NetInfoMiner: High-level Information Extraction From Network Traffic

Sultan Almuhammadi, Ahmad Amro, Sami Zhioua
College of Computer Sciences and Engineering
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
Emails: (muhamadi, g201404360, zhioua)@kfupm.edu.sa

***Abstract*—Due to the rapid increase of the Internet traffic encryption HTTPS, and the newly adopted protocols HTTP2 and SPDY, the need for a comprehensive high-level information extraction tool that supports the new protocols becomes essential for critical applications such as digital and network forensic and web penetration testing. In spite of the availability of big data from the Internet traffic, current network data mining tools do not support encrypted network traffic and the new protocols. This paper proposes a new tool for extracting high-level information such as visited links, user credentials and session cookies from HTTP and HTTPS protocols. It also allows extraction of user credentials from HTTP2 and SPDY.**

***Keywords*—Big data, traffic analysis, data mining, HTTP, HTTP2, SPDY, HTTPS.**

I. Introduction

Big data can be obtained from network traffic through network components. The rapid increase of Internet users make such data both available and valuable. Since 2016, Internet users have exceeded 3.3 billions [1], which directly affects the amount of traffic generated from users' interactions on the network level. According to Cisco Visual Networking Index (VNI), the amount of monthly data transferred on IP networks will reach 168 Exabyte by 2019 [2]. Part of this data includes private high-level information such as browsing history, users' credentials, session management cookies, and other information, which are typically encoded in low-level format within the network traffic trace.

Extracting interesting information from network traffic is hard and tedious. It requires understanding of all the protocols responsible for communicating information over the network, and the criteria of generating the traffic. The process of extracting information becomes more challenging if the network traffic is encrypted. According to a TLS implementation survey [3], about 53.6% of the industry top 140,000 websites apply inadequate security implementation, which means either insecure or absent implementation of TLS. This indicates that targeting both encrypted and non-encrypted network traffic for data mining is equally needed.

This paper proposes a new tool for extracting desired high-level information such as visited links, login events (including usernames and passwords), and session cookies. The proposed tool supports four protocols (HTTP, HTTPS, HTTP2, and SPDY). It is designed to process large amount of network traffic traces.

II. Backgroung

There are several networking protocols used to power up web applications on the World Wide Web (WWW), the most common ones are the first version of The Hypertext Transfer Protocol (HTTP), the second version HTTP2, the secure version HTTPS, and the predecessor of HTTP2 (SPDY, pronounced speedy). Although HTTP, specifically HTTP1.1 is still the most common negotiated protocol for carrying web contents, it has suffered from known limitations, such as head of line blocking and TCP handshake latency. Google initiated the SPDY protocol to improve both the security and the performance of HTTP. SPDY was the basis of HTTP2 [4].

It is important to mention two features of SPDY/HTTP2: (a) they are binary (while HTTP is textual); and (b) they multiplex several requests and responses under a single TCP connection to reduce the overhead on servers [4]. These features highly affect the processing of web transactions on network traffic traces. Processing SPDY and HTTP2 traces requires different parsing and different concept than basic consecutive request and response protocol. Since these protocols are negotiated and not fully supported by all browsers, traffic traces for retrieving the same web resource might be different from one client to another. This effect is discussed in Section (V-C).

Any secure connection to a web resource can be served over HTTPS, which is basically HTTP over TLS protocols. SSL is the predecessor of TLS, both are protocols used to establish encrypted channels between clients and servers to protect both privacy and integrity of connections. Recently, all chromium based browsers (Chrome, Opera, etc.) in addition to Firefox, begin to provide an option to log the TLS session keys. If the environment variable *SSLKEYLOGFILE* is set, these browsers log all keys used in the negotiation of TLS sessions to a file on disk. This option provides the ability to decrypt then analyze TLS traffic using $Wireshark$ [5].

Throughout this paper, all the described algorithms and approaches share some similarities between HTTP, HTTP2 and SPDY. So, for simplicity we refer to all of them as HTTP unless a distinguishing feature is needed. Moreover, we refer to SSL/TLS traffic as TLS since their differences are irrelevant to the work presented in this paper.

