MySecPol: An Architecture for Safe and Secure Browsing using Client-side Policy

Maj Amit Pathania
(163054001)

Indian Institute of Technology Bombay

June 29, 2018

Table of Contents

Introduction

Introduction

- Web browsers handle content from different sources making them prone to various attacks.
- A simple architecture for defining client side policies using `MySecPol`, proposed policy specification language. These access control policies capture the security requirements of the user.
- The client side policies give the user control to decide the content being served to him. User can define these policies independent of the browser or OS.
- These policies are then realized in the browser as an extension. The client side policies can capture the essence of existing security mechanisms and combine them to provide more robust protection.

Motivation

Motivation

- Existing **server side protection** involves web developers to rewrite code by restricting the use of vulnerable JavaScript functions or add additional security headers in HTTP responses from server.
- The **client-side protection** mechanisms include enabling browser security and privacy settings or use of browser extensions for protection against common attacks.
- The client has to either rely on either web developers for privacy and security or download different browser extensions for protection against different attacks.
- Policy based architecture provides **user controlled browser enforced security mechanism** for safe browsing and provides level of security as desired by the user.

Our Approach Via MySecPol

Key Idea : Policy prerequisites

- The client side policy *should be simple* and must not require special skill or expertise.
- The policy *should be non-conflicting* and should be able to resolve conflicts in case of either dependent or conflicting rules.
- The policy *should be implementable* in current Web context without browser modifications. Current web standards should support the defined policies.

