International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

A Secure Multilayer Honeypot in an E-Commerce Web Application

Gaurav Beriwal[1], Anuj Garg[2], Ravinder Jangra[3]

[1]Department of Computer Science, U.I.E.T. M.D. University, Rohtak, India

[2]Department of Computer Science, U.I.E.T. M.D. University, Rohtak, India

[3]Department of Computer Science, U.I.E.T. M.D. University, Rohtak, India

Abstract: ***E-commerce web applications are on a verge of not providing fair chance to all the consumers. E-commerce can be unfair especially in case of the check-out process as when many business trading corp. are vying for the limited supply item. Web applications security is more of a continuous plight as hackers and crackers are busy being creative avoiding/bypassing the many defensive tools to regulate security. The actuality is that the e-commerce application security is breached when some unethical corp. apply pre-formatted spiders or scripts to place orders. This gives them a very unjust advantage. Thus to rule out the problem i.e. to eliminate spiders/scripts in web applications by using a solution which is impractical to crack with no extra actions by the end user, this paper introduces an very innovative multilayer access to honeypots. This way is technically non-practical to crash or bypass proving secured web application forms.***

Keywords: Honeypot, e-Commerce, Security, Threat, Web Applications, Eliminate Spiders, CSS.

1. Introduction

In computer terminology, a honeypot is a trap set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems. Generally, a honeypot consists of a computer, data, or a network site that appears to be part of a network, but is actually isolated and monitored, and which seems to contain information or a resource of value to attackers. This is similar to the police baiting a criminal and then conducting undercover surveillance. Honeypots are run to gather information about the motives and tactics of the Blackhat hacking community targeting different networks. These honeypots do not add direct value to a specific organization; instead, they are used to research the threats organizations face and to learn how to better protect against those threats. Research honeypots are complex to deploy and maintain, capture extensive information, and are used primarily by research, military, or government organizations. If an attacker attempts to hack an application it is taken away by a honeypot instead, then information regarding IP address of the hacker can be traced. This can be further used to know the source of the attacker.

This paper represents the basis of the existent security solutions and will present a proposed solution. This is due to the ineffectiveness or limitation of the methods to remove malicious spiders and scripts, and to cease bots. Our paper explains the prescribed computer architecture for the proposed methodological analysis. This paper also describes the solution implementation and its effectiveness and necessity in the modern-era.

2. Form Honeypot

A present solution is using Form Honeypots. It is based on a concept in which a fixed, one or multiple invisible fields are present substituting as a honeypot which are dwelt by the spider, and the server logic is only capable to identify the spider using the backend and back-checking the value of those fields.

This existing honeypot is not giving a good amount of protection from the bots with brute forces, etc. This honeypot solution seems a good value of investment in reducing the hack attempts but this solution is fiddling and is prone to be hacked given the experienced hackers and advances in scanner codes. The hackers simply are able to bring down this method of one-dimensional honeypot approach as they forge a simple analysis of request/response of a valid form to recognize the expected fields by the server. Forging this hit and try method, the security of honeypots can be well-off defused therefore not providing the essential security against threats.

3. Proposed Solution

The basis of thought of this proposed solution is to create a two-dimensional honeypot solution. The very basic idea is a special path of differentiating between actual human involvement and a pseudo intervention. This differentiation would be done on a realm of computer technology which requires utilization of network connections and performance of various tasks using automated systems. This proposed solution comprises of the concept to limit the bot from recognizing a honeypot. This analytical method requires actual human involvement as the automated agent or bot would fall for the honeypots as it won't be able to recognize the valid form and the corresponding fields to it.

By using this two-dimensional method of honeypots the spiders forming a library of fields would be reckoning impossible as each and every field and form will have a different ID every time the page is reloaded.

3.1 Working

The proposed solution is shown in figure 1. Here we change

the previous concept of honeypot take it to the next level making it impractical to hack.

