

Discover Abnormal Behaviors Using HTTP Header Fields Measurement

Quan Bai, Gang Xiong^(✉), Yong Zhao, and Zhenzhen Li

Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, China
{baiquan, xionggang, zhaoyong, lizhenzhen}@iie.ac.cn

Abstract. In recent years, in order to be secure, more and more Intrusion Detection Systems (IDS) and firewalls have been used to detect and block malicious applications or even unknown protocols. As a result, some malicious applications begin to shape themselves as common application protocols to get rid of detection. Being an important protocol for many Internet services, HTTP is responsible more than half of the total traffic volume. As a result, many applications choose HTTP protocol as their shaping object, leading to many abnormal behaviors. In the paper, we study the problem of discovering these abnormal behaviors in HTTP protocol. A method based on HTTP header fields' measurement is proposed. We measure HTTP header fields' information from HTTP traffic from normal application such as IE-8, find some characteristics and we use them to find abnormal behaviors of shaping HTTP protocol.

Keywords: HTTP header fields · Measurement · Abnormal behaviors · Protocol format

1 Introduction

With the development of the Internet, more and more Intrusion Detection Systems (IDS) and firewalls have been used to detect and block malicious applications or even unknown protocols. As a result, some malicious applications begin to shape themselves as normal application protocols to get rid of detection.

However, a malicious application can do protocol confusion, but its web traffic may have abnormal protocol format. Houmansadr et al. [1] shows that, “unobservability by imitation” is a fundamentally flawed approach. The web traffic produced by malicious application has abnormal protocol format. So we can find those abnormal behaviors based on the discovery of abnormal protocol format.

Being an important protocol for many Internet services, HTTP is responsible more than half of the total traffic volume. Our measurement on HTTP traffic shows that, there are nearly 12000 types of HTTP header field in HTTP request traffic and nearly 32000 types in HTTP response traffic. The diversity of HTTP header fields of HTTP message brings a difficult problem for web applications based intrusion detection [2]. As a result, many applications choose HTTP protocol as their shaping object, which leads to many abnormal behaviors.

Take FTE (Format-Transforming Encryption) proxy as an example. FTE works by transforming encrypted data in words of a specific language based on a regular expression. It is now integrated in Tor and is mainly transformed into HTTP traffic. The principle of Format-Transforming Encryption is shown in Fig. 1.

The FTE client transforms original traffic into HTTP-like traffic using a regular expression shown in Fig. 2. The shaped data is detected as HTTP traffic by the firewall and get through the firewall. Then the proxy will reverse the data into original traffic. In this way, FTE sends its data through the firewall by shaping as HTTP [3].

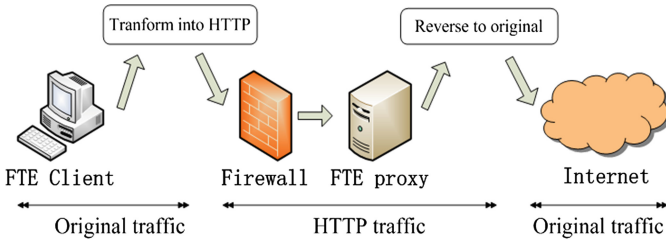


Fig. 1. FTE proxy principle

```
^HTTP/1\\.1\\ 200 OK\\r\\nContent-Type:\\ ([a-zA-Z0-9]+)\\r\\n\\r\\n\\C*$'
^GET\\ \\ \\ ([a-zA-Z0-9\\.\\ \\ ]*) HTTP/1\\.1\\r\\n\\r\\n$'
```

Fig. 2. Example of regular expression used by FTE proxy

These shaping applications hazard the Internet security seriously and they create many abnormal behaviors in the web. So it is necessary to find a method to detect abnormal behavior of web traffic. In the paper, we study the problem of discovering these abnormal behaviors in HTTP protocol. A method based on HTTP header fields’ measurement is proposed. We measure HTTP header fields’ information from HTTP traffic from normal application, find some characteristics and we use them to find abnormal behaviors of shaped HTTP protocol.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 shows our approach to find abnormal behaviors. A method base on measurement is proposed. In Sect. 4 we take HTTP protocol as an example to show that method based on measurement and statistics is useful to find behaviors of network evasion and protocol obfuscation. Finally, Sect. 5 concludes our work.

