to and including level 2. It also includes the incident block number, number of adversarial EOAs and adversarial transactions, as well as the level on which these are located. The two sections on the right represent the reduced numbers of transactions, as described in Section 4.1.5. Essentially, it is a systematic reduction that only keeps the transactions that are executed within $\pm$ 4 weeks and $\pm$ 2 weeks of the incident block. Furthermore, the numbers stated in the table represent the number of scraped transactions at the stage of data collection described in Section 3.1.3, before they are extended with the additional transactions. For incidents with their adversarial transactions found on level 0 or level 1, there is no need for further scraping on higher levels. This is represented with an 'x' in their 'level 2' column. Also, the incidents used for clustering in Section 4 are marked in magenta in the 'Vulnerable Smart Contract' column. In the following

we highlight a few noteworthy points.

First, in regards to the adversarial transactions. As mentioned before, in the 'Evil Under the Sun' dataset all adversarial transactions are located on level 0 or level 1 [1]. In contrast to that we find that in the 'SoK' dataset only a single incident has its adversarial transactions on level 0, with 16 further incident that have theirs on level 1, leaving the majority of incidents, i.e. 39, with their adversarial transactions on level 2 or higher. For ease of readability the incidents in the tables are colour-coded according to the level of their adversarial transactions. Level 0 and level 1, i.e. the status quo form 'Evil Under the Sun' are green, level 2, where most incidents from 'SoK' have their adversarial transactions located, are orange, and the incidents with even higher levels are red. This shift discovered in the modern dataset means that for 69.6% of incidents the vulnerable smart contract that is exploited, is not the main smart contract of the DApp with which other accounts interact, but it is one that the DApp calls internally. This, as well as the fact that only one attack executes an exploit directly without an intermediary smart contract, shows that the complexity of DApps has greatly evolved between the period of the 'Evil Under the Sun' dataset, i.e. 2016 to 2018, and to the period of the 'SoK' dataset, i.e. 2018 to 2022.

Another point to highlight is the vastly larger number of transactions that are collected. The averages over both tables, i.e. all investigated incidents, is found in Table 3.2. We find the sum of the average number of scraped transactions on level 0 and level 1 for the 'SoK' dataset gives 122'300.4. Unfortunately, we cannot make a direct comparison to the 'Evil Under the Sun' dataset, as they do not quote this number. Nevertheless, in the context of running DEFIER on unseen data, they state that for 104 DApps, their preprocessing results in 2'350'779 transactions [1], which translates to an average of 22'603.6 transactions per incident. Comparing these two averages, we already find an increase by a factor of 5.41. However, we need to take into consideration that their number already includes the extension by the additional transactions, as described in Section 3.1.3, while table 3.2 represents the number of transactions before that extension. From the incidents that we consider for clustering in Section 4, we observe that this extension on average constitutes a factor of 3.74. Consequently, taking that into account we find that the number of transactions, that need to be scraped in order to execute the DEFIER model, has increased on average by 20-fold. For one thing, this makes the entire DEFIER approach computationally more expensive as well as more time consuming during the data collection stage. The total size of the collected data is 102 GB. Apart from that though, it also shows that the amount of traffic, i.e. the amount of executed transactions on the Ethereum blockchain, has massively increased since the period of 2016 to 2018.

Lastly, we want to mention a discovery on the 'SoK' dataset related to the incident with its vulnerable smart contract with address at `0x876b...1dc8`, marked in blue in Table 3.2. For this incident normal transactions are scraped up to

and including level 4, but its adversarial transactions are not found among them, therefore their level is quoted as '$\geqslant 5$'. However, what appears strange is that in the two columns on the right of the table, where the reduced number of transactions are shown, it says there are none. This indicates that among the collected transactions for the incident there are none that were executed within $\pm$ 4 weeks of the incident block. This does not make any sense, because if there are no transactions involving the vulnerable smart contract executed at and around the incident block, then there was no incident. Therefore, we suspect there is a mistake in the data concerning that incident.

[1]Unfortunately, the file containing the level 2 transactions of this incident is missing at the time of submission. It is being re-scraped and will be provided with the rest of the data and the python code.

| Block No | Vulnerable Smart Contract | Adversarial EOAs | Adversarial tx's | Adversarial level | Number of tx's at level 0 | Number of tx's at level 1 | Number of tx's at level 2 | (block ± 4 weeks) Number of tx's at level 0 | (block ± 4 weeks) Number of tx's at level 1 | (block ± 4 weeks) Number of tx's at level 2 | (block ± 2 weeks) Number of tx's at level 0 | (block ± 2 weeks) Number of tx's at level 1 | (block ± 2 weeks) Number of tx's at level 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10898307 | 0x74bc...1fd6 | 7 | 17 | ⩾ 3 | 5'947 | 109 | 42'429 | 5'929 | 104 | 15'376 | 5'803 | 103 | 7'734 |
| 13417949 | 0x5bd6...dee4 | 1 | 2 | ⩾ 3 | 1 | 7'920 | 268'412 | 0 | 287 | 6'895 | 0 | 221 | 790 |
| 13786402 | 0x14e6...e4bd | 1 | 28 | 2 | 937 | 3'672 | 45'217 | 446 | 690 | 4'058 | 155 | 430 | 4'051 |
| 11242635 | 0x1cec...2004 | 1 | 17 | 2 | 1 | 3'024 | 774 | 1 | 398 | 270 | 0 | 169 | 143 |
| 10355807 | 0x0e51...3682 | 1 | 1 | 1 | 3'008 | 545'959 | x | 2'828 | 288'626 | x | 1'161 | 262'442 | x |
| 12514105 | 0x7b3b...c6ca | 1 | 1 | ⩾ 3 | 18 | 3'155 | 367'548 | 18 | 2'553 | 32'054 | 18 | 2'180 | 8'884 |
| 12865334 | 0x35c6...7810 | 1 | 2 | 2 | 10'000 | 57 | 20'071 | 10'000 | 43 | 56 | 0 | 21 | 56 |
| 13537922 | 0x2914...e807 | 1 | 2 | 1 | 131 | 3'093 | x | 68 | 608 | x | 38 | 341 | x |
| 12849787 | 0xa800...190d | 1 | 5 | 1 | 0 | 12'904 | x | 0 | 8'236 | x | 0 | 4'671 | x |
| 11012466 | 0x606e...8962 | 1 | 1 | 2 | 685 | 135'786 | 3'015'875 | 0 | 49'913 | 820'439 | 0 | 45'171 | 560'687 |
| 13221565 | 0xacbd...e747 | 1 | 1 | 1 | 2'433 | 2'293 | x | 34 | 173 | x | 10 | 33 | x |
| 11303123 | 0x6847...5210 | 1 | 1 | 2 | 383 | 54'730 | 768'594 | 265 | 7'069 | 78'860 | 55 | 3'919 | 32'370 |
| 11846605 | 0x33bf...8a27 | 1 | 9 | 1 | 2 | 4'068 | 69'178 | 2 | 2'161 | 31'015 | 2 | 1'939 | 25'636 |
| 11473330 | 0xae46...c8cf | 1 | 1 | 2 | 283 | 32 | 195 | 283 | 32 | 193 | 280 | 32 | 192 |
| 13695970 | 0x5377...ba2c | 1 | 2 | 2 | 253 | 10'287 | 416'980 | 253 | 726 | 69'271 | 243 | 708 | 25'388 |
| 10592428 | 0x951d...bfe2 | 1 | 3 | 2 | 55 | 10'517 | 21'843 | 55 | 979 | 1'529 | 55 | 545 | 874 |
| 11817852 | 0x3ec4...8af0 | 1 | 6 | 2 | 152 | 1'558 | 125'837 | 136 | 648 | 84'344 | 121 | 332 | 81'931 |
| 11541219 | 0xe0b9...aed5 | 2 | 13 | ⩾ 3 | 10'000 | 1'613 | 10'030 | 7'761 | 483 | 1 | 2'805 | 256 | 0 |
| 13848983 | 0xc9f2...14ef | 1 | 2 | ⩾ 5 | 1'732 | 439 | 11'397 | 277 | 77 | 405 | 145 | 41 | 309 |
| 13118320 | 0x55db...3830 | 1 | 1 | 2 | 2 | 1'210 | 63'018 | 0 | 322 | 209 | 0 | 69 | 103 |
| 14602790 | 0xc1e0...24c5 | 1 | 1 | ⩾ 3 | 20'000 | 39'281 | 2'099'534 | 10'028 | 20'240 | 541'093 | 10'028 | 20'163 | 451'919 |
| 10790337 | 0x8809...4d61 | 1 | 1 | 1 | 2'203 | 280'189 | x | 2'074 | 187'165 | x | 2'019 | 167'516 | x |
| 14028474 | 0x6b7a...1522 | 4 | 99 | 2 | 20'000 | 65'720 | 3'118'643 | 20'000 | 3'763 | 754'089 | 20'000 | 2'328 | 556'885 |
| 9899736 | 0x0eee...f5ea | 1 | 46 | 1 | 9'410 | 2'303 | x | 3'147 | 390 | x | 1'556 | 279 | x |
| 11205648 | 0x833e...743d | 1 | 1 | 1 | 7 | 809 | x | 7 | 481 | x | 4 | 115 | x |
| 13715026 | 0x66e7...ee63 | 1 | 1 | ⩾ 4 | 1 | 1'229 | 40'486 | 1 | 710 | 40'168 | 1 | 589 | 40'129 |
| 13591123 | 0x4e3f...9d2b | 1 | 21 | ⩾ 3 | 20'000 | 747'447 | 6'286'307 | 14'959 | 168'595 | 670'091 | 6'679 | 158'373 | 516'507 |
| 11940504 | 0x17e8...ca5f | 1 | 9 | 1 | 12'234 | 469'703 | x | 7'099 | 13'657 | x | 3'271 | 7'172 | x |
| 10723290 | 0x0e6f...4a30 | 1 | 2 | 2 | 6'629 | 2'801 | 151'580 | 6'629 | 1'089 | 18'385 | 6'628 | 903 | 15'986 |

Table 3.1: Statistics on the collected normal transactions per exploit incident up to and including level 2 - Part I. Including number of adversarial EOAs, adversarial transactions and level on which the adversarial transactions are located.

