2014 IEEE 11th Intl Conf on Ubiquitous Intelligence & Computing and 2014 IEEE 11th Intl Conf on Autonomic & Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops

Shutter: Preventing Information Leakage based on Domain Gateway for Social Networks

Tao Wu, Jianxin Li, Nannan Wu, Tao Ou, Borui Yang and Bo Li
State Key Laboratory of Software Development Environment, School of Computer Sci. & Eng.
Beihang University, China 100191
{wutao, lijx, wunannan, outao, yangbr, libo}@act.buaa.edu.cn

***Abstract*—With the explosion of Online Social Networking (OSN), participation in social networking sites has dramatically increased in organizations like enterprises, campuses, etc. Lots of privacy or security information can be shared by a vast network of people including friends and strangers, which brings potential threats of information leakage. However, existing information protection methods cannot effectively detect the fine-grained user request contexts to OSN sites, resulting in security vulnerability in the organizations. In this paper, we propose Shutter, an information protection system based on domain gateway for social networks. Shutter employs a fine-grained traffic filter to analyze a massive number of HTTP requests and a scalable message parsing scheme for both HTTP and HTTPS traffic, which utilizes the characteristics of OSN requests to accelerate the parsing phrase. Shutter also leverages a rule matcher based on layered trie and multi-pattern matching algorithm to achieve high matching throughput, as well as fast rules inserting and deleting operations. We perform our experiments by collecting real request traffic and using the traffic data as our test input. The results demonstrate that Shutter achieves high throughput and preeminent scalability detecting user requests to OSN sites with insignificant memory overhead.**

I. Introduction

Recent years have witnessed the prevalent of OSN sites such as Facebook and Twitter, which allow businesses to establish their presence online to reach customers and prospects. However, frequent OSN participation in organizations like enterprises increases the possibility of inadvertently revealing the sensitive information [11], [19]. On one hand, the advancements in the Internet services like BASA [15] facilitate people to reach the OSN sites through their own devices as shown in Fig. 1. Scouring tweets, blog entries, and status updates of careless employees can inadvertently expose company secrets such as the identities of key personnel, and even its financial status and internal problems [3]. On the other hand, thanks to the property of fast information diffusion in OSN, adversaries can leverage shared user generated contents (UGC) or user profiles in OSN for social engineering or other malefactions like hijacking one's private account.

Generally, there are two ways to protect the confidential data of the organizations. The first one is privacy preservation mechanisms or add ons in OSN sites to strengthen the security of personal information [6], [18], [20]. But these methods only focus on the user profiles or contents sharing authorizations, and fail to inspect users' uploading contents. The other is the traffic detection systems based on domain

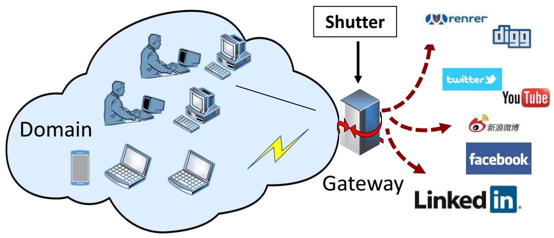


Fig. 1. Domain environment

gateways such as Network Intrusion Detection/Prevention Systems(NIDS/NIPS), which employ state machines to parse protocol data units (PDU) and then match the parsed fields by regular expression based rule engines [7], [13]. Despite the limitations of scope constraints, this method can effectively detect the contexts of traffic flow and block dangerous accesses, which are the root cause of information leakage.

However, traditional NIDS/NIPSes are not effective for detecting the user requests to OSN sites. Firstly, OSN sites tend to leverage long request messages to carry user behaviors and UGCs, but state machines require TCP reassembly for complete requests and parse every field of the messages, which is a time-and-memory-consuming work [9]. Secondly, NIDS/NIPSes report traffic at the protocol granularity [4], [5] like the protocol types and protocol header fields, but they fail to provide fine-grained traffic information for complicated OSN requests nowadays, e.g., Jane comments something to David's certain status. Lastly, although deterministic finite automaton (DFA) based matchers like ROOM's [16] or Netshield's [8] achieve high matching speed, they cost lots of time to rebuild the matchers once rules are added or deleted.

To solve the problems of NIDS/NIPSes above, we designed and implemented a gateway based system named Shutter to prevent information leakage for social networks. Shutter provides fine-grained user requests identification to specify the UGCs and user behaviors by a request elements parser. The parser takes advantage of TCP segments intervals and skips futile requests fields to accelerate long user requests parsing. Shutter also provides a new rule matcher leveraging a layered trie for user behavior matching and a multi-pattern matching algorithm for UGC inspecting. The rule matcher is able to achieve high throughput with fast operations of rules inserting and deleting. In addition, we also handle the non-trivial problems caused by HTTPS traffic and OSN sites' packets capture. The evaluation results indicate that our system is effective in preventing the information leakage through OSN requests and performs better in terms of throughput and scalability with insignificant memory overhead. The main contributions of this paper are as follows:

- Based on the analysis and statistics of OSN requests, we avoid futile fields parsing and leveraged TCP segments intervals to reduce the parsing time of long user requests.
- We designed a new rule matcher that leverages two different rule matching algorithms detecting user request contexts. The rule matcher achieves high matching speed, as well as fast rules inserting and deleting operations.
- We generated rules based on collected traffic data and simulate real environments for evaluations. The result shows that Shutter achieves high throughput and scalability for requests detection with small memory cost.