2 Related Work

2.1 Discovery of Abnormal Behaviors

There are many researches on discovery of abnormal behaviors, and many feasible methods are proposed. These methods have something in common, that is they are all

done in the application layer. These methods can be used to find abnormal behaviors such as network evasion and protocol obfuscation.

Roelker [4] reviews HTTP IDS (Intrusion Detection System) evasions approaches such as invalid protocol parsing and invalid protocol field decoding. And the work divides the detection methods into two classes: protocol analysis and pattern matching.

Hernacki et al. [5] overviews the existing network evasion methods based on abnormal protocol format. The work summarizes them into five classes: Tunneling, Flooding, Desynchronization, Encoding variations and Segmentation & reordering. They also show that network attack based on network evasion technology may affect the network security seriously.

Hjelmvik et al. [6] proposes a traffic classification method based on statistical analysis. This work divides traffic classification methods into four classes: payload examination, social host behavior, statistical flow fingerprints and obfuscated traffic. Besides this paper give some examples of protocol obfuscation behaviors. For example, Skype obfuscates VoIP protocol, BitTorrent has its Message Stream Encryption (MSE) protocol, which is also called Protocol Header Encryption, (PHE) and eDonkey obfuscates UDP protocol or TCP protocol. Using these protocol obfuscation behaviors, these P2P (Peer to Peer) applications get through the Intrusion Detection Systems and realized network evasion behaviors. At last, this paper proposes that one can improve the obfuscation behaviors' obfuscation performance by randomizing data stream structure, randomizing packet header, obfuscating the direction of packet and so on.

Mahoney et al. [7] proposes a method based on payload keyword to detect abnormal web behaviors. This work detects the application layer attack using specific keywords of the data packet payload and they also construct union attribute pairs (pairs of keyword and destination port) to find attack behaviors.

Wang and Stolfo [8] proposes a method based on statistical distributions of character of payload. They discover and detect application layer network attack using statistical distributions of character of payload.

Hjelmvik and John [6] also proposes a frame for protocol identification called SPID, which is short for Statistical Protocol Identification. The frame analyzes the payload by attribute classification and successfully identifies application protocol of obfuscated traffic based on the traffic characteristics of the session.

Shen [9] proposes a algorithm to detect web anomaly behaviors using the characteristic of length of HTTP request, characters distribution, structure of attribute fields, enumeration of attribute fields and so on.

2.2 Automatic Discovery of Protocol Format

There are also researches on distinguishing unwanted crawlers and valid ones.

Reverse engineering is the traditional method of protocol format analysis. Reverse engineering, also called back engineering, is the process of extracting knowledge or design information from anything man-made and re-producing it or reproducing anything based on the extracted information.

Traditional reverse engineering is done by artificial manual analysis, which is a Boring process and is easy to make errors. As a result, Analysis of automation

discovery is becoming a new Research point. Luo et al. [10] summarizes the methods of analysis of Unknown protocol into three classes.

Program Analysis. These methods are based on the protocol information in the application by using dynamic stain spread and binary tracking to track process of the protocol. Caballero et al. [11] proposes the system Polyglot to mine the packet format is based on dynamic stain technology. If the length of a field is controlled by certain parameters, then this field can be recognized as fixed-length fields. Unfixed-length fields are usually recognized by the length of the field and the separators. This system can not only extract the information format, but also discover protocol keywords by tracking stain data and comparing them with constant character strings.

Research on Protocol Similarity. These methods are all doing clustering with network data packets and using sequence pairs alignment, similar matrix, evolutionary tree and other algorithms to extract the protocol [12].

Data Mining. These methods extract protocol format by searching the common characteristics of the protocols.

The first class of method is Suitable for both text protocol and binary protocol. While the last two classes can get good results when dealing with text protocol but their accuracy is not ideal with dealing with binary protocol.

Above all we can find that there will be further works on discovering the protocol format automatically. At the same time, accuracy of the analysis results will become much higher. Realizing the automatic analysis with much higher accuracy will become hot research point on protocol structure. We can also see that, the existed works are mainly to get the abnormal format of protocol through snifferring certain applications. And what we do is to discover abnormal protocol format in the real network environment, and further more to find abnormal behaviors based on traffic measurement.

