Hovering Patterns: Clickjacking Defense Technique

Ahmed Anas, Sherif Khatab, Akram Salah
Computer Science Dept.
Faculty of Computers and information, Cairo University
Cairo, Egypt

Abstract
Clickjacking attacks is one of the evolving attacks that targets users web surfing integrity. Through the attack observation and analysis, we developed a new technique that enforces user awareness of sensitive UI actions he is about to perform. Proposed technique enforces user experience integrity by asking the user to interact with visual component through hovering over randomly generated points where a summary text of the critical action will be explicitly outlined.
The technique is protected by nature against Clickjacking attacks as it requires clicking to proceed with the required actions. The technique has trivial performance effect and can be integrated easily within a web widget.
Keywords:
Clickjacking, The random patterns, Anti-Clickjacking

Introduction

Applications can share some of their functionality allowing embedding themselves within other hosting applications. Embedding is a technique that allows a web application to display another web application’s functionality and features within the main web application pages. To achieve such goal, the original web application uses HTML attributes (e.g. Iframe). According to SmilarTech 28.04% of the top 10,000 websites in the world have some sort of Integration with Facebook Social Plugins[1] on their homepage. Also, a percentage of 20.9% of the websites have one of Twitter’s social tool,embedding is not limited to social media plugins, applications can share widgets for different reasons, such as open authentication widgets (OAuth), PayPal Widgets, etc. The desire of applications owners to increase the brand recall, engagement, time on their website, referrals and sales revenue by adding social media plugging expose their users to the risk of being targeted by Clickjacking attacks[2], [3].
Clickjacking is a technique used in order to deceive the user of the hosting applications to trigger unintentional actions on the embedded components or widgets.The attacker would achieve his goal using different techniques such as applying multiple transparent or opaque layers deceiving a user to click a button or link. One of the famous types of clickjacking attack is Like-jacking [4]; The attacker can create a webpage that has a Facebook like button with a hidden property. Along with overlying it with any other sincere button such as download button. Since the user can only see the download button, he may be deceived to click it generating fake likes or undesirable feeds to appear in user’s news feeds. Clickjacking attacks can be combined with cross site scripting, that may cause worm propagation [5] similar to the one that targeted MySpace in 2005 and millions of users affected.
Generally, one of the solutions of this type of attacks is using the frame buster techniques [6], which uses a JavaScript code to prevent the website from being rendered within IFrame. Another solution, the X-Frame-Options HTTP response header [7] that can control which hosting website are allowed to render the protected website in Iframes. The previous solutions are not fulfilling the business need of the application which intended to share its functionalities as widgets. Other solution introduced like NoScript, ClearClick and Firefox plug-in along with ClickIDS enhancement to let it work with anti-XSS techniques [8] and similar solutions which have a common issue of generating a user pop-up to ask the user confirming the action, and as most of the users are not aware of the deep technical details, this will not introduce them a real protection. Another solution introduced as HDTCV [9] that crawls the internet to build a database of websites which contain a Clickjacking attack. Unfortunately, such solution will not protect against newly crafted websites. Other solutions involve JavaScript disabling as in ClearClick or random generating Iframes introduce poor usability and incompatibility with existing web sites, or failure to defend against significant attack vectors.
Clickjacking undermines one of the two contexts of an object, or both of them at once for that matter. The first one is the visual context which typically means an object has been masqueraded as another baiting users into clicking like button as a download button for example. The second way is by manipulating the momentary along with physical properties (Temporal context). Thereby, baiting users to click an object, a like button for example, despite they target on clicking in different object at a certain time.
Our solution propose adding other visual components that guarantee that user will not be able to proceed with his intended action unless he has the visibility over the control in place. To achieve that, we need to ensure an HTML

Manuscript received February 5, 2018
Manuscript revised February 20, 2018

object containing a pattern which includes checkpoints being generated based on user interaction where user has to follow without single mouse click. Additionally, the panel area will show the third party reference identity.
This pattern can be configured to work in two different ways, first of them is to be totally randomized as shown in Figure 1.The other technique is to be persistent one registered with the widget site as a sort of authentication.

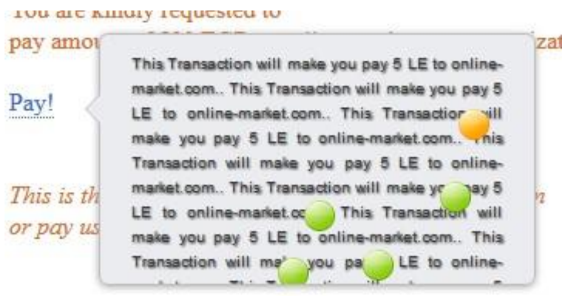
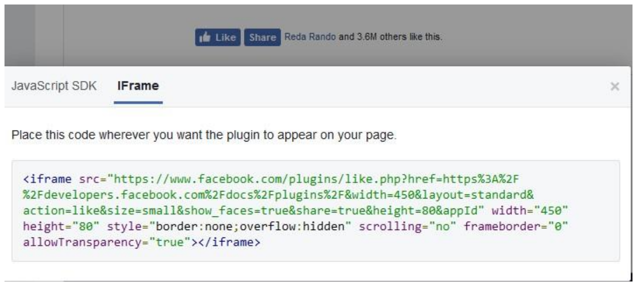

