Analysis of Web Browser Security Configuration Options

Ananth A. Jillepalli*, Daniel Conte de Leon, Stuart Steiner and Jim Alves-Foss
Center for Secure and Dependable Systems, University of Idaho
Moscow, ID 83843 - USA
[e-mail: {ajillepalli, dcontedeleon, ssteiner, jimaf}@uidaho.edu]
Corresponding author: Ananth A. Jillepalli

Received March 21, 2018; revised May 16, 2018; accepted July 8, 2018; published December 31, 2018

Abstract

For ease of use and access, web browsers are now being used to access and modify sensitive data and systems including critical control systems. Due to their computational capabilities and network connectivity, browsers are vulnerable to several types of attacks, even when fully updated. Browsers are also the main target of phishing attacks. Many browser attacks, including phishing, could be prevented or mitigated by using site-, user-, and device-specific security configurations. However, we discovered that all major browsers expose disparate security configuration procedures, option names, values, and semantics. This results in an extremely hard to secure web browsing ecosystem. We analyzed more than a 1000 browser security configuration options in three major browsers and found that only 13 configuration options had syntactic and semantic similarity, while 4 configuration options had semantic similarity, but not syntactic similarity. We: a) describe the results of our in-depth analysis of browser security configuration options; b) demonstrate the complexity of policy-based configuration of web browsers; c) describe a knowledge-based solution that would enable organizations to implement highly-granular and policy-level secure configurations for their information and operational technology browsing infrastructures at the enterprise scale; and d) argue for necessity of developing a common language and semantics for web browser configurations.

Keywords: Security policy, application hardening, secure configurations, enforcing least privilege, web browser security, phishing prevention

A shortened version of this paper appeared in IEEE SSCI 2016, December 06-09, Athens, Greece. This version is an expansion of the previously presented research. More information regarding the expansion's specifics are given in Section 1.3. We would like to thank the U.S. National Science Foundation (NSF) and the State of Idaho, for funding this research work. NSF under CyberCorps awards 1027409 and 1565572. Idaho under IGEM Cybersecurity grants of 2012 and 2016. We would also like to thank Mr. Venkata Anirudh Bhandari for his role in achieving this article's contributions.

1. Introduction

Applications connected to the Internet are one of the most vulnerable aspects of the enterprise-level Information Technology (IT) infrastructure. To narrow down the focus of this article, let us restrict the scope of discussion to web browsers. Web browsers are a prime target for malicious cyber-actors. Web browsers, if not configured for enhanced security, are vulnerable to client-side, command execution, and information disclosure attacks as classified by Web Application Security Consortium [1]. These include Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and Content Spoofing, among many others. Phishing and spear-phishing are today a common modus operandi to initiate such attacks. Some of these attacks remain functional even when a browser is fully updated.

1.1 Specific Problem and Proposed Solution

If a web browser is configured with default install-time configuration settings all functionality is enabled for all websites and all users. We cannot rely upon default web browser configurations because these do not implement the principle of least privilege. In this case, attackers would only need a victim to click on a malicious link to trigger execution of attacker controlled code. Such an event can lead to data exfiltration, modification, and system compromise. Furthermore, if the user is currently logged-on to internal corporate website(s), the risk for system and data compromise and upward attacker mobility is greatly increased. If the system being accessed is an industrial control system then such an attack avenue may enable the attacker to directly impact the physical world. In addition, detection of these attacks is particularly difficult due to blending of unauthorized access with normal system access. These attacks may also avoid provenance and dataflow alarms since the system access seems to be originating from an authorized user session. In addition, these attacks can also bypass most firewall rules, that are protecting the security perimeter.

To prevent or mitigate most of these attacks we need to deploy high-granularity enterprise-wide tailored security configurations in applications connected to the Internet. Tailored security configuration implies restricting selective execution of application features, such as JavaScript code and storage, depending on a combination of domain, user/role, client device, and application. An example of such a high-granularity configuration in web browsers would be: enabling JavaScript for user *Mike*, on *MikesPC*, while using Internet Explorer, and when accessing *intranet.example.corp* and disabling JavaScript and access to most other non-intranet sites using the same browser for the same user in the same device, and possibly all other devices. An all-spanning and enterprise-wide configuration to disable browser features like JavaScript and storage are not practical, since most sites would be rendered unusable, including trusted sites. Such highest privilege configurations result in an insecure browsing ecosystem. Therefore, high-granularity enterprise-wide tailored web browser security configurations may be the only solution to the widespread occurrence of security breaches resulting from attacks targeting the web browsing ecosystem.

1.2 Project Context and Research Questions

We undertook a quest to ascertain the requirements that, when fulfilled, would enable organizations to securely configure their web browsing infrastructure using a least privilege approach. We wanted to discover: a) Why organizations were not already implementing such an approach and b) What could be done to help facilitate implementation of such an approach. We started our quest by answering the following questions in the case of the three major web browsers: 1) Which security-relevant configuration options are available? 2) How similar or different are the available configuration options? 3) What kind of procedures and expertise are required to configure these available options? 4) How similar or different are the procedures and expertise required to configure the available options? 5) Which tools are available today, to help IT/OT system administrators, to configure web browsers using a highgranularity and least privilege approach? 6) What functionality should tools or configuration languages expose to facilitate a high-granularity web browser configuration approach?

We believe we have answered all these questions with our research and are moving forward with the development of prototype environments that would enable high-granularity and least privilege web browser and application-level secure configurations. The overall research project, *HiFiPol*, began with an initial exploration into questions 1-5 in Bhandari's thesis [2],[3] and for question 6 within Jillepalli's thesis [4],[5],[6]. This paper expands those earlier publications adding new results as detailed in following subsection.

1.3 The Contributions of this Article

The contributions of this article are as follows: First, we analyze and categorize secure browsing configuration options across the three major web browsers: Google Chrome (v37.0.2062), Internet Explorer (v10.0.9200), and Mozilla Firefox (v33.0.2). As of 03/19/2018, the latest versions of these three web browsers are: Internet Explorer: v11.0.56, Google Chrome: v65.0.3325, and Mozilla Firefox: v59.0.1. However, for the configuration options discussed in **Tables 5, 6, 7, 8, and 9**, we have verified that no significant change in the syntax and semantics has occurred for all three web browsers, between the versions that we analyzed and current versions. The high syntactic and semantic variability of web browser configuration options across all three major web browsers is still a major roadblock in the implementation of a secure browsing ecosystem for most organizations. Second, we describe, in detail, the similarities and differences between available configuration options across these three browsers. Third, we developed a tool named *Open Browser GP* that can remotely configure security related settings, based on policy, across Google Chrome, Internet Explorer, and Mozilla Firefox. Fourth, we propose the need to develop a set of common settings and a common language for web browser configurations.

This article is an expanded version of a conference paper [3] published by IEEE. The differences between the conference paper and this article are as follows: (1) performed a policy-based analysis of configuration options in three major web browsers and reported the results of this analysis in Section 6 and in **Table 9**; (2) discussed the results and implications of our findings in Section 7; (3) revised and enhanced the semantic and syntactic analysis and reported the new results in Sections 4, 5 and in **Tables 3, 4, 5, 6, 7, and 8**; (4) expanded the description regarding the development of the *Open Browser GP* tool in Sub-Section 8.1; (5) expanded the description of the categorization for configuration options and scope of our security configuration analysis in Section 3; (6) verified that the configuration options being discussed in the article are valid in the current versions of the three web browsers in Section 3; and (7) expanded the reasoning behind our argument concerning the necessity of a common language, syntactic, and semantics amongst web browsers, and added a policy-based specification case study in Section 9 and in **Listings 2 and 3**.