III. Related Work

In some cases, especially in digital forensics, a complete reconstruction of some users browsing session might be re-

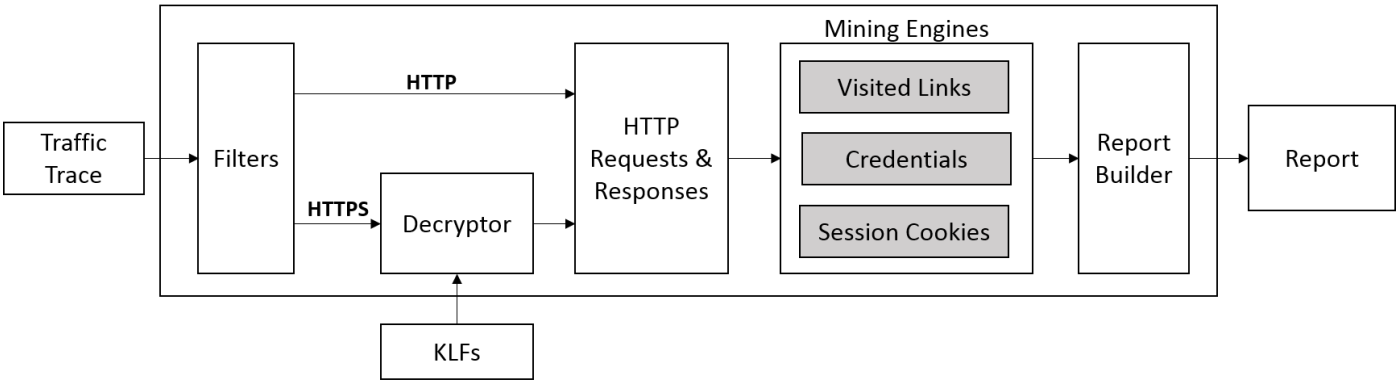

Fig. 1. NetInfoMiner Block Diagram

quired. Examples of reconstruction tools include PyFlag [6] and $ClickMiner$ [7].

In [6], a framework for network forensic ($PyFlag$) is presented. It argues the benefits of targeting network traffic for information extraction. $PyFlag$ can reconstruct HTML objects from network traffic. Since the framework was developed way before the rapid adoption of network traffic encryption, $PyFlag$ has been deprecated [8].

$ClickMiner$ [7] is introduced as a tool for reconstructing user browsing interactions from network traffic traces. It implements a referral-based click inference (RCI) initially proposed by $Resurf$ to extract the visited links [9]. $ClickMiner$ improves RCI by filtering known automatically generated advertisement requests which generates some false positive in $Resurf$. $ClickMiner$ follows the extracted clicks in a browser driver fed only with the information extracted from network traffic, claiming that up to 90% of the users interactions can be reconstructed.

$NetworkMiner$ is another tool for information extraction from traffic traces without session reconstruction. It extracts useful information (HTTP traffic, files, cookies, etc.) used in forensic applications [10].

IV. Motivation

Statistics show the validity of targeting network traffic for data mining and extracting information for many applications. In digital forensic, bits of information in digital investigations are added up to help making informed decisions. Current solutions to handle network data mining such as $ClickMiner$ and $NetworkMiner$, do not support all protocols and may provide some inaccurate results. For example, these tools do not provide support for HTTP2, SPDY, and HTTPS. Moreover, $NetworkMiner$ claims the ability to extract the credentials, where in fact it actually extract all the parameters and cookies of HTTP requests and displays them, even if these parameters hold undesired information like language, location, etc.

We propose a tool that addresses the aforementioned limitations and provides more accurate results. It extracts high-level information (visited links) ,reconstructs the login events (user credentials), and identify session cookies. Extraction of such information is the main aim of network forensic analysts to establish high-level facts regarding digital investigations [6].

Statistics show an increase adoption rate for the new HTTP2 by top web applications, from 1.4% on October 2015, to 10.3% on October 2016. On the other hand, a slower adoption rate for SPDY, from 6.3% on October 2015 to 7.3% October 2016 [11].

V. Proposed Work: NetInfoMiner

We present a new tool capable of processing large amount of network traffic traces and extracting high-level information. Our new tool is called: Network Information Miner ($NetInfoMiner$). It mainly overcomes the following issues:

- Current solutions for network traffic data mining suffer from limitations in supporting new adopted protocols: HTTP2, and SPDY.
- Current solutions do not mine credentials, and session cookies used in web applications authentication and session management with high accuracy.

The types of information targeted by the proposed tool are the visited links (click paths), credentials, and session management cookies. The block diagram of NetInfoMiner is shown in Figure 1. The tool receives captured traffic traces, and generates a summary report of the extracted information. It provides the option to decrypt HTTPS network traffic encrypted using TLS if the encryption keys are available.

The work presented in this paper can benefit the academic security community with following additions:

- The basic knowledge on the trends of authentication processes implementation by modern web authentications
- The effect of web browsing activities on the network traces in light of the adoption of new protocols (HTTP2, SPDY, and HTTPS).
- A tool capable of collecting datasets of usernames and passwords and session cookies from network traffic to benefit other research research goals.

