Design and experimental evaluation of Web Content Verification and Recovery (WCVR) system: A survivable security system

Shadi Aljawarneh, Christopher Laing, and Paul Vickers
School of Computing, Engineering and Information Sciences
Northumbria University, Newcastle upon Tyne, NE2 1XE, UK
{shadi.aljawarneh, christopher.laing, paul.vickers}@northumbria.ac.uk

Abstract

We have designed a novel security system, called Web Content Verification and Recovery (WCVR) system to solve unauthorised tampering on server-side web content. Our solution is implemented as a prototype. This prototype consists of three mechanisms: web register, HTTP interface, and response hashing. In this paper, we have conducted a set of experiential studies to meet the security and performance objectives. The results of an experimental study have shown that the proposed system (WCVR) provides a high coverage of detection and protection, and a low level of overhead times.

Keywords

Tampering, integrity of data, confidentiality of data, availability of data

1. Introduction

Although current security mechanisms could provide a protection against unauthorised access to system resources, several security incident reports from emergency response teams such as The Computer Emergency Response Team (CERT) clearly demonstrate that the available security mechanisms have not made system break-ins impossible [3]. Note that, the three basic objectives of web security include confidentially, integrity, and availability of data.

Data integrity has received little attention in information security research and technical security groups and communities. Furthermore, there is little published research in methods for testing web content integrity [7]. The published research and technical communities in web security area are generally more concerned with cryptographic rules, and algorithms. In an attempt to address this, our paper focuses on the integrity of data and does not delve into other issues of data. If the integrity of data is violated, its confidentiality and availability can be compromised. It should be noted that data integrity refers to the trustworthiness of information resources, thereby ensuring that only an authorised client can alter the data – unauthorised tampering may result in incorrect or malicious web application behaviour behind installed firewalls [4, 5, 11].

Server-side static and dynamic web content can be tampered with by modifying the style classes, referenced objects (images, audio, video, and other objects), the source code of the web page itself, and also by running malicious code on the server to intercept a requested page before the client receives it [4, 5, 6, 8, 11]. The attackers are interested on targeting the referenced objects and page code on the fly. For example, it is possible to replace an original image by another image containing malicious code. A victim requests the altered image and then it can disrupt the contents of a web server or client machine. In addition, the Cascading Style Sheet (CSS) object is threatened through a visualization spoofing attack. The strategy of this attack is to change any important information that is identified by a particular colour to another colour the objective is to manipulate the user into making a decision that is based on incorrect information.

The Secure Sockets Layer (SSL) protocol was developed to support the integrity of data transit [4, 5, 11], and as such, it does not provide an absolute solution. SSL is not capable of verifying the integrity of web content before a request or response enters the secure communication channel [11]. 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 [6, 10].

Dynamic data is also a critical issue [6, 9]. The generation of dynamic web content depends on user interaction. Different user information leads to different generated web content. Therefore, it is very difficult (even impossible) to

analyse the requested page of dynamic code before processing on a web server. The dynamic code of server programming languages needs to be processed on a web server before returning the response to a web browser. As a result, we cannot guarantee that dynamic code is not tampered with even if the static code is verified; therefore the generated web content should also be verified.

Indeed, the verification of web server content integrity is becoming more important, because many web applications generate units of web content on the fly. It is therefore important to develop systems for integrity verification that are able to provide web content protection, detection of malicious manipulation of web content, protection against tampering, and authentication of content [1, 2].

This paper is organized as follows: Section 2 gives an overview of the existing integrity verification approaches and systems. Section 3 outlines a proposed integrity verification system. Section 4 describes the prototype of WCVR implementation and its mechanisms. In Section 5, we present an experimental study to measure the performance through the proposed system on IIS and Apache Tomcat web servers. Finally, Section 6 draws conclusions.

2. Existing Integrity verification approaches

We now discuss the three recent approaches that attempt to address the tampering problems on the server-side web content. First, Hassinen and Mussalo [5] 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.

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. 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 [11] 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 [12] 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.

