Advanced Techniques for Reconstruction of Incomplete Network Data

Petr Matoušek(✉), Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmeť, Filip Karpíšek, and Martin Vymlátil

Brno University of Technology, Božetěchova 2, Brno, Czech Republic
{matousp,ipluskal,rysavy,ivesely,ikmet,ikarpisek}@fit.vutbr.cz,
xvymla01@stud.fit.vutbr.cz
http://www.fit.vutbr.cz

Abstract. Network forensics is a method of obtaining and analyzing digital evidences from network sources. Network forensics includes data acquisition, selection, processing, analysis and presentation to investigators. Due to high volumes of transmitted data the acquired information can be incomplete, corrupted, or disordered which makes further reconstruction difficult. In this paper, we address the issue of advanced parsing and reconstruction of incomplete, corrupted, or disordered data packets. We introduce a technique that recovers TCP or UDP conversations so they could be further analyzed by application parsers. Presented technique is implemented in a new network forensic tool called Netfox Detective. We also discuss current challenges in parsing web mail communication, SSL decryption and Bitcoins detection.

Keywords: Network forensic tools · TCP reassembling · Traffic reconstruction · Web mail · Bitcoin · SSL encryption

1 Introduction

Network forensics is an emerging area of digital forensics connected with the rapid network development. Many services and digital transactions are transmitted over the Internet where criminal activities and security incidents also occur. Network forensics provides post-mortem investigation of unlawful behavior using special tools that reconstruct a sequence of events occurred at the time of the attack. This reconstruction depends only on a captured network data. In some cases, these data are incomplete, corrupted, or out of order. In order to analyze the original communication using an incompletely captured data, advanced techniques of reconstruction and communication recovery are needed. Reconstruction of TCP streams is essential for any network forensic tool [1]. If the TCP reassembling fails, application data cannot be properly analyzed.

Recovery of incomplete data in network forensics is a similar task to data recovery from damaged media, e.g., hard drives, CDs, or DVDs. If some data are missing, it can be either replaced by empty data units or approximated

J.I. James and F. Breitinger (Eds.): ICDF2C 2015, LNICST 157, pp. 69–84, 2015.
DOI: 10.1007/978-3-319-25512-5_6

from known data. The goal is to provide enough data enabling reconstruction of the original content. To guarantee an admissibility of forensic results newly introduced data must be unambiguously distinguished from the original ones.

In this work, we deal with the analysis and reconstruction of incomplete or damaged network data. Our research includes the development of heuristic techniques that can detect incomplete or corrupted data on network and transport layer and restore original sessions that can be further analyzed using usual application parsers. The proposed technique was implemented in a new network forensic tool *Netfox Detective*.

1.1 Contribution

The main contribution of this paper addresses practical issues connected with network data reconstruction and proposes advanced techniques for parsing and recovery of network conversations. These techniques in combination with advanced application recognition methods increase the accuracy of content reconstruction. We also explain several issues connected with application analysis, especially with web mail services, SSL communication and Bitcoin transactions. We evaluate the implementation of proposed methods and compare them with other tools.

The paper is organized as follows: section two surveys current approaches and results in the domain of network forensic tools; section three examines issues related to network data parsing and reconstruction with focus on TCP reassembling and Layer 7 (L7, application) data reconstruction; section four deals with application detection and content analysis, which is demonstrated using examples of reconstruction of web mail, SSL traffic, and bitcoin transactions.

2 Related Work

There is a wide range of tools for network monitoring and forensics, i.e., Network Security and Monitoring tools (NSMs) and Network Forensic Analysis Tools (NFATs). NSMs include network analyzers (Wireshark, tcpdump), IDS systems (snort, Bro), fingerprinting tools (nmap, p0f), and others [2]. NFATs have similar functionality as NSMs, in addition, they also assist in a network crime investigation. They capture an entire network traffic and allow an investigator to analyze it and reconstruct the original communication. Most of the NFAT tools are proprietary, nevertheless, open source NFATs also exist, e.g., PyFlag, Network Miner, or Xplico.

In theory, parsing the network communication is straightforward. However, incompleteness and corruption of communication requires new methods involving robust parsers and complex recovery procedures. Surveys of different network forensic frameworks can be found in [2,3]. These papers discuss various approaches to network forensics, major challenges, and list available tools. In our paper, we mostly focus on techniques of network data parsing and recovery.

There are not many published works describing techniques incorporated in NFAT implementations, partly due to the protection of intellectual properties of the tools. An exception is Cohen [1] that describes several challenges connected with the stream reassembling (termination of streams, out of sequence packets, missed packets) and the combination of streams into conversations. In our work, we deeply examine issues that are essential for every network forensic tool. In addition to [1], we present an algorithm that deals with these issues, and also works with sequence number overflow, which is not discussed by other authors. A detailed description of TCP reassembling is analyzed by Paxson in [4]. However, Paxson focuses on robustness of TCP reassembling in the presence of adversaries that is out of the interest of this paper.