You are kindly requested to
pay amo
zat
Pay!
This is th
or pay us
This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-market.com.. This Transaction will make you pay 5 LE to online-

Fig. 1 An example of the proposed solution. A div area expands and contains an unpredictable pattern of checkpoints the user has to follow in order to make a money transaction for a website.

Applying this technique on widget controls will let the clickjacking attacks no longer visible, attempts to affect the user visibility of reference without previous knowledge of the predicted point location will result in user will not be able to make the action processed, and thereby will prevent the attack. The remainder of the paper is structured as follows: in section 2, background and detailed illustration of Clickjacking attack. In section 3, a detailed illustration of how our defense technique works, it includes the details regarding how the two patterns are going to defend against the attack. In section 4, the implementation section contains the details about the implementation along with security and performance analysis. In section 5,an overview of the previous studies related to our subject then the conclusion in section 6.

2. Background

When a browser renders an HTML page, it processes the page's Document Object Model (DOM) which contains nodes, widgets, div tags and event listeners associated with those components. In the HTML DOM, every HTML element is an object of a tree including the document itself, buttons, text boxes and text areas as well. Each DOM object has properties such as style object properties. [10]
In conjunction with HTML DOM objects, HTML DOM events allow JavaScript to register events handler on those objects. This event is usually assigned to fire pre-defined functions, HTML DOM click event and HTML DOM on-hover event would be considered as examples of the HTML DOM events
[10] There are some DOM elements that require attention such as the div tag that defines a division or a section in an HTML document [11] , Some DOM objects, such as IFrame, div, object and applet, are used to embed Widgets in an HTML page.
A widget [12] is a small application with a specific function (e.g., clock or a Like button). Figure 2 shows a like button, this like button code is typically an iFrame component rendered from third party website as a widget. Web site can embed widget that has been developed and published by third party



Place this code wherever you want the plugin to appear on your page.

```
<iframe src="https://www.facebook.com/plugins/like.php?href=https%3A%2F
%2Fdevelopers.facebook.com%2Fdocs%2Fplugins%2F&width=450&layout=standard&
action=like&size=small&show_faces=true&share=true&height=80&appId" width="450"
height="80" style="border:none;overflow:hidden" scrolling="no" frameborder="0"
allowTransparency="true"></iframe>
```

Fig. 2 Example of a like button widget embedded within hosting page.

and could be embedded within the hosting page as it is the original web-page that those widgets are embedded within. Applications which encourage embedding some of their components may be vulnerable to clickjacking [2]; clickjacking is a technique used in order to deceive the user of the hosted applications to fire unintentional actions within the embedded components or widgets; in order to do that, attacker would use multiple techniques for example, using multiple transparent or opaque layers to deceive a user into clicking on a button or link, as shown in Figure 3. The attacker may add the ”Opacity:0.001” style to like button in order to hide it and adding the ”z-index:-99” property ; attacker also may apply the ”pointer-events: none;” property in order to disable interaction with fake download button thunder image highlights where does the pointer action takes place.

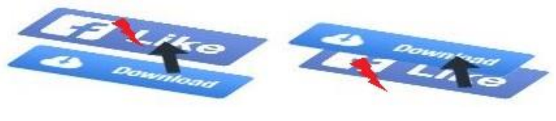


Fig. 3 Hiding Like button under Download button

One of the most common types of clickjacking attack is Like-jacking [4]; the attacker could craft a web page that has a hidden like button embedded within the web page deceiving users into clicking on a Facebook Like button by transparently overlaying it on top of legitimate UI element,

such as a download button. Hence, when the user tries to download the desired file, a feed appears in the user's Facebook friends' news feed stating that she likes the attacker web site.
Besides the basic clickjacking attacks, which only violates the visual integrity constraints, attackers could innovate different attack variants, for instance "Brain of Computer" demonstrate a technique called "Facebook Click Jacking with Floating Like Button" which is basically a JavaScript code that lets the like button sticks to cursor, and thereby, whenever the user perform a click, he will click the like button.
Another type of attacks which also based on cursor move called Cursor-jacking demonstrated by Bordi and Kotowicz, one of those attacks are using fake cursor plus hiding the real cursor using CSS styles deceiving the user to fire unintended hidden actions.

