CookieMiner: Towards Real-Time Reconstruction of Web-Downloading Chains from Network Traces

Zhipeng Chen, Peng Zhang*
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
chenzhipeng@iie.ac.cn
pengzhang@iie.ac.cn

Chao Zheng, Qingyun Liu
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
zhengchao@iie.ac.cn
liuqingyun@iie.ac.cn

***Abstract*—Network traces are one of the most exhaustive data sources for the forensic investigation of computer security incidents. Recent advances in capturing the network traces techniques have facilitated the forensic processing, including the reconstruction. Unfortunately, off-line web-downloading chain reconstruction could not meet the demands for real time processing. Furthermore, the packets in prior studies are mainly captured on the client or server side, which is difficult to monitor the network traffic in the real-time. Consequently, how to online reconstruct from the packets captured by the gateway is getting challenging.**

In this paper, based on the packets from the gateway, we propose a novel system, CookieMiner. CookieMiner first identifies the web-downloading resources, and then use the cookies to reconstruct the web-downloading chains reversely. All cookies would be firstly split into a series of tokens by semicolon and a threshold value will be set by the SET_K algorithm according to the number of tokens. Next, the HTTP packets whose tokens' number is greater than the threshold will be sorted by timestamp and then inserted into the corresponding web-downloading chain. Finally, the most frequent URLs are extracted as entry points from all the chains with the same web-downloading resources. In addition, by a user study involving 6 pair-wise downloading applications, CookieMiner can reconstruct the web-downloading chains and find their entry points with high precision and low false positive rate.

Keywords—real-time, network traffic, reconstruction

I. INTRODUCTION

HTTP is one of the most widely used protocols, contributing up to 80% of the traffic on some networks[1,8,16]. Consequently, to some extent, getting to know user behaviors on the internet means knowing what people have done over HTTP. Understanding user interactions with internet benefits many significant applications, not only computer forensics, but also web-usage mining[2]. There are many ways to learn about users' behaviors. Geetharani, S et al.[2] presents three ways. The first is to analyze the HTML Documents. The most documents accessed by users indicate users' interest. The second is to have clear of users' profiles, site intentions and server logs. The third is to use speed tracers to recognize user sessions and rebuild the user traversal paths. However, these methods are time-consuming and cost too much. They incline to be vague and deficient. HTTP network traces are one of the most exhaustive data sources for the forensic investigation of computer security incidents. Recent techniques in capturing the network traces have made great progress. The ability is important to enable the traffic inspection, for example, automatically rebuilding the web-downloading chains in the real-time or a detailed postmortem analysis of significant network events. So the more information people have about the HTTP traffic, the more effectively they can understand the network.

Much work regarding rebuilding the web-downloading chains and finding the entry points has been done. Firstly, prior work on the web traffic can be classified into two categories, namely the reconstruction approach[1] and the replay approach[4]. Reconstruction approaches based on Referer work as follows: for every Referer field in the HTTP request, the approach constructs the referrer graph as in ReSurf[1], where a node represents an HTTP request, and two nodes are linked together by a directed edge according to their Referer relationship. They are fast and light-weight, with low accuracy. The main reason is that most HTTP request heads are lack of Referer field, which is an optional field of HTTP head. According to the paper[6], only 17.7% HTTP requests have Referer fields. In order to address these above limitations, replay approach[4] has been proposed: for the captured HTTP traffic, the browser with a browser driver plugin implemented on top of Selenium WebDriver[5] replays the network traffic and reconstructs the user-browser interactions. However, the approach is time-consuming and limited by plug-ins. Secondly, to detect the common entry points of the web-downloading chains, several detection schemes have been proposed. Zhang, J., et al.[3] proposed a method to identify the entry points by aggregating drive-by download samples into different groups according to the downloading hash values and the structure named Hostname-IP Cluster(HIC). The detection relies on the hash values and the features of the landing URLs(e.g., hostname, ip). However, malicious users can easily fabricate these features or values. In order to address the above limitation, another method[6] is presented. It analyzes several correlated redirect chains instead of an individual redirect chain. Moreover, prior rebuilding the web-downloading chains and finding the entry points are facing new challenges which are listed below:

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

- Large-scale pages: The web is tremendous in size, and new pages (both legitimate and malicious) are added at a daunting pace[7]. In addition, the place capturing the packets is mainly on the server side or the client side. There are not enough resources to crawl adequate URL context information. Therefore, they are time-consuming and consume huge resources.
- Complex web pages: The modern web is serving highly dynamic pages that make full use of scripting languages, a variety of browser plug-ins and asynchronous content requests[4]. Often rendering a single page generates tens of HTTP requests towards different web servers[1]. Secondly, there are still some techniques bypassing the crawlers or providing an alternative URL[6], which means these URL context information is dirty.