3 Data Parsing and Reconstruction

NFATs are designed to parse captured data, process packet headers and reconstruct high-level protocol units. Application data are regularly transmitted using TCP or UDP protocols over IP networks. By definition, IP communication does not provide reliable data exchange [5]. Application data are segmented into TCP packets and encapsulated into IP datagrams. Furthermore, IP datagrams can be fragmented into smaller IP datagrams when required by an underlying link-layer technology. The main goal of an NFAT is to extract and reconstruct original application data from possibly incomplete captured collection of IP datagrams. The method for assembling IP packet-based communications into conversations is based on the following assumptions:

- An application conversation is distinguished by a pair of IP addresses, transport ports and a protocol type. The conversation consists of a pair of flows because the most of sessions are bi-directional.
- The beginning of a TCP session is identified by a synchronization TCP segment (SYN flag). A TCP segment with FIN/PSH/RST flag closes the session.
- A TCP session consists of a collection of TCP segments each associated with a sequence number. A sequence number determines an offset of the segment content in the TCP stream [6].
- An application message can be transmitted in one or more TCP segments. Receiver must reassemble several TCP segments to obtain the original message.
- The IP fragmentation happens independently on the TCP segmentation. The IP defragmentation has to be accomplished before the application content reassembling.

3.1 Challenges in TCP Reassembling

During our research of network data analysis, following challenges connected with reassembling of TCP sessions have been identified:

– *Missing FIN packets or overlapping of TCP conversations.*
Regularly, ephemeral source ports are dynamically assigned by OS to clients whenever a communication socket is created [7, p. 99]. It helps to distinguish several TCP sessions originating from the same node and targeting the same remote process. When the client finishes communication, these ports can be reused. Usually, the port number is not reused until the pool of ephemeral ports is exhausted. NFAT can exploit this behavior to recognize different TCP sessions safely. However, if there is a NAT translation along the communication path observable port numbers can be reused quickly. In such case, different TCP sessions can receive the same key fields within a relatively short period. While end systems and NAT can accurately track the use of port numbers, for NFAT system it may pose a problem as there is a very short interval between two TCP sessions with the same identification. NFAT can proceed as follows:
  1. FIN segment can determine closing of the first session segment while SYN segment defines a new TCP session;
  2. if these segments are missing in a captured collection, a flow needs to be detected by analyzing sequence numbers;
  3. if sequence numbers of two sessions overlap, the analysis of timestamps of expected L4 packets have to be carried out.
– *Combination of two L7 flows into a L7 conversation.*
NFATs try to reconstruct original bi-directional communication between applications. If more TCP conversations use the same IP addresses and ports (see NAT problem above), these ports are not sufficient to unambiguously combine corresponding L7 flows into a whole L7 conversation. The proposed solution suggests considering initial TCP sequence numbers. TCP three-way hand-shake starts with sending three synchronization segments between a sender and a receiver. The sender sends a SYN segment with his initial, randomly chosen, sequence number. The receiver replies with an SYN+ACK segment transmitting receiver's initial sequence number and sender's next sequence number. Based on hand-shake analysis, we can match initial TCP sequence numbers of every L7 flow and its opposite L7 flow, which is necessary to create bi-directional L7 conversation based on L4 header data only. If the hand-shake is not captured, L7 flows are considered as one-directional L7 conversations.
– *TCP sequence number overflow.*
Network data parsing and analysis is mostly based on a chronological order of packets in the flow using their sequence numbers. According to RFC 793 [6], sequence numbers occupy space up to $2^{32} - 1$ Bytes, which gives possibility to transmit maximum 4.29 GB data. This value seems large enough to avoid sequence number overflow. However, since initial sequence numbers are generated randomly, maximum data size is lower than this theoretical value. Figure 1 shows a snapshot of the distribution of maximum TCP message sizes based on randomly generated initial sequence numbers as observed on 14,000 TCP sessions. The picture does not show full distribution range. TCP sessions with possible payload greater than 500 MB are excluded, because of their irrelevance for our study. However, these data show that TCP sequence

number overflow should be taken seriously. For example, we can see that the sequence number would overflow in 0,12 % of TCP sessions with payload up to 5 MB. This situation can be solved by multi-pass processing of an L4 conversation and matching incomplete TCP sessions without SYNs when their initial sequence numbers are closed to 2^{32}.

