Gadget Detective: A novel open-source Network Forensics Analysis Tool

Anonymous Author(s)*

ABSTRACT

Network forensics is a major sub-discipline of digital forensics which becomes more and more important in an age where everything is connected. In order to cope with the amounts of data and other challenges within networks, practitioners require powerful tools that support them. In this paper, we highlight a novel open-source network forensic tool named — Gadget Detective — that outperforms existing tools such as Wireshark or NetworkMiner in certain areas. For instance, it provides a heuristically based engine for traffic processing that can be easily extended. Using robust parsers (we are not solely relying on the RFC description but use heuristics), our application tolerates malformed or missing conversation segments. Besides outlining the tool's architecture and basic processing concepts, we also explain how it can be extended. Lastly, a comparison with other similar tools is presented as well as a real-world scenario is discussed.

CCS CONCEPTS

• **Applied computing** → **Network forensics**; • **Networks** → *Network monitoring*; *Network protocols*; Transport protocols; Application layer protocols; • **Social and professional topics** → Computer crime.

KEYWORDS

Network Forensics, Protocol analysis, Web forensics, Network forensic analysis tool, Lawful interception

1 INTRODUCTION

Network forensics aims to understand/reconstruct events from network communication, which often requires expert knowledge (interpreting the low-level network protocols in order to see the big picture) [3]. To eliminate some of the complexity, adequate tools are essential [12, 13]. Specifically, tools should support investigators by summarizing, clustering and highlighting relevant information [1], e.g., provide contents of transmitted files, extract user credentials or perform analysis and visualize the data in an easily understandable form. While there are many different network forensic analysis tools [33] out there (details discussed in the upcoming sections), their functionalities, capabilities, and usability are not keeping up with traditional forensics toolkits [3] such as EnCase or The Sleuth Kit (TSK) & Autopsy.

Thematic classification: While network forensics and cloud forensics are related, the latter one is usually more complex, e.g., it may involve Software Defined Networking (SDN, [23]) which comes with additional evidence such as Logfiles from the SDN controller, compute nodes or cloud controller [40]. These networks also use state-of-the-art networking technology (100–400 Gbps) that cannot be monitored without hardware acceleration (typically FPGA), and even then, only selected flows can be fully captured [18] and used for further, detailed examination. Gadget Detective is intended for network forensic analysis and visualization on a PC and does not compete with these tools, but uses them to filter and capture data.

Terms and definition: For readers not completely familiar with the network terminology, we included an overview in Appendix A.

1.1 Analysis of Network Communication

Two of the most popular tools for Network Security Monitoring (NSM) are Wireshark and TCPDUMP, which are commonly used by network administrators to identify problems or security incidents [33]. Wireshark provides a large number of protocol parsers, can extract the content of the communication for several application protocols and offers a detailed view of the network communication. Although used by many network forensic practitioners, Wireshark was not designed for forensic purposes [33]. Its bottom-up analysis approach means that finding and extracting evidence often requires intensive labor. Gadget Detective partially addresses this by implementing advanced features such as heuristical TCP reassembling or L7 conversation tracking or reconstruction of forensic artifacts extracted from the communication. Furthermore, Wireshark does not scale well above hundreds of megabytes of source data, and thus, data preprocessing is necessary for large inputs. TCPDUMP, on the other hand, has only a command line interface that allows admins to inspect incoming and outgoing network traffic.

There are also more specialized tools that can extract valuable forensic information, for instance, ngrep, ssldump, or tcpxtract. These tools were created to solve specific problems such as searching for a phrase in network communication, decoding encrypted communication if a private key is known, or extracting transferred files from network communication, respectively. To take advantage of all tools, an investigator is required to combine them. For repeating tasks, one may write scripts to speed up the process and thus, reduce the amount of manual labor.

Without question, there are many practitioners who prefer featureful open-source tools [1, 11] although there is a risk that they are poorly documented, out-of-date, and even abandoned [12].

1.2 Expected Properties for Network Forensic Tools

According to [8], a network forensic analysis tool (NFAT) should provide a certain set of general features (listed as items 1-3 below). We further analyzed the demands and identified some more specific features yielding the following list of requirements:

(1) *Efficient processing of large capture files:* Current investigations deal with a big amount of data that needs to be analyzed. Tools are required to provide at least partial results quickly.
(2) *Extraction of high-level information:* Network communication can be analyzed at different levels but for digital investigation extracting artifacts from data sources is a priority.

(3) *Validation of results:* Applying reliable procedures and the possibility to validate the integrity of results is a crucial requirement on all forensic tools including NFATs.
(4) *Process non-standard or incomplete traffic*: Network communication should be correctly processed regardless of the acceptable deviations from the specification.
(5) *Robust data decapsulation:* Even in the presence of IP fragmentation and data stream multiplexing, the tool should be able to identify and compose unique application level conversations.
(6) *Support for overlay networks:* Network communication may be encapsulated using tunneling techniques, e.g., Virtual Private Networks. If possible, detection and extraction are then followed by the analysis of the encapsulated messages.
(7) *Application protocol identification*: Services communicating on non-standard or dynamic ports require advanced methods for application identification. Without the correctly identified type of communicating application, it is difficult to extract any high-level information.
(8) *Investigation process:* The tool should support the investigative process and guide the user. It is essential that even non-expert personnel can operate NFAT and extract evidence to support their cases.

The presented list is not exhaustive and stems from our experience in network traffic analysis and evaluation of existing NFATs. Some requirements are conflicting, for instance, processing of large data sources and in-depth analysis of conversations to extract high-level artifacts.

1.3 Network Forensic Tools

Besides Network Security Monitoring (NSM) tools that are intended for packet capturing, fingerprinting, or intrusion detection, there are some network forensic analysis tools (NFAT) specifically designed to support investigators. These aim to ease analysis by automating artifacts extraction and providing intuitive user interfaces. Usually, these tools have a top-down approach which makes the analysis simpler and saves time. In the following we briefly summarize the five prominent tools (numbers in brackets related to Sec. 1.2 and show missing properties):

- NetIntercept was one of the first NFATs [9]. It accepts PCAP files (no live captures), reassembles TCP flows and extracts artifacts from protocols running even on non-standard ports. Note: NetIntercept is closed source and to the best of our knowledge no longer available for download. Thus, we were unable to perform a more detailed evaluation.
- PyFlag [1, 3, 4, 6, 7, 8] "is a general purpose, open source, forensic package which merges disk forensics, memory forensics, and network forensics" [8]. By using specialized scanners, PyFlag can understand several application protocols and extract the communicated contents. However, according to Forensics Wiki, the tool is deprecated[1].
- XPlico [1, 3, 4, 5] is open source NFAT that is preinstalled on major digital forensics distribution such as DEFT, Security Onion and even Kali. It understands about 30 application protocols and can extract the content of emails, Session Initiation Protocol (SIP) or web communication.
- NetworkMiner [1, 3, 4, 8] is a passive network sniffer/packet capturing tool that can detect operating systems, sessions, hostnames, open ports, and more. It also allows extracting files from about a dozen commonly used application protocols. In the professional version, NetworkMiner also extracts VoIP calls, supports Geo IP localization, performs port-independent protocol identification, OS fingerprinting, and web browser tracing.
- TCPFlow [2, 3, 4, 5, 6, 8] "captures data transmitted as part of TCP connections (flows), and stores the data in a way that is convenient for protocol analysis and debugging. Each TCP flow is stored in its own file. Thus, the typical TCP flow will be stored in two files, one for each direction. TCPFlow can also process stored 'tcpdump' packet flows" [44]. It is important to note that TCPflow does not recognize IP fragments; therefore, reassembling of such conversations will be malformed.

While these tools have different strengths, our tool provides some unique features which are pointed out in Sec. 5.

1.4 Problem Description

Although many tools have been developed/exist, several tools are outdated, abandoned, or do not meet all expected properties (see Sec. 1.2). Additionally, current tools are not intuitive (require training), not (easily) expandable or can handle network traffic captures in the order of magnitude of gigabytes which were requirements/statements from the Lawful Enforcement Agency (LEA) officers. Last, existing tools do not particularly support the investigative process; commonly there is no case management, the linkage between investigations, and verification of results.