3. Proposed technique

In essence, our technique is based on using user interface verification control to ensure the integrity of actions. The technique comes with two flavors, the first one is related to generating a random pattern where user has to follow in order to proceed with the action Figure 2. The second one is by asking the user to draw a specific pattern, this pattern whom the user was already asked to register.
The random pattern: We enforce the widget to have an event listener (e.g., onhover event) which expands a div area that contains unpredictable pattern that consists of checkpoints along with background brief of the transaction you are about to acknowledge Figure 1. User has to follow those points in order to perform the desired action. Accordingly, the control would not allow the user to fire the action he desires by clicking the control itself.
As a high level, a verification process goes as follows when user hovers over sensitive action handler, message appears to user asking him to follow the pattern, a verification request message sent to server that accordingly prepare verification process creating a session object that carries array list of finite random points
Verification process goes as follows Figure 4 verification request message is sent to server with value "req", client side code receives (x,y) coordinates along with Len attribute, if len value equals zero, that means the server has successfully authenticated the user and performed the required action. Otherwise, it uses the (x,y) coordinated to draw the next point, and attach hover event as in Figure 5. When user hovers over point, we apply different style and then send point id to server that will verify the point id and send (x,y) coordinates along with Len. When len equal zero verification process ends, else, it draw next point attaching the same hover event to it.

Server side process in particular goes as the following Figure 6. When server receives verification request message, it creates a session object that carries an array list of finite random points. Then, it sends point attributes and checks whether it is the last point of the array when it receives the correct id.Thereby, authenticate the action and send Len flag with value zero. Also, there is a session timed out after specific time of inactivity to protect server memory. Figure 7 shows screenshots for the random pattern confirmation scenario from the user perspective.
The Persistent Pattern: We enforce the widget to have an event listener (e.g., onhover event) which expands a div area that contains a grid of checkpoints similar to pattern concept as used by the mobile users [49], User has to follow the points pattern in order to perform the desired action. This pattern will be already saved by the user. Accordingly, the control prevent the user to trigger the action by clicking the control itself. To implement this technique, there is no need to change neither

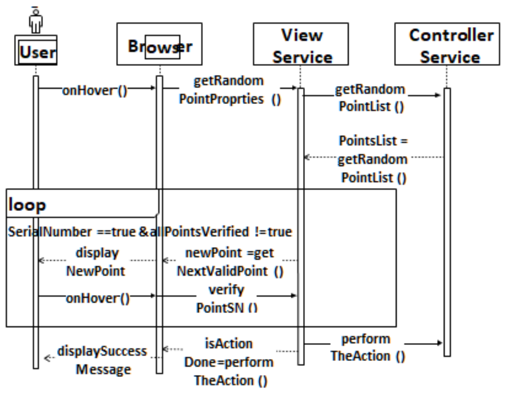

User
Browser
View Service
Controller Service
onHover()
getRandom PointProprties ()
getRandom PointList ()
PointsList = getRandom PointList ()
loop
SerialNumber ==true & allPointsVerified !=true
display NewPoint
newPoint =get NextValidPoint ()
onHover()
verify PointSN ()
displaySuccess Message
isAction Done=perform TheAction ()
perform TheAction ()

Fig. 4 Sequence Diagram to demonstrate the interaction between user, browser and server for random pattern technique

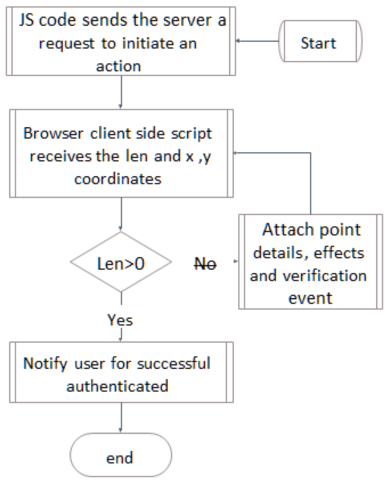

JS code sends the server a request to initiate an action
Start
Browser client side script receives the len and x ,y coordinates
Len>0
No
Attach point details, effects and verification event
Yes
Notify user for successful authenticated
end

Fig. 5 Flow Chart demonstrate the process of point properties receiving, rendering till action takes place

the browser behavior nor the http protocol. We need only to change the action listeners implemented for the third party widgets. For future deployment, our solution can be integrated with the browser.
While triggering a sensitive action, the user must be authenticated against the third party widget provider. When