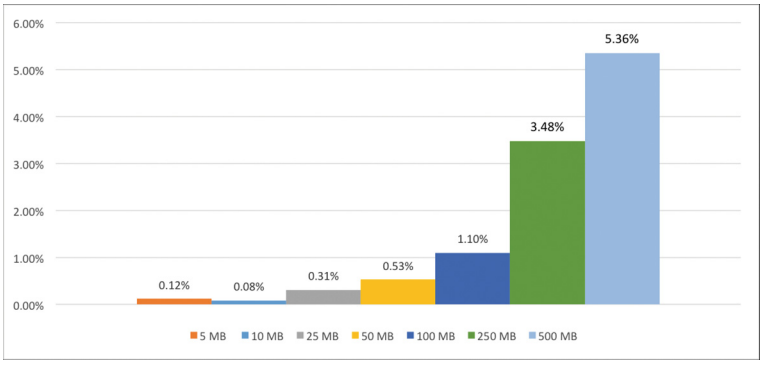

Fig. 1. Probability of TCP Seq numbers overflow related to maximal L7 payload size.

3.2 Building L7 PDUs from the PCAP File

The process of network data parsing starts with the tracking of L3 conversations based on sender's and receiver's IP addresses, see Fig. 2. Further, L4 conversations are identified using port numbers and L4 protocol type, than L7 conversations are created. In case of UDP protocol, two UDP sessions running between the same pair of ports cannot be distinguished. For example, SIP applications regularly employ the same source and destination ports, e.g., 5060, for all SIP conversations. Therefore, a L4 UDP conversation is considered to be a L7 conversation.

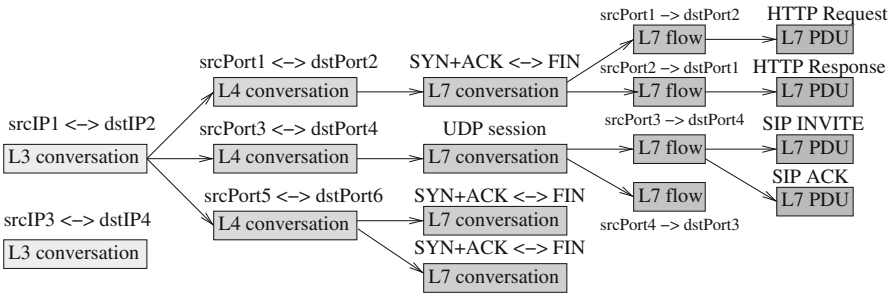

Fig. 2. Extraction of L7 PDUs from input packets.

In case of TCP protocol, the TCP reassembling is the key element in reconstruction. If all data have been properly captured, TCP reassembling is a simple task that involves port numbers, TCP sequence and acknowledgment numbers. If some packets are missing, a following procedure implementing our heuristic method can be applied to any network data. The procedure uses three heuristic parameters: *MaxLost*, which represents the maximal length of missing data that can be restored, *MaxTime*, describing the maximal permitted time delay between two consequent packets using timestamps, and *MaxPayload*, representing the maximum payload size in a TCP packet. Based on our experience, we use $MaxLost = 4kB$ and $MaxTime = 600\ sec$[1]. *MaxPayload* is computed on-the-fly as the length of the TCP packet with the maximal size of a payload in the L7 flow. Thus, application messages are built from captured data using the following steps:

1. Select L4 flows and sort packets using their sequence numbers.
2. Process each L4 flow and create L7 flows using TCP handshake. Start with the first SYN packet.
   (a) Create a new L7 PDU if does not exist or if a previous L7 PDU was closed.
   (b) Check packet sequence number Seq_{i+1}.
   (c) If $Seq_{i+1} \neq Seq_i + PS_i$ (PS stands for a payload size obtained from the packet header), i.e., the expected packet is missing, check timestamps TS and sequence numbers Seq as follows:
      i. If $TS_{i+1} - TS_i \leq MaxTime$ and $Seq_{i+1} - Seq_i \leq MaxLost$ then a virtual packet will be created to replace the missing packet.
      ii. If $TS_{i+1} - TS_i \geq MaxTime$ and $Seq_{i+1} - Seq_i \leq MaxLost$ then there is an overlapping of TCP sessions because $i+1$ packet belongs to a different L7 flow. Skip this packet and proceed with the next one.
      iii. If $Seq_{i+1} - Seq_i \geq MaxLost$ then there are too many missing data. The flow cannot be fully restored. Close it and proceed with next SYN packet.
   (d) If $Seq_{i+1} = Seq_i + PS_i$ the expected packet is present, add it into the L7 PDU.
   (e) If FIN/RST/PSH flag is found or $PS = MaxPayload$, close the L7 PDU.
   (f) GOTO 2a.
3. Process remaining packets without SYNs. Create new L7 flows using timestamps and sequence numbers only.
4. Process every L7 flow and create L7 PDUs using TCP reassembling

[1] *MaxLost* was experimentally set to 4 kB, which is more than two times greater than maximal Ethernet PDU size, i.e., 1500 Bytes. *MaxTime* is six times greater than recommended TCP connection failure timeout as defined in RFC 1122. These values say that packet loss longer than 600 secs or missing 4 kB cannot be successfully recovered.