3 Methodology

The purpose of protocol confusion is to implement the features of “unobservability”. Intuitively, unobservability means that a censor can neither recognize the traffic generated by the circumvention system, nor identify the endpoints engaged in circumvention. However, to implement unobservability, the malicious applications need to mimic the protocol in its entirety (include Correct, IntraDepend, InterDepend), mimic reaction to errors and network conditions (include Errors, Network), mimicking typical traffic (include Content, Patterns, geographic location), and mimic implementation-specific artifacts (include Soft, operating system). Houmansadr et al. [1] shows that, it is impossible to satisfy all these requests.

So we can see that although a malicious application may shape itself as a common protocol, it cannot get all the characteristics of the common protocol, especially the general characteristics of message structure in statistics.

Once again we take FTE as an example: although it can shape itself as HTTP protocol using a regular expression, it cannot get all the characteristics of fields of normal HTTP request message. We find a regular expression in its configuration file of Tor Browser, an application which FTE is integrated in. Its format is like below:

```
“^GET\\ \\ / ([a-zA-Z0-9\\.\\ /] *) HTTP/1\\.1\\r\\n\\r\\n$”
```

Figure 3 shows the HTTP request message created by Tor Browser using this regular expression. We can see that the shaped HTTP request message only has key words like “GET” and “HTTP” and even no HTTP Header Fields.

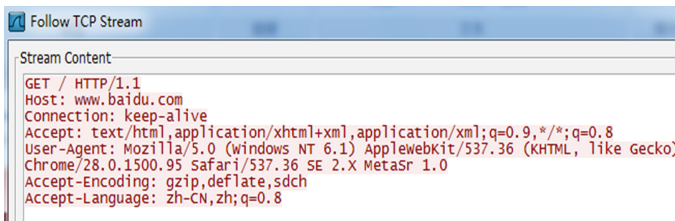


```
GET //
Hffe9ctIaZK/.w3VC.0M3yNms1Zn9QXks6TTmikswoqioh6Ext4qHQNVTV35Zs3rR0t1zet8Zz
vcOBSLjJrZZSKipCccK5IpoBcTeBQPu.qR5DOCGCN9TALQPfNLPamp.mBgPNfTwNq54T5fjKKV
8zEpye2cPNkkG2V/HZy3K4LwNns.uwkaPY44cJqjAbTzBwJhrWpt8x4F15kcPRH HTTP/1.1

Um...0.*.=.id.....43..t.q.@0.....uL.~.....0..a.x.r41...[pOY.m
<...1.i.N.....C...A...w.....<.e.|...y.R...E.m.V...|.....
{.B.$m..n...10.D.....W....]_7..)[...&!...!.k.$..e...;fy.).....G.....
I.Z.....L.cd.b...w.4t..8.....T...!=4...;.....x.....K.4`..)...
H.?dd^,.w2.w;..L2.....B.f..&
```

Fig. 3. HTTP request message created by FTE proxy

Figure 4 shows the HTTP request message created by normal web Browser (IE Browser). We can see that the normal HTTP request message not only has key words like “GET” and “HTTP” but also and has HTTP Header Fields like “Host”, “Connection”, “Accept”, “User-Agent” and so on, and they are placed in a specific order.



```
Follow TCP Stream
Stream Content
GET / HTTP/1.1
Host: www.baidu.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/28.0.1500.95 Safari/537.36 SE 2.X MetaSr 1.0
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
```

Fig. 4. HTTP request message created by normal web Browser

So we can see, for a specific HTTP application like IE Browser, its HTTP header fields is usually constant, some other applications may shape their data or declare themselves by User-Agent as traffic of IE Browser, but their HTTP header fields information will still be abnormal in fields’ details and orders. We can firstly measure the traffic of a certain normal application protocol in real network; and secondly we do statistics on HTTP header fields to summarize general characteristics; finally we use these characteristics to do match and those shaped traffic may be classified as abnormal ones.