| Block No | Vulnerable Smart Contract | Adversarial EOAs | Adversarial tx's | Adversarial level | Number of tx's at level 0 | Number of tx's at level 1 | Number of tx's at level 2 | (block ± 4 weeks) Number of tx's at level 0 | (block ± 4 weeks) Number of tx's at level 1 | (block ± 4 weeks) Number of tx's at level 2 | (block ± 2 weeks) Number of tx's at level 0 | (block ± 2 weeks) Number of tx's at level 1 | (block ± 2 weeks) Number of tx's at level 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12171155 | 0xe7f4...58e6 | 5 | 98 | 2 | 3'404 | 10'642 | 10'246 | 3'358 | 9'658 | 13 | 3'330 | 9'440 | 0 |
| 14684307 | 0x88cc...8b17 | 2 | 3 | 2 | 1 | 802 | 123'874 | 0 | 57 | 3'614 | 0 | 35 | 211 |
| 12688468 | 0xa231...7ef5 | 1 | 3 | 2 | 4 | 3'112 | 572'728 | 3 | 62 | 15'264 | 3 | 53 | 613 |
| 14182802 | 0x876b...1dc8 | 1 | 119 | $\geqslant 5$ | 27 | 20'038 | 4'215 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11129474 | 0xf035...edbe | 1 | 30 | 1 | 6'976 | 82'510 | x | 1'631 | 32'377 | x | 1'351 | 27'096 | x |
| 11720050 | 0xe11f...df50 | 1 | 6 | 0 | 655 | 33'072 | x | 145 | 12'116 | x | 133 | 5'838 | x |
| 14684686 | 0xfbd8...054c | 1 | 7 | 1 | 36 | 20'248 | x | 9 | 14'652 | x | 6 | 9'736 | x |
| 14006014 | 0x88e6...5640 | 1 | 4 | $\geqslant 2$ | 91 | 752'715 | 3'531'486 | 7 | 137'949 | 374'515 | 7 | 122'962 | 287'505 |
| 12995895 | 0x929c...49d6 | 1 | 2 | 2 | 2 | 3'571 | 25'828 | 2 | 2'358 | 4'335 | 2 | 1'944 | 2'920 |
| 9893295 | 0x3212...8923 | 1 | 1 | $\geqslant 3$ | 10'000 | 199'713 | 62 | 1'455 | 1'029 | 0 | 813 | 186 | 0 |
| 14469082 | 0x4622...21b8 | 1 | 1 | 1 | 25 | 3'031 | x | 1 | 221 | x | 0 | 145 | x |
| 14175830 | 0x3157...e758 | 1 | 97 | $\geqslant 5$ | 14 | 1'660 | 1'328'802 | 3 | 17 | 7'858 | 3 | 16 | 3'449 |
| 11817390 | 0x0efb...1c0b | 1 | 1 | 2 | 281 | 151 | 40'016 | 280 | 76 | 2'884 | 280 | 75 | 1'320 |
| 14506358 | 0x39b1...db15 | 1 | 2 | 1 | 0 | 411 | x | 0 | 294 | x | 0 | 285 | x |
| 11903783 | 0x9dae...f2f9 | 1 | 3 | 2 | 64 | 230 | 482'003 | 64 | 76 | 108'028 | 36 | 45 | 87'867 |
| 11256673 | 0xddd7...1101 | 1 | 1 | 2 | 1 | 2'216 | 109'524 | 1 | 859 | 77'800 | 1 | 584 | 60'785 |
| 10368560 | 0x7557...1d8b | 1 | 1 | 2 | 73 | 10'779 | 82'520 | 72 | 10'738 | 13'391 | 71 | 10'189 | 1'848 |
| 12394006 | 0x67b6...9c7a | 1 | 185 | 1 | 12'639 | 254'841 | x | 3'923 | 206'020 | x | 2'475 | 185'450 | x |
| 9484688 | 0x77f9...81bc | 1 | 1 | 2 | 10'782 | 254'534 | 1'982'873 | 7'357 | 124'955 | 579'939 | 5'067 | 111'825 | 405'173 |
| 11792184 | 0xacd4...f952 | 1 | 11 | 1 | 12'437 | 378'205 | x | 1'750 | 113'268 | x | 867 | 98'138 | x |
| 14465357 | 0xe952...1659 | 1 | 4 | 2 | 346 | 10'656 | 227'122 | 41 | 2 | 3'505 | 28 | 2 | 2'476 |
| 13229001 | 0xdac1...1ec7 | 1 | 1 | $\geqslant 2$ | 19'974 | 1'928'643 | ?[1] | 19'974 | 316'021 | ?[1] | 19'974 | 272'100 | ?[1] |
| 11345251 | 0x6684...a64f | 1 | 34 | 1 | 1'697 | 20'145 | x | 1'039 | 13'421 | x | 562 | 7'484 | x |
| 10852722 | 0xde74...9b25 | 1 | 9 | 2 | 1 | 4'322 | 151'990 | 1 | 1'894 | 25'339 | 1 | 1'471 | 19'947 |
| 9504627 | 0x818e...b755 | 1 | 1 | $\geqslant 3$ | 20'000 | 180'857 | 3'853'382 | 20'000 | 27'729 | 521'092 | 20'000 | 24'486 | 377'494 |
| 14420687 | 0x5a9f...6ed1 | 1 | 1 | 2 | 2'015 | 30'065 | 488'506 | 625 | 893 | 43'145 | 336 | 38 | 24'639 |
| 11272255 | 0x328d...5424 | 1 | 1 | $\geqslant 3$ | 1 | 1'692 | 1'186 | 1 | 919 | 154 | 1 | 644 | 63 |
| Total Average | | 1.3 | 16.7 | | 4'072.4 | 118'228.0 | 788'429.2 | 2'750.8 | 31'926.4 | 130'254.7 | 2'079.0 | 28'059.4 | 94'918.0 |

Table 3.2: Statistics on the collected normal transactions per exploit incident up to and including level 2 - Part II. Including number of adversarial EOAs, adversarial transactions and level on which the adversarial transactions are located. This is a continuation of Table 3.1 and gives the total averages over both tables.

Chapter 4

# Clustering

This chapter focuses on the transaction clustering process which is the next step in reproducing the DEFIER model. First, the technical method is explained, particularly how a fundamental problem, which is not actually in the 'Evil Under the Sun' paper, is approached and the resulting consequences. This is followed by some implications from the results in the previous Section 3.2.2. Afterwards, the clustering results for four of the exploit incidents are presented.

## 4.1 Technical Method

### 4.1.1 k-means Clustering

As discussed in Section 2.4.1 the operational intent of one or multiple EOAs generally consists of a sequence of transactions. In an effort to capture such an intent the normal transactions across all levels relevant to an exploit incident are clustered according to their similarity in execution trace as well as invocation time. This between-graph clustering problem is solved via a k-means algorithm [1]. The implementation is as follows.

Since the clustering shall be based on TG distances as defined in Equation 2.1, we first need to create a distance matrix that holds the pair-wise TG distances between all involved transactions. This is done in a similar fashion as in Section 3.1.3 in the context of the additional transactions. First, the execution traces are converted into directed graph objects from the NetworkX library [11]. Then, the pair-wise graph edit distances are computed via the 'GraphEditDistance.compare()' function from the GMatch4py library [12], with all graph edit costs set to zero. This way we create a graph edit distance matrix. Next, we calculate the pair-wise time differences in the unit of hours from the transaction's timestamps that are given in Unix time, which results in a time difference matrix. Finally, we calculate the weighted sum of these two matrices to get a combined TG distance matrix for all pair-wise combinations.

This combined distance matrix is then fed to the scikit-learn k-means algorithm

[14], which groups transactions into clusters by minimizing the within-cluster variance. The only input variable that is missing at this point is k, the number of clusters. Unfortunately, the 'Evil Under the Sun' paper does not specify how they determine the number of clusters for a particular incident. This presents a fundamental problem, because the k-means algorithm critically depends on a predetermined number of clusters. Since no obvious analytical solution presents itself, we decide to approach the problem empirically via Silhouette Analysis and the Elbow Method.

These methods are illustrated in the following two Sections 4.1.2 and 4.1.3 on the 'InverseFinance' incident with corresponding target contract address `0x39b1df026010b5aea781f90542ee19e900f2db15`. This incident has 2 known adversarial transactions that are located on level 1. It has 0 normal transactions on level 0 and 411 on level 1. After scraping the additional transactions, as described in Section 3.1.3, this results in 976 transactions to be clustered.

### 4.1.2 Silhouette Analysis

Silhouette Analysis [15] can be used to determine a suitable number of clusters by providing a metric for how well data points fit within their assigned clusters. It calculates a silhouette score for each point, which depends on the mean intra-cluster distance, i.e. a measure of cohesion, as well as the mean nearest-cluster distance, i.e. a measure of separation. The silhouette score ranges from -1 to +1, where a score close to +1 indicates that the point is well clustered, i.e. close to its own cluster and far from others, while a score near 0 suggests it is on the boundary between clusters, and a negative score indicates it is more similar to a different cluster. The overall silhouette score for a clustering is the average over all data points, with higher scores indicating better defined clusters.

This is implemented by performing k-means clustering over a wide range of cluster numbers, then calculating the silhouette score for each clustering via the 'silhouette_score' function from the scikit-learn library [15], and finally plotting silhouette score against cluster number for evaluation.

Figure 4.1 shows a bar plot of the silhouette scores for the 'InverseFinance' incident for a range of 2 to 200 clusters. We find a maximum score of 0.89 for 16 cluster, which is on the lower end of the spectrum and surrounded by a plateau of high values. There is also another local maximum of 0.85 for 153 clusters. This number might be more promising considering its neighbourhood. It constitutes the peak of a steady rise in score from around 75 clusters up to 150, and is followed by a steep falloff afterwards. In any case, both numbers should be considered.

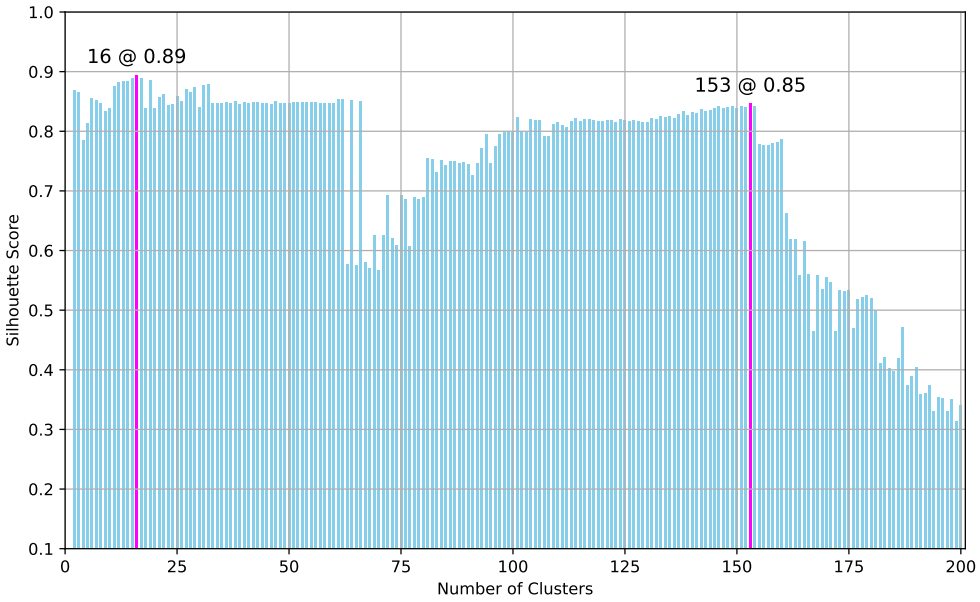


Figure 4.1: Silhouette Scores for 'InverseFinance' incident with corresponding target contract address `0x39b1df026010b5aea781f90542ee19e900f2db15` in a range from 2 to 200 clusters. The global maximum is 0.89 for 16 clusters with another local maximum at 0.85 for 153 clusters.

### 4.1.3 Elbow Method

In order to not rely on a single empirical metric for determining the number of cluster, additionally to the Silhouette Analysis the Elbow Method is considered.

The Elbow Method [16] is a technique used to determine the optimal number of clusters by balancing enough clusters to capture the structure and avoiding too many which might lead to overfitting. It is based on the inertia, i.e. within-cluster sum of squared errors, which is an intrinsic measure of the k-means algorithm on how well data points are clustered. As the number of clusters increases inertia decreases, because the data points are more finely clustered. This rate of decrease typically levels off at a sharp point, which is termed the 'elbow'. That trade-off point constitutes the optimal number of clusters.