The first challenge affects the efficiency of rebuilding the web-downloading chains and finding the entry points, due to the time and resource consumption, while the second one leads to the low accuracy. Consequently, the efficiency and accuracy of rebuilding the web-downloading chains and finding the entry points make it challenge to analyze the traffic.

Research Goals and Approach. In this paper, we propose CookieMiner, a novel online system that aims to automatically reconstruct the web-downloading chains and find the entry points. What's more, we also compare CookieMiner with Resurf[1], ClickMiner[4]. Both of them need prepare the captured traffic in advance from the client or server side, while our CookieMiner could find the entry points in real time based on the traffic captured from the gateway. Our CookieMiner tracks from the downloading resource to finding the entry point, while Arrow[3] and WarningBird[6] tracks from the entry point to the malicious resource.

The Resurf[1] approach is based on the referrer graph constructing the web chain. Two nodes are linked by their referrer relationship. The ClickMiner[4] filters out requests that are likely not user-click requests based on the Resurf[1] approach. Our CookieMiner, instead, takes a very different approach based on the cookies, Referer and Location. According to the most recent Web survey[9] only 22% of the Web users surveyed accept cookies from any source, while 23% receive a warning before cookies are set, allowing them to make a decision on a case-by-case, and approximately 6% refuse all cookies. Surprisingly, the remaining survey participants either didn't know what a cookie was or didn't care about having a policy[10]. Therefore, the approach based on the cookies is practicable.

Contributions. We make the following contributions:

- We focus on the packet captured from the gateway, not the packet captured from the client side or server side.
- We propose CookieMiner, that aims to automatically reconstruct the web-downloading chains from the packet captured from the gateway and find the entry points. Importantly, a new algorithm is presented to make the token threshold self-adaptive varying from the application.
- We report a user study that aims to demonstrate how CookieMiner can work effectively.

The rest of this paper proceeds as follows. In section II, we present the problem. Section III gives the CookieMiner in detail. In section IV, the user study is shown. Finally, we conclude this paper in Section V.

II. PROBLEM FORMULATION

Goal. Our main goal is to automatically reconstruct the web-downloading chains and find their entry points, based on only the packets captured from the gateway in real time.

Assumptions. We assume the traffic we acquire is normal. Namely, the HTTP cookies are not intentionally removed or banned in the network.

Downloading Resources. In this paper, we use the terms downloading resources and web-downloading interchangeably. Our definition of downloading resources is intentionally lax, and broader than the downloading. It also includes browsing some videos or searching some larger resources by browser.

Entry Point. In this paper, we use the terms entry point and the original page of the downloading resources interchangeably.

HTTPS traffic. More and more web services are transitioning to HTTPS, most recorded web traffic is also going to be encrypted, and all of the Resurf, ClickMiner, CookieMiner will become less useful in the near future. However, we also could get the packet traces by many techniques or tools, such as Fiddler[11]. Many modern enterprises already deploy proxies that could inspect the HTTPS traffic[4].

Caching. Unlike the Resurf[1], ClickMiner[4], the browser's cache has little impact on CookieMiner. For every downloading resource, we use a circular queue to store a series of web-downloading chains. The cache could only affect one web-downloading chain, but not all the chains.

Network traces. Typically, the captured network may contain a mix of traffic generated by many third-parties or embedded services. However, based on the cookie relationship mechanisms, we could easily filter out some irrelative traffic traces, such as asynchronous requests, to reconstruct the web-downloading chains.

III. COOKIEMINER SYSTEM

In this section, we give a detailed description of our CookieMiner System. We start an overview of how CookieMiner works in section III-A. In section III-B, we discuss the CookieMiner in detail.