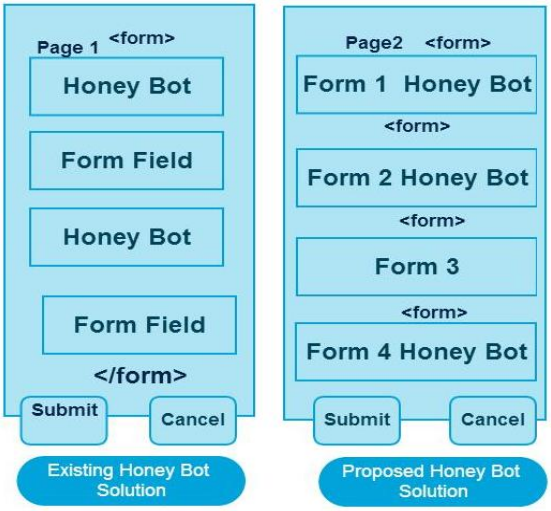
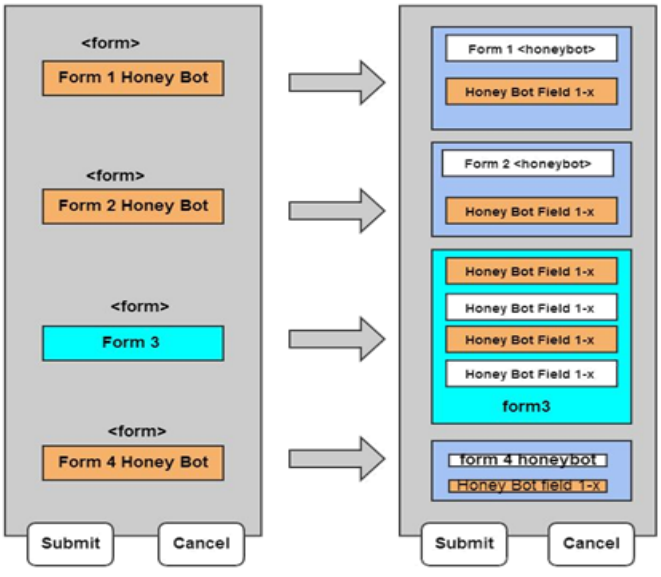


Figure 1: Existing vs. Proposed solution

In a web application, a) Every page on generation has multiple copies of the form, with slight differences of unique identifying hash, timestamp and many more; b) All forms are hidden through CSS; c) Once the page is submitted, a JavaScript application function retrieve the data from the server checking for the valid one; d) The rendered valid form is made visible altering its CSS.

It is undetectable by a spider or scanner as upon rendering of the page, the number and order of honeypots and the forms respectively are ever going to change. Observing the figure 2, we have two layer honeypot. There are many number of honeypot forms with the same field number and type as in the original form, the field names here are added randomly so as to make sure that the spider or bot is unable to save the valid field name to reassign it. Then further the server can only differentiate between the field names as to which is original human intervened and another identifying as a bot field or form making it very effective.



Two Dimensional Honey Bot Details

Figure 2: Solution Details

4. Solution Architecture

This type of architecture, as shown in figure 3, requires two main engines described below:-
a) Form Builder Engine
b) Form Manager Engine

Form Builder Engine is used to build similar honeypot forms as to the original one. Form Manager Engine is utilized for the management process and to add randomness in the forms and fields. It maps the forms and fields and stores the relevant information in the database so as when the form is sent through the validation engine it is able to decode using form/field ID.

This process is done to facilitate the identification of the original form and it"s representing fields. This secure system is forged by shuffling the original and honeypot forms altogether in a random order. It is second dimensional as the original form itself is shuffled in honey pot fields. Each and every form and field is given a unique ID. When the page is reloaded, the order of IDs is always changed randomly.

