Security policy framework and algorithms for web server content protection

Shadi Aljawarneh
School of Computing, Engineering and Information Sciences
Northumbria University
Newcastle upon Tyne, NE2 1XE, UK
Email: shadi.aljawarneh@unn.ac.uk

Christopher Laing
School of Computing, Engineering and Information Sciences
Northumbria University
Newcastle upon Tyne, NE2 1XE, UK
Email: christopher.laing@unn.ac.uk

Paul Vickers
School of Computing, Engineering and Information Sciences
Northumbria University
Newcastle upon Tyne, NE2 1XE, UK
Email: paul.vickers@unn.ac.uk

***Abstract*— A significant web security issue facing Internet users and organizations is the securing of web content against unauthorised tampering. Users must be comfortable with the security offered by web applications that sensitive web-based services. Some progress has been made in addressing the verification of web server content integrity, but current solutions are restricted by the limitations of the SSL protocol, the statelessness of HTTP, blind security mechanisms which is based on ad-hoc models, and difficulties with automatic code analysis. We present a web security real-time framework, a state protocol of web policies, and a number of particular algorithms that they can used to verify and protect the static and dynamic web content against unauthorised tampering. It is suggested that such a framework will offer a higher level of user confidence, and web service survivability.**

I. Introduction

A web security is still in abysmal state, where organizations that rely on information systems as the primary way to conduct sensitive transactions are being increasingly worried about their reputations when a web-based system is subverted [5], [12], [14]. The Computer Emergency Response Team (CERT) [3] has reported that there has been a dramatic increase (5990 in 2005 to over 8000 in 2006) in the number of security vulnerabilities which threaten web content. Furthermore, a study carried out by the Gartner Group has found that 75% of Internet assaults are targeted at the web applications level [2].

Static and dynamic web server content can be tampered with by changing (i) the style classes, (ii) referenced objects (images, audio, video, and other objects), (iii) the source code of the web page itself, and also by (iv) running malicious code on the server to intercept a requested page before the client receives it [5], [6], [10], [11], [14], [15]. Consequently, the integrity of web content can be violated on the server even though the communication channel between the server and client is secure.

Current research is generally more concerned with cryptographic rules, than with a security analysis of the system. In an attempt to remedy this, we focus on the integrity of data and do not delve into the confidentiality, and availability of data. We do not target authorization schemes and Access Control List (ACL), however, if the integrity of data is violated, the confidentiality and availability of data can be compromised. It should be noted that data integrity refers to the trustworthiness of information resources, thereby ensuring that only an authorized client can alter the data – unauthorized tampering may result in incorrect or malicious web application behaviour [5], [6], [14], [15].

The Secure Sockets Layer (SSL) protocol was developed to support the integrity of data transit [5], [6], [14], [15], and as such, it does not provide an absolute solution. Furthermore, several security tools are not capable of verifying the integrity of web content before a request or response enters the secure communication channel [14], [15]. This means they cannot distinguish between the original HTTP (Hyper Text Transfer Protocol) conversation, and the tampered-with HTTP conversation [5]. Therefore, there is no one-stop-shop protection method that can meet all the security requirements and design specifications of new or exiting web applications. As a result of the transparency of code at the web browser level, the following approaches can cause loss of data integrity: hidden fields and script manipulation, and analysis of validation code through reverse engineering techniques [10], [13], [14].

The HTTP Request-Response model can fail because web servers and browsers do not properly manage the statelessness of HTTP, in which each client request results in a new connection between a web browser and a web server [5], [6], [14]. Furthermore, because of the difficulty of code analysis, Jacobs and Malloy [7] suggest that software engineering principles (such as design, implementation and testing) should be integrated into web security to identify what a user and an organization need for every stage of this cycle - this can detect the vulnerabilities at each stage instead of processing them at the implementation stage. However, this integration may require the rebuilding of existing web applications - some web applications may be complex structures, consisting of multiple programming languages and imported binary components with little or no documentation.

This paper is organized as follows: Section 2 gives an overview of the existing approaches. Section 3 outlines a proposed integrity verification system. Section 4 discusses some issues related with security and performance. Finally, Section 5 draws conclusions and discusses possible future work.