   - Add every packet of the L7 flow into the L7 PDU until FIN/RST/PSH or $PS = MaxPayload$. Then close the L7 PDU and create new one for new packets.
5. Combine opposite L7 flows into a L7 conversation using corresponding SYN and ACK numbers.

The main benefit of this approach is the reconstruction of original UDP/TCP sessions even if some important packets are missing. Based on TCP initial Seq numbers, the algorithm combines two flows into a conversation. The algorithm deals with missing SYNs, FINs, overlapping sessions, or TCP numbers overflowing. As the result, we have L7 PDU objects that can be processed on L7.

Table 1 compares our approach with a few available NSMs or NFATs. For our study, we have chosen Wireshark, Microsoft Network Monitor, NetWitness and Network Miner. In the first test we used an artificially arranged dataset with (i) one FIN packet missing, (ii) one SYN packet missing, and (iii) two SYNs missing. Original 650 kB PCAP file contained 19 conversions. Further analysis showed that in case of missing SYNs and the same port numbers, Wireshark joins two conversations into one. MS Network Monitor works well with missing SYNs, but it is not able to properly close communication if a FIN is missing. In such case, it combines two conversations into one. NetWitness also joins two conversations into one. Network Miner works similarly to Wireshark.

Table 1. Detection of network conversation when missing SYN/FIN packets.

File	NFX Det	Wireshark	MS Monitor	NetWitness	Net Miner
One FIN missing	19	19	18	17	19
One SYN missing	19	18	19	17	18
Two SYNs missing	19	17	19	17	17

The second test used 8 MB PCAP file with some packets randomly deleted. Table 2 shows results when 0 %, 1 %, 5 %, or 10 % of packets were removed. Original file contained 126 conversations. Netfox Detective shows number of L7 conversations.

Table 2. Detection of network conversations when some data are deleted.

File	NFX Det	Wireshark	MS Monitor	NetWitness	Net Miner
0 % missing	126	126	132	128	76
1 % missing	126	126	132	128	75
5 % missing	129	125	129	127	71
10 % missing	131	125	129	127	66

The table shows that Netfox Detective finds more L7 conversations than originally stored in the in-corrupted file. The reason is that when some packets are missing, a corrupted L7 conversation is divided into several L7 conversation due to the large number of missing packets or large timestamp difference, see Fig. 3. Wireshark and NetWitness also miss a conversation. However, since they consider all packets between the same src/dst ports as one conversation, missing packets usually did not reduce number of all conversations. MS Network Monitor also shows stable results. The results of Network Miner are very different but we are not able to say why.

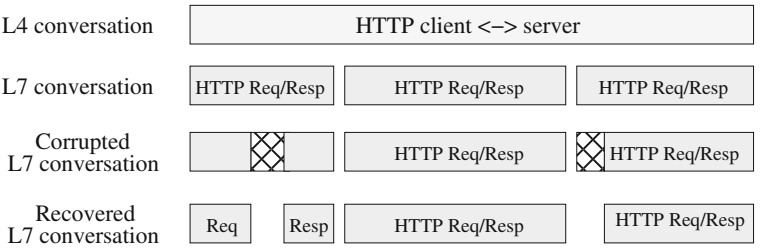

Fig. 3. Recovery of corrupted conversations.

3.3 Application Protocol Identification

The result of previously described reconstruction methods are L7 PDUs that represent L7 objects (payloads) prepared for L7 parsing. Before L7 parsing, L7 protocol should be identified in order to choose the right L7 parser. There are many methods for application protocol identification. The easiest method is based on well-known port numbers assigned by Internet Assigned Numbers Authority (IANA). Unfortunately, this method does not work well with applications using dynamic ports, peer-to-peer communication, video streaming, etc. More advanced methods use payload inspection that is suitable for protocols that can be recognized by some characteristic patterns either in a header or payload. There are also methods based on protocol fingerprinting or statistical data. In our approach, we combine several methods for application protocol identification.

1. *Identification using extended IANA database.*
   The first algorithm matches port numbers with extended IANA database of well-known ports. Our database extends IANA data by similarities, i.e., one input port number can match more applications. For example, Dropbox file hosting service can work on ports 80, 443, or 17500. Based on given application tags, L7 parser is chosen. Currently, our database can identify 1058 different application protocols.

