A Web 3.0 solution for restraining the web bots access to the on-line displayed content

Cosmin STRILETCHI, Mircea F. VAIDA
The Technical University of Cluj-Napoca, Baritiu Str. No. 26, Cluj-Napoca, Romania
cosmin.striletchi@com.utcluj.ro, mircea.vaida@com.utcluj.ro

Abstract. *This paper presents a technique meant for eliminating the access possibilities that web-bots can use in order to perform automatically some on-line tasks destined to the human operators. The applications that benefit from the proposed solution are client-to-business or business-to-business web distributed products. The solution refers to protecting the e-forms embedded into the web pages and the implementation can be regarded as a CAPTCHA. Unlike the classical approach to the matter, the solution inserts no supplementary fields in the page's content. The modules loaded onto the client machines can make the difference between a human person and a web robot.*

Keywords. On-line data security, web distributed applications, human and machine actions, web bots, CAPTCHA, information reliability, AJAX, Web 3.0

1. Introduction

Offering distributed data processing facilities can be considered as a must of the nowadays software applications. Whether they are business-to-client or business-to-business products, the automated software environments gain a lot by allowing the distant access to the offered facilities. Due to the fact that a large majority of the applications have to manipulate text and multimedia information and this aspect is closely correlated to an enhanced security need due to the sensitivity of the processed (entered, transmitted, stored, retrieved and displayed) data. Also, the larger is a software application's target, the higher is the pressure applied on it.

All distributed applications are (or is recommended to be) controlled by a series of server stored modules that monitor the entire data flow [1]. The main field of interest of this paper is to analyze the so called „mobile" software modules are loaded onto the client machines during the runtime and also to depict the inherent risks they manifest. The emphasis will be laid on the protection measures that need to be implemented for enhancing the stability of the entire software product.

The Internet environment is crawled by a lot of automated software applications. Some of them are responsible with data gathering, others have to use the offered communication tools (e-forms for controlling the system-user interaction) in order to send some information to the server that controls the application.

These web robots can be harmless, but there are also some situations where their activity creates a lot of prejudice to the targeted software application.

In order for a software application to be fully protected, its code has to take into consideration the latest technologies provided as support for the Internet distributed environment. The new Web 3.0 [2] attributes and data processing technologies and facilities open a series of opportunities that must be used in order for the developed web products to be fully up-to-date.

2. Existent context

Started with the premises that the server machine is protected against human or software based malware, the client side part of the application can be regarded as the only doorway that may be exploited for attacking the application. The software modules that are available to the visitors of a web accessible product can be regarded as tools (in fact the only available tools) that are to be used in order to perform some illegal actions.

The web accessible content consists in fact in a series of software modules that perform the I/O operations via a web interface. The problem that arises here is that the web accessible content can be approached in other ways than by regular web browsing and by using some specific dedicated

software applications (Mozilla Firefox [3], Internet Explorer [4], Opera [5], Safari [6], etc.).

There is an entire series of available technologies that allow the automatic communication with a target web page. In some cases, this process can be regarded as being benefic. But there are some situations when the very nature of a specific web-accessible software application requires that the distant entity that communicates with the server side software modules is to be strictly a human operator. Some examples of this kind of situations could include (but are certainly not limited to) the filling of a series of dedicated e-forms: registration modules, authentication modules, web-blogs posting modules, etc.

This requires for a practical solution that, once implemented, is able to make the difference between a human operator and an automatic software process trying to simulate a human presence.

The "web-bot" generic term is the one that denotes the software applications able to perform automatically a series of human reserved actions [7].

The already consecrated solution for limiting the web-bots actions is represented by the CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*) fields [8]. To put it shortly, a CAPTCHA field consists in an image that represents a text with some distorted characters along with a text field where the human operator is supposed to enter the exact character sequence presented in the image. Figure 1 depicts an example of such a CAPTCHA field.

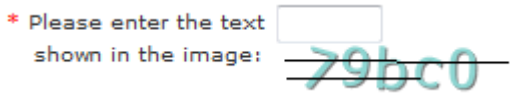

Figure 1. A CAPTCHA field example

This CAPTCHA technology comes with some consequences that tend to manifest as weak points of the entire software application that uses this kind of protection. Due to the fact that there are some web-bots capable of extracting the useful information from such an image, the generated images tend to be more and more complex. Sometimes, even a human operator has problems in reading the embedded active content, and this tends to become an inconvenient. In addition, if there is a web site that has several areas where this protection is needed, an eventual rightful user is very likely to become annoyed by the necessity of filling in all those CAPTCHA texts. In time, this can reduce the number of users that access the application.

