AN “ALMOST” SAFE CGI SCRIPT

Ulisses Thadeu V Guedes **- GURI SoftHouse**

Table of Contents

1 INTRODUCTION

I had involved with network security programming since 1997 and network programming since 1989. So for long time I had accumulate some experiences and research. Since 2005 I had observed some health changes and found “him”. “He” is there, inside, waiting for something that I don't know.

This article was resulted after one Symposium request to build a site where the internet user could submit papers and abstracts.

One of most common problem under this condition is the form field tempering. This action is responsible by ninety percent of fake report and broken system security.

The article must be considered as a directive programming and not a general solution of the purpose.

2 REVIEWING THE INTERNET SCENARIO

As we know, the questions start up research for answers.

Under Internet environment, is there any way to control one malicious user action, once each malicious user has the form under your own control? He can visit the site, save the form, tempers it by changing or removing any javascript validation and submit the data without those validation criteria.

This situation yet shows that the use of javascript, or any other language, executed at Internet user system may be changed. Once this is acknowledged then the web site developer must take care about CGI processes, checking a set of variables provided by HTTP application server. The real validation must be done by CGI, not in html form.

Each form is developed to attempt some project request. For instance, one CGI was developed to work with a form that will send data under one type of method: usually POST and/or GET. There are a set of other request method type available by HTTP protocol: PUT, HEAD, DELETE, and so far. These request methods are described at RFC2686, available at IETF site (www.ietf.org).

So, the CGI must check and accept only what was predicted by form method or methods and discard the other ones. Any other must result at a security response: block the access from that IP is better solution, but there is situations where the service must be suspended to avoid system intrusion or even CGI must has an "education criteria". This is the real means of security: The ability to keep at safe state while it is possible. There isn't a secure system. Exists system under security criteria.

Web-Robots (Google, Yahoo!, and so far) visit the site and pages allowed by robots.txt file, but an intruder has other purposes, no matter what the disclaimer or usage policy rules said. The human behind this scenario has only one goal: assume the site or system site control. Good internet users follow the Internet rules and protocols. Intruders follows scripts to be a winner and get success on attack, to change the pages (fake pages, warnings "The hacker group did it.."). It is the haven of glory.

Under "personal security" point of view, the best target must auto-protected using proper skills, but also following some security concerns. The target eyes must have the ability to check and evaluate the environment, the offender, and advise the fragile target about any kind of threat.

The whole process starts with the maximum security survey of the environment that the target will be exposed. In security systems, you must know the application running with it as a fragile element immersed in a hostile environment.

3 EVALUATING THE APPLICATION ENVIRONMENT.

Once the goal of the article is the CGI, then the caller, or the start-up execution control, is passed by HTTP application server. Thus, as the concepts of software engineering and security, we must know what is reported and how to use that information.

According to the sending form data method, the HTTP application server defines some variables under system environment. These variables may vary from one application to another. So these must be checked.

APACHE, for instance, attempts for a set of environment variables according to the method and the "enctype" defined at form. The datum reported by client must be discarded or used as an element to be matched against the same element detected by application. The programer must believe the facts he has in hands what is coming from a trusted source and not on data of questionable origin.

1) Under METHOD GET, enctype "application/x-www-form-urlencoded", APACHE returns:

SCRIPT_NAME, SERVER_NAME, HTTP_REFERER, SERVER_ADMIN, HTTP_ACCEPT_ENCODING, HTTP_CONNECTION, **REQUEST_METHOD**, HTTP_ACCEPT, SCRIPT_FILENAME, SERVER_SOFTWARE, **QUERY_STRING**, REMOTE_PORT, HTTP_USER_AGENT, SERVER_SIGNATURE, HTTP_ACCEPT_LANGUAGE, **REMOTE_ADDR**, SERVER_PROTOCOL, PATH, REQUEST_URI, GATEWAY_INTERFACE, SERVER_ADDR, HTTP_HOST, UNIQUE_ID.

CONTENT_TYPE and CONTENT_LENGTH are not available.

2) Under METHOD GET, enctype "multipart/form-data", APACHE returns:
The same as 1)

3) Under METHOD POST, enctype "application/x-www-form-urlencoded", APACHE returns:

SCRIPT_NAME, SERVER_NAME, HTTP_REFERER, SERVER_ADMIN, HTTP_ACCEPT_ENCODING, HTTP_CONNECTION, **REQUEST_METHOD**, HTTP_ACCEPT, SCRIPT_FILENAME, SERVER_SOFTWARE, **QUERY_STRING**, REMOTE_PORT, HTTP_USER_AGENT, SERVER_SIGNATURE, HTTP_ACCEPT_LANGUAGE, **REMOTE_ADDR**, SERVER_PROTOCOL, PATH, REQUEST_URI, GATEWAY_INTERFACE, SERVER_ADDR, HTTP_HOST, UNIQUE_ID, **CONTENT_TYPE** and **CONTENT_LENGTH**.