2. *RTP Fingerprinting.*
   If there is no match on input ports, RTP fingerprinting method is applied [8]. This method uses a multi-stage classifier that observes minimal RTP header length, RTP version number, and RTP payload type number. If a packet successfully passes this filtering, per-flow checking is applied using minimal number of packets in an RTP flow to reduce false positives.
3. *Statistical Protocol Identification (SPID).*
   This method developed by Erik Hjelmvik [9] is based on supervised learning using pre-classified samples of captured network traffic where application protocols are correctly annotated. The algorithm generates protocol model database that stores application fingerprints. Currently, our database can identify 20 protocols with an ability to add new protocols.

4 Application Parsing

After building L7 PDUs and successful L7 protocol identification, application data can be processed by L7 parsers. As mentioned in Chap. 3, TCP/UDP streams are reconstructed without any knowledge of higher layers. This helps in case when an application parser is not implemented for a specific protocol. In that case application data can also be extracted from communication.

Main goal of our approach is to augment the reconstruction process when some data are missing. As mentioned earlier if only a few data is missing, lost packets can be replaced by new packets with empty payload. If more packets are lost, an original stream will be recovered as a collection of shorter streams that formed the original stream.

In this section, we will discuss how data reconstruction influences L7 processing and data presentation in case of incomplete data. For demonstration, we choose three areas that build challenges for common network parsers: web mail communication, SSL/TLS encrypted traffic, and bitcoin transactions.

4.1 Web Mail Analysis

Web mail communication is very popular today. Web mail servers employ HTTP protocol to encapsulate transactions between a user web browser and a web mail server. Mail exchange between web mail servers is mostly provided using SMTP protocol. Forensic analysis of web mail services is different from common web browsing. Many web mail servers utilize advanced web technologies like JavaScript, AJAX, JSON that dynamically create web pages. Analysis and interpretation of captured web mail data are limited due to the usage of web browser caches that store frequently used HTTP objects. These objects are not present in captured traffic, therefore, they are unavailable for forensic analysis.

The web mail analysis includes two phases: (i) the identification of web mail data between other HTTP traffic and (ii) the analysis of captured web mail data. In addition, most of web mail transmissions are SSL/TLS encrypted, so SSL/TLS decryption is required if possible (see Sect. 4.2). If encrypted, web

mail traffic can be identified using a name or IP address of a particular web mail server, see Table 3. If not encrypted, a pattern matching on URLs can be applied.

Table 3. Identification of web mail services during SSL/TLS handshake.

Web mail service	Server name	Encoding
seznam.cz, email.cz	email.seznam.cz	FastRPC
Gmail	mail-attachment.googleusercontent.com	application/x-www-form-urlencoded ;charset=utf-8
Yahoo	mail.yahoo.cz	application/json multipart/form-data-incl JSON
MS Live	*various*	application/x-www-form-urlencoded
Centrum/Atlas	mail.centrum.cz	application/x-www-form-urlencoded
Roundcube	*private service hostname*	application/x-www-form-urlencoded
Horde	*private service hostname*	multipart/form-data

For processing of a captured web mail data, following observations were made:

- Web mail messages transmitted over HTTP can be detected using URL patterns: */mail/.** for Gmail, *o1/mail.fpp* for MS Live Mail, *appid=YahooMailNeo* for Yahoo, etc. However, these patterns usually change with a new version of the server.
- The communication from a user towards the server is transmitted via POST method of HTTP protocol [10]. GET method is employed for listing mail folders.
- Web mail messages are mostly encoded using simple *key=value* pairs in the URL. There are several types of actions that can be identified in a *key* field: *compose-message*, *send-message*, *save-draft*, *get-inbox*, *delete-message*. Each web mail service uses different names for these actions, so data analysis should be performed for every new web mail protocol.
- Some web mail objects can be transmitted as JSON objects in MIME structure, XML-RPC objects, etc.
- Because of dynamic web programming and client-based technologies (i.e., JavaScript), forensic page rendering of web mail is difficult and cannot be fully accomplished without having contents of web caches. Practically, investigator's view is limited to a simple textual form of analyzed data.

4.2 SSL/TLS Detection and Encryption

The SSL/TLS encryption is a big challenge for current NFAT tools because it completely hides the contents of the network communication. It forms a modular

framework that combines various cryptography mechanisms defined by a cipher suite [11]. Clients and servers can negotiate cipher suites to meet specific security and administrative policies during initial SSL/TLS handshake. The cipher suite defines following mechanisms:

- *A key exchange algorithm.* General goal of the key exchange process is to create a pre-master secret known to the communicating parties that is used to generate the master secret. Using master secret encryption keys and MAC keys are generated. Most common key exchange algorithms are RSA, Diffie-Hellman, ECDH, etc.
- *A peer authentication.* TLS supports authentication of both peers, the server authentication with an unauthenticated client, and total anonymity. Whenever the server is authenticated, the channel is secure against man-in-the-middle attacks. Server authentication mostly requires a RSA or DSA certificate to prove an authenticity of the server side.
- *Message integrity.* Message integrity is ensured using Message Authentication Code (MAC) algorithms like MD5, SHA1, or SHA256. A cryptographic hash (often called message digest) is computed using these algorithms and added to the end of each block.
- *A bulk cipher algorithm.* This algorithm is used for a message encryption. The specification includes the cipher type (stream, block, AEAD [12]), the key size, the block size of the cipher (applied only to block ciphers), and the length of initialization vectors (or nonces). Common bulk ciphers are RC4, 3DES, AES, IDEA, or Camellia.