1.5 Contribution and Paper Structure

This paper provides Gadget Detective[2]; a novel, easy-to-use, powerful network forensic platform for top-down investigations. Our tool covers examination, analysis, and investigation phases of the forensic model as defined by [33]. In detail, we provide the following contributions:

- Introduction of investigation profiles that contain all necessary data for sharing the case by just copying the investigation folder — Sec 3.3.
- The new method of TCP stream reassembling based on heuristics (method itself was previously published **(citation removed due to double-blind review)**, but the tool contains an improved version of it) — Sec 3.4 and Appendix E.
- Improved identification of application-level sessions within TCP streams; the system can identify more application sessions compared to other tools (see Table 1) — Sec 3.4.
- Seamless analysis across boundaries of multiple capture files that ensures correct processing of long-running conversations — Sec 3.4; that ensures overlapping conversations to be correctly processed.

[1] https://www.forensicswiki.org/wiki/PyFlag (last accessed 2019-05-17).

[2] Gadget Detective is a symbolic name used for a double-blind review process. It will be renamed in the final version.

- Support for analysis of traffic encapsulated in GSE protocol; to the best our knowledge, Gadget Detective is the only open-source NFAT that supports GSE — Sec 5.3.
- Novel approach for web page reconstruction; in comparison to other tools, we do not only extract objects from HTTP communication, but we also reconstruct the page entirely (rewriting sources of all intercepted objects like CSS, pictures, video streams, etc.). Pages are stored as a MAFF archive including snapshots that show how the page changed over time. The JavaScript is interpreted, and particular API calls are mocked to be injected with intercepted ones, like REST API calls — Sec 6.2.

Note, the system has a modular architecture where processing engine, data-access component, and visualization subsystem can be used separately. The function related to packet capture file processing, namely, file parsing, conversation tracking, application protocol identification, application data extraction, and analysis can also be used as a standalone console tool and integrated to automated investigation procedures and combined with other existing tools.

The source code is released on GitHub and under the Apache Licence 2.0: **link removed for double-blind review**. Additional information can be found on YouTube: **link removed for double-blind review**.

The remainder of this paper is organized as follows: Sec. 2 describes the system architecture, illustrates the frontend, and explains possibilities on how to extend Gadget Detective. Sec. 5 highlights some of the unique features of our tool as well as contains a comparison with other prominent network forensics/security tools. The last section concludes the paper.

2 GADGET DETECTIVE

Gadget Detective is a network forensic tool that was developed to support digital forensic practitioners to analyze network captures and to extract evidence from packet traces quickly. It allows to correctly identify network conversations, parse common Internet protocols, and extract metadata as well as content from the end-to-end communication. Additionally, it is possible to extend the tool with new functionality through a well-documented API.

The tool is a Windows application relying on the .NET Platform and is available as an installation package that performs necessary deployment steps. Our implementation exploits many advantages of this platform like the rich graphical user interface provided by Windows Presentation Foundation (WPF), short development times due to a high abstraction language (C#), and availability of many libraries provided through NuGet packages. Furthermore, the implementation utilizes the Task Parallel Library (TPL) for parallel processing.

The software consists of over 140,000 lines of code[3] organized in about 110 projects. While it currently does not support live analysis, it accepts a variety of different network capture formats such as libpcap, pcap-ng, and Microsoft Network Monitor (MNM) format.

Fig. 1 describes the architecture, which is composed of two main components:

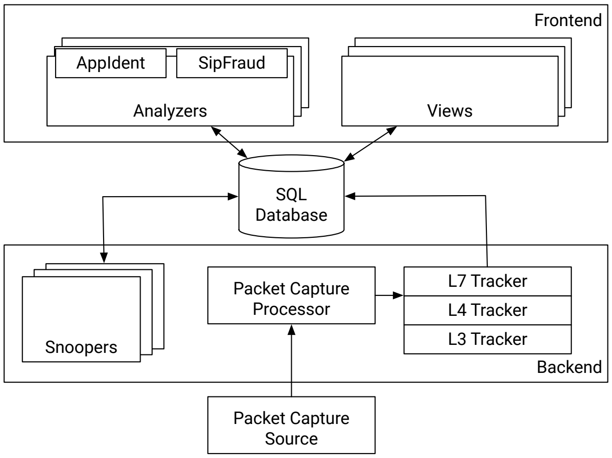


Figure 1: The overview of Gadget Detective Architecture.

Frontend is primarily a rich visual user interface (GUI, see Fig. 2) that is built on top of the backend and contains analysis capabilities [38]. *Analyzers* are frontend interfaces that allow adding new functionality. Details are outlined in Sec. 3.2.

Backend is a network traffic processing engine that performs: capture file processing, protocol parsing, traffic analysis, and metadata extraction. It is independent of the frontend (GUI) and comes with its own CLI which allows to integrate it in automated processing pipelines or to use it as a single-purpose tool. *Snoopers* are backend interfaces that allow adding new functionality. Details are outlined in Sec. 3.4.

Analyzers vs. snoopers: The tool can be extended through the implementation of snoopers or analyzers. Analyzers have more advanced functionality and different purpose than snoopers. The Analyzer API provides access to data storage as well as the user interface. An analyzer can be bound either to application or investigation scope. Thus, it is possible to integrate highly specialized analyzers for specific cases. Analyzers can create investigations, add capture files, or run any operation supported by Gadget Detective or access any data.

On the other hand, snoopers can access information from the processing pipeline through the database (metadata storage). Snoopers can extract objects from the source data but may also utilize other data such as flow records, log files, etc. Snoopers are intended to parse the application conversation protocols (L7, listed below) and extract data such as files, videos, or HTTP headers. More details about analyzers and snoopers are provided in Appendix B and C, respectively.

Note, Gadget Detective is too complex to explain every detail in this paper, and thus, we focus on some important design decisions in the next section. We plan on releasing more information/details over the years.

[3] Calculated by Visual Studio (code metrics) on the complete implementation; excludes white spaces, comments, usings, and third-party libraries.

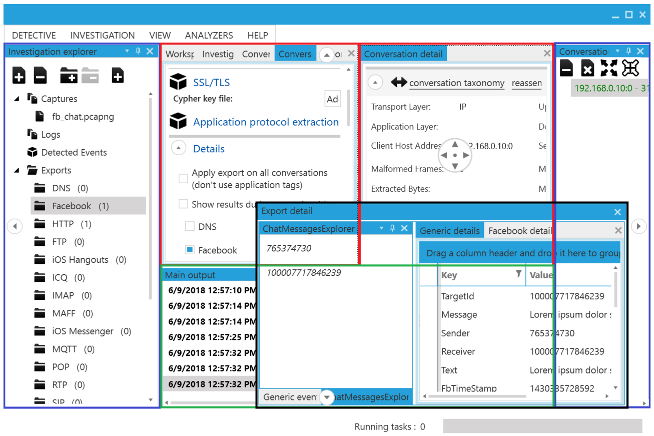
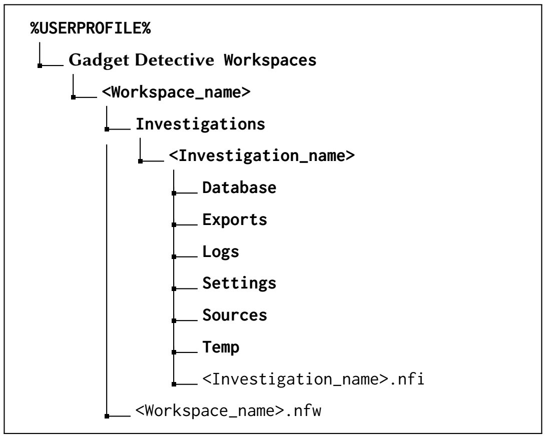


Figure 2: A screenshot of the UI design of Gadget Detective with highlighted dockable locations. Each pane can be moved and docked to any dockable location inside the Gadget Detective window, or drag & dropped outside the window to materialize a new one with the same dockable properties. This way, an investigator split the application across multiple screens.



```
%USERPROFILE%
└── Gadget Detective Workspaces
    └── <Workspace_name>
        └── Investigations
        |   └── <Investigation_name>
        |       ├── Database
        |       ├── Exports
        |       ├── Logs
        |       ├── Settings
        |       ├── Sources
        |       ├── Temp
        |       └── <Investigation_name>.nfi
        └── <Workspace_name>.nfw
```

Figure 3: The structure of an investigation folder. All workspaces are stored under the user's profile folder. Each workspace and each investigation has its name — suffixed with GUID for uniqueness. Each investigation contains a database, exports (extracted data from traffic), logs, settings, sources (copies of source data, e.g., PCAPs), and temp (for temporary data generated by snoopers and analyzers). Metadata about the workspace and investigation is stored in *.*nfw*, *.*nfi* files, respectively.

