Scenario-based Creation and Digital Investigation of Ethereum ERC20 Tokens

Simon Dyson, William J Buchanan, Liam Bell

Blockpass ID Lab, School of Computing, Edinburgh Napier University

Abstract

This paper examines the Ethereum network in the context of an investigation. The validation of data sources is achieved through different client software on both the Ropsten network and the live blockchain. New scenarios are also used test common patterns in order to track for start and end points for Ethereum and ERC20 tokens.

Keywords: Blockchain, Cryptocurrency, Ethereum

1. Introduction

1.1. Introduction

Due to the large sums of money that have become synonymous with cryptocurrency a rise in criminal activity has been recorded. These range from attempts to phish users in chat rooms to large scale protocol vulnerabilities notable, the "Dao" hack that netted the attacker millions of pounds worth of bounty from a code error [1]. Chainanalytics report that $225 million was made by criminals targeting Ethereum and coin offerings in August 2017. Reports of a large-scale mixer involved in tumbling nearly 68% of total Ethereum transactions also reveals how little is known about the billion-dollar infrastructure [2].

This paper investigates Ethereum blockchain technology in order to understand the transactional structures and outputted data. The data will be reviewed for evidential benefit, association, and attribution in order to apply a method for investigative processes, and investigates the tokens supported on its internal infrastructure, such as the ERC20 token standard [3]. The purpose is to create a method for investigating transactions across the network understanding the protocol and identifying tools for use by investigators. It outlines a methodology for recording transactions for timing and set in a certain order so that when data is recovered it can be tested against a known sequence of events. The testing is conducted on a testnet work that replicates the live network in order to gather data without costing for transactional testing. Utilising the method of a known sequential order the data retrieval can take place and the quality of the data analysed for use to the objective set out. The use of a number of different data sources will ensure reliable source data that can be validated and replicated.

In addition to the transactional nature of blockchain technology Ethereum has implemented smart contracts that contain computational steps that operate on the blockchain, so how do the contracts store and retain data, what can an investigator see when an entity transacts with a smart contract? Where is a contract deployed or controlled and what information can be attributed to linked accounts or the owner? The Ethereum Naming Service assigns humanly readable addressing to complex hash addressing, what information linked records or detail can be obtained by querying the system. Comparing the transactional events using multiple data sources and tools enables a comparison of core data fields and the ability map the fields where they differ in a standard naming convention.

The key themes that will be addressed through this paper are:

- Scenario Design, develop a set of scenarios that will replicate real-world transactions and use the protocols and activate responses for analysis.
- Scenario Set-up, complete all the stages in order to set-up the scenario this will include system and software set-up creation of accounts and obtaining of test ETH. Creating the transactions and recording the details.
- Data extraction, using identified systems and services obtained from the data on the network. This will replicate the observer following an event. The data will be obtained for further analysis.
- Data Analysis, compare the quality of the data for its use in the set objective and comment on further use cases. Develop key fields mirrored on the data sets to create a keymap.
- Feedback, review the above steps and if any key failings or key findings found and require expanding, validating or new scenarios devised then detail and implement.
- Present Data, use the obtained data to create a visual or present data in order to assist a non-technical viewer or ease the analysis by visualizing the data.
- Create Method review data source analysis the data quality and the presentation phase. Create a method that will enable a consistent approach to transaction tracking and account identification that will serve as a base procedure.

1.2. Financial Investigations

In the tracking of assets across a network there needs to be some understanding around the context of who is involved in the movement. Let's outline where tracking occurs following a third-party theft, where criminals use the network to obfuscate or transfer funds. Criminal gangs may utilise the international and non-regulated nature of cryptocurrency to launder money from any criminal activity. Ransomware utilises the medium of cryptocurrency as a primary form of payment in order to extort money from the victim. Large-scale attacks against the NHS, such as Wannacry, demonstrate the highly disruptive nature of such attacks. "Organised Cybercrime or Cybercrime that is organised?" [4], summarises some key concepts about the structure of organised cyber gangs. They establish that cyber offenders work in connected groups and they educate each other in order to upskill their criminal networks. The structure of the groups fall into three distinct groups, although not a strictly structured organisation. They comprise of "core members" who are highly skilled and fully engaged in the crime, "enablers" specialists for hire and "money mules" low paid members to facilitate the spread and conversion to value. The study involved the analysis of a number of criminal gangs across the globe. It also highlighted four types of networks:

1. Completely through off-line social contracts.
2. Off-line social contracts as a base and off-line forums to recruit specialists.
3. On-line forums as a base and off-line social contacts to recruit local criminals.
4. Completely through on-line forums.

The study found that criminal groups favoured (1) + (2) and contained between 5 – 10 members [4]. This can focus the investigator's mind that *trust* is favoured for criminal enterprise despite the technological ability to work worldwide, criminals prefer trusted social contacts. This approach will potentially enable geo-location opportunities if presented and should be sought to detect the group's location.

The Internet has enabled criminals to virtually interface with buyers directly in on-line markets that assist in lowering the risk of detection. Illicit criminal markets operate on the *darkweb* a privacy-centric part of the Internet that requires special software to access. The software is free and tutorials are readily available. The most famous criminal market was the Silk Road, its prominence and links to cryptocurrency are significant and merit discussion in a later section. *Darkweb* markets are similar to e-Bay for banned items such as drugs, weapons and indecent images of children.

A study into Internet drug selling activity highlights important areas of potential detection using the Hancock and Laycock script [5]. The conceptual framework identifies areas that criminals can utilise to expand their ability to interact, in this case, a drug supply operation over the Internet but it applies well to a number of cyber-based criminal activity [5].

1. Communication – Messaging platforms, social media and encrypted messaging.
2. Informational activities – Educational material, law.
3. Technical opportunities – Application of utilities to further objective, logistics, execution.
4. Managerial opportunities – Monitoring, chain management.
5. Organisational opportunities – Depending on crime type adaptive layers new intermediary's.
6. Relational opportunities – Expand or interact with new bad actors or customers.
7. Promotional opportunities – Interact with markets.
8. Persuasive opportunities – Customer reassurance on the safety of interaction and product.
9. Marketing and loyalty building – Customer building and retention.
10. Countermeasure opportunities – Preconditions to detect undercover officer activity.

The above categories demonstrate opportunities to connect and link potential criminal activity to accounts or cryptocurrency movement following a theft or utilisation of a criminal service. Investigators should trawl areas here for public addressing, for accounts belonging to criminals or utilised third-party services. The area where a criminal enterprise intersects with other legal or non-criminal enterprises provides possible chances to attribute to identifiable accounts [6].

2. Ethereum

2.1. An Ethereum transaction

In order to discuss how the Ethereum protocol executes transactions across the network a comparison against the Bitcoin protocol will assist in highlighting its differences. Bitcoin is the most well know and largest blockchain or decentralised ledger technology currently in operation. Bitcoin operates by utilising cryptographic public / private key signing of the transaction hash. A transaction is commenced by sending the amount of Bitcoin required along with the hash value of the previous block. The transaction is signed with a private key of the sender and the public key of the receiving party. This creates a chain that is verified at each stage. Bitcoin needs additional features to prevent double spend and secure the network. It requires that a time server hash is built in to ensure the time of transaction is recorded and becomes part of the signature negating forgeries. The transactions are then sent to the chain to be mined in the (POW) proof of work algorithm.

The Bitcoin chain takes the recorded ledger and creates the hash of the entire system and all the accounts. It then will allow user *A* to reduce their account balance and send to user B and add the value to user *B* including any return outputs. The snapshot of this new updated true state is then captured with the hash and signed with receivers' public key and the senders private key.

Ethereum uses *Ether* or *ETH* to power the network, this is sometimes confused as a cryptocurrency. Ether does have a store of value but was designed as the fuel for the system. Ethereum has a different approach to how a transaction takes

place across the network. It is important to separate in the first instance the two types of accounts that exist on the Ethereum network. These are *externally owned accounts*, accounts that are controlled by a private key and would include a common human user account. The other type of account is a "contract account". This is an account that is controlled by the code it holds within the contract itself on the Ethereum network [7]. This *contract account* is one of the most unique features of Ethereum and as it is a Turin complete language so it can carry out complex operations. Ethereum has created a distributed E.V.M Ethereum Virtual Machine that completes the programmable steps sent to the network.

The *contract address* holds the *smart contract* a set of instructions that operate when interacted with either via transactions or by an environment variable. The term "smart contract" will be used to describe this type of transaction throughout this paper and is considered a small program that can be executed by the EVM. The Ethereum platform operates as a base layer to utilise as a transaction of value, a platform or to host a smart contract or to build a *Dapp*, a decentralised application, running on the Ethereum Virtual Machine.

It is important to understand in basic terms what the Ethereum Virtual Machine (EVM) is in order to detail Ethereum's basic functions. The EVM is a virtual machine, an emulated system across the hardware and software of many nodes. The computer is a 256-bit machine that is made of nodes holding all the transactions on the local machine before calculating the state in sync with the full system. These nodes make up the Ethereum network and the EVM is the combination of these nodes communicating as a *Global singleton machine* [8].

The machine is thought of as one large object containing the values and having a shared state across the nodes. The "state" of the system is key to how Ethereum can run more complex computer operations by calculating the current status of the accounts and receiving object values before calculating the next state. This new state is captured by the new cryptographic hash. The hashes of the system state are then added to the blockchain and these structures build and form *Merkle trees* from a Genesis block to the root down the branch to the leaf. The use of a root hash as a Patricia Trie Merkle structure enables a more efficient search and storage mechanism. This is particularly useful for light-client nodes that do not have to download the entire blockchain [9].

The Ethereum Virtual Machine (EVM) has functions that are associated with computing such as read/write functions to memory and storage. The EVM uses operation codes that instruct what function to perform and utilises operands in order to perform instructions passed to the EVM. As previously discussed the system is Turin complete and can, therefore, build complex computational structures that enable advancements in Bitcoin's transactional ledger system. The ability to compute a *smart contract* provides new use cases for transactional behaviour to encompass trust and efficiency. Smart contracts were proposed in 1997 by Nick Szabo who compared them to vending machine transactions, entering into a contractual agreement by placing money into the vending system and then allowing the programmable conditions to dispense the output such as a chocolate bar or a canned drink.

Ethereum builds on this idea and gives users the ability to create complex conditional outputs. The system can settle financial derivatives, exchange currency from one cryptocurrency to another, or even to gambling platforms. Ambrosus is a company that is looking to create trust in the food and pharmaceutical industries by creating sensor technologies to improve efficiency in monitoring for company purposes but which also provide consumer trust. The blockchain has to interact with off chain assets that feed into the "smart contracts" to execute a result [10]. The consumer can monitor the food source and if the food was maintained in the correct ambient temperature throughout the supply chain.

Smart contracts are executed by code, they are effectively *bots*. Smart contracts can perform additional transactions on the Ethereum network when their internal code triggers additional instructions. All these actions are executed by the computer, the EVM, and are executed across all nodes on the network and agreed upon as consensus and written permanently to the Ethereum blockchain. They cannot be changed by human interaction. An example is to pay A a percentage of Y when the status of block time confirms the current date as Z, when the code is committed it is not possible to stop if the condition is then confirmed.

In Ethereum for every transaction and smart contract operation, the use of "Gas" is required. Although this causes some confusion for new users, it is a logical important network security. The term "GAS" is used when Ethereum performs instructions on the EVM and is required to be specified when a transaction or message is sent for execution. Each transaction that is sent to the EVM is calculated and is mined by a POW (Proof of work) algorithm, this is similar to Bitcoin. In basic terms, miners are rewarded for the work of calculating the transaction states. This process gives "consensus" across the network and allows all the nodes to trust the result. The trust of the system is supported by signed hashes that are used during the calculation, helping to ensure a bad actor doesn't replicate the identity of another signee using a Sybil attack. The Ethereum network is moving away from the POW system to POS (Proof of Stake) algorithm. This system requires nodes to financially be punished for malicious activity.

The two types of Ethereum accounts are externally-owned and contract. They both use key-value mapping to create an object with specific attributes that are used to set the system state. Each "account" is represented by a 20-byte long string and contains four data fields, "nonce", "current balance", "contract code" and "storage" [11]:

1. The *nonce* is a unique counter on the sending account and is utilised to ensure the account cannot double spend.
2. Current *Ether balance* the amount of Ether owned by the account.
3. If present any *Contract code* Code hash (Used by Contract accounts)
4. A *storage* field empty by default, storage trie root. (Used by Contract accounts)

On the Ethereum network a *Transaction* is a term used when sending a signed data package storing a *message* from an *Externally owned account*. Each message sent by an externally owned account as a "Transaction" includes the following items of information:

- The "TO" address the destination of the transaction.
- A signature of the sender in order to identify the sender. (EDSA signature values)
- A numeric value to designate the amount of "Ether" required to be sent to the "TO" address.
- An optional data field.
- The "STARTGAS" value, this represents the total number of allowed steps to be computed by the E.V.M.
- The value "GASPRICE" is the fee that the sender will pay per computational step sent to the E.V.M.

It is also possible for "contract accounts" to send a "message". The contract will use the opcode (operation code reference) "CALL" to create and execute a message. This operation is called entirely from within the contract code and is not operated by a user account. The set of values in the message are also structured like an "object" [12]. Each message has the following data:

- The "SENDER" of the message.
- The "TO" address the destination of the message.
- Amount of "Ether" to transfer alongside the message.
- An optional data field.
- The "STARTGAS" value, this represents the total number of allowed steps to be computed by the E.V.M.