Proposed Architecture

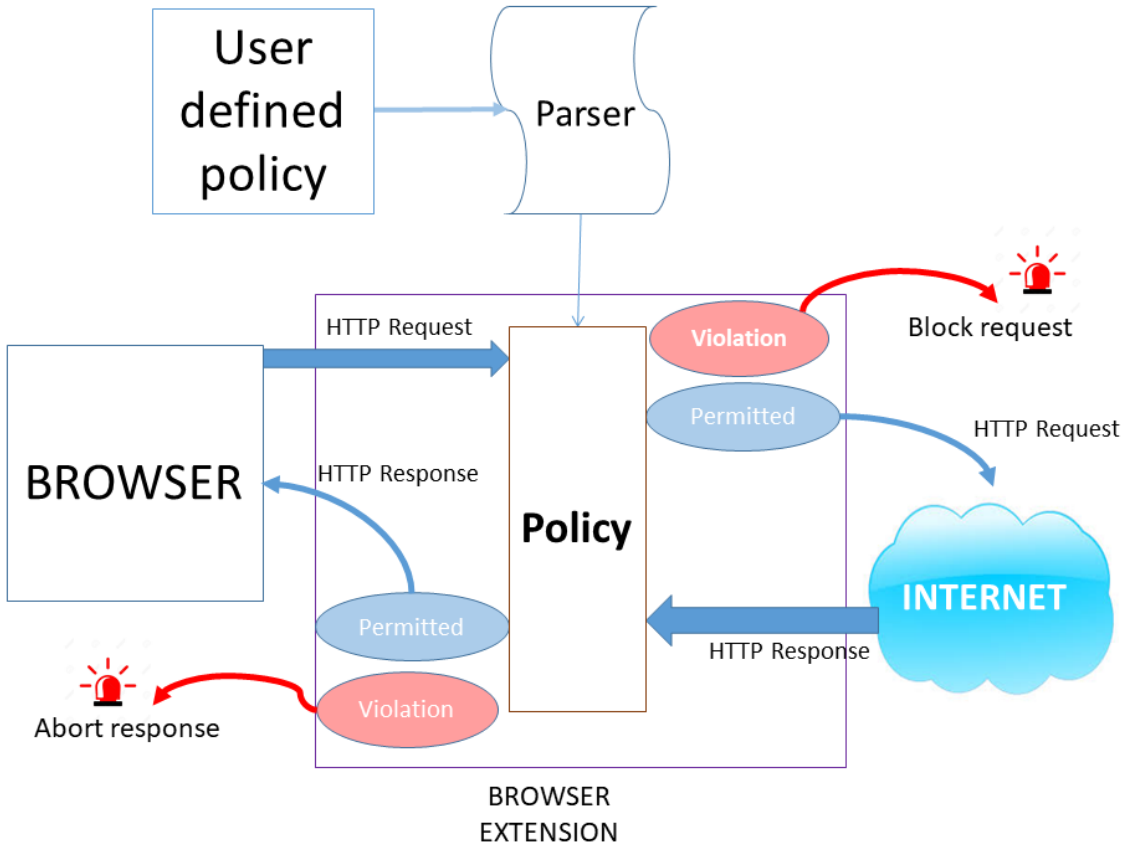

Figure: Proposed architecture for secure browsing

Syntax of client side policy

```
policy ::= rule *..
rule ::= action field domain-list
action ::= allow | deny
field ::= resource | browser-setting | HTTP-header | property
resource ::= JavaScript | image | iframe | font | object | XMLHTTPRequest
           | stylesheets | media
browser-setting ::= thirdpartycookies | autofill | safeBrowsingEnabled
                  | passwordSavingEnabled | doNotTrackEnabled
                  | webRTCIPhandling
HTTP-header ::= User-Agent | Referer
property ::= maxtabs | access | connection-type | downloads | executable
                  | HttpOnlycookies | cookies | auth-info
connection-type ::= https | http
domain-list ::= dstn-domain [host-domain(optional)]
dstn-domain ::= origin +.. | crossdomain | crossdomain-|*|*- |ANY
host-domain ::= origin +.. | ANY
origin ::= RFC 6454
```

Policy interpretation

Let P_U be user defined policy which contains set of rules R s.t. rules R_1, R_2, ... $R_n \in R$. Let P_E be the policy implemented by the browser containing set of rules R′ s.t. rules R'_1, R'_2, ... $R'_n \in R'$.

- **Property 1**: All rules defined by user in his policy P_U will be included in effective policy P_E if these rules are disjoint with each other. Two rules R_i and R_j are said to be disjoint if they are independent to each other and are non-contradicting.

```
Rule1: deny javascript www.xyz.com
Rule2: deny image      www.abc.com
Rule3: deny javascript www.abc.com
```

Policy interpretation

Let P_U be user defined policy which contains set of rules R s.t. rules R_1, R_2, ... $R_n \in R$. Let P_E be the policy implemented by the browser containing set of rules R′ s.t. rules R'_1, R'_2, ... $R'_n \in R'$.

- **Property 2**: If there are two rules such that one rule(R_j) is a subset of the other rule (R_i), then only rule (R_i) will be included in the effective policy P_E.

```
Rule1: deny javascript *
Rule2: deny javascript  www.abc.com
```

- **Property 3**: If two rules for a resource or domain are either dependent or conflict with each other, then the stricter of the two will be taken in the effective policy P_E ignoring the other.

```
Rule1: deny javascript *
Rule2: allow javascript www.abc.com
```

Security Policies

Blacklisting scripts

Websites also include tracking, advertising and analytic scripts for revenue generation or tracking user behavior. We can define policy that either blocks scripts from any specific domains or block all domains or only cross-origin domains.

```
deny    javascript   *://*.google-analytics.com/*
deny    javascript   *://*.zedo.com/*
deny    javascript   *
deny    javascript   crossdomain
```

Selective resources

An user can specify policies for various HTTP request for resources like image, flash objects, stylesheets, and so on.

```
deny    object          *
allow   image           *://giphy.com/*
deny    stylesheet      *://evil.com/*
deny    media           *
deny    XMLhttprequest  *
deny    iframe          *
```

Creating blacklist and whitelist

User may desire to either block access to certain untrusted websites by creating blacklist or allow access to only few web sites by creating whitelist.

```
deny    access   *://*.evil.com/*
deny    access   *://*.tracker.com/*
```

```
deny    access   *-
allow   access   *://*.iitb.ac.in/*
allow   access   *://*.abc.com/*
```