1.4 Outline of this Article

The remainder of this article is organized as follows: Section 2 provides information on currently available tools for remotely configuring web browsers. It also presents some of the limitations associated with these solutions. Section 3 describes our setup and classification of available configuration options across Internet Explorer, Google Chrome, and Mozilla Firefox. Sections 4, and 5 describe the process and results of our syntactics-based and semantics-based analysis of security-related configuration options across the three web browsers. Section 6 demonstrates the process of policy-based configuration of web browsers. We also make an observation regarding the complexity of such a policy-based configuration process. Section 7 discusses the results and implications of the analyses conducted in Sections 4, 5, and 6. Section 8 describes the design, development process, advantages, and limitations of a user-friendly, multiweb-browser, and multi-platform tool named as *Open Browser GP*. For the remainder of this paper *GP* should be understood as a reference to Group Policy. Section 9 discusses the need to create and utilize common settings and a common configuration language across all web browsers and all operating systems. Section 10 discusses another web browser configuration analysis work, and the differences between that work and ours. Section 11 briefly describe the ongoing and near term future work, respectively. Conclusion to the article, declarations, and references follow.

2. CURRENT TOOLS FOR REMOTE WEB BROWSER CONFIGURATION

We put forth information about the pros and cons of the currently available tools for remote web browser configuration, which are: Microsoft Group Policy, FreeIPA, and Quest KACE (previously known as Dell KACE). We also briefly discuss the pros and cons of these tools, with respect to the problem discussed in Subsection 1.1.

2.1 Microsoft Group Policy and SmartScreen

Microsoft Windows line of operating systems are equipped with Administrative Templates, which are a popular and useful way of configuring applications, including web browsers, in Microsoft Windows client systems. Files with .*ADMX* and .*ADML* extensions are Administrative Template Files which were introduced by Windows Vista and Windows 2008 server. .*ADML* files are referenced by .*ADMX* files. Where, .*ADML* files contain textual descriptions of the configurations and .*ADMX* files contain the actual configurations [7]. Group Policy and Group Policy Objects are used by Windows Server, to remotely configure multiple client systems that are joined to a Windows Domain Server. This method is used by organizations ranging from medium scale private companies to large scale government organizations to configure multiple Windows client operating systems by using a Windows Server infrastructure called Active Directory. The Local Group Policy Editor can also be used to configure web browser settings in client operating systems, which have the “Local Group Policy Object Editor” installed [8].

The .*ADMX* and .*ADML* files for Internet Explorer (IE) are already available in the Policy Definitions folder in all Windows operating systems and these files also receive an update whenever IE is updated. The .*ADMX* and .*ADML* files for Google Chrome (GC) are not installed by default in Windows. They have to be acquired from GC’s developer website [9], [10]. Mozilla Firefox (MF) does not support Group Policy functionality natively (as of 03/19/2018). It requires a third party add-on such as *GPO for Firefox,* to configure administrative templates and connect Firefox to the Windows registry [11],[12]. This means that the .*ADMX* and .*ADML* files for MF are not created and maintained by Mozilla. In addition, the installation process of a third party add-on involves manual configuration of different entries in Mozilla Firefox, which is a complex task requiring a high level of expertise and experience with configuring MF. However, as of 03/19/2018, even with the third party add-ons, MF doesn't have many settings which are compatible with Group Policy.

Active Directory (AD) is useful to manage and configure Windows machines only. In addition, even when using AD, system administrators have to manually configure each web browser setting to achieve the same configuration option in all web browsers. For example, even though all web browsers support the JavaScript Enable/Disable configuration setting, the web browsers use different configuration option locations, names, and values. Therefore, the process of configuration of all browsers using Group Policy usually requires training and a level of expertise.

Starting from Microsoft Windows 8, the Windows line of operating systems started using SmartScreen, which is a cloud-based anti-phishing white-list. SmartScreen is also embedded into various Microsoft applications: web browsers (Internet Explorer 7 and above and Microsoft Edge), e-mail suite (Outlook), and Exchange Server. SmartScreen is effective in blocking known, reported phishing resources. However, it is a white-list and not a granular, least-privilege configuration tool.

2.2 FreeIPA

Red Hat, Inc. created, develops, and sponsors an opensource project called the FreeIPA [13]. FreeIPA is used to emulate Microsoft's Active Directory functionality in Linux and other Unix-like computers. When it comes to implementation of tailored security configurations for web browsers, FreeIPA also has limitations similar to Active Directory. Two such limitations are as follows: FreeIPA does not map common configuration options across different web browsers. Which implies that system administrators have to manually configure common configuration options for each web browser individually. FreeIPA works across Linux- and/or Unix-based systems and requires Active Directory to configure browser settings in Windows client systems.

2.3 Quest KACE

The Quest KACE (previously Dell KACE) Systems Management Appliance is an appliance and software tool for remote management of client devices in an enterprise. It is the only tool we found that can remotely manage a multi-platform IT infrastructure. It provides: Inventory and IT asset management, systems deployment, patch management and security, and service desk. This utility allows system administrators to create a client-server infrastructure, to deploy software, run scripts, and manage security patches. This utility also provides a secure DNS infrastructure and a user-friendly GUI interface for systems administrators [14].

Nonetheless, akin to every remote management and configuration tool known to the authors, Quest KACE's remote deployment of configurations is based on running scripts for configurations. These scripts are manually developed by system administrators. To implement high-granularity configuration options on multiple web browsers across multiple platforms, system administrators would need to manually create separate scripts for each browser and each configuration group. This implies that organizations would still need system administrators with in-depth expertise regarding the naming and semantics of all configuration options for all web browsers used within their organization.

These drawbacks provided us with the motivation to: a) analyze security configuration options across Google Chrome (v37.0.2062), Internet Explorer (v10.0.9200), & Mozilla Firefox (v33.0.2) (Sections 4, 5, and 6), and subsequently: b) develop a knowledge- and policy based multi-browser and multi-platform browser configuration tool (Section 8).

3. Data Sources and Scope of Analysis

We present details regarding the setup, on which we carried out our analysis. First, we layout details regarding our analysis scope. Second, we categorize all available configuration options into four categories. To make the scope of our analysis manageable and realistic, we chose to work on three web browsers. They are: Internet Explorer (IE), Google Chrome (GC), and Mozilla Firefox (MF). These were the three most popular web browsers for desktops and laptop devices, as reported by StatCounter [15]. The versions we used for our analysis were: Internet Explorer: v10.0.9200, Google Chrome: v37.0.2062, and Mozilla Firefox: v33.0.2.

Table 1. Browser Configuration Option Categories. Table adapted from [2],[3]. GUI: Graphical User Interface and SEC: Security.

Browser Settings Categorization	
Category Name	**Category Description**
GUI-SEC	GUI and Security setting
GUI-NOSEC	GUI and Non-Security setting
NOGUI-SEC	Non-GUI and Security setting
NOGUI-NOSEC	Non-GUI and Non-Security setting

Table 2. Number of Analyzed Security Configuration Settings per Major Web Browser and per Category. Based on data reported in [2],[3].

