A Three Engine Application Level Firewall for Web Servers

Gowtham Mamidisetti, T.Divya

Abstract ***Due to insufficient checks on input data in many web applications web servers remain prone to external tampering. This paper proposes ALF (application level firewall) to protect web systems with three new mechanisms. First, ALF provides a fine grained access control policy. Second, ALF allows web application developers to specify the restriction on application running parameters. Finally, ALF collects web user behavior statistics.***

Index Terms—ALF (Application Level Firewall), Attack Signature, CGI (Common Gateway Interface)

I. NTRODUCTION

To counter web attacks, most web servers enforce coarse-grained access control to restrict the execution of web applications within a specified directory that CGI programs must reside. One can also deploy intrusion detection systems or vulnerability assessment systems with known attack signatures to detect malicious requests and vulnerabilities.

Unfortunately, the above approaches leave a lot to be desired. Coarse grained access control mechanisms are not flexible enough and often leave loopholes to attackers. Most IDS systems and vulnerability assessment systems rely on known attack signatures to protect web systems. However, it is hard to keep the attack signature updated with respect to the large number of vulnerabilities discovered daily. This paper proposes ALF (application level firewall for web servers), as a supplement to existing solutions, to help combat web attacks.

II. FOCUS ON CATEGORIES OF ATTACKS

ALF primarily focus on two categories of attacks:

1. Unauthorized accesses*:* Modern web systems usually provide coarse-grained access control to restrict that web applications can be invoked by web clients only if they reside in a specified directory (e.g., /cgi-bin). However, the coarse grained access control often gives attackers opportunities to exploit configuration error and compromise the web system. An example attack is what we will call the ***bypass execution*** attack. CGI programs that are invoked from user input by the web server often need to run helper scripts or programs internally.

The intent of the programmer is that the helper programs should not be invoked directly by a client. For example, a CGI program may authenticate a user and then invoke a helper perl script to access a database if the user is valid. Unfortunately, if the helper program is put in the same directory as the CGI program, it can be invoked by a malicious client directly (via the web server, but without going through the parent CGI program). Thus, attackers can bypass the user authentication and violate web server security.

2. Abuse of CGI programs with parameters:

CGI Developers are supposed to do input validation and filter out requests with invalid parameters, but they often fail to follow a sound security methodology and overlook the input error checking. Attackers can exploit the vulnerability of weak input validation to send CGI programs the parameters that do not meet the normal length or format restrictions and cause SQL injection or buffer overflow attacks.

Many database systems, such as MySQL, allow users to insert multiple records in a line, this SQL command will allow the attacker to insert two records instead of one as expected. The reason of this SQL injection attack is a security bug: the user input validation is insufficient.

III. LEVEL OF PROTECTION

ALF helps to protect against a wide-range of common vulnerabilities with the following three mechanisms:

1. To prevent unauthorized access to web files, ALF provides fine-grained access control policy and enforcing it at the perimeter of a web server. With this web administrators can classify web clients into variety of roles and specify their access permissions to web objects at the granularity ranged from directories to files. In addition, rather than allowing all files in */cgi-bin* directory to be executed by web clients, WSF allows a web application to be invoked only if it is explicitly specified as executable to web clients, which effectively prevents the bypass execution attack.
2. To prevent abuse of web applications, ALF proposes an input validity specification to allow developers to specify the valid input patterns instead of requiring enumeration of all possible malicious inputs, which substantially simplifies the input validation task.
3. ALF also collects user behavior statistics on a per-user/per-IP basis. The behavior statistics can be used to detect abnormal web activities and heuristically change the access policy to proactively delay or block the requests from malicious users.

IV. DESIGN OF ALF

A. System Overview

Manuscript received November, 2013.

Gowtham. Mamidisetti is an assistant Professor in Information Technology at Shri Vishnu Engineering College for Women, Bhimavaram, West Godavari Dist, Andhra Pradesh, India.

T.Divya is a student in Information Technology at Shri Vishnu Engineering College for Women, Bhimavaram, West Godavari Dist, Andhra Pradesh, India.

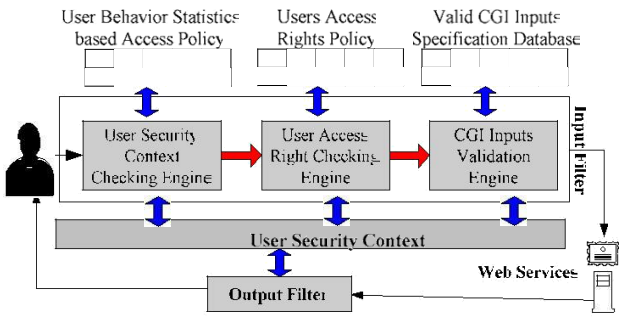

Fig 1. The architecture of ALF

As shown in Fig 1, ALF consists of the *input* and *output filters*. *Input filter* deep inspects the incoming HTTP requests to reject invalid web accesses. *Output filter* collects the status of outgoing responses. Response status information helps infer user behavior patterns.