3. Web Content Verification and Recovery (WCVR) System

As discussed above, the problem to be addressed is that the integrity of the dynamic and static web content on a server can be compromised even though the communication channel between the client and server-sides is secured. We have proposed a survivable system to identify tampering attacks, and therefore, we have developed a Web Content Verification and Recovery (WCVR) system to investigate into a server-side static and dynamic web content survivability before the client receives the requested page.

In this paper, the survivability is the capability of a web content to continue its mission over the HTTP request-response model even in the presence of illegitimate modifications (modifications caused by tampering attacks) to a web content. The question then arises, what happens when a altered web content has been detected? Our survivability strategy in the proposed WCVR system can be set up in two steps:

1. Detection and response by integrity verification process.

2. Recovering from tampering attacks by recovery process.

3.1. Architecture of web security framework

The proposed framework consists of a number of components as shown in Figure 1 [1]:

- DBMS tables: we create two DBMS tables: offline-transaction table for mapping the hash values of static web contents to their specific repositories of web servers, and online-transaction table for mapping the hash value of current dynamic web content to its server scripting web page.

- Web register component: calculates the hash values of static web contents that have developed for use in a secure web environment.

- Integrity verifier (manager): is positioned between the client machines and the target web server. This component manages the HTTP requests and responses via a state protocol that enforces a number of web policies (such as request availability policy, integrity failure policy, integrity passing policy, and recovery policy) that apply to the elements of the web system.
- Response hashing calculator: aims to do hashing calculations, and backup for the generated dynamic web content before response sends it back to the manager.
- Recovery component: recovers the tampered web content.

It should be noted that the proposed framework is separate from the web server. In addition, the components of the framework do not need to run on a dedicated machine, they can be run as separate processes on the server. The WCVR system has a number of advantages over other approaches:

1. It does not require modifications to existing web application architectures,
2. It does not require any additional changes on the client and server, and
3. It is compatible with all major web browsers.

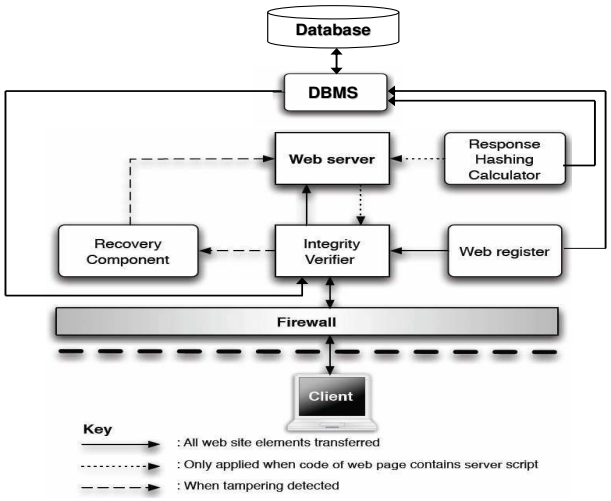

Figure 1. Schematic view of WCVR architecture

3.2. Functional Overview

We propose the functionality steps of the suggested solution. When a client request arrives, the following steps are performed [1]:

1. The integrity verifier (manager) component intercepts the HTTP request (such as web page, audio, video, images, and others), checks it, analyses it, extracts the hash value of the original copy of the static web content from DBMS offline-transaction table, recalculates the hash value of the web content, and compares the two hash values for integrity verification process. If they match, then the integrity of the requested web content is valid; otherwise, the the requested web content has been tampered with. The integrity verifier (manager) forwards the request to a web server if the enforced web policy is satisfied. If it is not satisfied, the integrity verifier sends the request to the recovery component to identify the tampering problem and reports this attack to web administrator.
2. Once a web server application has processed the request, the response hashing calculator component calculates the hash value of output response and makes a backup for the output response. The hash value of the response (dynamic web content) is appended to DBMS online-transaction table.
3. The response is intercepted by the integrity verifier component. The manager analyses the response, extracts its original hash value (this value is appended to the secured online-transaction table), re-calculates it, and compares the two hash values for integrity verification process. If they match, then the integrity of the response is valid; otherwise, the response has been tampered with. Therefore, if it is not valid, the manager sends the response to the recovery component to identify the tampering problem and reports this to the web administrator.
4. The integrity verifier component forwards the correct response to the target client.