Setting browser's privacy settings

Users' privacy can be protected by configuring various browser features like auto-fill option for web forms, password saving for different websites, third-party cookies, webRTC traffic handling, safe browsing mechanisms and doNotTrack header for HTTP requests. A sample privacy protection policy is given below.

```
deny    thirdpartycookies          *
deny    autofill                   *
allow   safeBrowsingEnabled        *
deny    passwordSavingEnabled      *
allow   doNotTrackEnabled          *
deny    webRTC                     *
```

Misc Policies

- Remove cookie information from cross-origin requests.
- Block non-HTTPS connections.
- Block all application or executable file downloads.
- Limit the number of opened tabs.
- Blocking User-Agent or referer headers.
- Restrict cookie type to 'HttpOnly' cookies.
- Create whitelist for cross-origin requests.
- Implementing Content Security Policy [1] at browser.

CSP at browser

deny	object	*	*://abc.com/*
allow	image	www.hdimg.com	*://abc.com/*
allow	javascript	www.xyz.com	*://abc.com/*
deny	iframe	*	*://abc.com/*

Above policy is equivalent to following headers sent by '*://abc.com/*' :-

```
Content-Security-Policy: default-src 'self';
                         object-src 'none';
                         img-src www.hdimg.com;
                         script-src www.xyz.com ;
X-Frame-Options: DENY
```

Implementation Details

Implementation Details

- Based on the policy defined by the user in MySecPol language, the parser will set browser extension parameters.
- We have used chrome.webRequest [2] and chrome.privacy APIs [3] in our prototype extension.
- The chrome.privacy API is used to control user's privacy settings in chrome and the chrome.webRequest API is used to intercept, block, or modify HTTP requests and responses.

Parser

Algorithm 1: Parser implementation

```
1  F = Set of fields (as defined in MySecPol)
2  Initialize variables for each field
3  for each rule R in policy P_U: do
4      if R.field ∈ F then
5          f = R.field
6          flag_f = true
7          if R.action == 'allow' then
8              WL_f.add(R.dstn-domain)
9          end
10         if R.action == 'deny' then
11             BL_f.add(R.dstn-domain)
12         end
13     end
14 end
15 //Resolve conflicts in user defined policy
16 for each field i in F: do
17     if '*' ∈ BL_i and WL_i != NULL then
18         WL_i = NULL
19     end
20     if '*' ∈ WL_i and BL_i != NULL then
21         WL_i.remove(*)
22     end
23     if 'domain' ∈ BL_i and 'domain' ∈ WL_i then
24         WL_i.remove('domain')
25     end
26 end
27 Write variable values to the browser extension
```

Policy implementation

Algorithm 2: Policy check by prototype extension

Input : HTTP request or HTTP response
Output: Allow or deny input

```
1  for each HTTP request / HTTP response do
2      if Policy flag set for Resource then
3          Get domain of request / response;
4          if Domain in whitelist then
5              Allow request / response ;
6          end
7          if Domain in blacklist then
8              Block or modify request/ response ;
9          end
10     else
11         Allow request / response;
12     end
13 end
```

Experimental evaluation

Soundness

- Tested prototype for different rules followed by various combinations of these rules for a complete solution.
- Monitored the HTTP requests sent, responses received and the content served by the browser in absence and presence of certain rules in the policy.
- Logged the violations of rules reported by the extension and analyzed the logs manually for verification.
- The effectiveness by design, since it intercepts and inspects all incoming and outgoing HTTP requests for possible violations against the policy.

Performance

- Additional checks for each outgoing requests and received responses adds overhead.
- Performance overhead in terms of page load time after policy implementation.
- Used an open source Google Chrome extension called 'Performance-Analyser' [4] to measure load time performance overhead.
- Measured load times and other parameters of Alexa Top websites with and without our prototype extension for different policies.

Performance: Policy to block all cross-origin scripts

Domain	**Load time**(in ms)			**Total requests**		**JS requests**	
	without extension	with extension	% increase	without extension	with extension	without extension	with extension
google.com	312	384.5	23.24	19	14	6	2
youtube.com	5285	1875	-64.52	67	28	10	0
facebook.com	2798	1334	-52.32	313	14	102	0
baidu.com	159	599	276.73	18	6	7	0
wikipedia.org	387	398.5	2.84	6	5	2	2
reddit.com	679.5	1432	110.74	68	31	15	0
yahoo.com	3479	1003	-71.16	150	31	82	0
qq.com	5861.1	1510	-74.24	150	114	26	6
google.co.in	354	376	6.21	18	12	5	2
taobao.com	2744.5	2800	02.02	112	12	29	0

Table: Performance with cross origin scripts blocked

Note: Increase in average load time for websites due to additional checks but it decreased for the websites where third-party scripts were more, thus reducing the total requests and fetched content.

