IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 17, 2017, accepted September 26, 2017, date of publication October 5, 2017, date of current version November 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2759766

Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core

LUUC VAN DER HORST[1], KIM-KWANG RAYMOND CHOO[2], (Senior Member, IEEE), AND NHIEN-AN LE-KHAC[3], (Member, IEEE)

[1]Dutch National Police, 3970 AA Driebergen, The Netherlands
[2]Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249-0631, USA
[3]School of Computer Science, University College Dublin, Dublin 4, Ireland

Corresponding author: Kim-kwang Raymond Choo (raymond.choo@fulbrightmail.org)

ABSTRACT Bitcoin cryptocurrency is reportedly one widely used digital currency in criminal activities (e.g. used for online purchases of illicit drugs and paying of ransom in ransomware cases). However, there has been limited forensic research of bitcoin clients in the literature. In this paper, the process memory of two popular bitcoin clients, bitcoin Core and electrum, is examined with the aims of identifying potential sources and types of potential relevant data (e.g. bitcoin keys, transaction data and passphrases). Artefacts obtained from the process memory are also studied with other artefacts obtained from the client device (application files on disk and memory-mapped files and registry keys). Findings from this study suggest that both bitcoin Core and electrum's process memory is a valuable source of evidence, and many of the artefacts found in process memory are also available from the application and wallet files on the client device (disk).

INDEX TERMS Digital forensics, bitcoin forensics, electrum forensics, bitcoin core, bitcoin client, cryptocurrency forensics, memory forensics.

I. INTRODUCTION

With recent advances in Information and Communications Technologies (ICT) and pervasiveness of popular consumer devices (e.g. Android and iOS devices), digital and cryptocurrencies such as Bitcoin [1], Litecoin [2], Freicoin [3], and Peercoin [4] are becoming increasingly popular in e-commerce. Such currencies, however, can be abused by criminals. For example, it has been reported that Bitcoin was used to purchase illicit drugs online (e.g. Silk Road) [5] to pay criminals in return for the decryption key/password in ransomware incidents [6], and for money laundering and terrorism financing [7]. It has also been reported that some Darknet marketplaces implement cryptocurrency wallets natively [8].

Despite Bitcoins and other virtual currencies being a potential source of evidence, there have been limited research on virtual currency forensics. Existing forensic research on Bitcoins focus on the Blockchain [9], rather than Bitcoin end-user software (e.g. mobile application or wallet) or on the potential to forensically recover data using memory analysis. This is the gap we seek to address in this paper.

In this paper, we examine two popular Bitcoin clients, namely: Bitcoin Core and Electrum, and seek to recover artefacts relating to Bitcoin use from memory.

We will briefly discuss background related work in the next section, prior to describing our approach in Section III. Specifically, we first analyze the properties of both Bitcoin Core v0.11.1 and Electrum v2.6.2 in different scenarios to help us make an informed guess of potentially relevant data objects in the process memory. Subsequently, a virtual machine running Microsoft Windows OS is set up, where the Bitcoin application is installed. Then, the RAM images of the virtual machines are investigated to locate the artefacts in the process memory. The conditions under which these artefacts appear may shed light on how these values can be uncovered from memory. Findings are presented in Section IV. In Section V, the implications of this investigation are discussed. The last section concludes this research and outlines future research.

II. BACKGROUND AND RELATED WORK

In this section, we present firstly an ecosystem of Bitcoin including a primer on Bitcoin and Bitcoin clients. We then show the criminal use of Bitcoin and related work on Bitcoin forensics.

A. BITCOIN PRIMER

Bitcoin is a digital currency and payment system introduced by the pseudonymous Satoshi Nakamoto in 2008 [23]. The system functions as a peer-to-peer, decentral network in which payments are send directly between the parties

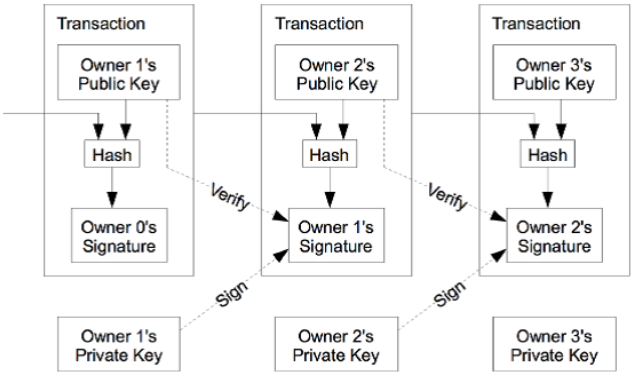

FIGURE 1. Bitcoin chain of transactions [23].

involved, instead of relying on central institutions to settle payments. Bitcoin uses cryptographic proof to make this happen. Hence, Bitcoin is also considered to be a cryptocurrency.

Each user in the Bitcoin network owns public and private key pairs. Bitcoin value is associated with a chain of transactions in which each transaction consists of the private key signature of the hash of the input transaction and the payee's public key as shown in Figure 1.

If the payer wants to send these Bitcoins to the payee owning a public/private key. To do so, the payer takes the hash of the input transaction – proof that he/she owns the Bitcoin value – and the public key of the payee. He/she then hashes these values and signs the hash with its private key, thus creating a new transaction to the payee. The transaction consists of the hash of the input transaction, the output address and the payer's signature. Since only the payer has the private key corresponding to the public address, (s)he is the only one capable of generating a transaction for this address via signing, thus spending the Bitcoin value associated with this address.

To avoid double-spending of Bitcoins, the system adopts a public ledger called the Blockchain in which newly accepted transactions are consolidated in a chain of blocks. Blocks are created by nodes in the network (i.e. Bitcoin miners), which have to solve a cryptographic problem to create a block, and when successful, are rewarded with new Bitcoins. Thus, these nodes invest CPU cycles to earn Bitcoins. In this way, new Bitcoins are added to the system while transactions are settled at the same time. Bitcoin miners fall outside of the scope of this paper. Each block consists of the new transactions and the hash of the previous block. To verify whether a payer owns the Bitcoins, clients in the network can verify the chain of transactions leading up to the payer's key pair in the Blockchain. For performance reasons, it is also possible to download only the headers of the Blockchain for transaction verification. This system is called Simple Payment Verification (SPV). Eventually, a Bitcoin user needs key pairs to own Bitcoin value and to start transactions. These keys are stored in the system running the Bitcoin node. A Bitcoin key store is generally referred to as a Bitcoin wallet.

Bitcoin uses the Elliptic Curve Digital Signature Algorithm (ECDSA) for its public/private keys. An ECDSA private key consists of 32 bytes of random data. The corresponding public key consists of 65 (uncompressed) or 33 (compressed) bytes of data.

For a payer to make a transaction, (s)he has to know the public key of the payee. To make it easier to share the (binary) public key, it can be changed into a character string using the base58 algorithm. The public key in base58 format is also called a Bitcoin address. Private keys can also be transformed to base58 to facilitate sharing. This format for private keys is called the Wallet Import Format (WIF). In addition, when encoding Bitcoin keys into base58 format, a four-byte checksum is added to the binary value [3].

Bitcoin transactions can be identified with the SHA256 hash of the transaction data. This value is called the transaction-ID of the transaction. Furthermore, in most clients, users can associate user labels with transactions and public keys in the wallet. However, these labels are not part of the transaction or Blockchain data.

Several newer Bitcoin wallets make use of hierarchical deterministic (HD) wallets, where keys in the wallets are derived from a master key pair. As a result, all keys can always be recovered with the master key. Since key derivation is fully deterministic, this setup makes it possible to store and create private keys on a different machine than the public keys. This offers improved security, as the 'private' machine does not have to be connected to the Internet.