II. Integrity Verification Approaches

We now discuss three more recent approaches that attempt to address one or more of these problems. First, Hassinen and Mussalo [6] propose a client-side encryption system to protect data integrity and user trust. The client encryption key is located on a client smart card or can be stored on the server and transferred over an HTTP connection. This system has two main requirements [6]: (i) it must be able to run on any major web browser, and (ii) without the need to install additional hardware or software.

However, integrity of data could be lost if this approach is adopted because Java applets can access the client's local file system. Thus, a criminal can replace the original signed applet with a faked applet to access the client's web content. Another potential weakness is the loss of the client smart card with its Personal Identification Number (PIN). Consequently, the whole web-based system can be compromised. Moreover,if a smart card is used, then the client machine requires a card reader and necessary drivers, which fails requirement (ii); no need to install additional hardware or software. Furthermore, applet and JavaScript methods can be bypassed. If this happens, the submitted values will be in plain text. Finally, existing web applications would require modification to implement this technique.

Sedaghat, Pieprzyk, and Vossough [14], [15] proposed a Dynamic Security Surveillance Agent (DSSA) tool on the server that automatically intercepts a request to verify the integrity of the requested page before the web server responds to the client. The verification uses the timestamp signature technique. However, tampering is still a potential problem because DSSA does not verify dynamic web content on the server.

Third, the Adaptive Intrusion-Tolerant Server system [16] consists of redundant servers, proxies that are positioned between web servers and client machines to verify the behaviour of servers. When a client request arrives a proxy leader intercepts the request, analyzes it, and forwards it to a number of application servers, depending on the enforced policy. Furthermore, a leader intercepts the responses to find a hash value for them. If they match, the leader sends a response to the user, otherwise, a report is sent to a monitoring component to take the correct action. However, it does not verify the integrity of referenced objects that are generated dynamically.

III. Web Content Verification (WCV) System

The integrity of web content can be compromised even though some security problems are discovered on web browsers and servers, while important security policies have yet to be answered in a systematic way. We are developing a server web content verification system to identify tampering. The WCV system comprises a number of components including mathematical models and a web security framework that enforces a set of web policies. We have two assumptions: First, SSL only secures the data in transit [5], [6], [14], [15]. SSL provides digital certificates to encrypt the data in transit against malicious coding and eavesdropping attacks. Second, the data resulting from user interaction on the client is untrustworthy. User input may contain malicious code that harms a web server or a backend database [13]. Furthermore, the client validation scheme can be violated. As result, all user data that is sent as a request to a server is untrustworthy. Therefore, we also assume a validation scheme operates on both sides (client and server) to validate the inputs of a request before it is processed on the web server. This means if the client validation scheme is subverted, there is still another validation scheme running on the server before processing a request.

A. Modelling Interaction Elements

We have formulated two separate mathematical models that aim to understand how to dynamically produce units of web content as shown in the prior work [1]. Moreover, another important aim is to simplify a method of analysis of a web site that contains a large number of elements and each web page can include a large number of interaction elements (i.e. HTML and non-HTML objects).

First, because the Bypass Test Strategy[1] [10] only deals with HTML inputs, we have formulated the new Client Interaction Elements (CIE) model, which extends Offut et al.'s HTML Input Units (IU) model [10],to deal with *every* interaction element (both HTML and non-HTML objects) in a web page. CIE has the following features: (i) formulating all the interaction elements (not just HTML input elements) (ii) adding other parameters to each interaction element to account for the user and data information required in sensitive web-based services such as banking, accounting, ticket reservations, etc. (iii) building a truth table to test our model, and (iv) a set of four usage patterns that are used to analyse and test the model.