4 Experiments and Results

4.1 Environment

We take *IE-8 Browser* as an example. We know that HTTP request message from normal *IE-8 Browser* has string like “*Mozilla/4.0 (compatible; MSIE 8.0;...*” [13]. According to our idea, we began with matching this User-Agent to do statistics with massive of *IE-8 Browser* traffic in the real network CSTNET, one of Chinese ISPs. We focus on how much HTTP header fields the traffic will have on average and the general order of them. We got some statistical characteristics of HTTP header fields to use them to discover abnormal behaviors of web flow. We did measurement for a full week.

4.2 The Basic Statistical Analysis of HTTP Header Field

We measured the count of HTTP header fields the traffic will have on average and the frequency of them. To get a more accurate result, we did this in two directions: C2S (which is short of Client to Server) and (S2C which is short for Server to Client).

There are 11248 kinds of HTTP header field keywords in the 208360203 measured HTTP flows in the direction of C2S. There are 9.80534 header fields on average, as is shown in Table 1.

Table 1. The basic statistical analysis in the direction of C2S

Ranking	HTTP request field	Count	Frequency
1	Host	137121850	65.81 %
2	Connection	125776314	60.99 %
3	User-Agent	121032201	58.57 %
4	Accept-Encoding	102451704	50.25 %
5	Accept	95398172	46.70 %
.....			

There are 31330 kinds of HTTP header field keywords in the 237540382 measured HTTP flows in the direction of S2C. There are 6.64656 header fields on average, as is shown in Table 2.

Table 2. The basic statistical analysis in the direction of S2C

Ranking	HTTP response field	Count	Frequency
1	Date	232314493	97.80 %
2	Server	231734078	97.56 %
3	Connection	214881782	90.46 %
4	Content-Type	194919502	82.06 %
5	Content-Length	175658286	73.95 %
.....			
22	Age	6546036	3.47 %

The results of the measurement can be used as a characteristic to discover Abnormal HTTP traffic, which we will use in Section C. And with these results, we can future mining the order of the Top 20 field keywords in the two directions.

4.3 The Mining Analysis of HTTP Header Field Order

Scheme I, Mining the header keyword based on position: Firstly, We the count of each HTTP header field of each position and sort them. And then we deal with each sorted position synthetically and get the most frequent HTTP header field of each position. For example, firstly we divide the data set into 20 files based on the position of each HTTP header field. As is shown in Fig. 5, “File 1” stands for HTTP header field keyword appeared in the first position in the whole data set. At last we summarize “File 1” to “File 20” to get the final file, which is the most common field keyword of the 20 positions.

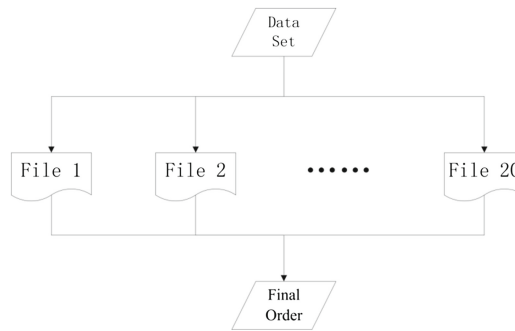


Fig. 5. Mining the header keyword based on position

Table 3 shows the result of order based on Scheme I.

The advantage of Scheme I is that its model is simple to understand, and the disadvantage is that it is sensitive to difference. If adding a new field at the beginning, all the fields' value will change.

Table 3. The result of order based on Scheme I

C2S	S2C
Host	Data
Connection	Server
Accept-Encoding	Content-Length
User-Agent	Connection
Content-Type	X-Rowered-By
Accept	Cache-Control
Referer	Accept-Ranges
From	Set-Cookie
.....

Scheme II, Mining the header keyword based on the weighted order: To overcome the sensibility to difference in Scheme I, We mining the data set again in another dimension. Scheme II is based on order rather than position. We scored each HTTP header field keywords based on their coming order. Keyword with higher score means it will coming earlier in the position.

We traversed each HTTP protocol request or response packet, the keyword coming first weight higher (N), and the keyword coming later weight lower, the weight value diminishes according to the order. The weight formula is shown below:

$$score[F] = \sum_j (N - order[F][j])$$

Score[F] stands for order score of HTTP header F, and order[F][j] means the order of F in the jth HTTP flow of the data set. N is set 20 because we only checked the top 20 header fields. Table 4 shows the result of order based on **Scheme II**.