The Ethereum transaction is defined in the white paper as the state transition function and is described as taking the following steps during execution:

1. Receive transaction ensure that all fields are present and the signature sent is valid, ensure the accounts *nonce* number is correctly in sequence. If any of the above is not met then return an error.
2. The transaction is required to be paid from the sender's address determined by the signature. The STARTGAS number of steps required to execute is multiplied by the GASPRICE giving the overall total required, the value is deducted if the funds are available. If the funds are not available then the system returns an error.
3. At this stage, the EVM is given the STARTGAS value and as each step is executed a portion of the GAS is removed per byte.

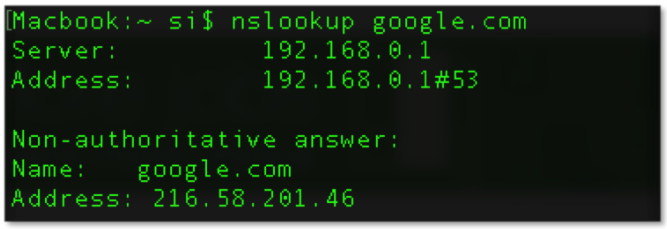

Figure 1: DNS "nslookup" against Google

4. Deduct the value of Ether as specified from the sender account to the receiving "TO" account. In the event that the account doesn't exist create a new account. If the address is a "contract account" then run the code until the contract is complete or the gas runs out.
5. In the event that the transfer has failed as the payment for the transaction is not met or the gas expired prior to finalisation then revert to all state changes to previous. The fee, however, will be deducted and paid to miners account.
6. If all steps are executed then any leftover gas can be returned to the sender and the fees paid for gas used can be paid to the miner.

The difference between account types and the ability to interact with smart contracts shows that it will be required not only to follow the money but also the code. The interaction with code creates new lines of inquiry. It may be possible to look at Gas pricing to establish patterns or assumptions of automatic or human traits.

2.2. Ethereum Name Service (ENS)

The Ethereum Naming Service "ENS" is a system that replicates the basic functionality of the Domain Name System (DNS) that currently operates on the Internet. The Ethereum Name Service is still in an early adoption phase, and there is an auction system in order to secure a domain or subdomain.

The objective is to replace the numeric IP (Internet Protocol) address in the DNS system or the hash value in the ENS, to a more human readable formatted address. In Figure 1 a lookup against the name server is performed against *google.com* and the IP address returned is 216.58.201.46. The term google.com is a much easier address for humans to remember in place of an IP address [13].

The Ethereum Naming Service (ENS) system provides a similar service in order to name wallets, contracts, resources on swarm and IPFS (Inter Planetary File System) [14].

Figure 2 and Figure 3 show Etherscan's "EWHOIS", and which allows a query for the details behind the "Ethereum.eth" [15]. Using this data, it is now possible to see the owners Ethereum account address. The name is shown as *ethereum.eth* below are the values for the LabelHash and the NameHash.

At Devcon 3, the Ethereum developers conference that took place in 2017, and further announcements were presented around the building of the permanent Ethereum Naming Service and future features. The system is currently running a version on the *ropsten* testnet that integrates with DNS. The system

Figure 2: Ethereum "EWHOIS" against the address "ethereum.eth"

Figure 3: Ethereum "EWHOIS" address information

will utilise the DNSSEC protocol that integrates signed DNS records that will allow the ownership of other top-level domains [16]. The ENS currently only allows *.eth* and *.test* TLD top-level domains to interact with Ethereum. The test DNS TLD allows *.xyz* on the system and up to three-quarters of other TLD will be included when the ENS integrates to the main network. The DNSSEC needs to allow for RSA encryption and SHA256 hashing [17]. The ENS will use an Oracle to mediate between the ENS and DNSSEC sources [18]. The integration of ENS to the new DNSEC services is still in development and it is expected that further changes will be implemented during and after the submission of this thesis [19].

Investigating the Ethereum Naming Service is another source of potential evidential product or intelligence. The combination of DNSSEC will add a further layer of potential investigative leads. Taking a view of security DNS or forensic papers could reveal techniques that may transpose to Ethereum investigations. Use of commands and sections relating to registrants and DNS owners enables a systematic approach to locating owners or controllers [20]. These stages will need to be tested against an ENS and DNSSEC hosted address to see if it possible to retrieve similar information.

3. Methodology

This section outlines the setting up of scenarios will be explained alongside the thematic areas to be investigated.

3.1. Scenario Implementation

The method sets out the main stages of requirements in order to reach the objectives. In the subsequent sections, detailed descriptions of how this will be achieved will cover the main areas highlighted previously. The study sits in the positivism method of truth discovery and the objective is determining a method and detailing a sequence of events and the identity of the entities involved.

3.1.1. Scenario design

In order to enable a real-world analysis of transactions, they must replicate what is likely to be found by an investigator. The first stage of the transaction is where a person in control of an account will send a store of value to another account, in this case, it will be "ETH" Ethereum on the testnet. A number of accounts will be created in order to facilitate the patterns and then a series of transactions will be sent in the patterns laid out to replicate what might be expected. In order to ensure consistency with they will all be sent via the same machine, service and in the same way. This will reduce any environmental contamination from another wallet, service or system specific residue.

Ethereum "ETH":

1. A to B
2. A to B to A
3. A to B to C
4. A to B to C + D
5. A to C + D and C + D to A
6. A to B to C to D to E
7. I to F to E to D to H to G to C to B to A
8. A to B + C + D and B to E and C to F and D to G and E to H and F to H and G to H

The above scenario will test how a method can interpret the flow and movement of transactions. ERC20 "Token":

1. i. Token from A to B

3.1.2. Testing for variation

Using a straight forward transaction pattern A to B a number of different account providers, wallets and service are used to create a transaction. This test will look for environmental variables where an analysis will focus on any difference that can be detected. This can be conducted during the live scenario as this uses the live network and will provide a more realistic picture. As this is about environmental difference Ropsten operates on different GAS conditions.

3.1.3. Live scenario

A live scenario will be conducted that will incorporate different wallets, ethereum, and ERC20 tokens. The scenario will replicate real activity involving movement to and from a cryptocurrency exchange and a shapeshift swap from one token to another. This will all be conducted on the live network using real cryptocurrency. This is to validate the methods tested on the Ropsten testnet.

3.1.4. Smart-Contract

As Ethereum has the ability to use smart-contacts to interact on the network it is important to see how contract interaction differs to that of a regular transaction. If contract *A* is interacted with what information is available about the controlling addressing of the contract. As ERC20 tokens, ENS and DEX decentralised exchanges are contract-based, these will be discussed as to what information is available. An ERC20 token will be deployed as a contract on the Ropsten network.

3.1.5. Ethereum Naming Service

The ENS system assigns a human-readable name to be applied to the hash addressing that is more often associated with cryptocurrency/blockchain addressing. In order to test how the addressing works. A transaction will be performed after an ENS name has been assigned to the wallets. The process will be looked at for the assigning and lookup of the registry:

1. ENS address A to ENS address B

3.1.6. Wallet files, Processes

This section is to complement the study as it will assist investigators and researchers to locate files of interest in Forensic or security related tasks. The location of key files or mechanisms used for key back-ups or recovery. A review of password cracking tools for specific Ethereum key files will be visited. This section will not be a full forensic study as this is out of scope in order to focus on the key area of transactional recovery. In addition to files on disk the recording of processes utilised by the wallets or software / services will be noted.

3.2. Data extraction

The use of extracted data is paramount to the thesis objectives. In order to review the post-event data to interpret a sequence of temporal events and the entities involved the source of data must be accurate. The data source requires gathering in a way that preserves the integrity of the data. A number of data streams are to be used in this study in order to make an assessment of data quality the study will also validate the sources. Following the study, a number of key fields will be highlighted so key mapping can occur if not already formed into a standard. Where differences exist in the data standards comment will be made to review for the objective. Where data contradicts or variables occur it will be important to draw conclusions on the cause. The data extraction will occur via a number of methods interacting with blockchain software libraries, blockchain wallets/explorer, special websites and designed software.

Data sources:

- MIST – Wallet / Browser the official Ethereum browser and wallet.
- Parity – An Ethereum network client with a number features.
- Geth – A Command Line Interface (CLI) toolset for the Ethereum blockchain in "Go".
- Etherscan Web – An Ethereum block explorer, web-based includes additional tools.
- Etherscan API – Application programming interface for Etherscan.
- Blockseer – A web-based tool for Bitcoin and beta Ethereum investigation includes visualisation.
- ENS Domains – Web-based look-up for ENS.

The abstraction of data will be done by querying the services and requesting the data. The data will be extracted in its native form or preferred is the JSON format. The conversion will take place to JSON where possible.

3.2.1. Data analysis

Manual mapping of fields to create a keymap of corresponding data sections will enable comparison and validation of data extracted. The analysis will be done across a number of scenarios to compare the data quality. Following an assessment, a data source or sources will be chosen in order to perform the analysis of the scenarios. The use of visual graphing or flow analysis will be attempted to present the data in the most logical way. The thesis is not to prototype a new system. Sections of code may be used in stages produce the end product. The findings may become the basis for a prototype structure.

3.3. Test Scenarios

The following section describes the testing phase methods, set-up, and scenario results. These include a detailing the services used, hardware testing and protocol descriptions.

3.3.1. Hardware and Software

The testing was conducted on a MacBook Pro (13-inch Mid 2012), 2.9 GHz Intel core i7, 16GB Ram, and a 1TB SSD drive. Operating system Mac OS Sierra 10.12.6. Software includes Atom code editor 1.24.0, Google Chrome 64.0.32.82.167, Firefox Quantum 58.0.2. Virtual box was used to create virtual machines for testing, Version 5.0.11. Python was used for the programming language and is Version 2.7.14, this was chosen due to a number of useful libraries available.

The virtual machines used for testing and hosting the Ethereum blockchain nodes had installed Linux Ubuntu 16.04 LTS (Long Term Support), this operating system was selected due to general support across a number of blockchain and Ethereum projects.

In order to establish the data quality, the testing scenario will allow an opportunity to review what is known to compare against the represented post-event data. As the study is focused on the Ethereum network it is imperative that the transaction from a user account transferring Ethereum or ETH is broken down and fully understood. As covered previously in the Literature review, a number of values are essential to conducting the transaction. A transaction in Ethereum scenario *i* sees Ethereum moving from Account *A* to Account *B* on the Ropsten Testnet see Table 1. This simple transaction allows a view from the tools and services.

3.3.2. Ropsten Testnet

The transaction was deployed on the Ropsten testnet. The Ropsten testnet is one of a number of testnets for Ethereum Technologies. These can be summarised as Kovan, Rinkeby and Ropsten. Ropsten was chosen due to the fact that it is the closest to the production environment. Ropsten has other advanced implementations for testing network conditions such as the Swarm protocol compatibility [21]. As Ropsten provides

FROM	TO	TXHASH	GMT	DATE
0x00C5Df3caC1794a407b350b129ABF456382c2e8a	0x0001f872567B5764709386a9Bea4F6C66cF8f8Cc7	0x0f7edb867487dfbd83e3b41f327b5ad290fb23198ab883d91e91e0010b6c8802	8201	24/02/18

Table 1: Ethereum Transaction

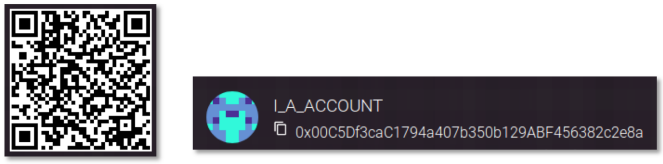

Figure 4: QR code, blocky identicon and 20-byte account hash

the closest environment to the production Ethereum network this was chosen for all the testing unless otherwise specified.

3.3.3. Web interface and Ethereum Client

The client, in addition to the network chosen to operate the Client, provides another choice for use. There are a number of choices for implementation, Parity provides an interface for advanced wallet management, Dapp support, and contract deployment. Parity also has the ability to be used in *Geth* the go-ethereum mode allowing the MIST browser and other functions to be tested. This was used in order to reduce the number of times systems would be required to sync to the respective blockchain [22]. A number of clients and different services will be documented in this paper to confirm what system was used to conduct transactions. The default if not specifically noted will be Parity Version 1.9.3. The default "GAS" and associated gas prices are used unless specified as changed in a transaction.

4. Scenario and Investigation

This section outlines the setup of the scenario.

4.1. Ethereum "ETH" Transaction

A transfer between an externally owned account to another externally owned account is conducted to send an amount of Ethereum on the Ropsten testnet. The user can use a number of methods to send Ethereum from a wallet to another wallet there are web-based applications or software solutions. A wallet is represented by a 20-byte address in our example this is *0x00C5Df3caC1794a407b350b129ABF456382c2e8a* the wallet can also be displayed as a QR code. Accounts can be represented in a colourful blocky identicon that is built from the hash of the address in some account services such as Parity, Mist, and Meta-Mask Figure 4 shows the QR code and a blocky identicon with an address.