Web applications contain static web files and programs that dynamically generate HTML/XHTML pages [10], [11], [14]. Each web page, whether a static web file or dynamically generated, can have zero or more interaction elements that make users interact with a web server via a web browser. An interaction element I is characterized by five parameters: $I = (IT, SP, D, T, U)$. IT is the type of interaction element that can be used for interacting with a web server. The data inputs D are sent to a server page SP for processing user requests. D is a set of ordered parameter-value pairs (p_i, v_i), where p_i is a parameter name, and v_i is the value of a data item (e.g. text box, selection list, etc) that can be assigned to p_i. T is the HTTP transfer mode (viz. GET, POST, DELETE, PUT, and HEAD). U is a user who interacts via a web browser to conduct HTTP conversations. A user U can legitimately access web content on designated directories of a web server through secure authentication and authorization schemes.

$$(1)\quad I = (IT, SP, D, T, U), D = \{(p_j, v_j \mid j = 0 \ldots n)\}$$

$$(2)\quad P = \begin{cases} x_i & \text{if } 0 < x \leq 1, \\ 0 & \text{if } x = 0. \end{cases} \text{, where } x_i = \frac{n_i}{N}$$

[1]It aims to create tests on the client for web applications that violate checks on user inputs.

Equation (1) is used to simplify the analysis of a web page through extracting the parameter values of the interaction elements in the target web page by the integrity verifier component of the proposed framework (see Figure 1) before processing a request. P in Equation (2) is the probability of generated units of web content through interacting with interaction element I in a web page, where n_i is the number of interactions of a single user with I during a fixed period of time t, and N is the number of interactions all users with I during the same t. If the interaction element exists on a web page, the number of possible units of web content can be generated depending on user inputs. The second case ($P = 0, x = 0$) means a static web page that does not contain any interaction elements or contains interaction elements without user inputs. For example, it is possible to have a fully plain text web page (e.g. one that contains a description of a place, or an information brochure).

The CIE model is analyzed and tested through building a truth table as shown of five input parameters: IT, SP, D, T, and U and is based on five usage patterns as shown in the pervious work [1] to represent dynamic web content. According to 5-parameter Karnaugh map [8] to identify the simplest set of inputs required to match rows, the simplified result is:

$$F(IT, SP, D, T, U) = (IT) \cap (SP) \cap (T) \quad (3)$$

F is a function of the 5 input parameters. As such, generation of dynamic web content units require interaction elements to have three parameters: IT, SP, and T (see Boolean Equation 3). Therefore, since we focus on integrity of data and are not concerned whether the user U is authorized (or not), and sent data D is validated (or not), we are only interested in extracting the values of IT, SP, and T.

Second, we formulate a model of web site elements from which we develop the new hashing scheme and the registration technique of web site elements through the derivation of mathematical equations (see Equations 4, 5, 6) that simplify the analysis of target web site elements. This model is based on a set of assumptions and specifications as shown in the previous work [1]. The notation key of the following equations is:WS denote a target web site, WP be a web page, O be a referenced object and let C denote the source code of a $WP, \forall C \in WP_i$, where $_i$ is the index of WP in web pages (n), m is the number of referenced objects in WS, and $Hash$ is the hash function.

$$WS = \{WP_0, WP_1, \ldots, WP_n\} : 0 \leq i \leq n \quad (4)$$

$$WP_i = \{(O_0, C_i), \ldots, (O_l, C_i)\} \quad (5)$$

$$WS = \{WP_{ji} \mid j = 0 \ldots m, \mid i = 0 \ldots n\} = \{(O_j, C_i) \mid j = 0 \ldots m, \mid i = 0 \ldots n\} \quad (6)$$

From the transitive property of equality we get:

$$Hash(O_i \in WP_1) = Hash(O_i \in WP_2)$$
$$Hash(O_i \in WP_2) = Hash(O_i \in WP_3)$$
$$\therefore Hash(O_i \in WP_1) = Hash(O_i \in WP_3)$$

Merkle [9] proposed a tree scheme of digital signatures [4] - which provides a different message to be signed for each a node in a tree. If two or more trees reference the same object in a node, then the signature of that node will be different in each tree. Therefore, our scheme is different from Merkle's in which shared referenced objects have the same signature even though they belong to different web pages. This will achieve balance between performance (minimising number of calculated signatures of each referenced object in a web site) and security.