The remainder of the paper is organized as follows: We first describe related work in Section II. Next we present our analysis of user requests in Section III. Then we propose the design architecture of Shutter in Section IV followed by implementation details in Section V. Finally, we evaluate our system in Section VI and summarize our work in Section VII.

II. Related work

OSN Information Leakage. Lipford et al. [10] argued that the privacy settings of the OSN sites cannot facilitate users to customize a complete protection of their information access control. Xia, Ning et al. [12] investigated the identifiable digital footprints and the association between users and their mobile devices, finding that traffic data can reconstruct the user profile revealing user identity and other personal information. By using relational classification method, Zheleva et al. [21] proposed a practical model to predict the private attributes of users with a mixture of public and private user profiles.

OSN Privacy Protection. Fong et al. [6] proposed an access control model to formalize and generalize the privacy preservation mechanism of Facebook-style social network systems. Krishna et al. [**?**] analyzed the privacy protection state of social content-sharing applications and proposed anonymization techniques to protect users from phishing and other attacks. Shirin et al. [20] preserved user privacy with cryptographic techniques and stored data in a decentralized fashion. However, all these methods focus on the access control paradigm or personal information concealing, lacking attention to the security vulnerabilities of the upload contents.

Intrusion Detection/Prevention Systems. Snort [7] uses the PCRE library for regular expression matching guarded by a string matching based filter, lacking the ability to do multi-field matching. Moreover, the worst case performance is mainly decided by the PCRE library, which is a nondeterministic finite automaton and performs quite slow [14]. Bro [13] is another popular NIDS with a signature rule engine to match the packets' segments. It also uses an event engine to reassemble the complete packets and a policy script interpreter to generate rules from semantic information. As a result, it is hard for Bro to optimize its speed for detecting long user requests.

Rule Matching. Zhichun Li et al. [8] proposed NetShield that splits all the rules into different protocol fields and employ different DFA matchers for each fields to match multiple protocol signatures simultaneously. Hao Li et al. [16] proposed ROOM that divides the whole rule set into sub-sets according to the rule fields and constructs DFA matchers on the rule subsets, which are carefully organized for matching optimization. However, Netshield costs extra space to save the intermediate result of each matcher and extra time to merge them, and both of them cost lots of time to rebuild the DFA matchers whenever rules are appended or deleted.

III. User request analysis

In this section, we extract traffic marks from user requests and leverage the specific marks to identify the user request contexts. Traffic marks are the strings in HTTP requests constructed in name-value fashion. Particularly, cookies and entities in HTTP requests are composed of many name-value pairs, namely traffic marks. By inspecting the values of specific traffic marks in a request, we are able to perceive the user request context and prevent information leakage in this request.

A. Request context description

The information leakage to OSN sites in organizations can be attributed to the careless updating or uploading activities, i.e., updating sensitive statuses, uploading inappropriate photos or adding commercial adversaries as friends. And according to the rfc2616[1], only PUT and POST HTTP methods update or increase the resources in the sites' servers. But PUT method is seldom used in OSN sites because of the side-effects that multiple identical requests behave like a single request. Therefore, we analyze the POST requests for traffic marks in the following.

We assume that the user requests can be described by a **quadruple** $\langle Subject, Action, Object, Content\rangle$:

- $Subject$ is a finite set of users.
- $Action$ is a finite set of primitive behaviors conducted by a $Subject$.
- $Object$ is a finite set of object identifiers and $Subject$ is a subset of it.
- $Content$ is a union set of UGCs appeared in the HTTP request entities.

The first three elements refer to the user behavior of a request and the last one states the UGC, thus the user request context can be described by these four elements. For example, if Alice commented on Bob's certain status, then "Alice" is defined as $Subject$, "comment" as $Action$, the commented status as $Object$ and the comment stuff as $Content$. Elements can be null in some situations like Alice requested Bob as her friends, where the $Content$ element is vacuous. And we denote the vacuous or arbitrary element value as a $wildcard$ in the quadruple. Next, we figure out the traffic marks in a user request to depict the four elements.

[1]http://tools.ietf.org/html/rfc2616

B. Marks identification

We dissected all kinds of user requests collected in Section VI into name-value pairs and leveraged the OpenAPI services to identify the traffic marks. OpenAPI is the data services provided by OSN sites as write and read interfaces for users to update or fetch the site's resources. We took advantage of the OpenAPI and figured out the traffic marks to the quadruple elements in the following steps. Firstly, we fetched user activities data from the OSN sites through OpenAPI over the same period of time as the collected requests. Secondly, according to the annotations of OpenAPI parameters, we classified the fetched data into different elements of the quadruple. Finally, we matched the known element value with the name-value pairs of user requests and got the marks that represent the elements. Table I lists 10 types of traffic marks that are most commonly found in Facebook. The marks for $Object$ and $Content$ elements are closely correlated with the request types and most of them locate in the entities of HTTP requests.