This work can also be used in web penetration testing to identify the security of the authentication process (username and password sent in clear, hashed, or encrypted). Digital forensic and network forensic can also benefit from the proposed tool.

A. Methodology

Before extracting information from TLS traffic, two components are needed: (a) traffic trace of the entire TLS session, and (b) the server private key or the session keys. First, the traffic trace of the entire TLS sessions is needed because each step in the handshake process might be split into a number of packets due to packet length restrictions. Therefore, partially captured TLS sessions might make the decryption impossible [5]. Second, we need either the client's session keys of the recorded sessions which can be acquired through setting the *SSLKEYLOGFILE* environment variable, or the private keys of the servers in case the negotiation was established with no Perfect Forward Secrecy (PFS) i.e. RSA key negotiation.

Initially, $NetInfoMiner$ attempts to decrypt HTTPS traffic if the encryption keys are provided. Otherwise, it only processes the plain HTTP traffic.

1) Visited Links: The first information $NetInfoMiner$ targets for extraction is the click paths that clients followed in their browsing sessions. To analyze encrypted HTTPS traffic, with the additional support for HTTP2 and SPDY, a different RCI is implemented by adding the idea of filtering known automatically generated advertisement requests mentioned in [7]. Figure 2 summarizes the new RCI (Referral-based Click Inference) implementation. The original RCI approach [9] simply claims that a visited link can be detected by matching a referer URI with a previously recorded request. We introduce a new concept of a *browsing session* which is a series of visited links by a specific client using a specific browser without the client being idle for more than a period of time τ. Links visited by the same client using a different browser or after being idle for more than τ are represented in a new browsing session. We observe that this representation helps understanding clients' behavior better than a list of visited links. A sample output of a browsing session is shown in Figure 3.

The description of the XML fields in the extracted browsing sessions are mentioned below:

- Time: The timestamp of the first visited link included in the current browsing session.
- Client: The IP address of the machine the client used to perform the current browsing session.
- Duration: The time between the first and last detected visited link in the current browsing session.
- Browser: The application from which the client conducted the current browsing session.
- visit_time: the timestamp of the visit of the link.
- link: the URI of the visited link

2) Credentials: The second information targeted by $NetInfoMiner$ is web applications credentials (usernames and passwords). To effectively detect the existence of credentials in network traffic we studied web authentication mechanisms from two perspectives, how web applications implement and enforce policies on authentication, in addition to the observed trends users tend to follow in choosing passwords and usernames. To clarify this, we explain the different detections levels required to extract the credentials transmitted over the network. We distribute the detection over four levels: protocol level, method level, parameter naming, and heuristics.

- Protocol Level: HTTP1.1 is the main networking protocol. Until October 2016, HTTP2 the successor of SPDY has been adopted by 10.2% of all websites including most of the WWW giants Google, YouTube, Facebook, Yahoo, Wikipedia, Twitter, Instagram and others [12]. So, when detecting login credentials in network traffic, it is a crucial step to identify the protocol used to deliver such credentials from the client to the server. Our solution currently supports HTTP, HTTP2 and SPDY in addition to the possibility of decrypting HTTPS sessions, which to the best of our knowledge, covers almost all known websites.
- Method Level: After studying how web based authentication over HTTP can be accomplished, we came across the following methods.
  - HTTP POST request with Content-Type "application/x-www-form-urlencoded" where the credentials exist in the HTTP request body included in a query string composed of pairs of keys and values assigned to each other by the equal symbol (=), all pairs are separated by the ampersand (&). An example query string looks like "key1=value1&key2=value2".
  - HTTP POST request with Content-Type "application/json", where the data object of the HTTP request includes a set of members, each member combines a key and a string value assigned to each other by the colon symbol (:), all members are separated by the comma symbol (,). Credentials are carried across one or multiple of these members. An example data object looks like "{"Key1":"Value1","Key2":"Value2"}".
- Parameters Naming: Each HTTP POST request contains a query string. This query string includes a group of keys (famously known as parameters) and values. In our experiments we observed that almost all web applications that implement HTTPS don't bother to complicate the naming styles of these parameters, username parameters most likely contain some variation of the keyword username or email examples (Uname, Login_username, session_email, etc.) which make them easier to detect. On the other hand, HTTP applications tend to make parameter naming styles not as clear as in HTTPS applications because developers know that data is sent in clear. Naming obscurity and changing of parameter naming styles make it insufficient to depend only on parameter naming.
- Heuristics: To make the solution naming-independent, further processing can be applied on the extracted parameters by applying some heuristics to identify parameters that possibly contain usernames and passwords. Das et al. studied the password policies for the top 25 websites ranked by Alexa and found out that such policies make passwords identifiable [13]. Based on Das et al. policies in addition to a group of others collected from different studies such as the

