UDC621.39

MULTILEVEL INTELLECTUAL APPROACH TO HTTP-REQUESTS LEGITIMACY VALIDATION

Volodymyr M. Kononenko[1], Serhii O. Kravchuk[1],
Yurii V. Ivlev[1], Liubov A. Kononenko[1]

[1]National Technical University of Ukraine "Kyiv Polytechnic Institute", Kyiv, Ukraine

In the paper a multilevel intellectual approach to HTTP-requests legitimacy validation is proposed. The approach is devised for HTTP-flood DDoS-attacks detection and prevention in telecommunication networks with a web-server as the target attack object. The analysis of HTTP-requests attributes and their signatures is provided. On the basis of the analysis the attributes are separated into several levels that allow us to design a flow analyzer in a form of the multilevel block. Due to a multilevel structure of the flow analyzer a minimization of resources, spent for a request handling, is achieved.

Keywords: DDoS-attack, HTTP-flood, request, validation, legitimacy, signature method.

Introduction

Network information resources defense against external attacks of malefactors is one of the vital problems nowadays. The most widespread attacks at present are DoS-attacks (Denial of Service). The main goal of such attacks is to make a network resource inaccessible to its intended users. Generally an attack is launched on a big amount of hosts, i.e. an attack is distributed, or DDoS (Distributed Denial-of-Service). The process is controlled by malicious software, which is installed using client's workstation software or protocols vulnerabilities (the most common software is a browser, its plug-ins, Java and Flash) [1].

Defense against Denial-of-Service attacks involves blocking a malicious traffic without obstructing a legitimate users' traffic (clients' requests and server replies). By the term "a legitimate request" we mean a request, which is not indented to cause a denial of system's service and is not sent by malefactor. So, "a legitimate client" is a client, who sends legitimate requests. In the case of non-distributed attack (DoS) the problem of blocking malicious traffic is not an intricate one. It is enough to detect an IP-address, which causes the greatest activity, i.e. uses a wide bandwidth, has a high requests frequency, a big amount of POST requests etc.

In the case of a distributed attack (DDoS) obvious signs, which can be used for malefactor detection are missed.

A significant amount of works is devoted to the problem of resources defense against DDoS-attacks. Among them are the works [8], [9], [10].

The **goal** of this paper is to devise the approach for HTTP-flood DDoS-attacks detection and prevention in telecommunication networks with a web-server as the target attack object in which a used amount of computing resources varies depending on the difficulty of malicious requests detection.

Problem definition

Let's consider the defense system against DDoS-attacks based on a flow analyzer (fig.1). The mixed flow of legitimate users' requests and malicious requests comes to the system input. Ideally, a flow analyzer must make a decision of requests legitimacy and accomplish a distinct separation of legitimate and malicious requests. In the sequel, legitimate requests must be handled with a web-server and then a response must be sent. In the case of unsuccessful legitimacy test several variants are possible: error code is sent to a client; close connection without sending any response is exploited; malicious IP address is blocked with a firewall and the connection closes.

The goal of a defense system is to determine an affiliation of incoming request to one of the sets – "legitimate requests" or "illegitimate requests" based on defined set of HTTP-request attributes. It is necessary to note, that an important value is an amount of resources, spent by flow analyzer for the handling of a request. This value should be minimized.

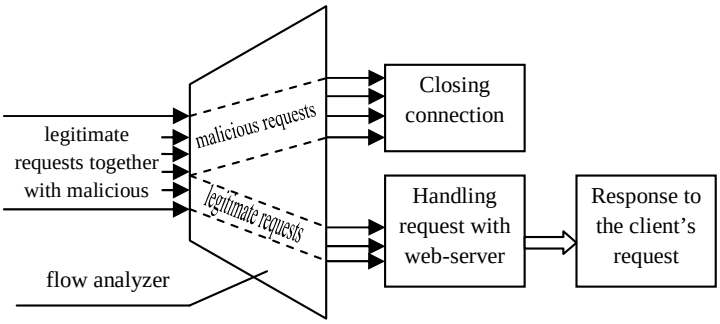

Fig. 1 – Separation of legitimate requests and malicious requests

Under these conditions the problem is reduced to the following one:

Given:

1) r – incoming HTTP-request
2) $x = \{x_i | i = \overline{1,n}\}$ – defined set of HTTP-request attributes, where n – the amount of determined attributes for the i incoming request;
3) $R = \{R_L, R_{NL}\}$ – set of request classes: R_L – legitimate, R_{NL} – illegitimate;
4) P_i – computing resources (power), spent by a flow analyzer for one attribute check.

Find:

Algorithm $a: x \to R: P \to min$

Analysis of a set of HTTP-request attributes. Let's consider an example of a request in a form of the following listing (listing 1) to define a set of attributes, which are typical for an average legitimate request:

Listing 1 – Example of HTTP-request

```
GET /page.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu;
Linux x86_64; rv:18.0) Gecko/20100101
Firefox/18.0
Accept:
text/html,application/xhtml+xml,appli
cation/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.example.com
Cookie: c1=cookie1_value;
c2=cookie2_value
Connection: keep-alive
```

On the analysis of such a request it is possible to determine the set of attributes, summarized in Table 1 [2]:

Table 1.
Request attributes and their signatures

x_i	Request attribute	Signature
x_1	Method to be performed on the resource identified by the Request-URI	Specified as the first parameter in the first row
x_2	Request string	Specified as the second parameter in the first row
x_3	Protocol of the requests/protocol version	Specified as the third parameter in the first row
x_4	Version of the client software, operating system of the host, where client's software is running	Header "User-Agent"
x_5	Specification of certain media types, which are acceptable for the response	Header "Accept"
x_6	Content-codings acceptable in a response	Header "Accept-Encoding"
x_7	The set of natural languages that are preferred as a response to the request	Header "Accept-Language"
x_8	Specification of options that are desired for a particular connection	Header "Connection"
x_9	Headers order	A full list of request headers is analyzed before receiving a double end-of-line symbol (CRLF)
x_{10}	Client IP-address	Specified on network layer of TCP/IP model and is available on application layer
x_{11}	Request frequency per IP address	Specified on web-server level
x_{12}	Presence of specific headers	From, Proxy-Connection, Via etc.

Based on these attributes a flow analyzer can make a decision of request legitimacy. A process of particular attributes check differs according to the level of difficulty. That's why the amount of computing resources may vary depending on requests.

Let a flow analyzer check all the defined attributes during request handling. Then an amount of computing resources (power) P spent by a flow analyzer during a check of one attribute, is equal to:

$$P = \sum_{i=1}^{n} P_i$$

As it can be seen, all the attributes are checked, resources are not minimized. The resultant value of the expression under summation will be reduced under the two following conditions:

1) P_i values are reduced;
2) some of P_i terms are excluded from the expression.

Values of P_i are considered to be constant values.

Besides, obtains an ambiguity factor. For instance, missing of User-Agent header or blank string as a User-Agent value unambiguously signifies an illegitimacy of the request [2].

Based on cogitation about computing resources minimization and attributes' ambiguity, we propose to make a flow analyzer as a modular multilevel non-monolithic block (fig. 2).

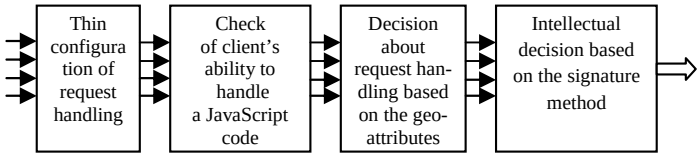

Fig. 2 – Modular multilevel structure of a flow analyzer

The flow analyzer modules form a chain, and requests flow comes to input of this chain. Actually the input of the first module is the input of a flow analyzer. At output of each module two scenarios are possible:

1. to block requests and interrupt the chain, if a decision of request's illegitimacy is made;
2. to transfer requests flow from output to input of the next module for further analysis.