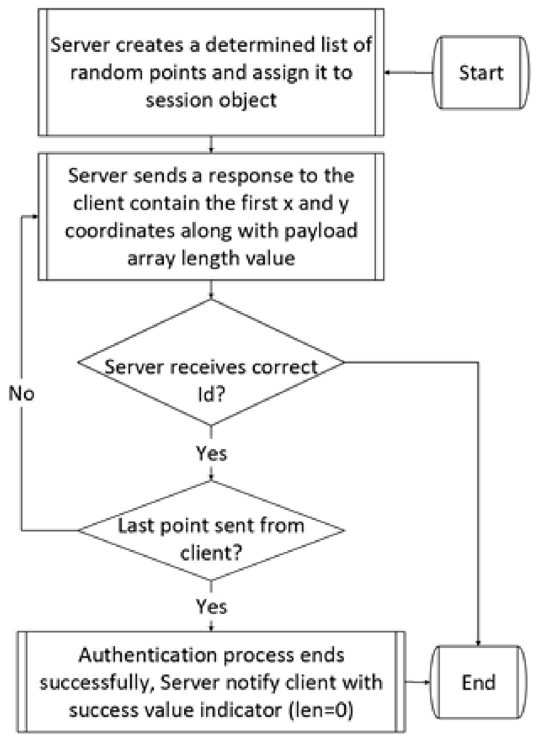
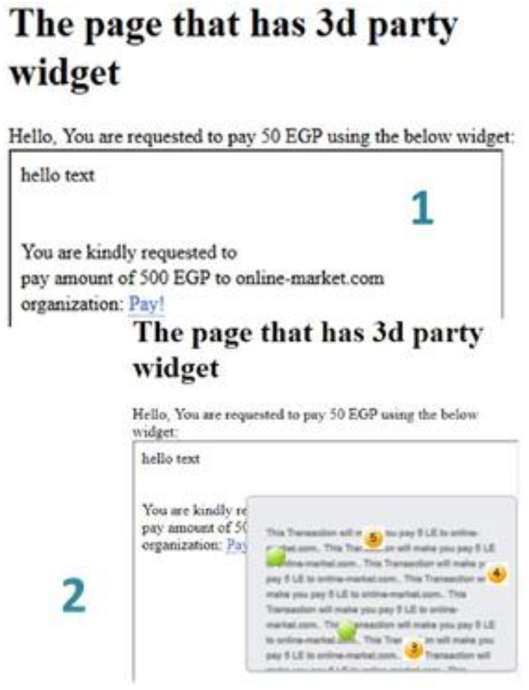

Start
Server creates a determined list of random points and assign it to session object
Server sends a response to the client contain the first x and y coordinates along with payload array length value
Server receives correct Id?
Yes
No
Last point sent from client?
Yes
Authentication process ends successfully, Server notify client with success value indicator (len=0)
End

Fig. 6 Flow Chart demonstrates server procedure to generate and validate the pattern point list

One when user already has a registered pattern and the second when user does not have a registered pattern. In case of user already has registered pattern, a message appears to user with instructions prompting him to draw a nine points pattern, thus, user has to draw the correct pattern exactly matches which saved on the server, when user finishes drawing the pattern, he should move the cursor out of the red boarder in order to send the pattern to server for verification, server compares the received pattern against the one already saved on database, if both match, it grant the action, otherwise, it will send the user hovers over the handler there will be one of two scenarios, first a rejection message
In case that user does not have a registered pattern, user will be instructed to create a new pattern. Using the pop-up window or through navigating to pattern registration screen, user pattern registration from client side perspective should go as in Figure 9. User choose to create a new verification pattern, then a panel contains a grid of nine points appears to user after that user has to draw a pattern by hovering over number of points,and after that, user has to move the cursor


The page that has 3d party widget

Hello, You are requested to pay 50 EGP using the below widget:

hello text

1

You are kindly requested to
pay amount of 500 EGP to online-market.com
organization: Pay!

The page that has 3d party widget

Hello, You are requested to pay 50 EGP using the below widget:

hello text

2

You are kindly re
pay amount of 5(
organization: Pay

Fig. 7 Figure demonstrates user confirms payment

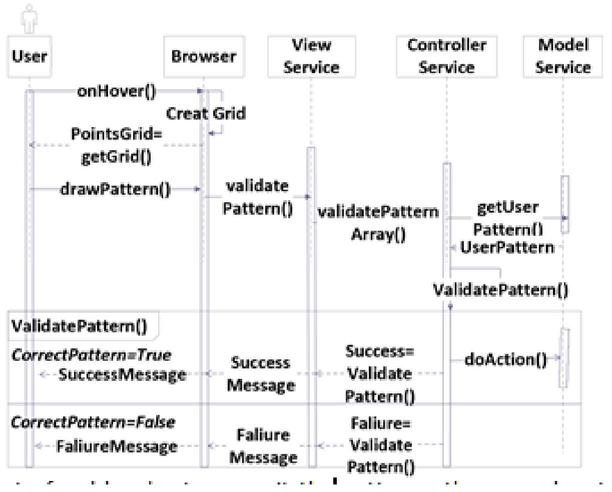
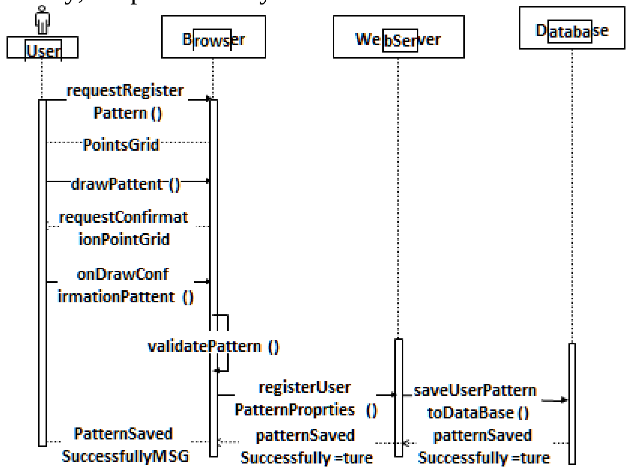