In order to select the conspicuous marks for certain elements in the quadruple, we evaluate the traffic marks in the metrics of uniqueness and persistence for certain types of requests such as comments, updating statuses.If a mark is assigned to an element, the uniqueness score describes the possibility that the traffic mark leads to the specific element; the persistence score describes the possibility that a certain type of requests contain the traffic mark. We denote p as a certain element in quadruple, r as a request and P_r as the collection of marks in r that each can represent p. Using the indicator function, the two metrics are defined:

Uniqueness. Given a collection of certain types of requests $R(m)$ where each request r contains the mark m, the uniqueness denoted by $\Psi(m)$ is defined as $\Psi(m) := \sum_{r\in R(m)} I_{P_r(m)}/\,|R(m)|$.

Persistency. Given a collection of certain types of requests $R(p)$ where each request r contains p, the persistency denoted by $\Phi(m)$ is defined as $\Phi(m) := \sum_{r\in R(p)} I_{P_r(m)}/\,|R(p)|$.

We show that the evaluation results of top 15 common traffic marks in Facebook in Fig. 2. The $\Psi(m)$ and $\Phi(m)$ of some mark is not 1, because it represents more than one quadruple elements or disappear in some requests. We use these results to conduct our parsing priority, i.e. though both URL and Referrer can represent $Action$, we prefer to parse Referrer for $Action$. And apparently, when a mark get full $\Phi(m)$, we release other marks that represent the same quadruple element.

C. Rules for user requests

Since information leakage happens during the interactions with OSN sites, the rules for OSN requests detection must contain every aspects of user request contexts in case of omissions in detections. And as described above, the quadruple is able to represent the contexts of user requests that consist of user behaviors and UGCs. Therefore, we accommodate the OSN access rules to the quadruple elements and offer **triad rules** $\langle S, A, O\rangle$ for user behavior detection, **keywords** for UGC inspection.

TABLE I
THE TOP 10 MOST COMMONLY FOUND TRAFFIC MARKS IN FACEBOOK

Traffic marks	Where to find	Request type	Representation
URL	Request line	All	$Subject, Action$
Referrer	Header	All	$Action$
id	Cookies	All	$Subject, Object$
c_user	Cookies, Entity	All	$Subject$
__user	Cookies	All	$Subject$
object_id	Entity	comment	$Object$
source_id	Entity	share	$Object$
comment_text	Entity	comment	$Content$
message_text	Entity	status	$Content$
content	Entity	note&event	$Content$

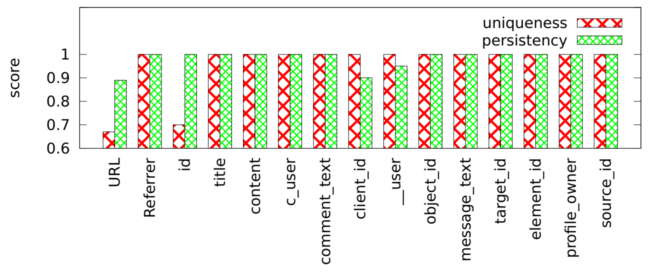


Fig. 2. The uniqueness and persistence score of top 15 common traffic marks in Facebook

IV. THE DESIGN OF SHUTTER

In order to prevent information leakage in organizations, Shutter is designed based on the following four principles and requirements. And the architecture of Shutter is demonstrated in the second part of this section.

A. Design principles and requirements

(i) HTTP and HTTPS traffic should be covered. Because OSN sites like Renren and Weibo use HTTP protocol for user requests, but Facebook and Twitter etc. leverage HTTPS to perform OSN features. (ii)Shutter is required to detect user requests fast while keeping per-flow state small in order to deal with large numbers of sessions and prevent state-holding attacks [8]. (iii)We should handle the field dependencies. Because a user request context includes several key marks and the detection verdict is determined by their combination. (iv)The rules managing methods must be fast. Because rules can be frequently deleted or appended by all the workers in the organization.

B. System architecture

Fig. 3 depicts the architecture of Shutter. Although there are efficient techniques for traffic sniffing [1], we need to deal with non-trivial problems in data stream capture, such as IP addresses collection and HTTPS traffic handling. Our work is focusing on the design of the core engine for the OSN-oriented information protection system, which is efficient for long requests in both HTTP and HTTPS protocols.

For each request, we invoke the element parser to parse the request context for key marks. Provided the required marks, the rule matcher matches two kinds of rules: triads for request behaviors and keywords for UGCs. The triads are held in a behavior matching module and inputted as a string i.e.

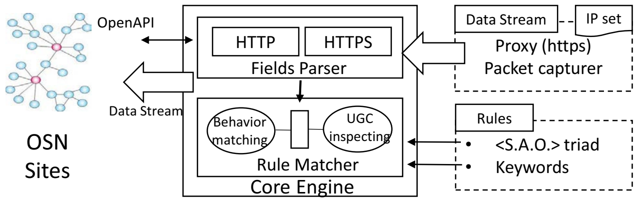


Fig. 3. The architecture of Shutter

jane.v.vo&&addfrient&&zaid.mrz||kate.lz, which means that *jane.v.vo* cannot request the other two people as friends. The names of people are translated to digital IDs through OpenAPI before rule matching. The keywords are sensitive words like company addresses or project names handled by a UGC inspecting module. We merge the detecting results of two rule matching modules by an AND operation as the final verdict. The parser and the matcher are tightly coupled in a pipelined fashion; that is, mark values are delivered to the matcher once parsed and the matcher terminates the parser once the request hits or not.