The user selects the account to send funds from and then keys the address, cut & pastes or scans the QR code for the receiving account. The amount is required to be entered alongside the Gas values as previously discussed. The signature process differs depending on the wallet and enabled security steps, often a user enters a password to sign or confirm the transaction. Once

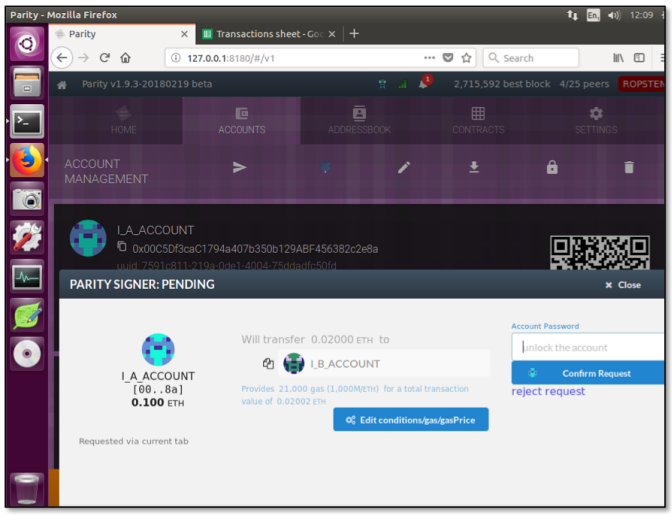

Figure 5: Parity web client interface used to perform transaction

Figure 6: Raw transaction as displayed on Etherscan

the transaction is signed and the transaction is processed as described in Section 3.1 it is processed by the Ethereum Virtual Machine and transmitted to the network. Figure 5 shows the wallet capability and shows the signing process.

At this point, the transaction is sent to the network for transmission to the miners using the proof of work (POW) algorithm at the time of writing. Proof of stake (POS) is due in the future on as the new consensus implementation. The mechanics of a transaction include the signature and the RLP (recursive length prefix) raw hash creating the 256-byte transaction hash TXHASH [23]. The transaction is sent from the wallet and its hosted connection to the blockchain and propagated to the other nodes to be transmitted to the miners [24]. The mined block then creates the new system state and the transaction is recorded in the blockchain and adds a new block to the Merkle tree [25].

The RAW transaction hex is formatted using the data for Nonce, gasprice, startgas, to, value, data, the (v, r and s values that make the ECDSA signature). The fields are then serialised into the RLP format (Recursive Length Prefix). It is possible to reverse the RAW format using a decoder or de-serialising the transaction [26][27].

The raw transaction hex is displayed in Figure 6, the component parts are displayed in the other returns for data, decoding so was unnecessary for this thesis but could feature as relevant if changes occur to the API returns.

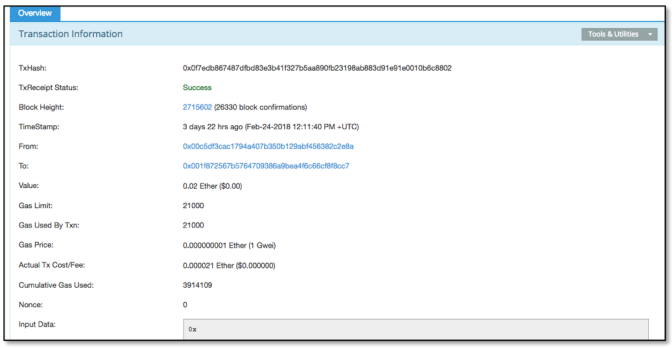

Figure 7: Etherscan web return for the transaction

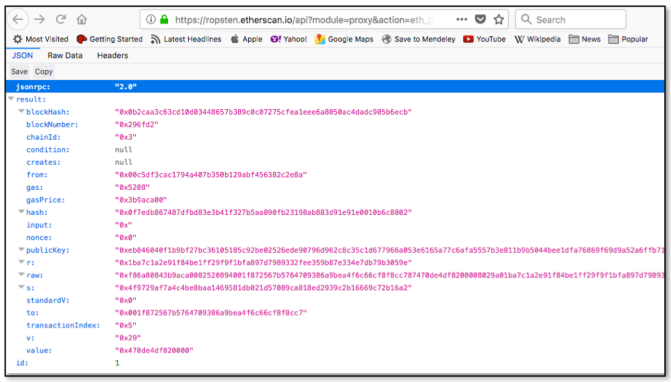

Figure 8: JSON return value pairs using the Etherscan API

```
web3.eth.getTransaction('0x0f7edb867487dfbd
```

Figure 9: Parity web3 shows the command

Etherscan web services are one of the most popular Ethereum block explorers and it enables users to search and view transactions and network information. Etherscan provides a web interface to search for transactions by hash, account, block, token or ENS. Etherscan provides a number of services including an ENS lookup and token tracking service. Etherscan provides an API (Application Programming Interface) that requires an API key and provides limited to access to its services. Figure 7 shows a typical web return for a transaction.

The API is called by formatting the URL to the specification as shown on the developer section of the Etherscan developer web page. The command request in requests is the transaction by the transaction hash and requires a valid API key:

"https://api.etherscan.io/api?
module=transaction&action=getstatus
&txhash=0x15f8e5ea1079d9a0bb04a4c58a
e5fe7654b5b2b4463375ff7ffb490aa0032f3a
&apikey=YourApiKeyToken"

Figure 8 shows the example given [28]. The API call then returns a JSON (Javascript object notation) encoded a set of values, Firefox can view the JSON return in a human-readable form as shown in Figure 8.

The Ethereum blockchain runs on hosts distributed globally on web.js a Javascript set of libraries that enables users to interact with the Ethereum system. The connection is available via HTTP or IPC and allows queries to be made to the system. Console commands are available using Geth console or in the example shown below the command was executed on the Parity Web3 console see Figure 9.

The commands are listed in the web3.js readme documents and allow a call to be made using the command line or programming languages to build Dapps and other web or software interfaces for Ethereum [29]. The command returns the same JSON response the only noticeable difference been the formatting of the value response.

At this stage Blockseer is not covered due to it not supporting the Ropsten testing environment, the service will be revisited when a test is carried out during the live scenario.

4.1.1. Internal Transaction

The Ethereum blockchain also has a class of transaction called the internal transaction. This transaction is where a contract creates a call from within a contract to create a "message" similar to a transaction where an action is instigated. A common use is sending of "ETH" to an account holding a contract that following a set of conditions sends a CALL that instigates the return of a number of tokens. It is possible to then list these internal transactions alongside the transaction list. This clearly is important when dealing with fraud and criminality as these may conceal large amounts of value. On the first inspection, it may be difficult to interpret. Contracts are complex in their nature as they a fully Turin complete. A contract could be designed specifically to obfuscate or frustrate and conceal activity. There are a number of OPCODES (operation codes) that are used to operate on the EVM and they are documented in the appendix of the "Yellow Paper" alongside the GAS costs [9].

Internal transactions triggered in a contract are ultimately triggered by the behest of an externally owned account. These occur either directly as a start instruction that triggers the CALL or when set conditions are reached following the initial transaction posted by an external account. Messages don't require the GasLimit setting in the transaction, this is already assumed as calculated to perform all computational steps and subsequent calls. Failure on GAS price or conditions results in the reverting of the internal transaction state.

4.1.2. Contracts and Encoding

The ABI Application binary interface is used to standardise the lengths and encodings of the input and how information is sent during a message of a transaction. The specification is held with the Solidity documentation and sets out with examples the format required. As this ABI input information is highly valuable to an investigation and the flow of understanding of the accounts activities the following test script was created.

Ethereum uses the SHA3-256 Keccak hashing as the standard hashing for encoding [30]. In the ABI documents, it in-

Figure 10: Full Keccak hash of the ascii “foo(uint32,bool)”

Figure 11: Input to the conversion scripy

dicates that the first four bytes of the input are the Keccak hash of the function operation. If a function is created as *foo(uint32,bool)* then the Keccak hash of the ASCII text of the function is calculated shown in Figure 10.

The *Method ID* is then calculated as the first four bytes as 0xcc822237. Confirmation of the manual calculation of the text is shown in Figure 11 as the input and Figure 12 as the output of the SHA-3 Python script. Figure 13 shows the Python script used.

The ability to interact with contracts via the ABI and Ethereum network is still in rapid development along with the Solidity language. There have been numerous coding bugs or contract logic anomalies that have resulted in significant losses, most notably the DAO and Parity [31]. The scope of this paper is not a deep dive into contract specifics and function. It should always be in a investigators mind that a complex layer lies beneath the veneer of transactional movements. Coding exploits can sit within innocuous transactions such as using CALLDATALOAD and CALLDATASIZE arguments holding additional instruction [32]. These transactions are contracts operating on the EVM a world computer, computational attacks can be present in the software. Logic in contracts and memory flaws may also exist.

The input data field, however, contains information that is useful to review and in general terms, it can be split into the following format. The hex identifier *0x* is followed by the method ID four bytes then following are increments of 32 bytes lengths depending on the function requirement. The 32 bytes can contain a number of differently constructed values as per the ABI specification, these are often padded to fit the 32-byte structure:

```
0xa5643bf20000000000000000000000000000000000000000000000000000000000000060
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
a000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

Figure 12: Output from the SHA3 python script conversion

```
import sha3
input_str = raw_input('Enter string : ')

encodings = {}

var = sha3.keccak_256()
var.update((input_str))
encodings["Ascii to sha3 Keccak 256 ....>  "] = var.hexdigest()

print encodings
```

Figure 13: Python script to encode to Keccak 256

```
0000000000000000000000000000000000000000000000000000000000000004
Hexidecimal to ASCII ....>

hexidecimal to Decimal ....>  4

6461766500000000000000000000000000000000000000000000000000000000
Hexidecimal to ASCII ....>  dave

hexidecimal to Decimal ....>  4540348615263367678604353243915104957
```

Figure 14: Partial return of the 32-byte section as a ASCII and decimal value

```
0000000000000000000000000000000000000000000000000000000000000003
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000100000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0200000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000003
```

An additional script inputsplit.py was created in order to obtain the *Method ID* then drop the 32-byte values into separate strings, before reading them as hex or ASCII to output. This is a crude method but can be refined as required.

From the ABI specification examples, the longer ABI is listed above, this is run through inputsplit.py [33]. The four-byte Keccak SHA3 is *0xa5643bf2* from the hash of the function ASCII value. This is crude to demonstrate the proof of concept and can be used where the method is known to build specific templates in response to the returned Method ID. Figure 14 shows a partial return of the console to demonstrate the readability of the response. This enables the decoding in this example to read the decimal value of 4 and the ASCII value *dave* from the encoding in the input fields.

4.2. ERC20 Token

The ERC20 was developed and came from an Ethereum Improvement Proposal a mechanism used by the Ethereum team to allow community and developers to put forward new features on the Github system [34]. The catalyst for the standardisation of contracts on the Ethereum network came from the founder Vitalik Buterin in his post on the Ethereum wiki that sets out standardised API contracts [35]. The ERC or Ethereum Request for Comment is a standard adopted from IETF Internet Engineering Task Force and the ISOC Internet Society. It is simply a standard way to introduce change to the Ethereum protocol in a regulated method of obtaining consensus with engineers, developers and the community. The ERC20 represents a standard format used by developers for a contract built on the

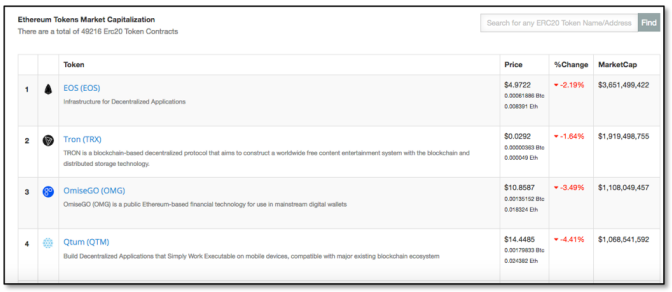

Figure 15: ERC20 token market cap from Etherscan

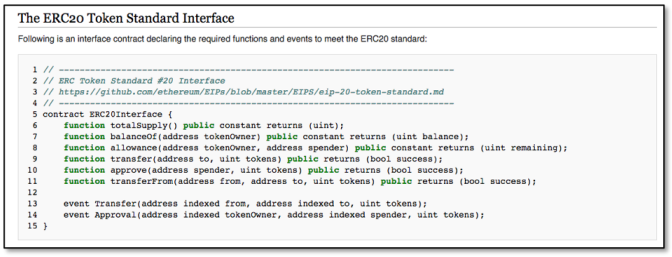

Figure 16: Displays the six functions and two events for the ERC20 token standard

Ethereum network to represent a token, the 20 is a numerical reference to serial the EIP reference number, therefore, has no other meaning. The token standard enables developers to reuse code and enables other services and exchanges to support the token without having to redesign new standards and adoption for every new coin. The ability to create tokens and be listed in an exchange or held in a supported ERC20 compliant wallet, therefore, strengthens the uptake and usefulness of the token and overall eco-system. The standard also enables confidence with developers that the smart contract logic is sound by using battle-tested code. The ERC20 was introduced in 2015 and is common in the ICO (Initial Coin Offering) boom that followed the adoption of Ethereum.

There are 49,216 different ERC20 tokens listed and the top three coins market caps are worth billions of U.S dollars as shown in Figure 15[36]. The ERC20 token can be used in any manner of tokenised system and these include tokens of value, financial derivatives, cloud computing, power grid, gambling, storage and many others. The coin, therefore, has a number of specifically allowed contract items. The full final specification is documented as EIP20/ERC20 /citeVogelsteller2015EIPs/eip-20.mdGitHub.

Naming conventions from the standard API format. Table 2 shows the convention and an example. These formats are important to define the use of the discovered code.

In Table 3 the functions are written in the format required and detail the input variables and show the response formats. It is possible also to see how the Keccak method ID is created by the hashing of the function. It is possible to create a list of signature Method IDs similar to the previous internal section. The token standard is documented by Codetract and it is useful to compare the results as shown in Table 3, the calculations are made from the short form of the function and hashed with the