B. BITCOIN CLIENT

Bitcoin end-users need specific client software (in this paper, the terms Bitcoin client and Bitcoin application are used interchangeably) to store Bitcoin keys and to initiate new transactions. Bitcoin is based on a consensus model and there is no formal specification of the protocol available. The Satoshi client [1] serves as a reference implementation for new clients. Specifically, its implementation of the Bitcoin network protocol is generally considered the *de facto* standard, at least to the extent that it facilitates interoperability with new Bitcoin software [3].

C. BITCOIN TUMBLERS

Bitcoin Tumblers or so-called Bitcoin Mixers, have been exploited as a money laundering facility by exchanging Bitcoins into conventional currencies such as U.S. dollars and Euros. Popular services include Darklaunder, Bitlaunder, CoinMixer, and Helix. Each service normally offers one or more types of laundering with different levels of security. These services claim that they do not store or collect personal information. However, such claims have not been verified and are not in the scope of this paper.

D. BITCOIN CRIMINAL USE

As previously discussed, Bitcoin is increasingly popular, including with criminals, and therefore the importance of

cryptocurrency forensics. Some of the criminal uses of Bitcoin are outlined below.

1) MALWARE

One of the earliest known abuses of Bitcoin relates to the use of botnets for Bitcoin mining. In the simplest variant, mining software is downloaded and executed by the bots on infected machines. This software will then contribute to mining pools which in turn cash out to criminals. This modus operandi monetizes a previously untapped resource of infected computers: CPU cycles. It can easily be combined with other botnet uses such as click fraud or spam attacks. Huang *et al.* [13] discussed (pseudo-)anonymous set-ups of mining botnets and profitability, and they listed nine different examples of mining botnets. Other uses of malware in relation to Bitcoin were provided in by Dell SecureWorks Counter Threat Unit [29]. According to this analysis, cryptocurrency stealing malware (CCSM) can, for instance, focus on stealing credentials, wallet files or posing as a transaction for a man-in-the-middle attack.

2) DDOS ATTACKS

In distributed denial of service (DDOS) attacks, Bitcoin is one of the ways that victims pay the cybercriminals orchestrating such attacks. For example, a criminal group using this modus operandi was reportedly taken down by law enforcement agencies in December 2015 [9]. In the study of Vasek *et al.* [33], it was determined that up to 60% of large Bitcoin mining pools had experienced a DDOS attack to some extent. Consequently, the hash rate of the affected mining pools decreased, and resulted in a better position for other mining pools (e.g. those under the control of the attackers).

3) MONEY LAUNDERING

Due to the pseudo-anonymous and decentralized nature of cryptocurrencies have lead criminals to use it for money laundering extensively. Since cryptocurrencies may not have in place anti money-laundering / counter terrorism financing (AML/CTF) practices such as customer due diligence (CDD) and know your customer (KYC), criminals may exploit such currencies to launder criminal proceeds or finance terrorists. To date, a number of darknet marketplaces include a native bitcoin wallet per account or a bitcoin-based escrow system. However, FBI assessed with relatively high confidence that it would be possible to deanonymize launderers when they exchanged their bitcoins for fiat money [24]. Additionally, due to the underlying Blockchain technology, one could potentially identify bitcoin users based on their prior transactions.

4) THEFTS AND EXIT SCAMS

Cryptocurrencies and their exchange markets can also be targeted by criminals and malicious market operators (e.g. the collapse of Mt. Gox bitcoin exchange and Evolution darknet market [8]).

E. BITCOIN FORENSICS

Similar to other areas of digital forensics such as cloud forensics, evidence relating to the use of Bitcoins can potentially be located in different locations, such as Blockchain, client software and network protocol. Existing Bitcoin forensics appear to focus on the Blockchain [10]–[13]. We only located two papers on Bitcoin client analysis and none on Bitcoin network protocol analysis. This is, perhaps, due to the fact that Blockchain data is publicly available whereas traffic data and client data have to be generated by the researchers (e.g. purchase and transact using Bitcoins to generate the artefacts for analysis).

One of the first successful attempts to deanonymize Bitcoin users based on Blockchain data was by Meiklejohn *et al.* [10]. The authors showed that it is possible to deanonymize Bitcoin users using Blockchain analysis, in the sense that Bitcoin addresses can be clustered based on certain protocol properties. All addresses in a cluster can then be attributed to the same end user. If the end user of a particular address in the cluster is known, then the other addresses in the cluster can be attributed to this user as well. This approach has been adopted by various other researchers as well. For example, Spagnuolo *et al.* [11] developed BitIodine, a Blockchain forensic tool. They demonstrated its utility on SilkRoad wallets and addresses related to the CryptoLocker ransomware. A number of commercial tools for Blockchain analysis such as Chainalysis [12] and Numisight [13] have also been presented.

Möser *et al.* [14] examined the potential of using Bitcoins in money laundering activities. The authors used test transactions to reverse-engineer the behavior of the Transaction Anonymization Services, which mixes the Bitcoin transactions to unlink sender identities and receiver identities. They concluded that ''budget-constrained cybercrime fighters are effectively set back by two of the three tested services.'' [14]. In other words, existing research on Blockchain properties suggested that it is possible to make useful deductions from this source and even users could potentially be deanonymized.

Early attempts of Bitcoin client software forensics were conducted by staff from Magnet Forensics Inc., the developers of Internet Evidence Finder (IEF) in 2013 [15], [16]. IEF parses two on-disk resources related to Bitcoin clients, namely: wallet files and client log files. This feature has been implemented since version 6.1. Although it is not completely clear from the vendor's website, the tool appears to support only Bitcoin Core client forensics. Montanez in 2014 [2] analyzed Bitcoin wallets such as Litecoin and Darkcoin installed on iOS and Android devices. The author was able to recover relevant metadata, such as installation date and time stamps and usage indicators, using Cellebrite UFED Physical Analyzer, iFunBox (for iOS) and ADB (for Android). For one specific wallet on iOS, bitWallet, a private key was also recovered in plain text. However, Montanez appears to focus only on human-readable data, e.g. in wallet log files and

database files. It is unclear the types of binary data that could be recovered.

In 2015, the SANS Institute published the findings of a forensic analysis of Bitcoin-QT and Multibit applications and of the Bitminter mining software on a Windows device [17]. Similar to Montanez, the research reported in [17] provides a list of application files with their purpose and forensic value. In the analysis, only string-based (human-readable) data was taken into account. This is the only study that we are aware of that examines memory-resident data. Specifically, in [17], memory snapshots were taken from machines running the client software. In turn, analysis of the snapshots was performed by running a keyword search in EnCase. This is a form of unstructured memory analysis. While the search yielded many hits, no detailed analysis of memory location, the data format and its implications were not presented.

While there are a small number of tools designed for Blockchain forensics, only IEF has been shown to support forensic analysis of a client application (i.e. Bitcoin Core). There is, arguably, a need for forensic research into other Bitcoin clients in order to provide the forensic community with an in-depth understanding of the types and locations of artefacts that could be recovered.

III. RESEARCH METHODOLOGY

The research presented in this paper focuses on digital evidence present in memory. Applications running on Windows devices/machines could result in data stored in the memory in the form of memory areas (VADs) with private application data, memory-mapped files, being either executable files and data files and handles to other system resources such as registry keys, connections, and so forth.

Memory-resident data of an application can be analyzed in a structured and an unstructured manner. The structured approach focuses on the OS perspective on memory (processes, files, tables, etc.), whereas the unstructured approach regards the memory dump as a collection of values that can be crawled using tools such as strings and grep. Bitcoin applications function as a storage for Bitcoin keys (wallet) and can contain data of forensic interest, such as public and private keys, addresses, user labels and transaction details.

There are multiple Bitcoin clients with varying user-friendliness and security features. Bitcoin Core [23] and Electrum [24] are two of the most popular clients. Both Bitcoin Core and Electrum are available for Windows machines. While Bitcoin Core lacks a number of features, it is generally regarded as one of the most stable and secure clients. On the other hand, Electrum has more features such as brain wallets and hierarchical deterministic (HD) wallets [24].