3 DESIGN DECISIONS

While we made many decisions along the way, the following subsections discuss the most important ones: GUI design, investigative process workflow, and packet processing pipeline.

3.1 No live captures

Gadget Detective does not support live captures but accepts several input formats, which had several reasons. First, lawful interception deployment contains one or more capturing probes that store data on drives locally, or on remote storage [16]. Secondly, the analysis is often performed on more powerful equipment rather than the capturing probe. Third, this was not a requirement by LEA.

3.2 GUI Design

The GUI follows the principles of Master/Detail screen layout [24] supported by the navigator panels as shown in Fig. 2. This organization is ideal for creating an efficient user experience [37] when the user needs to navigate between linked items [1]. The user interface provides a high degree of customization, utilizing a grid layout of dockable views. The application has three main areas, namely, left-hand side, upper right and lower right, that host basic visual components:

- *Investigation Explorer* is the main navigation panel of the application. It organizes Captures, Logs, Detected Events and Exported objects (see the *left blue box* in Fig. 2). More details about the structure are given in Fig. 3, and discussed in the *Investigation Explorer* paragraph.
- *Conversation View* provides a list of all tracked conversations in source capture files (see *left red box*).
- *Conversation Detail* provides information for the selected conversation. The presented content may contain links for additional data and detailed information on the target object (see *right red box*).
- *Detail View, e.g., Export Detail,* provides additional information for specific object types. The content uses links to navigate via multiple views (see *the black box at the bottom*).
- *Conversation Explorer* contains a list of conversations that were associated with investigated objects, e.g., conversation or export object (see *right blue box*).
- *Output Window* contains a list of events generated during the processing. These events may be informative, warnings or errors raised during source data processing (see the *green box*, only partially shown).

3.3 Investigative Process Workflow

The application was designed according to already well-established concepts known from *Integrated Development Environments* that programmers use to organize complex software designs [24]. With respect to digital forensics, we consider an *Investigation* to be an equivalent to a *project*; *Investigations* are combined into a *Workspace* that is equivalent to a *Solution.* An investigator can choose on which *Investigation(s)* s/he wants to work on and add data in the form of *PCAP* files or *logs.* Data is processed, and all gathered information is stored in an *Investigation's* scope; nothing is shared beyond that. In case several PCAPs are added (e.g., cause they have been split previously), across analysis is conducted (they will be treated as one PCAP internally for tracking and reconstruction of events). While data is never shared between investigations, we allow opening multiple investigations (in separate docked panes) which allow comparing data from multiple sources.

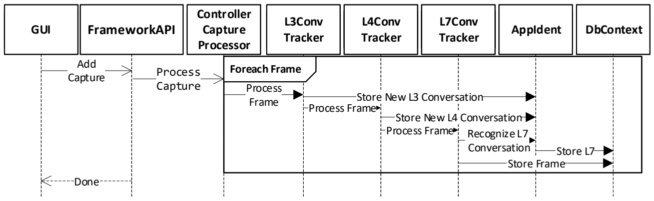


Figure 4: Abstract capture file processing scheme with a sequential passage. Data dependencies between models are omitted. The ultimate goal is to identify and collect application level conversations. In order to accomplish this, communication at low levels need to be properly identified, messages parsed, relevant data extracted, and packet composed. This is achieved by conversation trackers.

3.4 Packet Processing Pipeline

To master the challenges of parsing and to polish all information gathered, it consists of several interconnected implementation blocks which compose a packet processing pipeline. The pipeline (lower right-hand side of Fig. 1) performs (i) packet file loading and processing, (ii) conversation tracking, (iii) application recognition and (iv) extracted (meta)data storing. Thus, the processing pipeline handles the identification of protocols for each packet, performs defragmentation, and does stream reassembly for TCP communication (L7 Tracker). A detailed view is provided in Fig. 4. Note, the snoopers allow to extend the backend and will be discussed in Sec. B.

Packet file loading and processing. (i.e., components Packet Capture Source, Packet Capture Processor, L3-L7 Trackers, and AppIdent): Source packet capture files are processed by the corresponding packet file loader depending on their file type. The processing of the frames is sequentially where each loaded frame is dissected into the low-level protocols to identify its key properties, such as a physical address, network address, or ports. The dissected packet is forwarded to the next component (i.e., L3 Tracker) which performs further processing.

Conversation Tracking. Conversation tracking is a critical component of the system as it examines each dissected packet and associates it with the corresponding conversation[4]. A conversation is considered as the basic data object for further analysis. The system identifies conversations at different network layers:

- Packets sharing the same source and destination addresses belong to the same network layer conversation (L3). Every pair of devices shares a single L3 conversation.
- Packets with the same network source and destination addresses, transport layer source and destination ports and a specific transport protocol belong to the same transport layer conversation (L4). At this layer, the conversation mostly corresponds to a pair of TCP streams or UDP data exchanges.
- Lastly, application layer conversations (L7) are identified using various TCP heuristics we have developed previously (**citation removed due to double-blind review**) and improved for this article. The heuristics solve the problem when dealing with incomplete data or multiple sessions that are merged into a single transport layer conversation. L7 conversations reflect a single session between a client and a server application.

Correct identification of conversations from source packets is a challenging task as several issues may arise, e.g., out of sequence packets, missing packets, fragmented packets, or missing termination packets. To succeed, we use several heuristics to identify and collect as many conversations as possible, even in corrupted or incomplete data sources. Additionally, the tool addresses the need for fast processing by using available processor cores, implementing concurrent conversations processing.

Metadata Storage (database). Extracted information, e.g., conversations at different layers, application layer data units, and other relevant information, is stored in a SQL database. The bulk insert method is used to obtain better performance. Thus integrity is not guaranteed until all data is inserted. The user interface is aware of this and handles temporally incomplete data correctly. The database is accessed through persistence providers that allow to easily add support for different databases[5].

4 TESTING

Given the complexity of our application, testing was (is) an essential part throughout the development process, where we followed a Test-Driven Development (TDD) methodology. TDD requires writing tests first, then production code that passes the tests and lastly to refactor the code to improve its structure. We utilized unit tests, which then also ensures integration/regression testing and ensures the correctness of new versions. Because our focus is very specific (network data parsing and analysis), mocking the data would be tedious [31]. Therefore, we omit the unit tests in favor of integration/system tests that use data loaded from PCAP files processed (in-time of the test) by our processing pipeline.

To develop and test modules (snoopers/analyzers), we started by collecting testing data first, where we either downloaded available PCAPs or created our ground of truth utilizing our private networks. In the latter case, we then filtered the captured data using Wireshark, which ensured that we only deal with one application message, action, or scenario at a time. If Wireshark supported the application protocol, we compared both results (ours and Wireshark's).

In the beginning, we also used Microsoft Network Monitor (MNM), which allowed us to develop parsers written in *Network Parsing Language* (NPL). In other words, we created parsers for two different frameworks and compared results. Given that MNM is outdated, and this is not the most reliable method for testing, we abandoned MNM.

After carving basic events from the protocol messages worked correctly (single packet), we created more complex scenarios (e.g., a login scenario which has multiple packets) and manually verified the results. Lastly, we created a comprehensive dataset and extracted key data (e.g., the summary of extracted events) which we then used as benchmark data for new version testing to prevent

[4] Note, conversations are also called bi-flows in some literature.

[5] Currently, the tool supports Microsoft SQL and in-memory data storage.

regression bugs. Currently, Gadget Detective contains 1000+ tests that are automatically executed whenever new code is committed and run approximately 46min. In case that a regression bug is found, the merge is denied until the bug is fixed.

5 EVALUATION

The rest of this section discusses the efficiency (see Sec. 5.1) followed by a summary of carving capabilities. In Sec. 5.3 we compare Gadget Detective to other exiting tools before we provide a real-world example. The last section explains the Reconstruction of Web Pages; a very unique feature of our application.

5.1 Efficiency Assessment

Although Gadget Detective is an offline analysis tool, runtime/memory footprint are essential aspects. Thus, this section discusses the runtime efficiency in comparison with Wireshark and NetworkMiner. To measure the efficiency, we used the M57-Patent scenario[6] PCAP files which consist of several PCAP files with a total size of 4.8 GB and 5, 707, 845 frames (we combined them into a single PCAP). Note, given that each tool performs very different tasks, this is only a rough comparison.