There are two basic approaches for SSL/TLS decryption [13]:

- *A getting server private key.* This key can be used to calculate a session key that have encrypted the conversation. The session key is generated during the key exchange.
- *A MitM attack on SSL/TLS connection.* Another method to get decrypted contents is to use man-in-the-middle (MitM) attack employing a special proxy server to track the communication between the client and server. At the same time, the communication with the user node employs different TLS keys generated by the proxy server. In this case, proxy server should offer a fake certificate in order to impersonate the original server. There are several tools implementing this proxy, e.g., SSLsplit, Fidler, etc.

Bulk cipher algorithms incorporate methods of a block cipher or stream cipher encryption that defines how a block or stream of a plain text will be encrypted and how the encryption key is generated for each data block, e.g. CBC (Cipher Block Chaining), GCM (Galois/Counter).

- The Cipher Block Chaining requires complete data for successful reconstruction because of data dependency, see Fig. 4A. If data are corrupted, successful analysis can be provided until the first error occurs in the stream. In such case, only meta information about the conversation are available, e.g. TCP completeness, probable conversation length, duration, etc.

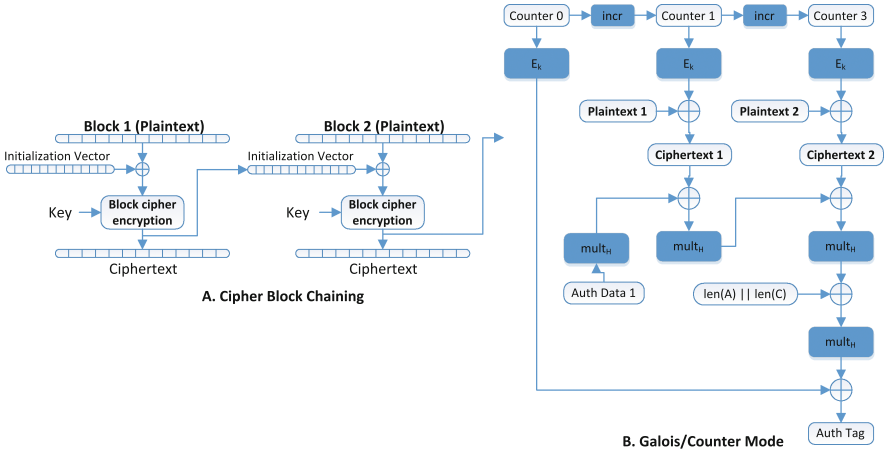

Fig. 4. CBC and GSM encryption.

– The Galois/Counter mode can be reconstructed even if some data are missing because cipher blocks are independent, see Fig. 4B.

Currently, our tool Netfox Detective supports analysis and decryption of various cipher suites, see Table 4.

Table 4. Cipher suites supported Netfox Detective.

TlsRSAWithAes128CbsSha	TlsRSAWithAes256CbsSha
TlsRSAWithAes128CbsSha256	TlsRSAWithAes256CbsSha256
TlsRSAWithAes128GcmSha256	TlsRSAWithAes256GcmSha384
TlsRSAWithRc4128Md5	TlsRSAWithRc4128Sha

If a server key is available, this communication can be decrypted as presented in Fig. 5. This picture shows a successful decryption of web mail communication encrypted using TLS.

4.3 Bitcoin Detection

Bitcoins as currency (BTC) are getting more and more popular since 2008, especially because of their anonymity. Bitcoin network is secure by design against correlating transactions with individual users. However, forensic tools can at least detect bitcoin traffic within a network.

Bitcoin operates over peer-to-peer (P2P) network consisting of two node kinds: (i) clients, which send, receive, or relay BTC transactions; and (ii) miners, which verify transactions using a special proof-of-work algorithm.