Fig. 8 Sequence Diagram to show user interaction with server and server processing to validate persistent pattern technique

out of red border to commit the pattern , then user has to redraw the pattern for confirmation, finally message appears to client that he can use the pattern for verification.
Server side process goes as follows: After the server has authenticated the client identity, it will receive the confirmed pattern and save it to database.

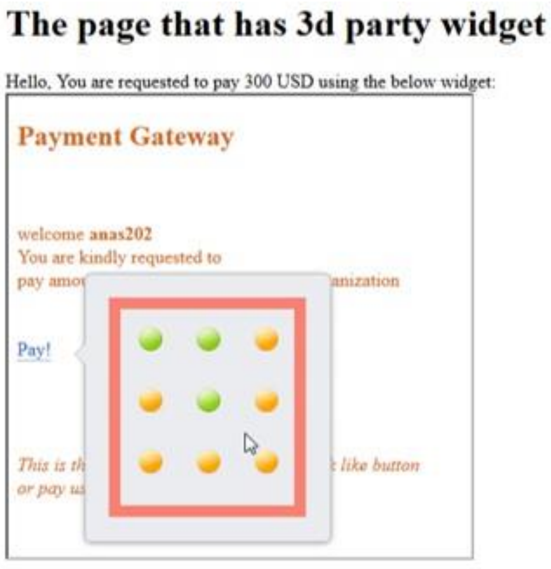
We are going to analyze our solution as a whole in addition to pattern generation in particular from the following perspectives: The system performance will be evaluated and complexity analysis will be performed to analyze running time of the pattern generation algorithm. Finally, the predictability



User
Browser
WebServer
Database
requestRegister Pattern ()
PointsGrid
drawPattent ()
requestConfirmationPointGrid
onDrawConfirmationPattent ()
validatePattern ()
registerUser PatternProprties ()
saveUserPattern toDataBase ()
PatternSaved SuccessfullyMSG
patternSaved Successfully =ture
patternSaved Successfully =ture

Fig. 9 Sequence Diagram to demonstrate user pattern registration for persistent pattern registration

of the generated patterns will be analyzed.
Figures 10, and 11 illustrate the confirming persistence pattern scenario.


The page that has 3d party widget

Hello, You are requested to pay 300 USD using the below widget:

Payment Gateway

welcome **anas202**
You are kindly requested to
pay amo anization

Pay!

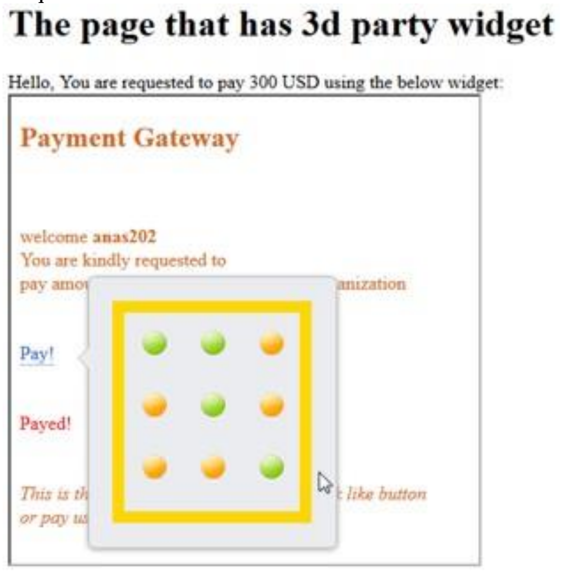
This is th t like button
or pay us

Fig. 10 User draws the pattern hovering over points

When the user does not have a registered persistence pattern, third party widget directs the user to create one by drawing a pattern of his choice and reenter it as confirmation Figure 12 is a screen-shot that illustrates recording the confirmation pattern.

