Method for Two Dimensional Honeypot in a Web Application

N. Nassar[1], G. Miller[2]
[1]IBM, Somers, NY, US nnassar@us.ibm.com
[2]IBM, Arvada, CO, US millerg@us.ibm.com

***Abstract*—Web applications Security is an ongoing dilemma as hackers and bots are getting more and more innovative bypassing the various defensive tools implemented to enforce security. e-Commerce Applications, such as those used for the check-out process, could be in a position of not providing a fair chance to all consumers. This is especially true when a commerce site offers hot inventory items where many traders are competing to get a limited supply item. What happens is the e-Commerce sites security is compromised when some of the traders utilize pre-formatted scripts/ spiders to place orders, thus giving them an unfair advantage The problem is: how to eliminate scripts/spiders in a given web application flow by using a solution that is non-practical to crack with no additional actions taken by the end user. Our paper introduces an innovative multilayer approach to honeypots cashing or bypassing it is technically impractical, resulting in well secured web forms.**

Keywords-Honeypot; e-Commerce; Security; Vulnerability; web Applications

I. INTRODUCTION

A honeypot is a trap set to detect, deflect, or in some manner counteract attempts at unauthorized use of information systems. Generally, it 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 [1]. A honeypot works by fooling attackers into believing it is a legitimate system; they attack the system without knowing that they are being observed covertly. When an attacker attempts to compromise a honeypot, attack-related information, such as the IP address of the attacker will be collected. This activity done by the attacker provides valuable information and analysis on attacking techniques, allowing system administrators to "trace back" to the source of attack if required.

In this paper we will provide a background about existing security solutions that is been used in the enterprise to limit bots and malicious scripts, and the limitation of these tools. We will introduce our proposed solution and illustrate its effectiveness. We will also describe the recommended architecture for the proposed methodology. Finally, we will demonstrate various embodiments to our proposed solution

.

II. BACKGROUND

There are many known methods to protect e-Commerce / flow based websites from bots [2][3][5]. Bots are used to gain access to shopping carts to quickly purchase limited items, such as concert tickets, before a human shopper can do so. All solutions to prevent bots are geared toward identifying who is submitting the requests: a machine or a human. Some of existing solutions are,

A. CAPTCHA

Completely Automated Turing test to tell Computer from Human Apart. This technique is based on providing a challenge in the form of distorted image of letters and numbers used to prevent automated use of websites. CAPTCHA solutions (comes in a verity of flavors [4]) requires a person to read the distorted letters and type them into a field, something a bot cannot do. This proves that the page is being accessed by a person. Figure 1 shows examples of CAPTCHA.

Figure 1. CAPTCHA

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States

DOI 10.4108/icst.collaboratecom.2012.250743

The existing solution counts on generating a combination of letters and numbers that are distorted and displayed as an image a person must interpret and re-type into a field. The approach can be problematic for users, as they cannot always read the letters or numbers because of too much distortion. The main drawback is that hackers developed smart spiders where it builds a library of images that would allow figuring out the content of the CAPTCHA.

B. *Form Honeypot*

Another existing solution is using Form honeypots. However, this solution is trivial and easy to crack given the sophisticated form of scanners and experienced hackers. Form honeypot is based on providing a 'fixed' one or multiple invisible fields serving as a honeypot where they get populated by the spider then using the backend, server logic would be capable of identifying the spider made request by looking the value of these fields(s) [16].

While using a honeypot trap sounds promising and interesting, advanced code scanners and experience hackers easily crack this single dimensioned honeypot approach by creating a simple analysis of request/response of a given form to identify the proper fields expected by the server. Using try and error, honeypots could be easily identified and defused. As a result, the key issue is that existing honeypot solution is not providing the right/ expected protection from bots [15].

III. PROPOSED SOLUTION

The idea of this concept is to provide a unique way of discriminating human involvement in a transaction that would occur on a computerized experience that utilizes internet or network connected interactions that require an endurance of actual human intervention and not a "pseudo intervention" that could be performed by an automated system or application in the realm of computer technology. We propose using two dimensional honeypot for security access which eliminates the vitality of form scanning automation so that the spider will not be able to identify which form and which field is the honeypot. Our proposal revolves around limiting the bot's ability to determine the honeypot Based on the implementation of our proposed solution, spiders building a library of fields will be computationally impossible as each fields and each form are different every time the page is loaded.

This methodology requires human interaction because any script or automated agent will not be able to identify the 'valid' form and corresponding fields. Instead, it will fall into the honeypots, figure 2.

A. *How it works*

Our proposed solutions, as shown in figure1, take the concept of honeypot, but go further with it. In a given web application,

- Each page upon generation includes multiple copies of the form, with minor differences (timestamp is different, a unique identifying hash, etc).
- Initially all forms are 'hidden' via CSS[1].
- Upon rendering the page, a JavaScript function performs a callback to the server, asking which form is the valid one.
- The valid form's CSS is modified to make the form visible.