A. CookieMiner System Overview

Fig. 1 presents the architecture of the CookieMiner System. The input of the system is the continual HTTP packets, and the output is a series of web-downloading chains and entry points. There are two processing components in the system. The first component, Papp, is responsible for collecting the current packets from the gateway and sending these capsulated by the Google protocol buffers[12] to the Sapp in real time. We choose to leverage Protool Buffers because it is compatible with most major formats, automatically generates serialization and deserialization code avoided the need for hand parsing, and

if you want to parse some new fields from these HTTP flows, these fields could be easily introduced, and intermediate servers that don't need to inspect the data. In a word, protocol buffer is a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler. The second component, Sapp, aggregates all the packets in the queue from the Papp and then identifies the downloading resources, reconstructs the web-downloading chains, generates the downloading resource entry points by the frequency and timestamp for all of the web-downloading chains.

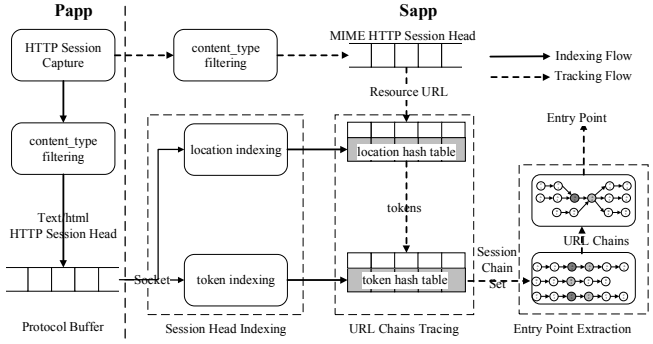

Fig. 1: CookieMiner system overview

A simplified view of the process described below is provided in Fig. 2. The Papp's function is similar to tcpdump command. There are two working mechanisms. One is RealTimeModel, whose function is continually capturing the HTTP flows in real time(1), the other one is ReadPcapModel, whose input is pcap file, storing those captured HTTP flows. The model is fit for off-line testing. Firstly, the HTTP flows is capsulated with the Protocol Buffers[12], then these flows are sent to Recv Queue(2) by sockets. When the Recv Queue is not empty, Sapp automatically initiates some threads to acquire the flows from the Recv Queue, and then every HTTP traffic flow goes in different ways according to their head fields' features(3). If the content type of HTTP head session is MIME, such as video, application/octet-stream, this flow is a downloading resource. The HTTP head session is stored in the DownLoad Queue(4). If the content type of HTTP session is text/html and the cookies exist, the cookies are split into a series of tokens and stored into the Token Hash Table(14), which takes the token as key, HTTP session containing this token as value. If the Location field exists in the HTTP session, then the Location Hash Table is built(5), which takes the URL specified by Location as key, HTTP head session as value. When the DownLoad Queue is not empty, some threads are initiated automatically. These threads are used to acquire the downloading resource HTTP flows. The URL in the DownLoad Queue is always set as the last node in the web-download chain. The step(6) parses out HTTP head fields from the HTTP flows. If the downloading-URL exists in the Location Hash Table(8), then the corresponding value is regarded as the last second node in the web-download chain. If the downloading-URL flow has the Referer field, then the URL specified by the Referer is added to the web-download chain(7). The other nodes in the web-download chain are acquired in a different way. We first select the new node(downloading node, referrer node, or the location node) in the web-download chain and split its cookies into tokens, then take the token as key to search in the token hash table(9) and select HTTP traffic flows containing at least two or more(the value of K) tokens(10), and sort these HTTP traffic flows by timestamp to build the web-download chain(11). When the web-downloading chain is built(12), we could get the count of each URL in all the chains for every downloading resource. The most frequent URL is extracted as an entry point. If two or more URLs in these chains have the same frequency, we will select the URL near the downloading resource as the entry point(13).

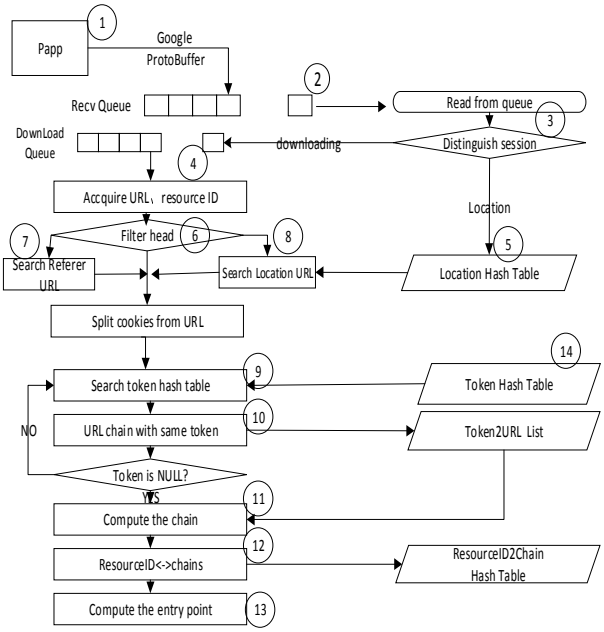

Fig. 2: CookieMiner's process(simplified)