4. Implementation

Following snippet code 1 shows the function which the point id value and coordinates assigned in random pattern technique:


The page that has 3d party widget

Hello, You are requested to pay 300 USD using the below widget:

Payment Gateway

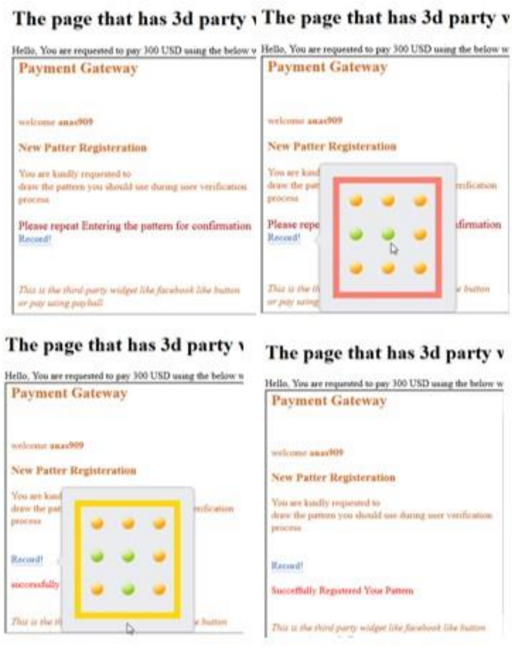
welcome **anas202**
You are kindly requested to
pay amo anization

Pay!

Payed!

This is th t like button
or pay us

Fig. 11 User moves the pointer out of the panel boarder to submit the action


The page that has 3d party w

Hello, You are requested to pay 300 USD using the below w

Payment Gateway

welcome **anas909**

New Patter Registeration

You are kindly requested to
draw the pattern you should use during user verification
process

Please repeat Entering the pattern for confirmation
Record!

successfully

Succeffully Registered Your Pattern

This is the third party widget like facebook like button
or pay using payball

Fig. 12 User asked to confirm pattern

Following snippet of server side code describe how to validate the user sent Id against the next point expected id:
A. Security Analysis

Variety of precautions have been taken to protect the technique, for example sending a random nonce along with point id, to protect against CSRF attacks.

```
function WriteNextPoint(x, y, id) {
        var elem= document.createElement("img");
        elem.setAttribute("src", "ready.png");
        elem.setAttribute("id", id);
        document.getElementById("auth").
        appendChild(elem);
                $("#" + id + "").addClass("point");
        $("#" + id + "").css({"left": x});
        $("#" + id + "").css({"top": y}); $("#" + id +
        "").hover(function() { doEffect(this);
        authNext(this.id); $("#"+id+"").
                        off('mouseenter mouseleave');

});
}
```

Listing 1: Code Snippet for Adding hover Event Client Side

```
if ((((ArrayList<Point>) session.
        getAttribute("Points")).get(i).
          isVisited() == false)) { if (((ArrayList<Point>)
                session.
                        getAttribute("Points")).
                          get(i).getId().
                                  equalsIgnoreCase(pointId)){

                                          //send the next correct point
                          }
```

Listing 2: Code Snippet for Server Side Validation for Point Ids

One of the major attacks that may target our mechanism is to deceive the user browser to zoom out the text, yet, this can be solved by mixing up the panel background by images and text phrases.

Other attacks may target hiding parts of the panel to prevent the user from reading the notation, yet, hiding part of the panel would prevents the user from completing the pattern Transportation layer should be encrypted using https to prevent the attackers from intercepting the server responses and resending Ids values back.

Nevertheless on a panel of 240p as width and 130px as height, and Point size of 24p as width and 24p height the probability of correct pattern would be 240∗ 130 = 312005 which approximately equals 1/31023 Persistent Pattern:

Anti-CSRF technique has been implemented to prevent resending the authentication pattern, if the attacker could make a successful attack and get user's secret panel. Double Submit Cookies technique [13] has been implemented by sending a nonce values along with secret pattern matches a cookie value for hosted widget domain

Transportation layer should be encrypted using https to prevent the attackers from intercepting the user's secret pattern and lunch another clickjacking attack against him tricking the user to draw the revealed secret pattern

Experimental Evaluation: Experiment setup will included personal computer with Apache Tomcat server installed, Internet Explorer and Mozilla FireFox browsers used along with Process Explorer application to measure the CPU and memory utilization. Personal Computer has Intel Core i-7 CPU 2.6 GHz and 8.00 GB Ram running 64-bit Microsoft windows 7 operating system

Number of points: We chose to use the five point pattern which generates probability of 3*10 23

Parameters for the expriement are defined as follows, Number of points: 5, Panel size: 240px*130px, Browser type: Mozilla Firefox and Microsoft, Server type: Apache Tomcat installed on local machine, Round trip time average: 2.08 seconds, Panel size without jQuery java script library: 44 KB, Random Generation Type: SecureRandom.