The results are provided in Table 1. As can be seen, Gadget Framework is slightly faster than Wireshark despite the TCP reassembling of all sessions. Note, when opening the case the 2nd time, all data is extracted from the database which is completed in a matter of seconds. However, we require more memory footprint (RAM). Gadget Detective is slightly slower than Gadget Framework as it visualizes the information. NetworkMiner is about 4-7 times slower than the other tools. The average CPU usage is not reaching 100 % with Gadget Framework and Gadget Detective because of the thread synchronization, I/O operations, Garbage Collection, and back pressure in the processing pipeline that balances overall performance and resource utilization. Overall, the Mbps per tool vary between 15 and 100.

Additional efficiency indicators are given in Table 2, where we focus on rows 12 and 13 (processing speed and parallel processing; remaining rows are discussed in Sec. 5.3). As shown, Gadget Detective allows parallel processing, which should make it faster than the deprecated PyFlag. On the other hand, [8] points out that PyFlag is not intended for high-speed. Concerning XPlico, more research is needed as it also processes data in parallel, and we did not find information on processing speed.

5.2 Event Carving Capabilities

The next important aspect for forensics is *event carving*, i.e., restoring particular events such as an FTP Login, a DNS query or sending emails from a comprehensive stream. This section primarily focuses on NetworkMiner (NM) and Gadget Framework and their capabilities; Wireshark does not incorporate advanced forensic features such as emails or web page reconstruction as it is intended for Network Security Monitoring [33, 39].

For comparison, we decided to focus on detected sessions, TCP reassembling, and DNS records where the results are shown in Table 1. These properties strongly depend on how a tool was implemented. Higher numbers reflect finer granularity (this does not mean that higher (or lower) numbers are better).

Sessions: the number of TCP and UDP sessions recognized by each tool. This feature strongly depended on the mechanism handling missing fragments, see Appendix E. If there is no packet loss; the tools should report the same number of TCP sessions; UDP sessions can differ in case the tool uses an inactivity timeout threshold to split UDP sessions (the UDP protocol does not carry any signaling information that can be used to determine the end of a session).

TCP missing: signifies how much information is lost and cannot be recovered, e.g., capturing problems, packet loss, or storage issues. All issues are related to actions that occurred before processing of the capture file, i.e., they are not caused by Gadget Detective. There are different ways to calculate the loss as shown in Eq. 1 or Eq. 2:

$$lost\_packets/all\_packets[\%] \tag{1}$$

$$lost\_bytes/all\_bytes[\%] \tag{2}$$

Gadget Detective uses the Eq. 2 as we believe that if a sequence of packets is lost, their count is unknown and can be approximated using a heuristic based approach on MTU or previously observed segment sizes. However, we had to utilize Eq. 1 as Wireshark does not explicitly count *lost_bytes*.

DNS records: the number of events carved from DNS traffic. Gadget Detective extracts much more events compared to NM that only considers DNS response packets [27] and ignores query packets [26]. NM also ignores some other record types such as PTR, SRV or MX that may carry valuable forensic information, e.g., a mapping of IP address to the domain name (PTR), a definition of the service location (the hostname and port number (SRV)), or domain names of mailing servers (MX). This additional information may be useful in case of DNS spoofing attacks/investigations [15]. Lastly, NM only shows the first record from an answer section. In contrast, Gadget Framework processes all, i.e., all records from Question, Answer, Authority, Additional from both packet types (not only response).

Emails and errors: reflects the number of extracted emails. NM identifies more emails as Gadget Framework currently only considers emails sent through the SMTP protocol; NM also processes emails sent through webmail[7].

FTP: the number of events identified in the FTP session. While NM extracts only transferred files, Gadget Detective and Wireshark show other related (meta-)information about the FTP sessions such as the login or list-command.

Web pages: the number of reconstructed web pages using our module. In total, 182 HTTP objects were found which created six MAFF Archives containing full offline web page snapshots including CSS and other downloaded objects. For additional details we refer to Sec. 6.2.

In summary: each of the tools has its strengths and weaknesses, and one has to choose the best tool for the job. For instance, Gadget

[6] https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario (last accessed 2019-05-17).

[7] This was a scenario we have not considered. We will update our module in the near future.

Table 1: Performance of selected operations using the M57 case PCAP files. Machine configuration: CPU i7-4790, 4.00 GHz, 64 GB DDR4, Crucial MX100 SSD, Windows 10. Experiments were repeated 10-times, measured by *time* and *Perfmon* utilities.

	Operation	Backend	Frontend+Backend	Wireshark†	NetworkMiner†	tcpflow†,△
1	Total time	6m 14s, $\sigma \approx 15.23\,s$	9m 36s, $\sigma \approx 30.12\,s$	8m 48s, $\sigma \approx 17.34\,s$	41m 23s, $\sigma \approx 124.43\,s$	13m 39s, $\sigma \approx 64.21\,s$
2	Max RAM usage	8.3 GB	8.5 GB	7.1 GB	20 GB	243 MB
3	Avg CPU usage	76 %, $\sigma \approx 8$ %	66 %, $\sigma \approx 18$ %	12 %, $\sigma \approx 3$ %	15 %, $\sigma \approx 2$ %	3 %, $\sigma \approx 1$ %
4	Sessions (TCP+UDP)	118,709	118,709	98,084	49,865	93,619
5	TCP - missing	3.9%*	3.9%*	0.6%*	N/A	N/A
6	DNS - records	238,531	238,531	150,426	183,527	N/A
7	Emails	28	28	N/A	39	N/A
8	FTP	16	16	N/A	1	N/A
9	Complete Web pages	6	6	N/A	N/A	N/A
10	Speed	101.8 Mbps	66.1 Mbps	72.1 Mbps	15.3 Mbps	46.5 Mbps

(*) Gadget Detective computes TCP loss based on lost segment size (see Eq. 2). For Wireshark, we computed it by applying the *tcp.analysis.lost_segment* filter and then utilized Eq. 1. This does not mean that the tool lost the data but they were not present in the capture, i.e., the capturing probe lost them.
(†) The tool was downloaded as a binary release.
(△) The tcpflow 1.4.4 was ran with parameters *-r file.pcap -a -Fm* to do ALL post-processing and split output in 1M directories.

Detective has focused on carving capabilities from conversations containing missing data.

5.3 Comparison to Existing Tools

This section compares Gadget Detective against other applications concerning capabilities, functionality, and features. A summary of the results is provided in Table 2 and will be briefly discussed in the upcoming paragraphs.

In its current version, Gadget Detective does not allow *live data capture* or *PCAP-over-IP* and thus is not as flexible as NetworkMiner or XPlico. However, it supports various capture file types. Note, this was a design decision: we work under the premise that data is gathered on capturing probes and uploaded for analysis after the capture ends (or parts of the ongoing capture are provided).

In terms of support for encapsulation protocols, NetworkMinor has a wide variety of supported protocols. However, to the best of our knowledge, Gadget Detective, and Wireshark are currently the only applications that support Generic Stream Encapsulation (GSE). In comparison to other protocols, GSE frequently uses multiple encapsulations, whereas other protocols usually do not. That requires a significant change in the tool's architecture.

Rows 6-8 deal with application and their protocols. While Gadget Detective uses a variety of different algorithms to identify the protocol, NetworkMiner and XPlico rely on SPID and PIPI. Furthermore, Gadget Detective tries to identify applications as well as application protocols, e.g., HTTPS-Firefox, HTTPS-Chrome. **Citation removed due to double-blind review.** However, further testing is required to make a qualified decision which tool works the most reliable. Concerning supported application protocols, our tool supports a wide variety of different ones, including some unique protocols like Facebook Messenger, Hangouts, Twitter, or Warcraft. Note, since those are implemented using snoopers, there will be more in the future.

OS fingerprint (row 9) is supported by NetworkMiner and Gadget Detective. While we rely on the *AppIdent* analyzer, NM uses statistical based SPID algorithm [14].

In case that user credentials are observed in a communication, Gadget Detective, and NetworkMiner allow to extract them where the two tools focus on different protocols. Another major feature is the handling of malformed, incomplete network traffic. This is based on our previous work, **(citation removed due to double-blind review)** where we showed that the risks of undesired association of the unrelated conversation fragments yielding twisted evidence. We could not find information for NetworkMiner; however, as shown in Table 1, NetworkMiner identifies significantly fewer sessions (maybe due to combining unrelated conversations). Advanced analytical views address visualization capabilities where Gadget Detective is very flexible due to the Analyzer API (see Sec. C), which ensures that the tool can be extended with pluggable modules. In terms of XPlico, we were unable to find detailed information; besides a reference to a PHP Framework named cake-php[8].