Category Name	**Number of Settings**		
	IE	**GC**	**MF**
Total	*1191*		
Sub-total	*770*	*158*	*263*
GUI-SEC	260	058	047
GUI-NOSEC	300	078	147
NOGUI-SEC	210	022	069
NOGUI-NOSEC	*Not analyzed*		

Table 3. Number of Syntactically (and Semantically) Similar Configuration Options across the Three Web Browsers per Category. These 13 settings are described in **Tables 5 and 6**.

Category Name	**Number of Settings - All Web Browsers**
Total	*13*
GUI-SEC	6
GUI-NOSEC	6
NOGUI-SEC	1
NOGUI-NOSEC	*Not analyzed*

Table 4. Number of Semantically (but not Syntactically) Similar Configuration Options across the Three Web Browsers per Category. These 4 settings are described in **Tables 7 and 8**.

Category Name	**Number of Settings - All Web Browsers**
Total	*4*
GUI-SEC	2
GUI-NOSEC	1
NOGUI-SEC	1
NOGUI-NOSEC	*Not analyzed*

Web browsers have several settings and options available to be configured. Since security settings are our focus for this article, we first classified all available configuration options. See **Table 1** for a list of categories. We classified Machine-level configuration options for each of three web browsers into a Cartesian product of (1) GUI or NOt-GUI and (2) SECurity or NOt-SECurity related. Machine-Level configuration options refer to options that are applicable to a device rather than to the individual users of that device. Results of this classification are shown in **Table 2**. To create this classification, we analyzed 770 Internet Explorer, 158 Google Chrome, and 263 Mozilla Firefox configuration options. We did not consider configuration options that were in the NOSECurity category.

We used Python scripts to extract the data and the predetermined configuration option names from *.ADMX* and *.ADML* files of IE and Chrome. Firefox configuration options were extracted by processing the entries described in the Mozilla's database describing Firefox's internal configuration options (about:config). In this article, we will only be looking at excerpts from the extracted data (**Tables 5, 6, 7, 8, and 9**). To access the entire data tables we refer the reader to Bhandari's thesis [2].

```
<command>
<name>IE_Registry_Configurations</name>
<executable>IE_Registry_Configurations.bat</executable>
<expect></expect>
<timeout_allowed>no</timeout_allowed>
</command>
<active-response>
<command>IE_Registry_Configurations</command>
<location>local</location>
</active-response>
```

Listing 1. An excerpt of the configuration command additions required to be made to the vanilla OSSEC configuration file in the server system.

Table 5. The 13 Syntactically (and Semantically) Similar Configuration Options amongst the three Web Browsers, Part A. *N/A* = Not Available. The syntactic similarity can be observed across the horizontal axis, i.e., each row lists a fuctionality and its' corresponding configurations across three web browsers, which are syntactically similar.

Global Configuration Name	**Internet Explorer Configuration Name**	**Google Chrome Configuration Name**	**Mozilla Firefox Configuration Name**
Global Configuration Description			
Cache_Szie_Setting	DefaultDomainCache LimitInMB	DiskCacheSize	Cache_Size
Used to set the cache size in IE, GC, and MF. It corresponds to (Set default storage limits for websites) in IE, (Set disk cache size in bytes) in GC and (Set Browser Cache Size) in MF.			
DNSPrefetching	*N/A*	DnsPrefetching Enabled	DNS
Used to activate or deactivate DNS prefetching. If we enable this setting DNS prefetcing is activated and deactivated if we disable this setting. It corresponds to (Enable network prediction) in Google Chrome and (Disable DNS Prefetching) in Mozilla Firefox.			
DefaultBrowser Check	*N/A*	DefaultBrowser SettingEnabled	Check_Default _Browser
Configures to check whether the browser is the default in a given system in GC and MF. It corresponds to (Set Chrome as Default Browser) in GC and (Check if Firefox is the default browser) in MF.			
DeveloperTools	DisableDeveloper Tools	DeveloperTools-Disabled	*N/A*
Configures whether a browser allows or disallows access to developer tools. If we enable this configuration option developer tools cannot be accessed by a user. It corresponds to (Turn off Developer Tools) in Internet Explorer and (Disable Developer Tools) in Google Chrome.			
Display Images	*N/A*	DefaultImages Setting	Permission_Images
Configures whether we can display images or not while pages load in Google Chrome and Mozilla Firefox Browsers. It corresponds to (Default images setting) in Google Chrome and (Allow or disallow images to load) in Mozilla Firefox.			
Download DirectorySetting	*N/A*	DownloadDirectory	Download_Dir
Used to set the download directory of the browser. It corresponds to (Set download directory) in Google Chrome and (Set Download Directory) in Mozilla Firefox.			
Geo Location Setting	GeoLocationDisable	DefaultGeoLocation Setting	Geo_location
Configures whether a browser can track GEO location of the system. It corresponds to (Turn off browser geolocation) in Internet Explorer, (Default geolocation setting) in Google Chrome and (Setting to enable or disable GEO location) in Mozilla Firefox.			

Table 6. The 13 Syntactically (and Semantically) Similar Configuration Options amongst the three Web Browsers, Part B. N/A = Not Available. The syntactic similarity can be observed across the horizontal axis, i.e., each row lists a fuctionality and its' corresponding configurations across three web browsers, which are syntactically similar.

Global Configuration Name	**Internet Explorer Configuration Name**	**Google Chrome Configuration Name**	**Mozilla Firefox Configuration Name**
Global Configuration Description			
HomePage	*N/A*	HomepageLocation	Home_Page
Configures home page of Google Chrome and Mozilla Firefox Browsers. It corresponds to (Configure the home page URL) in Google Chrome and (Home Page) in Mozilla Firefox.			
JavaScript	IZ_PolicyActive Scripting_1	DefaultJavaScript Setting	JavaScript-Enabled
Configures whether Javascript is enabled or disabled. It corresponds to (Allow active scripting) in Internet Explorer, (Default JavaScript setting) in Google Chrome and (Setting to enable or disable Javascripts) in Mozilla Firefox.			
Max_Proxy_Setting	*N/A*	MaxConnections -PerProxy	Max_Proxy
Used to set the maximum number of connections per proxy in GC and MF. It corresponds to (Maximal number of concurrent connections to the proxy server) in GC and (Set maximum number of connections to proxy server) in MF.			
PopUpBlocker	IZ_PolicyBlockPopup Windows_1	DefaultPopupsSetting	*PopUpsDisabled*
Configures if pop-ups are allowed or disallowed. It corresponds to (Use Pop-up Blocker) in Internet Explorer, (Default popups setting) in Google Chrome and (Setting to configure Pop-ups) in MF.			
PrintSetting	NoPrinting	PrintingEnabled	*N/A*
Configures whether we can display images or not while pages load in GC and MF Browsers. It corresponds to (Default images setting) in GC and (Allow or disallow images to load) in MF.			
SafeBrowsing Setting	*N/A*	SafeBrowsingEnabled	Safe_Browsing
Used to activate or deactivate safe browsing to detect phishing malware while loading websites. It corresponds to (Enable Safe Browsing) in GC and (Enable Safe Browsing) in MF.			

Table 7. The 4 Semantically (but not Syntactically) Similar Configuration Options amongst the three Web Browsers, Part A. N/A = Not Available. The semantic similarity can be observed across the horizontal axis, i.e., each row lists a fuctionality and its' corresponding configurations across three web browsers, which are syntactically (but not semantically) similar.