Another difference between these two clients is that Bitcoin Core is written in C++, and Electrum is written in Python. Also, Bitcoin Core is a full client in the sense that it downloads a complete copy of the Blockchain for transaction verification. Electrum, on the other hand, connects to a central server from which it downloads the Blockchain headers for Simple Payment Verification (http://docs.electrum.org/en/latest/spv.html). Thus, Electrum enjoys a better performance, but it is generally considered to be less secure since trust is partially delegated to the owner of the server.

In this paper, we focus on the (memory) forensic analysis of Bitcoin Core and Electrum. Not much research has been performed to these clients. The Electrum client has not even been previously studied in the forensic literature, to the best of our knowledge. However, it must be noted that the methodology we adopt in this paper can be used for the forensic analysis of other Bitcoin clients.

A. OVERVIEW

This research takes the following approach to discover relevant data in process memory of an application. First, the application under investigation is analyzed with respect to the relevant data it may store (temporarily) in memory. This may vary between applications. In the context of Bitcoin clients, such data include Bitcoin key material and user labels. For example, Bitcoin Core and Electrum may support application-specific functions which could be of relevance to a forensic investigator. The relevant application functions and memory-resident data of potential interest are described in the next section.

When the relevant data are identified, the next step is to set these to controlled, pre-known values. Thus, the target application will be installed on machines dedicated for this forensic study (e.g. machines are wiped prior to installing the Bitcoin clients). In the application, the values are entered and memory images are made at specific points in time. The lab set-up and the state of the application for each memory snapshot are described in Section ''Memory Image''. Subsequently, the memory images created in the previous step will be analyzed with the memory forensic framework, Volatility. Since the values identified in the first step are pre-set, it is possible to trace their existence, location and format in memory. The search process is described in Section ''Analysis''. Additionally, memory mapped files, registry keys and connections of each application are explored for completeness.

The rationale between this method is twofold. First, uncovering which artefacts can be uncovered from memory per application provides forensic investigators with clues on the relevance of memory forensics with respect to these applications. Secondly, the conditions under which these artefacts appear in memory can be used as heuristics to dig up these artefacts from other memory images.

B. APPLICATION DATA

A Bitcoin client generally has three major functions: 1) to store keys and user data securely, 2) to initiate Bitcoin transaction from the wallet, and 3) to request for Bitcoins. A Bitcoin client may support other features as well, but these are beyond the scope of this research. An example of one such function supported by both Bitcoin Core and Electrum is the option to sign and verify human-readable messages.

In Bitcoin Core, keys and user data are stored in a Berkeley database. This database is written in C and stores the

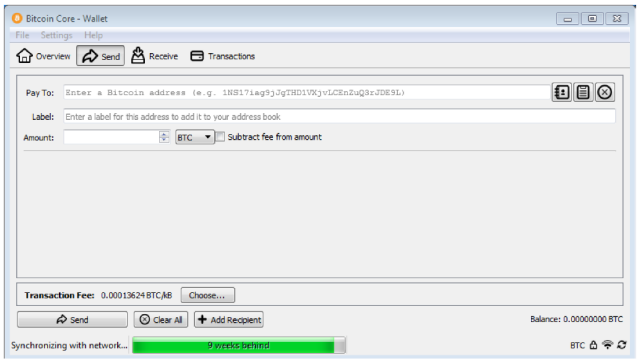

FIGURE 2. Bitcoin Core Send tab.

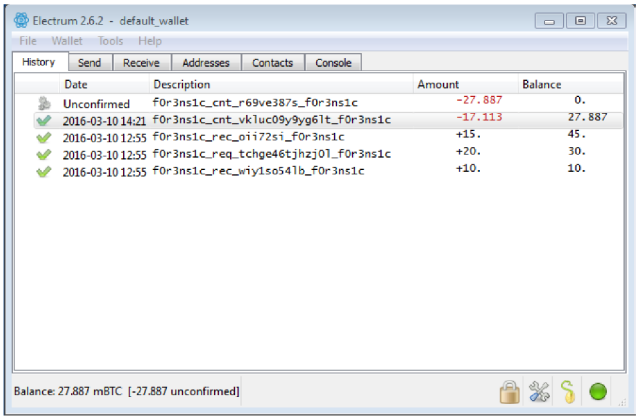

FIGURE 3. Electrum transaction history.

key material as binary data. By default, this database is not encrypted, but the user can choose to do so via the menu option 'Settings >Encrypt Wallet'. Apart from the Bitcoin private and public keys in the wallet, the database stores various user data, such as contacts – basically an association between a label and a public address – and the transaction history of the wallet. Bitcoin Core also enables the user to make a backup of a wallet file to a location on disk.

The user can send Bitcoins from the wallet via the 'Send' tab, which is shown in Figure 2. To do so, the user enters the Bitcoin address, label and amount for the transaction in this screen and presses 'Send'. Optionally, the user can add additional addresses to the same transaction or override the default settings for determining the transaction fee. Note that the label is not part of the public Bitcoin ledger, but is stored in the wallet for bookkeeping purposes.

To request a payment to the wallet, the user has to enter the label, amount and message in the 'Receive' tab in Bitcoin Core and press 'Request Payment'. In turn, the application generates the request for a Bitcoin address available in the wallet. This address corresponds to a public and private key pair in the wallet. The request is represented as a URI string and as a QR code, which then can be shared with other Bitcoin users for them to initiate the payment requested.

Differences between Electrum and Bitcoin Core include the following. In Electrum, data is stored in JSON format in a file; consequently, key material is stored as text-based data. Also, the private data (seed and master private key) in Electrum is encrypted by default. When the user creates a wallet, (s)he has to supply a passphrase for encryption. Electrum uses hierarchical deterministic wallets in which all keys are derived from a single master key pair. After wallet initialization, 37 key pairs would have been pre-calculated and visible in the 'Addresses' tab in the wallet. Furthermore, Electrum generates a seed of 13 natural language words during wallet initialization from which the master private key (and thus all private keys) in the wallet are derived. Seed phrases are described in Bitcoin Improvement Proposal 39 (https://github.com/Bitcoin/bips/blob/master/bip-0039.mediawiki). Note that Electrum's seed phrase is similar to BIP39, but not completely BIP39-compliant. This seed can be memorized by the user and allows the user to restore all values in the wallet. Two other (minor) differences between Electrum and Bitcoin Core are as follows. During payment request, Electrum allows the user to set a specific expiry date for a request. It is not possible for an Electrum user to specify a distinct 'message' for the request, as the user can only specify a label.

TABLE 1. Application data in bitcoin core and electrum.

Application	Function	Data
Bitcoin Core	Secure Storage	Public and Private keys, transaction data (address, label, transaction-id, amount, fee and timestamps), contacts, passphrase, backup location (s)
	Send Bitcoins	Address, label, transaction-id, amount, fee, timestamp
	Receive Bitcoins	Public and private keys, address, label, message
Electrum	Secure Storage	Master public key, master private key, derived public and private keys, transaction data (address, label, transaction-id, amount, fee and timestamp), contacts, passphrase, seed, backup location(s)
	Send Bitcoins	Address, label, transaction-id, amount, fee, timestamp
	Receive Bitcoins	Public and private key, label, amount, expiry date

Figure 3 presents a screenshot of the Electrum client showing its transaction history screen, and Table 1 summarizes the application data corresponding to each application function in Bitcoin Core and Electrum.

In this paper, we seek to determine whether data in the third column of Table 1 are present in the process memory of Bitcoin Core and Electrum. Although all data in Table 1 have forensic relevance, not all can be easily traced back in the process memory. In particular, it is not clear in which format that timestamps and transaction amounts and fees are processed in memory. Hence, these values are excluded from the analysis below.