The output of the last module is the output of a flow analyzer.

Each module corresponds to a certain subset of attributes $X_i \subset X$. In such a way the problem of minimization is solved. The remaining problem is divided into four subtasks (by the number of flow analyzer modules):

To find an algorithm

$$a_i : x \in X_i \to R, i = \overline{1,4}$$

The first three modules of a chain handle the check of necessary, but not sufficient conditions for making decision about request legitimacy, while conditions for making decision of illegitimacy are sufficient.

From our point of view, modular approach has the following advantages:

- independence of the logic of each module from other modules in the chain;
- ability to disable particular modules;
- ability to extend a flow analyzer by adding new independent modules to the chain.

Thin configuration of requests handling. Modern web-servers allow intellectual agents to take a set of preventing steps against DDoS-attacks. During the research a set of request handling principles, which can be observed with a native web-server configuration, is formed:

1. Not to handle client requests (close connection or return error code):
   - in which header User-Agent is missing;
   - in which User-Agent header value is an empty string;
   - in which the following substrings: "bot", "index", "spider", "crawl", "wget", "slurp", "libwww", "curl", "wget", "lynx", "urllib", "ruby", "php", "perl", "python", "java", "http://" are found while checking User-Agent header value with a regular expression.
2. To restrict an allowed request frequency per IP.

$X_1 = \{x_4, x_{11}\}$ – defined subset of HTTP-request attributes

Substrings = {"bot", "index", "spider", "crawl", "wget", "slurp", "libwww", "curl", "wget", "lynx", "urllib", "ruby", "php", "perl", "python", "java", "http://"}

So, checking the incoming HTTP-request by this module is handled according to the following rule:

$$IF\ (x_4\ is\ missing)\ OR\ (x_4 \cap Substrings)\ OR\ (x_{11} > T)\ THEN\ r \to R_{NL},$$

where T is an allowed threshold of requests frequency.

Checking the client's ability to JavaScript code processing. During DDoS-attack HTTP-requests are sent through web-browser not by user, but by specialized intellectual agents with limited functionality. Generally they are not able to process JavaScript, Adobe Flash, to send or accept cookies. In our view, these affinities may be used for testing requests legitimacy.

Common scheme of testing request legitimacy is shown on Fig. 3.

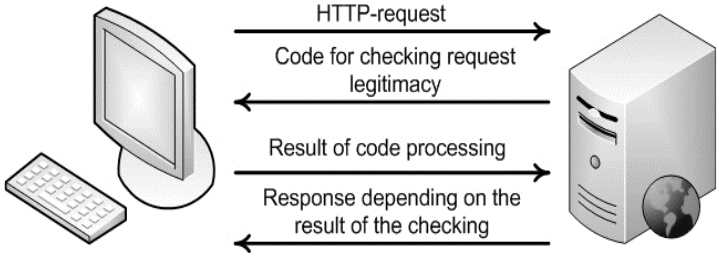

Fig. 3 – Common scheme of testing request legitimacy

A classical request processing involves two stages: request and response [1]. The proposed scheme contains two extra stages. After receiving a client request a server sends a code for checking request legitimacy. It may be math calculation with JavaScript, Adobe Flash, handling cookies or other methods intrinsic for web-

browser. In the case of a successful code execution the client receives the content, which has been requested and specified in request headers [3].

Let's denote the ability to process a JavaScript code as x_{13}.

$X_2 = \{x_{13}\}$ – defined subset of HTTP-request attributes

$$x_{13} = \begin{cases} 0, & \text{if the client has not handled the JavaScript code} \\ 1, & \text{if the client has handled the JavaScript code} \end{cases}$$

A check of incoming HTTP-request by this module is handled according to the following rule:

$$IF\ (x_{13}\ IS\ EQUAL\ TO\ 0)\ THEN\ r \rightarrow R_{NL}$$

Making decision on request processing based on geo-attributes. In some cases it is appropriate to restrict a request processing based on clients belonging to certain geographical territory. It concerns web-resources, a core target audience of which is concentrated in certain geographical territory. For example, a company, which carries on business in one or several countries, is interested in successful access to its website clients from these countries first. From this point of view, requests from other geographical territories may be neglected.