Performance: Policy to block all executable downloads

% increase in load time	% of websites
less than 10%	50%
10% -50%	31%
50% -90%	8%
more than 90%	11%

Table: Performance with policy to block all executable downloads

Note: Average increase of 30% in load time with this policy as each response was monitored. However, this overhead was less than 10% for fifty percent of websites.

Performance: Policy to block all iframes

Domain	**Load time** (in ms)			**Total requests**	
	without extension	with extension	% increase	without extension	with extension
google.com	312	336	7.69	19	19
youtube.com	5285	4200	-20.5	67	56
facebook.com	2798	3884	38.81	313	244
baidu.com	159	538	238.3	18	17
wikipedia.org	387	395	2.06	6	5
reddit.com	679.5	1663	144.6	68	62
yahoo.com	3479	1859	-46.56	150	150
qq.com	5861	6052	3.25	150	150
google.co.in	354	377	6.49	18	16
taobao.com	2744.5	2365	-13.81	112	105

Table: Performance with policy to block iframes

Note: Increase of 26.96% in average page load time. Increase was less than 10% for most websites. Decrease in load time of the websites which contain iframes for embedded content like videos.

Compatibility

- Defined in terms of loss of functionality and ease of usage for the given website.
- More restrictive policy means more loss of interactive features of the websites.
- Manual verification was done to check any loss of functionality.

Compatibility: Policy to block all cross-origin JavaScripts

- No degradation of useful web content displayed for sites which use cross-domain scripts for ads and analytics.
- Complete loss of functionality for a few websites like youtube.com, facebook.com and reddit.com.
- Reason: Scripts loaded on these websites are from different domains (`https://static.xx.fbcdn.net` for `facebook.com`, `https://s.ytimg.com/` for `youtube.com` and `https://www.redditstatic.com/` for `reddit.com`).
- Solution: Able to retrieve most of the functionality on these websites by whitelisting these scripts manually.

Compatibility: Policy to block all iframes

- The loss of functionality was either not found or was very limited.
- Few websites which use iframes to display site content faced the significant degradation of services.
- Solution: We can modify 'x-frame-options' response header to allow iframes from SAMEORIGIN or from whitelisted domains.

Compatibility

- No loss of functionality with policy to block executable downloads.
- No significant loss of interactivity for most of the websites when we blacklisted common advertising and analytic domains except for those websites which deny their content when the domains were blocked.
- Some websites which require third-party cookies to be enabled give error when we disable third party cookies by our policy.

Related work

JavaScript Subsets and Rewriting

Work	Brief explanation	Drawback
evalinsandbox() [5]	Sandboxing mechanism to run code using *eval()* with reduced privileges. The principal for code running in the sandbox is defined in the constructor and properties available to code running in the sandbox are also specified	Dependent on web developers for security and privacy. Vulnerability due to bad coding practices, use of unsafe JavaScript functions, lack of input sanitization and unrestricted access to third party scripts.
ECMAScript 5 strict mode [6]	Standardized subset and restricted variant of JavaScript. Invoked by statement *'use strict'*	
Caja Compiler [7]	Makes third party content safe to embed in the website	
ADsafe[8]	Subset of JavaScript that restricts the third-party code from doing any malicious activity	

Browser Modifications

Work	Brief explanation	Drawback
ScriptInspector [9]	Modified Firefox browser capable of intercepting and recording sensitive API calls from third-party scripts to critical resources. It records accesses that violate the policy for a given domain.	Dependent on web developers for defining security policies.
ConScript [10]	Introduces a new attribute 'policy' to the HTML <script> tag that can store a policy defined by the web developer.	
FlowFox [11]	Modified Firefox browser that implements information flow control for scripts by assigning labels. Uses Secure Multi Execution [12].	No widespread implementation due to browser modifications. User can't define security policies.
Virtual Browser [13]	Virtualized browser which runs JavaScripts in a sandboxed environment to visualize their interaction with sensitive data in controlled environment.	