C. MEMORY IMAGES

Memory images are acquired from a virtual machine running in various states. The virtual machine runs the Microsoft Windows 7 Enterprise SP1 (64-bit) operating system in a licensed VMWare Fusion Professional Edition v6.0.65 environment. The VM is configured with 1 GB of RAM. The operating system was fully patched in all machines. Additionally, all virtual machines had VMware Tools installed. In the VM, Electrum v2.6.2 was installed and brought to a specific state. Bitcoin Core v0.11.1 based on QT v5.5.0 was used in the Bitcoin Core scenarios. Memory snapshots were acquired by suspending the virtual machine and copying the file with the .vmem extension from the virtual machine's folder.

Table 2 describes the state of each application and the properties of each memory image created in this research. Bitcoin Core has been investigated both in an unencrypted and in an encrypted state. Since Electrum v2.6.2 only supports a wallet with encrypted private data, this application cannot be analyzed in an unencrypted state. During the creation of the memory images, all user data (e.g. labels, passphrases) and application data (e.g. addresses, public and private key values) were documented in detail.

During memory analysis, these values were traced back in the memory images to determine their format, location and context (see the next paragraph). Incoming transactions to the wallets under investigation come from an external wallet, whereas all outgoing transaction return to this same external wallet. Furthermore, all labels defined in the scenarios follow a predefined format, namely:

MAGIC_CATEGORY_RANDOM_MAGIC

In this format, MAGIC is a fixed value set to f0r3ns1c to facilitate tracing in memory. CATEGORY indicates for which purpose the label was created (e.g. the value 'req' for a payment request). In turn, RANDOM consists of a variable-length alphanumerical random string which ensures the uniqueness of the label.

TABLE 2. Memory images per application.

No	Application	State	Memory Image
1	Bitcoin Core	Unused Client (benchmark)	Unused client with initialized, unencrypted wallet. The wallet is fully synchronized with blockchain.
2	Bitcoin Core	Used Client (unencrypted)	Based on scenario 1, after usage. The wallet is not encrypted. Entries of sending and receiving addresses (contacts) are created with user-specified labels. The wallet has several incoming transactions. An on-disk backup of the wallet is made.
3	Bitcoin Core	Used Client (encrypted)	Based on scenario 2, but the wallet is now encrypted.
4	Bitcoin Core	Used Client after reboot	Based on scenario 3. The virtual machine is rebooted and an outgoing transaction is initiated.
5	Electrum	Unused Client (benchmark)	Unused client with initialized, encrypted wallet. A seed has been generated for the wallet and a passphrase is set. The wallet is fully synchronized with the Electrum server.
6	Electrum	Used Client (encrypted)	Based on scenario 5, after usage. Entries of sending and receiving addresses (contacts) are created with user-specified labels. The wallet has several incoming transactions. An on-disk backup of the wallet is made.
7	Electrum	Used Client after reboot	Based on scenario 6. The virtual machine is rebooted and an outgoing transaction is initiated.

D. ANALYSIS

In the next stage, the memory images are analyzed using Volatility v2.5 [18] and standard Linux command-line tools in a virtual machine running Kali GNU/Linux v2.0 (64-bit) operating system. Memory snapshots were stored on the host system and available to the analysis machine via an HGFS mount point. The memory images were analyzed in two different ways.

Most importantly, all application data known to be processed by the application as specified in the previous step were traced back in memory. This analysis is similar to the unstructured approach but Volatility makes it possible to attribute values found in memory to a particular process. For this analysis, the yarascan plugin in Volatility was used. We then modified the yarascan plugin, which includes detailed metadata of the VAD area in which a particular value was found in its output. For completeness, the full memory image was searched, rather than only the (private) process memory of the Bitcoin application. Binary values, such as public and private keys, were searched both as string and as binary value.

All known user strings found in memory were analyzed for their format, their location including their VAD properties, and their immediate context. Based on these results, it is possible to determine which application data is likely to be memory-resident and how it might be retrieved from memory when their values are not previously known (as is the case in a regular forensic investigation).

Apart from the unstructured analysis described above, a more structured analysis was performed on the Bitcoin application process in each memory image. In particular, the memory-mapped files, registry keys and connections of the application were examined using standard Volatility plugins. This is to ensure that no important forensic clue is missed. Also, if it is possible to extract Bitcoin data files (i.e. wallet files and log files) from memory, then these files would be processed as standard on-disk artefact. Thus, further detailed memory analysis becomes unnecessary.

IV. FINDINGS FOR BITCOIN CORE

A. PROCESS MEMORY

We now describe and explain which data were excluded from the final result set, prior to describing the outcome of the search process such as the occurrences, format, location and context of all values traced back in the process memory of Bitcoin Core.

1) DATA EXCLUSION

All occurrences of known forensic values were identified in the memory images with the Bitcoin Core application (memory images of scenarios from 1 to 4 in Table 2, and hereafter respectively referred to as memory image 1, memory image 2, memory image 3 and memory image 4). The results from the process vmstoolsd.exe were omitted because this application will not be running on a typical system setup. Similarly, occurrences of known artefacts in kernel memory were omitted from the analysis, since it is not clear why the values found in kernel memory were present there and whether they can be attributed to typical application use. For instance, during configuration of the application, information such as labels and bitcoin addresses were copied to and from the virtual machine. The occurrences in kernel memory of these labels and addresses can be a result of those actions. Moreover, comparison between kernel memory and process memory of Bitcoin Core showed that any data present in kernel memory would also be present in the process memory. Hence, the analysis of kernel memory has no added forensic value.

Search hits for the default public address label "default" were excluded from the dataset due to the overwhelming number of false positives. In addition, only six occurrences of four-byte checksum values related to public and private keys were found. Manual analysis of the memory locations of these hits showed that these occurrences had no relation to the full key values and could not be attributed otherwise to relevant forensic information. Hence, these occurrences were excluded from further analysis.

2) OCCURRENCES

The number of occurrences of known forensic artefacts are shown in Figure 4. The results from the search process are described in more detail below.

Item Type	Memory Image				Total
	1 (Unused)	2 (Used/unencr.)	3 (Used/encr.)	4 (Used/reboot)	
Private key (binary)	27	50			*77*
Private key (WIF)					*0*
Public key (binary)	71	155	51	85	*362*
Public key (address)	3	127	62	101	*293*
Labels		96	62	86	*244*
Transaction ID			2		*2*
Passphrase					*0*
File location		18		63	*81*
Total	*101*	*446*	*177*	*335*	*1059*

FIGURE 4. Number of items found in process memory of the Bitcoin Core client by type and memory image.

a: PRIVATE KEYS

All (nine) known private keys were located in binary format in the process memory when the wallet was unencrypted (memory images 1 and 2). When the wallet was encrypted (memory images 3 and 4), no private key could be traced back in memory, not even directly after a transaction was initiated (memory image 4). Furthermore, no occurrence of private keys in Wallet Import Format (WIF) was found. This observation is not surprising, as private keys are stored in binary format in the wallet database and WIF-formatted keys are only created when exporting keys.

b: PUBLIC KEYS AND ADDRESSES

All (nine) known public keys in binary format were located in the process memory in all images. Addresses were only present in memory images 2, 3 and 4, with the exception of the address related to the "default" label. Because of this, it is likely that addresses are only calculated from the binary public key in the wallet when a label is associated with them. Not all public keys and addresses occurred equally often. However, no correlation was found between particular categories (send address, receiving address or payment request) or the usage in transactions on the one hand and the number of occurrences of the associated public keys and addresses.