This is implemented similarly to the Silhouette Analysis, by performing k-means clustering over a wide range of cluster numbers. The inertia for each clustering is extracted from the 'KMeans' object directly via the 'inertia_' attribute. Then we plot inertia against cluster number to find the sharp decrease in the rate of change of inertia.

Figure 4.2 shows an elbow plot for the 'InverseFinance' incident, again for a range of 2 to 200 clusters. The global and local silhouette score maxima discovered in

the previous Section 4.1.2 are marked by vertical magenta dotted lines. It is clearly noticeable that at a cluster number of 153, the inertia shows a sharp change in its rate of change. At cluster number 16 this change is not quite as prominent, but the slope is steeper before than after cluster 16 as well. There are no other significant changes observed in the inertia's rate of decrease in this plot. Consequently, both numbers should be considered, and clustering result for both are presented in Section 4.2.1

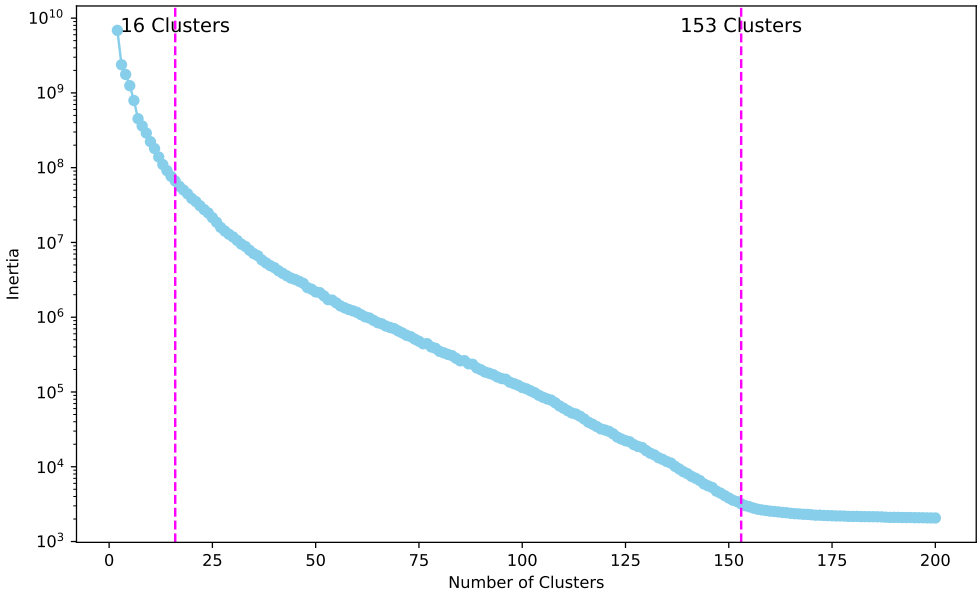


Figure 4.2: Elbow plot for 'InverseFinance' incident with corresponding target contract address `0x39b1df026010b5aea781f90542ee19e900f2db15` in a range from 2 to 200 clusters. The local and global maxima discovered in the silhouette plot at 153 and 16 clusters respectively are marked by dotted magenta lines. Both mark a point corresponding to a change in the slope of the inertia, although this change is more prominent at 153 clusters than it is at 16.

### 4.1.4 Evaluation Metric

The empirical approach to determine the optimal number of clusters via Silhouette Analysis and the Elbow Method, as describe in the previous two Sections 4.1.2 and 4.1.3, raises an important consequence. Specifically, the need for a metric to evaluate whether the chosen number of clusters result in a clustering that in fact groups transactions of similar intent together and is therefore capable of capturing an overall operational intent.

The best case scenario for this purpose would be a labelled dataset, where every transaction is assigned to one of the four stages of the attack lifecycle for a

specific attack, such as the one used in 'Evil Under the Sun'. Unfortunately, we do not have such a dataset at our disposal. Therefore, we approach this problem with the only tangible metric available to us. Specifically, we consider for our clustering testing, incidents with more than one corresponding known adversarial transaction. In that case, we can check, after the clustering is performed, whether the adversarial transactions are in fact grouped in the same cluster.

This should theoretically be the case, as the adversarial transactions from the 'SoK' dataset constitute exploit transactions of the corresponding incident. Consequently, they should be part of that incidents main exploitation stage, at least as long as they originate from the same EOA. For some incidents, however, we have a larger number of known adversarial transactions from different EOAs. Therefore, it is also possible, although not necessarily the case, that they comprise the exploitation stages of more than one attack on the victim DApp, exploiting the same vulnerability. In that case, they may also be grouped in multiple clusters, e.g. two clusters representing the exploitation stages of two different attacks. For this reason, we only consider incidents with more than one known adversarial transaction, but originating from a single EOA.

We also compare numerical results to some values from the 'Evil Under the Sun' paper. Unfortunately, they do not quote average values across their dataset. However, they do mention that for one clustering example, they have 11'088 transactions resulting in 142 clusters, with an average TG distance of 0.2 and time difference of 1.5 h. This gives an average of 78.1 transactions per cluster. This incident is from their labelled dataset (henceforth referred to as the 'example incident'). As previously mentioned they also run DEFIER on unseen data, where they cluster 2'350'779 transactions into 342'224 clusters, giving 6.9 transactions per cluster (henceforth referred to as the 'unseen data'). These values show a large spread, nevertheless we reference them for comparison [1].

### 4.1.5 Statistical Implications for Data Collection

As is described in Section 3.2.2 and presented in Tables 3.1 and 3.2 all of the incidents, except for one, have their adversarial transactions located at least on level 1, with the majority on level 2 or higher. Since all normal transactions up to and including the level of the known adversarial transactions must be included in the clustering, the large numbers of collected transactions corresponding to an incident present a problem in regards to the required time for processing. Especially, considering that the numbers stated in the table are those collected at the different levels, but before scraping the additional transactions, as described in Section 3.1.3. Therefore, we choose to do the clustering testing on two incidents with a comparatively low number of transactions, and for two more incidents we systematically reduce the number of transactions by only including those that are executed within $\pm$ 2 weeks of the incident block.

To give some reference in this context, the incident with the most transactions for which we do clustering testing is 'CreamFinance', with its adversarial transactions located on level 1. For level 0 and level 1 combined we have collected 4'070 transactions in total, after reduction to include only transactions within ± 2 weeks of the incident block, we are left with 1'941 transactions. For these we need to scrape the additional transactions, i.e. all transactions within ± 1 day of each of the existing transactions, originating from the same EOA, which preliminarily extends our set to 64'553 transactions. Next, we need to scrape all of their transaction execution traces, which cannot be done in bulk, but we must make individual API calls for each transaction. From those traces we need to calculate the TG distances in order to decide which of these preliminary transaction are kept, i.e. those with a TG distances smaller or equal to 3. Eventually, this results in 19'593 transactions to be clustered, where in order to determine a suitable number of clusters, we need to run the clustering for a wide range of cluster numbers to calculate silhouette scores and inertia values. The corresponding distance matrix for this incident on which the k-means clustering is based has a size of 9 GB when save in plain text as a .csv file.

## 4.2 Results

In this section we present the results of the k-means clustering testing for four of the exploit incidents from the 'SoK' dataset. For each incident, we start with a short introduction on number of clustered transactions. This is followed by the interpretation of its Silhouette Analysis and Elbow Method plot to determine the suitable number of clusters. Finally, we present the result of the clustering.

### 4.2.1 InverseFinance

This exploit incident is on the InverseFinance DApp with a vulnerable smart contract with the address at `0x39b1df026010b5aea781f90542ee19e900f2db15` and 2 known adversarial transactions, located at level 1. During data collection we find 0 normal transactions at level 0 and 411 at level 1, giving us a total of 411 transactions. These are preliminarily extended to 2'354 transactions, and after disregarding those with TG distances greater than 3, it leaves 976 transactions for clustering.

#### Silhouette Analysis and Elbow Method

The Silhouette Analysis as well as the Elbow Method for this incident is described are Sections 4.1.2 and 4.1.3 in the context of the technical method for the clustering section. They yield 16 and 153 clusters as the best suited choices.

### k-means Clustering

We preform k-means clustering for both cases of 16 and 153 clusters and the results are displayed in Figures 4.3 and A.1 (appendix) respectively.

The results for 16 clusters show a mean number of transactions of $61.0 \pm 115.4$ which is close to the mean of 78.1 from the 'example incident', however, the standard deviation is very high, putting more than half the clusters beyond 100 transactions. The mean TG distance is $13.84 \pm 9.45$ which is far greater than the 0.2 mentioned in the 'example incident', and the mean time difference shows a similar divergence at $113.08 \pm 112.99$ h compared to 1.5 h. In particular is the standard deviation of the time difference almost as large as the mean time difference itself. Finally, the 2 known adversarial transactions are clustered into different clusters, namely cluster 0 and cluster 12. All of this indicates, that the transactions are not clustered well in regards to our purpose of grouping similar transactions of close temporal proximity in order to capture their common intent.

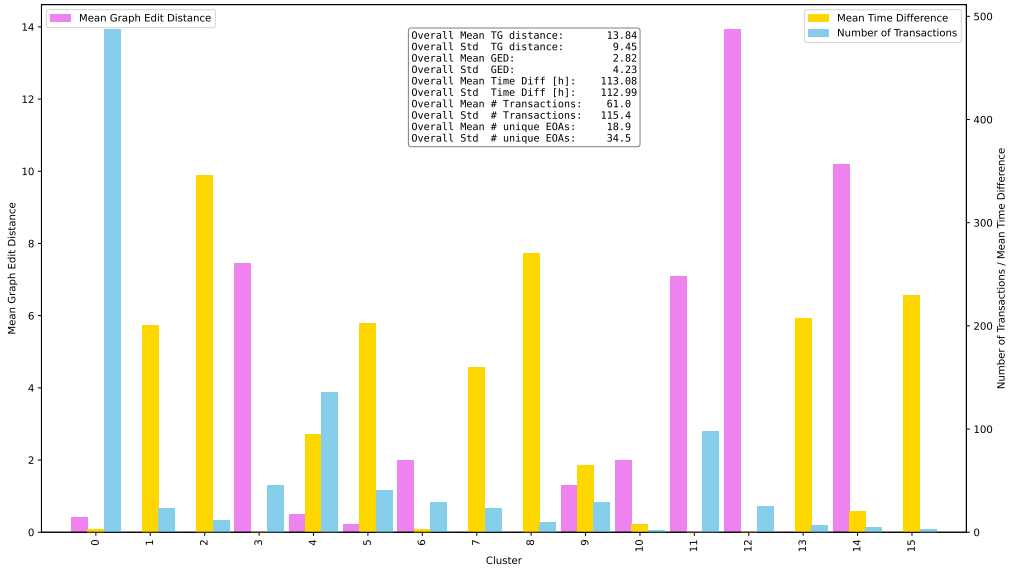


Figure 4.3: k-means clustering with 16 clusters for the 'InverseFinance' incident. The 2 known adversarial transactions are clustered into different clusters, namely cluster 0 and cluster 12.

The results for 153 clusters show a mean number of transactions of $6.4 \pm 15.4$ which is close to the mean on the 'unseen data' of 6.9 as quoted in 'Evil Under the Sun'. However the standard deviation actually exceeds the mean pointing to a very large dispersion. Effectively, it results in the majority of clusters having less than 4 transactions and some with more than 60. In the values for TG distance and time difference the standard deviation also exceeds the mean at $0.30 \pm 0.66$ and $0.10 \pm 0.34$ h. In this clustering the 2 known adversarial transactions are

also clustered into different clusters, namely cluster 25 and 37. Again, all of this indicates that the transactions are not well clustered.