B. Architecture of web security framework

The proposed WCV system will include a web security framework (Figure 1). This framework consists of a number of web-based components: web register, response hashing calculator, integrity verifier, and recovery. The web register component has two stages: monitoring and registration. The integrity verifier (manager) component mediates between the web server and client machines by managing the HTTP requests and responses via a state protocol that enforces a number of web policies that apply to the elements of a web-based system. The web policy outlines who is responsible for maintaining each component and when the policy is enforced. The response hashing calculator component calculates the hash value of each HTTP response on the web server before sending the response to the integrity verifier for further processing. Finally, the recovery component recovers the marked data, which is stored if the action of the enforced web policy from the integrity verifier is invalid. In addition, Figure 1 illustrates how the proposed framework is separate from the web server. Note that the components of the framework do not need to run on a dedicated machine, they can be run as separate processes on the server.

We classify the server page *SP* into two types: client-scripting file, and server-scripting file. If the file contains server- and client- scripting codes then type of file is server-scripting file. The client-scripting file is possible to analyse through the manager before sending the request to a web server because this type is run on a web browser for rendering it, therefore, the manager does not intercept the response of this type. On the other hand, the manager intercepts the request and response if the type of *SP* is server-scripting file because this file is run on a web server, therefore, it is much easy to analyse response of this file after processing the request on a web server.

Our proposed system offers integrity of data, and a higher level of trustworthiness to an organization and the user. We believe that the proposed framework will be capable of verifying web pages and referenced objects on the designated directories of the web server, and web content on the fly

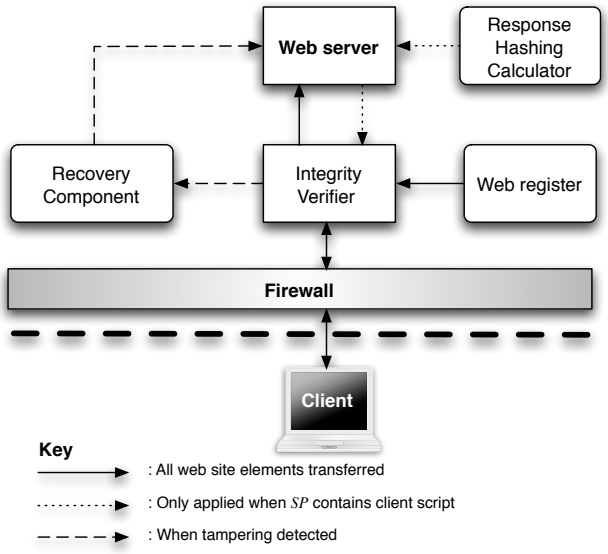

Fig. 1. Schematic view of WCV architecture

against tampering. To ensure the survivability of web services, we enable the WCV system to detect and protect web content against tampering and recover the original copy of the compromised object. We are using risk analyzer techniques to protect user information on the web server if the web server has been tampered with. The WCV system possesses a number of properties: (i) it does not require modifications to existing web application architectures, (ii) it does not require any additional changes on the client and adopts minimal changes on a server, and (iii) it can be run on all major web browsers.

C. Functional Overview

When a client request arrives, the following steps are performed:

1) The integrity verifier (manager) intercepts the HTTP request, checks it, analyses it, extracts the hash value of original copy of server page *SP* code element from the secure repository (all web site elements must be registered before they are published over HTTP Request-Response model), calculates the hash value of *SP* copy which is stored on a the designated directories of a web server, and compares the two hash values to verify the integrity of static *SP* code element; if they match, then the integrity of *SP* is valid; otherwise, the *SP* is illegally tampered with. Similarly, if *SP* is a client-scripting file, the manager extracts the original hash value of referenced objects, calculates the hash value of *SP* referenced objects, and compares the two hash values; if they match, then the integrity of *SP* referenced objects is valid. The manager forwards the request to a web server if the current policy is valid. If not valid, the manager forwards to recovery component to identify the tampering problem.
2) The application server processes the request. The current web policy, which is relied on the type of server page *SP*, determines whether the response hashing calculator component calculates the hash value of response or not. The hash value of response is appended to the response content if the current web policy is valid (i.e. the type of *SP* is server-scripting file).
3) The response and in addition to the calculated hash value are sent to the manager. The manager analyses a response, extracts the original hash value of a response (this value is appended into the response), recalculates the hash value of response, and compare the two hash values to verify the integrity of response; if they match, then the integrity of response is valid; otherwise, the response has been tampered with.
4) If the current policy is valid, the manager analyses the response to verify the integrity for every referenced object element which is linked in the response code element. Otherwise, the manager forwards the response to recovery component.
5) The manager forwards the correct response to the target client if the web policy is valid.
6) The monitoring proxy if presents, monitors the transaction to ensure correct manager behaviour.