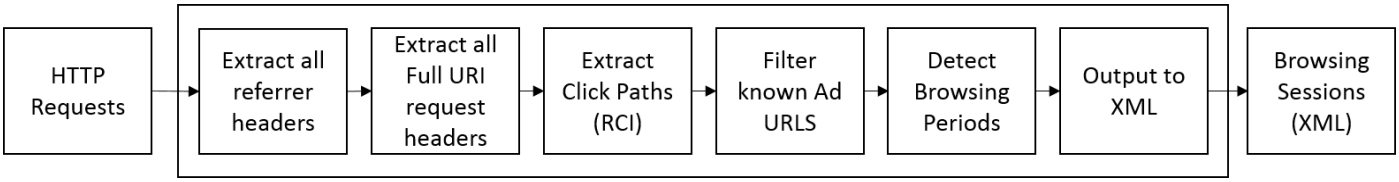

Fig. 2. Visited Links Mining Engine

```
<BrowsingSession>
<time>Wed Sep 21 13:49:12 2016 </time>
<client>192.168.1.6</client>
<browser>Chrome 53.0.2785.116</browser>
<duration>+73 s</duration>
<Clicks>
<visit_time>Wed Sep 21 13:49:12 2016 </visit_time>
<link>https://myaccount.jawwal.ps/.../Url=%2f</link>
<visit_time>Wed Sep 21 13:49:32 2016 </visit_time>
<link>https://myaccount.jawwal.ps/.../InfoPage</link>
<visit_time>Wed Sep 21 13:50:25 2016 </visit_time>
<link>https://support.f5.com/.../sol12569.html</link>
</Clicks>
</BrowsingSession>
```

Fig. 3. Extracted Browsing Session Sample

ones performed on the 10 million stolen usernames and password dataset [14] and the experiment on the password creation trends [15], we implemented these conditions into a heuristic engine and calculated a score for all parameters. Then, any parameter exceeding a certain fixed threshold value will most likely be either a username or a password. More precisely, the heuristics we applied are listed below:

- username as email: it is a convention for usernames to be the email addresses of the users.
- username and password parameter names tend to have more letter characters than non-letter characters.
- passwords might have a combination of upper and lower case letters, symbols, and numbers.
- password might include the phrase ”password” in any of its forms ”Password” ”Pa$$w0rd” ”P@ssword” etc.
- 4.2% of 10 Million passwords were found to end with a number and in 23.84% of these passwords the number was 1.
- Out of 10 Million usernames 97.5% have 4-16 characters without including the domain name in case of emails.
- Out of 10 Million passwords 94.4% have 4-11 characters.
- Passwords might have letters then numbers or vice versa.

A summarized sequence of steps of our credential mining methodology as shown in Figure 4 can be described as follows:

- The first step is detecting all the HTTP login requests (HLR's). An HLR is the request that is sent from an HTTP client to an HTTP server that contains the parameters required by the server to authenticate the client, such as the username or the email, the password and sometimes other parameters. The criteria we defined for an HLR is the following:
  - Protocol type: HTTP, HTTP2, or SPDY.
  - Method: POST.
  - Content-Type: URL encoded form or JSON.
  - URI Full Request that includes a keyword suggesting a login page such as (login, logon, username, etc).
    It is important to mention that in our experiments we have found that this criteria can result some false positives and false negatives. A false positive means a request that matches the criteria of an HLR but it is not actually an HLR, for instance, some web applications login mechanism splits the login process into two requests, the first one is to provide the username and the second one is to provide the password like the case in Google and Yahoo. This false positive is not essentially bad; it gives an information about the login type in that web application as a Two-page login. In the other hand a false negative means that a login request was not detected for some web application, due to the fact that the implemented login mechanism generates an HLR that doesn't match our defined criteria. To make our solution extensible, we built it to accommodate other login mechanisms with minor modifications.
- After acquiring the HLR's, our mining engine doesn't require the entire request body. So, the parameters themselves will be extracted for further processing.
- The first parameter processing stage is Parameter Naming, where possible usernames and passwords parameters can be identified through their naming style (UserName, session_username, Pword, etc.).
- Further undetected parameters will be transferred into the second parameter processing stage, the heuristics engine, where possible usernames and passwords are identified through matching their identified characteristics against a set of known usernames and passwords characteristics.
- The final step is exporting the detected credentials into an XML structure. A sample extracted credential is shown in Figure 5. The actual value of the username and password are hidden in this sample for privacy reasons.