This paper will furthermore present an alternate technology that is supposed to be able to distinguish the web-bots from the human operators without using the CAPTCHA fields. In order to do that, the approach will consider the vulnerable areas that are susceptible of being attacked, the behavior of the web-bots and the proposed solution.

3. Web-bots generic presentation

A web-bot is a software application able to perform automatically a series of I/O operations with a remote machine, usually the host that supplies some content that is supposed to be accessed via a web browser. The automatic access of a web server consists in reading or writing data from/to it.

The input operations performed by the web-bots can be harmless and a suggestive example would consist in retrieving automatically the weather forecast for some specific geographic areas. But not all the web displayed information is supposed to be stored by the client machines and sometimes, depositing onto the local machine some server side data can be regarded as stealing. The unauthorized saving of the content of a virtual discussion room can be regarded as a privacy attack. And the examples could continue.

On the other hand, sending some data to the server in an automated way could also be interpreted as a possible attack. If a web-bot fills in an infinite loop the e-form that is supposed to create user accounts, the server's database will be filled with useless data and the reliability of the entire application is seriously compromised. There can be said that the malevolent actions of the web-bots that send data to the web servers are practically unlimited.

Both operation types (input and output) that can be performed by the web-bots are possible only after performing some preliminary steps. First of all, the programmer that will implement the web-bot needs to analyze the context in which his product will operate. Due to the fact that the I/O actions have as target a web page, knowing its coding parameters is absolutely necessary for implementing a valid web-bot. Since all the web browsers have the option of displaying the code that produces the output seen on the screen, the analyzing process is highly facilitated.

After examining the code that will be accessed by the web-bot, the programmer needs to isolate the interest areas from the entire page code. There can be affirmed that both the input and the output operations will rely on some specific I/O fields. Once identified (considering the name or the ID of the targeted HTML entities) the web-bot implementing process can be started. Once finalized, the web-bot can be launched in order to read or write data from or to the targeted web server.

The suggested protection methods will defend the web distributed applications against automatic output operations. This will incapacitate the web-bots that send data to the web servers by automatically filling in the e-forms meant to be accessed by the rightful human users.

4. The attacking possibilities available for a data sending web-bot

A web-bot created for sending data to the distant machine will always make use of the available GUI (Graphical User Interface) provided by the server. As stated before, by analyzing the HTML code of the target web page, the attacker will be able to identify the exact fields of interest that will be used during his web-bot's runtime.

The consecrated manner that is meant to allow a user to submit some data to the server is to make use of the e-forms. The HTML tag that encapsulates an e-form usually looks as it follows:

<form id="f_id" enctype="encoding_type" action="target_page"
method="method_name">
<!-- form fields -->
</form>

The terms presented in the syntax above enclosed in quotation marks are detailed below:

- *"f_id"* is composed by some alpha-numeric characters and represents the e-forms unique identifier;
- "*encoding_type"* denotes the way the data to be sent to the server is packed. The possible values are *"multipart/form-data"*, *"application/x-www-form-urlencoded"* or *"text/plain"*.
- "*target_page"* represents the script that will receive and process the data sent from this specific e-form.
- *"method_name"* stands for the HTTP method used in order to send the data to the target page. It can have one of the following values: GET, POST, DELETE, HEAD, etc.

All the fields that are enclosed inside the e-form (represented above with the HTML comment syntax *<!-- form fields -->*) will have their values sent to the server by attaching them to their specific names and identifiers. As a simple example, let's consider a text field named *"text1"* that has the predefined value *"one"*:

<input type="text" name="text1" id="text1" value="one">

There are several input fields types that can be used inside a web e-form (text fields, text areas, password zones, single or multiple selection fields (radio buttons, checkboxes, drop-down lists), file selection fields, push buttons, etc.), but their implementation is not the aim of this article. What's important is that regardless of the e-form field type and behavior, the appropriate data will be sent to the target page in *[name, value]* pairs, using the selected HTTP sending method.

To summarize, in order to be able to start implementing his work, the programmer of an attacking web-bot has to know the target page where the data is to be submitted, the method used for packing and sending an e-form's data and the field types, names and their IDs. After that, all the fields can be filled automatically and the web server that supports the application can be put on a lot of stress.