```
https://api.etherscan.io/api?module=logs&action=getLogs
&fromBlock=379224
&toBlock=latest
&address=0x33990122638b9132ca29c723bdf037f1a891a70c
&topic0=0xf63780e752c6a54a94fc52715dbc5518a3b4c3c2833d301a204226548a2a8545
&apikey=YourApiKeyToken
```

Figure 17: Etherscan API developers example and formatting for Event logs

SHA3 Keccak as previously shown with the python script [37]. The short form is derived from the longer form shown in the Ethereum wiki.

Longform as:

```
Approval(address indexed \_owner,
address indexed \_spender, uint256 \_value)
```

Shorter form:

```
Approval(address ,address ,uint256)
```

Recognising an ERC20 event in a post-event investigation can be broken down into a couple of features. Once retrieving a transaction list for an account using the Etherscan API a look at the transaction value is required. The transaction value is calculated in Gwei an ETH subdivision so a valid transaction of ETH must contain a value above zero. If a transaction has a zero it indicates a potential to be a transaction of a token as it has been returned to the transaction list and not an internal contract transaction. Further examination is required and the input field should be the next area to analyse.

There is no direct ability to obtain token events within a simple query it requires a deeper programmed search and chain of search. The Etherscan API, however, provides an event log search. This search is formatted as Figure 17 shows from the Etherscan Developers API [28].

In order to query the transfers, it is required to format the API request with the correct filter data this is done using the documentation and additional posts in the stack exchange forum [38].

The block “to” and “from” value is to be entered and is limited to 1000 return so pagination will be required above those limits. The address field requires the contract address for the token, each token is essentially a contact using the 20byte address, however, the address contains contract data, unlike a standard externally owned address. As in this example, the return required is a token transfer event it is required to enter Topic 0 with the Keccak SHA3 of the event:

```
0xddf252ad1be2c89b69c2b068fc378da
a952ba7f163c4a11628f55a4df523b3ef
```

Topic 1 is the “from” or source address of the token transfer and requires to be padded to 32 bytes. Topic 2 is the “to” or destination address and again this requires to be 32 bytes padded. Topic 1 and 2 are both an optional variable and depending on the requirement of the search they can be omitted:

- Obtain the transaction list and find zero value transactions
- Check the address for contract data and set as contract address

TYPE	CONVENTION	EXAMPLE
Function	Camel case	camelCase
Event	Upper camel case	CamelCase
Input variables	Underscore prefixed lower Camel case	_camelCase
Output variable	Underscore lowercase “r”	_r
Success	Boolean	True “1” or False “0”
Address	Underscore lowercase	“addr” _addr
Specific addr	Underscore lowercase	_from or _to

Table 2: Naming convention for function and variables

TYPE	CODE	STANDARD	METHOD ID
FUNCTION	name()	ERC20 – optional	0x06fdde03
FUNCTION	symbol()	ERC20 – optional	0x95d89b41
FUNCTION	decimals()	ERC20 – optional	0x313ce567
FUNCTION	totalSupply()	ERC20 – Standard	0x18160ddd
FUNCTION	balanceOf(address)	ERC20 – Standard	0x70a08231
FUNCTION	transfer(address,uint256)	ERC20 – Standard	0xa9059cbb
FUNCTION	transferFrom(address,address,uint256)	ERC20 – Standard	0x23b872dd
FUNCTION	approve(address,uint256)	ERC20 – Standard	0x095ea7b3
FUNCTION	allowance(address,address)	ERC20 – Standard	0xdd62ed3e
EVENT	Transfer(address,address,uint256)	ERC20 – Standard	0xddf252ad
EVENT	Approval(address,address,uint256)	ERC20 – Standard	0x8c5be1e5

Table 3: ERC20 optional and standard function / event codes

- Pad out the “to” and “from” address to 32 bytes
- Search Topic 0 with the token transfer events
- Search with Topic 1 & 2 for events
- Search only “from” address and “to” address
-

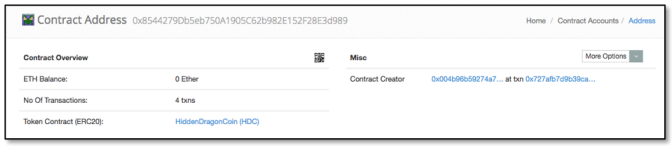

Figure 18: Contract details for ERC20 token HiddenDragonCoin

Figure 19: Part of the Solidity code for the ERC20 token HiddenDragonCoin

4.2.1. ECR20 Coin creations

An ERC20 compliant code is the set of standards set out previously. A coin is created by the creation of a contract using a compliant, most commonly solidity code. There are numerous examples and tutorials to create a token and the Ethereum foundation have instructions and code on their website to follow [39]. A solidity script used coin.sol to create an ERC20 token called “HiddenDragonCoin (HDC)” Figure 19 shows a screenshot of the solidity code. The coin is a contract that is deployed to the network by an externally owned account, this is important if the coin itself is the focus of an investigation. The transaction that creates the contract has the contract address field entered as the new contract address and this also contains a large data field that contains the code of the new coin. Figure 18 shows the data on Etherscan, note the contract creator and the transaction ID when it was created, this enables attribution to other accounts.

To test the method the developed scripts were used to collect the data from the contract creator account and parse for token events. The script documented the events and gives a view of the token movements and contract interaction. Using the kevalscsv.py script it takes the data from the extracted JSON files from the Etherscan API and creates a CSV file, one field is the input Method ID. This enables a quick view to discover the movements and additional ERC20 transactions using the codes in Table 3. This was validated using Parity client’s app GAV coin (Gav Wood coin) this was done to ensure the method worked on different clients and virtual machines. The tracking of ERC20 events can be tracked accurately using this method, verified by a number of different token movements using different clients.

4.3. Key Values

In previous sections, there are a number of returns by the services either directly querying the Ethereum network on the console or by the web portals. A number of Key values are identified for their beneficial nature in the investigative process. The transaction list function is one of the primary sources of data for the purpose of transactional flow. The following returns are from the Etherscan API they are a JSON key and value return.

- 'isError' Relates directly to the response of the API request to the system this is a Boolean, True "1" or False "0" the value for a successful transaction is "0" no error occurred. Etherscan API only.
- 'timeStamp' The time stamp is a UNIX timestamp at the point the transaction is conducted. Displayed as a 10-digit decimal representation.
- 'nonce' The nonce value is an important incremental serial number assigned to every sending "from" transaction. This is used by the Ethereum network to detect bad nodes and double spend attacks. Alongside the timestamp, this number will assist in the temporal information and create accurate event reconstruction. Returns a hexadecimal value.
- 'from' This is the sending or source address of the transaction, this is the 20-byte address format.
- 'to' This is the receiving or destination address of the transaction, this is the 20-byte address format. The field can be blank if it relates to contract creation. (The input field would contain additional information that will confirm this).
- 'hash' The Transaction TXHASH is a 32byte hash and is essential in identifying a specific transaction. This is created as a hash of the transaction detail and signature process (Codetract, 2017).
- 'value' The value field is a decimal number that represents the value in "Wei".
- 'input' This field contains the ability to attach data for use with the transaction. This will have data or "0x" as an empty field. This field requires analysis when data is present. This will indicate the method event/function conducted and give values that allow for location of new transactions or logs.
- 'gas' The amount of gas provided by the sender to conduct the transaction. This is the amount to cover the gasPrice x number of steps computed on the EVM.
- 'gasPrice' The gas as calculated in "Wei" price of gas. This is dependent on network congestion the price for a slow or quick transaction is varied.
- 'gasUsed' The amount of gas used for the transaction, this is the gas steps multiplied by the gas price.

TRANSACTION DATA	*RETURN TYPE*				
FIELD	DATA RETURNED FORMAT	Etherscan JSON API via PY	Etherscan Web Transaction info	Parity web3.eth.getTrans	Blockseer
BlockHash	32 Byte hash	x		x	
blockNumber	Hexadecimal value / Decimal	x		x	
chainId	Hexadecimal value	x		x	
condition	null	x		x	
creates	null	x		x	
from	20-byte address	x	x	x	x
to	20-byte address / Null contract creation	x	x	x	x
gas	Hexadecimal value	x	x	x	
gasPrice	Hexadecimal value	x		*	
hash	32 Byte hash	x	x	x	x
input	Input data provided to transaction	x	x	x	
nonce	Hexadecimal value / Decimal	x	x	x	
publicKey		x		x	
r		x		x	
s		x		x	
v		x		x	
value	Hexadecimal value in wei	x	x	*	x
raw		x		x	
TxRecieptStatus			x		
TimeStamp			x		x
ActualTx Cost			x		
Cumlative Gas Used			x		x
BlockHeight			x		x

Figure 20: Comparison of data from services

- 'blockHash' Each Ethereum mined block contains a 32byte hash address this relates to the block that the transaction is contained within. This enables confirmation of transactional data.
- 'blockNumber' The block is also given a block reference number. This number is a decimal number.
- 'confirmations' This field relates to a number of confirmations the block has received from miners. It is worth ensuring the number is above a minimum of 30.
- 'contractAddress' The contract address follows the same format as an externally held account as they are considered equal. This is a 20byte address.
- 'transactionIndex' Index hexadecimal of position of the transaction in the block.
- 'txreceipt_status' A Boolean of "1" for True a receipt exists in the receipt trie root or "0" for False a receipt is not recorded in the trie.

Figure 20 Shows a table of the Key values returned from the services.