### 4.2.2 SushiSwap

This exploit incident is on the SushiSwap DApp with a vulnerable smart contract with the address at `0xe11fc0b43ab98eb91e9836129d1ee7c3bc95df50` and 6 known adversarial transactions, located at level 0. During data collection we find 655 normal transactions at level 0. These are preliminarily extended to 12'829 transactions, and after disregarding those with TG distances greater than 3, it leaves 671 transactions for clustering.

#### Silhouette Analysis

The results for the Silhouette Analysis are plotted in Figure 4.4 for a range of 2 to 200 clusters. We find a maximum at 0.80 for 2 clusters. This is disregarded, because conceptually it does not make sense to cluster 671 transactions into only 2 clusters. Furthermore, there is a local maximum at 0.63 for 12 clusters, which is preceded by a sharp rise and followed by a steady decline in Silhouette scores. This is considered going forward.

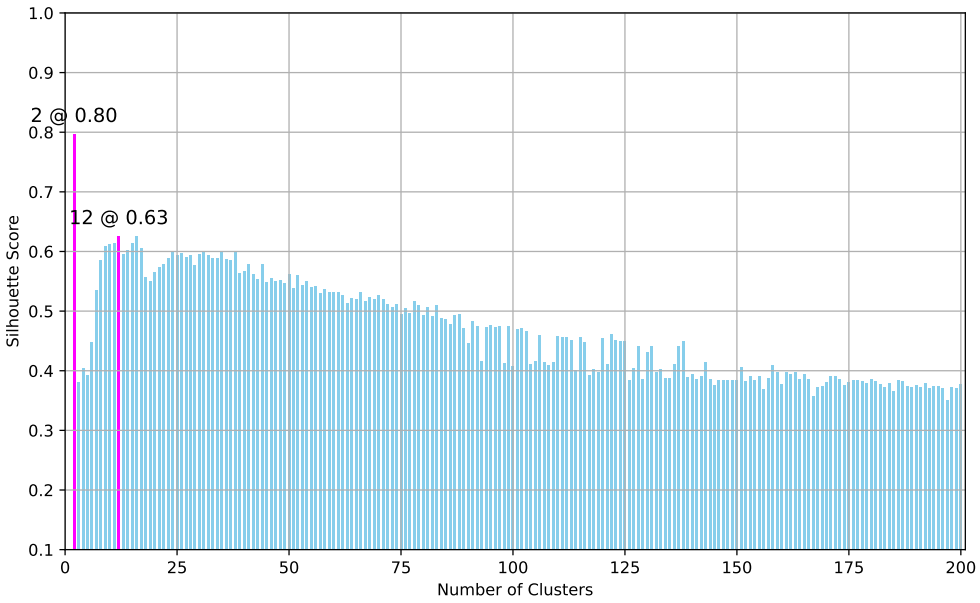


Figure 4.4: Silhouette Scores for 'SushiSwap' incident with corresponding target contract address `0xe11fc0b43ab98eb91e9836129d1ee7c3bc95df50` in a range from 2 to 200 clusters. The global maximum is 0.80 for 2 clusters with another local maximum at 0.63 for 12 clusters.

### Elbow Method

Figure 4.5 shows the Elbow plot for a range of 2 to 200 clusters and the local maximum found by the Silhouette Analysis is marked by a dotted magenta line. The neighbourhood around 12 clusters, however, shows a steady slope, i.e. no sharp decline in the rate of change of inertia. The plot does exhibit such a sharp change though, located at 48 clusters. In order to explore all possibilities, both values are considered for clustering.

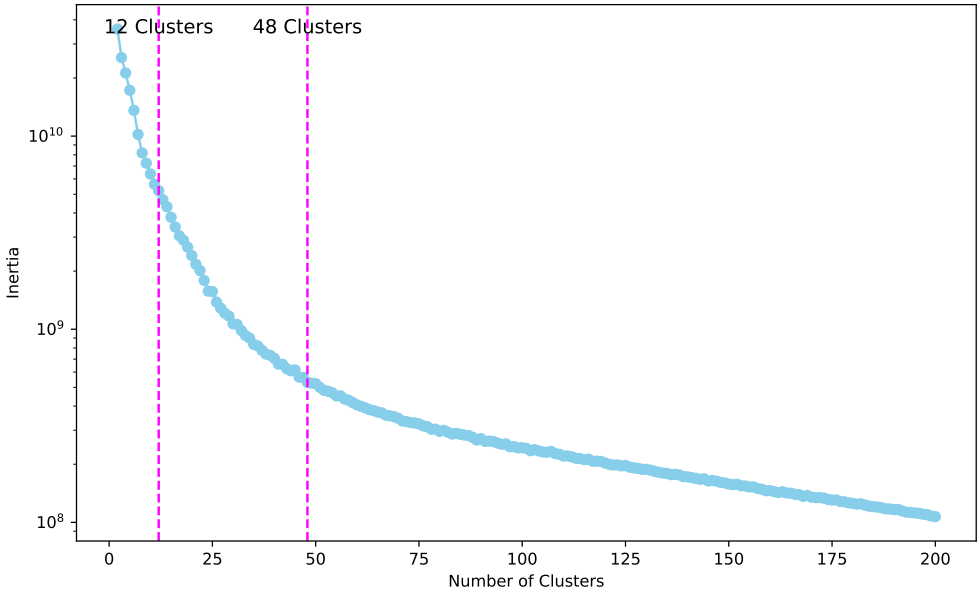


Figure 4.5: Elbow plot for 'SushiSwap' incident with corresponding target contract address `0xe11fc0b43ab98eb91e9836129d1ee7c3bc95df50` in a range from 2 to 200 clusters. The local maxima discovered in the silhouette plot at 12 clusters is marked by dotted magenta line, as well as the observable sharp decline in the rate of change of inertia at 48 clusters.

### k-means Clustering

We preform k-means clustering for both cases of 12 and 48 clusters and the results are displayed in Figures A.2 (appendix) and 4.6 respectively.

The mean number of transactions for 12 clusters are $55.9 \pm 34.6$, which is close to the mean of the 'example incident' at 78.1 transactions. Also, 5 of the 6 known adversarial transactions are in fact clustered into the same cluster, namely number 5, with only the last transaction being in a different cluster, namely cluster 2. In this regard the results appear quite reasonable, even though the standard deviation of the number of transactions is rather high. Concerning

however, are the mean values of mean TG distance and mean time difference at $472.15 \pm 338.52$ and $663.45 \pm 569.41$ h respectively. These values and their standard deviations are extremely large compared to the 'example incident' with 0.2 and 1.5 h. They indicate that very different transactions with a large variance in invocation time are clustered together, which is very counter productive for our purposes.

The results for 48 clusters give a mean number of transactions of $14.0 \pm 11.5$ which is closer to the mean of 6.9 quoted for the 'unseen data'. In this clustering also, 5 of the 6 known adversarial transactions are in fact clustered into the same cluster, namely number 8, with only the last transaction being in a different cluster, namely cluster 12. Again, this appears as a reasonable result, however, we observe the same behaviour in regards to mean TG distance and mean time difference, although not quite as extreme as for the 12 clusters result. The values are $374.34 \pm 345.27$ and $185.90 \pm 149.55$ h respectively. The values and their standard deviations again, are extremely large compared to those from the 'Evil Under the Sun' paper, and indicate that transactions with very different execution traces and large temporal differences are grouped together. Therefore, we do not consider these transactions well clustered for the purpose of discovering a collective operational intent across a specific cluster.

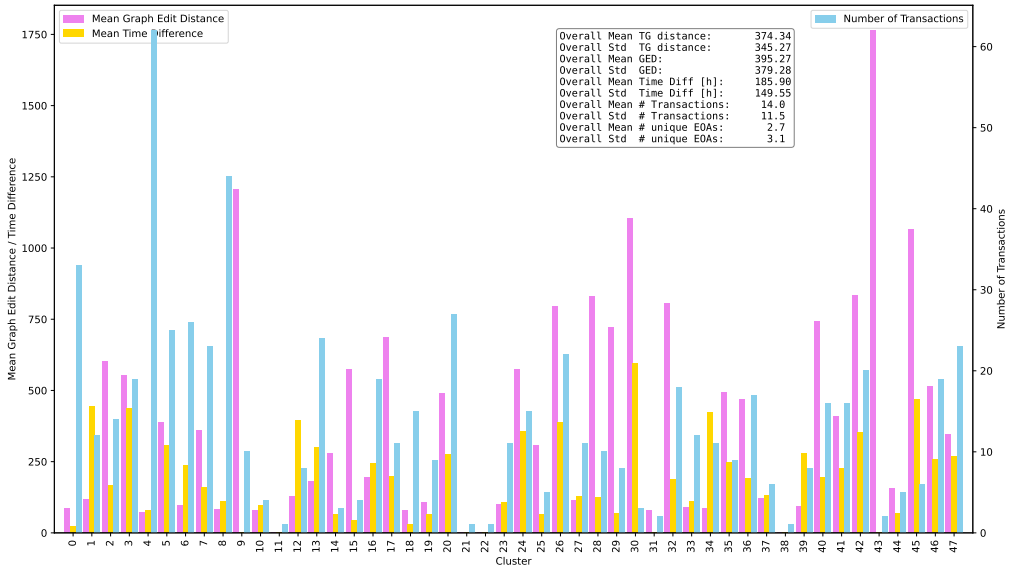


Figure 4.6: k-means clustering with 48 clusters for the 'SushiSwap' incident. Out of the 6 known adversarial transactions, 5 are clustered into the same cluster, namely cluster 8, while the last one is in cluster 12.

### 4.2.3 Akropolis

This exploit incident is on the Akropolis DApp with a vulnerable smart contract with the address at `0x1cec0e358f882733c5ecc028d8a0c24baee02004` and 17 known adversarial transactions, located at level 2. During data collection we find 1 normal transaction at level 0, 3'024 at level 1 and 774 at level 2, giving us a total of 3'799 transactions. This set is reduced by only including those transactions that lie within ± 2 weeks of the exploit's block number, resulting in 312 transactions. These are preliminarily extended to 4'699 transactions, and after disregarding those with TG distances greater than 3, it leaves 1'153 transactions for clustering.