Row 16 addresses the querying/filtering capabilities of the corresponding tools. NetworkMiner, Wireshark and PyFlag include basic query functionality (e.g., keyword searches), XPlico and Gadget Detective require third-party tools (e.g., one may query the database using analytical third-party applications or write a new snooper). If support for hitherto application protocol is required, the advanced investigator can create a new snooper module that will be dynamically be loaded without a need of recompilation of the Gadget Detective. In comparison to Wireshark, creation of a new snooper is straightforward imperative programming based on an enriched API of a data stream that handles several types of application protocol behaviors, like request-response, asynchronous message exchange, etc., that helps to handle missing/not-captured data.

To sum it up: While there are aspects where other applications like NetworkMiner are superior, Gadget Detective has a lot of unique functionality/features and is under active development — new features can be expected. Especially the number of supported application protocols, the incomplete or malformed communication handling make and the expandability, make it a great forensics tool. Additionally, we believe that one of the major difference is usability and the amount of expertise needed (especially compared with Wireshark).

6 EXAMPLE FEATURES

The Gadget Detective contains many advanced features. We picked two that we are the proudest. These features have been tested in real deployments and helped LEA investigators to solve cases.

[8] http://wiki.xplico.org/doku.php?id=interface (last accessed 2019-05-17).

Table 2: Gadget Detective in comparison to major open-source network forensic tools. The provided information was gathered from official sources provided by the tool authors. N/A indicates that we could not find any details regarding the particular feature. We deliberately do not add any information that is not stated by authors, such as processing speed.

	Tool Feature	Gadget Detective	NetworkMiner	XPlico	PyFlag
1	**Live data capture**	NO	YES	YES	NO
2	**PCAP-over-IP**	NO	YES	YES	NO
3	**Supported file types**	libPcap, Pcap-NG, MNM	libPcap, Pcap-NG	libPcap	libPcap
4	**IPv6**	YES	YES	YES	NO
5	**Encapsulation protocols**	GRE, 802.1Q, GSE	GRE, 802.1Q, PPPoE, LLMNR, VXLAN, OpenFlow, SOCKS, MPLS and EoMPLS	L2TP, VLAN, PPP	NO
6	**Application Protocol Identification**	SPID, NBAR, ESPI, Bayessian, Random Forests	SPID, PIPI	PIPI	NO
7	**Supported application protocols**	HTTP, SSL/TLS, MAFF. XMPP, YMSG, OSCAR, Facebook Messenger, Hangouts, Twitter, XChat, IMAP, POP3, SMTP, Gmail, Yahoo, RTP, SIP, Minecraft, Warcraft, Facebook, Stratum, DNS, FTP, SPDY, MQTT	FTP, TFTP, HTTP, SMB, SMB2, SMTP, POP3, IMAP, YouTube	HTTP, POP3, SMTP, IMAP, SIP, RTP, SDP, FTP, DNS, IRC, IPP, PJL, MMS, SLL	DNS, HTTP, MSN, Gmail
8	**Applications Identification**	YES	NO	NO	NO
9	**OS Fingerprinting**	YES (using typical applications)	YES	NO	NO
10	**Credentials Extraction**	Facebook, IMAP, SMTP, POP3	SMTP, HTTP Digest Authentication	NO	NO
11	**Incomplete or malformed communication**	TCP data loss, IPv4 fragmentation	N/A	NO	NO
12	**Processing speed**	100 Mbps	11.92–18.49 Mbps	N/A	N/A
13	**Parallel processing**	YES	NO	YES	NO
14	**Advanced analytical views**	YES	NO	YES	NO
15	**Persistent storage**	MSSQL, in-memory	CSV / Excel / XML / CASE / JSON-LD	SQLite, MySQL or PostgreSQL	VFS
16	**Querying/filtering**	3-rd party tools on SQL DB	keyword search	3-rd party tools on SQL DB	YES

6.1 An Example: SIP Fraud Analysis

This section reviews Gadget Detective in use based on a simulated SIP (Session Initiation Protocol) Fraud case. The SIP Fraud attack exploits a misconfiguration of the SIP server where the attacker tries to guess a secret prefix that is used to initiate a call from a VoIP network to PSTN (public switched telephone network). If the attacker finds the correct prefix, the Gateway (Callee) replies with a *200 OK* SIP message. The attacker then uses the discovered prefix to initiate a call on a premium number. The costs of the call are charged to the owner and will profit the attacker.

To tackle the challenge, we developed the SIP Fraud Analyzer that can perform a postmortem analysis of possible SIP fraud attacks in given PCAPs. The exact procedure is best explained by Fig. 5. The upper part is an interactive animation that reflects the actual state of the system (commodity server with hardware-accelerated network card), the IPFIX collector and NEMEA system [5] (note, this is *not* part of Gadget Detective but external equipment/software). In a nutshell, the hardware (left-hand side) captures information and forwards it to NEMEA. Once an attack (or false-positive thereof) is identified [17], NEMEA notifies the appliance, which then captures all evidence (generates a PCAP) and stores it on the hard drive. This file then serves as input for Gadget Detective.

Knowing the workflow, we now focus on the analyzer and its responsibilities. First, NEMEA can notify Gadget Detective about its current state which allows us to update the view (e.g., the red arrow pointing from NEMEA to FPGA). Thus, an investigator can see (live) the current processing. Second, NEMEA can notify Gadget Detective when capturing is completed and trigger the analyzer to download and visualize the PCAP (the bottom pane in the figure).

Fig. 5 shows the SIP Fraud Analyzer main view that visualizes the attack pattern. The evidence has the form of prefix guessing activity represented by several SIP INVITE messages that differ by the prefix of the callee number (here it is the number 031...@65518.... and a lot of seemingly random prefixes which reflect the attacker guessing them). In other words, if the analyzer shows a graph like this, one knows an attack occurred; if we find *200 OK* message, we know that the attack was successful.

The system was tested/developed with a confidential dataset from the National Research and Education Network (NREN). During the experimental deployment of this system, we were able to successfully extract evidence, and based on that we informed victims about their misconfiguration in SIP's PBX[9].

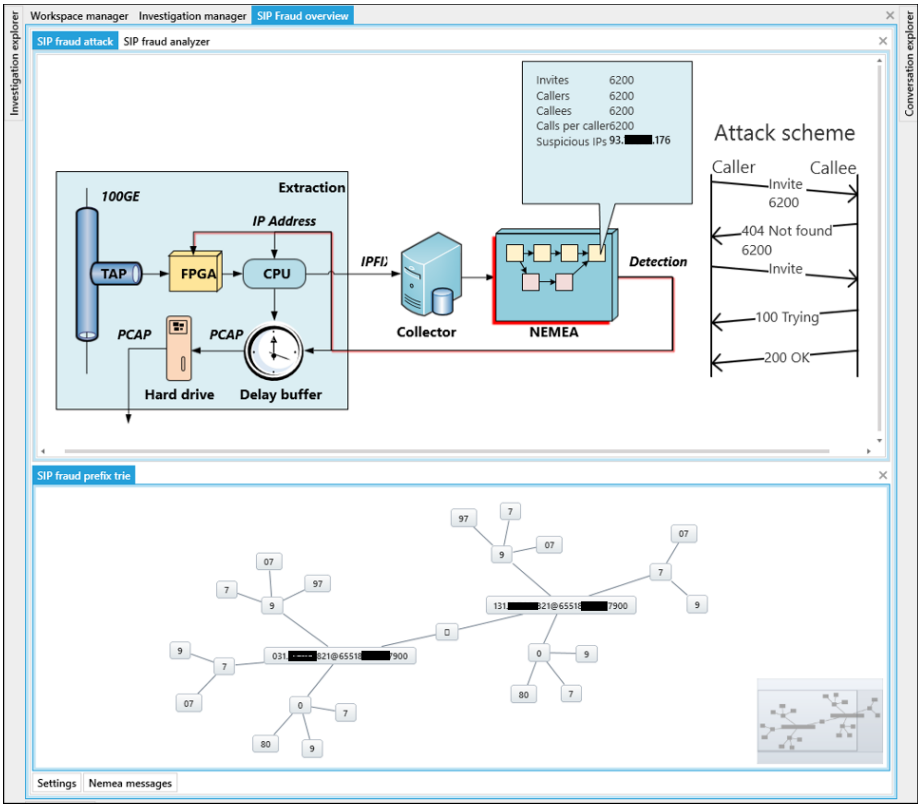
6.2 Reconstruction of Web Pages

Another feature of the Gadget Detective is web page reconstruction. First, the *SnooperHTTP* extracts all HTTP objects and stores the contents on disk. Second, *SnooperMAFF* iterates through the HTTP objects to identify all *HTML* documents. Subsequent analysis of these documents yields all linked objects, e.g., CSS files, JavaScript scripts, media streams, and so on. Lastly, all references to web resources are rewritten (e.g., `<a href="http://.../photo.png"` will be replaced by `<a href="./photo.png"`), and then the *HTML* documents including all resources are packed into Mozilla Archive Format (MAFF) archive.