D. Challenge Response Protocol

The integrity verifier (manager) component periodically launches a challenge response protocol to check the server and other framework components. This protocol introduces two main purposes: (i) It provides integrity check for the liveness and freshness of the servers. If the manager does not receive a response within some delay[2], it alerts the web-based system elements. (ii) The protocol checks the integrity of web files which are stored on the designated directories of a web server. Further details are described in the next sections.

E. Challenge Request Protocol

The manager component provides integrity check for the liveness and freshness of the servers at the request level. The protocol checks the integrity of static server page *SP* which is extracted from Equation (1) and generates the correct web policy to take the next action. Further details are described in the next sections.

F. Security Framework Components

1) Web Register Component: This component aims to register web site elements before publishing them over HTTP Request-Response model. Another aim is to monitor all web site elements in their designated places for any authorised change that occurs. This requires a new registration for every element that legally is updated. The registration process can take place when one condition of three is satisfied: when a new element is published, when a published element is legally changed, and when certificate of a published element is expired. However, some developers may not take into account the certificate expiration of published elements because they rely

[2]Functionality of challenge response delay will be developed in the next phase of this research.

on a web browser and operating system settings. Therefore, criminal can replace an original element with another expired original element to evade web content of a web server or client machines [14], [15].

1: **algorithm** *registeredCode*($WS_i \in WS$, $SK \in CharacterSet$) **return** $codeRecordset$
2: $C : Codeset/ * C = \{C_i \mid 0 \leq i \leq n\}, C_i = (ID, CV, Cs, Path)$*/
3: $ID : LongIntegerSet$ /* Identification number*/
4: $CV : CharacterSet$ /* Code content value */
5: $CS : CharacterSet$ /* Code signature*/
6: $Path : CharacterSet$ /*Path of Page code on a web server*/
7: $C \leftarrow \emptyset$
8: **foreach** $WP_i \in WS$ **do**
9: $CV \leftarrow$ *extractCode*(WP_i)
10: /* *extractCode()* extracts the code of WP_i*/
11: **if** (not *Register*(CV) or *Changed*(CV)) **then**
12: $CS \leftarrow$ *Sign*(CV,SK)
13: $Path \leftarrow$ *extractPath*(WP_i)
14: $C_i \leftarrow C_i \cup (ID, CV, CS, Path)$
15: **end if**
16: **end for**
17: $DB \in DatabaseSet$
18: /*To store the code details in Database*/
19: $DB \leftarrow$*connectDatabase*()
20: $codeRecordset \leftarrow \emptyset$
21: **foreach** $C_i \in WS$ **do**
22: $codeRecordset \leftarrow codeRecordset \cup$ *Put*(C_i)
23: **end for**
24: Output($codeRecordset$)
25: **end algorithm**

Fig. 2. Registration stage of code element

The proposed registration technique which is based on Equations (4, 5, 6) provides the calculation property of the hash value for every site element and stores their values on a secure repository (such as secure database). Note, this repository might be logically or physically independent of main server. Furthermore, this technique uses the proposed hashing scheme which is based on the Equations (4, 5, 6) in which the shared referenced objects have the same signature even they belong to different web pages [1]. The process of this registration technique passes into two stages: registration process of page code element and registration process of referenced objects elements. First, the registration process of page code: It is important to register the page code at any change that occurs. This change can take place either on the view content, or the structure of a web page, or both. As a result, this can cause to change the presentation of a web page. We develop an algorithm (see Figure 2) that describes the functions and the steps to register a page code. Note, this algorithm can be applied for any page code regardless type of client- or server-scripting language, and size of page code. It consists of four sections: the first section (line 1) presents the inputs of this algorithm: the target web site WS and secret key SK which is used to calculate the hash value of page code and relied on the size of page code and the time factor which specifies the expiry of a secret key to identify this kind of tampering attack. The second section (Lines 2-7) assumes a set of codes C and their elements, where C is derived from Equation (5) that shows a web page is a set of referenced objects and the page code itself. The third section (Lines 8-16) includes two basic functions: *extractCode()* function is to extract the page code, and *Sign()*function is to extract the hash value for every element in codes C set. Finally, the fourth section (lines 17- 24) outputs $codeRecordset$ and stores in secure database DB set.