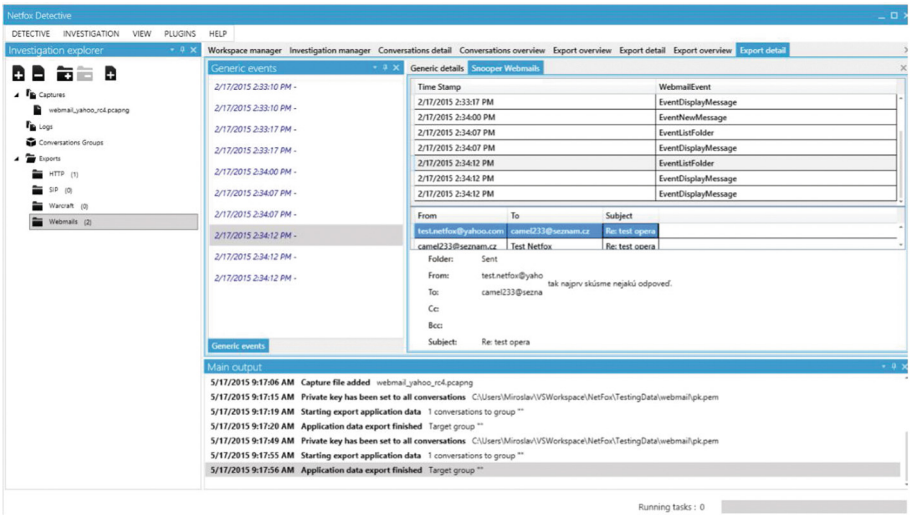

Fig. 5. Reconstruction of encrypted web mail data.

BTC uses three different protocols for its functionality where each protocol has a different value for the forensic investigation. These protocols are as follows:

1. Bitcoin v.1 protocol[2] is employed for P2P communication between peers (connected nodes). For forensic analysis, its detection can help to identify the end stations running Bitcoin client software. The protocol runs over TCP, port 8333. It transmits messages required for both a node discovery and Bitcoin transactions.

   Node discovery is provided twice in Bitcoin network:

- Upon software start-up, a client looks for special domain names (e.g., bitcoin.sipa.be, dnsseed.bluematt.me) in DNS in order to discover initial set of peers to get connected. Usually, the client uses a list of pre-configured stable nodes of the Bitcoin network.
- Upon successful connection to a node, the client may request a list of neighboring peers to expand its connectivity graph.

The protocol messages that helps us to detect a communication within Bitcoin P2P network area as follows: `version` and `verack` (useful for connection initiation), `address` (to detect a communication graph and provide information of known nodes), and `ping-pong` (a keep-alive mechanism). For forensic purposes, also messages `inv`, `tx`, and `block` are important since they transmit valuable information about processed transactions. The list of all Bitcoin v.1 messages is shown in Table 5.

[2] See https://bitcoint.org/en/developer-documenation, June, 2015.

Table 5. Bitcoin v.1 protocol.

Messages	Description	Message	Description
version, verack	Opening messages	*tx, notfound*	Responses to *getdata*
getaddr, addr	List of known peers	*ping, pong*	Keepalive messages
inv	A new object announcement	*alert*	Broadcast notification
getdata	Request for object value	*mempool*	Retrieving a transaction
getblocks, blocks	Retrieval of a block	*filterload/add*	Bloom filter operations
getheaders, headers	Retrieval of a header	*reject*	Negative response

2. Another group of protocols (e.g., Getwork, Getworktemplate, Stratum) is used for work distribution for miners cooperating in the pool. The detection of these protocols implies an existence of bitcoin miner in the local network.
3. The last protocol group involves remote procedure call (RPC) messages that are employed for remote control of various Bitcoin related services (e.g., remote wallets controlled by a smart phone, on-line trading on Bitcoin exchanges, etc.).

Netfox Detective currently supports decoding of Bitcoin v1 protocol that helps to detect devices that run Bitcoin clients, work as Bitcoin miners, or access Bitcoin related services, see Fig. 6.

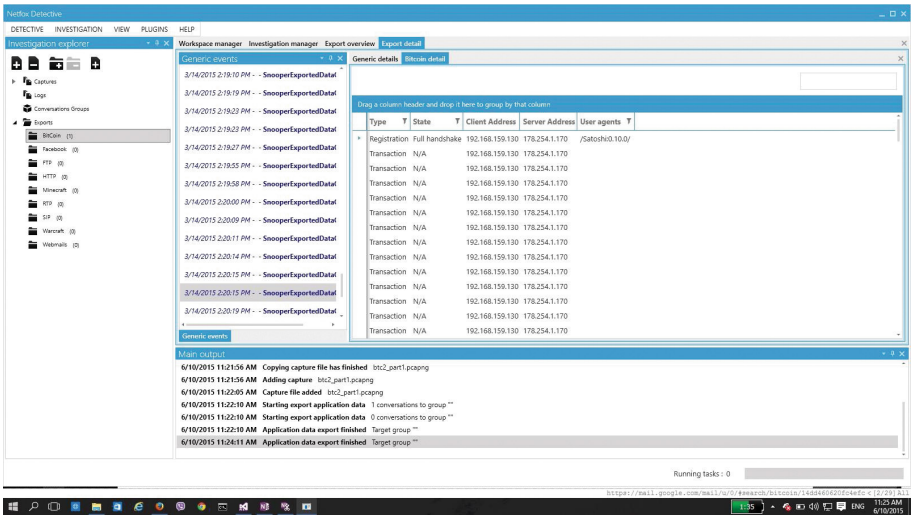

Fig. 6. Bitcoin analysis using Netfox Detective.