4) Under METHOD POST, enctype "multipart/form-data", APACHE returns:

SCRIPT_NAME, SERVER_NAME, HTTP_REFERER, SERVER_ADMIN, HTTP_ACCEPT_ENCODING, HTTP_CONNECTION, **REQUEST_METHOD**, HTTP_ACCEPT, SCRIPT_FILENAME, SERVER_SOFTWARE, **QUERY_STRING**, REMOTE_PORT, HTTP_USER_AGENT, SERVER_SIGNATURE, HTTP_ACCEPT_LANGUAGE, **REMOTE_ADDR**, SERVER_PROTOCOL, PATH, REQUEST_URI, GATEWAY_INTERFACE, SERVER_ADDR, HTTP_HOST, UNIQUE_ID, **CONTENT_TYPE** and **CONTENT_LENGTH**.

The variables under bold characters are those that must be verified and handled by CGI.

The **REQUEST_METHOD** may be forged but the programmer knows the real method defined at form file. If it was forged then all other data sent MUST be immediately discarded due to confidence broken. It stores the action verb of request: GET, POST, HEAD, DELETE and others, as predicted by HTTP protocol (See RFC2826 or later).

The **QUERY_STRING** has the arguments of the action verb and contains the values of form fields at same sequence that they are defined between <FORM> and </FORM> tags. Its value ALWAYS must be handled as data. In fact, QUERY_STRING is a character chain with form field name and values encoded. Someone, in the past, had the "star idea" of execute that chain after replace the small changes, opening a big-bang and black hole. So, don't make these mistakes again.

The **REMOTE_ADDR** contains the IP address of the browser (client application). This information is useful to limit, block or allow resources usage provided by application server (HTTPd). The CGI can do those but under limited restrictions. The best security policy for blocking is that where the CGI calls and interact to a packet filter (iptables, ipfw, ipchains and so far). Usually packet filter runs under privileged account and CGI runs under low level privileged account. This scenario requests the usage of pipes or sockets and taking care to avoid expose environment variables directly from web client to packet filter.

CONTENT_TYPE has the 'enctype' information predicted by html form. It is passed by client application and also can be forged. Although, and again, the web developer knows the enctype predicted by form. If it is matched with the real enctype this let a confidence weight about the client. Once this confidence is broken then some security action MUST be applied. When the enctype is 'multipart/form-data' then the CONTENT_TYPE also stores the boundary string and it MUST also be used for confidence balance.

CONTENT_LENGTH is provided with useful value when the REQUEST_METHOD is POST or PUT. The value stored at CONTENT_LENGTH is filled by client application. It also may be forged what limits its usage, only, to match the real value computed by CGI.

3.1 SCENARIO 1

The first step of this approach is check the METHOD predicted by form. If the form uses GET than the REQUEST_METHOD also MUST BE GET. Any other method must discard the

contents, because it means that the form was tempered. One security approach is: insert the IP address (REMOTE_ADDR) as suspected. Is that IP already exists as suspected, then is possible to block the access from that IP. (See NetIptables description).

At first scenario (REQUEST_METHOD = GET and CONTENT_TYPE=NULL, that has the default of application/x-www-form-urlencoded), the HTTPd provides the values encoded at QUERY_STRING, but does not provide the size of QUERY_STRING.

But this may be turn around because the size of each field of the form is known by developer. He sets that size and the names of the fields. This gives implicit information.

If it assumes the fields, with names "f1", "f2" and "f3", and maximum sizes "10", "20" and "15", respectively. The QUERY_STRING must have the maximum size:

f1=0123456789&f2=01234567890123456789&f3=012345678901234 (Complete string)

123456789012345678901234567890123456789012345678901234**6**
000000000111111111122222222223333333333444444444455555**5**

or the minimum size

f1=&f2=&f3=

12345678901
00000000011

So the maximum size of QUERY_STRING must be **56** and the minimum size must be **11**.

Although, the contents of QUERY_STRING are encoded since the ENCTYPE of the form was so defined. This situation let the developer to write the QUERY_STRING contents to a temporary file.

DO NOT HANDLE THE QUERY_STRING CONTENT UNDER MEMORY. This approach MUST be avoided because any malicious action can be done. Storing it at file, any malicious code will not be executed and the results stored there, and not in memory.

By writing the contents to file, under encoded form, the CGI can detects malicious action because it will handle the string as data and all malicious code may be detected, as it will be presented later. The real size of QUERY_STRING is the size of the file subtracted by 1 byte (the EOF byte).