Challenges: Intuitively, due to the cookie relationship mechanisms, CookieMiner doesn't need to consider those complicated components embedded in the web pages any longer, there still exists some new problems. Due to Papp's working mechanism, we have to study how to guarantee that capturing the HTTP network traces from the gateway is bidirectional, namely, every flow contains an HTTP request and its corresponding response. However, the complexity of the HTTP flows routing paths and the dynamic variety of the network bandwidth aggregate the difficulties. What's more, the same downloading resource can be distributed by many different servers, which means many different web pages are possibly embedded with the same downloading resource, thus how to distinguish them is confusing. In addition, modern large-scale and complex web pages as well as the real-time capability pose a number of challenges to automatically reconstruct the web-downloading chains and find their entry points. In the following sections, we will discuss how CookieMiner copes with these challenges.

B. CookieMiner System Details

1) Papp capture HTTPnetTraces

In this paper, we use the term HTTP packets and HTTP flows, HTTP traffic interchangeably. HTTP flow is composed of a sequence of HTTP requests and responses. When a user requests a website or clicks a URL hyperlink, it causes many

HTTP requests and responses. Every packet goes in different route paths, including HTTP request packet and its corresponding response packet. Therefore, two factors decide whether the captured traffic flows are bidirectional or not. One is the network condition, the other one is the Papp module. The former is provided by the Internet Service Provider(ISP). If the ISP provides a low-quality network causing many packets lost during the round-trip, the client or the server has to send the requests or responses repeatedly. At the worst, we lose some packets. This leads to the low efficiency and accuracy. The network condition is constant at a certain condition. Therefore, we have to consider the latter factor. The simple way is choosing the appropriate location for the Papp module setting up to make sure every packet could be captured, including requests and their corresponding responses. That means that we should know where the border route is. The place of the border route is the place where Papp deploys.

2) Identify the Downloading Resource

In the modern web, the same downloading resource can be distributed by many different servers, which means many different web pages are possibly embedded with the same downloading resource. Furthermore, some URLs could be frequently changed, but the downloading resources in these web pages do not change. To address these limitations, in this paper, we adopt the cumulative hash algorithm[14] to compute a 64 bit checksum to identify the downloading resources, where the algorithm only performs calculation on the first 2M bytes(could be set manually) sampling resources.

3) Rebuilding Downloading Chain

Flows Preprocess: The arriving HTTP flows are cached in the Recv Queue. Then multiples of threads are initiated automatically to fetch these flows in the Recv Queue. Traditional web chain reconstruction is backwards just as the URL visiting sequence, while CookieMiner is not. It is forwards from the last visiting page to the first visiting one. At first, CookieMiner parses out all fields' value of the HTTP heads. Six kinds of fields are at most concerned by us in the HTTP heads: content type, content length, referrer, location, cookies and timestamp.

If the content type in the HTTP head is video or application/octet-stream, we can detect that the flow is a downloading resource and then compute a 64 bit checksum to identify the downloading resource. After that, the downloading resource with the checksum identification is sent to the DownLoad Queue.

If the Referer field exits in the HTTP head, we could build a series of Referer Graphs[1] based on the Referer fields. We build a directed acyclic graph $G = (V, E)$ where each node in V represents an HTTP flow. Assume FlowA and FlowB are two nodes in such graph. A directed edge $(A \rightarrow B) \in E$ exists if the absolute URL related to FlowA is referred in the Referer field of FlowB.

If the Location field exits in the HTTP head, we could also build a series of Location Graphs based on the Location fields. The building principle is similar to the Referer Graph. What's more, we build a location hash table to store every pair of such flows, such as from FlowC to FlowD. We take the URL specified by the Location field of the FlowC as key(which is the same as the URL in the FlowD), and corresponding URL in the FlowC as value. If the content type of HTTP head session is text/html and the cookie field exists, the cookies are split into many tokens by semicolon. All of these tokens are stored in the TokenList. In addition, all HTTP traffic flows containing cookies are added up to the TokenList. Then the Token Hash Table is created which takes the token as key, and the corresponding TokenList as value. Fig. 3 shows in detail.