Global Configuration Name	**Internet Explorer Configuration Name**	**Google Chrome Configuration Name**	**Mozilla Firefox Configuration Name**
Global Configuration Description			
CrashRestore	DisableACRPropmt	*N/A*	Crash_Restore
Allows us to configure the browser to prompt when the browser tries to recover from any crash sessions. It corresponds to (Turn off Automatic Crash Recovery) in IE and (Crash Recovery) in MF.			
Restore_Previous_ Session	ContinuousBrowsing	RestoreOnStartup	Start_Up_Pages
Configures the browser such that, it restarts with the web pages from last browsing session. It corresponds to (Start Internet Explorer with tabs from last browsing session) in Internet Explorer, (Action on startup) in Google Chrome and (Set how the browser should start) in Mozilla Firefox.			

Table 8. The 4 Semantically (but not Syntactically) Similar Configuration Options amongst the three Web Browsers, Part B. N/A = Not Available. The semantic similarity can be observed across the horizontal axis, i.e., each row lists a fuctionality and its' corresponding configurations across three web browsers, which are syntactically (but not semantically) similar.

Global Configuration Name	**Internet Explorer Configuration Name**	**Google Chrome Configuration Name**	**Mozilla Firefox Configuration Name**
Global Configuration Description			
Plugin_Prompt_ Setting	IZ_PolicyRun ActiveX_Controls_1	DefaultPluginsSetting	Plugin_Prompt
Configures whether a browser should run plugins only after click or run plugins automatically. It corresponds to (Run ActiveX controls and plugins) in Internet Explorer, (Default plugins setting) in Google Chrome and (Setting to run plugins only on click) in Mozilla Firefox.			
Plugin_Setting	IZ_PolicyRunActiveX Controls_1	DefaultPluginsSetting	*N/A*
Configures whether a browser allows or disallows plugins. It corresponds to (Run ActiveX controls and plugins) in Internet Explorer and (Default plugins setting) in Google Chrome.			

Table 9. The policy-based configuration options across three web browsers. DEF = Placeholder for white-listed web domains and N/A = Not Available.

Policy Name	**Internet Explorer Configuration Name**	**Google Chrome Configuration Name**	**Mozilla Firefox Configuration Name**
Policy Description			
Selective Enabling of JavaScript	IZ_PolicyActiveScripting_0 IZ_ZoneSettingValue_2 IZ_ZoneSettingSites_DEF IZ_ZoneActiveScripting_1	DefaultJavaScript-Setting:0 AllowJavaScriptForUrls:DEF	*N/A* natively. Requires 3rd party add-ons
Enables loading of JavaScript only for some web domains and disables it for all others. This is also known as JavaScript white-list.			
Selective Enabling of Images	IZ_PolicyShowPictures_0 IZ_ZoneSettingValue_2 IZ_ZoneSettingSites_DEF IZ_ZoneShowPictures_1	DefaultImages-Setting:0 AllowImages-ForUrls:DEF	*N/A* natively. Requires 3rd party add-ons
Enables loading of Images only for some web domains and disables it for all others. This is also known as Image White-list.			
Selective Enabling of Cookies	IZ_PolicyDisableCookies_0 IZ_ZoneSettingValue_2 IZ_ZoneSettingSites_DEF IZ_ZoneDisableCookies_1	DefaultCookies-Setting:0 AllowCookies-ForUrls:DEF	*N/A* natively. Requires 3rd party add-ons
Enables loading of Cookies only for some web domains and disables it for all others. This is also known as Cookie White-list.			

4. Syntactic Analysis of Configuration Options in Web Browsers

We present criteria, process, and results for our syntactic analysis of security configuration options. The syntactic similarity amongst configuration options of three web browsers, was determined by checking and matching names of machine-level configuration options. For example, let's consider a scenario where we need to check the syntax of configuration option corresponding to JavaScript in all three web browsers. By observing the names of each configuration option, we can find that the JavaScript configuration option is similarly named in all three web browsers. The functionality is available as: *IZ_PolicyActiveScripting_1* in Internet Explorer, *DefaultJavaScriptSetting* in Google Chrome, and *JavaScriptEnabled* in Mozilla Firefox.

In our matching, we found that only 13 configuration options were syntactically similar (**Table 3**). These 13 configuration options are displayed in **Tables 5 and 6**. Amongst the 13 options, only 4 configuration options are syntactically similar between all three web browsers and the remaining 9 are similar between two. Note that these are all of the semantically similar configurations, not just the security-related ones.

Each row of **Tables 5, 6, 7, and 8** consists of two sub-rows. The first column of the first sub-row contains the global configuration option name which has been devised by us. We devised these names to serve as a blanket term for a browser functionality, across all three web browsers. The second, third, and fourth columns of the first sub-row present the global configuration name's equivalent configuration option names as associated with: Internet Explorer, Google Chrome and Mozilla Firefox respectively. The second subrow of each row contains a description of the respective configuration option. We created these descriptions manually, for each common configuration option. The term *N/A* indicates that the particular configuration option is not available in the corresponding browser [2].

5. Semantic Analysis of Configuration Options in Web Browsers

In this section, we present criteria, process, and results for our semantic analysis of security configuration options. In addition, an observation regarding multiple semantically similar functionalities being represented by a same configuration option is made. We also explain the relationship between syntactical and semantic commonality.

The semantic commonality amongst configuration options of three web browsers was determined by checking and matching functionality of machine-level configuration options. For example, let's consider a scenario where we need to check the semantics of configuration option corresponding to the functionality of being able to restore last browsing session in all three web browsers. By observing the functionality of each configuration option, we can find that this functionality can be achieved in all three web browsers. The functionality is available as: *ContinuousBrowsing* in the Internet Explorer, *RestoreOnStartup* in the Google Chrome, and *Start_Up_Pages* in the Mozilla Firefox.

In our matching, we found out that only 4 configuration options were semantically similar, as seen in **Table 4**. These 4 configuration options are displayed in **Tables 7 and 8**. Out of the 4 options, only 2 configuration options are syntactically similar between all three web browsers and the remaining 2 are similar between any two. Note that these are all of the semantically similar configurations, not just the securityrelated ones.

5.1 Multiple Functionalities, One configuration Option

Our semantic analysis produces an interesting type of commonality: multiple semantically similar functionalities are represented by only one configuration option. As observed in **Table 8**, both *Plugin_Prompt_Setting* and *Plugin_Setting* functionalities are controlled by the same configuration option in Internet Explorer and Google Chrome. Which are: *IZ_PolicyRunActiveX-Controls_1* and *DefaultPluginsSetting* options, respectively. Whereas, out of the two functionalities, only the first one can be implemented in Mozilla Firefox. This phenomenon goes on to compound the already complex problem of policy-based and functionality-based secure configuration of multiple web browsers in an enterprise.

An interesting observation here is that: by virtue of working principle (nature), syntactic similarity is a proper subset of semantic similarity. Which means: all the configuration options that are syntactically similar, are also semantically similar. This would mean that the total number of semantically similar configuration options is 17 (14 syntactic + 3 semantic). Note that these are all of the semantically similar configurations, not just the security-related ones. For syntactic and semantic similarity definitions, read second paragraphs of Sections 4 and 5 respectively. All semantically common configuration options are not syntactically common.

6. Policy-based Analysis of Configuration Options

We present an analysis of the configuration options for three web browsers, when a configuration is initiated via implementation of policies.