The self-contained MAFF archive[10] contains all data that is related to each web page that was viewed. Experimentally, in case of the dynamic web that loads data continuously, we try to create

[9] PBX – Private branch exchange used to relay VoIP communication to the PSTN – public switched telephone network.

[10] Note, Mozilla discontinued MAFF support in newer Firefox versions. We advise using SeaMonkey with MAFF plugin https://addons.thunderbird.net/en-us/seamonkey/addon/mozilla-archive-format/.



Invites 6200
Callers 6200
Callees 6200
Calls per caller6200
Suspicious IPs 93.[redacted].176

Figure 5: This figure describes *SIP Fraud Analyzer*. The view is an interactive animation that reflects the actual state of the deployed 100GE hardware-accelerated network card with the IPFIX collector and the NEMEA system that detects network incidents based on IPFIX records. *SIP Fraud* is visualized on the upper right side with a count currently analyzed messages, i.e., 6200. At the bottom, a tree-like structure visualizes a *prefix tree* that is an interpretation of the attack. The root node in an interconnection between the same roots of telephone number attacked from different IP addresses. The path from a leaf node to the root aggregate node represents a prefix combined with a PSTN number that was tried to be called. Sensitive information, as a part of called number and IP addresses, was omitted.

multiple so-called snapshots that approximate how the web page may have looked. The snapshot is created with each significant change to the web page. The investigator is warned that this is experimental approximation and not an accurate replica. We do this approximation by interpreting JavaScript scripts and supplying it with resources previously extracted. Hence, we can reconstruct some dynamic pages like *webmails*, *chats*, or *video streaming* services. These approximations are stored inside the MAFF archive as additional tabs.

Note, web page reconstruction is only possible if the session is established using plain HTTP. Otherwise, it requires the investigator to get into the middle of the communication using a MitM proxy like SSLSplit that can capture unencrypted traffic or store SSL / TLS session keys [36].

7 CONCLUSIONS

The amount of data sent over networks increases daily, and so does the number of devices connected to it. Additionally, analyzing the data becomes more complex due to encryption, the large number of different protocols or tunneling. As a consequence, forensic investigators are overwhelmed with data (possible evidence), and traditional workflows are outdated (i.e., manually combing several specialized tools like SSLSplit, TShark, or Wireshark). To cope with these challenges, it requires automated, extendable tools that support practitioners by summarizing data and providing visualization, which allows easy comprehension of the information [1].

In this article, we presented Gadget Detective which is a comprehensive open-source network forensic analysis tool (NFAT) available under the Apache 2.0 License. By design, our application can be expanded by implementing new modules; backend modules are called snoopers and frontend modules (which allow more complexity) are named analyzers. To enable researchers to create new modules, we have a well-documented API including several examples. The GUI follows the principles of a Master/Detail screen layout and uses dockable views, which makes it intuitive and easy-to-use. We achieve better performance than comparable tools because of the parallel pipeline processing. As a side note: it was used by CESNET[11] for SIP Fraud Detection as mentioned in Sec. 6.1.

The evaluation and comparison with existing tools show that Gadget Detective has a good efficiency as it makes use of all cores. Additionally, it has some unique features, that cannot be found in any other NFAT, e.g., a large number of supported application protocols as listed in Table 2, support for GSE tunneling, or heuristic extraction from malformed data.

For the future, we plan on expanding Gadget Detective by creating new modules (features), e.g., finding similarities using approximate matching [2]. We also plan on changing the mechanisms for data processing to allow computation on clusters. In terms of interoperability, we intend to add exporting capabilities into standard formats, e.g., Advanced Forensic Format [7] or CybOX [4]. Lastly, we want to create training sessions and material which will allow practitioners to become familiar with our tool.

REFERENCES

[1] Nicole Beebe. 2009. Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics*. Springer, 17–36.

[2] Frank Breitinger, Barbara Guttman, Michael McCarrin, Vassil Roussev, and Douglas White. 2014. *Approximate Matching: Definition and Terminology*. Special Publication 800-168. National Institute of Standards and Technologies. http://dx.doi.org/10.6028/NIST.SP.800-168

[3] Eoghan Casey. 2004. Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. *Digital Investigation* 1, 1 (2004), 28 – 43. https://doi.org/10.1016/j.diin.2003.12.002

[4] Eoghan Casey, Greg Back, and Sean Barnum. 2015. Leveraging CybOX™ to standardize representation and exchange of digital forensic information. *Digital Investigation* 12 (2015), S102–S110.

[5] Tomas Cejka, Vaclav Bartos, Marek Svepes, Zdenek Rosa, and Hana Kubatova. 2016. NEMEA: a framework for network traffic analysis. In *Network and Service Management (CNSM), 2016 12th International Conference on*. IEEE, 195–201.

[6] Mihai Christodorescu, Xin Hu, Douglas L Schales, Reiner Sailer, Marc Ph Stoecklin, Ting Wang, and Andrew M White. 2015. Identification and classification of web traffic inside encrypted network tunnels. US Patent 9,106,536.

[7] Michael Cohen, Simson Garfinkel, and Bradley Schatz. 2009. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *Digital Investigation* 6 (2009), S57–S68.

[8] M. I. Cohen. 2008. PyFlag - An Advanced Network Forensic Framework. *Digital Investigation* 5 (Sept. 2008), 112–120. https://doi.org/10.1016/j.diin.2008.05.016

[9] Vicka Corey, Charles Peterman, Sybil Shearin, Michael S. Greenberg, and James Van Bokkelen. 2002. Network Forensics Analysis. *IEEE Internet Computing* 6, 6 (Nov. 2002), 60–66. https://doi.org/10.1109/MIC.2002.1067738

[10] EnCase. cited January 2019. URL: https://www.guidancesoftware.com/encase-forensic.

[11] Dan Farmer and Wietse Venema. 2009. *Forensic Discovery* (1st ed.). Addison-Wesley Professional.

[12] Simson L Garfinkel. 2010. Digital forensics research: The next 10 years. *Digital Investigation* 7 (2010), S64–S73.

[13] Vikram S Harichandran, Frank Breitinger, Ibrahim Baggili, and Andrew Marrington. 2016. A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later. *Computers & Security* 57 (2016), 1–13.

[14] Erik Hjelmvik and Wolfgang John. 2009. Statistical protocol identification with spid: Preliminary results. In *Swedish National Computer Networking Workshop*. 399–410.

[15] Markus Huber, Martin Mulazzani, and Edgar Weippl. 2010. Who on Earth Is "Mr. Cypher": Automated Friend Injection Attacks on Social Networking Sites. In *Security and Privacy – Silver Linings in the Cloud*, Kai Rannenberg, Vijay Varadharajan, and Christian Weber (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 80–89.

[16] Invea. cited January 2019. URL: https://www.invealawfulinterception.com.

[17] Tomáš Jansky, Tomáš Čejka, and Václav Bartoš. 2017. Hunting SIP Authentication Attacks Efficiently. In *Security of Networks and Services in an All-Connected World*, Daphne Tuncer, Robert Koch, Rémi Badonnel, and Burkhard Stiller (Eds.). Springer International Publishing, Cham, 125–130.

[18] Lukáš Kekely, Jan Kučera, Viktor Puš, Jan Kořenek, and Athanasios V Vasilakos. 2016. Software defined monitoring of application protocols. *IEEE Trans. Comput.* 65, 2 (2016), 615–626.

[19] James F Kurose and Keith W Ross. 2016. *Computer networking: a top-down approach*. Vol. 7. Addison Wesley Boston.

[20] libPcap. cited January 2019. URL: https://www.tcpdump.org/.

[21] M57. cited January 2019. URL: https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario.

[22] MAFF. cited January 2019. URL: http://maf.mozdev.org/maff-specification.html.

[23] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[24] Microsoft Corporation. 2017. Master/details - Windows UWP applications. https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/master-details URL: https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/master-details.

[25] Microsoft Network Monitor. cited January 2019. URL: https://blogs.technet.microsoft.com/netmon/.