Based on these information, it is possible to create Bitcoin communication graphs and correlate the pool member and mining rig owner.

Captured network data can be used to provide an evidence that the seized server really conducted Bitcoin transactions, see Fig. 7.

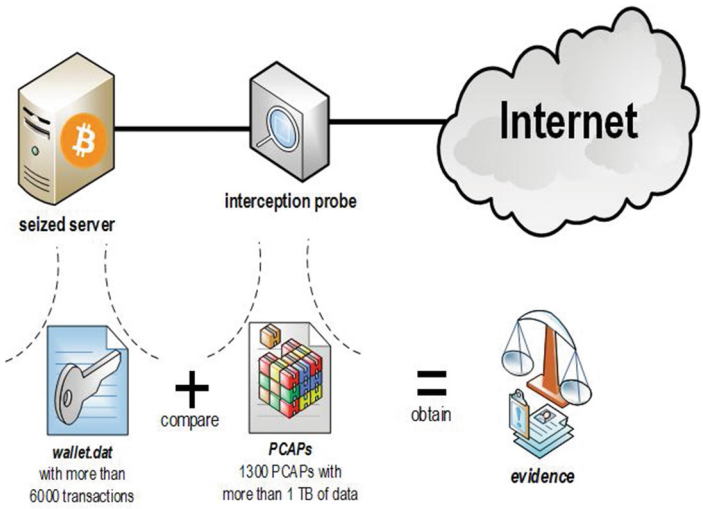

Fig. 7. Digital investigation of Bitcoin transactions.

5 Conclusion

Network forensics represent several challenges for security analysts. Network data are volatile what causes that communication traces are not captured completely. In addition, plenty of protocols are utilized in the current network communication. Many network applications also employ application-level protocol HTTP only as a data channel offering end-to-end connection. With the increased amount of traffic being encrypted, it is even complicated to recognize classes of applications in the captured communication.

In this paper, an overview of issues related to a recovery of the application content from captured traffic was presented. For identified problems, proposed methods were tested by implementing them in a novel network forensic tool. Based on the comparison to related tools, achieved results are promising for the further development of our NFAT tool.

Future work is delineated by the stated facts. Because of widely used traffic encryption, NFAT tools have to analyze meta-information associated with the traffic, e.g., recognizing events from communication, identifying end users, or approximate the meaning of information hidden in the encrypted communication. Also, the amount of communication requires NFATs to handle big data from various sources. Finally, NFATs should be extensible to deal with various classes of applications, e.g., web mail or Bitcoin traffic.

Acknowledgment. Research in this paper was supported by project "Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet", no. VG20102015022 granted by Ministry of the Interior of the Czech Republic and an internal University project "Research and application of advanced methods in ICT", no. FIT-S-14-2299 granted by Brno University of Technology.

References

1. Cohen, M.I.: PyFlag - an advanced network forensic framework. Digit. Investig. **5**, 112–120 (2008)
2. Pilli, E.S., Joshi, R.C., Niyogi, R.: Network forensic frameworks: survey and research challenges. Digit. Investig. **7**, 14–27 (2010)
3. Hunt, R., Zeadally, S.: Network forensics: an analysis of techniques, tools, and trends. Computer **45**, 36–43 (2012)
4. Dharmapurikar, S., Paxson, V.: Robust TCP stream reassembly in the presence of adversaries. In: USENIX Security Symposium. (2005)
5. Postel, J.: Internet Protocol. RFC 791 (1981)
6. Postel, J.: Transmission Control Protocol. RFC 793 (1981)
7. Stevens, W., Fenner, B., Rudoff, A.M.: UNIX Network Programming: The Sockets Networking API, 3rd edn. Addison-Wesley, Reading (2004)
8. Matousek, P., Rysavy, O., Kmet, M.: Fast RTP detection and codecs classification in internet traffic. J. Digit. Forensics Secur. Law **2014**, 99–110 (2014)
9. Hjelmvik, E., John, W.: Statistical protocol identification with SPID: preliminary results. In: Swedish National Computer Networking Workshop (2009)
10. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Barners-Lee, T.: Hypertext Transfer Protocol - HTTP/1.1. IETF RFC 2616 (1999)
11. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246 (2008)
12. McGrew, D.: An Interface and Algorithms for Authenticated Encryption. IETF RFC 5116 (2008)
13. Davidoff, S., Ham, J.: Network Forensics: Tracking Hackers through Cyberspace, 1st edn. Prentice Hall, Upper Saddle River (2012)