Table 4. The result of order based on Scheme II

C2S	S2C
Host	Server
Connection	Date
Accept-Encoding	Content-Type
Accept	x-powered-by
Content-type	Content-Length
Accept-Language	Connection
User-Agent	Cache-Control
X-Requested-With	Pragma
.....

The advantage of Scheme II is that it is based on order rather than position, so that it can recover the sensibility in Scheme I. And the disadvantage is that it will ignore special header field. As it is just addition simply, the field appears more will get more changes to addition which will lead to a higher score in position order.

Scheme III, Mining the header keyword based on the average weighted order: To recover the problem of field appears more will score higher in Scheme II, we modified the formula on the basis of Scheme II: we added a variable count[F] to record the coming count of field keyword, and averaged the value at last, The weight formula is shown below:

$$score[F] = \frac{\sum_j (N - order[F][j])}{count[F]}$$

Variable count[F] stands for count of HTTP request messages that contain HTTP header F. Table 5 shows the result of order based on Scheme III.

Table 5. The result of order based on Scheme III

C2S	S2C
Host	Date
Connection	Server
Accept-Encoding	Connection
Accept	Content-Length
User-Agent	Content-Type
Accept-Language	X-Powered-By
Content-Type	Cache-Control
X-Requested-With	Last-Modified
.....

The advantage of Scheme III is that it is based on order rather than position, and it can recover the disadvantage in Scheme II. But the disadvantage is that this scheme is strongly statistical. However, in the real network, it is reasonable for HTTP protocol header field keywords to have a variety of orders. This is similar to the overfitting in machine learning.

We compared the different results of 3 schemes and took C2S direction as an example, the result is shown in Fig. 6.

C2S Scheme I	C2S Scheme II	C2S Scheme III
Host	Host	Host
Connection	Connection	Connection
Accept-Encoding	Accept-Encoding	Accept-Encoding
User-Agent	Accept	Accept
Content-Type	Content-type	User-Agent
Accept	Accept-Language	Accept-Language
Referer	User-Agent	Content-Type
From	X-Requested-With	X-Requested-With
Accept-Language	If-Modified-Since	If-Modified-Since
.....

Fig. 6. Comparison of different results of these three schemes

Firstly, Scheme II and Scheme III are in the dimension of order, Scheme III is an improvement of Scheme II, while Scheme I is in the dimension of position. As a result, the results of Scheme II and III are relatively more similar than Scheme I.

Secondly, although the results of these three schemes are different in detail, they still have some similarities. For example, the first three header field keywords of three schemes are the same, they are “Host”, “Connection” and “Accept-Encoding”. As for the 4th to 7th keywords, although the orders of them from three schemes have some differences, they are just simply different in position; they are still the same in the view of a set.

So we can see, in the view of set, the header field keywords order are the same in block from three schemes. We can use keywords block instead of keywords position: the 1st block includes keywords like “Host”, “Connection” and “Accept-Encoding”; the 2nd block includes “Accept”, “Content-Type”, “Accept-Language”, “User-Agent”...

and so on. In the view of keywords block, we can also solve the problem of overfitting in Scheme III.

4.4 Recognize Abnormal HTTP Flow

We analyzed the data we got and found some statistical characteristics of HTTP header fields. Table 1 shows the main header fields of IE-8 Browser traffic and their percentage of appearance.

Table 6. Statistical characteristics of HTTP header fields

HTTP header name C2S	HTTP header name S2C
Host	Date
Connection	Server
Accept-Encoding	Connection
Accept	Content-Length
User-Agent	Content-Type
Accept-Language	X-Powered-By
Content-Type	Cache-Control
X-Requested-With	Last-Modified
If-Modified-Since	Accept-Ranges
From	Pragma

Firstly, we used the count of HTTP header fields the traffic have on average and the frequency of them we got in Tables 1 and 2. As for the result may do not cover all the cases, for example, some HTTP request message contain header field of cookie while others do not. So we only checked the common fields listed in Table 6.

And then we sorted the score[F] in **Scheme III** and got the general order of header fields of IE-8 Browser traffic, the result of C2S direction is shown in Fig. 7.