[26] P Mockapetris. 1987. RFC 1034 Domain Names - Concepts and Facilities. http://tools.ietf.org/html/rfc1035http://tools.ietf.org/html/rfc1034

[27] P. Mockapetris. 1987. RFC 1035 Domain Names - Implementation and Specification. http://tools.ietf.org/html/rfc1035

[28] Andrew Moore, Denis Zuev, and Michael Crogan. 2013. *Discriminators for use in flow-based classification*. Technical Report. Qween Mary University of London.

[29] NetworkMiner. cited January 2019. URL: https://www.netresec.com/?page=NetworkMiner.

[30] ngrep. cited January 2019. URL: https://github.com/jpr5/ngrep.

[31] Roy Osherove. 2015. *The art of unit testing*. MITP-Verlags GmbH & Co. KG.

[32] Pcap-NG. cited January 2019. URL: https://github.com/pcapng/pcapng/.

[33] Emmanuel S. Pilli, R. C. Joshi, and Rajdeep Niyogi. 2010. Network Forensic Frameworks: Survey and Research Challenges. *Digital Investigation* 7, 1-2 (Oct. 2010), 14–27. https://doi.org/10.1016/j.diin.2010.02.003

[34] Jan Pluskal, Ondrej Lichtner, and Ondrej Rysavy. 2018. Traffic Classification and Application Identification in Network Forensics. In *Advances in Digital Forensics XIV*, Gilbert Peterson and Sujeet Shenoi (Eds.). Springer International Publishing, Cham, 161–181.

[35] PyFlag. cited January 2019. URL: https://github.com/py4n6/pyflag.

[36] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[37] Bill Scott and Theresa Neil. 2009. *Designing web interfaces: Principles and patterns for rich interactions*. O'Reilly Media, Inc.

[38] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2019. An Evaluation Framework For Network Security Visualizations. *Computers & Security* (2019).

[39] Rommel Sira. 2003. Network forensics analysis tools: an overview of an emerging technology. *GSEC, version* 1 (2003), 1–10.

[40] Daniel Spiekermann, Jörg Keller, and Tobias Eggendorfer. 2017. Network forensic investigation in OpenFlow networks with ForCon. *Digital Investigation* 20 (2017), S66 – S74. https://doi.org/10.1016/j.diin.2017.01.007 DFRWS 2017 Europe.

[41] ssldump. cited January 2019. URL: http://ssldump.sourceforge.net.

[42] SSLSplit. cited January 2019. URL: https://www.roe.ch/SSLsplit.

[43] TCPDUMP. cited January 2019. URL: https://www.tcpdump.org/.

[44] TCPFlow. cited January 2019. URL: https://github.com/simsong/tcpflow.

[45] tcpxtract. cited January 2019. URL: http://tcpxtract.sourceforge.net/.

[46] The Sleuth Kit (TSK) & Autopsy. cited January 2019. URL: https://www.sleuthkit.org/.

[47] TShark. cited January 2019. URL: https://www.wireshark.org/docs/man-pages/tshark.html.

[48] Wireshark. cited January 2019. URL: https://www.wireshark.org/.

[49] XPlico. cited January 2019. URL: https://www.xplico.org/.

[11] CESNET is a developer and operator of national e-infrastructure for science, research, development, and education in Czech Republic.

A TERMINOLOGY AND DEFINITION

There are several definitions in the community regarding flow, conversation, etc. For this work, we used the Microsoft Network Monitor (MNM) terminology[12] which is very close to the well established terminology used by [19].

Frame is a data link layer (L2) protocol data unit.

Packet is an internet layer (L3) protocol data unit.

Datagram is a transport layer (L4) protocol data unit.

Protocol / application message is a application layer (L7) protocol data unit (PDU). A message is a collection of one or more L7 PDUs.

L3 flow is a sequence of packets having the same source and destination IP addresses. It represents an uni-directional transmission of packets between two network nodes.

L3 conversation is a pair of L3 flows with mutually transposed source and destination IP addresses. It represents bi-directional transmission between two network nodes.

L4 flow is a sequence of packets with the same source and destination IP addresses and ports, and an L4 protocol number. It represents uni-directional communication between processes, e.g., data sent by an HTTP client to an HTTP server, possibly in several L4 half sessions. An L4 flow consists of one or more L7 flows.

L4 conversation is a pair of L4 flows with mutually transposed L3 and L4 identifiers (src / dst IP addresses and src / dst ports). It represents bi-directional communication between processes, e.g., requests and responses between an HTTP client and server. The L4 conversation may contain several L4 sessions (L7 conversations) between the same network nodes using the identical src / dst ports and the L4 protocol.

L7 flow is a part of the L4 flow that represents a transport session, e.g., one UDP or TCP session. For TCP, an L7 flow is bounded by its initial SYN packet and its closing FIN or RST packet. For UDP, an L7 flow corresponds to an L4 flow. One L4 flow may include several L7 flows that are logically independent, e.g., several TCP sessions (HTTP requests) with the same src / dst ports and IP addresses may compose one L4 flow. A TCP L7 flow also includes starting SYN and ACK packets without any L7 payload, if present.

L7 PDU represents an approximation of an application message, e.g., HTTP request. L7 PDU is a logical object that contains an L7 payload of one or more packets belonging to the same L7 flow. It is created using TCP reassembling. L7 PDU objects are processed by application parsers called L7 Snoopers. In a case of UDP, an L7 PDU is created for every L4 payload, i.e., there is an 1:1 relation between UDP payload and application message.

L7 conversation is a pair of L7 flows. It represents logical application data that are subjected to the forensic analysis. L7 flows are interconnected to the conversation according to SYN and SYN+ACK sequence numbers in TCP. An L7 conversation includes meta data such as timestamps of the first and last PDU — selected from both directions whichever is prior and posterior, number of frames of L7 conversation, collection of virtual frames representing missing (expected) frames, type of encryption, cipher keys (for TLS decryption), collection of probable application tags (types of L7 protocol, e.g., HTTP, SMTP, etc.).

L7 Snooper is an application data analyzer (application parser, dissector). Snooper reads L7 PDUs from L7 conversations and performs L7 processing, analysis, and visualization. L7 snoopers can co-operate with each other, e.g., SIP snooper co-operates with RTP snooper, WebMail snoopers with HTTP snooper, etc.

L7 Analyzer is a less strict abstraction, module that encapsulates predefined behavior that applies to processed data or directly controls data processing. L7 Analyzers have full access to Gadget Detective platform and can change, extend any functionality used for processing or analysis.

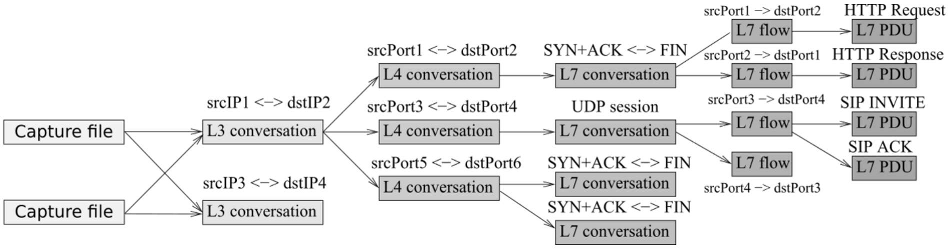


Figure 6: Figure describes relations between encapsulations on various levels of networking stack reflected by object hierarchy serving as containers. Data is segregated into a particular container based on common identifiers described in Section A. One *L3 Conversation* can contain frames from multiple *capture files* and have a relation *one to many* with *L4 Conversations*. The rest of graph is read similarly.

[12] The terminology was determined by study of MNM manual, and blog — https://blogs.technet.microsoft.com/netmon/ (last accessed 2019-05-17).

B SNOOPERS (BACKEND MODULES)

The backend supports modules, called snoopers, that can access information from the processing pipeline through the database (metadata storage). Snoopers extract objects from the source data but may also utilize other data such as regular log files. Therefore, snoopers parse the application conversation protocols (L7, listed below) and extract data such as files, videos, or HTTP headers. These extracted objects are then either stored in the database or pushed to the *Investigation Explorer* (grouped by a protocol) or can be accessed from the special *Export Overview* pane where they are grouped by event type, e.g., emails, images, chat messages. The following protocols for metadata and/or content extraction are supported:

- Common internet protocols: *DNS*, *SPDY*, and *SSL / TLS*.
- Selected application protocols: *HTTP(S)*, *IMAP*, *POP3*, *SMTP*, and *FTP*.
- Email services: *Gmail*, *Yahoo*, and other *webmails*.
- Instant messaging: *XMPP*, *YMSG*, *OSCAR*, *Facebook Messenger*, *Hangouts*, and *XChat*.
- Social networks and gaming: *Twitter*, *Facebook*, *Minecraft*, and *Warcraft*.
- Bitcoin communication: *Stratum*.
- Voice over IP systems: *RTP* and *SIP*.
- Internet of Things communication: *MQTT*.