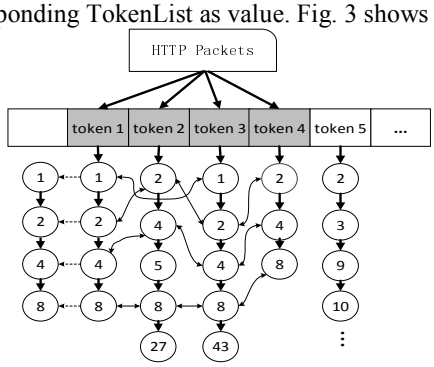

Fig. 3: Token Hash Table

Seeking Chains: Some threads are initiated automatically to get the HTTP flows from the DownLoad Queue, which are regarded as the last nodes named EndNode in the web-downloading chains. For every EndNode, there are three ways to build the next series of nodes in the web-download chain.

If the Referer field exits in the EndNode, then the flow containing the value of Referer is added up to the web-download chain.

If the EndNode's URL exists in the Location hash table, then the corresponding hash value is added up to the web-download chain.

Algorithm 1: Set_K Algorithm

Input: K=2

Output: the threshold value(the best value of K)

1: **run** the cookieMiner, set K=2, get the downloading resource's and the entry point page's cookies, respectively

2: **split** the two cookies into tokens, stored as tokenListA, tokenListB, respectively.

3: **insert** the two tokenLists into two hashtables, respectively, find their common tokens iteratively.

4: **set** the number of these common tokens **as** the threshold value

5: **return** K

No matter whether any of two ways creates the second node in the web-download chain or not, then one of the last two nodes' cookies(if exist) is split. Then we begin to take every token as key to search in the token hash table to get HTTP TokenList, which consists of a series of URLs in the HTTP flows. For example, assume three TokenList exist in the Token Hash Table, such as token1, token2, token3 in the Fig. 3. Then we create another hash table named frequency Hash Table, which takes the URL in TokenList as the key and the combination of the URL's frequency and the timestamp as the value. We count all the frequency of the URLs in these TokenList(token 1, token 2, token 3), as we know, some URLs

exist in multiplies of TokenLists. That means, a4, b3 and c3 may be the same URL, shown in Fig. 4. We set a pre-defined threshold $K = 2$. The K, not more than the size of token hash table, is determined by different applications. The best value of K is the number of the common tokens between the entry point and the downloading resource. Algorithm 1 presents how to set the threshold value, namely, the best value of K. The time complexity is linear. Then the URL whose cookies is not less than K will be sorted by timestamp and inserted into the download chains. Algorithm 2 shows the detailed process.

Algorithm 2: Reconstruction Process

Input: HTTP Flows

Output: web-download chain

```
Search referrer URL in the download HTTP Flow
Search HTTP location Hash Table
token[n]=Split cookies by semicolon
for i:=0->n do
    Token List forms: search Token Hash Table
end for
for j:=0->i*n do
        if exists HTTP[j] in HTTP frequency hashtable
        frequency++
end for
if frequency >=K then
    select HTTP[j], add to the queue
end if
sort by the timestamp in all selected HTTP, insert to the chain
return web-download chain
```

4) Finding the Entry Point

When multiple users visit large varieties of downloading resources, the same downloading resource implies a series of web-download chains. We build a dynamic circular queue to store these web-download chains containing the same resource.

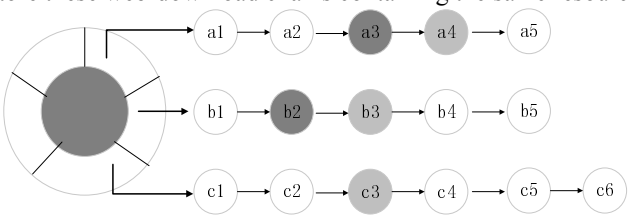

Fig. 4: Circular Queue

Fig. 4 shows the Circular Queue in detail. One circular queue contains multiplies of web-downloading chains with the same downloading resource. The size of circular queue could be set in the configuration file. The mechanism is FIFO(first in, first out). As we know, different TokenList may have the same URLs. Just as shown in Fig. 4, two groups are the same. One group is a3 and b2, the other one is a4, b3 and c3. The goal is to find the entry point among all the chains in the circular queue.