Browser Extensions

Work	Brief explanation	Drawback
Noscripts [14]	Provides anti-XSS and anti-Clickjacking protection using white-listing mechanisms. NoScript blocks all JavaScript, Java, Flash Silverlight and other executable contents by default.	Protect against XSS attacks. No flexibility.
Ghostery [15]	Detects and blocks tracking technologies on the websites user visit.	Protection against tracking only. User can't define security policies.
Abine [16]	Controls third-party services which exist on the current page.	
CsFire [17]	Strips authorization information from cross-origin HTTP requests, except for expected requests to mitigate CSRF attacks. Uses either client defined policy or server supplied policy.	Can't handle genuine cross origin requests in absence of server or user supplied whitelist. Protects against CSRF and clickjacking attacks only.

Other related work

Work	Brief explanation	Drawback
Cross-Origin Request Policy (CORP) [18]	Enables a server to control cross-origin requests by policy defined by a web developer. Policy is sent as additional HTTP response header.	Protect against CSRF and clickjacking attacks only.
Browser Enforced Authenticity Protection [19]	Modifies HTTP headers by stripping authorization information from all cross-origin requests after checking referrer header	Protection against clickjacking attacks. User can't define security policies.
Allowed Referrer Lists (ARLs) [20]	Browser security policy restricts the browser from sending ambient authority credentials with HTTP requests. The sites can specify whitelist of allowed referrer URLs, to which browsers can send authorization state.	No widespread implementation due to changes required. Protect against CSRF and clickjacking only.

Demonstration

Conclusion

Conclusion

- MySecPol is independent of platform/browser so easy to port policies from one browser to other, easy to understand and intuitive to write and can integrate several existing policies.
- Implemented it as a browser extension for Google Chrome browser on Windows 10 and Ubuntu 16.04. Other browsers also provide similar APIs.
- The experimental results show that our solution provides effective security with low-to-moderate overhead for a spectrum of users/user applications.
- MySecPol provides flexibility to import existing ad-hoc solutions against clickjacking, CSRF, phishing, etc., by adding new fields or keywords.
- A paper has been submitted to **ACASC(Annual Computer Security Applications Conference)** 2018.

References

References I

[1] W3C Working Group. Content security policy, 2015.

[2] Google Developers. chrome.webrequest - google chrome, 2018.

[3] Google Developers. chrome.privacy- google chrome, 2018.

[4] Michael Mrowetz. Performance-analyser, May 2015.

[5] MDN Web Docs. Evalinsandbox reference, 2017.

[6] MDN Web Docs. Javascript strict mode reference, 2018.

[7] Ben Laurie Ihab Awad Mark S. Miller, Mike Samuel and Mike Stay. Caja: Safe active content in sanitized javascript, June 1, 2017.

[8] D. Crockford. Adsafe: Making javascript safe for advertising, 2008.

References II

[9] Yuchen Zhou and David Evans. Understanding and monitoring embedded web scripts. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 850–865, Washington, DC, USA, 2015. IEEE Computer Society.

[10] Leo A. Meyerovich and Benjamin Livshits. Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 481–496, Washington, DC, USA, 2010. IEEE Computer Society.

References III

[11] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. Flowfox: A web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 748–759, New York, NY, USA, 2012. ACM.

[12] D. Devriese and F. Piessens. *Noninterference Through Secure Multi-Execution*. In Proceedings of the IEEESymposium on Security and Privacy, pages 109-124, 2010.

References IV

[13] Yinzhi Cao, Zhichun Li, Vaibhav Rastogi, Yan Chen, and Xitao Wen. Virtual browser: A virtualized browser to sandbox third-party javascripts with enhanced security. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 8–9, New York, NY, USA, 2012. ACM.

[14] InformAction. Noscript, 2018.