To implement this level of flow analyzer a GeoIP database may be used. On the basis of this database an IP detects a geographical location of network host. During a request an IP address is sent to a GeoIP database. It is possible to receive a two-letter country code according to ISO-3166-1 [4] as a response.

Let's denote a belonging of client IP-address to certain geographical territory as x_{14}.

$X_3 = \{x_{14}\}$ – defined subset of HTTP-request attributes.

$C = \{C_i\}$ – set of two-letter country codes, access from which is allowed.

A check of incoming HTTP-request by this module is handled according to the following rule:

$$IF\ (x_{14} \notin C)\ THEN\ r \rightarrow R_{NL}$$

Intellectual decision-making based on the signature method.

Application of the signature method on HTTP protocol level allows us to detect an intellectual agent with a high probability. This method of identification is based on the matching client's request headers to a set of predefined criteria. In such a case a signature is a result of checking headers matching the criteria, which is recorded in a particular order defined by the algorithm. It is possible to identify a client, having a previously prepared list of signatures.

It is necessary to notice, that one client can match more than one signature. It is caused by different principles of HTTP request headers generation depending on the type of request data and a stage of a dialog with server [2].

The length of a signature (Fig. 4) is fixed and is equal to 48 bits, or bytes, or 12 characters in hexadecimal.

FF0-57F4CCBB8

Fig. 4 – Example of a signature

A signature data structure is summarized in Table 2:

Table 2.
A signature data structure

Bits range	Bits count	Meaning
1-8	8	Information on a client support of latest protocols, presence of required headers and absence of malformation.
9-12	4	Flags, by which a client is leveled according to the algorithm categories ("connection via proxy server", "mobile device", "has strong characteristics of an intellectual agent").
13-16	4	Particular client attributes, specific protocols assistance, data syntax etc.
17-24	8	Availability of headers from the predefined set (from, user-agent, host, connection, accept, accept-encoding, accept-language, accept-charset).
25-48	24, 8 to 3	Order of headers, listed above.

The list of criteria, which we used in the algorithm, is sorted by bits order:

1) HTTP protocol version is not 1.0, if header "accept" contains "text/html".
2) Headers "host", "connection", "user-agent", "accept" are present.
3) Header "connection" contains "keep-alive" and does not contain "close".
4) Length of header "User-agent" is more than 16 symbols and contains "(" symbol.
5) Header "accept" contains "*/*" element, header "accept-language" does not contain "*" and in the case of "accept-charset" header presence, it contains "*".
6) Header "accept-encoding" does not contain substrings "identity", "x-gzip", "te", "keep-alive", "z-uidh";
7) Header "accept-encoding" contains "gzip" substring.
8) Header "Accept-encoding" contains "deflate" substring.
9) At least one of the headers: "x-forwarded-for", "via", "x-bluecoat-via", "x-proxy-id", "x-

piper-id", "clientip", "proxy-connection" is present.

10) At least one of the headers: "x-operamini-features", "x-operamini-phone", "x-operamini-phone-ua", "x-nokia-musicshop-version", "x-nokia-musicshop-bearer", "x-wap-profile", "x-att-deviceid", "x-ebo-ua", "device-stock-ua" is present. Or header "user-agent" contains a substring: "android", "bada", "iphone", "ipad", "ipod", "symbian" or "windows ce".
11) Header "from" or "x-goog-source" is present. Or header "user-agent" contains substrings: "craw", "bot", "slurp", "spider", "agent".
12) Header "User-agent" contains substrings: "libwww", "curl", "wget", "lynx", "urllib", "ruby", "php", "perl", "python", "java", "http://". Or the 4th ot 5th criteria is not satisfied. Or the 10th criteria is not satisfied and a tag "pragma" is present, or the headers "host", "connection" or "user-agent" are missing.
13) Value of header "accept" is not equal to "*/*" and does not contain "text/html".
14) Header "accept-encoding" contains substring "sdch".
15) Header "connection" contains uppercase symbols.
16) Header "accept-encoding" does not contain " " (space) symbol.
17) Header "from" is present.
18) Header "user-agent" is present.
19) Header "host" is present.
20) Header "connection" is present.
21) Header "accept" is present.
22) Header "accept-encoding" is present.
23) Header "accept-language" is present.
24) Header "accept-charset" is present.