Let us consider the scenario – an organization named Acme has the following organizational policy: All systems in the organization must have three dedicated web browsers for accessing one of the following three web domains. *Acme* domain/Intranet (trusted), Internet (trusted only for some), and Internet (untrusted). A web browser assigned to one particular web domain cannot access any other web domains. As of 03/19/2018, there is no native configuration option to restrict browsing completely for a list of web domains. Therefore, to restrict a web browser's browsing access for certain web domains, system administrators have to configure the web browsers such that no functionality works, apart from simple text pages or trust and use third-party white-list add-ons.

For the sake of this article, let us take a look at three configuration policies: 1) Selective enabling of JavaScript, 2) Selective enabling of Images, and 3) Selective enabling of Cookies. The detailed description of these policies and their corresponding configurations for the three web browsers are given in **Table 9**.

Each row of **Table 9** consists of two sub-rows. The first column of the first sub-row contains the policy name which has been devised by us. We devised these names to serve as a blanket term for a browser functionality, across all three web browsers. The second, third, and fourth columns of the first sub-row present configuration option required to implement the policy as associated with: Internet Explorer, Google Chrome and Mozilla Firefox respectively. The second sub-row of each row contains a description of the respective policies and their implementation across the three web browsers. As seen in **Table 9**, we can discern that the process of configuring multiple web browsers, to implement a policy, is a complex task. To ease this process, a proof-of-concept tool is presented in Section 8.

7. Results and Implications

A syntactic-similarity and semantic-similarity analysis of configuration options across Internet Explorer, Google Chrome, and Mozilla Firefox was discussed in Sections 4 and 5. This analysis helps in understanding the relationship between security-settings in these web browsers. We found that only 17 configuration options out of 1,191 were similar across these three web browsers. Our finding sheds light on the existing disparity between the names and semantics of security configuration options within the three major web browsers. One direct implication of this disparity is the high learning curve for system administrators to learn about all configuration options in these web browsers.

The analysis presented in Section 6 helps in understanding the difficulty of implementing secure web browser policies at the enterprise-level. A direct implication of such finding is that the current configuration-based approach paired with the disparity of names and semantics of the security configuration options makes it extremely hard to implement a security policy across the enterprise.

To streamline the process of implementing web browser security policies we created a prototype version of a user-friendly, multi-web-browser, and multi-platform toolset named *Open Browser GP*. We describe this toolset in Section 8. In Section 9, we advocate that a common configuration language amongst all major web browsers is needed.

8. Security Orchestration with Open Browser GP

In this section, we present Open Browser GP: a ‘proof-of-concept’ tool developed to help ease the complexity of configuring web browser security options in a technologically diverse enterprise. Open Browser GP is an enterprise, multiplatform, and a multi-web-browser policy-based configuration tool.

8.1 Development of Open Browser GP Tool

Firstly, using python scripts, we extracted configuration options from: a) ADMX files of Internet Explorer and Google Chrome, and b) vanilla *about:config* entries of Mozilla Firefox. We compiled all the extracted configuration options into an Erlang database. Secondly, we wrote batch scripts to automate the process of configuring three web browsers: Internet Explorer, Google Chrome, and Mozilla Firefox. Thirdly, we setup OSSEC [16] server-client infrastructure to transfer and execute the batch scripts in client machines. Finally, we created a web-server based GUI using YAWS [17], so that users can provide configuration input, thereby triggering the corresponding configuration batch scripts. All these steps resulted in the creation of a tool, which we named Open Browser GP (OBGP).

While installing OSSEC, we have to enable *Active Response* functionality. Also, IP (Internet Protocol) addresses of the client systems should be added to the active-response whitelist. OSSEC has a provision for transferring batch scripts from a server system to a client system and executing these batch scripts in the client system. After installation, to execute configuration-automation batch scripts in the clients, we have to add some XML tags into the configuration file of OSSEC. All of these XML tags are displayed in **Listing 1**. In context of **Listing 1**, the `<executable> </executable>` tag is used to execute the files defined within the tag. The `<expect> </expect>` tag is used to provide any additional input parameters to be executed along side `<executable> </executable>` tags. The `<timeout_allowed> </timeout_allowed>` tag is used to specify the waiting time before stopping the execution, in case of any interference. All the other tags are obvious and self-explanatory. After saving these new additions to the configuration file, we had to restart OSSEC server.

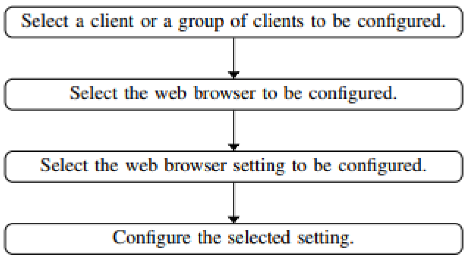
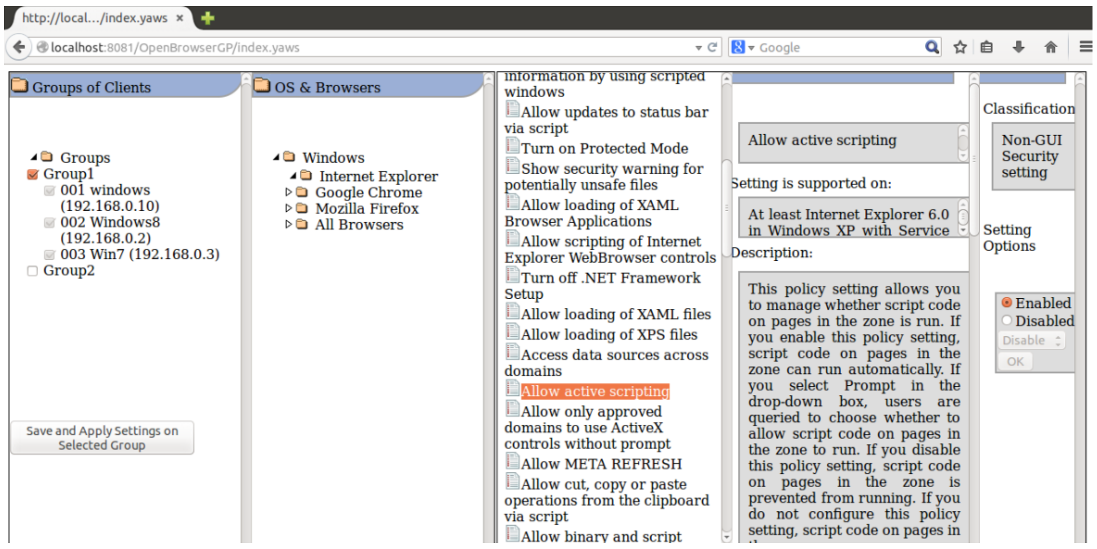

Select a client or a group of clients to be configured.
Select the web browser to be configured.
Select the web browser setting to be configured.
Configure the selected setting.

Fig. 1. A flowchart representing steps involved in maneuvering through OBGP's web interface.