To provide a system to capture the information and create the following basis for the proof of concept a number of python scripts were developed. These included Ethereum capture, incorporating Balance, transaction list, internal transactions, and receipts. These searched for data on the Main Ethereum and Ropsten network, this was by account, transaction hash or block number and index number Figure 21 shows the menu screen. This was conducted by constructing arguments for the API and then serving and retrieving the data in a JSON returned file. The system was a proof of concept and was used to test to validate the concepts and methods. The software is not designed to run optimal code or even load data to a database that would be required for any production-ready tool.

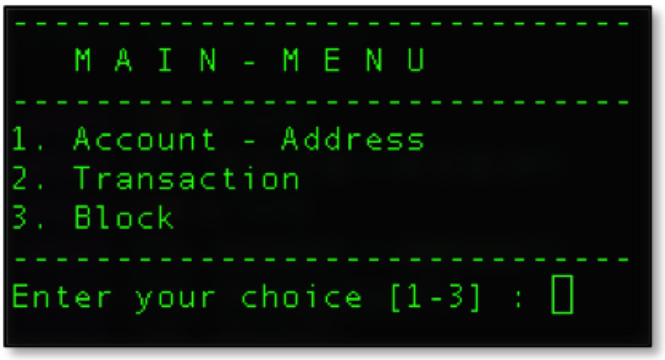

Figure 21: Main menu for the created script

A script was designed for Token events, using the returned data the script parses the retrieved transactions and obtains the key values including the Method ID. The script takes an event ID in Topic 0 that details a token transaction before correctly requesting the relevant Event log that includes all the required token to and from data when using the correct data. Smaller scripts were used for some basic formatting:

- Key value extraction, the script obtains identified key values required for accurate tracking and outputs to JSON and CSV for use in other processes.
- Conversion a number of smaller scripts to enable testing include conversion from text to hexadecimal. A number of SHA3 Keccak conversions for formatting and simple graphing scripts. Graphing became a deep subject in itself so a basic construction was used for proof of concept.
- Inputsplit to take inputs to format to the correct length.

4.4. Blockchain investigation

4.4.1. Ropsten network tracking

An investigator presented with large amounts of textual and numeric information is required to construct a method to conceptualise the event or movement of the token value. The capture of the raw transaction data is formed into a graphical representation of a node and edge graph. This enables a visual flow that will assist the investigator to find the network nodes and edges.

In the scenarios laid out the movements described below can be viewed in a graphic view and it is possible to understand the flow and find the start and end points. Ethereum "ETH":

1. A to B
2. A to B to A
3. A to B to C
4. A to B to C + D
5. A to B + C and B + C to A
6. A to B to C to D to E
7. I to F to E to D to H to G to C to B to A
8. A to B + C + D and B to E and C to F and D to G and E to H and F to H and G to H

timeStamp	from	to	hash	gas	gasPrice	No of ETH
1519474300	0x00c5...2e8a	0x001f...8cc7	0x0f7e...8802	21000	1000000000	0.02

Figure 22: Transaction completed on the Parity client / wallet

Figure 23: Transaction completed on the Parity client / wallet

Figure 22 displays the scenario (i) where the transactions A to B was conducted using the Parity client to send a transaction to the Ropsten network. The table is stripped down and the hashes edited for presentation. Figure 23 shows a transaction been conducted on the network using the Parity client.

Figure 24 shows the movement patterns and the start and end points moving between accounts. The graph is a directional node and edge graph. Each letter represents an account entity. The graph pattern assists the viewer to see the direction and movement of the funds in the scenarios set out earlier. It is possible to view the resting points or edges (endpoints). Endpoints can be an exchange, ATM, merchant or a mixing service. Node numbering can assist in the identification of large exchanges or mixers. Nodes in frequent use will have a node number in the thousands, open source research will more than likely identify the service or owner.

4.4.2. Scenario tracking live network

The developed script was used to acquire data from the Etherscan API. The target account was entered and the script downloaded all the transactions on the targeted account this includes balance, transactions and internal transactions. To follow the money the subsequent "to" address would require downloading to see the next node in the chain. This should be done until it is at rest in an account (holding account) or reaches an edge, such as an exchange (out/edge). Inputs (in/edge) or "from" values to the target account will need to be applied manually to ensure they are relevant, this is an intelligence-led subjective decision. An additional script is developed to extract the token transactions and then create an event API call to request the log for the token transfer, this is important as these transfers can hide value transformation. An additional script was created to take the returned JSON files and create a CSV file for mapping and graphing purposes. In order to visualise the transactions, the use of the scripts enables the ability to view the directional flow of the value tokens. The "keyvalues" are identified as above and extracted in the scripts to build the .csv. The following directional graphs were generated from the data and additional annotation given to enable discussion.

Figure 25 shows the ETH only transactions and this gives an idea of some of the interactions of value transfers. Figure 26 shows the token only transfer, this enables the investigator to view value transfers of different tokens. If the additional tokens are not tracked the value can be transferred and cashed out of

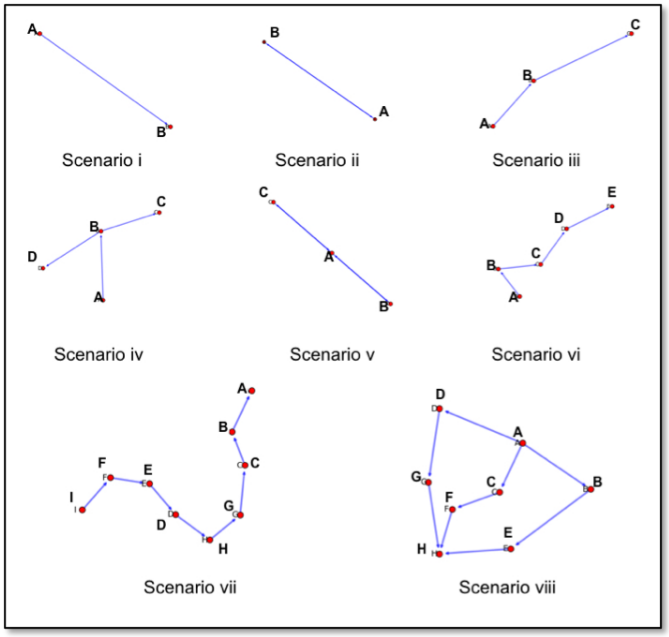

Figure 24: Graph structure and flow between the accounts

the system.

The transactions took place sending Eth initially from the Kraken exchange across the two subject accounts "0x63a0...bb51" and "0xc9b1...33bc". Shapeshift is a service that allows the exchange of cryptocurrency from one token to another. Jaxx has an in-built Shapeshift function that allows the swap of funds from one form to another. Shapeshift was additionally used in the Jaxx wallet to transfer ETH to Golem GNT and Augur REP tokens. In another process, the web version of shapeshift was used to transfer Golem into REP. The remaining ETH and REP tokens were then sent back to the Kraken exchange. It is possible to link the services of Kraken to these interactions due to how they receive funds into a specific wallet before using a contract to retrieve the funds back into Krakens main account.

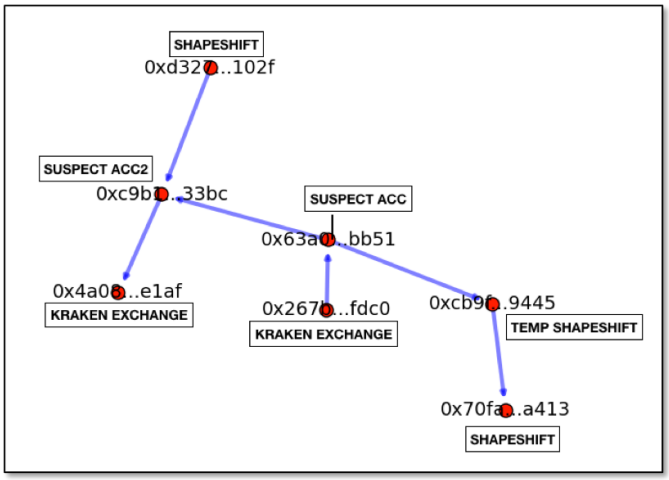

Figure 25: ETH transactions on the live network

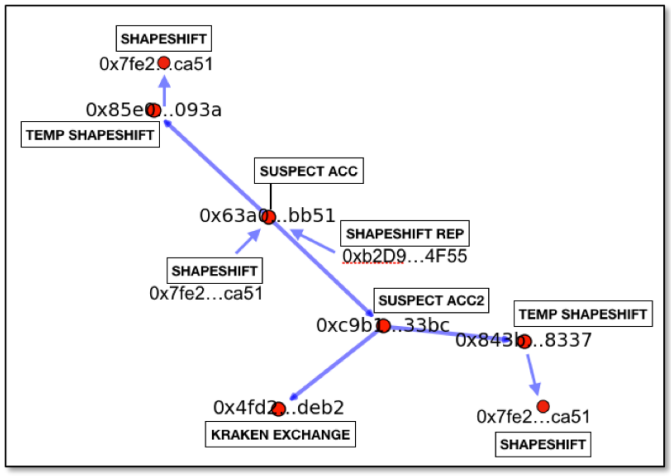

Figure 26: Token GNT and REP on the live network

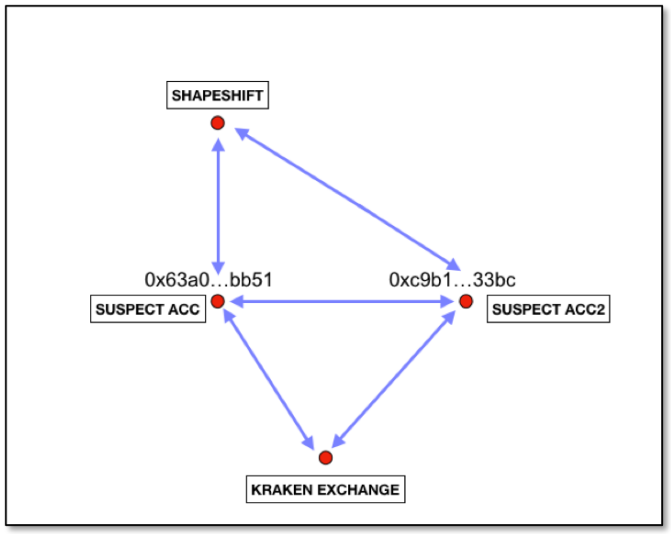

Figure 27: Simplified ETH, GNT and REP cluster addressing

Figure 27 shows a simplified version where the clustering of addresses is possible via metadata collection. Etherscan shows account names where associations are made. This view enables an approach to the exchange Kraken or to Shapeshift for information on customer data where available. Kraken will as a full exchange retain KYC information that is likely to be available for law enforcement or civil litigation where the correct court instrumentation is presented.

The use of the created scripts worked in the collection of the data to build a comprehensive record of ethereum transactions. The obtaining of the event logs is required to log the token movements. One fault observed was a transaction of one of the initial token events was missed, the system is programmed to scan the transaction list and detect a token event. In this example, the original transaction list did not contain a transaction for the transfer, so the subsequent script search would not detect the token event. In this example, the Topic1 is not known.

A wider range of event scanning would find all events if the Topic 2 the receiving address was displayed with no source ad-

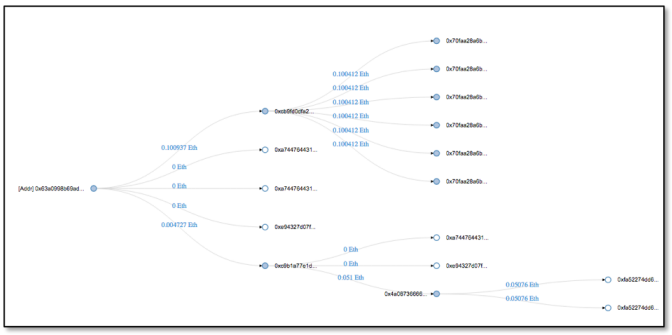

Figure 28: Beta visualisation in Etherscan

dress, as this was unknown in this case. This would need to be tested alongside other permutations for blanket trawls for source and destination token events without specifying the contract address or opposite topic. In essence search all token transfers "from" this account, then re-run all token transfers "to" this account. The formula could be applied to detect all of a certain function such bid in ENS address for example. Another fault discovered was the decoding of the value from Wei to a decimal of Eth in Keyvals.py, the value was calculated as an integer. An integer in Python is unable to calculate decimal points so a floating point is used to correctly display the values see Appendix A.

In the live net test, the ethereum in the account was sent to another account while tokens remained in the wallet. This, however, meant the account was devoid of an eth and gas balance, without gas the ability to ask the EVM to perform any action is missing. This was a mistake during testing but highlighted a common mistake. The tokens are stranded until some gas is added to the account to move the tokens elsewhere. An important measure could be where tokens exist on an account and where the balance drops to 0. A subsequent small deposit is issued prior to movement of the tokens. This may indicate ownership and attribution that can highlight linked or associated accounts.

Etherscan web contains very good textual data and the visual transaction is promising. It does not show token transfers as such and there are some anomalies in the above graphing Figure 28.

4.4.3. BlockSeer

Figure 29 shows the Blockseer application graphing showing auto detection of the Kraken exchange. There is still a requirement for the investigator to expand the sections choosing what to expand. If the investigator expands all transactions in the exchanges the graph will expand to thousands of transactions. The graphing is mapped well in Blockseer but there is still a requirement for the investigator to make associations and research on accounts despite the automation available.

4.5. Contract Methods

4.5.1. ERC20 and Decentralised Exchange (DEX)

As identified in the above sections the use of Method ID and contract formatting enables the EVM to understand what

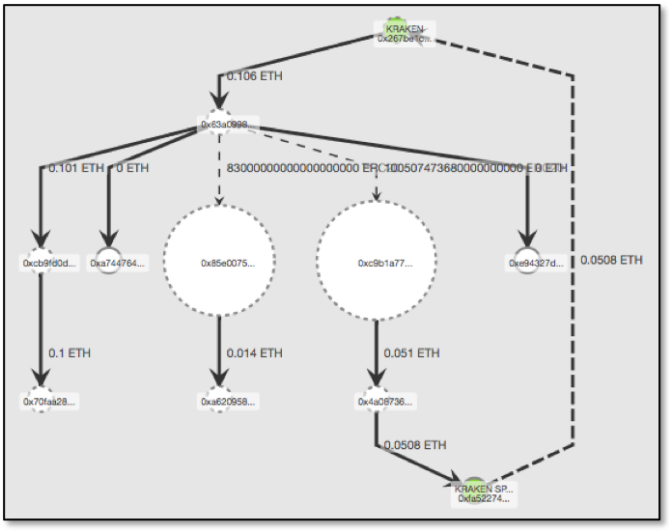

Figure 29: Graph from Blockseer