V. The IMPLEMENTATION of Shutter

In this section, we discuss in detail the three key components of Shutter: packet capturer, element parser and rule matcher.

A. Packet Capturer

In order to perform requests detection, we have to fetch the network packets towards the OSN sites on the gateway. However lots of the OSN sites use HTTPS, which means that the application layer contents are encrypted and the packets can only be captured according to their IP fields. Besides, the IPs translated by DNS are often changed and performing DNS lookup once a packet arrives is unacceptable because of the network delay. Thus, the challenge is finding a way to gather the IP addresses of the OSN sites before packets capture.

An autonomous system (AS) contains the connected IP routing prefixes under the control of one or more network operators such as large organizations. We can leverage the IP routing prefixes of an AS that contains the IP addresses of a certain OSN site for packets capturing. Take Facebook as an example, we first use *dig* (a DNS lookup utility) to get an IP address, then get the AS by running a *whois* (an utility used for querying the registered users or assignees of an Internet resource) against the IP, and at last, run another *whois* against the AS to obtain the list of IP prefixes. Since the IP prefixes can be newly announced or withdraw, we update our IP prefixes at 4-hour intervals. Table II shows partial results of collected 52 Facebook IP prefixes.

Because lots of OSN sites leverage HTTPS to provide secure access against accounts hijacking and credentials sniffing, we must solve the problem of HTTPS encryption for requests parsing. We use a proxy deployed on the gateway to intercept SSL connections through mimicking server certificates. When an intercepted connection is received, we first connect to the OSN server using SSL and receive the server's true certificate, then use the host name inside the certificate to generate a

TABLE II
Partial IP Prefix set of Facebook

Prefix	Aggregation	Description
31.13.74.0/23	+ Announce	
31.13.74.0/24	- Withdrawn	matching aggregate 31.13.74.0/23
31.13.75.0/24	- Withdrawn	matching aggregate 31.13.74.0/23
69.171.224.0/19		
173.252.64.0/18	+ Announce	aggregate of 173.252.64.0/19
173.252.64.0/19	- Withdrawn	matching aggregate 173.252.64.0/18
179.60.193.0/24		

fake one and impersonate the OSN server while still using the already established secure connection to the OSN server. Therefore, we can fetch the unencrypted request data and still communicate with the OSN server. In addition, we took advantage of *iptables* (a packet filtering tool) and gathered IPs to perform coarse-grained filtering by routing the requests traffic of OSN sites to the proxy.

B. Elements Parser

Once receiving the packets of requests, we parse the packets to fetch the traffic marks that describe the user request contexts. Next, we illustrate the characteristics of OSN requests, and show our parsing scheme based on these characteristics.

Characteristics of OSN requests: Four major characteristics make the OSN requests parsing problem unique: (i) Generally, the length of the request is long because of the user generated and tracking contents. Long requests generate TCP segments and reassembling them is time-and-memory-consuming. (ii) OSN sites' requests are of diversification and each request owns huge amounts of traffic marks. But most of the marks are noises for us. (iii) Different marks in the same request can represent the same quadruple element as indicated in Table I. (iv) Parsing URLs or cookies is time intensive, not only because they are long and contain numbers of traffic marks, but also they are in many forms.

HTTP: Two observations derived from the first two characteristics of OSN requests facilitate the parsing process. One is that a request is made up by multiple TCP segments that reach the gateway at different times. The other is that the traffic marks for resolving is a small amount of the whole marks.

Based on the observations above, we perform the following procedure for HTTP protocol. Firstly, we construct a DFA for string searching based on the target traffic marks. Then we search the payload area of the arrived TCP segment for target marks regardless of the segment's order, and deliver the mark that has the highest $\Psi(m)$ to the rule matcher. The only problem is that we may ignore the suffix of some marks if they are separated into two segments. To handle this problem, we store the TCP sequence number and the first sentence of the payload in a hash table indexed by the IP and port value. Therefore, we can compare the sequence number of other segments from the same host and concatenate the mark value. We store the hash table with some fields instead of the whole TCP segments and parse the current segment while waiting for others, thus reducing both the memory and processing time.

HTTPS: This protocol is actually the result of simply layering

TABLE III
AN EXAMPLE OF TRIAD RULES

RuleID	Subject(S)	Action(A)	Object(O)	Action Description
1	12	1	0	comment
2	12	1	2	comment
3	13	*	*	arbitrary
4	31	*	1	arbitrary
5	31	2	*	upload photos
6	33	0	*	update status

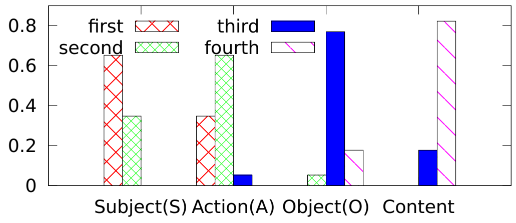


Fig. 4. Quadruple elements appearance order possibility

the HTTP on top of the SSL/TLS protocol. However, the method above is unable to apply to HTTPS traffic, because a TCP segment cannot be deciphered until reassembled. Though we have to reassemble the segments, we can still leverage the properties of the requests to accelerate our processing. The last two characteristics of OSN requests indicate that we can circumvent the parsing of some complicate HTTP fields like URL and cookies and resolve other marks for request contexts first. Then we send the marks with the highest $\Psi(m)$ to the rule matcher. And once some elements are missing at last, we would come back to the URL and cookies fields.