#### Silhouette Analysis

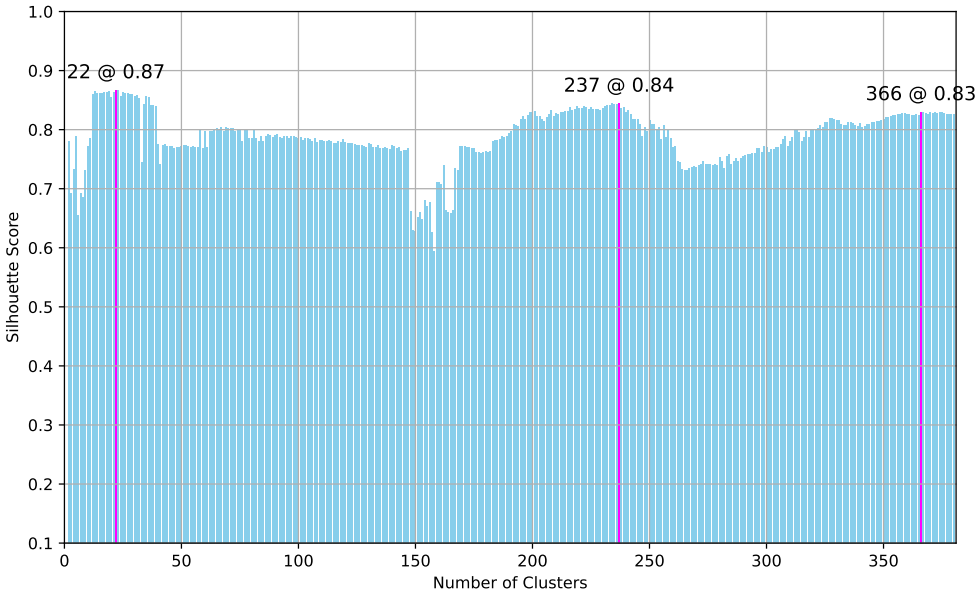


Figure 4.7: Silhouette Scores for 'Akropolis' incident with corresponding target contract address `0x1cec0e358f882733c5ecc028d8a0c24baee02004` in a range from 2 to 380 clusters. The global maximum is 0.87 for 22 clusters with one local maximum at 0.84 for 237 clusters and another local maximum at 0.83 for 366 clusters.

The results of the Silhouette Analysis are shown in Figure 4.7 for a range of 2 to 380 clusters. We observe a global maximum at 0.87 for 22 clusters, as well as two other local maxima at 0.84 and 0.83 for 237 and 366 clusters respectively. The first local maximum for 237 clusters describes a peak after a steady increase in silhouette scores, followed by a sharp decline afterwards. The second local maximum for 366 clusters also marks a peak after a steady increase, but is followed

by the scores levelling off, rather than declining afterwards. All three values are considered for clustering.

### Elbow Method

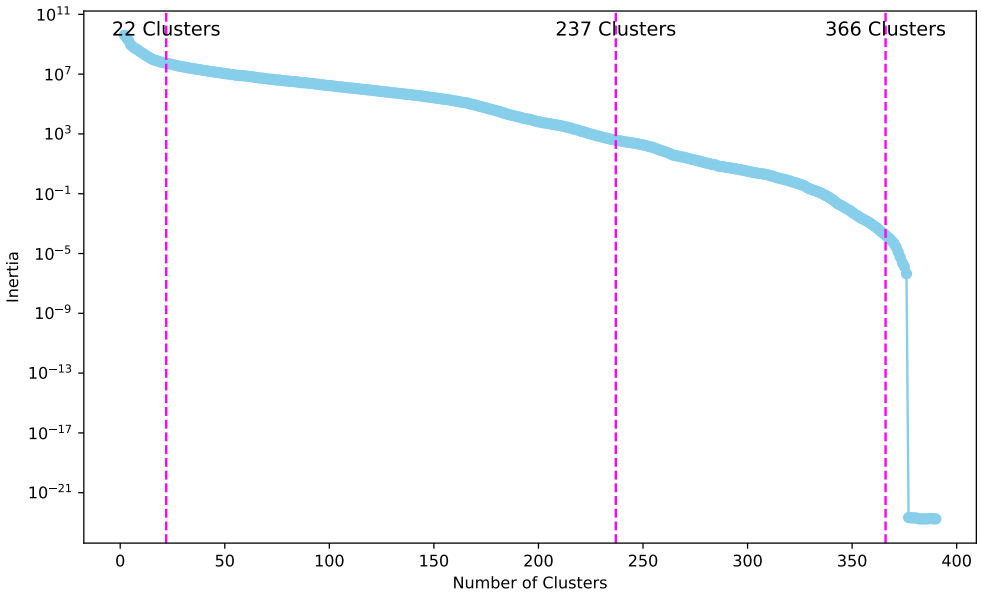


Figure 4.8: Elbow plot for 'Akropolis' incident with corresponding target contract address `0x1cec0e358f882733c5ecc028d8a0c24baee02004` in a range from 2 to 380 clusters. The local and global maxima discovered in the silhouette plot at 237, 366 and 22 clusters respectively are marked by dotted magenta lines. The global maximum at 22 clusters sits in a neighbourhood of sharp change of slope, while the first local maximum at 237 clusters is surrounded by a steady slope. The second local maximum at 366 clusters is again situated where the rate of change changes sharply, however, in this case it changes from a flat to a steep slope.

Figure 4.8 shows the Elbow plot for a range of 2 to 380 clusters and the local and global maxima found in the Silhouette Analysis at 237, 336 and 22 clusters respectively are marked by dotted magenta lines. The neighbourhood around 22 clusters, i.e. at the global silhouette score maximum, does exhibit a sharp change in the rate of change of inertia, meaning that for this value both methods agree. The first local maximum at 237 clusters on the other hand is located in a neighbourhood of a steady slope. The second local maximum at 366 clusters is at a position marking a sharp change, however, changing from a slow rate of change to a fast rate of change, which is the opposite of what we are looking for. It is noteworthy to point out as well, that at the second local maximum the

value of the inertia is already extremely low, i.e. smaller than 1, and drops off by 20 orders of magnitude afterwards. Still, in order to explore all possibilities, all three values are considered for clustering.

### k-means Clustering

We preform k-means clustering for all three cases of 22, 237 and 366 clusters and the results are displayed in Figures 4.9, A.3 (appendix) and A.4 (appendix) respectively.

The results for 22 clusters exhibit a mean number of transactions of $52.4 \pm 112.6$ which is once again close to the 78.1 mean from the 'example incident'. However, the standard deviation is greater than the mean pointing to a large dispersion. Consequently, we see 3 clusters with over 200 transactions and a majority with very few. In regards to adversarial transactions, out of the 17 known, 13 are clustered into the same cluster, namely cluster 1, 2 more are located in cluster 13, and the remaining 2 are spread over 2 more clusters, 3 and 15. This appears acceptably well clustered with the majority of known adversarial transactions grouped in the same cluster. However, the values for the TG distance and time difference are once again very high compared to the reference from the 'example incident'. We see $104.30 \pm 134.58$ and $49.70 \pm 67.52$ h respectively. Additionally, we observe a standard deviation greater than the mean again. All of this indicates that very different transaction are grouped together within a single cluster, which is counterproductive for our purposes.

We see a mean number of transaction of $4.9 \pm 11.2$ for the result of 237 clusters. This is closer to the 6.9 mean number of transactions on the 'unseen data'. Out of the 17 known adversarial transactions, 11 are clustered into the same cluster, namely cluster 2, 2 more are located in cluster 226, and the remaining 4 are spread over 4 more different clusters, 13, 95, 145 and 161. This result has the known adversarial transactions more spread out than for 22 clusters, indicating that an increase in the number of clusters, does not seem to improve the results for this incident. Appearing promising are the low values for mean TG distance and mean time difference at $0.07 \pm 0.42$ and $0.03 \pm 0.15$ h respectively. However, from the plot we observe that this likely results from most of the clusters only having a single transaction, while a few of them have larger numbers above 40. This again indicates that the transactions are not well clustered.

The result for 366 clusters appears to continue in this trend with the known adversarial transactions spread over 13 different clusters and the values for mean TG distance and time difference of the order less than $10^{-3}$. And again, we observe most clusters with only a single transaction.

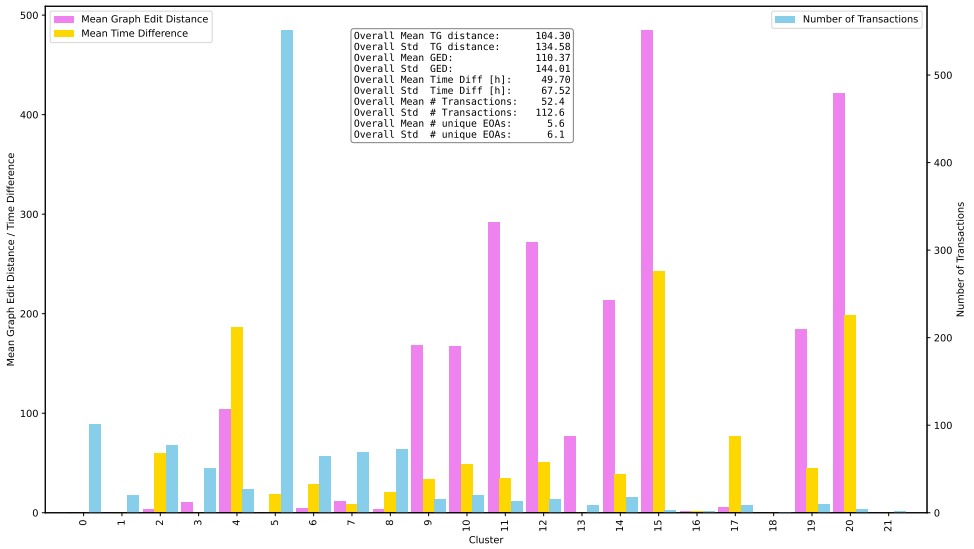


Figure 4.9: k-means clustering with 22 clusters for the 'Akropolis' incident. Out of the 17 known adversarial transactions, 13 are clustered into the same cluster, namely cluster 1, 2 more are located in cluster 13, and the remaining 2 are spread over 2 more clusters, namely 3 and 15.

### 4.2.4 CreamFinance

This exploit incident is on the CreamFinance DApp with a vulnerable smart contract with the address at `0x33bf0bb8e1405dc440eccb97ffd92fef438c8a27` and 9 known adversarial transactions, located at level 1. During data collection we find 2 normal transaction at level 0 and 4'068 at level 1, giving us a total of 4'070 transactions. This set is also reduced by only including those transactions that lie within $\pm$ 2 weeks of the exploit's block number, resulting in 1'941 transactions. These are preliminarily extended to 64'553 transactions, and after disregarding those with TG distances greater than 3, it leaves 19'593 transactions for clustering.

#### Silhouette Analysis

The results of the Silhouette Analysis are presented in Figure 4.10 for a range of 2 to 1'500 clusters. We first looked at a similar range to the previous incidents of 2 to 400, but this range only shows a global maximum at 0.86 for 18 clusters. This appears very low considering there are 19'593 transactions to be clustered for this incident compared to 1'153 transactions of the last incident. Therefore the range is extended. Eventually, we find another local maximum at 0.76 for 1050 clusters. It is located after a steep rise in the silhouette scores and followed by a slow decline afterwards. This value appears rather high. Nevertheless, both

values are considered for clustering.

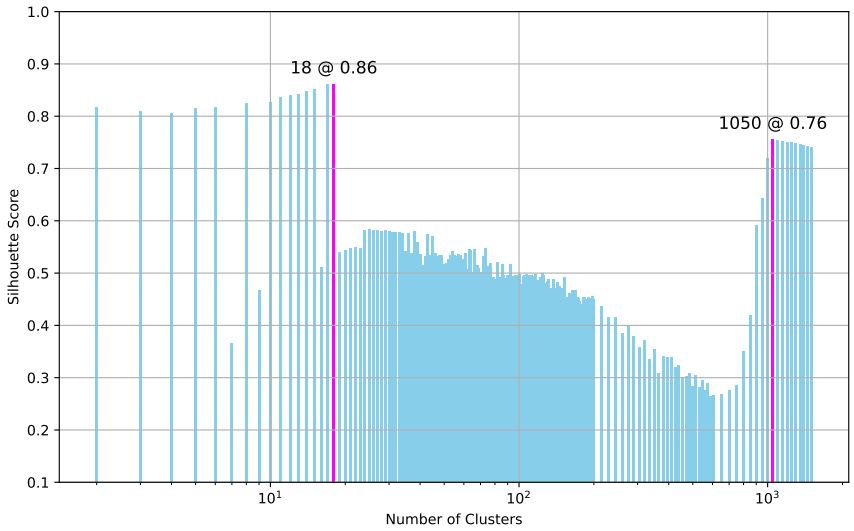


Figure 4.10: Silhouette Scores for 'CreamFinance' incident with corresponding target contract address `0x33bf0bb8e1405dc440eccb97ffd92fef438c8a27` in a range from 2 to 1'500 clusters. The global maximum is 0.86 for 18 clusters with another local maximum at 0.76 for 1'050 clusters.