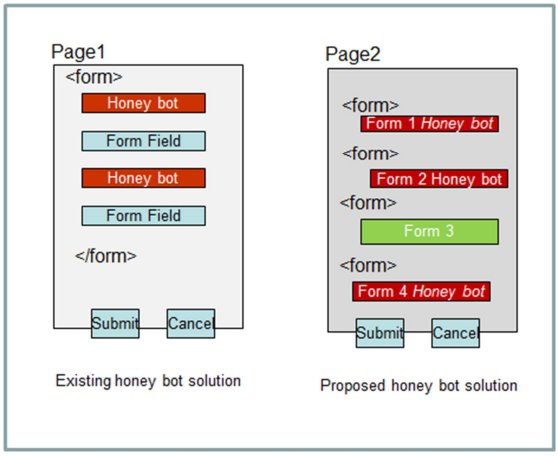

Figure 2. Existing vs. Proposed solution

When the page is rendered, the order of the forms and the number of honeypot forms will always change so that it is never detectable by a screen reader or a spider.

Looking inside page 2, as seen in figure 3, below, we will find that there will be n number of honeypot forms and each form would have the same field number and field type as the original form. The only different is that all of these fields represent the second dimension of our proposed two layer honeypot. In addition, field names are generated with added randomness so that the spider wouldn't be able to store the actually / valid field name to re play it. This technique will be orchestrated by a server chirographer layer that decipher the field names and understand which form and which field is the bot.

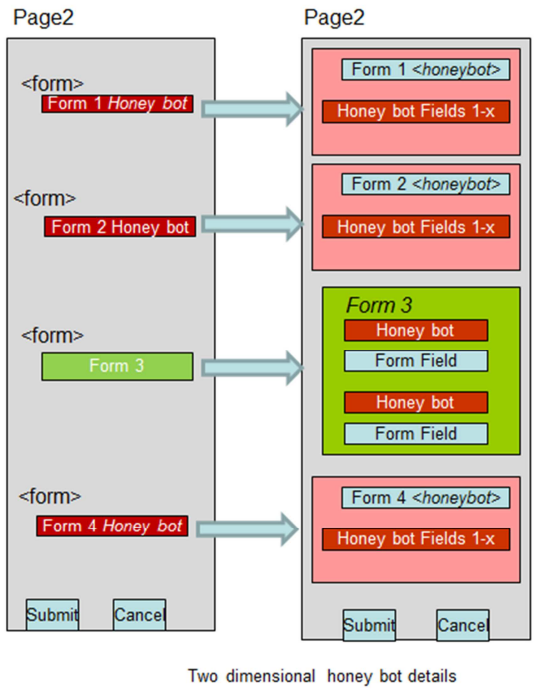

Figure 3. Solution details

A. *Solution Architecture*

As shown in figure 4, the system to provide such a solution consists of two main engines. A Form Builder Engine is to build honeypot forms similar to the original form. A Form Manager Engine is targeting the randomization and the management process.

The two dimensional protection is created by shuffling the honeypot forms with the original form in a random order, so that when the page is refreshed, it is never the same order any more. The second dimension is shuffling honeypot fields with in the original form itself. Naming each field and each form with in each form is a key to create the right level of protection.

The Form Manager Engine maps which form is which and which field is which and stores this relationship in the database so that when the form is sent back from the browser, validation engine would be able to decode form and field IDs in order to identify the original form and its corresponding fields.