c: LABELS

Known labels have been found in all memory images in which they could be present (i.e. memory images 2, 3 and 4). All 16 different labels were found, four labels for each category (i.e. message, receiving address, payment request and send address). In memory image 3, in which the application was just opened and no user actions had been taken, the minimum number of occurrences for each label was three. In memory images 2 and 4, in which the user had interacted with the application in various ways, the minimum number of occurrences for each label was four. Apart from these observations, labels for payment requests appeared more often than other types of labels, approximately twice as often (six or more times in memory image 2, eight or more times in memory images 2 and 4). In any case, all labels appeared more than once in process memory. However, because labels are not known upfront and do not follow a fixed pattern, they can be hard to locate for an investigator. This could only be done based on the context of the user labels (see below under Context).

d: TRANSACTION IDs

Transaction IDs were only found in memory image 3. Both search hits corresponded to the Transaction ID of the last transaction initiated from the client.

e: PASSPHRASE

The passphrase used for wallet encryption, was not encountered in process memory.

f: FILE LOCATIONS

Before memory images 2 and 4 were created, a backup of the wallet file was saved to a user-specified file location. The

file name of the backup file was encountered in both memory images. More specifically, the full path of the backup file was present in memory image 2 four times. In memory image 4, the full path of each of both backup location was present once. When scanning for all file paths in the process memory, many other file paths were present as well. Hence, an investigator should scan through these manually to determine whether they are linked to a backup file (e.g. based on file name or file location, e.g. a thumbdrive or user folder).

A total of 14 occurrences of the backup file name (without the file path) was encountered in memory image 2, whereas memory image 4 contained a total of 61 occurrences of backup file name. Of the latter, 30 occurrences referred to the first backup and 31 occurrences to the second backup.

3) FORMAT

The public and private keys in binary format were also present as binary data in memory. All other values, namely addresses, transaction-IDs, labels and file locations, appeared as string values in memory.

4) LOCATION

All values encountered were without exception present in VAD regions with the following characteristics: VadType: VadNone; VadPermissions: PAGE_READWRITE; VadFile: no memory-mapped files; VadFlags: Private Memory. In other words, all values were found in private (non-shared) read/write regions in process memory. No correlation was found between the memory location and VAD on the one hand and the type of data on the other hands. Therefore, no conclusions could be drawn on the co-existence of particular (types of) values in process memory.

5) CONTEXT

The context of the items encountered in process memory are described below per item type.

a: PRIVATE KEYS

Analysis of the direct context in which occurrences of private keys appeared, showed that by most of the occurrences were consistently preceded by the fingerprint 0xf70001d63081d30201010420. At least four instances of each known private key were preceded by this fingerprint. Thus, private keys can be easily retrieved from memory by carving sequences of 32 bytes directly following this fingerprint. Doing so revealed the existence of multiple unknown private keys. This is expected, as not all private keys in the wallet are involved in the user actions during image creation and included in the search item list.

b: PUBLIC KEYS AND ADDRESSES

Through all images, all known public keys in binary format were directly preceded by two different fixed strings, namely key! and keymeta!. At least one occurrence of a public key followed each of these strings.5 Similarly, the fingerprints name'' and purpose'' preceded at least one instance of each known public key address in base58 format across all images. Analysis of the fingerprints showed that these are in fact Berkeley database tags for public keys in the wallet.dat file. Comparison with the data in the respective wallet files confirmed that the memory regions in which the values were uncovered were in fact memory-resident parts of the wallet file. Hence, it is very likely that the same information can also be extracted from the wallet file. In case no wallet file is available to the investigator, extracting all binary public keys and public key addresses from process memory is a straightforward process. Public keys in binary format have a fixed length6 and public key addresses have a predictable length and format. All such values following the fingerprints mentioned above will yield the correct results.

c: LABELS

No fingerprint could be identified to systemically extract labels from process memory with one exception. In some cases, the public key address and its corresponding user label co-occurred in the same memory region. More specifically, for a given public address having the name'' tag (see above), its corresponding label consisted of the first human-readable string preceding this address. However, systematic retrieval of this information is not trivial as the label is variable length and a variable number of bytes, typically 4 to 7, was present between label and the public address.

d: TRANSACTION IDs

Only two transaction IDs were traced back in memory. No relevant findings could be derived from these memory context, except for the fact that one of the IDs was preceded by the string ''AddToWallet''. This string occurs in the debug.log when a new transaction ID is added to the wallet. This finding gives rise to the idea that the memory presence of transaction IDs is linked to logging. As such, this information can be better retrieved by analysis of the memory-resident debug.log file as described in the next paragraph.

e: FILE LOCATIONS

No conclusions could be drawn from the context of the memory locations of the hits on file location data (paths and file names).

B. FILES

The Bitcoin Core processes in each image were explored on open file handles to forensically relevant files. Two application files are specifically important in this respect: the wallet file (wallet.dat) and the application log file (debug.log). The wallet file is the keystore containing the bitcoin keys and all user data (as described earlier). In an encrypted wallet, the private keys are only readable when the correct passphrase is known. However, all public keys, transaction data and user labels are not encrypted and form a valuable resource to the investigator. The debug.log file, on the other hand, can be interesting due to two reasons: it shows the transactions initiated by the wallet with their date and time and, possibly

more important, it shows the full file location of any backup created by the user. Hence, it is possible to identify wallet backups on user locations and attribute these to the use of the bitcoin client.

Analysis of the handle table8 was used to determine which application files might be present in memory. This analysis showed that a handle to the debug.log file was open in each process instance, but an active handle to the wallet file was only present in memory image 1 and 3. The memory-resident instances of the wallet and debug log files could be easily exported from the images using Volatility's dumpfiles plugin. However, this plugin traverses the handle table of the process to find the relevant memory regions. Since no active handles to the wallet.dat files were present in memory image 2 and 4, it was not possible to export this file from these images. It may even be the case that no memory-resident instance of this file was present in these images at all.

The log file is human-readable and can be searched for transaction IDs and backup locations (see above). Comparing the memory-resident version of this file with the on-disk instance showed that the memory-resident file only contained the last entries of the log file and is incomplete. Hence, not all available log information was present in the memory-resident version of the debug.log file.

In the cases where it was possible to export application files, analysis of these files is relatively trivial. Where it was possible to export a wallet file (image 3), this wallet file can be copied to the application directory of an instance of Bitcoin Core. When the application is opened, the wallet file is than loaded without issues and can be investigated via the Bitcoin Core graphical user interface.

Additional file metadata (e.g. mac times) of the application files can be obtained by memory-resident information in the MFT table, which is available through the mftparser Volatility plugin. In all images, both the wallet and log files had records in the MFT table. Apart from the application files discussed above, any (memory-resident) backups of the wallet may be of importance. None of the images had open handles to the wallet backups created. In theory, wallet backups can be exported to a user location with a user determined file name. Hence, it can be hard to identify a particular file as a wallet backup. Apart from the presence of the file path en location in process memory, the debug.log is still the best source as any backup is registered in this file.

C. REGISTRY KEYS

Registry keys used by the Bitcoin Core client were identified using the handles plugin. Registry keys on the following locations were in use by the application:

- HKEY_CLASSES_ROOT\bitcoin
- HKEY_LOCAL_MACHINE\SOFTWARE\Bitcoin Core(64-bit)
- HKEY_CURRENT_USER\Software\Bitcoin\Bitcoin-Qt
- HKEY_CURRENT_USER\Software\BitcoinCore (64-bit)

The presence of these keys serve as an indicator for the presence of an active instance of the Bitcoin Core client. Moreover, relevant configuration for the bitcoin client could be obtained from these keys. For this purpose, the location HKEY_CURRENT_USER\Software\Bitcoin\Bitcoin-Qt contained most relevant application settings, e.g. the application directory, proxy settings and transaction fee settings.

D. CONNECTIONS

The netscan plugin shows an overview of the connection objects encountered in the memory image. In all images, IP addresses of neighboring bitcoin peers were visible. These connections can be identified based on the TCP port number 8333. Such connections can serve as an indicator for the presence of an active bitcoin client, but further forensic usefulness is yet unclear.