### Elbow Method

Figure 4.11 shows the Elbow plot for a range of 2 to 1'050 clusters. The global maximum of the Silhouette Analysis at 18 clusters exhibits the sought-after sharp change in slope in its neighbourhood. The other local maximum, however, is located in a region of steady rate of change of the inertia. Still, both values are considered for the clustering.

### k-means Clustering

We preform k-means clustering for both cases of 18 and 1'050 clusters and the results are displayed in Figures 4.12 and A.5 (appendix) respectively.

The 'CreamFinance' clustering with 18 clusters is the only result where all known adversarial transactions are grouped in the same cluster, namely cluster 7. On first glance, this appears very promising, however, this is likely the result of the extremely large mean number of transactions per cluster at $1088.4 \pm 4008.6$. Furthermore, the entire distribution is very uneven, with cluster 1 including 17'614 transactions and a TG distance of 1.77, while 16 of the 17 remaining clusters have TG distances greater than 200 and contain exactly 100 transactions each,

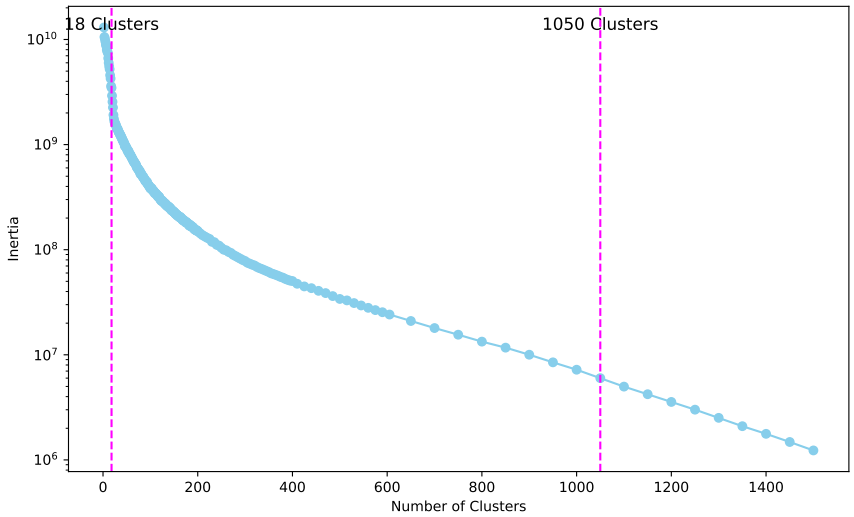


Figure 4.11: Elbow plot for 'CreamFinance' incident with corresponding target contract address `0x33bf0bb8e1405dc440eccb97ffd92fef438c8a27` in a range from 2 to 1'500 clusters. The local and global maxima discovered in the silhouette plot at 1'050 and 18 clusters respectively are marked by dotted magenta lines.

which is a strange feature in itself. All of this indicates that the transactions are not well clustered.

In the results for 1'050 clusters we observe a mean number of transactions of $18.7 \pm 36.3$, which is more reasonable when comparing to the reference from 'Evil Under the Sun'. In regards to the known adversarial transactions, 2 out of 9 are clustered into the same cluster, namely cluster 92, 2 more are located in cluster 316, and the remaining 5 are spread over 5 more clusters, namely 420, 493, 636 and 689. This shows again a result, where transactions that are supposed to be grouped together, are instead spread across multiple clusters. These results also exhibit the same strange behaviour as with 18 clusters, where a large number contain exactly 100 transactions. The reason for that is not clear at this point. The mean TG distance and time difference take the values $13.05 \pm 17.18$ and $1.48 \pm 2.35$ h respectively. The time difference appears reasonable, while the TG distance value is high compared to the 0.2 from the 'example incident'. Once more, the standard deviation for both values is greater than the mean, indicating large dispersions as well. The combination of these observations, suggest that the results are not usable for our purpose.

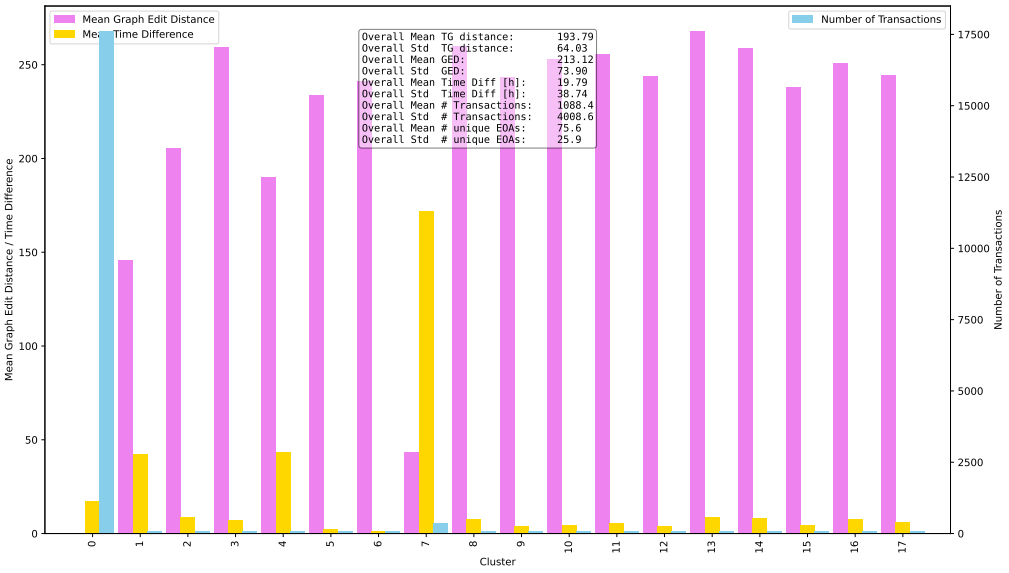


Figure 4.12: k-means clustering with 18 clusters for the 'CreamFinance' incident. All 9 known adversarial transactions are clustered together in the same cluster, namely cluster 7.

Chapter 5

# Conclusion

In an effort to reproduce and validate the DEFIER approach on a modern dataset in order to potentially build on it and further the understanding of and capability in detecting exploit transactions on DApps, specifically in the DeFi space, this study discovers various problems and limitations, as well as gains critical insight into the evolution of DApps between 2016 and 2022.

## 5.1 Summary

This section summarizes the results and findings of this investigation into the detection of exploit transaction on vulnerable smart contracts.

### 5.1.1 Data Collection

Potentially the most fundamental discovery of this study is in regards to the complexity and structure of DApps in the evolution of blockchain technology. The dataset from the 'SoK' paper on which the study is based is focused on the DeFi space, however, it stands to reason that the insights are transferable to DApps in other segments. It is reported in the 'Evil Under the Sun' paper, based on their dataset, spanning transactions in a period from 2016 to 2018, that all exploit transactions are located on level 0 or level 1, i.e. directly interacting with the DApp's vulnerable smart contract, or via one intermediary smart contract[1]. However, as this investigation shows, this is no longer the case. 69.6% of the incidents from the more recent 'SoK' dataset, containing transactions between 2018 and 2022, have their exploit transactions located on level 2 or higher. This means that they interact with the vulnerable smart contract via two or more intermediary smart contracts. It shows that the attacks exploit vulnerabilities not in the DApp's main smart contract, the point of interaction with other accounts, but in a smart contract that the DApp calls internally. This transition represents for once, how the complexity of the structure of DApps has grown between the time frames of these two datasets, but also how exploit efforts have adapted to this transition.

Another important discovery is in the context of adoption of blockchain technology. During the data collection process, when scraping transactions for an incident on different levels, i.e. direct interactions with the target smart contract or via one or more intermediary smart contracts, we find that the average number of transactions has greatly increased. The 'Evil Under the Sun' paper reports an average of 22'603.6 scraped transactions on level 0 and 1 per DApp after running DEFIER's preprocessing on 104 DApps to search for unknown exploits. In our data collection we find that the sum of the average level 0 and level 1 transactions is 122'300.4, which represents an increase by a factor of 5.41. However, the number from 'Evil Under the Sun' stems from the end of preprocessing, meaning it includes the additional transactions, as defined in Section 3.1.3, while the average quoted from our data collection refers to the state before this extension. We observe that on the 'SoK' dataset the extension itself constitutes an average factor of 3.74. In combination, this results in an average 20-fold increase in the number of transactions that need to be scraped to run the DEFIER model. Operationally, this makes the entire DEFIER approach more time consuming and computationally more expensive, which is also shown in the total size of the collected data at 102 GB. Note, that this value only contains the extended transactions for four of the 56 incidents under investigation. Furthermore, on a qualitative level, this shows that the amount of traffic on the Ethereum blockchain, i.e. executed transactions, has vastly increased since the period of the 'Evil Under the Sun' dataset of 2016 to 2018.

### 5.1.2 Clustering

The fact that neither the code nor the dataset from the 'Evil Under the Sun' paper is available means that the DEFIER approach must be reproduced from scratch only by reference to its description in the paper. This makes its validation on a modern dataset such as the one from the 'SoK' paper vastly more challenging.

Specifically in the context of transaction clustering, a fundamental problem arises, because Su et al. do not specify how they determine the number of clusters for the k-means algorithm for a specific incident. As there is no obvious analytical way of determination, the logical choice is an empirical approach. For that purpose this study has chosen the Silhouette Analysis and Elbow Method. However, in the absence of an available labelled dataset, such as the one created and used in 'Evil Under the Sun', this problem gains an extra level of complexity. An empirical approach to determine the number of clusters for an incident fundamentally requires a rigorous metric by which the results can be evaluated, and without a labelled dataset it is not entirely clear how to achieve this.

In this study we have chosen to look at whether the known adversarial transactions of an incident originating from the same EOA are clustered together, which they should be under the assumption that they constitute the main exploitation

stage of the attack, or at least part of that stage. However, being clustered together is certainly a necessary condition, but definitively not a sufficient one as is shown in the clustering results with 18 clusters, of the 'CreamFinance' incident. This is the only clustering result where all known adversarial transactions are grouped in the same cluster. Nevertheless, the transactions can not be considered as well clustered, because out of 19'593 transaction fed to the k-means algorithm 17'614 are grouped in cluster 1 with a mean TG distance of 1.77, while 16 of the 17 remaining clusters contain exactly 100 transactions with mean TG distances greater than 200. All in all, it is fair to say that the clustering efforts across the four investigated incidents are quite unsuccessful. At this point unfortunately, it is not entirely clear if that is a result of the technical method, or due to other factors. It is entirely conceivable that while the complexity of DApp structures has evolved over time, the complexity of calls and subcalls within a transaction have evolved as well. If this should be the case, then it is possible that the concept of the TG distance as defined in Equation 2.1, which is based on a weighted sum of execution time difference and the graph edit distance between graph representations of transaction execution traces, is no longer a viable metric for such clustering purposes.