We count the frequency of each URL in the web downloading chains in the circular queue. The most frequent URL is extracted as the entry point. If two or more URLs have the same frequency, we will select the URL nearest the downloading resource as the entry point. Fig. 5 presents the detailed formation process. We could find that the result would be a4 in Fig. 5.

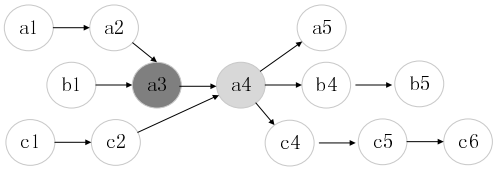

Fig. 5: the Entry Point(a4)

IV. EVALUATION

To evaluate CookieMiner, we conducted a user study involving 6 different downloading resource in a controlled environment, which allowed us to record most HTTP download request. The experiment runs on a machine with 8-core 2.2GHz Xeon processor with 126GB memory. For every downloading resource, we performed two times according to the different value of K. Just as shown in table 1, the ID is the name of HTTP traces captured from the gateway; Req is the number of requests in the captured network traces; Record Req is the number of requests recorded in our study; Mined Req indicates the number of request referred by CookieMiner. Match Req indicates the number of mined requests that match a recorded request. Finally, TPR is the true positive rate, i.e., the percentage of recorded downloading resource requests that match a requests referred by CookieMiner, whereas FPR is the false positive rate, i.e., the number of mined requests that do not match any recorded requests.

TABLE 1: COOKIEMINER RESULTS

ID	K	Req	Record Req	Mined Req	Match Req	TPR (%)	FPR (%)
baidu	2	1330	20	181	20	100.00	12.29
	8	1330	20	161	20	100.00	10.76
360	2	550	24	30	24	100.00	1.14
	4	550	24	26	17	70.83	1.71
rutube	2	3254	37	298	37	100.00	8.11
	4	3254	37	245	37	100.00	6.47
wired	2	2648	22	42	22	100.00	0.76
	5	2648	22	42	22	100.00	0.76
AOL	2	3332	21	22	21	100.00	0.03
	4	3332	21	22	21	100.00	0.03
msn	2	2068	17	62	17	100.00	2.19
	4	2068	17	51	17	100.00	1.66
Avg	3	2197	23.5	98.5	23	97.57	3.83
Stddev	2	1052	6.7	97	7	8.42	4.39

From the table 1, we can see that the K has an impact on the FPR. That is because it affects the length of the reconstruction downloading resource chains. At a certain time, the larger K is, the less chain is. If the K is getting larger and larger, it will make the TPR decrease, just as the 360 trace. If the downloading chain is acquired by Referer field, the result has nothing to do with K, just as the wired trace and AOL trace. What's more, we have to present that ClickMiner has enough

time to compute offline. As a result, large numbers of requests are mined effectively. Instead, CookieMiner is in the real time. Cookie relationship mechanism filters out many irrelative requests. Consequently, the average FTR compared with ClickMiner is a little bigger. In addition, the average of computing time in every HTTP requests is much faster than that using ClickMiner method as shown in Table 2.

TABLE 2: THE TIME OF COOKIEMINER VS. CLICKMINER

Time	ClickMiner (no-cache)	ClickMiner (cache)	CookieMiner
Total(min)	76	34	2.83
Avg(μs)	95473.39	79186.40	12891.03

We also analyzed the compute time of finding the downloading resource chains and their entry points. Fig. 6 presents the time of finding the downloading resource chains and entry points. The horizontal axis is the number of recorded HTTP requests. The longitudinal axis is the time(microsecond), which has been computed by logarithm.

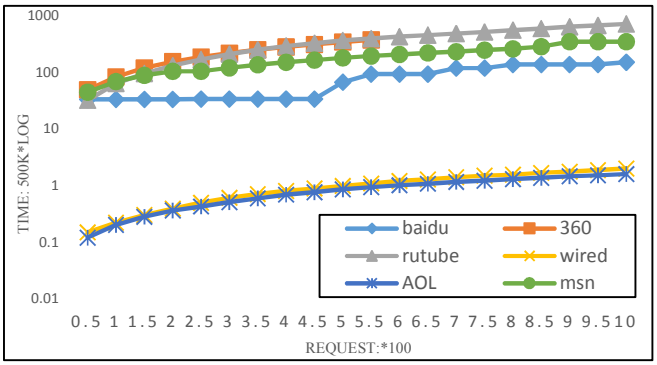

Fig. 6: Time on finding the entry point

From the Fig. 6, we can find the compute time is similar according to the way of finding the chains. For example, the wired trace and AOL trace are both computed by their Referer fields. Although the computing time based on the Referer is shorter, the average of mined HTTP request is less compared with our CookieMiner(Fig. 7).