C. Rule Matcher

Once a quadruple element is parsed from a request, the rule matcher matches the element value with the predefined rules. As user request contexts include behaviors and UGCs, we separate the matching process into two steps: behavior matching and UGC inspecting.

1) Behavior matching: Based on tries, We design the layered trie structure for behavior matching. To present the problem, we follow two steps to convert behavior rules—triads into a two-dimensional table. First, we normalize triads by splitting the triad that uses $\|$ (OR) operation into multiple triads. Second, we use OpenAPI to convert the names in normalized triads to the digital presentations. The *Action* elements are also converted to digits that have no common prefix for the sake of pruning in the following operations. A simple example is shown in Table III. Our goal is to match each row in the table with the request marks fast, yet keeping fast rules inserting and deleting operations. The layered trie algorithm and its analysis are shown below.

Triad inserting. Like tries' inserting operations, we convert a triad rule to a string by concatenating each elements and walk the layered trie according to the string appending new nodes for the suffix of the string that is not contained in the layered trie. But two problems need to be solved in our system. One is how to arrange the concatenating order for

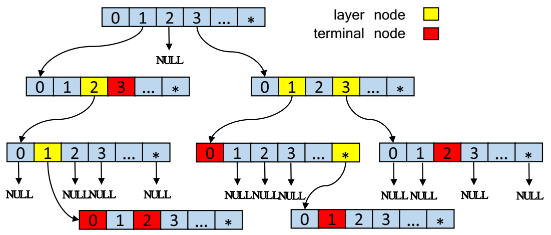


Fig. 5. Layered trie for the triads in Table III

high performance. The other is how to leverage the synthetic string for operating accelerations. For the first problem, we argue that since elements match separately in a request, the concatenation order must conform to the elements appearance order. Therefore, we arrange the concatenating order in the sequence of S, A and O according to the quadruple elements appearance order shown in Fig. 4, which is concluded from the dataset in Section VI. For the second problem, we label each element in the layered trie for pruning. For instance, if $\langle s, *, * \rangle$ or $\langle s, a, * \rangle$ is inputted, we delete the tree branch under s and a respectively since they are the terminal elements for matching. We show the layered trie for Table III in Fig. 5, where some rules are pruned and nodes are labeled as layer nodes and terminal nodes.

Triad searching. Searching operation is similar to inserting operation, but we only compare the characters and move down. In order to reduce memory and speed up the process, we record the layer node once matching a mark, and continue to match the successors of the node if the next mark is being parsed. Searching terminates due to end of triad or lack of string in layered trie. In the former case, if we meet the terminal node, then the triad exists in the tree and we quit the behavior matching process. In the second case, the search terminates without examining the rest characters, since the triad is not present in the tree. After that, the parsing and rule matching processes are complete.

Triad deleting. During the deleting operation, we delete the triad in bottom up manner using recursion. Four conditions must be considered during the deleting operation. (i) Triads may not be there in the layered trie tree. Delete operation should not modify the tree. (ii) If a triad presents as unique triad, meaning no part of triad contains another triad (prefix) and vice versa, then we delete all the nodes. (iii) If a triad is the prefix of another triad in the tree, then we should unmark the terminal label of the triad. (iv) If a triad presented in the tree has at least one other triad as prefix, then we delete nodes from the end of the triad till first terminal node of other triads.

Algorithm Analysis. We believe that tries are the best basic data structures for triads matching. Generally, there are four basic string searching data structures: trie, binary search tree (BST), DFA and hashtable. Unlike other algorithms, tires consume almost identical time for inserting, deleting and searching operations and lose little performance when the indexed items increase [2]. (i) Tries are faster than BSTs, because looking up a key of length m takes at worst $O(m)$

amount of time for tries but $O(mlogn)$ for BSTs where n is the number of keys in the tree. (ii) Although DFAs also take $O(m)$ time for searching, they have to rebuild the whole automaton for inserting or deleting operations. (iii) Hashtables cost almost the same time as tries in terms of searching [2]. But tries are more space-efficient, since nodes in tries are shared when the stored keys contain common prefixes while hashtables need extra memory to avoid collisions.

We analyze the complexity of the layered trie algorithm. The memory space saving for layered trie is threefold. Firstly, in the insertion operation, we search the tree and share the nodes of common prefixes that are natural S and A in triads. Secondly, triads can be aggregated for pruning. Finally, the element values in triads are composed by digits, thus reducing each nodes' memory space. As a result, layered trie requires at most $O(MN)$ memory space, where M is the average string length of triads and N is the number of triads, and the factor can be very small depending on the input. As for runtime, we accelerate the matching process by triads aggregation and pruning, and the average runtime for each operation is $O(M)$.