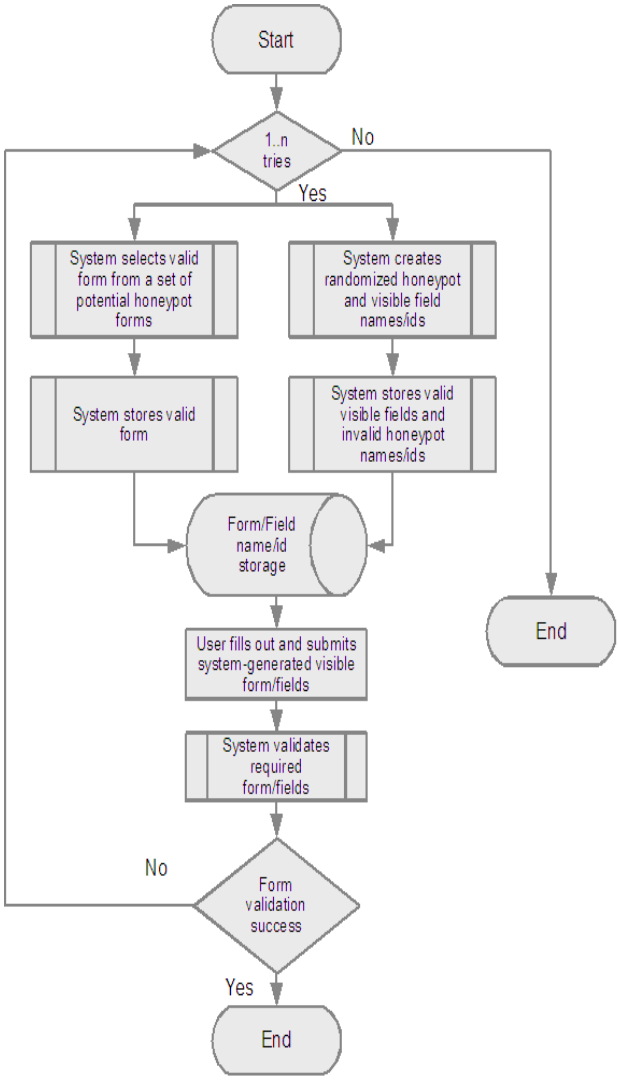

Figure 4. Solution Architecture

B. *Solution Implementation*

Implementing this idea could be very interesting as spiders need to be fully in the dark of what is original and what a honeypot is. For example, using field Type attribute, figure 5, as hidden may be a very easy indication for the automated script to identify what not to fill which may defeat the purpose.

The better approach in this case would be manipulating cascaded style sheets, CSS, files in such a way the screen reader/ spider would be unable to build a history of events which make it as secure as intended. With CSS manipulation, and using display attribute, the end user will be able to fill only the

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 5, Field Type Attributes [1]

form/fields marked with display(block) while sections with display (none) will be hidden the user, but visible to the bot. As seen in figure 5, the top div will be hidden from the end user while the bottom one will be visible. However, the script would fill up any field marked with type 'text' and that could indicate spider/auto form filler.

While figure 5 illustrates a simple honeypot structure, our proposed two dimensional honeypot includes

- Dynamic Form Id/name,
- Dynamic field Id/name
- Use of display attributes with in a CSS file
- Dynamic class Id with in CSS.

The combination of these four items will make it almost impossible for the spider to detect a honeypot, and be ready to detect it.

```
<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

As described in the solution architecture, each form will have an Id that is dynamically assigned every time the page is rendered, figure 6. While there are four forms with random names in page 2, below, three of these forms are meant to be honeypots.

If the page is refreshed, the page is re rendered with different number of forms, and each form will have a different/randomized Id and name attributes. In this case any spider or screen reader will have no chance to keep history of the html layout and the page behavior.

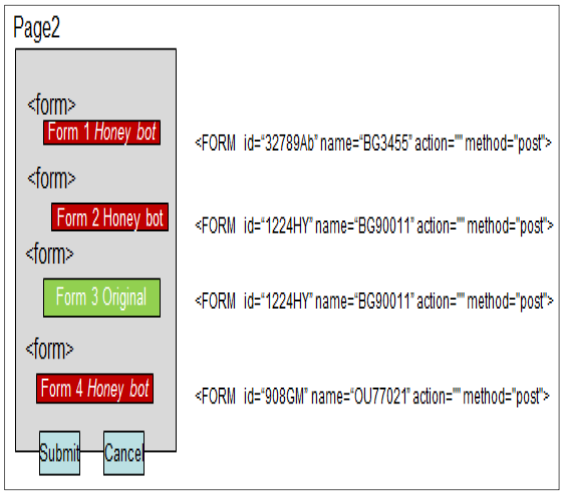

Figure 6. Sample form honeypot (first dimensional)

The second dimensional to this proposal is implemented within the form itself which is a typical honeypot fields, but hidden from the front end user via CSS. The same dynamics applied in the form Id, applies for the field Ids, div Ids, and class Id. The random generation of these Ids grants a high level of security which is untraceable / undetectable by spiders. As illustrated in figure7b, below, the implementation of one page according to our approach would result in a random number similarly generated forms wrapped by CSS governed divs, but the front end would reflect no signs of the honeypot as shown in figure 7a.

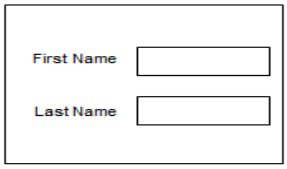

Figure 7a, Two dimensional honeypot implementation as seen by the browser

[1] http://www.w3.org/Style/CSS/

```
<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 7b. Two dimensional honeypot implementation

While the source of the page shows the multiple identical forms. In figure 7b, form Id 01500 is the original form, but within that form itself, the second dimensional honeypot resides which is "*Maiden Name*" field.

C. *Solution Security*

The key difference here is that in this process all fields are potentially honeypot fields, even ones that were legit in a prior request cycle.