8.2 Open Browser GP Tool Web Interface

For easier usage, we designed a single-screen web-based interface for OBGP. **Figs. 1 and 2** display the interface of OBGP. **Fig. 2** illustrates the five-sectional division of the tool. The five sections are as follows:


http://local.../index.yaws
localhost:8081/OpenBrowserGP/index.yaws
Google

Groups of Clients

Groups
Group1
001 windows (192.168.0.10)
002 Windows8 (192.168.0.2)
003 Win7 (192.168.0.3)
Group2

Save and Apply Settings on Selected Group

OS & Browsers

Windows
Internet Explorer
Google Chrome
Mozilla Firefox
All Browsers

information by using scripted windows
Allow updates to status bar via script
Turn on Protected Mode
Show security warning for potentially unsafe files
Allow loading of XAML Browser Applications
Allow scripting of Internet Explorer WebBrowser controls
Turn off .NET Framework Setup
Allow loading of XAML files
Allow loading of XPS files
Access data sources across domains
Allow active scripting
Allow only approved domains to use ActiveX controls without prompt
Allow META REFRESH
Allow cut, copy or paste operations from the clipboard via script
Allow binary and script

Allow active scripting

Setting is supported on:

At least Internet Explorer 6.0 in Windows XP with Service

Description:

This policy setting allows you to manage whether script code on pages in the zone is run. If you enable this policy setting, script code on pages in the zone can run automatically. If you select Prompt in the drop-down box, users are queried to choose whether to allow script code on pages in the zone to run. If you disable this policy setting, script code on pages in the zone is prevented from running. If you do not configure this policy setting, script code on pages in

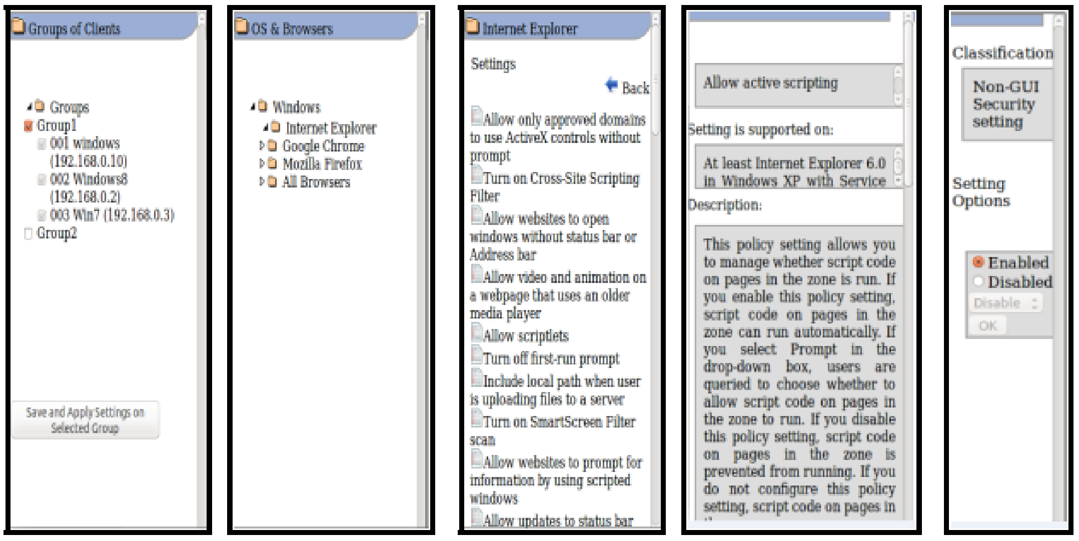
Classification

Non-GUI Security setting

Setting Options

[✓ Enabled] Disabled
Disable
OK

Fig. 2. Open Browser GP: A Multiplatform and Multibrowser Policy Configuration Tool. Figure re-used with permission, from a previous article [3].


Groups of Clients

Groups
Group1
001 windows (192.168.0.10)
002 Windows8 (192.168.0.2)
003 Win7 (192.168.0.3)
Group2

Save and Apply Settings on Selected Group

OS & Browsers

Windows
Internet Explorer
Google Chrome
Mozilla Firefox
All Browsers

Internet Explorer

Settings

Back

Allow only approved domains to use ActiveX controls without prompt
Turn on Cross-Site Scripting Filter
Allow websites to open windows without status bar or Address bar
Allow video and animation on a webpage that uses an older media player
Allow scriptlets
Turn off first-run prompt
Include local path when user is uploading files to a server
Turn on SmartScreen Filter scan
Allow websites to prompt for information by using scripted windows
Allow updates to status bar

Allow active scripting

Setting is supported on:

At least Internet Explorer 6.0 in Windows XP with Service

Description:

This policy setting allows you to manage whether script code on pages in the zone is run. If you enable this policy setting, script code on pages in the zone can run automatically. If you select Prompt in the drop-down box, users are queried to choose whether to allow script code on pages in the zone to run. If you disable this policy setting, script code on pages in the zone is prevented from running. If you do not configure this policy setting, script code on pages in

Classification

Non-GUI Security setting

Setting Options

[✓ Enabled] Disabled

Disable

OK

(a) Client Groups (b) OS and Browsers (c) Settings in Corresponding Browsers (d) Description of Browser Settings (e) Classifications and Browser Setting Options

Fig. 3. Individual Sections of the OpenBrowserGP Tool GUI: (a) Client Groups, (b) OS & Browsers, (c) Available Settings, (d) Description of Browser Settings, and (e) Browser Settings. Figure re-used with permission, from a previous article [3].

1. `Client Groups` section: Displayed in **Fig. 2(a)**. This module contains a group of client system(s) connected to the server. Tool-users can select any group to be configured, as per policy requirement.
2. `OS & Browsers` section: Displayed in **Fig. 2(b)**. This module contains all of client system's operating systems (OS) and web browsers. Tool-users can select any OS and/or web browser to be configured, as per policy requirement.
3. `Settings in Corresponding Web Browsers` section: Displayed in **Fig. 2(c)**. This module contains all the settings which are available in each web browser present in the client systems. Tool-users can select any setting(s) to be configured, as per policy requirement.
4. `Description of Settings` section: Displayed in **Fig. 2(d)**. This module contains the selected configuration option's display name, version-support information of the option, and a brief description of the selected setting. Tool-users are provided an option to hide this module by clicking on the Standard View option.
5. `Classification and Setting Options` section: Displayed in **Fig. 2(e)**. This module consists of the assigned category for the user-selected configuration option, and different possible options to configure the selected setting.

Fig. 3 is a flow chart representing steps involved in navigating through OBGP's web interface. **Fig. 2** represents a user enabling the configuration option `Allow Active Scripting` in `Internet Explorer` for `Group 1` cluster of clients. To achieve the screen seen in **Fig. 2**, a user has to load the web-based interface of OBGP tool. Next, the group of clients was chosen. The group of clients is the target of configuration option.

This selection opens up the `OS & Browsers` module, where a user can select the web browser to be configured. Selecting a web browser unlocks the `Settings in Corresponding Web Browsers` module, where a user can select the required setting to be configured. This selection unlocks the `Description of Settings` screen, and `Classification and Setting Options` screen, where a user can choose between the corresponding setting's options. To apply the changes selected through OBGP, a user must also click on `Save and Apply Settings on Selected Groups` button. Doing so will initiate the OSSEC remote deployment process, which deploys the new configuration to the selected group of client system(s).

8.3 Advantages and Limitations of Open Browser GP Tool

OBGP tool allows tool-users to configure options of three major web browsers using a single interface. Subsequently, OBGP users are then able to map similar security configuration options across different browsers. Such a mapping empowers OBGP users with the ability to update their configuration policies for web browsers. OBGP's web-based interface is similar to *Group Policy Management* utility (GPM) of Microsoft Windows, which makes it easy-to-learn for users who are familiar with GPM. OSSEC is used by OBGP to manage authentication of remote client connections. All technologies used in development of OBGP are open source in nature, providing flexibility with modification(s) and usage. OBGP categorizes every configuration option of three web browsers, which helps OBGP users in managing configurations.