We reviewed the data by matching the header fields and the general order of them by formula below. N stands for the set of header fields listed in Table 1, and F stands for the set of fields of HTTP request message being tested. Card (N) is the number of elements of set N. Inverse (F) means the inverse number compared with the common order of F. When the value_x(F) is less than the threshold value we set, this HTTP request message will be recognized as an abnormal one.

$$\text{value_x}(F) = a \frac{\text{card}(F \cap N)}{\text{card}(N)} - b * \text{Inverse}(F)$$

We found that most of the request messages passed the test, while some are recognized as abnormal ones. We checked those data and found them indeed different from the common ones. They do not even have GET fields, which mean they do not for the purpose of requesting.

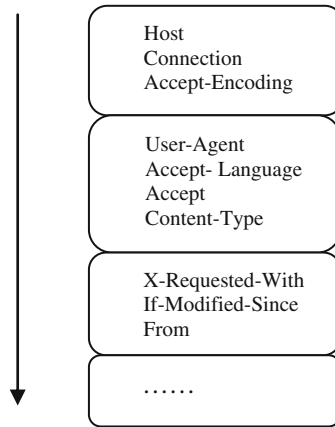


Fig. 7. General order of HTTP header fields of IE-8

There are also abnormal behaviors of IE-8 traffic with other request method. An example is shown in Fig. 8.

```
CONNECT api.foxitcloud.com:443 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT
6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0;
.NET4.0C; .NET4.0E)
Host: api.foxitcloud.com:443
Content-Length: 0
Proxy-Connection: Keep-Alive
Pragma: no-cache
```

Fig. 8. An abnormal HTTP flow of IE-8

This packet is started with request method **CONNECT** and it is not pass the formula test. The method name **CONNECT** is used for a proxy that can dynamically switch to being a tunnel, and we can see that this packet is used for Foxit cloud to switch to a SSL tunnel.

5 Conclusions

In the paper, we study the problem of discovering these abnormal behaviors in HTTP protocol. A method based on HTTP header fields' measurement is proposed. We measure header fields' information from HTTP traffic from normal application, find some characteristics in statistics such as count of header field in each packet on average, the frequency of each field and the order block of them. Using them we found some abnormal behaviors of shaped HTTP protocol. The result shows that it is possible to discover abnormal web behaviors using HTTP header fields' measurement.

Acknowledgements. This work is supported by the National Science and Technology Support Program (No. 2012BAH46B02); the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA06030200).

References

1. Houmansadr, A., Brubaker, C., Shmatikov, V.: The parrot is dead: observing unobservable network communications. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 65–79. IEEE (2013)
2. Hjelmvik, E., John, W.: Breaking and improving protocol obfuscation. Technical report 123751, Chalmers University of Technology (2010)
3. Dyer, K.P., Coull, S.E., Ristenpart, T., et al.: Format-transforming encryption: more than meets the DPI. The IACR Cryptology, p. 494. ePrint Archive (2012)
4. Roelker, D.J.: HTTP IDS evasions revisited. Sourcefire Inc. (2003)
5. Hernacki, B., Bennett, J., Hoagland, J.: An overview of network evasion methods. Inf. Secur. Tech. Report **10**(3), 140–149 (2005)
6. Hjelmvik, E., John, W.: Breaking and improving protocol obfuscation. Technical report 123751, Chalmers University of Technology (2010)
7. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 376–385. ACM (2002)
8. Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)
9. Shen, X.: A implementation of intrusion detection system based on web anomaly detection. Shanghai Jiaotong University (2010)
10. Luo, C., Zhang, Y., Wang, Q., et al.: Automatic network protocol analysis and vulnerability discovery based on symbolic expression. J. Grad. Univ. Chin. Acad. Sci. **30**(2), 278–284 (2013)
11. Caballero, J., Yin, H., Liang, Z., et al.: Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 317–329. ACM (2007)
12. Li, W.M., Zhang, A.F., Liu, J.C., et al.: An automatic network protocol fuzz testing and vulnerability discovering method. Jisuanji Xuebao (Chinese Journal of Computers) **34**(2), 242–255 (2011)
13. Microsoft Developer Network (MSDN): Understanding user-agent strings. [https://msdn.microsoft.com/zh-cn/library/ms537503\(en-us\).aspx](https://msdn.microsoft.com/zh-cn/library/ms537503(en-us).aspx)