ALF maintains a per-user *security context*. A security context in ALF is indexed either by the user's IP address or by a user ID (if the user authenticated to the web service). The security context contains the user's past behavior statistics, such as the number of invalid requests, the number of failed requests, and the number of requests during a specified time interval. All those behavior statistics are updated by the input and output filters.

V. INPUT FILTERS

The input filter deploys three engines: *security context checking engine, access right checking engine,* and *CGI input validation engine*. These engines check the incoming requests one by one. An incoming request will be forwarded to the protected web server only if it goes through the checks of the three engines.

1. The Security-Context Checking Engine

The *security-context checking engine* examines the user ID and the IP address of the request to see if requests from the IP address or the user ID should be blocked or delayed. Administrators can use the security-context checking engine to temporarily block a user's access to the web server if their statistical behavior, recorded in the security context, violates specified limits (e.g., too many failed authentication requests within a short interval). Therefore, the security context essentially works as a "credit history report" to help ALF monitor a client's abnormal behavior pattern.

User Behavior Auditing

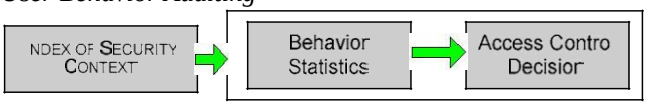

Fig 2. ALF Security Context

As a complementary mechanism, ALF also supports tracking and auditing of web user behaviors. ALF maintains a security context for each web client.

The security context is indexed with the client's user ID if the client is an authenticated user.. If the client is an anonymous guest, the security context is indexed with the client's IP address. As Figure 2 shows, the ALF security context contains three parts of user security information:

1. Index of the security context (User ID or IP address);
2. Behavior statistics;
3. Access control decision based on the behavior pattern.

ALF uses the *index of the security context*, IP address for unauthenticated user and User ID for an authenticated user, to locate a user's security context.

The *behavior statistics* part contains cumulative user behavior patterns, measured over multiple configurable time-intervals on a per-user/ IP basis:

The number of received requests. This data is collected by the input filter.

The number of bytes sent out. This data is collected by the output filter.

The number of invalid requests. This data is collected by the checking engines in the input filter. Any request that violates ALF security policies will be counted as an invalid request.

The number of failed requests. This data is collected by the output filter. Any request with the HTTP status code that does not fall into the period between 200 and 307 will be counted as a failed request.

The number of failed authentication requests. The field helps to prevent brutal force password guessing attacks. It is collected by the output filter.

The user behavior statistics help to detect abnormal behavior pattern and proactively adjust access control policies. For example, excessive authentication failures of a specific user may indicate that a hostile party is mounting brutal force password guessing attack or this user forgets the password. To thwart password guessing attack, web administrators can configure ALF to suspend this user's further authentication requests for several seconds upon the number of failed authentications exceeding the specified threshold.

2. The Access Right Checking Engine

The *access right checking engine* checks the requested URI against the access right policy. With the access right control, ALF can limit authenticated or unauthenticated users to only specified web files/services and prevent unauthorized access to the sensitive files that are left accidentally in public web directories. The access right checking engine provides fine-grained control, rather than standard access control imposed by web servers.

Access Control Policy

ALF defines an access control policy language to allow administrators to explicitly define the access rights to web entries, including normal data files and CGI programs.

An access rule is a mapping as follows:

Web_Entry → Web_User: Access_Right

The *web entry* defines the object on which the access rule should apply. It can be a specific file, a class of files with a wildcard pathname or a directory. The *web user* defines the subject that is allowed to access the web entry. It can be a specific user or a web group. The *access right* defines the authorization under which a web user can access a web entry. The access right mapping means: the "*web_entry*" can and only can be accessed by the "*web_user*" under the "*access_right"* authorization.

An access policy usually includes three parts:

1. Definition of valid user set and user groups
2. Definition of default accessible file types
3. Definition of access right rules of web entries

The first part defines the valid user set and user groups. The second part contains the default accessible file types (i.e. **.html* and **.jpg* files*)* for the web system. The accessible file types can be defined by file type extensions or certain file name patterns. By default, only common web file types are included, which helps prevent unauthorized accesses to sensitive files, such as "creditcard.dat", that are left in the public web directory.
The third part specifies the access right of users to web entries. An access right policy may include multiple access rules. Each rule defines the access right of one URI entry. A URI entry can be defined as a specific file, a class of files with a wildcard pathname or a directory. Wildcards are allowed and only allowed in file name to represent multiple files with similar name pattern. If an access rule defined for a directory, this access rule applies to all files and sub-directories under this directory if they are not associated with access rules. In other words, if no access rule is defined for a directory or a file, permissions are inherited from the parent directory. The access right rules are prioritized as follows:

root directo ry → sub -directory(level1) →

sub - directo ry(level2)... → a class of files → single file

The access rule of root directory has the lowest priority and access rules of single files have highest priority. Rules with higher priority have precedence in policy enforcement.