2) UGC inspection: We inspect the UGC by matching the $Content$ in a request against the predefined keywords (a.k.a. patterns). Since a DFA allows fast transitions between failed pattern matches, DFA based matchers are faster than other matchers in long text matching. And we use the Aho-Corasick (AC) algorithm [17] to do keywords matching, which is one of the most effective DFA based algorithms for multiple pattern matching. AC algorithm combines all the patterns into a DFA and then process the text one character at a time with the state transition for every character. As a result, AC algorithm achieves linear time consumption based on the length of searched UGC and the length of patterns.

VI. Evaluations

In this section, we measure the performance of the Shutter prototype. We introduce the evaluation environment and datasets first. Then we evaluate the two components of the core engine of Shutter respectively in the metrics of throughput, memory and scalability. At last, we evaluate the accuracy and effectiveness of Shutter.

A. Environment and Datasets

The evaluations were performed on a platform with an 8-core CPU (1.90GHz), 32GB memory and Linux 3.8.0 kernel. And we used the symmetric encryption AES [22] and hash algorithms SHA1[2] to simulate the SSL/TLS layer after HTTPS handshake, as well as normal distribution of packets delay to simulate organization environments.

The experiment datasets were captured from some experimenters on the gateway of our lab for 30 days. We used different methods to fetch the OSN requests. One was for the HTTP traffic to OSN sites like Renren and Weibo. We used the *libpcap* (for packet capture) and *libnids* (for TCP reassembly) library to capture the complete request messages from the gateway. The other was for the HTTPS traffic to Facebook, Twitter etc. We leveraged the HTTPS proxy Squid[3] to intercept user request data. All the traffic capture actions have been authorized.

[2]http://tools.ietf.org/html/rfc3174

TABLE IV
The characteristics of the dataset

OSN sites	Renren	Facebook	Weibo	Twitter
App layer size	84.9MB	10.2MB	86.0MB	7.6MB
request number	37.8K	3.6K	48.9K	5.2K
Avg. size	2.3KB	2.9KB	1.8KB	1.5KB
# of picked req.	1870	553	1754	823
# of merged triads	693	264	377	359

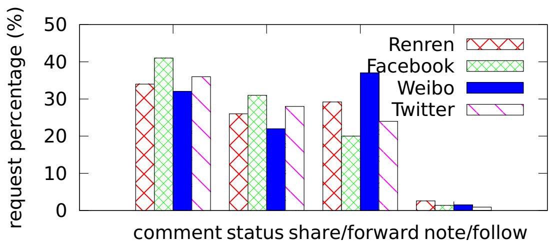


Fig. 6. Percentage of most frequently used HTTP POST request types

The captured data was screened for only HTTP POST requests to avoid noises and the requests were categorized according to different sites and different user behaviors. Table IV shows the size and number of requests in the dataset. The requests of Facebook and Twitter are far less than the other two because people in China are more prone to Renren and Weibo. Fig. 6 shows the most frequently used features of these sites. As for the rule set, since there is no rule dataset suitable for our system at our best knowledge, we generated the rules in the following steps. First, we randomly selected 5000 requests from the dataset. Then we extracted the triad elements and a keyword from each of them. Finally, we merged the triad rules by wildcards. The results are also illustrated in Table IV.

B. Core Engine Performance Analysis

We evaluate the performance of the core engine offline using the simulated environment, because online evaluation takes much more engineering effort and we lack the environment to perform large-scaled evaluations. Therefore, the evaluations reported are not what a real system actually achieves. However, if the offline evaluations are excellent, the bottleneck will no longer be the proposed core engine in the online performance.

1) Parsing performance: We first evaluate the HTTP and HTTPS parsing method respectively, and then compare the results with the HTTP state machine parser used in Nginx[4]. In addition, we add a simulated SSL/TLS layer to the HTTP state machine denoted as SNginx for comparison.

Throughput. Table V shows the throughput and the average memory consumption of a request. Except for the HTTP parser, the other three are all based on the state machine. The result shows that the HTTP parser achieves about twice

[3]www.squid-cache.org
[4]http://nginx.org

TABLE V
PARSING RESULTS

Parser	HTTP	Nginx	HTTPS	SNginx
Throughput(Gb/s)				
Renren	51.8	29.4	18.1	15.3
Facebook	60.3	33.9	19.4	16.3
Weibo	43.5	24.5	14.7	12.5
Twitter	38.1	22.1	14.0	11.6
Avr. Mem. Per Req. (KB)	0.12	2.3	2.5	2.5

TABLE VI
EXECUTION PROFILING RESULTS

Parser	HTTPS			SNginx		
Header	URL	Cookies	Other	URL	Cookies	Other
parsing # (k)	42.3	37.4	95.5	95.5	95.5	95.5
time ratio	22.2%	7.4%	70.4%	31.4%	14.3%	54.3%

as much throughput as the Nginx parser and reaches over 38Gbps, because it parses the current segment while waiting for the next one. For the last two parsers, we apply the encryption and secure hash algorithms to test the parsing performance for HTTPS traffic. The measurements indicate that our HTTPS parser outperforms the SNginx and achieves over 14Gbps. To further understand the performance boost for HTTPS traffic, we profile parsing execution of these two methods. As shown in Table VI, Shutter saves the parsing overhead of URL and cookies in some requests, which are the most time intensive aspects in the parsing process.