The description of the XML fields is mentioned below:

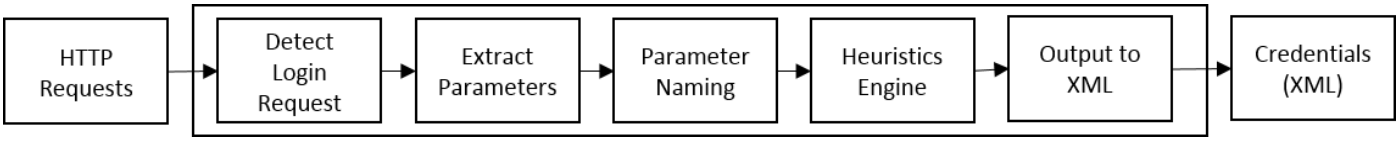

Fig. 4. Credential Engine

- Client: The IP address of the machine the client used to perform the login operation.
- Host: The host name of the HTTP server that the client is trying to log into.
- Browser: The application from which the client used to perform the login operation.
- LoginType: the type of the login process whether a Single-Page or Two-Page login. A Single-Page login is a login type where both the username and the password are sent in a single HLR. On the other hand, a Two-Page login is login type where the username is sent in the first semi-HLR then the both the username and password are sent together in the second complete HLR. We believe this information might be helpful in forensic applications.
- DetectionType: specifies the way the username and password were detected (Parameter nameing or Heuristics).
- LoginTimeUsername: the timestamp when the username was initially sent to the server.
- UsernameKey: the parameter name of the username.
- UsernameValue: the actual username string, might be an email.
- LoginTimePassword: the timestamp when the password was initially sent to the server. In Single-Page logins, the LoginTimeUsername and the LoginTimePassword will be exactly the same, since both parameters are sent together. While in Two-Page logins the timestamps will be different. We believe the timestamp difference will reflect some information regarding the login behavior of the client, i.e. a very small difference might suggest an automated password guessing attack, a relatively longer but still small difference reflects normal login attempt while a relatively longer difference might suggest a manual password guessing attempt.
- PasswordKey: the parameter name of the password.
- PasswordValue: the actual password string.
- UsageCount: Number of times this credentials where used.
- LoginTime: the timestamp of beginning the login attempt into the above host with the above username and password. Since there might be several attempts.

The existence of credentials on network traffic might not mean necessarily a web application login. Alternatively, it might be one or several failed login attempts, or it might be an account hijacking attack. Hence, confirming that existing

```
<Credential>
<Client>192.168.1.6</Client>
<Host>login.yahoo.com</Host>
<Browser>Torch 51.0.0.11603</Browser>
<LoginType>Two-Page Login</LoginType>
<DetectionType>Parameter Naming</DetectionType>
<LoginTimeUserName>DD MM dd hh:mm:ss YYYY</LoginTimeUserName>
<UsernameKey>username</UsernameKey>
<UsernameValue>*****</UsernameValue>
<LoginTimePassword>DD MM dd hh:mm:ss YYYY</LoginTimePassword>
<PasswordKey>passwd</PasswordKey>
<PasswordValue>*****</PasswordValue>
<UsageCount>1</UsageCount>
<UsageList>
<LoginTime>DD MM dd hh:mm:ss YYYY</LoginTime>
</UsageList>
</Credential>
```

Fig. 5. Extracted Credential Sample

credentials actually yielded a successful login to a web application requires more investigation. For forensic applications, we decided to extract each detected occurrence of a credential even if it didn't lead to a successful login. This way, more information can be reflected regarding the behavior of a client when logging to his account. For instance, an indication to whether the client simply forgot his password then remembered it, or possibly a password guessing attack is being launched, etc. Such information can help forensic examiners to reach conclusions in their digital investigations regarding account login activities.

To distinguish a successful login attempt from a failed one, we assumed that a successful login leads to the creation of sessions by web applications. These sessions need to be managed and maintained. We studied different session management implementations. We found that cookies are used by 50.4% of web applications to maintain session states of connected users [16]. Other mechanisms exist for maintaining session states like URL rewriting in addition to GET or POST request parameters, but we only targeted cookies at this stage.

3) Session Cookies: The Third type of information extracted by $NetInfoMiner$ is session cookies.

Web application servers communicates the establishment of a cookie to clients using the set-cookie header in HTTP responses. Upon a successful login, the client receives a Set-Cookie header from the server with the value of the cookie that will be later included in every transaction between the client and the server in the current session. Apart from authentication, Web applications use cookies for other goals. Hence, not all set-cookie headers contain a session cookie. To distinguish a session cookie from others, we look into the naming of the newly set cookies. Famous web development framework can be fingerprinted using their cookie naming, for instance PHPSESSID indicates PHP, JSESSIONID in J2EE, CFID and CFTOKEN in ColdFusion, etc. [17]. The web framework usage statistics justify our direction, since we have collected

session cookie names that covers more than 70% of the mostly used frameworks, for instance PHP is used in 26% of all websites, several ASP versions around 30%, J2EE with 9% etc. [18]. After performing a quick study on the applications of cookies on the web, we were able to come up with a list of cookies that are known to be used for advertisement purposes rather than session management, these cookies will be dropped from further analysis.