The main acting way preferred by web-bots is composed by the following steps. It:
- impersonates a web browser;
- loads the targeted page;
- fills in the desired fields located in the target e-forms;
- simulates a press on the e-form's submit button;

The web-bots implementing technologies also have some limitations. What a web-bot cannot do is to add some new code to the original web page. For instance, it cannot feed into the targeted web-page some new functions that add a series of data processing capabilities to the original JavaScript code. A web-bot's one and only access gate is the web code delivered by the server for being displayed inside a web browser. The web-bot's actions depend strongly on the information obtained after analyzing the target

web page so the attacker that implements such a robot has to use the web page's source code the best he can.

The statements above describe the starting point of the proposed anti-web-bots protection method. The technique is based exactly on the impossibility of adding new code in the existent web pages.

5. A non-CAPTCHA protection method against data sending web-bots

The proposed solution that is to be described as it follows is based on a series of AJAX (Asynchronous JavaScript And XML) [9] specific technologies. No additional coding needs to be done on the server side of the application and the entire responsibility is delegated to the code loaded onto the client machines during web browsing. To be more exact, by making use of the event handling mechanism that is supported by all the web browsers, the web page code itself can decide whether the content has been filled automatically by a web-bot or in a natural way, by a human operator. This approach needs to be carefully implemented, because all the web code interpreted and displayed by a web browser is very exposed to un-authorized "peeking". Practically, the code that controls a web page's visible content is as public as a code can be.

The JavaScript code lines are client specific so they have to be embedded in the web pages delivered by the server to the client machines. It means that this code is also visible for anybody who wants to take a look to the implementation.

Embedding a web page's protection in the JavaScript code zone is extremely risky, because an intruder needs to be stopped in his actions with a series of techniques that are totally available for him. His malevolent actions are to be deployed in a zone with all the security measures exposed.

To summarize, the proposed solution gains its strength not from its secrecy but from the advantages that result from its very conceptualization and implementation.

All the GUI components that are to be found inside a web page can monitor the user actions performed upon them. There is a series of methods that react to every mouse move, key press, content change, etc. [10]. All the actions that take place inside a web browser on the client machine trigger a series of specific event. The default pre-implemented mechanism leave the web page's behavior in its natural state, as represented in Figure 2.

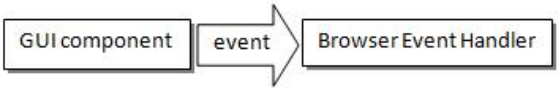

Figure 2. A web browser's default event handling mechanism

Every GUI component behaves normally, as it should. For example, any key typing performed inside a text box will result in inserting the appropriate character at the cursor's position.

But the programmer that implements the web page's behavior can intercalate his own code that is to be executed when a certain event occurs. The default event handling mechanism is suppressed and everything is filtered by an additional code that is capable of taking decisions. The event can be processed as desired by adding some extra functionality to it and even by allowing it to propagate only if a series of conditions are met (Figure 3).

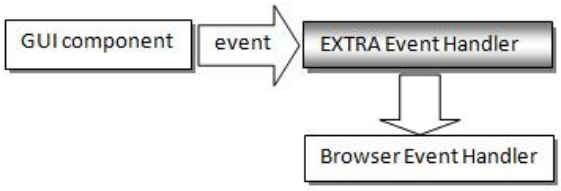

Figure 3. The insertion of an extra event handler

This way, the web programmer is in total control of the behavior of the GUI components inserted in the e-forms. There can be said that all the actions performed by an accessing entity can be monitored by the web page's code. The decision of letting go or of blocking some event can modify the web page's behavior accordingly to the actions of the entity (human operator or web-bot) that accesses the web page.

Considering the aspects mentioned above, the proposed solution would consist in allowing an e-form to submit its data only if the JavaScript code that controls the page decides that the information was entered by a human person and not by an automatic external software process (a web-bot). In order to accomplish that, there has been established a behavioral pattern of the human operator. Any difference between the desired behavior and any other detected filling way will result in rejecting the submission of the e-form's data.

The desired behavioral pattern concerns the generic characteristics that are to be found at any field type that can be found inside an e-form. The effect consists in the possibility of reusing the protection code for all the fields, regardless the e-form's length or specific content.

A web-bot can inject data inside any field located inside an e-form. The difference between this and the filling way of a human operator resides in focusing the GUI components. No human operator will ever be able to change the value of a field without setting first the focus on the component. No matter what method the human operator uses in order to access a specific field (using the mouse (by clicking the component) or the keyboard (with tabbed navigation)), the component will first receive the focus and only after that its content will be changed. On the other hand, the web-bot is technologically unable to set the focus to a certain component. This is exactly how the JavaScript code that monitors a web page can decide whether a certain e-form has been filled automatically or manually. Figure 4 synthesizes the aspects presented above.