OBGP is currently a prototype with the following limitations: a) It can only configure Internet Explorer (IE), Google Chrome, and Mozilla Firefox; b) It can only configure web browsers on Microsoft Windows operating system (version 7 and above); c) Due to OSSEC limitations, OBGP tool cannot configure User-level settings of IE; d) Even when not selecting a particular group of clients, the group will still receive all transmissions initiated by OSSEC; and e) Due to limitations of Mozilla Firefox, OBGP uses a third party add-on called *GPO for Firefox* to configure options in Firefox.

9. Toward a Standard for Web Browser Configuration Settings

In Sections 4 and 5, we found that only 17 configuration options out of 1191 were similar across three web browsers: Internet Explorer, Google Chrome, and Mozilla Firefox. In Section 6, we observed the complexity of implementing a new security policy in an organization with respect to changing policy-corresponding web browser configurations. Given the problem, and given the analysis results and observations, we argue that there are two current requirements: a) developing a single common language for configuring all web browsers, and b) an agreement between web browser developers regarding usage of common syntactics and semantics for configuration options.

When both the requirements are met, we get two positive outcomes: a) an enterprise's system administrators have to develop configuration options once, for each policy group, and deploy the options within their client systems; and b) future automated tools for a policy-based development and deployment of web browser configurations would not need to convert configurations into multiple disparate configuration languages. Thereby - greatly facilitating the development of such automated tools [5], [6]. For example XML, or other specification formats could be used to express the desired browser configurations. However, the verbose and nestingbased characteristics of XML, and other similar specification formats do not enable a easily reconfigurable policy-based configuration environment.

```
Policy: PID_01_000
{
  Description: "Enabling Plug-in func. universally."
  Rationale: "Staff requires plug-ins for designing ads."
  Status: "Enabled"
  Field: Plugins: "Enabled"
  ApplyToSys: "ALL"
  ApplyToApp: "APP.ALL"
}.

Policy: PID_01_001
{
  Description: "Disabling Plug-in automatic exec. func."
  Rationale: "Principle of least privilege."
  Status: "Enabled"
  Field: Plugins.Auto: "Disabled"
  ApplyToSys: "ALL"
  ApplyToApp: "APP.ALL"
  Parent: PID_01_000
}.

Policy: PID_01_002
{
  Description: "Enabling Plug-in automatic exec. func."
  Rationale: "Digital Rights Management."
  Status: "Enabled"
  Field: Plugins.Auto: "Enabled"
  ApplyToSys: "ALL"
  ApplyToApp: "APP.GC"
  Parent: PID_01_000
}.
```

Listing 2. HERMES specification corresponding to the three policies discussed in Section 9. Corresponding XML specification is given in **Listing 3**.

To achieve the goal of an easily reconfigurable policy-based configuration environment, we have developed a language called HERMES: a High-level, and Easily Reconfigurable Machine Environment Specification language. HERMES has been introduced and documented in a prior publication [6]. To see the difference between HERMES and XML with regards to policy-based specification usage, let us use configurations listed in **Table 8** and prepare three policies. 1) Plug-ins are allowed universally, across all the web browsers and systems of an enterprise. 2) Plug-ins are not allowed to run automatically, across all the web browsers and systems of the enterprise. 3) Plug-ins are allowed to run automatically only in Google Chrome browser, across all systems of the enterprise. In **Listings 2 and 3**, we express the above three policies in XML and HERMES specifications, respectively.

```xml
<policy>
  <id>PID_01_000</id>
  <description>
    Enabling Plug-in functionality universally.
  </description>
  <rationale>Staff requires</rationale>
  <status>enabled</status>
  <field><plugins>enabled</plugins></field>
  <applyToSys>ALL</applyToSys>
  <applyToApp>ALL</applyToApp>
</policy>

<policy>
  <id>PID_01_001</id>
  <description>
    Disabling Plug-in automatic exec. func. globally.
  </description>
  <rationale>Principle of least privilege.</rationale>
  <status>enabled</status>
  <field><pluginsAuto>disabled</pluginsAuto></field>
  <applyToSys>ALL</applyToSys>
  <applyToApp>ALL</applyToApp>
  <parentid>PID_01_000</parentid>
</policy>

<policy>
  <id>PID_01_002</id>
  <description>
    Enabling Plug-in automatic exec. func.
  </description>
  <rationale>Digital Rights Management.</rationale>
  <status>enabled</status>
  <field><pluginsAuto>enabled</pluginsAuto></field>
  <applyToSys>ALL</applyToSys>
  <applyToApp>GC</applyToApp>
  <parentid>PID_01_000</parentid>
</policy>
```

Listing 3. XML specification corresponding to the three policies discussed in Section 9. Corresponding HERMES specification is given in **Listing 2**.

10. Related Work

An analysis comparing security of Internet Explorer, Google Chrome, and Mozilla Firefox web browsers had been performed by Accuvant Labs in 2011 [18]. As a result of this analysis, Accuvant Labs proposed a set of security measures for increasing web browser's protection. In their analysis, Accuvant Labs comparatively measured effectiveness of some of the web browser security features to determine which web web browser was better in effectively implementing each individual security feature. These features include Sandboxing, URL Blacklisting, Plug-in Security, and Just-In-Time (JIT) Hardening.

The difference between Accuvant's analysis and our analysis is that Accuvant's analysis tested the correctness of each web browsers' security features. By contrast, our analysis identified syntactic and semantic differences between security configuration options across Internet Explorer, Google Chrome, and Mozilla Firefox. Our analysis also looked at the challenges and advantages of using a policy-based approach to secure web browsers in an enterprise. Our analysis also informed us that, currently, even when using leading remote client configuration tools, web-browser-specific configuration expertise is needed to remotely configure web browsers within an enterprise-level IT/OT infrastructure. On the front of a common configuration language for web brwosers, several languages have been created for specification of configurations. Some examples of such configuration specification languages are: High-level, Easy-to-use, Reconfigurable, Machine Environment Specification language (HERMES) [6], Security Policy Assertion Language (SecPAL) [19], Open Vulnerability and Assessment Language (OVAL), eXtensible Access Control Markup Language (XACML) [20], Extensible Configuration Checklist Description Format (XCCDF)[21], Margrave (focused on firewall analysis) [22], [23], and Security Policy Language (SPL) [24]. Some of these are part of the Security Content Automation Protocol (SCAP) specification [25].

11. Ongoing and Future Work

We foresee future work in three related areas of research. They are: a) development of common settings and a common configuration language across multiple web browsers - such a task is ideal to be undertaken by web browser developers; b) further validation, expansion, and development of Open Browser GP tool is needed to use it across all operating systems, including mobile ones; and c) the analyses of this article demonstrate that the server-based client configuration model can be expanded across multiple operating systems and multiple applications, such as: e-mail clients and firewalls - not just web browsers.

12. Conclusions

A central Microsoft Windows-based server is used by many system administrators in both: public and private sectors, to secure and configure web browsers of multiple Microsoft Windows client machines. This mechanism fails to configure web browsers across multiple operating systems. Such a failure decreases overall security, increases inconsistency, and creates a new avenue for security breaches. As a result, maintenance of critical/sensitive/private data in organizations, of both private and public sectors becomes a riskier endeavor. In this article, we present four contributions which help protect security of data management through web browsers. The four contributions are as follows:

The contributions of this article are not necessarily sufficient by themselves to mitigate every security vulnerability across web browsers. Nonetheless, the contributions of this article will enable system administrators to: a) categorically manage configuration options across Internet Explorer, Google Chrome, and Mozilla Firefox; and b) deploy changes/updates to the configurations as per policy change, using Open Browser GP tool. This article's results inform us that using certain processes, it is possible to configure the security settings of Internet Explorer, Google Chrome, and Mozilla Firefox web browsers, across multiple platforms of an organization. Two articles present a step-by-step tutorial of configuring multiple web browsers using Active Directory and Group Policy in Microsoft Windows [26].