```
Function: trade(address src, uint256 srcAmount, address dest, address destAddress, uint256 maxDestAmount,
uint256 minConversionRate, address walletId) ***

MethodID: 0xcb3c28c7
```

Figure 30: Decentralised trade Method ID and variables

is been asked of it. This however also enables investigators an opportunity to scan for the Topic and create a Method ID signature to allow an understanding of what functions are been performed. In the section ERC20, the method ID detected the function for a transfer and then on viewing the proceeding event the full hash was viewable with the detail of "to", "from" and values. This use of a *signature* can be applied to detect items of interest in a number of contracts or protocols on Ethereum. An example would be to detect a trade on a decentralised exchange using the MethodID 0xcb3c28c7 as Figure 30 shows an example from Etherscan. The trade is found and the topics can be parsed to obtain the events and see the additional functions or events. The below example will show the source and destination address with the associated values. This enables a graph and flow and if the funds in the live job had gone to a DEX decentralised exchange it is very unlikely the trade would be discovered and shown using this method.

4.5.2. Beyond ERC20, Cryptokitties

To extend the importance of this MethodID and subsequent investigation logic it can be applied to the movement of other value assets on the network. Cryptokitties was an Ethereum based digital collectible game and trading platform. This very popular game highlights how a non-traditional value asset can be traded for profit [40]. Appendix C shows part of the ERC721 contract with available function and events allowing for method ID signature creation. The bidding on Cryptokitties shows as a bid in the internal transaction show in Figure 31 from Etherscan. Following the auction, the events show an "AuctionSucessful" and subsequent token transfer using the same method ID transfer for movement as the ERC20 token. This

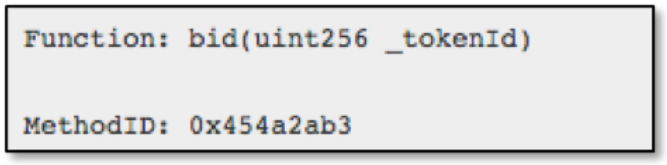

Figure 31: an internal transaction on Cryptokitties

{"address":"323238d07e24459551732aa51add69d4c8ce569d","crypto":{"cipher":"aes-128-ctr","ciphertext":"0222b304602c208679108e09799d31d4e0ee45c03591efff09f3ece13834d9dc","cipherparams":{"iv":"8b6efc14d2a8674bd1cd118967576fb6"},"kdf":"scrypt","kdfparams":{"dklen":32,"n":262144,"p":1,"r":8,"salt":"7720867e351348beba343b46afb708d98002cd352836618252782318 1e78ea91"},"mac":"cfddd0d41a3b919a9b8aca46625040517ea33114387cd71e1447de8a45a704ee"},"id":"efc6e100-6e99-4475-ab11-9636b3f10828","version":3}

Figure 32: JSON format

detail shows the "to" and "from" accounts and the account id in this instance. The token transfer hash:

`0xddf252ad1be2c89b69c2b068fc378da`
`a952ba7f163c4a11628f55a4df523b3ef`

This demonstrates the ability to add new transfer types as new token standards are brought into circulation. The rise of new digital shareable as a gamified collectible product will develop and the ecosystem will need to be followed [41]. The link between "skin" trading as a crime eco system for cybercrime can be found in gaming circles [42]. The ability to trade and share skins via smart contracts direct from games or marketplaces will be facilitated with ease on numerous up and coming networks [43][44]. In the future, the ability to virtualise or emulate a contract to logic test it for security is likely.

The ability to detect contract use for law enforcement could replicate a contract discovered and test its logic to verify the intent. An example would be what appears to be a gambling smart-contract between two players, however, player 1 always loses due to the contract obfuscated code logic. Player 1 is, however, a criminal who has arranged to pay his debts by the use of the gambling site in order to maintain a cover for criminal money laundering. The discovery would only be shown by deep analysis of code or if the replication and testing and could be attempted and run. A percentage is given on the number of win or loss results.

4.5.3. GAS

Gas price and the amount of gas assigned to ensure the transaction steps are all completed are required for all transactions. In the above live scenario testing, the following observations can be noted. Wallets do not have the same default settings for gas and gas price. This enables the potential for wallet or service identification. The Jaxx wallet has the default Gas as 25000 with a 2 Gwei Gas Price while Meta-Mask had Gas at 21000 and 1 Gwei Gas Price. The price paid from the other services and contracts are much more specific to real gas requirements, likely to the contract pre-calculating or live calculating of a current gas condition value. Defaults assist users to not have to check the current gas requirements for network congestion to ensure the transaction goes through, this was an issue during large ICOs and when Cryptokitties was at its peak [45].

Gas prices can be set too high in a wallet and although this only accounts for a small amount per transaction it is a legitimate concern [46]. Further research would need to establish if the default Gas settings are updated by the wallet when network usage is high or if the defaults are set per version of the wallet. Attribution could be possible if baselines for major wallets are kept against a timeline and if user's transactions mirror the baseline measurements this would indicate service use. It should be noted that a user can change the defaults and this itself may assist with attribution if a user reuses a non-standard value. Gas use may change with future versions of Ethereum to assist in wide-scale adoption.

5. Forensic Analysis

In this section, the use of forensic analysis is examined to perform a number of specific password attacks and key recovery. It looks at opportunities for investigators to discover accounts from seized machines. This section will cover common password formats and crack encryption. In addition, the discovery of word seeds and how to recover potential accounts linked to the seed words.

5.1. Password formats

The following format types were detected when investigating the password formats associated with the Ethereum clients used. The latest version of the Ethereum wallet used "Scrypt" while Parity used the previously implemented PBKDF-HMAC-SHA256. There are a number of blogs and resources that cover the cracking of the Ethereum passwords. The recovery would assist a forensic function to discover, or to seize funds from criminal's dependent on legislative support. The cracking of the password however may assist with asset recovery and discovering passwords for other services such as drive encryption. The keys are stored in the "keystore" and these differ slightly through the various operating systems see Table 4.

Hashcat was used in this experiment as it is a comprehensive password cracking tool with GPU support, contains a large implementation of methods and is widely written about [47]. The keystore is a .json file that contains numerous key value sets containing the cipher values and encryption detail. The Ethereum Github sets out the values and the minimum implementation and required settings [48]. In the example of the JSON file is given in Figure 32, and it shows a typical layout of key values.

In order to identify the encryption, scheme the "kdf" value displays "scrypt" or "pbkdf2" this indicates the scheme see Figure 32 for an example file. Alongside the AES 128-bit encryption, the use of counter code block cipher uses a stream cipher as also shown in the returned values. The ability to extract the hash is made easy using two Github python scripts developed for the job. The "ethereum2john.py" creates a suitable Hashcat output for the Scrypt variant [49]. The PBKDF version of the Ethereum key as used by Parity is documented in the

Operating System	Path
Mac	"/Users/username/Library/Ethereum/Keystore"
Windows	C:/Users/username/AppData/Roaming/Ethereum/Keystore
Linux	/.ethereum/keystore

Table 4: Operating system and paths

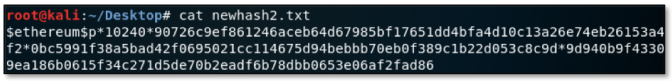

Figure 33: Ethereum wallets formatted keystore to "Scrypt" hash ready for cracking

Figure 34: Parity wallet keystore "PBKDF" hash made ready for cracking

"ether2hashcat.py" project that can extract the correct hash format for Hashcat [50].

The following keys were used in order to test the above-mentioned scripts to extract the hash and place into a Hashcat formatted command and crack the wallet password. An Ethereum wallet Version 0.9.3 was created by backing up the key. The format as shown in the blog article and replicated by the script orders the values into a format as shown below, these are inserted into a test file for use by Hashcat.

*ethereum*s*n*r*p*salt*ciphertext*mac

As shown in Figure 33 we can see the Scrypt version from the Ethereum Wallet and the PBKDF parity wallet values in Figure 34. The values were extracted using the Python scripts and saved to a text file.

Hashcat supports three different hash types for Ethereum wallet types. These are the Ethereum pre-sale wallet, Hashcat reference -16300, Ethereum Wallet – PBKDF-HMAC-SHA256, Hashcat reference – 15600 and Ethereum Wallet – Scrypt Hashcat reference – 15700. In Figure 35, a command is used to crack the hash of the Parity password. The command evokes Hashcat and uses –force as it used on a virtual machine and not utilising GPU processing the code for the hash type is used and the hash text file selected and a pass.txt containing six possible words. The cracking of the pre-sale wallet is not covered in this thesis as its use is limited but should be noted that Hashcat supports the cracking function.

The password cracking was done using a small number of passwords: six in this case. Figure 36 shows the returned result alongside the cracked password.

The Hashcat shows the password as *plantpot123*. This is the correct password and Hashcat is able to crack both the files us-

Figure 35: Hashcat command used to crack the Parity key hash

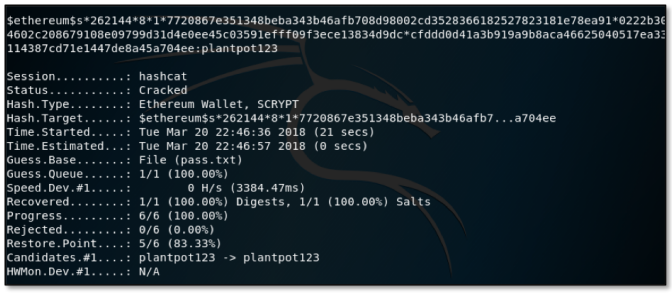

Figure 36: Cracked Ethereum Wallet return in Hashcat

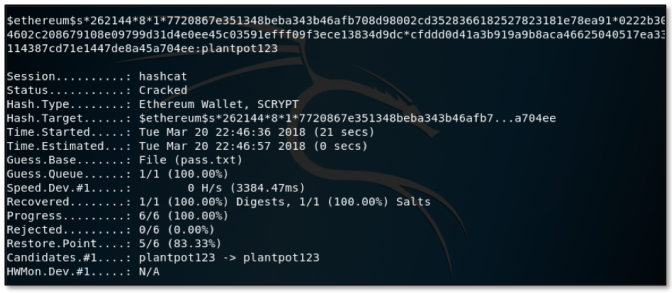

Figure 37: Jaxx local storage database viewed in the SQLite viewer

ing the methods described above.

5.2. *Recovery seeds*

In order to look at other applications and storage of keys and to assist in the study of the transactions across the network the Jaxx wallet was installed. This was to enable another view so the testing was used against the Desktop application on Windows 10. The version was Jaxx desktop Version 1.3.15. The wallet is a multi-platform supported wallet including web-based extensions for Chrome and Firefox and operation system support for Mac and Linux. There are also additional mobile versions of the wallet for Android and Apple OS. The wallet utilises the 12-word phrase systems that have developed from the cryptocurrency establishment under the BIP standards the word phase can restore the private key. A vulnerability is documented in a blog post that exposes the mnemonic word system to a decryption attack [51].

The Jaxx wallet was installed and running on the Windows 10 virtual machine. The location of the Local Storage of the application was found at the following location under "user1" the local profile containing the application. Appendix B documents other operating system locations. The database is a stored as an SQLite database and the database was loaded into DB Browser as shown in Figure 37.

```
qyp1D54khppYhq+Ae+bpB2b7jDnRmKrx7AxvZuxUDZX9RSBa61GfXFwBg8V8wur
Zf6tF+DArHB1/vG3wfscTWCiJsfkpDK+8qYfVuMYPDQg=
```

Figure 38: The exported stored string