$NetInfoMiner$'s approach for detecting session management cookies as shown in Figure 6 can be summarized as follows:

- Extract all HTTP responses with Set-Cookie headers.
- Detect a cookie with a name matching the known names of session cookies (i.e. JSESSIONID).
- Filter out known non-session management cookies (i.e. _ga, __utma, __utmb, etc.).
- The remaining session cookies are grouped in cookie sets, each set represent the cookies defined between a client and a server.
- The final step is exporting the detected cookie sets into an XML structure. A sample extracted cookie sets is shown in Figure 7. The values of the cookies are not shown in this sample for privacy reasons.

B. Implementation

The current version of $NetInfoMiner$ is implemented only for Windows operating system with the plan to extend it to Linux shortly. Currently, the tool is separated into three main engines, one for each targeted information type. The general implementation approach for each engine is shown in Figure 8.

$NetInfoMiner$ takes advantage of the capabilities provided by $Wireshark$ [19], especially the command line tool $tshark$. $NetInfoMiner$ will receive the network traffic trace file as an argument and optionally the TLS key log file (KLF) or RSA private key if available. Then, it will initiate calls to $tshark$ to extract the desired types of information by applying a group of highly customized filters, each filter is responsible for keeping only packets needed in extracting certain type of information (i.e. HTTP2 headers with Set-cookie values). The output of $tshark$ is a text file consisting of a set of lines, each one includes a group of strings separated by a delimiter, each string represent a desired field (Time, Source IP, etc.) extracted from a network packet that matches the criteria defined by the filter. In some cases, $tshark$ output might require some modification due to some implementation limitations (i.e. HTTP2 data objects cannot be extracted in ASCII encoding directly). Afterwards, the modified results will be sent to the appropriate mining engine programmed in C programming language for further processing.

NetInfoMiner is designed to process large amount of network traffic. It depends on $tshark$ to decrypt then filter network traffic. The processing capabilities of $tshark$ mostly depends on the available memory on the running machine. In order to process large traffic traces, the mining engines in NetInfoMiner are implemented with dynamic memory allocation.

Furthermore, there are huge differences between the syntax of HTTP protocol in one hand and both HTTP2 and SPDY on another. For instance, in a single HTTP POST request, all the headers and data of the same request are close together making it easier to extract and process them. While in HTTP2 and similarly SPDY, the requests and responses might be multiplexed and the data segment and the header segment of the same request might be interleaved with other segments of other requests, which makes the process of extracting the same information type i.e. credentials in HTTP2 and SPDY more complex than in HTTP. So, the processing performed by $NetInfoMiner$ for HTTP2 and SPDY has a lot of differences and require further steps not required for HTTP. While the general approach shown in Figure 8 is still the same, but for the same targeted information type, totally different filters and output pre-processing are implemented. The additional processing includes two parts, first, the current implementation of $Wireshark$ (Version 2.2.1) doesn't provide well organized segmentation of the HTTP2 and SPDY headers, which is not the case in HTTP, for instance the $user-Agent$ header in HTTP can be directly accessed by the column identifier $http.user\_agent$ while in HTTP2 and SPDY such identifier doesn't exist yet. The second additional processing is for having some required information type (i.e. credential parameter) in a different packet than the header segment of the same request if they are not transmitted together. This requires $NetInfoMiner$ to initiate an additional call for $tshark$ to extract the required headers of a previously processed data segment.

Table I shows a comparison of the capabilities of NetInfoMiner to other existing tools.

TABLE I. THE PROPOSED TOOL COMPARED TO OTHER TOOLS

Extracted Info.	Protocol	PyFlag	Click-Miner	Network-Miner	NetInfo-Miner
Visited Links	HTTP	-	✓	✓	✓
	HTTPS	-	-	-	✓
	HTTP2/SPDY	-	-	-	In Progress
Credentials	HTTP	-	-	Unfiltered *	✓
	HTTPS	-	-	-	✓
	HTTP2/SPDY	-	-	-	✓
Session Cookies	HTTP	-	-	Unfiltered *	✓
	HTTPS	-	-	-	✓
	HTTP2/SPDY	-	-	-	In Progress

* A dump of all cookies and parameters extracted from traffic without filtering undesired data.