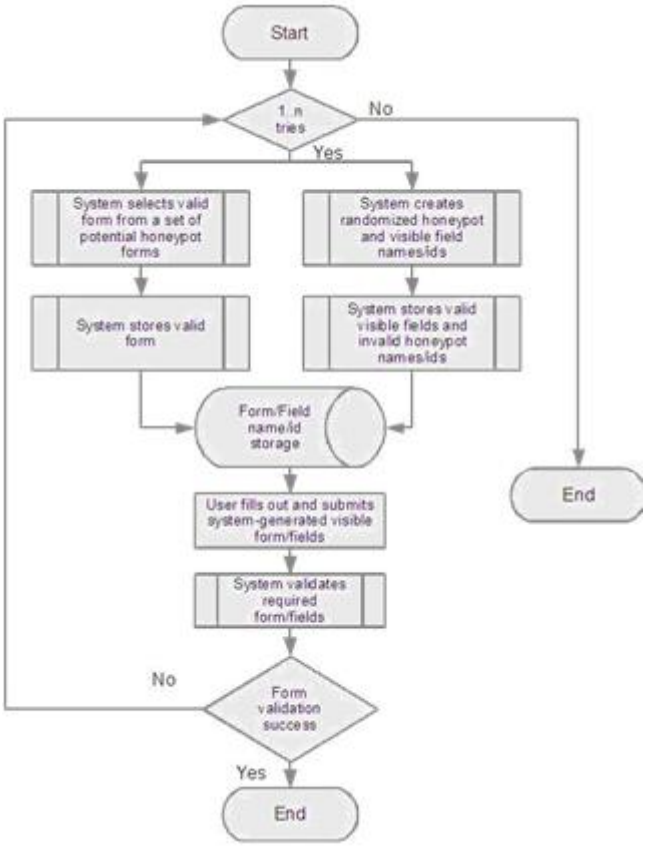


Figure 3: Solution Architecture

4.1 Solution Implementation

This can be a very interesting idea to implement as it is almost impractical to hack as the spiders are ever going to be confused between an original and a honeypot. If field Type attribute is used as hidden then it may give a very clear indication to the bot of what not to fill which ultimately will leave it unsecured defeating the sole purpose of security.

Syntax

```
<input type="value" />
```

Attribute Values

Value	Description
button	Defines a clickable button (mostly used with a JavaScript to activate a script)
checkbox	Defines a checkbox
file	Defines an input field and a "Browse..." button, for file uploads
hidden	Defines a hidden input field
image	Defines an image as a submit button
password	Defines a password field. The characters in this field are masked
radio	Defines a radio button
reset	Defines a reset button. A reset button resets all form fields to their initial values
submit	Defines a submit button. A submit button sends form data to a server
text	Defines a one-line input field that a user can enter text into. Default width is 20 characters

Figure 4: Field Type Attributes

Instead we should use manipulating Cascaded Style Sheets (CSS), so as the spider or scanner is unable to build a serial history of the events that took place making it largely secure. Using CSS manipulation and display attribute, the user end, if it‟s a human will only be able to fill the fields coded with „display: block‟ and won‟t be able to see the fields coded „display: none‟.

But the trick is that the bot will see both. Making it practical to fall for it and we can be notified of it and can trace it back. As in the figure 4, only the bottom field is visible at user end but the automated script won‟t differentiate in this and will fill any field coded as type „text‟ giving an indication of and spider/bot filler of form.

```
<div id="honeypot-div" style="display:none">
<input type="text" name="honeypot" value="" />
</div>

<div id="Original-div" style="display:block">
<input type="text" name="original" value="" />
</div>
```

Figure 5: Simple honeypot example

The figure 5 depicts an old honeypot structure, the proposed two-dimensional honeypot structure includes: - a) Dynamic form ID and field ID; b) Display attributes usage within CSS file; c) Dynamic class ID within CSS. These above three items when together used makes it nearly impossible for the automated script or bot to detect a honeypot, instead is ever-ready to back-trace it.

Each and every form will have different IDs which are randomly assigned upon the loading of page. When the page is reloaded every form will have new dynamic name and IDs making it impractical for the spider to store its history of the page layout.

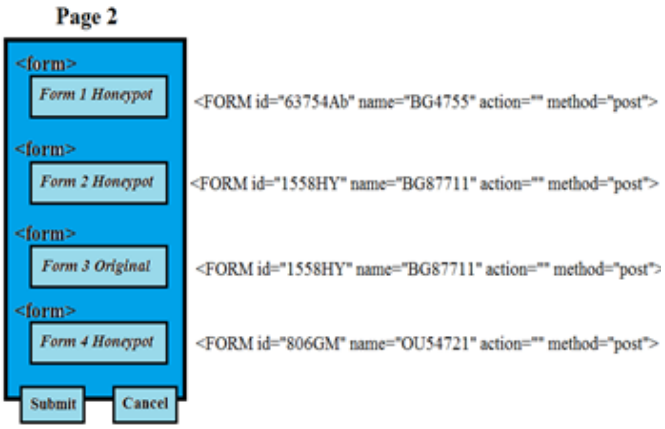


Figure 6: Sample form honeypot (first dimensional)

In the figure 6, there are four forms with different names out of which three are honeypots. It is second dimensional as the dynamics of randomness that applies on form IDs, further applies to the field, class and div IDs too; the total added randomness of these IDs ensures a very high level of security which is untraceable by the bots.