HTTP-request (example is on Fig. 1) contains header User-Agent with data of client's software and its versions. Application of the signature method gives us a possibility to check information, specified in request headers. Before handling the request a new-formed client signature is compared with signatures, which meet the software specified in User-Agent string. If a client falsifies software, trying to send false User-Agent, a flow analyzer detects the substitution with a high probability. On this ground the decision of request's affiliation to intellectual agent, and in the case of DDoS-attack – to malicious client is made.

Let's denote the client signature of incoming HTTP-request as S_r.

$S = \{S_i | i = \overline{1,k}\}$ is set of database signatures, corresponding to the software, specified in User-Agent HTTP header. This module checks incoming HTTP-request according to the following rule:

$$IF\ (S_r \notin S)\ THEN\ THEN\ r \rightarrow R_{NL}\ ELSE\ r \rightarrow R_L$$

Example of incoming HTTP-request check

Let's assume that incoming HTTP request is one from the listing 1.3. The client IP address belongs to China, and the list of allowed countries includes only Ukraine and Russia. The HTTP-request frequency from the client IP is 10 requests per minute, while the configured threshold is 1 request per second.

1) x_4 = {"Mozilla/5.0","(", "X11", "Ubuntu", " Linux", "x86_64", " rv:18.0", ")"}

$$x_{11} = \frac{10}{60} = \frac{1}{6}(requests\ per\ second)$$

Substrings = {"bot", "index", "spider", "crawl", "wget", "slurp", "libwww", "curl", "wget", "lynx", "urllib", "ruby", "php", "perl", "python", "java", "http://"}

$T = 1$

The condition checked in accordance to the rule is wrong, so the check passes to the next module.

2) Let's assume that a client handles JavaScript code successfully. Then $x_{13}=1$. *The condition checked in accordance to the rule is wrong, so the check passes to the next module.*

3) x_{14} = {"CN"}

C = {"UA", "RU"}

$x_{14} \notin C \Rightarrow r \rightarrow R_{NL}$

The third module makes a decision of HTTP-request illegitimacy and breaks the chain. The fourth module is not involved in the process of checking due to the previous module decision.

Results

Combination of the obtained theoretical and experimental results let us form a thin Nginx [5] and Lighttpd [6] web-server environment.

To exaggerate the malicious client problem of determination the logic of the sent JavaScript-code, a code compilation with further obfuscation is applied on the second level of a flow analyzer. Herewith the readability of a script significantly deteriorates.

Maxmind GeoIP database "GeoIP Country" is used for making decision of request handling based on geo-attributes. Depending on specifics of defended web-resources other versions of GeoIP may be used [7].

Applying the signature method, a web-sites www.amonis.com.ua and www.kononenko.ws visitors' (legitimate clients and intellectual agents) signatures database is formed. The web-sites belong to the authors of the paper. Google, Yandex, Yahoo and Bing intellectual agents' signatures are shown as an example in Table 3.

Table 3.
List of intellectual agents' signatures

Intellectual agent	Signatures
Googlebot 2.1	ff33fc4e0340
Googlebot Mobile	ff73fc4e0340
Google Translate	9e316d9d1a00
Google Web Preview	b73b744cd000, 9f316c86a000, 973964350000
Bingbot 2.0	ff33fc728440, ff337c72a200
Yahoo Slurp 3.0	ff317f466bd8, ff317f466bd8
YandexBot 3.0	f733fe4e5c40
YandexNews 3.0	943970458000

To check a system's defense the synthetic test was carried out. An attack was performed by requests of different difficulty levels, taking into consideration the logic of the system. The chart (Fig. 5) is created in a real-time mode. Requests to HttpStubStatusModule were sent with 30-seconds interval and "Active connections" values were taken.