[15] Ghostery Inc. Ghostery. `https://www.ghostery.com/`, June 2018.

[16] Abine Inc. Abine blur. `https://www.abine.com/index.htm`, May 2018.

References V

[17] Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens, and Wouter Joosen. Csfire: Transparent client-side mitigation of malicious cross-domain requests. In Fabio Massacci, Dan Wallach, and Nicola Zannone, editors, *Engineering Secure Software and Systems*, pages 18–34, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[18] Krishna Chaitanya Telikicherla, Akash Agrawall, and Venkatesh Choppella. A formal model of web security showing malicious cross origin requests and its mitigation using CORP. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pages 516–523, 2017.

References VI

[19] Ziqing Mao, Ninghui Li, and Ian Molloy. Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 238–255, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] Alexei Czeskis, Alexander Moshchuk, Tadayoshi Kohno, and Helen Wang. Lightweight server support for browser-based csrf protection, 05 2013.

[21] Willem De Groef, Dominique Devriese, and Frank Piessens. *Better Security and Privacy for Web Browsers: A Survey of Techniques, and a New Implementation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[22] Mozilla Inc. Postmessage.

References VII

[23] Acunetix Ltd. Types of xss: Stored xss, reflected xss and dom-based xss. last checked: 13-10-2017.

[24] Jakob Kallin and Irene Lobo Valbuena. Excess xss: A comprehensive tutorial on cross-site scripting. last checked: 13-10-2017.

Thank You!

Same Origin Policy(SOP)

- Content received from one website is allowed to read and modify other content received from the same site but is not allowed to access content received from other sites.
- Scripts contained in a first web page can access data in a second web page, but only if both web pages have the *same origin*.
- Malicious script on one page is prevented from obtaining access to sensitive data on another web page through that page's Document Object Model.
- Strict SOP disallow third-party images and style sheets to be loaded making browser very dull [21].
- Vulnerable to XSS attacks.

HTML5 PostMessages for cross domain messaging

- This method is used to pass data between two different domains (Cross domain communication).
- The pages can send messages with code[22]: newWindow.postMessage(message, target);
- Receiver window can listen for dispatched messages by executing event listener function and after verifying event.origin can read event.message.
- **Vulnerabilities**: Use of the wildcard domain "*" as *target* or if sender's identity is not verified using the *event.origin* and the *event.data* is inserted into HTML DOM without proper validation.

Cross-Origin Resource Sharing (CORS)

- Cross-origin HTTP request is made by webpage when it requests a resource (image,CSS stylesheets,scripts) from different domain.
- Since XMLHttpRequest or Fetch can only make HTTP requests to its own domain as per SOP for resources, HTML5 CORS permits a developer to set up an access control list to allow other domains to access resources [**?**] .
- This can be controlled through the following headers:
  Access-Control-Allow-Origin,
  Access-Control-Allow-Credentials,
  Access-Control-Allow-Methods

Vulnerabilities:Use of regular expressions or wildcard domain "*" to check whether given domain is permitted .

Content Security Policy(CSP)

- The CSP provides http header that allows websites to declare approved sources of content that browser is allowed to load in that page. This helps to reduce XSS risks on modern browsers[1].
- Helps to create a whitelist of sources of trusted content, and browser can execute or render resources from these sources only.
- If CSP is defined as *Content-Security-Policy: default-src 'self'; media-src * ; script-src trustedcode.com* then videos can be loaded from any domain(*) and scripts can be executed from domain trustedcode.com.
- **Vulnerabilities**: Setting permitted source as wild card (*) for scripts or images.

Type of XSS attacks

- **Persistent XSS** [23] involves an attacker injecting a script into a web application that stores user-supplied data into a server-side data store. Web site will then inserts the data into dynamically assembled pages delivered to all users. The example of such an attack involves a message board or web forum or comment field.
- **Reflected XSS**. The malicious string is part of the victim's request to the website.The attacker frames the victim to make a request to the server which containing malicious code which gets reflected and executed inside the browser.
- **DOM based XSS**[24]. The malicious string is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed. The malicious script is inserted as part of innerHTML and executed at client side itself.