V. FINDINGS FOR ELECTRUM

A. PROCESS MEMORY

Similar to the previous discussion on the findings for Bitcoin Core, we now present the findings for Electrum.

1) DATA EXCLUSION

All occurrences of known forensic values were traced back in the memory images with the Electrum client (memory images of scenarios from 1 to 3 in Table 2, which will be referred to as memory images 5, 6 and 7, respectively). The results from the process vmstoolsd.exe were omitted because this client will not be running on a typical system setup. This process is associated with VMWare Tools running in the virtual machine. Similarly, occurrences of known artefacts in the kernel memory were also omitted from the analysis, since it is not clear why the values found in the kernel memory were present and whether they can be attributed to typical application use. For instance, during configuration of the application, information such as labels and Bitcoin addresses, were copied to and from the virtual machine. The occurrences in the kernel memory of these labels and addresses can be a result of those actions.

Two instances of the Electrum.exe process were found in the process list of each memory image, in which one process was a child process of the other process. No forensic data was found in the parent instance and hence, only the child instance was investigated. Scanning process memory for separate words from the seed phrase resulted in an overwhelming number of false positives, which can be expected. Hence, only full complete seed phrases are taken into account.

2) OCCURRENCES

The number of occurrences of known forensic artefacts are shown in Figure 5.

a: PRIVATE KEYS

No private key or checksum of private keys was located in process memory.

Item Type	Memory Image			Total
	5 (Unused)	6 (Used/encrypted)	7 (Used/reboot)	
Private key (binary)				*0*
Private key (WIF)				*0*
Public key (binary)	208	360	162	*730*
Public key (address)	241	667	428	*1336*
Labels		135	95	*230*
Transaction ID		115	86	*201*
Seed phrases	2			*2*
Passphrase				*0*
File location		39		*39*
Total	*451*	*1316*	*771*	*2538*

FIGURE 5. Number of items found in process memory of Electrum by type and memory image.

b: PUBLIC KEYS AND ADDRESSES

In memory images 5, 26 distinct known private keys were found back in binary format with their corresponding public key address. In memory images 6 and 7, all 41 known values of public key addresses were found as well as all 37 known binary values for public keys (not all addresses are native to the wallet and hence, no corresponding binary key is known or present, e.g. for public key addresses for outgoing transactions). Not all values occurred equally often. However, no correlation could be detected between the usage of a particular key (e.g. in a payment request) and the number of occurrences. In any case, many more occurrences were found in the application which was recently used (memory image 5) compared to the application which was just opened (memory images 5 and 7). In addition, the master public key could be traced back in process memory in all three images.

c: LABELS

All known labels were present in the process memory in memory images 6 and 7. In memory image 7, labels linked to payment request occurred much more often than other labels, but this was not the case after a reboot (memory image 7). Furthermore the label associated with a recent outgoing transaction occurred much more often in an image.

d: TRANSACTION IDs

All three known transaction IDs were found at various times in process memory on memory images 6 and 7. Each transaction ID occurred about 33 times on average in each image.

e: PASSPHRASE

The passphrase for the wallet was not encountered in process memory.

f: FILE LOCATIONS

The full path of the backup file location was only found in memory image 6, whereas only the file name occurred 38 times in this image. No reference to the backup file was found in memory image 7.

3) FORMAT

Almost all data appeared as string values in memory. However, public keys associated with key pairs involved

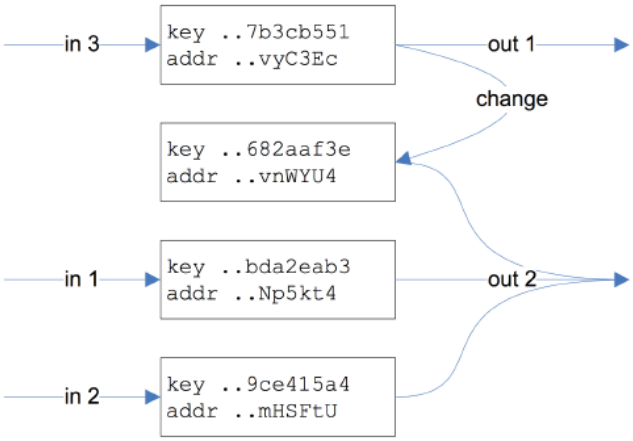

FIGURE 6. Public keys associated with the wallet's transactions, which occurred as binary values in memory.

in the wallet's transactions were an exception. More specifically, the public key of the input transaction(s) for outgoing transactions initiated just before the image was made occurred in the binary format. The transaction flow through the wallet is depicted schematically in Figure 6.

Three incoming transaction were initiated to three different addresses in the wallet (address: in memory image 5, address: in memory image 6 and address: in memory image 7). Just before memory image 6 was created, one outgoing transaction was initiated (address: out 1) which had one input to public key ..7b3cb551. This key was found in binary format in memory image 6. Similarly, just before memory image 7 was created, one outgoing transaction was initiated (address: out 2) which had three inputs. The three public keys associated with these inputs were found in binary format in memory image 7. Based on these findings, the presence of binary public keys in process memory suggested that this key is actively involved in the transaction flow through the wallet.

4) LOCATION

Similar to the Bitcoin Core client, all forensic values located in Electrum's process memory were present in VAD regions with the following characteristics: VadType: VadNone; VadPermissions: PAGE_READWRITE; VadFile: no memory-mapped files; VadFlags: Private Memory. In other words, all values were found in private (non-shared) read/write regions in process memory. There was one exception: one occurrence of the file name of the wallet backup file (not the full path) was found in a memory-mapped DLL file (shlwapi.dll). Naturally, the properties of the VAD region of this occurrence had different properties. No correlation was found between the memory location and VAD on the one hand and the type of data on the other hands. Therefore, no conclusion could be drawn on the co-existence of particular (types of) values in process memory.

5) CONTEXT

Analysis of the direct context in which artefacts appeared showed that most were present as part of JSON-formatted data. A closer analysis revealed that those occurrences were part of memory-resident data from the Electrum wallet file. The wallet file consists of a JSON file in which the user and key data is ordered with various tags, e.g. addr_history and master_public_keys. Scanning process memory with these tags provides a straightforward way to extract useful data from process memory in cases were an on-disk representation of the wallet file is unavailable.

A comparison between memory images 5 to 7 revealed that only memory image 6 had a complete memory-resident representation of the wallet file. Memory images 5 and 7 had only minor parts of the wallet data in memory, although this data is not associated with a memorymapped file object. In any case, the artefacts present can be traced back only by using the JSON tags from the wallet format. For example, wallet transaction data could be retrieved also in memory image 7 by scanning for the tag ''txi'' and analyzing the data directly following this tag. In some cases, the relevant forensic values were preceded by another fixed string. In particular, all public key addresses associated with the key pairs in the wallet were preceded by the string blockchain.address.subscribe. According to the Electrum documentation [1], this string is associated with the protocol between Electrum client and Electrum server. With this command, the client subscribes for updates on the address passed to the central server. By doing so, the client keeps track of the current status of the addresses in its wallets. As such, any occurrence of a Bitcoin preceded by this string is a clear indication that the key pair of this address is present in the wallet. No conclusion could be drawn from the context of binary public keys and as a consequence, these are hard to trace back in memory.

B. FILES

The Electrum wallet file is the most important component from a forensic perspective. It contains the wallet key material, user data and transaction history. The default location of the wallet file is in the wallet's directory in USER_DIRECTORY\AppData\Roaming \Electrum\. Additionally, this directory contains various files with (forensically) interesting application data, namely: the file contacts containing a list of the wallet's contacts with their corresponding addresses; the file config containing the console history and recently opened wallets; the file recent_servers containing an overview of recently used Electrum servers.