All fields of the form will have a set of allowed characters due to the characteristics of the each field. One field, that will store the complete name, for instance, must have only letters from "a" upto "z", "A" upto "Z" but not numbers. Some language may use accentuation and punctuation. So, these last letters types will be stored under encoding format and must be decoded. You can program that by using 'awk', 'sed', or some application that takes those values as data and will not allow execution at background.

The decoded string, after using some specific application, will result into another temporary file. The resulted file will have all letters decoded, and the minimum and maximum size of that file can be matched with the maximum and minimum size as predicted by form field sizes. The size of fields must be large enough but not so large as possible.

Until now, we have decoded the full QUERY_STRING contents, that allow us to check the minimum and maximum size of the that.

WARNING: Some browser also inserts the name of the form at end of QUERY_STRING. So this name must also be computed as part of QUERY_STRING size.

As commented before, each field has its own allowed kind of characters. The QUERY_STRING can be now, split into form field name and form field value, by replacing the characters "&" and "=" by space. The browser encodes the space character of one input form value as the symbol "+". Once split, the odd indexes of the string array will have the name of the form field and the even indexes will have their values.

Now, it is time to check the sequence of the field transferred. One real browser sends the form fields at same sequence that they appear in the form (HTML file). If the sequence was broken then it means that the form was tempered and those data MUST BE DISCARDED. Nor PhP, neither Python neither ASP neither Perl provide the sequence information, unless the developer CGI program uses low-level or basic functions. The same security approach said before is applicable (mark the IP as suspected).

If all things are right for while, that is time to validate each field value by restricting the range of valid characters. Field Names may not contain numbers, and field for numbers must not contain letters or punctuations, except the symbols (+ and/or -, if they are applicable). For instance, zip codes are numbers and have a specific format at some country: xxxxx-yyy. For phone number, some developers follows the format (international code) (phone operator)(region code) (phone number). The suggestion is to split the phone number at several fields. The same is applicable to Email address, because the symbol (@) has special means for a lot of command interpreter or language used at CGI programs. So one e-mail must be composed by two fields.

The value of field validation consists at to remove any valid character. If one or more characters remain then it is, or they are, invalid and the security approach may be applied.

Everything was checked. Only now your CGI can handle the form data. They are safe!

3.2 SCENARIO 2

The first step of this approach is check the METHOD predicted by form. If the form uses POST then the REQUEST_METHOD also MUST BE POST. Any other method must discard the contents, because it means that the form was tempered. One security approach is: insert the IP address (REMOTE_ADDR) as suspected. Is that IP already exists as suspected, then is possible to block the access from that IP. (See NetIptables description).

At first scenario (REQUEST_METHOD = POST and CONTENT_TYPE=application/x-www-form-urlencoded), the HTTPd does not provide the size of QUERY_STRING (it is empty) and all data are available at buffer STDIN of CGI. The data follows the same format as described at SCENARIO 1, but are available at another instance (the STDIN buffer).

Even in C language, the STDIN buffer of File Descriptor, does not provide its size information, that is needed for consistency check. So, a suggested solution is copy the STDIN contents to a temporary file. The file descriptor of a file always provided its size.

At this scenario, the CONTENT_LENGTH and CONTENT_TYPE are available. The value of CONTENT_LENGTH does not include EOF mark, or the 0x0A value. So the content of the file must be, the value of CONTENT_LENGTH +1. If the size of values differs then this means data tempering and the security approach is applicable. The CONTENT_TYPE MUST

be application/x-www-form-urlencoded. If this value differs then the security approach is applicable.

The first check is to verify the REQUEST_METHOD value (MUST be POST) and the CONTENT_LENGTH value, as observed before.

By saving the STDIN contents into file this will let apply the same field validation used for SCENARIO 1.

3.3 SCENARIO 3

This scenario differs from the others because the enctype. Usually this method is common to send local files (submit files) to HTTPd.

The CONTENT_TYPE variable brings two important values. The first is the "multipart/form-data" string, and the second, (separated by the character ";" from the first) is the boundary string.