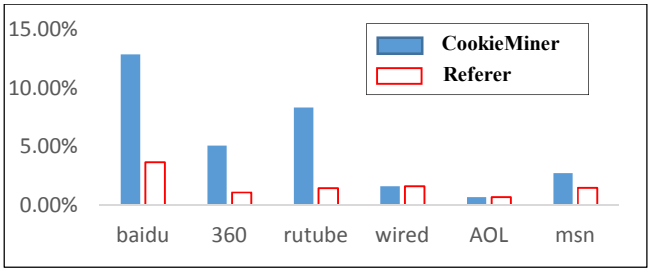

Fig. 7 : Mined Request(avg)

Fig. 7 shows the comparison of mined requests average between the Referer Method and our CookieMiner. We can find by Referer method, the length of the reconstruction downloading chains is short. On average, we observe the length of chains based on the Referer is only 2. As a result, the precision would be rough and the method based on the Referer could not facilitate the forensics research effectively.

Out of total HTTP Traces, almost all traces' TPR with appropriate K value is 100% with low FPR. The FPR is mainly due to those traces generated by machine automatically(e.g., OS/updates, ads, Flash, JavaScript). In addition, CookieMiner is dominated by the websites mechanisms. If there is no Referers, Locations or cookies, it will not work. Fortunately, that is rare, because most inter-domain traffic is HTTP[16].

V. CONCLUSION

In this paper, we proposed a novel system named CookieMiner for rebuilding the chains and finding the entry points. There's tremendous value in aiding the forensic analysis of web traffic traces, for example to help in the investigation of the user-browser interactions in real time. Through a user study, we show that CookieMiner can correctly reconstruct the downloading chains with high true positive and low false positive, and that it outperforms a previously proposed refferer-based approach and ClickMiner in some features.

ACKNOWLEDGMENT

The research work is supported by Supported by Strategic Priority Research Program of the Chinese Academy of Sciences under Grant (No.XDA06030602) and National Natural Science Foundation under Grant (No.61402464).

REFERENCES

[1] Xie, G., et al. Resurf: Reconstructing web-surfing activity from network traffic. 2013..

[2] Geetharani, S. and S. Priyadharshini, A Survey on Web Usage Mining. International Journal of Emerging Trends in Science and Technology, 2015. 2(03).

[3] Zhang, J., et al. Arrow: Generating signatures to detect drive-by downloads. 2011.

[4] Neasbitt, C., et al. Clickminer: Towards forensic reconstruction of user-browser interactions from network traces. 2014.

[5] Seleniumwebdriver,2013. http://docs.seleniumhq.org/projects/webdriver/.

[6] Lee, S. and J. Kim, Warningbird: A near real-time detection system for suspicious urls in twitter stream. Dependable and Secure Computing, IEEE Transactions on, 2013. 10(3): p. 183--195.

[7] Invernizzi, L., et al. Evilseed: A guided approach to finding malicious web pages. 2012.

[8] Ihm, S. and V.S. Pai. Towards understanding modern web traffic. 2011

[9] Pitkow, J., et al, GVU's WWW User Surveys. http://www.cc.gatech.edu/gvu/user_surveys/.

[10] The World Wide Web. http://dangermouse.brynmawr.edu/~dblank/pub/www/

[11] Fiddler. http://www.telerik.com/fiddler

[12] Google Protocol Buffer. http://developers.google.com/protocol-buffers/docs/overview?csw=1

[13] Google Protocol Buffer. http://www.ibm.com/developerworks/cn/linux/l-cn-gpb/

[14] Debiez, J., J.P. Hughes, and A. Apvrille, Data integrity check method using cumulative hash function. 2003, Google Patents.

[15] Zheng, C., et al. , Poster: An Efficient and Scalable Cyberlocker Traffic Tracking Method.

[16] Labovitz, C., et al. , Internet inter-domain traffic. ACM SIGCOMM Computer Communication Review, 2011. 41(4): p. 75—86