While examining the handle tables from the Electrum processes, no reference to wallet files or other relevant Electrum files was found. Hence, extracting these files using Volatility's dumpfiles plugin was not possible. However, analysis of the memory-resident MFT table revealed the entries for all these files. In addition to the metadata available in the MFT table, file content of the config, contacts and recent_servers files was found in the $DATA section of MFT records due to their small file size.

No reference to wallet backup files was found in the handle tables in any of the images. Hence, wallet backups can only be attributed to the Electrum process by manually analyzing the file paths in process memory as mentioned above. Furthermore, only memory image 2 contained an entry for the wallet file in the MFT table.

C. REGISTRY KEYS

No application-specific registry key was in use by the Electrum application.

D. CONNECTIONS

The Electrum application opens multiple connections TCP port 50002 and one connection on TCP port 443, to the seed server. These connections are visible in the memory image with the netscan plugin and can serve as an indicator for the presence of the Electrum application.

VI. DISCUSSION

A. BITCOIN CORE

The Bitcoin Core wallet is not encrypted by default. Without encryption, the private keys of the wallet are all present in process memory and easily retrievable with the preceding fingerprint 0xf70001d63081d30201010420. The private keys consist of all 32 byte values following this fingerprint. However, when the wallet was encrypted, no unencrypted private keys could be found in memory. Additionally, the wallet passphrase was not found in the memory. Public keys, public addresses and labels were always found in the process memory, regardless of the state of the application. The memory-resident instances of these values appeared to be parts of the wallet file in the Berkeley database format. The keys used in this database format directly preceded all known values, key! and keymeta! for public keys in binary format and name'' and purpose'' for labels. With these fingerprints, all public keys1 and labels could be forensically recovered from the process memory.

Only some instances of Transaction IDs related to the wallet's transactions were found in the memory. The instances found in one image had links to the debug.log file. Possibly, the fingerprint AddToWallet might be used to find these values, but this could not be established with certainty. The file location of wallet backups were found in the memory amongst many other file paths and file names, but not in all images. In addition, no open handle to backup locations was linked to the process instance. Hence, it might be hard (or even impossible) to identify a specific file path and name in process memory as a backup location in the first place. The best heuristic for finding wallet backup locations from values in the process memory would be to create a list of full file paths occurring in memory and checking for user-specific file locations and file names.

Many, if not all, artefacts found in the process memory are present in the wallet.dat and debug.log files present on disk. The wallet file stores all public keys, addresses, labels and transaction data. In addition, transaction IDs and wallet backup locations are both logged in the debug.log file. If no on-disk instance of these files is available to the investigator, then it might be possible to extract a memory-mapped instance of these files. However, this appeared not to be possible in all cases as the wallet file is not always memory-resident. Furthermore, only the last part of the debug.log file was found in the memory. The memory-resident Master File Table holds valuable metadata on the application files. Finally, VAD properties do not appear to be useful as a heuristic for retrieving specific forensic artefacts. Also, registry keys and connections related to the Bitcoin Core application were found, but the forensic relevance of these data appears limited.

B. ELECTRUM

As previously indicated, in Electrum, the wallets are encrypted by default. No private key or encryption passphrase was located in any of the images. The seed phrase from which the private keys are derived was found in memory just after wallet initialization, but not after use or reboot. Public keys, addresses, labels and even transaction IDs were found in process memory, including the master public key from which all public keys in the wallet are derived. Analysis of the values in memory showed that these were part of memory-resident parts of the wallet file in JSON format. As such, the tags used in the wallet file can also be used to trace back wallet contents in process memory. Moreover, all public keys in the wallet appeared to be monitored by the client on the Electrum server via the Electrum protocol. In memory, such addresses are directly preceded by the string value blockchain.address.subscribe. All public keys in an Electrum wallet can easily be extracted from process memory using this fingerprint.

It must be noted that almost all data of forensic interest in Electrum's memory are processed as strings. However, public keys in binary format were also found when they were involved in an outgoing transaction. There is, however, no straightforward way of retrieving these keys from memory. It is possible to pre-calculate a list of binary public keys from the master public key and trace these back in process memory. If keys are present in binary format, then this is an indication that it has been involved in an outgoing transaction. The full path of the wallet backup location was only found in process memory in one of the images and Electrum has no handle to wallet backups. Hence, it will be challenging to determine whether a backup has been made, and if so, what the location of the wallet backup was.

All keys, addresses, labels and transaction data were also found to be available in Electrum's wallet files on disk. Electrum has no open handles to the wallet file and, as a consequence, this file could not be exported from the memory image. However, as noted before, analysis of data in process memory revealed that the wallet file can be completely or partially present in memory, most likely as residual data from memory-resident instances of the wallet file. Furthermore, other Electrum application files also held relevant data, most notably the config, recent_servers and contacts files. These files' metadata and their contents were present in the memory-resident Master File Table (MFT).

Finally, VAD properties appeared to be useless as a heuristic for retrieving specific forensic artefacts. Also, some connections related to the Electrum application were found, but the forensic relevance of these data seems limited. Table 3 summarizes the results of the experiments.

A number of forensic artefacts were located in the process memory of the Electrum client. In particular, extracting public keys, addresses and user labels was relatively straightforward using tags from the wallet file format. For the Electrum clients, transaction IDs could also be traced back in memory. Private keys and seed phrases, however, could only be extracted in specific circumstances. Private keys were only available for the Bitcoin Core client with an unencrypted wallet. The seed phrase was only available for an unused and recently initialized Electrum client. Passphrases were not encountered at all.

The artefacts found in the process memory can help to attribute Bitcoin public keys and addresses and the transactions on these addresses to the user of the Bitcoin clients involved. Subsequently, these findings can then be combined with findings from Blockchain forensics to attribute the flow of (criminal) value through the Bitcoin system. The presence of user labels can also provide forensic investigators additional clues on the particular use of Bitcoin addresses, contacts and transactions.

However, given the absence of seed phrases, passphrases and private key material in the process memory of used, encrypted clients, it is highly unlikely that process memory analysis will make it possible to seize value from the client's wallet. This will only be possible in the case of an unencrypted Bitcoin Core wallet, but any security-aware user will make sure (s)he will not leave this wallet unencrypted.

Information uncovered in this research could be used to extract private and public keys, addresses and labels from process memory. It may also be possible to acquire relevant memory-resident file and registry data associated with the Bitcoin clients. For instance, it was possible to export complete or partial application files from memory images. For the Electrum client, the contact list and other application files were present in the memory-resident MFT table.

Memory-resident data are also of interest in a forensic investigation, particularly when the wallet file and other application files are not available from the disk. For instance, a security-aware user might choose to protect his Bitcoin assets by storing the wallet in an encrypted container or on an external storage media not available to the investigator. We also remark that conducting a detailed disk forensic analysis would require more time, in comparison to undertaking memory forensics. Memory-mapped files can also be

exported from the memory and analyzed as disk artefacts. For Bitcoin Core, the wallet file thus might be available, which in turn can be loaded in Bitcoin Core and reviewed via the GUI. Also, a buffered part of the debug.log file can easily be exported from memory images with the Bitcoin Core process. In Electrum, application files are likely not available as these are not encountered as memory-mapped executive objects governed by Windows' Object Manager.

It may also be possible to trace wallet files and application files by manually scanning file paths in process memory for wallet backup locations. It is also possible to extract file contents from application files from the MFT table.

VII. CONCLUSION AND FUTURE WORK

In this paper, we forensically examined Bitcoin Core and Electrum, two popular Bitcoin clients. We demonstrated that data of forensic interest can be extracted from memory by scanning the process memory for fingerprints identified in this research or by searching fixed patterns with regular expressions (e.g. Bitcoin addresses or file paths). Despite the potential to use the located evidence to attribute Bitcoin transactions to the user of the computer, it is unlikely the evidence located will directly result in the seizure of the assets stored in the wallet. It must be noted, however, that most data found in memory are also available in application and wallet files on disk, with a few exceptions. As such, process memory analysis is potentially beneficial to a forensic investigation, particularly when application and wallet files are not available.