```
wrap athlete turn dial royal sunset flush web joy powder lunar razor
```

Figure 39: Recovered word seed from the Jaxx Windows 10 desktop wallet

Cell 204 in Figure 37 is entitled mnemonic and this field contains the data required. At this point, an export to a text file then drops out the following text string Figure 38.

In order to decrypt this string into the 12 words seed a simple script is required and can be used to take the string and return the passphrase, required is the Cryptojs module. The script is shown in Appendix A and is used to validate the method. When run against the recovered mnemonic phrase from the Windows Jaxx wallet the recovered is shown as Figure 39, the private key for the Jaxx wallet.

This demonstrates the ability to be able to recover passwords from physically held machines or if an attacker has access to files on the system machine remotely.

5.3. Scanning for word seeds

In addition to the recovery from wallets using a password attack, the use of words seeds gives investigators an additional opportunity to recover criminal assets or identify targeted attacks. Scanning seized items or triaging at a scene specifically for word seeds will enable a new opportunity for account recovery and discovery. Ethereum although significantly different in its core infrastructure to Bitcoin still contains some remnants from its evolved history. The use of a similar standard for the wallet structure and protocol is still based on Bitcoin BIPs Bitcoin Improvement Proposal. The construction for Ethereum wallets are yet to be standardised but can be implemented using a number of proposals. Common implementations include a number of ongoing EIPs, these are EIP 600 and EIP 601 the discussion enables a view of differing priorities. Ethereum has however had developers using the BIP44 standard that itself was a development of BIP43. In addition to those, there are uses of BIP32 and BIP39 with different wallet developers.

The design of the Hierarchical Deterministic Wallets uses the seed to generate the key space for the wallet to create new public addressing as shown in Figure 40. This system is to enable privacy when returning money to a change address using a new account for each transaction was encouraged in Bitcoin for privacy. The use of change addressing and the reuse for privacy is down to the UTXO Unspent Transaction Output. The protocol is a key feature of Bitcoin and many cryptocurrencies that rely on a fork of the Bitcoin source code. The use of BIP39 Word Seeds allows the word seed to be the master key and to act as the master private key to unlock the numerous public addresses and subsequent wallets in the key space. The BIP39 specifies 2048 words that can be called to create the private key, there are rules specified that detail the initial entropy length and checksum (“bips/derivation.png at master · bitcoin/bips · GitHub,” 2013).

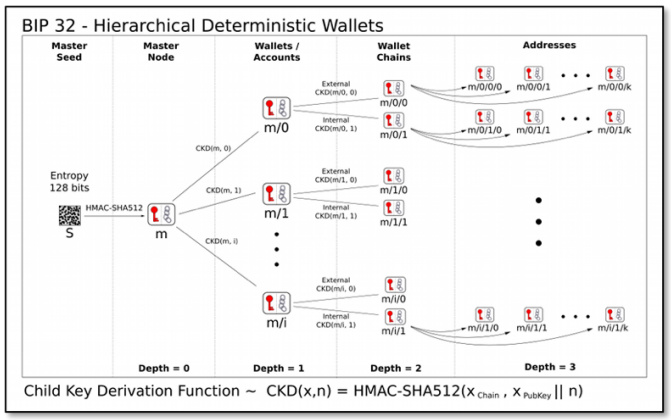

Figure 40: Shows the creation of a Hierarchical Deterministic Wallet

The key lists are available in a number of languages and detailed on the Github site listing all the accepted seed words [52]. The word seeds are 12 words in a UTF-8 encoding in the English language version using words between three and eight characters in length, this differs for implementations in different languages. There are additional numbers of words also used that will be covered later. The following points highlight a method to scan for word seeds and some unique features and checks that could determine if present:

- There are a set number of spaces between the words, excluding the before and after position, as these may contain quotations or another delimiter.
- The set of words is between 3 and 8 letters in length these can be scanned for any punctuation or special characters if none are contained this raises the possibility of a key phrase. The phrase should be used as a lower case but a user or wallet design may change the case to auto-correct sentence case or full upper case.
- Once discovered compare the words to the word list from the BIP 39 Github repository. This will give an additional check to reduce false positives.
- If the above is detected for a 9-seed word then expand by word increments 11, 12, 13, 24 to determine the number of words. (Ensure any additional other sized keys are found, historic and future.) Note an additional password can also be applied to the word seed.
- UTF-8 encoded words could be held in another encoding such as ASCII depending where and how recorded, they may be stored as a note or in a word document. Scan different encodings for the same method.
- Following a seed discovery populate the possible wallet pubic keys, it is possible to differentiate between Ethereum and Bitcoin addressing due to the encoding outputs. Scan for other currency [53][54]
- Scan the generated public addresses and owned wallets search the blockchain for activity, balance, and transactions. If a response is given the wallet is located for the

Figure 41: The 12-seed word from the Jaxx wallet

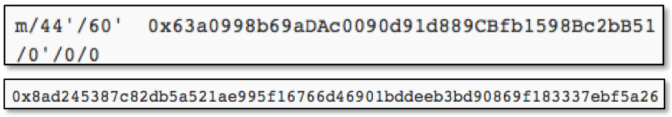

Figure 42: Public and private address returned by the mnemonic recovery

investigator, others can remain linked or marked for future events to attribute to the user.

- Scan for Bitcoin, Ethereum, Alt-coins for private and public keys across the recovered media / hard drive or ram capture.

As the Jaxx wallet used earlier has a 12-word seed following the decryption this enables the method to be tested and validated to search for the address of the wallet only using the seed phrase. This can be achieved using the BIP standards as the wallet is BIP44 compliant and uses BIP39 word seeds and hierarchal dynamic structure. Using a web-based BIP mnemonic converter it is possible to enter the 12-word seed and look set the BIP standards and deviation paths accordingly. There is an off-line version of the site and the code is also available to inspect, entering a word seed into an on-line untrusted source is dangerous. Figure 41 shows the mnemonic code for the Jaxx wallet entered and the Coin is selected as ETH.

The BIP32 root key and BIP39 seed are displayed alongside other Bitcoin-specific keys. The derived address is shown as above Figure 42 as the root-derived account using the M/44'/60'/'/0/0 deviation path Figure 43 is the keys from inside the wallet application. This shows that the restoration of seed words against even all the potential coins available and a number of addresses could prove potentially very useful to investigators. If an incorrectly formatted hash is checked against the wrong blockchain it will return a null. Clashes or brute-forcing passwords is theoretically possible but unlikely, this type of information would be further corroborated with additional investigative actions reducing the risk of a false positive. In addition to the Ethereum address recovery above the same was done by changing the coin option on the site recovery to Bitcoin and also Litecoin, Jaxx rendered the first address for each coin within the application to the mnemonic tool value verifying this method. Revealing of accounts could be cross-matched with forensic artifacts recovered from a device, cross-matching of addressing can be attributed to use. This method can also enable access to the recovered funds and wallets for asset recovery where the necessary jurisdiction and legislation allows for such activity.

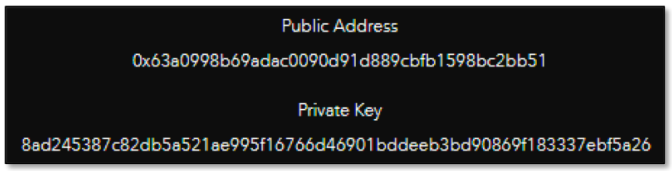

Figure 43: Public and private address displayed from the Jaxx Wallet

6. Network Attribution

There are a number of additional areas that can be examined in order to associate a user to a network or identify use. The ENS system is investigated in this section to look at how the system operates and if any opportunities are available for association.

6.1. Ethereum Naming Service

As discussed in the proceeding literature review the use of a human-readable name can assist the user and wider adoption where a complex large hash is difficult to use for day to day services. In order to demonstrate how the ENS works and to replicate a transaction across the network using ENS a scenario is set-up with two names for two accounts on the Ropsten Testnet. Firstly the ENS name has to bid for and won, the current system of auction is unlikely to remain over the coming years so will not be documented here ("What is and how to use the Ethereum Name Service (ENS) [55]. The bid process was eventually completed with a number of failed transactions due to high GAS load on the network and a final completion step that was originally missed. The set-up has two accounts that use a human-readable name and that can receive transactions. These are alice-wallet.eth and bob-wallet.eth. There are limited resources on Ropsten at this time for interacting with the ENS, myetherwallet.com was used to register the names on the Ropsten network. The live network has a number of management repositories that can be used to manage purchase or look up. These include, ENS listing, Name Bazaar, myetherwallet and ENS manager that importantly includes reverse lookup [56].

In Figure 44 it shows the newly acquired domain name "alice-wallet.eth" the labelhash is a SHA3 Keccak 256 Hash of the "alice-wallet" to produce the Labelhash value. The Namehash is similarly a Keccak hashed value but this includes a concatenation of the Top-Level domain and subsequent Label hashing. Figure 45 shows a python script from the ENS documentation on how this created, this enables cross-referencing. An example of use would be where an ENS name is used in a cybercrime the Namehash can be replicated and searched on the blockchain for events in order to locate associated activity.

Lookup data provides a number of potentially important information for investigators these include the Owner, Highest Bidder and resolved address. In Figure 46 a created domain exists for Bob, this is bob-wallet.eth.

The label hash is the SHA3 Keccak result and the Namehash is created. The additional fields then show the owner and deed owner. Figure 44 shows the results of the search on alice-wallet.eth the owner, highest bidder and resolved address are

Figure 44: Lookup using myetherwallet ENS service

```
def namehash(name):
  if name == '':
    return '\0' * 32
  else:
    label, _, remainder = name.partition('.')
    return sha3(namehash(remainder) + sha3(label))
```

Figure 45: Namehash algorithm

all the same. This is because the owner who bid on the auction won has then chosen to set the resolver address to direct to the same wallet address. However, when examining Figure 46 the owner and bidder are shown as the address used for alice-wallet.eth and the resolved address is different. In this example, Alice owns the name after the winning bid, however, chooses to assign to the wallet controlled by Bob. ENS works in a similar fashion as DNS and the transferring and redirection of the address are possible. ENS works for a number of services on the Ethereum network. In the example presented it relates to humanly readable wallet addressing. The address requires registering by the owner to a resolver in this example and in many cases, this will be the public ENS resolver, but note other registers can be managed by registrars for other purposes. An address must, therefore, be set on the Registry contract to set the resolver used, in the example the public resolver. Then in the chosen resolver contract it requires setting to the account address to be resolved with the relevant Namehash. These are all actions and interactions with a contract on the blockchain from the account owner, a number of blogs "readme docs" detail this [57]. Table 5 shows the Name and subsequent definitions.

To replicate a transaction the private key was taken from the Meta-mask client and restored in Meta-mask on the Windows 10 node to ensure there is no underlying address caching processes. The client is then receiving addressing information only

Figure 46: Result of a search for bob-wallet.eth

from the blockchain, Ropsten in this example. The transaction test will be to send an amount of eth from Alice's wallet and send to the address bob-wallet.eth.

The transaction is created as normal and where the hexadecimal code is entered the human-readable form is entered. In Figure 47 the words bob-wallet.eth are preceded by a green tick this indicates a valid address. The next part of the transaction section shows that Meta-mask has in fact rendered the address to the hexadecimal address having consulted the public resolver prior to conducting the transaction. The transaction is then conducted as standard, there are no additional data requirements other than standard GAS pricing and limits.

6.1.1. Method for locating ENS data

Utilising the look up naming services previously referenced requires the human-readable name to entered to obtain the resolver and account owner detail. Reversing the name hash of the subject human-readable name will enable a search of activity or events that may allow the linking of the ENS address to an account. In the example bob-wallet.eth the Name hash is:

36bf55530c272ae1020abaa0b31f857485573448e6c
369e465baa8a6cc652a81

Setting the ENS entry requires the registry to set the resolver, the phone directory, or where the address will be recorded. Then an entry is made into the directory of where and who the entry relates this links the unique Namehash to an account. As both of the above actions are a contract process an entry exists on the blockchain, Figure 48 and Figure 49 show the MethodID and topic fields from the transactions from Etherscan.

It is then possible to use the event filters in the Etherscan API to produce event logs, where the filter looks at the registry contract and returns where Topic1 is the Namehash. This command is needed to register any hex address to human-readable address, if there is no resolver assigned then there is no route to another entity. This could indicate where the account owns the Namehash but is domain squatting, waiting to sell or not yet set-up. This method, therefore, enables an investigator to search for an active Namehash. Block restrictions on the API are important here and the use of a staged search would be required - Figure 50 shows an API request.

The full JSON response returns the details for the resolver address and that can then be subsequently filtered to search for the name hash. Figure 52 shows the transactions index this then gives the sending address, the owner, and attribution can then made as the other ENS listings and interactions setting the address to the resolver gives bobs hexadecimal account address. Full attribution for the flow of funds and the linked ownership and management is possible using this method. ENS is similar to DNS that after market sales and transfers are possible, this should be taken into account when attributing ownership, purchase and current activity.

This feature is used but not currently a highly adopted practice, however, the introduction of new TLDs (Top Level Domains) and upgrades alongside DAPP integration may increase its use. Despite some advantages of the protocol it is likely that there will be a continued misuse of the service, phishing using different encodings that appear correct to a human eye

Function Name	Function definition
Name	The human readable address
Namehash	This is the important hash value that is set on the blockchain
Resolved address	The location / address of the destination

Table 5: Name and definition Table

is reported. This used alongside long-standing similar addressing and social engineering will increase if and when large-scale adoption occurs [58]. Bad Dapp another vector could be the use of a bad Dapp that has an in-built resolver if this is set with false ENS addresses to benefit the attacker. A man in the middle attack similar to a DNS attack would I think be possible. If the resolver uses a registrar alongside the public resolver this would perhaps satisfy this issue.

The ability to use the ENS to point to hosted files or web content stored on the swarm distributed node network is moving from testing and onto the main network [59]. If the file or index.html is hashed and has been linked to a resolver then the trace of the resolved address and the "bzz" hash can be located on the ENS and relevant owning account. The swarm protocol is still new and is operating alongside other distributed storage systems implementations such as IPFS technology and Storj and Sia. The scope of analysis of research into these areas will be necessary over the coming months as those projects mature. The ability to host material in a censorship-resistant distributed protocol could, however, cause great concern for the hosting of indecent images of children or terrorist doctrine with no ability to remove or prevent additional publication. Combined with encrypted protocols and smart contract mechanisms for payment and delivery of content this perhaps a new dawn and frontier for law enforcement. As law enforcement and Internet providers tackle the sharing and hosting of images using hash sets a new set of distributed hashes will be required for future enforcement [60].

6.2. Mined blocks

While establishing the activity of the subject account or potential sources of incoming Ethereum mining should be considered. The current proof of work mining process is similar to the Bitcoin mining protocol and is still present but is in the phase-out towards the proof of stake release. The proof of work (POW) uses the computational power of large graphics processing units to create a large number of calculations to mine a block. The process of mining is well documented and requires numerous multi-cored expensive GPUs normally installed in rigs combining CPU, ram and multiple GPU cards [61]. Solo mining is unprofitable due to the probability of finding blocks so mining pools are used to group together the chance of finding blocks and sharing the proceeds. Subjects accounts can be viewed on Etherscan or an API call to view blocks mined by the account. Due to the power involved in the POW to be profitable if mining is discovered the use of mining pools are highly likely. Mining pools are often labelled up on Etherscan and other services and have very large numbers of transactions from the account including mined blocks. Figure 53 shows the Etherscan result from a mining pool.

If a subject is receiving numerous deposits from a mining pool with no obvious hardware it could indicate additional offsite premises with mining rigs. Another popular trend is cryptojacking the deployment of malicious scripts running in websites, effectively a drive-by download where theft of CPU power is used. Ethereum is heavily intensive and is less likely to be used for this type of attack but historically was more profitable, Monero and Zcash are the favoured coins of choice due to the mining methods [62]. The use of compromised botnets are also prevalent utilising the compromised machine resources to mine for cryptocurrency, other more specific attacks include vulnerable routers to target Ethereum miners [63]. If a suspect is receiving regular mining payments without hardware criminal activity is very likely.

6.3. Network and Processing

It was not possible during this thesis to replicate a satisfactory environment to reliably port scan and document an external view of the visibility of a running node from a penetration testing perspective. The following observations are made from running the Linux Parity node and the Windows 10 Geth-based Ethereum wallet. A before and after snapshot of processes and network connections were captured. A process was recorded in Linux as "parity" post-launch of the network. Windows 10 process "tasklist" recorded two Ethereum Wallet.exe and a geth.exe following the launch of the Ethereum wallet.

When utilising the "netstat" command before and after node launch it is obvious in the results with the number of 100+ increased connections. Numerous connections are made using the standard TCP port for Ethereum networks on 30301 and the majority on 30303 or other non-standard port settings. The ability to attach nodes to the network and monitor I.P data is similar to the Bitcoin network and the ability to harvest node information has been performed to map some of the Bitcoin network previously. It is not possible to make an assessment if I.P information could be linked to transaction propagation without a detailed research program. As a potential area of interest, the nature of transactional attribution and I.P attribution could have merit depending on network propagation difference between Ethereum and Bitcoin [64]. It is possible to discover nodes on the Ethereum network using Ethernodes service, required is either the "enode" address this was displayed in the Parity client or using the I.P and port address [65]. There are packages and scripts to enumerate exploitable Geth nodes (Eldad, 2016).

7. Conclusions

The techniques discussed were replicated and tested for accuracy using testnet and the live network. As tools focus on

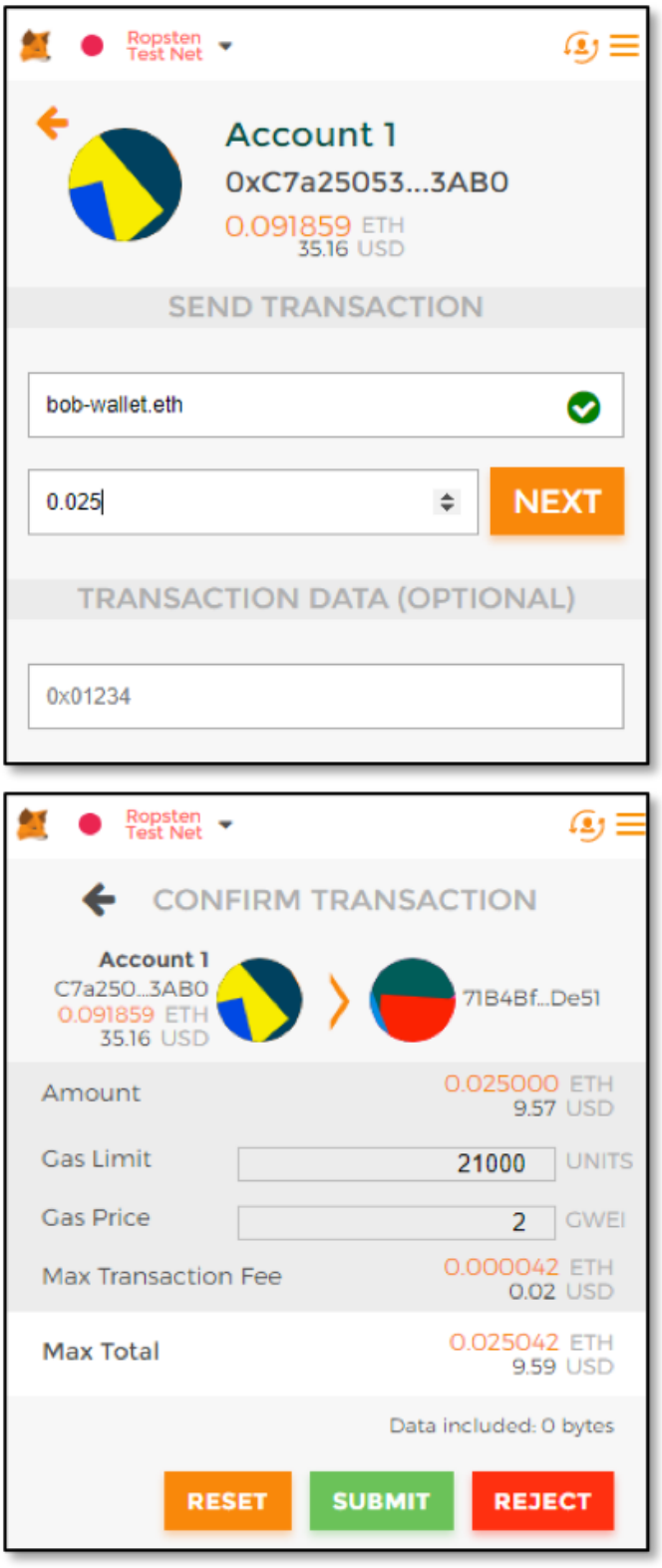

Figure 47: Entering the bob-wallet.eth and the tick confirming recognition