Similarly, the second stage of registration process calculates the hash value for every referenced object in the target web site WS. We have developed an algorithm for registering the referenced objects of a target web page but it is not shown in this paper because it is similar to registration of page code element algorithm. The hashing functionality of referenced objects will be described in the next phase of this research.

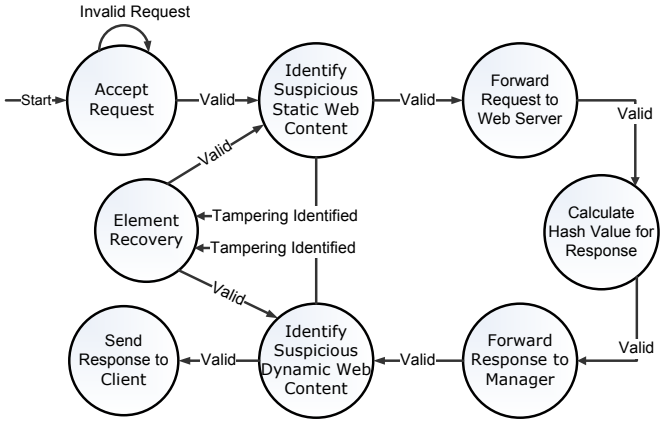

Fig. 3. The finite automata of state protocol

2) Integrity Verifier Component: The integrity verifier component (manager) is positioned between the client machines and target web server. This component manages the HTTP requests and responses via a state protocol that enforces a number of web policies that apply to the elements of a web-based system. The web policy specifies the action to take next whether the request or response is valid or not, and when the policy is enforced. This component generates dynamically a collection of enforced web policies that include registry checking policy, monitoring policy, integrity of request policy, integrity of response policy, and integrity of referenced objects policy, recovery policy, risk policy, and report policy. Figure 3 shows the finite automata of the state protocol that contains a number of steps to generate the enforced web policies. The manager comprises a number of the following stages to enforce the correct web policy.

Stage 1, this stage aims to produce checking policy that implies whether the server page copy SP is found or not in

the designated directories of a web server. Function *extractRequestedPage()* in Figure 4 illustrates how to extract *SP*, which it represents the path of requested page on the web server, through Equation (1) before processing a request on a web server as shown in lines 4-6, and the path of *SP* assigns to $requestedPath$ variable(see lines 6-7).

1: Input($I_j \in I$)/*I is the interaction element*/
2: **function** *extractRequestedPath*($Input$) **return** $requestedPath$
3: $requestedPath \in CharacterSet$
4: $I_j \leftarrow (IT, SP, D, T, U)$
5: $requestedPath \leftarrow I_j(SP)$
6: Output($requtestedPath$)
7: **end function**

Fig. 4. Extraction of *SP* Path

The second task of this stage is to check whether *SP* is found or not on a web server as shown in Figure 5. This function returns the value of current policy (registry checking policy) that should be enforced to take next action.

Second 2, this stage includes three tasks. The first task is to calculate the hash value of *SP* code which placed on designated directories of a web server. The second task of this stage is to extract the original hash value of *SP* from the secure repository (Database Set). The third task is to enforce the integrity of request policy. The following function in Figure 6 illustrates a comparison between two elements, original hash value of a page code element ($originalHV$) which is retrieved from Database set, and the hash value of page code ($requestedHV$) which is placed on web server directories. Lines 4-8 in Figure 6 specify the comparison operation to return the correct current web policy. Therefore if the matching is true, then the static page code on a web server is free from any tampering problems. Otherwise, the static page code includes tampering problem and then a request is sent to recovery component.

Stage 3 is conduced if the current policy is valid. It aims to protect a backend database on a server through building risk analyser algorithm (see Figure 7) through generating risk policy. This policy applies on the server-scripting file because this file contains the commands of database. The following function in Figure 7 shows how to insert a risk through highlighting the interesting data (SQL deleting and updating operations) and backup them in secure repository. This stage requires minimal change on a request details for further security. Line 8 in Figure 7 returns the correct current policy.