4. Implementation of WCVR system

The WCVR system is implemented in Java, Servlets, and Filters. The DBMS Microsoft Access 2007 Database is selected as the repository for storing and retrieving details about static and dynamic web contents. In order to demonstrate that our WCVR system is able to ensure the survivability of server-side web content against tampering, we have undertaken some experimental testing (see Section 5).

4.1. Architecture Design of the Prototype

The WCVR prototype consists of three mechanisms: Web register, HTTP interface, and response hashing mechanism.

4.1.1. Web Register Mechanism

The functionalities of the web register mechanism are summarized into the following:

- Reading in binary format for every static web content in the secure repositories.
- Calculating the hash value for every static web content using SHA1 function.
- Requesting Microsoft Access DBMS to store details for every static web content.
- Checking the modification status for every static web content, if modified, recalculate the hash value with taking into account the new assembly of a private key which is used in SHA1 hashing function.

4.1.2. Response Hashing Mechanism

This mechanism aims to calculate the hash value of the output response (dynamic web content) which is generated by a server scripting language such as JSP, ASP, PERL, and others, and to make online backup for the output response (the produced dynamic web content) in a secure server repositories. The hash value of dynamic content is stored in the DBMS online-transaction table for integrity check before the client receives the requested page.

4.1.3. HTTP Interface Mechanism

The HTTP interface mechanism is the manager of the WCVR system, and is based on the integrity verifier component. This component launches a state protocol to enforce the target web policy. The value of the web policy determines the action(s) should be taken by the HTTP interface mechanism. All the web policies have been implemented in this mechanism. The goals of HTTP interface mechanisms are:

- Online verification of integrity of server-side static web content.
- Online verification of integrity of server-side dynamic web content.
- Online recovery of server-side static web content if the static web content has been tampered with.
- Online recovery of server-side dynamic web content if the dynamic web content has been tampered with.

In this paper, we have developed multi-threaded java application for handling concurrent connections (requests in parallel) using multiple threads that increase the power and flexibility of a web server and client programs significantly.

5. Evaluation

We tested our system in environment which is composed of two web servers: Apache 1.3.29 with Tomcat container 5.01 on MS Windows Server 2003, and IIS 6.0 on MS Windows Server 2003. The two web servers contain a copy of target web site and shopping cart application. The choice of two web servers are dictated by the fact that they contain many tampering vulnerabilities such as visualisation spoofing attacks that can be easily exploited. Over 45 attacks have been performed against the server-side generated static and dynamic web content security properties. We have exploited different type of vulnerabilities that allows for the modification of files in the designated directories of a web server (attack against integrity). During the testing, all the attacks launched against the web servers were detected and recovered by the WCVR system.

5.1. Case Study for Micro-benchmarking Performance

We measured the runtime performance of the web register mechanism with a set of micro-benchmarks. We measured the latencies of web register mechanism in two different cases, namely, SHA1 (10 digits), and SHA1-extended (16 digits). In the SHA1 case, we calculated the hash value of a web content by SHA1 function where number of digits was 10. SHA1-extended represented the case when we calculated the hash value of a web content by SHA1 function where number of digits was 16. Since the goal is to measure the latency, we ran the web register mechanism 15 times on over 200 entries of different sizes for every case (SHA1 and SHA1-extended) under MS Windows XP Professional.

An illustration of results is presented in Table 1. It is clear from the table that the overhead of web register in the case of SHA1 (10 digits) is low – the average running time was 1.4274 seconds to run (the average of time taken to run 15 trails), less time than the case of SHA1-extended which it was 2.2176 seconds to run. Note that these cases did not only measure the overhead of the hash value itself, it also measured all functions in a web register mechanism for both cases, so that this difference of overhead results from using two different cases of SHA1.

We have concluded that the SHA1-extended case is clearly the most costly in performance terms to execute. This is reasonable, because the SHA1-extended contains 16 digits instead of using 10 digits.