References

[1] Web Application Security Consortium, "WebAppSec: threat classification," online, January 2010. [Visited: March 2018]. Article (CrossRef Link).

[2] Venkata Anirudh Bhandhari, "Analysis of Web Browser Security Policies and A Multiplatform and Multibrowser Browser Configuration Tool: Open Browser GP," *Master's thesis, University of Idaho*, August 2015. Article (CrossRef Link).

[3] Daniel Conte de Leon, Venkata Anirudh Bhandari, Ananth A. Jillepalli, and Frederic T. Sheldon, "Using A Knowledge-based Security Orchestration Tool to Reduce the Risk of Browser Compromise," in *Proc. of 2016 IEEE 07th Symposium Series On Computational Intelligence (SSCI-2016)*, December 2016. Article (CrossRef Link).

[4] Ananth A. Jillepalli, "HiFiPol:Browser - Securing the Web Browsing Ecosystem," *Master's thesis, University of Idaho*, May 2017. Article (CrossRef Link).

[5] Ananth A. Jillepalli and Daniel Conte de Leon, "An Architecture for a Policy-oriented Web Browser Management System: HiFiPol: Browser," in *Proc. of 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC-2016)*, June 2016. Article (CrossRef Link).

[6] Ananth A. Jillepalli, Daniel Conte de Leon, Stuart Steiner, and Frederick T. Sheldon, "HERMES: A High-level Policy Language for High-granularity Enterprise-wide Secure Browser Configuration Management," in *Proc. of 2016 IEEE 07th Symposium Series On Computational Intelligence (SSCI-2016)*, December 2016. Article (CrossRef Link).

[7] Jeremy Moskowitz, "Inside ADM and ADMX Templates for Group Policy," online, January 2008. [Visited: March 2018]. Article (CrossRef Link).

[8] Judith Herman, "Managing Group Policy ADMX files: A Step-by-step Guide," online, June 2007. [Visited: March 2018]. Article (CrossRef Link).

[9] Jack Stromberg, "Configuring Google Chrome via Group Policy," online, August 2013. [Visited: March 2018]. Article (CrossRef Link).

[10] National Security Agency - Central Security Service, "Deploying and Securing Google Chrome in a Windows Enterprise," online, October 2012. [Visited; March 2018]. Article (CrossRef Link).

[11] Redkitten Corporation, "How to Install a Firefox Add-on," online, June 2015. [Visited: March 2018]. Article (CrossRef Link).

[12] MozillaZine Knowledge Base, "about:config Entries," online, June 2015. [Visited: Mach 2018]. Article (CrossRef Link).

[13] Red Hat Inc., "FreeIPA," online, June 2015. [Visited: March 2018]. Article (CrossRef Link).

[14] Quest Software Inc., "KACE Systems Management Appliance: Patch Management and Endpoint Security," online, November 2016. [Visited: March 2018]. Article (CrossRef Link).

[15] StatCounter, "Statcounter global stats," online, December 2014. [Visited: March 2018]. Article (CrossRef Link).

[16] Daniel B. Cid, "Open Source Security [OSSEC]," online, June 2012. [Visited: March 2018]. Article (CrossRef Link).

[17] Zachary Kessin, "Building Web Applications with Erlang," 1st Edition, O'Reilly, Springfield, Missouri, June 2012.

[18] Accuvant Labs, "Browser Security Comparison - A Quantitative Approach," online, December 2011. [Visited: March 2018]. Article (CrossRef Link).

[19] Microsoft Research "Policy Language Research," online, June 2015. [Visited: March 2018]. Article (CrossRef Link).

[20] Tao Xie, Vincent Hu and Rick Kunh, "Testing and Verification of Security Policy," online, June 2016. [Visited: March 2018]. Article (CrossRef Link).

[21] National Institute of Standards and Technology, "XCCDF - The Extensible Configuration Checklist Description Format," online, September 2011. [Visited: March 2018]. Article (CrossRef Link).

[22] Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi, "The Margrave Tool for Firewall Analysis," in *Proc. of USENIX Large Installation System Administration Conference (LISA)*, San Jose, CA, November 07-12, 2010. Article (CrossRef Link).

[23] Dan Dougherty and Kathi Fisler and Shriram Krishnamurthi, "The Margrave Policy Analyzer," online, May 2015. [Visited: March 2018]. Article (CrossRef Link).

[24] J. B. Bernabe, J. M. M. Perez, J. M. A. Calero, J. D. J. Re, F. J. Clemente, G. M. Perez, and A. F. Skarmeta, "Security policy specification," *Network and Traffic Engineering in Emerging Distributed Computing Applications*, J. Abawajy, M. Pathan, M. Rahman, and M. M. Deris, Eds. Oxford: IGI Global, ch. 04, pp. 66–93, 2013. Article (CrossRef Link).

[25] National Institute of Standards and Technology, "Emerging Specification Listing," online, March 2018. [Visited: March 2018]. Article (CrossRef Link).

[26] Ananth A. Jillepalli, Daniel Conte de Leon, Stuart Steiner, Frederick T. Sheldon, and Michael A. Haney, "Hardening the client-side: A guide to enterprise-level hardening of web browsers," in *Proc. of 2017 IEEE 15th Dependable, Autonomic and Secure Computing (DASC-2017)*, Nov. 2017. Article (CrossRef Link).

Ananth A. Jillepalli, M.S. is a PhD Candidate at the University of Idaho. He received his Master of Science in Computer Science (Cybersecurity) degree from University of Idaho in 2017. His research interests include design and analysis of risk assessment methodologies in cybersecurity domain. Email: ajillepalli@ieee.org

Daniel Conte de Leon, Ph.D. is a cybersecurity researcher and educator at the University of Idaho. He received his PhD degree in Computer Science with a focus on security and safety of critical systems in 2006 from the University of Idaho. Dr. Conte de Leon performs research on the development of processes, methods, and tools for the design, development, configuration, and maintenance of safe and secure systems. Currently, with a focus on adequate and complete application of secure design principles, such as least privilege, to system architecture, design, implementation, and maintenance. He also works on developing innovative hands-on methods and tools for cybersecurity education and training. Dr. Conte de Leon's teaching experience comes from many years of teaching across the Computer Science curriculum. Email: dcontedeleon@uidaho.edu

Stuart Steiner, M.S. is a PhD Candidate at the University of Idaho, where he is actively researching web security. Stuart is a member of the University Idaho Center for Secure and Dependable Systems (CSDS). Stuart is also full-time faculty at Eastern Washington University where he primarily teaches lower-division programming courses. Stuart is an active member of the ACM and IEEE, where he actively recruits and promotes Women and Diversity in Computing. Email: ssteiner@uidaho.edu

Jim Alves-Foss, Ph.D. is a professor in the Department of Computer Science at the University of Idaho and director of the university's Center for Secure and Dependable Systems. His research interests include the design and analysis of secure distributed systems, with a focus on formal methods and software engineering. Alves-Foss received a PhD in computer science from the University of California, Davis. He is a senior member of IEEE and ACM. Email: jimaf@uidaho.edu.