Fig. 6. Session Cookies Engine

```
<CookieSetsList>
<CookieSet>
<Client>192.168.1.109</Client>
<Host>37.75.144.186</Host>
<Cookie>
<CookieInitiationTime>DD MM dd hh:mm:ss YYYY</CookieInitiationTime>
<CookieProperty>JSESSIONID=XXXXXXYYYYY</CookieProperty>
<CookieProperty> HTTPOnly</CookieProperty>
<CookieProperty> Path=/</CookieProperty>
<CookieProperty> Secure</CookieProperty>
<CookieProperty> HttpOnly</CookieProperty>
</Cookie>
<Cookie>
<CookieInitiationTime>DD MM dd hh:mm:ss YYYY</CookieInitiationTime>
<CookieProperty>JSESSIONID=XXXXXXZZZZZ</CookieProperty>
<CookieProperty> HTTPOnly</CookieProperty>
<CookieProperty> Path=/</CookieProperty>
<CookieProperty> Secure</CookieProperty>
<CookieProperty> HttpOnly</CookieProperty>
</Cookie>
</CookieSet>
</CookieSetsList>
```

Fig. 7. Extracted Cookie Set Sample

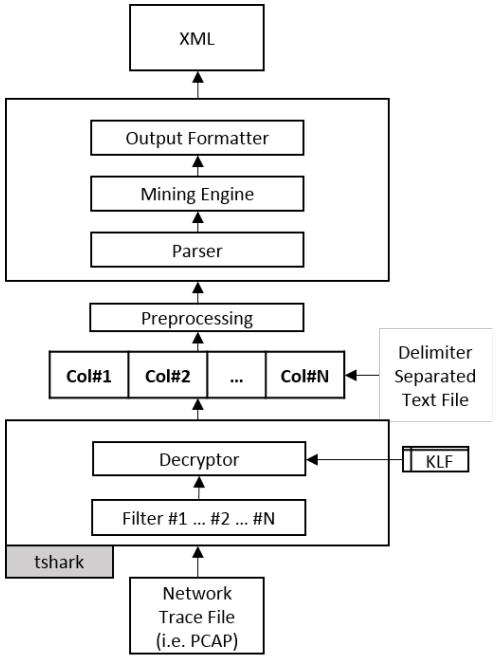

Fig. 8. NetInfoMiner General Engine Implementation

C. Evaluation

The evaluation of the visited links mining engine was done by comparing the results with the browser history, the results show that $NetInfoMiner$ successfully identifies all the visited links with few false positives resulted by uncontrolled redirects as mentioned in the original paper [9], but the current implementation is still a work in progress. The current list of websites already used in the evaluation of the credential and session cookie mining engines is shown in Table II, and we are targeting to cover the Alexa 500 most visited websites list to further evaluate the tool. As shown in Table II, the credential engine has shown high level of accuracy reaching 93% in detecting the existence of credentials in captured network traffic. On the other hand, and due to the fact that the current implementation of session cookie mining engine doesn't support HTTP2 and SPDY yet, all websites that transmits cookies using HTTP2 or SPDY are not yet covered by $NetInfoMiner$, but they will be soon.

TABLE II. TESTED WEBSITES BY NETINFOMINER

Protocols	Site	Category	Credentials Detection	Cookie Detection
HTTPS, HTTP	Arab Bank	Banking	TRUE	TRUE
HTTPS, HTTP	Jawwal	Telecommunication	TRUE	TRUE
HTTP	Kooora	Entertainment	TRUE	TRUE
HTTPS, HTTP	Yahoo	Mail	TRUE	TRUE
HTTPS, HTTP2	Facebook	Social	TRUE	**FALSE**
HTTPS, HTTP2	Tumblr	Social	TRUE	**FALSE**
HTTPS, HTTP2	Twitter	Social	TRUE	**FALSE**
HTTPS, HTTP2	Instagram	Social	TRUE	**FALSE**
HTTPS, HTTP2	Youtube	Entertainment	TRUE	**FALSE**
HTTPS, HTTP2	Pintrest	Social	**FALSE**	TRUE
HTTPS, HTTP	Linkedin	Social	TRUE	TRUE
HTTPS, HTTP2	Google	Search/Social	TRUE	**FALSE**
HTTPS, HTTP2	Dropbox	Storage	TRUE	**FALSE**
HTTPS, HTTP	Sharelatex	Education	TRUE	TRUE
HTTPS, HTTP	KFUPM-Office	Education	TRUE	TRUE

The effect of the negotiation of different protocols by clients and web applications is mentioned in Section (II). To evaluate how $NetInfoMiner$ adopts with such effect, a quick experiment was conducted. Firefox browser enables the clients to disable or enable the support for HTTP2 protocol. We visited and logged in to the twitter website with HTTP2 enabled, then disabled the support for HTTP2 and re-logged again. We then analyzed the captured traffic of the experiment using $NetInfoMiner$. The tool successfully detected both login operations even though one was carried over HTTP2 while the other was over SPDY.