Findings from this research contribute to an improved forensic understanding of Bitcoin clients in several other respects. First, unlike earlier studies, we focus not only on string values, but also binary-formatted values stored as string or binary. For instance, Electrum is implemented in Python and the wallet consists of (string-based) JSON data. With the exception of keys involved in transactions, all in-memory artefacts were string-based data.

Future research includes forensically examining newer (versions of) Bitcoin clients, with the aims of proposing a forensic taxonomy of Bitcoin client artefacts, similar to the approaches undertaken in [19]–[22].

REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Jan. 2017. [Online]. Available: http://Bitcoin.org/Bitcoin.pdf

[2] A. Montanez, "Investigation of cryptocurrency wallets on iOS and Android mobile devices for potential forensic artifacts," Dept. Forensic Sci., Marshall Univ., Huntington, WV, USA, Tech. Rep., 2014.

[3] Accessed: Jan. 2017. [Online]. Available: http://freico.in/

[4] Accessed: Jan. 2017. [Online]. Available: https://peercoin.net/

[5] Accessed: Jan. 2017. [Online]. Available: https://www.usmarshals.gov/news/chron/2015/100515.htm

[6] Accessed: Jan. 2017. [Online]. Available: https://www.wsj.com/articles/in-the-Bitcoin-era-ransomware-attacks-surge-1471616632

[7] K.-K. R. Choo, "Cryptocurrency and virtual currency: Corruption and money laundering/terrorism financing risks?" in *Handbook of Digital Currency*, D. K. C. Lee, Ed. New York, NY, USA: Elsevier, 2015, pp. 283–307.

[8] D. Sui, J. Caverlee, and D. Rudesill. *The Deep Web and the Darknet: A Look Inside the Internet's Massive Black Box*. Accessed: Jan. 2017. [Online]. Available: https://www.wilsoncenter.org/sites/default/files/stip_dark_web.pdf

[9] T. Pantas. *BlockChain Technology*. Accessed: Jan. 2017. [Online]. Available: http://scet.berkeley.edu/wp-content/uploads/Blockchain Paper.pdf

[10] S. Meiklejohn, M. Pomarole, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of Bitcoins: Characterizing payments among men with no names," in *Proc. Conf. Internet Meas. Conf. (IMC)*, New York, NY, USA, 2013, pp. 127–140.

[11] M. Spagnuolo, F. Maggi, and S. Zanero, "BitIodine: Extracting intelligence from the Bitcoin network," in *Financial Cryptography and Data Security*. New York, NY, USA: Springer, 2014, pp. 457–468.

[12] Chainanalysis Inc. *Chainanalysis*. Accessed: Jan. 2017. [Online]. Available: https://www.chainalysis.com

[13] D. Ferrin. *Numisight*. Accessed: Jan. 2017. [Online]. Available: http://www.numisight.com

[14] M. Möser, R. Bohme, and D. Breuker, "An inquiry into money laundering tools in the Bitcoin ecosystem," in *Proc. IEEE eCrime Res. Summit (eCRS)*, Sep. 2013, pp. 1–14.

[15] Magnet Forensics Inc. *Bitcoin Forensics Part II: The Secret Web Strikes Back*. Accessed: Jan. 2017. [Online]. Available: https://www.magnetforensics.com/computer-forensics/Bitcoinforensics-part-ii-the-secret-web-strikes-back/

[16] Magnet Forensics Inc. *Bitcoin forensics—A Journey into the Dark Web*. Accessed: Jan. 2017. [Online]. Available: https://www.magnetforensics.com/computer-forensics/Bitcoinforensics-a-journey-into-the-dark-web/

[17] M. Doran. (2015). A forensic look at Bitcoin cryptocurrency. SANS Inst. InfoSec Reading Room. Accessed: Oct. 2017. [Online]. Available: https://www.sans.org/reading-room/whitepapers/forensics/forensic-bitcoin-cryptocurrency-36437

[18] *Volatility Foundation, Volatility v2.5*. Accessed: Jan. 2017. [Online]. Available: http://www.volatilityfoundation.org/#!releases/component_71401

[19] A. Azfar, K.-K. R. Choo, and L. Liu, "Forensic taxonomy of Android social apps," *J. Forensic Sci.*, vol. 62, no. 2, pp. 435–456, 2017, doi: 10.1111/1556-4029.13267.

[20] A. Azfar, K.-K. R. Choo, and L. Liu, "Forensic taxonomy of Android productivity apps," *Multimedia Tools Appl.*, vol. 76, no. 3, pp. 3313–3341, 2017, doi: 10.1007/s11042-016-3718-2.

[21] A. Azfar, K.-K. R. Choo, and L. Liu, "An Android communication app forensic taxonomy," *J. Forensic Sci.*, vol. 61, no. 5, pp. 1337–1350, 2016.

[22] A. Azfar, K.-K. R. Choo, and L. Liu, "Forensic taxonomy of popular Android mHealth apps," in *Proc. 21st Amer. Conf. Inf. Syst. (AMCIS)*, Aug. 2015, pp. 13–15.

[23] *Bitcoin Core Developers, Bitcoin Core*. Accessed: Jan. 2017. [Online]. Available: https://bitcoincore.org/

[24] T. Voegtlin. (2011). *Electrum Application*. Accessed: Jan. 2017. [Online]. Available: https:// electrum.org/#home

LUUC VAN DER HORST received the M.Sc. degree in forensic computing and cybercrime investigations from the School of Computer Science, University College Dublin and the MA degree in linguistics from the Radboud University, Nijmegen, The Netherlands. He is currently a forensic investigator with the Dutch National Police, The Netherlands.

KIM-KWANG RAYMOND CHOO (SM'15) received the Ph.D. degree in information security from the Queensland University of Technology, Australia. He currently holds the Cloud Technology Endowed Professorship with the University of Texas at San Antonio, and is a Fellow of the Australian Computer Society. He was named one of ten Emerging Leaders in the Innovation category of The Weekend Australian Magazine/Microsoft's Next 100 series in 2009, and was a recipient of various awards, including the ESORICS 2015 Best Research Paper Award, the Highly Commended Award from Australia New Zealand Policing Advisory Agency, the British Computer Society's Wilkes Award, the Fulbright Scholarship, and the 2008 Australia Day Achievement Medallion. He serves on the editorial board of *Cluster Computing*, *Digital Investigation*, IEEE Access, IEEE Cloud Computing, IEEE Communications Magazine, Future Generation Computer Systems, *Journal of Network and Computer Applications*, and PLoS ONE. He also serves as the Special Issue Guest Editor of *ACM Transactions on Embedded Computing Systems* (2017), ACM *Transactions on Internet Technology* (2016), *Digital Investigation* (2016), *Future Generation Computer Systems* (2016, 2018), IEEE Cloud Computing (2015), IEEE Network (2016), IEEE Transactions on Cloud Computing (2017), IEEE Transactions on Dependable and Secure Computing (2017), *Journal of Computer and System Sciences* (2017), *Multimedia Tools and Applications* (2017), *Personal and Ubiquitous Computing* (2017), *Pervasive and Mobile Computing* (2016), *Wireless Personal Communications* (2017).

NHIEN-AN LE-KHAC (M'07) received the Ph.D. degree from the Institute National Polytechnique Grenoble, France. He is currently a Lecturer with the School of Computer Science, University College Dublin (UCD). He is currently the Director of UCD Forensic Computing and Cybercrime Investigation Programme-an International Programme for the law enforcement officers specializing in cybercrime investigation. He has published over 90 scientific papers in international peer–reviewed journal and conferences in related disciplines.