Let one example of the CONTENT_TYPE:

CONTENT_TYPE: multipart/form-data; boundary=----------------------------9161194987703

Boundary is a string mark used to define and separate each field name from its value. At the example, below, we consider those 3 previous fields including the fourth field of a file that will be uploaded.

```
----------------------------9161194987703
Content-Disposition: form-data; name="f1"
<CR>
0123456789
----------------------------9161194987703
Content-Disposition: form-data; name="f2"
<CR>
01234567890123456789
----------------------------9161194987703
Content-Disposition: form-data; name="f3"
<CR>
012345678901234
----------------------------9161194987703
Content-Disposition: form-data; name="f4"; filename="Curriculum.pdf"
Content-Type: application/pdf
<CR>
%PDF-1.4 ……..
…..
%%EOF
<CR>
----------------------------9161194987703--
```

Let us understand the content syntax:

The contents starts with the boundary string provided by CONTENT_TYPE/boundary part. The next line has the string, following the syntax:

Content-Disposition: form-data; name="field name"

All the tree first fields are TYPE=”text” at form. The latest is TYPE=”file”, what explain why the latest string of Content-Disposition has the label 'filename'.

Also, since there is a file to be transferred, it is needed the MIME Content-Type of that file. This explain the “Content-Type: application/pdf” of example. Some broken browsers did not inform the real contents sending text/html content-type.

Once know the sequence, it is easy to program what must be waited for each record.

The value of each field is not encoded. They are at RAW mode. So, if the user typed “/0xcc” the value will have that same value. This brings some restriction. If the browser overwrites the character-set than it means trouble for data handling.

Until now, there is no javascript function that provides the data encoding type at sending time. Some browsers implement functions such as:

document.charcode;
document.charset;
document.characterSet;
document.defaultCharset;
document.getUserData;

and so far;

Although these javascript functions are not implemented at all browsers according to http://code.google.com/p/doctype/wiki/DocumentObject, neither the same way (resulting the same answer). This did not means that once defined accept-code of a form field will be filled with that character code, once the user can configure the browser to overwrite those settings.

Following the HTML specifications, for each form field you MUST define the encoding character set, by inserting the tag 'charset=”char-set-code”'.

A good practice is inform the “good” user to use one specific charset for compatibility. “Bad” users (for malicious action) will fake that information, that will become a good source of internet user confidence. After data conversion may possible detects the presence of invalid characters at form field values and, of course, will result at execution of some security rule.

One sample of code to read the form-filed data and values presented above, follows:

(a) get boundary from environment (boundenv)
(b) read boundary
(c) match boundary and boundenv
(d) read record with field name
(e) match syntax, checking the full string, restoring the name of field.
(f) Check if the sequence of field name
(g) if it is out of sequence call security action and exit
(h) read record with size =1 (CR)
(i) if it has not 1 byte then call security action and exit
(j) read record with value, storing it at specific memory region.
(k) check for invalid characters. On invalid chars then call security action and exit
(l) store the value.

… repeat the flowchart (b) upto (l)for the next 2 fields.

The last field value is:

(e1) match syntax, checking the full string, restoring the name of field and the filename.
(f1) Check if the sequence of field name
(g1) if it is out of sequence call security action and exit
(h1) validate the filename syntax. On invalid chars then call security action and exit.
(i1) read record content-type and match if the type is allowed. On error call security action and exit with size =1
(j1) read record with size =1 (CR)
(k1) if it has not 1 byte then call security action and exit
(l1) at this time the read pointer is pointing to the first byte of uploaded file. Call 'stat' or equivalent to know the size of the file, remove the boundary size, remove 2 bytes related to line control and 2 bytes of last boundary mark "--".
(m1) Read all bytes storing it to file with same name provided prefixed or suffixed with some unique identifier.

You are ready for data handling. They are safe!.

As you can see, the CGI process complexity can increase a little bit according to the ENCTYPE value, but it is not impossible at point of view to use perl, php or any other Ready Script Processor. Perl and PhP scripts available did not cover all security concerns creating security holes.

4 THE NETIPTABLES

Iptables is the common packet filter available at Linux. Under windows system the packet filter rules may be programmed by using netsh command. Both request a privileged account to run and implement some security action.

Usually, the HTTPd runs under low privileged account. So, the only way to allow interaction between iptables and httpd is through loopback network connection over TCP protocol, or Unix socket, or pipes, or fifos. Why TCP and not UDP? UDP allows fake return address. TCP did not because the TCP state machine. Packets from network interface can not reach the loopback interface. This is possible to be done using UDP, but it means additional security actions. TCP/IP is available at all systems (Unix, Windows, Linux, IOs, etc).

So, a simple program may listen the loopback interface and execute the security action, blocking the access trough Internet interface to be or had being used by intruder or potential intruder.

NetIptables can be used, also, listening at intranet interface. So you can let your DMZ services secure about intrusion due to HTTPd/CGI and Netiptables interaction.

I hope that this article provides some useful information for developers and TI system managers. If you have some suggestion that could be included here then let me know.

"*There is no way to build a system completely secure and safe against intruders, but there is a way to allow only valid information given to the application the rights of self-defense. This is the goal of basic security fundamental!*"