## 5.2 Going Forward

In regards to further investigation into exploit transaction detection, particularly in the context of the DEFIER model, the recommendations are clear. There is a strong need for a labelled dataset, specifying transactions at a specific stage of the attack lifecycle. Without it, it remains extremely challenging to evaluate the clustering results based on an empirical determination of the suitable number of clusters, since there is no rigorous analytical metric for it. Furthermore, beyond the clustering step lies the sequence-based classification, which also requires a labelled dataset for training of the neural network.

In this context the question also arises, if the four-stage attack lifecycle, which constitutes the basis of the entire approach, as discovered in the 'Evil Under the Sun' paper, is still valid. The paper mentions for example a case where the propagation stage is based on multiple gambling DApps utilizing the same flawed random number generator, which does not constitute a complex vulnerability. With an evolution in the complexity of DApps it is reasonable to assume that there is also an evolution in the complexity of vulnerabilities. This might lead to the result that reusing, even of adjusted exploitation code is no longer possible.

As the reproduction of this approach, for the purpose of validation on modern data in order to then build upon it, poses significant challenges, it is consequently a very time consuming task. Coupled with the possibility, that its underlying premise, i.e. the validity of the four-stage attack cycle, may no longer hold true, due to an evolution in the complexity of DApps as well as their vulnerabilities,

we would recommend to proceed in one of two ways. The first option is to obtain a working implementation of DEFIER for the original source, i.e. Su et al., and run it on modern data such as the dataset from 'SoK'. If possible, this is the more straight-forward approach, and will enable a swift evaluation, if the underlying premise is still valid. Alternatively, if that is not possible, our recommendation is to start with an evaluation of the validity of the four-stage attack lifecycle, which likely entails manual assessment of exploit transactions. We suggest this course of action, because if the concept of the attack lifecycle should indeed no longer apply, then a reproduction of the DEFIER model becomes a vain effort.

# Bibliography


[1] L. Su, X. Shen, X. Du, X. Liao, X. Wang, L. Xing, and B. Liu, "Evil under the sun: Understanding and discovering attacks on ethereum decentralized applications," *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[2] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "Sok: Decentralized finance (defi) attacks," *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[3] [n.d.], "Coingecko," last accessed 26 August 2024. [Online]. Available: https://www.coingecko.com/

[4] [n.d.], "Ethereum," last accessed 26 August 2024. [Online]. Available: https://ethereum.org/

[5] [n.d.], "Alchemy," last accessed 26 August 2024. [Online]. Available: https://docs.alchemy.com/

[6] C. Russo, *The Infinite Machine: How an Army of Crypto-Hackers Is Building the Next Internet with Ethereum.* HarperCollins, 2020.

[7] [n.d.], "Etherscan api," last accessed 27 August 2024. [Online]. Available: https://docs.etherscan.io/api-endpoints/accounts

[8] [n.d.], "Etherscan," last accessed 27 August 2024. [Online]. Available: https://etherscan.io/

[9] [n.d.], "Blockscout api," last accessed 27 August 2024. [Online]. Available: https://docs.blockscout.com/developer-support/api/rpc-endpoints/account

[10] [n.d.], "Infura api," last accessed 27 August 2024. [Online]. Available: https://docs.infura.io/api/networks/ethereum/json-rpc-methods/trace-methods/trace_transaction

[11] [n.d.], "Networkx - directed graph," last accessed 27 August 2024. [Online]. Available: https://networkx.org/documentation/stable/reference/classes/digraph.html

[12] [n.d.], "Gmatch4py," last accessed 27 August 2024. [Online]. Available: https://github.com/jacquesfize/GMatch4py

[13] [n.d.], "Bloxy api," last accessed 27 August 2024. [Online]. Available: https://bloxy.info/api_methods

[14] [n.d.], "scikit-learn, kmeans," last accessed 28 August 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[15] [n.d.], "scikit-learn, silhouette_score," last accessed 28 August 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

[16] R. Thorndike, "Who belongs in the family? pyschometrika 18 (4): 267–276," 1953.

Appendix A

# Plots for k-means Clustering Results

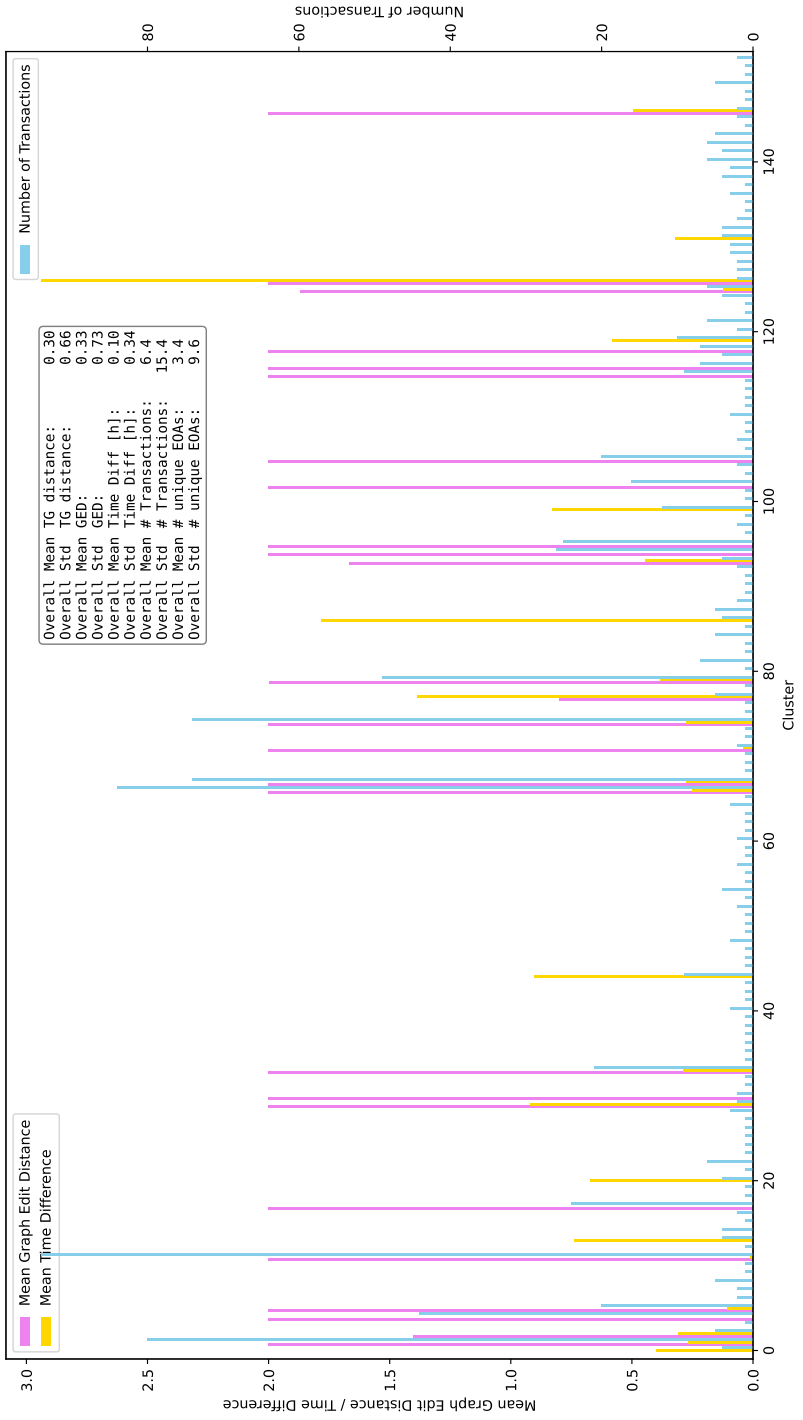


Figure A.1: k-means clustering with 153 clusters for the 'InverseFinance' incident. The 2 known adversarial transactions are clustered into different clusters, namely cluster 25 and cluster 37.

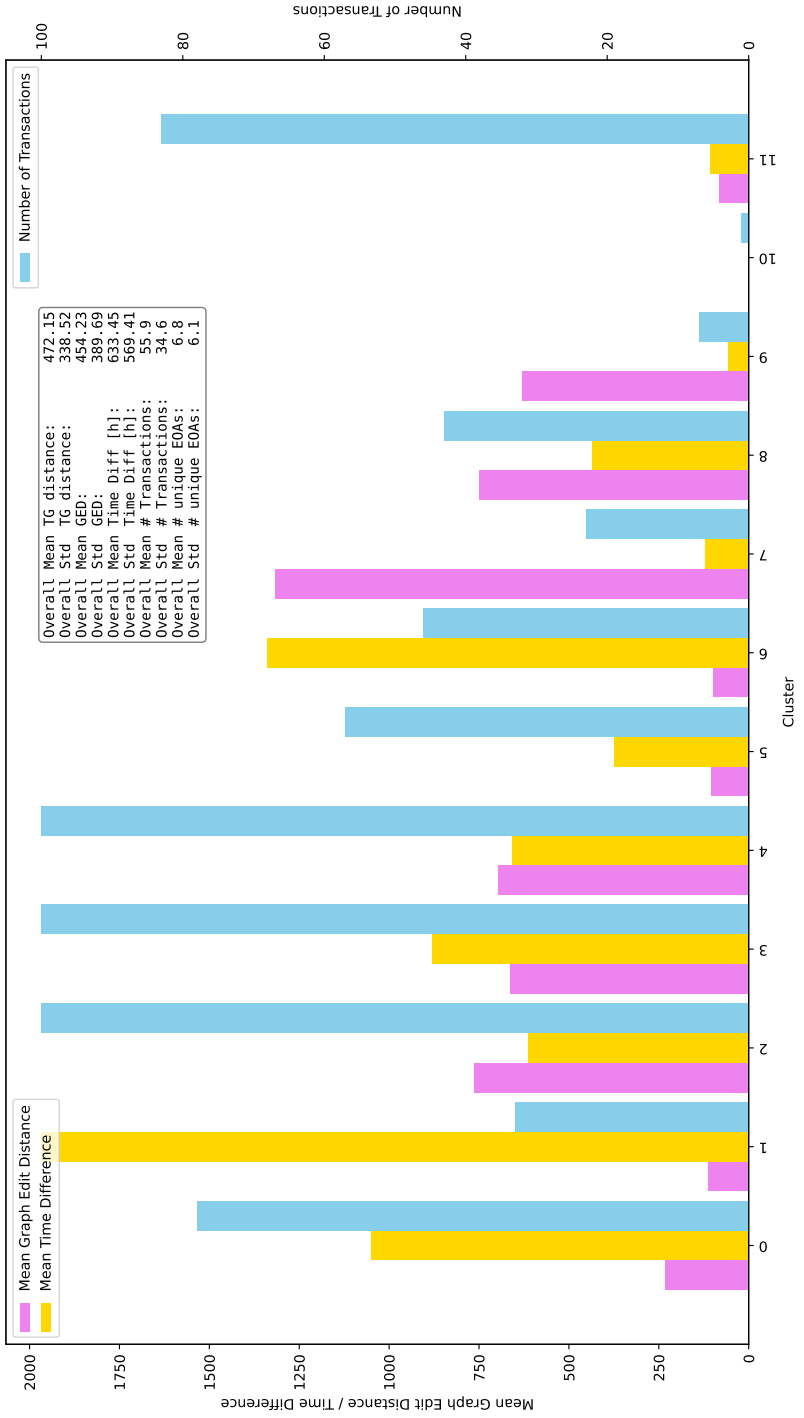


Figure A.2: k-means clustering with 12 clusters for the 'SushiSwap' incident. Out of the 6 known adversarial transactions, 5 are clustered into the same cluster, namely cluster 5, while the last one is in cluster 2.