3) Response Hashing Calculator: Response hashing calculator is invoked when finishing the process of response on a web server. The purpose is to calculate the hash value of response before sending it back to the manager. The risk policy determines whether the response hashing calculator component calculates the hash value of current response or not. This hash value is added to the current response. Therefore, the manager intercepts the current response, extracts the hash

1: Input($requestedPath \in CharacterSet$)
2: **function** *extractCheckingPolicy*($Input$) **return** $checkingPolicy$
3: $checkingPolicy \in CharacterSet$
4: **foreach** $WP_i \in WS$ **do**
5: **if** ($requestedPath \in WP_i$) **then**
6: $checkingPolicy \leftarrow$ "$Valid$"
7: Exit Loop
8: **else**
9: $checkingPolicy \leftarrow$ "$Invalid$"
10: **end if**
11: **end for**
12: Output($checkingPolicy$)
13: **end function**

Fig. 5. Extraction of checking policy

1: Input($requestedHV \in CharacterSet$,$originalHV \in CharacterSet$)
2: **function** *matchHV*($Input$) **return** $integrityRequestPolicy$
3: $integrityRequestPolicy \in CharacterSet$
4: **if** ($requestedHV \doteq originalHV$) **then**
5: $integrityRequestPolicy \leftarrow$ "$Valid$"
6: **else**
7: $integrityRequestPolicy \leftarrow$ "$Invalid$"
8: **end if**
9: Output($integrityRequestPolicy$)
10: **end function**

Fig. 6. Generating request policy

value of response ($originalHV$), recalculates the hash value of this response ($responseHV$), and compares between two the two of hash values to verify the integrity of response code element (see Figure 8). Lines 4-8 specify the comparison statements to return the correct web policy. Therefore if the matching is true, then the dynamic web content of response code on a web server is free from any tampering problems. Otherwise, the dynamic web content includes tampering problem and then a request of user sends to recovery component for tampering problem. Finally, we develop another algorithm (it is similar to algorithm in Figure 8) for a comparison between two elements, the hash value of referenced objects and the original hash value of referenced objects. Therefore, if the generated policy is invalid, the server page *SP* sends to recovery component. Otherwise, the response sends to the correct client.

IV. Discussions

There is always a tradeoff between the security and performance. In this paper, we are concerned more in the security design of our proposed system to ensure the survivability of web services than performance issues. We focus on the integrity check on the original element to detect malicious codes added in an unauthorised manner. The WCV system can

```
1: Input(requestedPage ∈ CharacterSet)
2: function insertBackup(Input) return riskPolicy
3:    riskPolicy ∈ CharacterSet
4:    foreach SQLCommand ∈ requestedPage do
5:       requestedPage ← insertSQLCommand()
6:    end for
7:    riskPolicy ← "Valid"
8:    Output(riskPolicy)
9: end function
```

Fig. 7. Building risk analysis model

```
1: Input(responseHV ∈ CharacterSet,originalHV ∈ CharacterSet)
2: function matchResponseHV(Input) return integrityResponsePolicy
3:    integrityResponsePolicy ∈ CharacterSet
4:    if (responseHV ≐ originalHV) then
5:       integrityResponsePolicy ← "Valid"
6:    else
7:       integrityResponsePolicy ← "Invalid"
8:    end if
9:    Output(integrityResponsePolicy)
10: end function
```

Fig. 8. Generating response policy

detect and protect the page code and every referenced object on the designated directories of web server and on the fly against tampering problems. This causes problems for attackers who target the referenced objects and page code - for example, it is possible to replace any original image by another image that contains a malicious code where, a victim requests the altered image and then it can destroy a web server or client machine. Another example, the Cascade Style sheet (CSS) object might be threatened through visualization spoofing attack. The strategy of this attack is to change illegally any important information that is shaded by a particular colour to another colour . This can trick the users of a web page to take a decision that can be different if the shaded text was the correct colour. Therefore the visualization spoofing attack can be a risk to the appearance of a web page - the WCV system is capable of protecting web server content and preventing various tampering attacks.