If the communication is not encrypted (or the server's private key is provided, and the server's configuration allows it), the snoopers can extract the communication content, e.g., transmitted files. For secured communication, only traffic metadata is available.

In order to create new snoopers, there are three abstract classes that need to be inherited:

SnooperBase can be seen as the extractor that will handle the identification of objects. The registration of a new snooper and its integration is automated as long as the snooper's *DLL* resides in the root directory of the application. Details about the snooperBase are provided at the end of this subsection.

SnooperExportObjectBase stores the actual objects. For instance, an application protocol parser will dissect the communication and create instances of domain objects. Those objects might also implement various interfaces like `IChatMessage`, `ICall`, `IEMail`, `IPhotoMessage`, etc. to automatically integrate exported objects in generic views.

SnooperExportBase encapsulates (meta-)information about the export process. For instance, the source of an L7 conversation. Additionally, it contains all extracted objects `SnooperExportObjectBase`.

SnooperBase. To create a new snooper, a new class that inherits from the abstract class `SnooperBase` including the class members, such as `Name`, `Description`, `KnownApplicationPorts`, `CreateSnooperExport`, and `ProcessConversation`, needs to be implemented. Additionally, the class defines multiple abstract methods that represent callback functions executed during conversation processing. An example is given in D. The functionality has to be implemented in the following methods:

- `OnConversationProcessingBegin` – any relevant activity for creating a new object to be populated by the module.
- `OnConversationProcessingEnd` – any required processing before the new object is stored in the database.
- `OnBeforeProtocolParsing` and `OnAfterProtocolParsing` – takes care of the internal state of an object and handles exceptional cases that are assigned to 'parsing state'.
- `OnBeforeDataExporting` and `OnAfterDataExporting` – takes care of the internal state of an object and handles exceptional cases that are assigned to information 'extraction state'.

Each snooper is executed on-demand, on the selected PCAP or a collection of them, according to the tool configuration. While modules can use the information provided by other modules, their basic use case is to implement extraction capabilities for application protocols. For more complex analysis, we use analyzers.

For readers interested in demonstration *How to develop a new snooper in abstract code*, we include Appendix D.

C ANALYZERS (FRONTEND MODULES)

Analyzers extend Gadget Detective with more advanced functionality that cannot be implemented as snoopers. The Analyzer API provides access to data storage as well as the user interface. An analyzer can be bound either to application or investigation scope. Thus, it is possible to integrate highly specialized analyzers for specific cases. In order to grasp the concept of analyzers, we discuss their capabilities based on the *AppIdent* — an application identification analyzer. AppIdent assigns an application protocol (or even network application) to every flow in the source data. The goal of the analyzer is to recognize applications (e.g., Google Drive, iTunes, or OneDrive) in network traffic instead of just the application layer protocol used (e.g., HTTPS).

The analyzer is implemented using machine-learning [6] and statistical methods, in particular, Bayesian belief network, Random Forests, and Enhanced Statistical Probability Identification, to make the decision. Because supervised learning methods are used, there are two running modes:

The learning mode is used to build the models which required annotated data. To generate the data, we produced local network traffic and captured the communication using Microsoft Network Monitor, which automatically enriches the capture with information about the running processes[13]. Note, the training data was annotated with the application process instead of the application protocol. On the other hand, our backend engine was extended to extract the process information for learning purposes. The feature vector characterizing the application protocol was specified according to the work by [28], and customized for L7 conversation-based approach instead of packet-based.

The classification mode of the analyzer is used for annotating conversation with recognized protocols and applications. It is not an easy task, and the precision varies for the classification methods and the target set of applications. For more details see [34] who demonstrated that it is possible to distinguish between communications traces of OneDrive, Skype, iTunes, Spotify, Steam and µTorrent clients, although all of them use HTTPS.

Usually, traffic classification is a black box (e.g., in security software/hardware like IDS/IPS) and depends on the model. However, for practitioners, it may be helpful to get more insight and therefore *AppIdent* can provide additional computed results in a visual manner. In other words, we implemented views allowing the comparison of the classification results of different methods, classifier performance analysis, and hyper-parameter tuning.

[13] In detail, MNM creates a Process Info table that stores information on the socket and the process that created it.

D ABSTRACT CODE FOR AN EXAMPLE PROTOCOL SNOOPER CREATION

```
public class SnooperExample : SnooperBase {
  public override string Name => "Example Protocol";
  protected override SnooperExportBase CreateSnooperExport()
      { throw new NotImplementedException(); }
  public override string Description => "Description of Example Protocol";
  public override int[] KnownApplicationPorts => new[] { 42 };

  protected override void ProcessConversation()
  {
      // we need a stream to read from
      var stream = new PDUStreamBasedProvider(this.CurrentConversation,
                                             EfcPDUProviderType.Breaked);
      // now we can create a reader for the stream
      var reader = new PDUStreamReader(stream, Encoding.GetEncoding(437), true);

      // reader will spawn messages, cycle through them
      do
      {
          this.OnBeforeProtocolParsing();

          // parse the protocol
          var message = new ExampleProtocolParseMsg(reader);
          if (!message.Valid){
              //TODO report error
              continue;
          }

          this.OnAfterProtocolParsing();
          // TODO some additional integrity checks perhaps
          this.OnBeforeDataExporting();

          var exportedObject = new SnooperExportedDataObjectExampleProtocol
                                  (this.SnooperExport){...};
          this.SnooperExport.AddExportObject(exportedObject);

          this.OnAfterDataExporting();
      } while (reader.NewMessage());
  }
}
```

E SIMPLIFIED REASSEMBLING ALGORITHM IMPLEMENTED IN GADGET DETECTIVE.

```
(1) Select L4 flows and sort packets using their sequence numbers.
(2) Process each L4 flow accordingly:
    (a) Start following iteration with a SYN packet, i.e., $P_i$.
    (b) Increment $Seq_i$, i.e., $Seq_i + = 1$.
    (c) Create a new L7 Flow to be a collection of L7 PDUs for following algorithm. Set $P_{init} = P_i$.
    (d) Create a new L7 PDU if does not exist or if a previous L7 PDU was closed.
    (e) If $Seq_i \neq Seq_{i-1} + |P_{i-1}|$ (the expected packet is missing, check timestamps $TS$ and sequence numbers $Seq$ as follows:
        (i) If $TS_i - TS_{i-1} \leq MaxTime$ and $Seq_i - Seq_{i-1} \leq MaxLost$ then a virtual packet will be created to replace the missing packet.
        (ii) If $TS_i - TS_{i-1} \geq MaxTime$ and $Seq_i - Seq_{i-1} \leq MaxLost$ then there is an overlapping of TCP sessions because $i$ packet, i.e., this packet, belongs to a different L7 flow. Skip this packet and proceed with the next one.
        (iii) If $Seq_i - Seq_{i-1} \geq MaxLost$ then there are too many missing data. The flow cannot be fully restored. Close it and proceed with next SYN packet.
    (f) If $Seq_i == Seq_{i-1} + |P_{i-1}|$ the $P_i$ packet is expected, i.e., $P_i$ contains following data segment, add it into the L7 PDU created in 2d.
    (g) If FIN/RST/PSH flag is found or $|P| == MaxPayload$, close the L7 PDU.
    (h) If $P_{init} == P_i$, break iteration.
    (i) Increment $i$, i.e., $i+ = 1$ and GOTO 2d.
(3) If there remains any SYN packet in the current L4 flow, GOTO 2a
(4) If the L4 flow contains any unprocessed packet, i.e., captured communication is incomplete and heuristic methods (2e) have to be applied.
(5) Select packet $P_i$ that has maximal $Seq_i - Seq_{i-1}$ and GOTO 2c
(6) Combine opposite L7 flows into an L7 conversation using corresponding SYN and ACK numbers.
```

P_i — represents the packet on the i-th index
$|P_i|$ — represents a payload size obtained from the packet header
Seq_i — represents *sequence number* of packet on i-th index
P_{init} — stores the reference to the packet that the reassembling algorithm started with
TS_i — represents *time stamp* of the packet on the i-th index
$MaxTime$ — variable, empirically set to $600\,s$
$MaxLost$ — variable, empirically set to $3800\,B$
$MaxPayload$ — variable, empirically set to *maximal expected MTU*