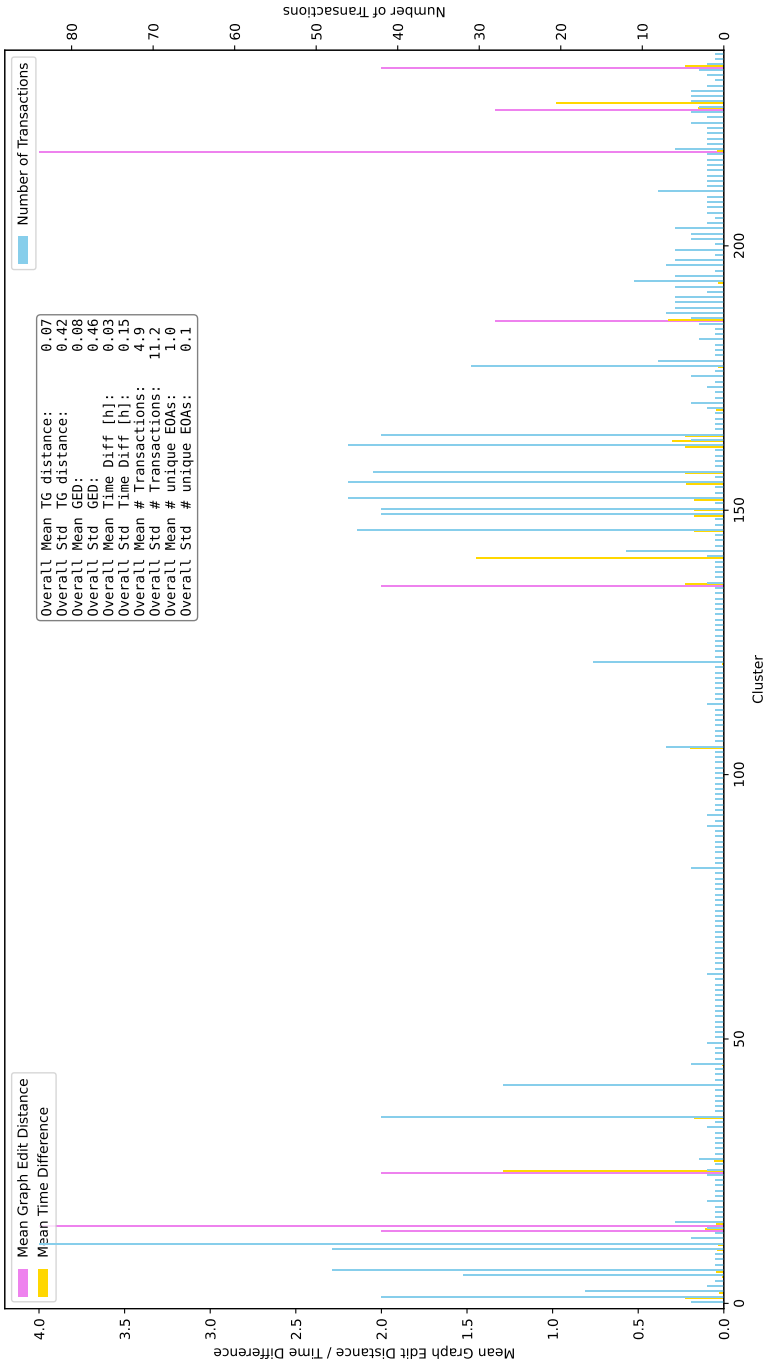


Figure A.3: k-means clustering with 237 clusters for the 'Akropolis' incident. Out of the 17 known adversarial transactions, 11 are clustered into the same cluster, namely cluster 2, 2 more are located in cluster 226, and the remaining 4 are spread over 4 more different clusters, namely 13, 95, 145 and 161.

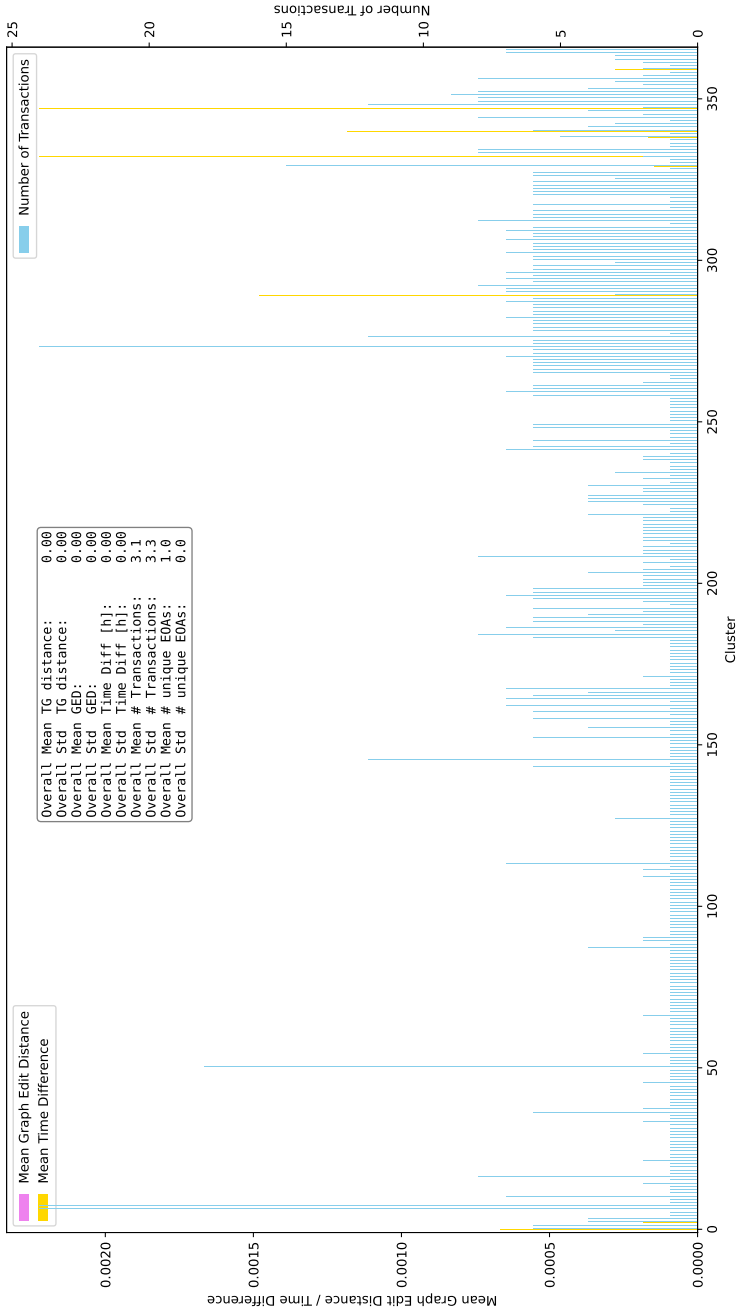


Figure A.4: k-means clustering with 366 clusters for the 'Akropolis' incident. Out of the 17 known adversarial transactions, 3 are clustered into the same cluster, namely cluster 340, 2 more are located in cluster 2, and another 2 are located in cluster 332. The remaining 10 are spread over 10 more different clusters, namely 23, 95, 149, 163, 228, 263, 318, 339, 354 and 358.

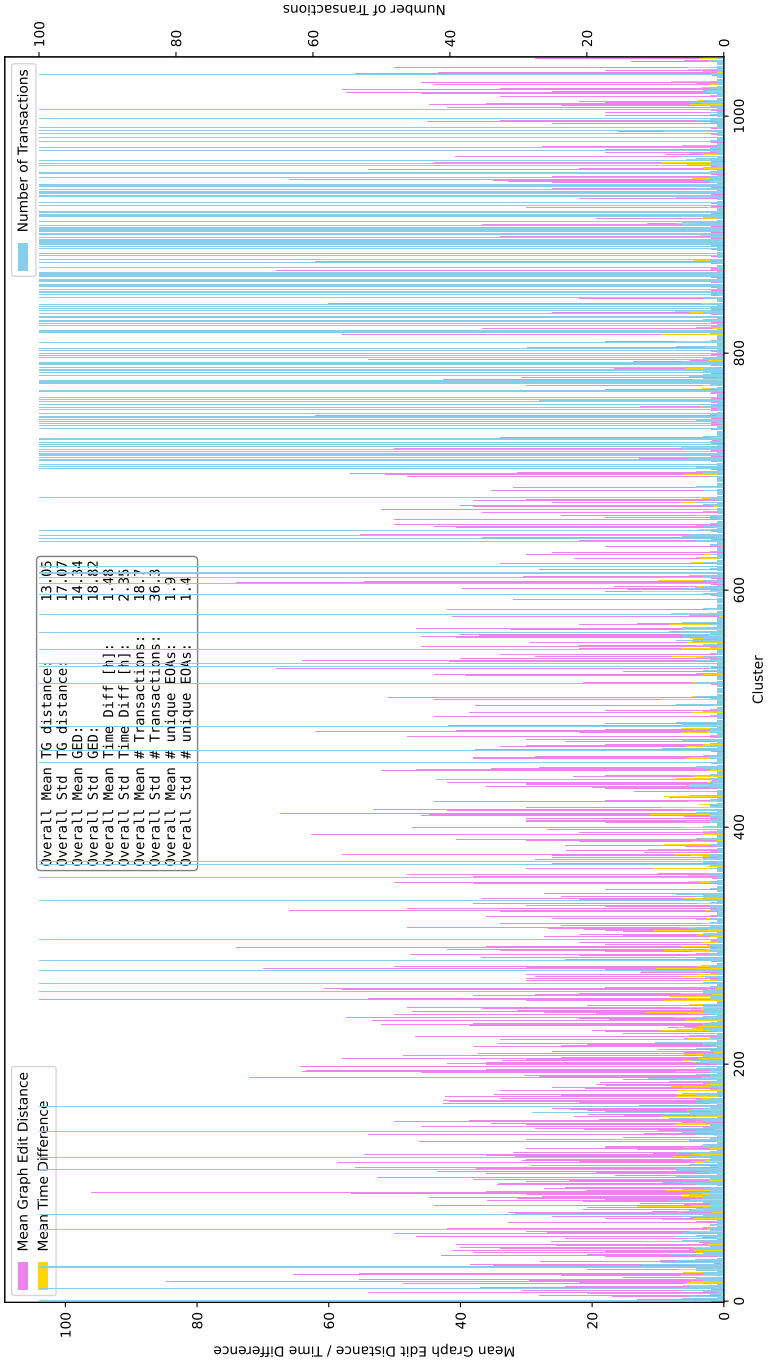


Figure A.5: k-means clustering with 1'050 clusters for the 'Akropolis' incident. Out of the 9 known adversarial transactions, 2 are clustered into the same cluster, namely cluster 92, 2 more are located in cluster 316, and the remaining 5 are spread over 5 more clusters, namely 420, 493, 636 and 689.