3. CGI Input Validation Engine

The CGI input validation engine checks the parameters carried in the CGI request against the input validity specifications. Only requests with valid inputs can be sent to the web server. The CGI input validation helps mitigate many buffer overflow attacks and SQL injection attacks that compromise web systems via sending malicious parameters to CGI programs.
CGI Input Validity Specification
Because the inputs to CGI programs are complex, fixed attack signatures are often not flexible enough to tell a valid input from invalid ones.
To deal with this problem, ALF provides a fine-grained way to specify constraints on inputs of CGI programs. We use an example to describe how validity specification works: suppose we have a user login script /cgi-bin/login.cgi, it only allows parameter transferred with POST method; the expected input at the user name field is a string composed by 3-8 letters or digits and the expected valid password is a string composed by 6-15 letters and digits. No special character is allowed in the username and password parameters. The validity specification can be defined as follows:

```
< Rule>
   <URI> /cgi-bin/login.cgi <\URI>
   < Method> POST <\ Method>
   < Parameter>
<Name> username </Name> <Value>
      ^[a-zA-Z0-9]{3,8}$ </Value>
   </ Parameter>
   < Parameter>
      < Name> password </Name>
      < Value>          ^[a-zA-Z0-9]{6,15}$
   </Value> </ Parameter>
   <SIG_CHECKING> NO </SIG_CHECKING>
</Rule>
```

The *URI* section contains the URI of the CGI program.

The *Method* section configures which methods are allowed for this URI. The methods that are often used are GET and POST. Other HTTP methods like PUT, TRACK must be used carefully as they may bring vulnerabilities like cross site script attack.
The *Parameter* section defines the validity specifications for parameters of this CGI program. Each possible parameter must have a *Parameter* definition. The validity specification of each parameter consists of two parts: parameter name and parameter value. The parameter name field is the parameter name to be checked while the parameter value field shows the valid parameter value pattern. The valid parameter value pattern is defined with regular expression. If there is no restriction on a parameter, the valid parameter value pattern can be empty. Based on the configured validity pattern, the input validation checking engine can then check whether the user inputs carried in a CGI request is valid or not. Note that only parameters listed in this section will be regarded as valid and checked against the corresponding validity specification. For those parameters whose names are not on the valid parameter list, the input validation engine will directly regard them as malicious. This mechanism effectively prevents many buffer overflow attacks.
The above example shows, the rule clearly defines what inputs are expected by the programmer developers. The CGI program, at a minimum, must take care of inputs that satisfy the above specification. Any other unexpected inputs will be blocked by this specification directly at the firewall. This mechanism does not require developers to enumerate all possible invalid input patterns. Instead, web application developers only need to express their intention of valid inputs with regular express, which substantially simplify the input validation procedure.

VI. OUTPUT FILTERS

The *output filter* checks the status of outgoing replies and updates the behavior statistics in the security context. In addition, the output filter also helps the input filter to track the user information and generate the user tracking tag for each source.

VII. CONCLUSION

ALF proposes a policy-based framework to provide perimeter security for those web services. With proper policies, ALF can help to thwart unauthorized accesses

to system sensitive files and achieve flexible, role-based access control. To prevent attackers from sending allows administrators to explicitly define the input validity specification for each accessible CGI program. Instead of inferring all possible attacks from known attack signatures, ALF checks incoming requests against the input validity specification, which simplifies the procedure to determine *whether a use input is valid or not. In addition, ALF collects* user behavior statistics, which helps web administrators to detect abnormal user behaviors and proactively adjust the access control maliciously manipulated requests to CGI programs, ALF policies.

REFERENCES

1. Scott, D. and R. Sharp. Abstracting Application-Level Web Security. in Proceeding of the eleventh international conference on World Wide Web (WWW'2002). 2002.
2. 2003 ACM Press: Washington D.C., USA p. 251-261.
3. CERT Center, Microsoft Internet Information Server (IIS) vulnerable to cross-site scripting via HTTP TRACK method, 2004. CERT Advisory, "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL, 2001.
4. Anley, C., Advanced SQL Injection In SQL Server Applications, 2002.
5. SAINT Corp., SAINT vulnerability scanner. Nikto, Nikto 1.32. Symantec Corp., Symantec NetRecon.
6. Kruegel, C. and G. Vigna, Anomaly detection of web-based attacks in Proceedings of the 10th ACM conference on Computer and communications security.
7. Ristic,I.,Introducingmod_security,2003. http://www.onlamp.com/pub/a/apache/2003/11/26/mod_ security.html
8. Vigna, G., et al. A Stateful Intrusion Detection System for World-Wide Web Servers. in Proceedings of the 19th Annual Computer Security Applications Conference. 2003.
9. Nessus, NESSUS Scanner, 2004. Forristal, J. and G. Shipley, Vulnerability Assessment Scanners.

AUTHOR PROFILE

Gowtham.Mamidisetti is an assistant Professor in **Information Technology** at **Shri Vishnu Engineering College for Women**, Bhimavaram, West Godavari Dist, Andhra Pradesh, India.

T.Divya is a student in **Information Technology** at **Shri Vishnu Engineering College for Women**, Bhimavaram, West Godavari Dist, Andhra Pradesh, India.