Memory Consumption. In Table V, we also illustrate the average memory size required per request. Shutter's HTTP parser costs about 0.12KB and reduces lots of memory comparing with other parsers, because parsers based on state machines have to spend much memory space on TCP reassembly.

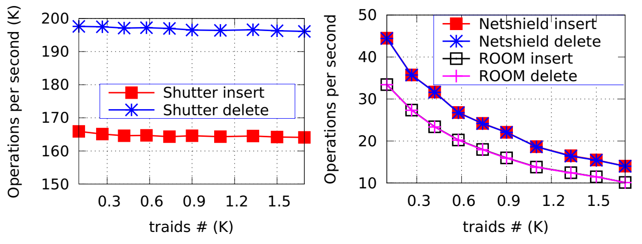
2) *Rule Matcher performance:* We also implement the rule engine prototypes of NetShield and ROOM in the related work as comparisons to Shutter, because these two system contain the most effective rule matching engines at the best of our knowledge. NetShield and ROOM use DFA matchers for each fields in rules and sub-set of fields in rules respectively. We preload both the dataset and the rules into the memory to avoid the bottleneck issues in the hard disk I/O.

Throughput. Table VII shows that Shutter achieves almost the same throughput as ROOM, but higher than Netshield. Take requests in Renren as an example, Shutter obtains 23.3Gbps and ROOM gets 23.0Gbps while Netshield reaches only 17.7Gbps. The reason is that although all of them cost $O(M)$ time for rules matching, where M is the length of the content for matching, Netshield costs extra time for intermediate matching results saving and merging. The throughput of Renren and Facebook are a little higher than Weibo and Twitter respectively, because they have longer request lengths than their counterparts.

We calculate the speedup ratio of our rule matching by comparing it with the sequential matching, which checks each field of the rule sequentially, i.e. comparing the action mark with each action string in the rule set. The speed-up ratio is computed as the sequential matching time over Shutter's rule

TABLE VII
PERFORMANCE OF PARSING AND MATCHING

OSN sites	Renren	Facebook	Weibo	Twitter
Throughput(Gb/s)				
Shutter	23.3	8.6	21.4	7.9
ROOM	23.0	8.7	21.6	7.8
NetShield	17.7	6.5	16.7	6.1
Matching Time(ms)				
layered tire	0.002	0.003	0.002	0.003
keywords	0.017	0.019	0.015	0.014
speed-up ratio	13.1	14.7	13.4	14.3
Matcher state Mem. Per Request. (Bytes)	47	52	45	51



(a) Shutter (b) ROOM and Netshield

Fig. 7. Performance of triads inserting and deleting

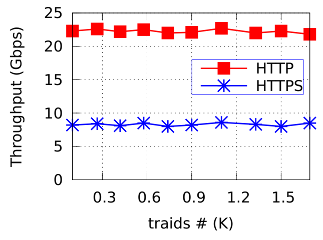
matching time which is added up by layered tire and keywords matching time. As Table VII shows, we speed up the matching over 13 times.

We measure the inserting and deleting operations of triads shown in Fig. 7. Both operations in Shutter are able to perform over 160000 times per second immensely surpassing the other two prototypes. The reason is that Shutter inserts or deletes triads by walking through the triads' characters, while both ROOM and Netshield have to rebuild the whole DFA matchers once a triad is inserted or deleted, which is a time intensive task especially when there are plenty of triads. The gap between inserting and deleting operations in Shutter is caused by the triads aggregation. But since the keywords matching in all the prototypes are based on DFAs, they have almost the same speed for inserting and deleting operations. Therefore, the average time for rules inserting and deleting operations in Shutter is much shorter than those in ROOM and Netshield.

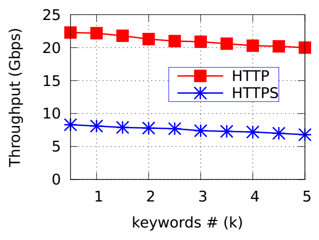
Memory Consumption. There are two types of memory consumption: the matcher states maintained for each request and the data structures shared by all the connections that represent the rule set. Table VII shows the average memory consumption of the first type of memory consumption. The rule matcher state for each request contains a node pointer in the layered trie and the marks waiting for matching. The memory consumption for each OSN site is almost the same and reaches about 50 Bytes for each request. We show the second type of memory

TABLE VIII
SIZE OF MATCHING STRUCTURES ON 1693 TRIADS AND 5000 KEYWORDS

Shutter	96.4KB
Shutter Breakdown	
Layered trie	28.1KB
DFA	68.3KB



(a) Scalability to the # of triads



(b) Scalability to the # of keywords

Fig. 8. Scalability to the # of rules

consumption in Table VIII. Shutter need less than 100KB memory on 1693 triads and 5000 keywords, in which the triads cost less than 30KB. The memory usage of triads is small because memory can be saved by sharing the nodes in layered trie when the new triads have common prefixes with the others.