Unlike Merkle's scheme [9], we propose a new hashing scheme in which the shared referenced objects have the same signature even though they belong to different web pages. This achieves balance between performance (minimising number of calculated signatures of each referenced object in a web site) and security. Further details are described in [15]. The worst case latency of the WCV system occurs when the manager intercepts the request and response and *SP* contains SQL commands. This performance issue increases if the *SP* is client-scripting file. However, the performance can also decrease because we adopt static analysis concept in a number of developed algorithms to produce the correct web policy. In future work, we intend to emphasize performance of our system.

V. Conclusion

Because of the statelessness of HTTP and the limitations of SSL, data integrity can be violated on the server even though the communication channel between the server and client is secure. We present the proposed WCV system which includes a web security real-time framework, work assumptions, conceptual models, and a state protocol of a set of enforced web polices. This framework consists of a number of web-based components: web register, response hashing calculator, integrity verifier, and recovery. Furthermore, we propose a registration technique of codes and referenced objects elements, and new hashing scheme. In the next stage of this research, we will develop the recovery component of the proposed framework and will discuss how to calculate the hash values of various referenced objects (e.g. audio, video, and other objects)- it is suggested that this could be able verify the static and dynamic web content of requested web pages.

Acknowledgment

The authors would like to thank Prof. Maia Angelova for her suggestions in testing the proposed models.

References

[1] S. Aljawarneh, C. Laing, P. Vickers, and M. Angelova. Formulating models to protect web server content against tampering. In *Submitted to IAS '07*, Manchester, UK, 2007.

[2] T. Bass. CEP and SOA: An open event-driven architecture for risk management. IT Financial Services '07, Portugal, 2007. www.idc.pt/resources/PPTs/2007/Financial_Services/7_TI BCO.pdf.

[3] CERT. CERT statistics 1988–2006. http://www.cert.org/stats, 2006.

[4] C. Coronado. On the security and the efficiency of the Merkle signature scheme. Cryptology ePrint Archive, Report 2005/192, 2005. http://eprint.iacr.org/.

[5] B. Gehling and D Stankard. eCommerce security. In *Proceedings of Information Security Curriculum Development (InfoSecCD) Conference 05*, pages 32–37, Kennesaw, GA, USA, Sep 23–24 2005.

[6] M. Hassinen and P. Mussalo. Client controlled security for web applications. In B. Wener, editor, *The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 810–816, Australia, 2005. IEEE Computer Society Press.

[7] D.P. Jacobs and B.A Malloy. An application-centred course on data-driven web sites. In *Proceedings of Frontiers in Education 2001*, pages F2D–10–F2D–14, Reno, NV, Oct 10–13 2001.

[8] T.R. Kuphaldt. *Lessons In Electric Circuits: Volume IV–Digital*. Open Book Project, http://www.ibiblio.org/obp/, 2006.

[9] R.C. Merkle. A certified digital signature. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[10] J. Offutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In *ISSRE 2004 15th International Symposium on Software Reliability Engineering*, pages 187–197. IEEE Computer Society, Los Alamitos, CA, 2004.

[11] C. Reis, J. Dunagan, H.J. Wang, O. Dubrovsky, and S. Esmair. Browsershield: Vulnerability-driven filtering of dynamic HTML. In *Proceedings OSDI 06 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 61–74. USENIX Association, Nov 6–8 2006.

[12] A.D. Rubin and D.E. Geer. A survey of web security. Computer, 31(9):3441, 1998.

[13] D. Scott and R. Sharp. Specifying and enforcing application-level web security policies. *IEEE. Knowl. Data Eng*, 15(4):771–783, 2003.

[14] S. Sedaghat. Web authenticity. Master's thesis, University of Western Sydney, Australia, 2002.

[15] S. Sedaghat, J. Pieprzyk, and E. Vossough. On-the-fly web content integrity check boosts users' confidence. *Commun. ACM*, 45(11):33–37, 2002.

[16] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stavridou, and E. Uribe. An architecture for an adaptive intrusion-tolerant server. In Bruce Christianson, James A. Malcolm, and Michael Roe, editors, *Security Protocols Workshop*, volume 2845 of *LNCS*, pages 158–178. Springer Verlag, 2002.