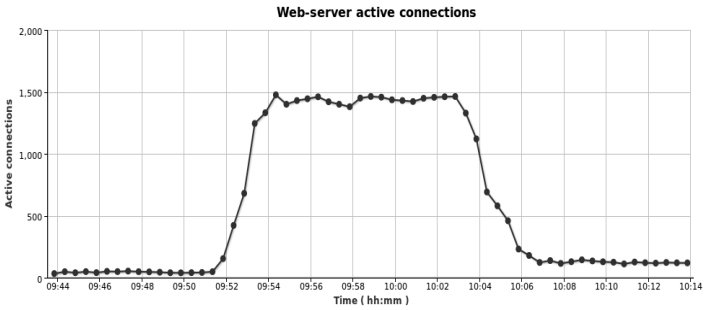

Fig. 5 – Web-server active connections, the requests of which are being handled

The plot can be divided into four specific areas:

before 09:51 – there is no attack. Web-server is processing requests in normal mode;

09:52 - 10:03 – DDoS-attack is active. Defense system is disabled.

10:03 - 10:07 – DDoS-attack is active. Defense system is enabled. Gradual closing of malicious client's session is observed;

after 10:07 – DDoS-attack is active. Defense system is enabled.

As we can see, the amount of active connections during DDoS-attack with enabled defense system exceeds the same value during a normal mode. And at the same time a defense system provides conditions, which allow successful processing of legitimate requests.

Conclusion

In the case of distributed attack (DDoS) obvious signs, which can be used for malefactor detection are missing. Checking HTTP-request legitimacy is not provided by HTTP-protocol or TCP-protocol, which is used as transport protocol. In this paper the method of defense against DDoS-attacks is proposed. It is based on a prior analysis of incoming HTTP-request. Due to a multilevel structure of a flow analyzer a minimization of resources, spent by a flow analyzer for request handling, is achieved.

Malicious requests complete filtration is not achieved. This fact is obvious from the plot, which was built during the synthetic test. But at the same time a defense system provides conditions, which allow a successful processing of legitimate requests.

References

1. RFC4732: Internet Denial-of-Service Considerations: http://tools.ietf.org/html/rfc4732

2. RFC2616: Hypertext Transfer Protocol – HTTP/1.1: http://tools.ietf.org/html/rfc2616

3. Kononenko V.M. A defense model from failure attacks in HTTP-flood servicing / V. M. Kononenko, S. O. Kravchuk // Modern problems of radio engineering and telecommunications "RT–2012": Materials of the 8th international youth scientific conference, Sevastopil', April 23-27, 2012 / Sevastopil' national technical university; ed. J.B. Himpilevych. — Sevastopil': Sev NTU, 2012. – p. 118. [in Ukrainian]

4. ISO 3166-1 decoding table: http://www.iso.org/iso/home/standards/country_codes/iso-3166-1_decoding_table.htm

5. Nginx documentation: http://nginx.org/en/docs

6. Lighttpd documentation: http://redmine.lighttpd.net/projects/lighttpd/wiki#Documentation

7. GeoIP databases and web services: http://www.maxmind.com/en/geolocation_landing

8. Stephen M. Specht, Ruby B. Lee "Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures": http://palms.ee.princeton.edu/PALMSopen/DDoS%20Final%20PDCS%20Paper.pdf

10. Jelena Mirkovic, Peter Reiher "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms": http://www.eecis.udel.edu/~sunshine/publications/ccr.pdf

11. Jelena Mirkovic "Distributed Defense Against DDoS Attacks": http://www.isi.edu/~mirkovic/publications/udel_tech_report_2005-02.pdf

Received in final form April 4, 2013