Factorial Design:

Based on table I, Most effective parameter in response time is the total number of points (N)

Table 1: RESPONSE TIME STATISTICS

Total number of points (N)	Number of points per require (K)	Response time (including rendering times + thinking time) in seconds	
		Chrome	FireFox
2	1	2.5±0.348	3.5±0.468
2	2	3.1±0.579	3.5±0.468
5	1	5.5±0.896	5.8±0.751
5	5	5.8±0.522	5.5±0.468

Based on table II, most effective parameter in client memory is the browser type

Table 2:CLIENT MEMORY STATISTICS

Total number of points (N)	Number of points per require (K)	Client Memory in MB	
		Chrome	FireFox
2	1	.35±.205	1.58±0.41
2	2	1.46±0.92	0.856±0.399
5	1	.56±.24	1.34±0.21
5	5	.4±.27	1.4±0.24

Based on table III, most effective parameter in client CPU is the browser type

Table 3: CLIENT CPU STATISTICS

Total number of points (N)	Number of points per require (K)	Client CPU in Percentage	
		Chrome	FireFox
2	1	3.574±2.156	2.264±0.637
2	2	3.512±0.7098	2.234±1.317
5	1	4.31±1.21	3.343±0.95
5	5	3.11±0.53	3.181±1.19

Based on table IV, Most effective parameter on server memory is number of points per request (K)

Table. 4: SERVER MEMORY STATISTICS

Total number of points (N)	Number of points per request (K)	Server Memory In MB
2	1	16±0
2	2	25.4±1.82
5	1	38.77±3.40
5	5	15.41±6.5

Based on table V,Most effective parameter on server CPU is the total number of points

Table. 5: SERVER CPU STATISTICS

Total number of points (N)	Number of points per request (K)	Server Memory In MB
2	1	1.74±1.02
2	2	2.28±0.66
5	1	0.88±0.38
5	5	0.91±0.31

The experiment assumes that users are successful in all of their attempts to follow the patterns and thereby doing the intended action

In N5K5, We have to implement margin to ensure that user did not accidentally hover over wrong point as points may be rendered within same coordinates.

Performance Analysis:

In order to assess the application performance in massusage, we launched the following experiment: Number of users/Threads: 1000 user, Loop Count: 50, Total number of Request: 50,000.

we define response time in the experiment as the difference between time when request was sent and time when response has been fully received.

The experiment examine n5k5 request structure which consists of two http requests, one as a retrieval point array (Retrieve N5K5 request) and the other request is to submit the point ids list(Submit N5K5). Another separate run for the unsecured action has been launched, the experiment assumes that users are successful in all of their attempts to follow the patterns and send correct ones. The following consolidated graph IV-A demonstrates the average response time for submitting secure requests comparing to the original request with no security technique in place.

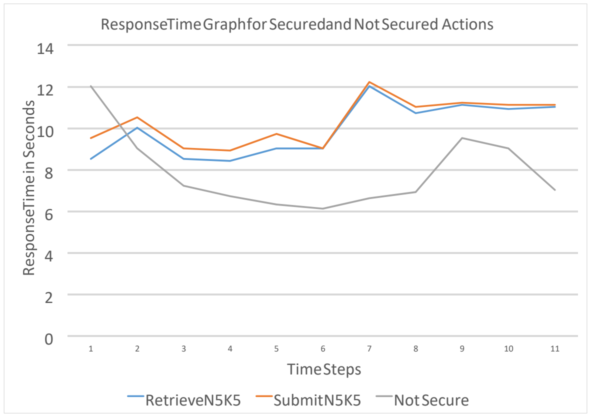


Fig. 13 Response Time Graph illustrates that implementing secure token solution has a light footprint despite the number of requests

Following micro bench mark VI illustrates a comparison between each of the techniques in terms if number of request, newt work overhead and total time used over the network to have the action done.

Table. 6: MICRO BENCHMARK

Items	N2K1	N5K1	N5K5
Number of requests	5	8	8
Network overhead	2.1 KB	2.3	7
Elapsed Time	769ms	2.95 S	484MS

5. Related work

Clickjacking attacks has wide range of vectors, one of those attacks are using fake cursor plus hiding the real cursor using CSS styles tricking the user to trigger actions he do not know about [14], [15], in Sebastian solution [16] the author suggest to disable the CSS style on the cursor, which would be a serious limitation in many types of applications (e.g. text editors), Through the idea of using hovering instead of clicking, our proposed solution by definition solves most of the limitations that are in other solutions, we also do not require milliseconds of waiting in case pop ups window as defense, in HDTCV [9] the author made a technique in came up with an offline analysis for websites; the offline methodology will not be able to introduce protection if the attacker used new website to launch their attack through it, one of the irrelevant researches also.

Mechanisms like randomly rendering [17] the buttons faces wide rejection as it has usability issues, besides, the attack could be conducted by trails and errors. One of the client side countermeasures was NoScript ClearClick Firefox plug-in [18], ClearClick mechanism is by creating two screenshots, first one as screenshot of the framed page and the second is by executing the real code, if the two screenshots differ, plugin warns the user by a pop-up, besides, solution suffers from false positives, other related proposed solution called ClickIDS [19] been developed to reduce with a goal to reduce false positive, yet, it cannot detect some types of attacks like partially overplayed or cropped elements, besides, due to the mechanism is being used to compare the bitmaps screenshot, it cannot detect types of cursor spoofing attacks.