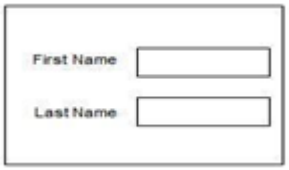


Figure 7: Two dimensional honeypot implementation as seen by the browser

```
<div id="02707" class="802711">
 <form id="02708" name="BG90009" action="action1" method="post">
  <input id="02710" type="text" name="First Name" class="802712" value="" />
  <input id="02711" type="text" name="Last Name" class="802713" value="" />
 <input id="02714" type="text" name="Meden Name" class="802715" value="" />
 <form>
</div>

<div id="11703" class="311707">
 <form id="11704" name="311705" action="action2" method="post">
  <input id="11706" type="text" name="First Name" class="311708" value="" />
  <input id="11707" type="text" name="Last Name" class="311709" value="" />
 <input id="11708" type="text" name="Meden Name" class="311711" value="" />

 <form>
</div>


<div id="01500" class="201504">
 <form id="201501" name="201502" action="action3" method="post">
  <input id="201503" type="text" name="First Name" class="201505" value="" />
  <input id="201504" type="text" name="Last Name" class="201506" value="" />
 <input id="02714" type="text" name="Meden Name" class="201507" value="" />

 <form>
</div>
```

Figure 8: Second dimensional honeypots with further field names

It has typical honeypot fields but is hidden from the front user-end and is shown to the bots only through CSS, as shown in figure 7. Its implementation would have a page with many similar forms covered by CSS in which the honeypots are hidden to the front end. The source of such type of page using two dimensional honeypot forms would have multiple identical form with only a single valid form. As in figure 8, the original form is ID 01500, but it further includes honeypots in the second dimensional with a field „meden name".

5. Solution Security

The major difference is that in this proposed solution most of the fields are honey pot fields, even those which were the original in a pre-request. Example, in the first request#1, the form#4 was the original one then on re-rendering of the page on the second request#2, the form#6 would be the original one. Statistically, the automated script/ spider have a very substantial chance of getting into the wrong fields. Rechecking this proposed solution, by passing it via a security scanning tool would create a „no error" remark under a scanner. It is since that the scanner is a type of smart form reader with complicated test policies, would not distinguish honey pots as a black-box scanner it is. Brute force attack, which is considered very effective by the hackers would also be impractical to use as the hacker won't be knowing the location of the valid form.

Even if in the rarest cases the attacker is able to distinguish the original form then since it is a two-dimensional security the hacker won't be able to differentiate within the honey pot fields in the form.

6. Conclusion

It will help web applications to have multiple honey pots with random form and field IDs making it extremely difficult for the bot to check the valid one as the IDs are randomly generated every time the page is reloaded disabling it to keep a history of events. Therefore allowing only the front-end user to see the right form and enter the right fields. A governance server module that maps the forms and fields with random IDs would only be able to differentiate between the valid fields and honeypots. In this proposed solution the bot/spiders should not supply with the incorrect solution and only a single correct one in field would be accepted. This focuses not only on security of web applications but as well as also tracing back the bot efficiently.

Its intent is to avail seamless work flow in which the user doesn't require to write any CAPTCHA word or field to prove that he is human, rather the protection from the automated scripts and spiders is catered in the back-scene. We try to make honeypots almost impossible to detect due to their complicated two-dimensional systems.

7. Acknowledgement

The authors wish to thank Ravi Shankar and Kartik Mudgal.

References

[1] IBM. Method and system to generate human knowledge based CAPTCHA. IP.com number: IPCOM000184977D, July, 2009
[2] K. Elissa, "Title of paper if known," unpublished. The Government of the Hong Kong Special Administrative Region, http://www.infosec.gov.hk/english/technical/files/honeypot s.pdf.
[3] Bronstein A,NAME, U.S. Patent, 7,841,940 issued, Nov 30, 2010

[4] Pratte et al., „Method And Apparatus Eor Network Authentication Oe Human Interaction And User Identity", U.S.
Patent, 20,080,216,163 issued, Sep 4, 2008.
[5] Osborn et al., „Graphical Image Authentication And Security System", U.S. Patent, 20,090,077,653 issued, March 19, 2009.
[6] Carter et al., „Method, System And Computer Program Product For Access Control", U.S. Patent, 20,070,124,595 issued, May 31, 2007.
[7] Mates J, „Generatinga Challenge Response Image Including A Recognizable Image", U.S. Patent, 20,090,313,694 issued, Dec 17, 2009.
[8] Carter et al., *'Method, System And Computer Program Product For Access Control'*, U.S. Patent, 20,070,124,595 issued, May 31, 2007.
[9] Mates J, *'Generatinga Challenge Response Image ncluding A Recognizable Image'*, U.S. Patent, 20,090,313,694 issued, Dec 17, 2009.