For example, in Request #1, Form #8 was valid, but if the user reloads the page generating Request #2, Form #3 would be the valid one. Unless the bot process actually runs the JavaScript to determine the correct field, it will have a statistically significant chance of entering the wrong fields. For a blanket-fill bot, that chance is practically 100%.

By the same token, the use of brute force will also not be so practical since the hacker doesn't know which form is the original one. Even if the attacker was lucky enough to find the right form, which his highly unlikely, then the second dimensional would provide the needed security to the form and the attacker will not be able to distinguish the honeypot field within the form.

Running this solution through a security scanning tool would generate no errors as the scanner 'which is nothing more than a smart form reader with sophisticated testing policies' [1] would not identify any of the forms as honeypots since it is a black box scanner[2].

IV. FUTURE WORK

We are planning to implement this system and prototype as a secure e-Commerce application and the plan is to test it against various bots. Usually bots will try to fill each and every field in a given form then execute the associated form action. Given the nature of the proposal, Application vulnerability scanners are not expected to pass the pages protected by our solution. We are planning to publish the experimental results of this study in a separate paper and highlight the reaction of various scanners toward our implementation.

Also, our future work will include confidence based security where the level and the type of security in a web application are adjustable based on the level confidence or the type of credentials the user exhibits including context and location awareness. This work is coupled with runtime security architecture to provide dynamic challenges to the user based on the user behavior.

V. CONCLUSION

In this paper, we introduced the advantages of using multi-layer honeypot over known solution.
The intent of this methodology is to provide a seamless work flow where the end user is not asked to write any CAPTCHA fields, instead, bot detection and protection is provided behind the scene. Our proposal is making honeypots virtually impossible to detect because of the comprehensive architecture of the honeypots which made any script unable to detect the honeypot.

The intent of this methodology will enable web application pages to have multiple honeypot forms each has random Id. Each field in each form has a randomly generated and unique id which makes it extremely difficult for a bot to query and store correct form Id or fields Id because simply the correct form Id and field Ids are regenerated every time the page is loaded. Thus only front-end user would be able to enter the right form and its fields.

When generating the forms, including the honeypot, generate fully randomized field names for all fields in all forms. A governance module on the server that list and map which random field names are valid, and what real fields they mapped to. This architecture represents a 2-layer honeypot, meaning the bot would have to supply "only" the correct.

---

[1] http://samate.nist.gov/index.php/Web_Application_Vulnerability_Scanners.html

[2] Grendal-Scan, http://www.ehacking.net

.

This proposal is not focusing on the honeypot itself, but focusing on the bot's inability to determine the honeypot detection fields, both from a name and a no-invalid entries point of view.

REFERENCES

[1] Chew, M. et al. (2004).Image recognition CAPTCHAs. In Proceedings of the Information Security, Conference (ISC 2004), LNCS 3225,268-279.

[2] Ahn, L. et al.. Telling humans and computers apart automatically. Communications of The ACM, February,2004

[3] Elson, J. et al. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. ACM, CCS'07, October-November, 2007.

[4] IBM. Method and system to generate human knowledge based CAPTCHA. IP.com number: IPCOM000l84977D, July, 2009

[5] IBM, A new method for telling humans and computers apart automatically, http://ip.com/IPCOM/000188716, October, 2009.

[6] Chickering et al. , '*Image-Based Human Interactive Proofs*', U.S. Patent, 20,100,162,357 issued, June 24, 2010.

[7] Lamberton et al., '*System And Method For Sì/Èfiíelementing A Robot Proof Web Site'*, U.S. Patent, 20,080,209,217 issued, Sep 28,2008

[8] Bronstein A,NAME, U.S. Patent, 7,841,940 issued, Nov 30, 2010

[9] 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

[10] Osborn et al., '*Graphical Image Authentication And Security System*', U.S. Patent, 20,090,077,653 issued, March 19, 2009

[11] Carter et al., '*Method, System And Computer Program Product For Access Control*', U.S. Patent, 20,070,124,595 issued, May 31, 2007

[12] Mates J, '*Generatinga Challenge Response Image Including A Recognizable Image*', U.S. Patent, 20,090,313,694 issued, Dec 17, 2009

[13] Reshef et al., *'Method And System For Discriminating A Human Action From A Computerized Action'*, U.S. Patent, 20,050,114,705 issued, May 26, 2005

[14] Von Ahn, et. al. Proceeding UROCRYPT'03 Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques, 2003

[15] Sidiroglou Styliano, et al. ip.com, WO US2006/014704 , Systems and methods for detecting and inhibiting attacks using honeypots, http://ip.com/pat/WO2006113781A1, Oct 26, 2006

[16] The Government of the Hong Kong Special Administrative Region, http://www.infosec.gov.hk/english/technical/files/honeypots.pdf