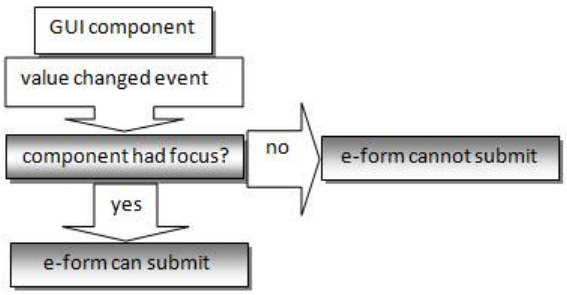

Figure 4. The principle of detecting the human operator's actions

In order to be able to complete the mechanism described above, the web page that implements this mechanism has to define a series of specific JavaScript functions for every component in the GUI that will react to the *OnFocus* and *OnChange* DOM (Document Object Model) events. The default e-form submission mechanism is also replaced by supressing the *OnSubmit* event generated by the e-form and by replacing it with a specific JavaScript code that will submit the data filled in the e-form only if all the checks performed upon the web page are completed successfully. In this case, the code that controls the web page can conclude that the pattern of human behavior has been successfully tested and no web-bot presence has been detected.

Considering the mechanism described before, the web-bots are stopped in their attempt of sending the automatically filled data using the e-forms loaded from the server machine via the web pages.

6. Conclusions

Using the proposed method, the web distributed software applications are enhanced from the security point of view. The web-bots that might try to abuse the online e-forms for sending to the server some large (and possibly un-wanted) amounts of data are blocked and their actions are transformed into impossible operations.

During the tests, the suggested protection method was proven to be successful. A test web page containing an e-form was offered as target for a series of already developed web-bots that could communicate successfully with some third parties web pages. All the robots were stopped in their attempt of simulating the human presence by filling automatically the e-form's fields.

The main advantage of the described solution is that the protected web pages require no additional actions or skills from the users that access them. Also, the content displayed by the client machines is strictly related to the useful data and is therefore kept in its original form.

The concurrent solution to the matter of restricting the web-bots access (the CAPTCHA technology) has evolved very much since it was first started and the complexity of the texts that the users must recognize can sometimes raise some serious problems. Also, if the protection is required several times during an activity session performed in a web distributed environment, the user could become slightly disturbed in his actions. All of these disadvantages of the CAPTCHA technology are eliminated by the proposed solution.

The main setback of the approach is that all the protection is performed in a highly exposed context, since all the JavaScript modules that perform the necessary validations are embedded into the client displayed web pages. But even if the attacker knows the exact protection mechanism, he is technologically limited in the attempt of implementing a working web-bot because of the intrinsic limitations manifested by the artificial form filling processes.

As the technology will evolve, the behavioral pattern of the human operator will have to be enhanced because the web-bots will probably be able to simulate more and more actions in a targeted web page. In any case, using the Web 3.0 AJAX facilities (for the distant machines) combined with a series of strong server stored software modules, the web pages are and will be able to perform all necessary validations in order to block almost any attacking possibility.

7. Acknowledgements

The research activity is supported by PN2 IDEI project, no. 1083 from Romanian National Authority for Scientific Research. (2007-2010).

8. References

[1] Cosmin STRILETCHI, "WEB_MATRIX - A secured framework for developing business-to-client web applications", 11th IEEE International Conference on Intelligent Engineering Systems INES 2007, 29 June - 1 July 2007, Budapest, Hungary

[2] http://en.wikipedia.org/wiki/Web_3.0 [01/05/2009]

[3] http://www.mozilla-europe.org/en/firefox/ [01/12/2009]

[4] http://www.microsoft.com/windows/products/winfamily/ie/default.mspx [01/12/2009]

[5] http://www.opera.com/developer/ [01/12/2009]

[6] http://www.apple.com/safari/ [01/13/2009]

[7] http://en.wikipedia.org/wiki/Internet_bot [12/12/2008]

[8] http://ro.wikipedia.org/wiki/CAPTCHA [12/12/2008]

[9] http://www.w3schools.com/Ajax/Default.Asp [10/08/2008]

[10] http://www.w3schools.com/jsref/jsref_events.asp [06/15/2008]