One of the solutions related to ours is to introduce new types of controls that require user focus (e.g. a Swipe or holding the mouse for a certain amount of time, etc.) [5]. One of the solution is to dim the screen except the control that will receive the action to prevent other controls to by overlapped with, For the time being, none of these prosed solutions has been implemented by any of the major internet browsers [3].Adapting Huang et al. [3], Context consists of visual context and temporal context, Visual context is about the screen sensitive components view state and pointer feedback sbtate right before user's action, and to ensure its integrity we need to guaranty its total visibility to the user and Temporal context refers to that no changes have happened to the targeted element at the point of time right before the user action, a popular example for that would be by launching a bait-and-switch attack by first baiting the user with a "claim your free iPad" button and then switching in a sensitive UI element right before the anticipated time of user click.

Our proposed solution is different to CAPTCHA [20] , CAPTCHA does not solve clickjacking problem, moreover, one of the ways to bypass CAPTCHA can be done through attack similar to clickjacking, a famous attack against CAPTCHA is typically goes as following firstly, automated bot extracts the CAPTCHA content from the CAPTCHA protected form, secondly, the attacker injects the extracted CAPTCHA challenge content needs to be solved in phishing website luring its users to solve CAPTCHA in order to get the phishing site services (e.g. free downloading or cracks, etc...), thirdly, when the users answers that hijacked CAPTCHA, attackers use the answers to solve the CAPTCHA challenge mentioned at the first step. On the other hand, our proposed solution is not a replacement for CAPTCHA as it does not prevent against automation since all expected input is being sent to client side.

6. Conclusion

Clickjacking attack is one of the challenges that faces websites which trying to expose its features as widgets, the hovering patterns technique is expected to guarantee an actual level of protection against the attack with neglectable performance effect as it guarantees the total intention of the user ensuring both of visual and temporal context integrity. Finally, the fact that this technique is not using the clicking concept and using the hovering instead will make all current running attack invisible

References

[1] SimilarTech. https://www.similartech.com/categories/widget/, 2015.

[2] A.Sankara Narayanan. Clickjacking vulnerability and countermeasures. International Journal of Applied Information Systems (IJAIS), 4, 2012.

[3] Lin-Shung Huang; Alex Moshchuk; Helen J. Wang ;Stuart Schechter ; Collin Jackson. Clickjacking: Attacks and defenses. USENEX Security, 2012.

[4] Wikipedia. Likejacking. http://en.wikipedia.org/wiki/ ClickjackingLikejacking.

[5] Brad Hill. Anti-clickjacking protected interactive elements. January, 2012.

[6] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. volume 2, 2010.

[7] David Ross and Tobias Gondrom. Http header field xframe-options, 2013.

[8] Nikhil Limaje Abhilash Gupta Mridul Jain Bernard Menezes Kanpata Sudhakara Rao, Naman Jain. Two for the price of one: A combined browser defense against XSS and clickjacking.

[9] D Kavitha, S Chandrasekaran, and SK Rani. Hdtcv:

[10] Hybrid detection technique for clickjacking vulnerability. In Artificial Intelligence and Evolutionary Computations in Engineering Systems, pages 607–620. Springer, 2016. [10] The World Wide Web Consortium (W3C). Document object model (dom). http://www.w3.org/DOM/what.

[11] The World Wide Web Consortium (W3C). Html/elements/div. https://www.w3.org/wiki/HTML/Elements/div.

[12] R. Lal. Web widget. http://www.widgetsgadgets.com/2007/08/what-is-web-widget.html.

[13] Paul Petefish, Eric Sheridan, and Dave Wichers. Crosssite request forgery (csrf) prevention cheat sheet, 2011.

[14] E. Bordi. Proof of concept - cursorjacking. http://static.vulnerability.fr/noscript-cursorjacking.html.

[15] K. Kotowicz. Cursorjacking again (january 2012). http://blog.kotowicz.net/2012/01/cursorjackingagain.html.

[16] M. Johns;L. Sebastian. Tamper-resistant likejacking protection. 2013.

[17] Brad Hill. Adaptive user interface randomization as an anti-clickjacking strategy. May, 2012.

[18] G. Maone. Noscript clearclick. http://noscript.net/faqclearclick, January 2012.

[19] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pages 135–144. ACM, 2010.

[20] Bin B Zhu, Jeff Yan, Guanbo Bao, Maowei Yang, and Ning Xu. Captcha as graphical passwords—a new security primitive based on hard ai problems. IEEE transactions on information forensics and security, 9(6):891– 904, 2014.