VI. CONCLUSION

This paper discusses high-level information extraction from big data generated by network traffic. We present a tool, $NetInfoMiner$, capable of extracting: visited links, user credentials and session management cookies. The extracted information is useful in many applications such as web penetration testing, digital and network forensics. Although traffic is sent in clear, it is challenging to retrieve sensitive information (browsing history, cookies, etc.) because each web application has a different implementation of these features and uses different naming, formats, etc. The proposed tool can retrieve

and recognize most of the sensitive data inside the network traffic by relying on extensive list of naming styles, variants and using new heuristics.

$NetInfoMiner$ consists of three mining engines, the visited links engine, the credential engine, and the session cookies engine, with the ability to analyze HTTPS traffic.

The evaluation of $NetInfoMiner$ displays 93% accuracy of the tested websites by the credential engine. It requires further work to improve the other two engines to support HTTP2 and SPDY with higher accuracy.

VII. Future Work

The proposed $NetInfoMiner$ is a part of an ongoing research. Although it supports HTTP2 and SPSDY for user credentials, the priority now is to add the support for HTTP2 and SPDY protocols to both the visited links, and session cookies engines. Also, the heuristics engine needs more testing on large sets of usernames and passwords to further improve its accuracy. Moreover, we intend to add support for other session management methods like URL rewriting. The concept of user tracking though Facebook social plug-ins (i.e. Like buttom) is used to track real-life identities of users [20]. We plan to build on this concept the ability of identifying user's real-life identities from cookies set by Facebook tracking system.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for supporting this reasearch. Figures and descriptions in this paper were provided by the authors and are used with permission.

References

[1] internetworldstats.com. World internet users statistics and 2015 world population stats. (n.d.). http://www.internetworldstats.com/stats.htm . Accessed: April 30, 2016.

[2] Cisco Visual Networking Index. The zettabyte era–trends and analysis. *Cisco white paper*, 2013.

[3] trustworthyinternet.org. Trustworthy internet movement. https://www.trustworthyinternet.org/ssl-pulse/. Accessed: October 26, 2016.

[4] Alessandro Finamore and Konstantina Papagiannaki. Is the web http/2 yet? In *Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31-April 1, 2016. Proceedings*, volume 9631, page 218. Springer, 2016.

[5] Sally Vandeven. Ssl/tls: Whats under the hood. *SANS Institute InfoSec Reading Room*, 13, 2013.

[6] MI Cohen. Pyflag–an advanced network forensic framework. *Digital investigation*, 5:S112–S120, 2008.

[7] Christopher Neasbitt, Roberto Perdisci, Kang Li, and Terry Nelms. Clickminer: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1244–1255. ACM, 2014.

[8] forensicswiki.org. Pyflag, this tool is deprecated. http://www.forensicswiki.org/wiki/PyFlag. Accessed: November 1st, 2016.

[9] Guowu Xie, Marios Iliofotou, Thomas Karagiannis, Michalis Faloutsos, and Yaohui Jin. Resurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[10] Erik Hjelmvik. Passive network security analysis with networkminer. *IN) SECURE*, (18):1–100, 2008.

[11] w3techs.com. Usage of site elements for websites. https://w3techs.com/technologies/history_overview/site_element/all. Accessed: October 20, 2016.

[12] w3techs.com. Usage of http/2 for websites. https://w3techs.com/technologies/details/ce-http2/all/all. Accessed: October 20, 2016.

[13] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[14] wpengine.com. Unmasked: What 10 million passwords reveal about the people who choose them. http://wpengine.com/unmasked/. Accessed: October 20, 2016.

[15] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. " i added'!'at the end to make it secure": Observing password creation in the lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 123–140, 2015.

[16] w3techs.com. Usage of cookies for websites. https://w3techs.com/technologies/details/ce-cookies/all/all. Accessed: October 19, 2016.

[17] R Siles. Session management cheat sheetsession id properties. *Online at https://www. owasp. org/index. php/Session_ Management_ Cheat_ Sheet# Session_ ID_ Properties*, 2013.

[18] trends.builtwith.com. Statistics for websites using framework technologies. https://trends.builtwith.com/framework. Accessed: October 20, 2016.

[19] Gerald Combs et al. Wireshark. *Web page: http://www. wireshark. org/last modified*, pages 12–02, 2007.

[20] Güneş Acar, Brendan Van Alsenoy, Frank Piessens, Claudia Diaz, and Bart Preneel. Facebook tracking through social plug-ins. 2015.