Scalability. Fig. 8 shows the scalability of Shutter in the metric of throughput when the rules increases, evaluated with the whole collected requests. In Fig. 8(a), the system throughput degrades little as the number of triads increases and achieves about 23Gbps for HTTP and 8Gbps for HTTPS. This is because that the searching performance of the layered trie is attributed to the average length of the triads, which is almost a constant for OSN sites. And the HTTPS throughput is about one third of HTTP throughput for the sake of encryption algorithm and TCP reassembly. And since the AC algorithm is mainly affected by the UGC length rather than the keywords number, the throughput in Fig 8(b) shows a gentle descending curve when adding more keywords. As a result, the system throughput remains almost the same regardless of the number of rules showing prominent performance of scalability.

C. Accuracy and effectiveness evaluation

We conducted the following evaluation to prove accuracy and effectiveness . First, Shutter handled all the requests from the four OSN sites in turn, and count in the illegal requests that violate our rule set. All the predefined illegal requests in our dataset were detected and precisely handled. In this way, we announce that Shutter produces no false positive result. On the other hand, in order to assure the veracity of predefined illegal requests, we asked the experimenters in the lab to set their own rules and used Shutter to check their historical requests to the four sites. All the generated alerts were manually checked, and we found that the corresponding requests actually violated one of the rule set, meaning that there was no false negative alert. Therefore, Shutter can effectively detect user requests and prevent information leakage for OSN sites.

VII. Conclusion

In this paper, we analyze the request messages to OSN sites for fine-grained user request identification. Based on the analysis and statistics, we propose Shutter that takes deep inspections on user request messages to prevent information leakage to OSN sites. The experimental results indicate that Shutter achieves higher throughput than the traditional state machine based protocol parsers. And by comparing with ROOM and NetShield, Shutter's matching throughput is not less than its counterparts, but it shows immense improvement on the rules inserting and deleting speed, as well as prominent performance of scalability. In addition, Shutter also shows high accuracy rate in detecting the requests that contain sensitive information. Based on the results above, Shutter is proved as an effective and efficient tool for the OSN requests detection.

Acknowledgement

The work in this paper is supported by China MOST project (No. 2012BAH46B04), NSFC Programs (No. 61202424, No.4122012), SKLSDE-2014ZX-04, and the Fundamental Research Funds for the Central Universities.

References

[1] DAG card. http://www.endace.com/dag-8.1sx.html
[2] Trie. http://en.wikipedia.org/wiki/Trie
[3] Trend Micro. http://www.trendmicro.com/cloud-content/us/pdfs/business/tlp-likes-links-and-lessons-learned.pdf
[4] T.T.T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 2008.
[5] Moore, Andrew W., and Konstantina Papagiannaki. "Toward the accurate identification of network applications." *Passive and Active Network Measurement*. Springer Berlin Heidelberg, 2005. 41-54.
[6] Fung, Benjamin, et al. Privacy-preserving data publishing: A survey of recent developments.*ACM Computing Surveys (CSUR)* 42.4 (2010): 14.
[7] Roesch, Martin. Snort: Lightweight Intrusion Detection for Networks. *LISA*. Vol. 99. 1999.
[8] Zhichun Li, et al. Netshield: massive semantics-based vulnerability signature matching for high-speed networks. *ACM SIGCOMM Computer Communication Review* 41.4 (2011): 279-290.
[9] M.Becchi and P.Crowley. A hybrid finite automaton for practical deep packet inspection. In *Proc. of ACM CoNEXT*, 2007.
[10] Lipford, Heather Richter, Andrew Besmer, and Jason Watson. "Understanding Privacy Settings in Facebook with an Audience View." *UPSEC* 8 (2008): 1-8.
[11] Madejski, Michelle, Maritza Lupe Johnson, and Steven Michael Bellovin. "The failure of online social network privacy settings." (2011).
[12] Ning Xia, et al. "Mosaic: quantifying privacy leakage in mobile networks." In *Proc. of ACM SIGCOMM*, 2013.
[13] V.Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31, 1999.
[14] Smith R, Estan C, Jha S, et al. Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. In *Proc. of ACM SIGCOMM*, 2008.
[15] Daqing Zhang, Haoyi Xiong, Ching-Hsien Hsu, Athanasios V. Vasilakos: BASA: building mobile Ad-Hoc social networks on top of android. *IEEE Network* 28(1): 4-9 (2014).
[16] Hao Li, et al. ROOM: Rule Organized Optimal Matching for fine-grained traffic identification. In *Proc. of IEEE INFOCOM* 2013.
[17] Aho, A. V. and Corasick, M. J. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6), 333-340.
[18] Baden R, Bender A, Spring N, et al. Persona: an online social network with user-defined privacy. In *Proc. of ACM SIGCOMM*, 2009.
[19] Gross, Ralph, and Alessandro Acquisti. "Information revelation and privacy in online social networks." *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM, 2005.
[20] Nilizadeh, Shirin, et al. "Cachet: a decentralized architecture for privacy preserving social networking with caching." In *Proc. of ACM CoNEXT*, 2012.
[21] Zheleva, Elena, and Lise Getoor. "To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles." In *Proc. of ACM WWW*, 2009.
[22] Aes, N. I. S. T. "Advanced encryption standard." *Federal Information Processing Standard*, FIPS-197 12 (2001).