```
Function: setResolver(bytes32 node, address resolver)

MethodID: 0x1896f70a
[0]:  36bf55530c272ae1020abaa0b31f857485573448e6c369e465baa8a6cc652a81
[1]:  0000000000000000000000004c641fb9bad9b60ef180c31f56051ce826d21a9a
```

Figure 48: Contract interaction with the ENS service setting the resolver

```
Function: setAddr(bytes32 node, address addr) ***

MethodID: 0xd5fa2b00
[0]:  36bf55530c272ae1020abaa0b31f857485573448e6c369e465baa8a6cc652a81
[1]:  0000000000000000000000071b4bf6d63dfe43d851c3f147ef91dbff50bde51
```

Figure 49: Contract interaction with the resolver updating the Namehash

https://api-ropsten.etherscan.io/api?module=logs&action=getLogs&fromBlock=2956170&toBlock=2956190&address=0x112234455C3a32FD11230C42E7Bccd4A84e02010&topic1=0x36bf55530c272ae1020abaa0b31f857485573448e6c369e465baa8a6cc652a81&apikey=*APIKEY*

Figure 50: Etherscan API search for the resolving data


```
{
  "status": "1",
  "message": "OK",
  "result": [
    {
      "address": "0x112234455c3a32fd11230c42e7bccd4a84e02010",
      "topics": [
        "0x335721b01866dc23fbee8b6b2c7b1e14d6f05c28cd35a2c934239f94095602a0",
        "0x36bf55530c272ae1020abaa0b31f857485573448e6c369e465baa8a6cc652a81"
      ],
      "data": "0x0000000000000000000000004c641fb9bad9b60ef180c31f56051ce826d21a9a",
      "blockNumber": "0x2d1b93",
      "timeStamp": "0x5ac25673",
      "gasPrice": "0x9502f9000",
      "gasUsed": "0xb84d",
      "logIndex": "0x2",
      "transactionHash": "0x6321e2b8347425f2621df162e688064c950d75045838f144153c85f9c4282701",
      "transactionIndex": "0x3"
    }
  ]
}
```


Figure 51: The JSON return from the above previous API request

bob-wallet.eth is already owned:

Name:	bob-wallet.eth
Labelhash (bob-wallet):	0x09471a12ace64433b857feb06365da8104b99e33f773602dd460b3dc06b2b14b
Namehash (bob-wallet.eth):	0x36bf55530c272ae1020abaa0b31f857485573448e6c369e465baa8a6cc652a81
Owner:	0xc7a2505346b65e702c637d089c2a91277c743ab0
Highest Bidder (Deed Owner):	0xc7a2505346b65e702c637d089c2a91277c743ab0
Resolved Address:	0x71b4bf6d63dfe43d851c3f147ef91dbff50bde51

Figure 52: Return from the registry feature following addressing changes

Overview | f2pool_2

ETH Balance:	1,421.977023842452417582 Ether
ETH USD Value:	$521,268.34 (@ $366.58/ETH)
Mined:	344083 blocks and 41827 uncles
No Of Transactions:	4194102 txns

Figure 53: F2pool, mined blocks and no of transactions indicate a mining pool

the movement of Ethereum from one account to another the use of tokens and exchanges within the contract structure must be understood to see the real picture. The use of decentralised exchanges enable movement from one chain to another using different tokens and this will be a new challenge. Transfer of value can be conducted through digital collectibles with new classes of token, money laundering could be conducted by the exchange thousands of dollars' worth of digital Cryptokitties. The use of ENS to direct funds to human readable names from the abstract hash address values have shown the ability to identify owner accounts to follow funds. The use of ENS can also direct to distributed stored files or web documents that creates a censorship resistant web hosted entity. The need to access event logs for an ENS set address function will enable attribution to an owner's wallet address. MethodID registry of contract functions to parse data on transactions is essential as demonstrated with display token movements.

8. Appendix

Please refer to this link for the associated content: http://asecuritysite.com/ethpaper

References

[1] D. Siegel, "Understanding The DAO Attack - CoinDesk," 6 2016.
[2] Cyber-blog, "Huge Ethereum Mixer – cyber • Blog," 12 2017.
[3] "ERC20 Token Standard - The Ethereum Wiki," 2017.
[4] E. R. Leukfeldt, A. Lavorgna, and E. R. Kleemans, "Organised Cybercrime or Cybercrime that is Organised? An Assessment of the Conceptualisation of Financial Cybercrime as Organised Crime," *European Journal on Criminal Policy and Research*, vol. 23, no. 3, pp. 287–300, 10 2017.
[5] A. Lavorgna, "Internet-mediated drug trafficking: towards a better understanding of new criminal dynamics," *Trends in Organized Crime*, vol. 17, no. 4, pp. 250–270, 10 2014.
[6] E. W. Kruisbergen, E. R. Kleemans, and R. F. Kouwenberg, "Explaining attrition: Investigating and confiscating the profits of organized crime," *European Journal of Criminology*, vol. 13, no. 6, pp. 677–695, 10 2016.
[7] V. Buterin, "Devcon2: Ethereum in 25 Minutes," 11 2016.
[8] C. Dannen, *Introducing Ethereum and Solidity*. Berkeley: Apress, 10 2017.
[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.
[10] Ambrosus, "Ambrosus-White-Paper-V8-1.pdf," 5 2017.
[11] E. Foundation, "Devcon3 Day 2 Stream - Afternoon," 2017.
[12] V. Buterin and others, "A next-generation smart contract and decentralized application platform," *white paper*, 10 2014.
[13] "How DNS Works In Six Steps - Verisign," 2017.
[14] E. Foundation, "Ethereum Foundation Devcon3 Day 1 Stream," 2017.
[15] "Ethereum Name Service Lookup," 2017.
[16] "DNSSEC - ICANN," 2017.
[17] "Ethereum Name Service," 2017.
[18] N. Johnson, "How to claim your DNS domain on ENS," 11 2017.
[19] ——, "dnssec-oracle: A DNSSEC oracle for Ethereum," 2017.
[20] B. J. Nikkel, "Domain name forensics: a systematic approach to investigating an internet presence," *Digital Investigation*, vol. 1, no. 4, pp. 247–255, 2004.
[21] "ropsten - Comparison of the different TestNets - Ethereum Stack Exchange," 2017.
[22] "Parity."
[23] Codetract, "Inside an Ethereum transaction – CodeTract – Medium," 2017.
[24] M. Murthy, "Life Cycle of an Ethereum Transaction – BlockChannel – Medium," 2017.
[25] P. Kasireddy, "How does Ethereum work, anyway? – Preethi Kasireddy – Medium," 2017.
[26] R. Costa, "RLP · ethereum/wiki Wiki · GitHub," 2018.
[27] "ethereum-tx-decoder," 2018.
[28] "Ethereum Developers APIs," 2018.
[29] "web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation," 2018.
[30] "Keccak Team," 2018.
[31] A. Hern, "'$300m in cryptocurrency' accidentally lost forever due to bug | Technology | The Guardian," 2017.
[32] P. Bylica, "How to Find $10M Just by Reading the Blockchain – The Golem Project," 2017.
[33] "Application Binary Interface Specification — Solidity 0.4.22 documentation."
[34] "GitHub - ethereum/EIPs: The Ethereum Improvement Proposal," 2018.
[35] V. Buterin, "Standardized_Contract_APIs · ethereum/wiki Wiki · GitHub," 2015.
[36] "Etherscan Token Tracker Page," 2018.
[37] "CodeTract - Token standard," 2016.
[38] Benjaminion, "etherscan - Obtain token transfer data from an address - Ethereum Stack Exchange," 2017.
[39] "Create a cryptocurrency contract in Ethereum," 2018.
[40] S. Liao, "This man has made more money trading cryptokitties than investing in his IRA - The Verge," 2017.
[41] Wax.io Web, "Gaming Skin Trading: A $50 billion industry ripe for blockchain disruption," 2017.
[42] C. Hall, "Valve can't stop Steam account theft and customers are suffering because of it - Polygon," 2015.
[43] J. Nation, "Ubisoft Eyes Digital Collectibles On Blockchain - ETH-News.com," 2018.
[44] S. Walters, "Review of the Loom Network: Powering Ethereum Collectibles - Coin Bureau," 2018.
[45] "ETH Gas Station | Consumer oriented metrics for the Ethereum gas market," 2018.
[46] J. Buntinx, "Popular Ethereum Wallets Suffer From Inflated Default Gas Prices – The Merkle," 2017.
[47] "Ethereum Wallet Cracking | Stealthsploit," 2017.
[48] Babay88, "Web3 Secret Storage Definition · ethereum/wiki Wiki · GitHub," 2018.
[49] D. Kholia, "JohnTheRipper/ethereum2john.py at bleeding-jumbo · magnumripper/JohnTheRipper · GitHub," 2017.
[50] Chick3nman, "ether2hashcat.py/ether2hashcat.py at master · Chick3nman/ether2hashcat.py · GitHub," 2017.
[51] "Extracting the Jaxx 12-word wallet backup phrase. – vxlabs," 2017.
[52] "bips/bip-0039-wordlists.md at master · bitcoin/bips · GitHub," 2017.
[53] "Address - Bitcoin Wiki," 2018.
[54] V. Kobel, "Create full Ethereum wallet, keypair and address," 2017.
[55] "What is and how to use the Ethereum Name Service (ENS) | CryptoCompare.com," 2018.
[56] "ENS Manager," 2018.
[57] W. Deck, "Tutorial for Setting up a Domain on the ENS and Hosting Decentralized Content on Swarm [Part 1]," 2018.
[58] M. Samy, "Dealing with ENS Names? Beware of this phishing attack. . . ," 2017.
[59] W. Deck, "Tutorial for Setting up a Domain on the ENS and Hosting Decentralized Content on Swarm [Part 2]," 2018.
[60] K. Fiveash, "IWF shares 'hash list' with web giants to flush out child sex abuse images online • The Register," 2015.
[61] A. Heritg, "How to Mine Ethereum - CoinDesk," 2018.
[62] D. Goodin, "A surge of sites and apps are exhausting your CPU to mine cryptocurrency | Ars Technica," 2017.
[63] C. Cimpanu, "Satori Botnet Is Now Attacking Ethereum Mining Rigs," 2018.
[64] P. L. Juhász, J. Stéger, D. Kondor, and G. Vattay, "A Bayesian Approach to Identify Bitcoin Users," *arXiv preprint arXiv:1612.06747*, 2016.
[65] "ethernodes.org - The ethereum node explorer," 2018.