We have also presented the registry performance of a web content as a function of file sizes. We measured the web register mechanism running time for the both cases: SHA1 and SHA1-extended, varying the input file sizes. The results are shown in Table 2. When the file size is large, the hashing overhead can be significant for both cases. For

example, measuring a 64 Kilobytes file had taken about 12.47 milliseconds in case of SHA1-extended, where it had taken about 3.2 milliseconds in case of SHA1. Another interesting result, measuring a 13 Megabytes file recorded about 1531 milliseconds in case of SHA1-extended, where it recorded about 620.067 milliseconds in case of SHA1. The running time increased close to a linear state as the size of file increased. Moreover, the hashing overhead of case SHA1-extended takes more running time than case SHA1 and this increases as the size of file increases.

Table 1. Overhead of a web register mechanism

Web Register Call	*Overhead (ms)*
SHA1-extended (16 digits)	2217.6 (2.2176s)
SHA1 (10 digits)	1427.4 (1.4274s)

Table 2. Registry Performance for both SHA1-extended and SHA1 as compared with File Sizes.

File Size (Byte)	*Overhead (ms) with SHA1-extended*	*Overhead (ms) with SHA1*
1KB	0.64	0.627
16KB	5.13	2.13
64KB	12.47	3.2
2MB	163.73	118.73
5MB	348	251.97
13MB	1531	620.067

The impact of hashing and encryption are issues that increase the overhead and they are rarely considered in the area of web engineering and design. When using the encryption and hashing, some users have observed that the CPU overhead of sending and receiving encrypted requests, and the hashing verification to be as high as 100 to 200 milliseconds per request, easily overwhelming any other processing [13]. This overhead varies widely by implementation, key length and other factors, but it is always costly in performance terms. Therefore, we have benched the data for two cases: SHA1 (10 digits), ad SHA1-extended (16 digits).

We use the second case of hashing function (case of SHA1-extended) in the WCVR prototype. Although this case is more costly in performance terms (see Table 1 and Table 2) because by design, longer keys take much more computing resource to decrypt, making them less vulnerable to attack by repetitive means. Unfortunately, this also means that legitimate users pay a substantial cost for security.

5.2. End-to-End Performance Evaluation

A load test can be used to test an application's robustness and performance, as well as its hardware and bandwidth capacities. Therefore, we used the Neoload[1] application which is a stress and load testing tool to (i) test a web site's vulnerability to crashing under load and (ii) check response times under the predicted load.

As the verification and recovery processes are performed online in real-time, it should induce a time overhead in the service. The results presented in this section have been obtained on the same set of requests, using the same architecture of web servers.

Note that, the duration of the test was almost exactly 30 minutes where the run-time policy was ramping up (i.e. Generating a number of virtual users that increases throughout the test) from 2 users adding 2 users every 2 minutes. The virtual users were connecting at 100Mbps through a local network.

All these measurements were performed from the client point of view. Each row in the table displays the average response time (request), maximum response time (request), and minimum response time (request) in seconds, of all requests during the test and average page response time for all pages where each page may contain a number of requests. Note that the average response time is the mean-time necessary to process a request by each web server when the proxy, browser, and the WCVR system are active. Activating the WCVR implies the creation of lots of communications, the activation of the verification process, and activation of recovery component if the server-side static and/or dynamic web content has been compromised and tampered with. The communications (network response time) are parts of the measured durations. The WCVR tested is a prototype, and thus is not really optimised.

5.2.1. Experimental Case Study

This experimental study has been conducted by undergraduate computing student at Northumbria University and has consisted of two parts: case study for static web content, and case study for dynamic web content. Case study for static web content is represented by experiments one and two, whereas case study for dynamic web content is represented by experiment three. We have summarised the results in a specific table for each part.

Case Study - Static web content

Experiment one and two are individually carried out on MS Windows environment, IIS and Tomcat web servers. Experiment one is designed to show end-to-end performance measurement through the proposed WCVR system

[1] http://www.neotys.com/

and the DSSA existing system on IIS web server over local network for verification of server-side static web content integrity, whereas experiment two is designed to show end-to-end performance measurement through the proposed WCVR system and the DSSA existing system on Tomcat web server.

We obtain a statistics summary after running each system individually, as shown in Table 3. In this study - experiment one, 9130 hits were created, 6096 web resources were requested out, 57.22MB (total throughput) were received, and number of virtual users were launched in this test was between 114 and 207. In experiment two, 11041 hits were created, 6116 web resources were requested out, 89.84MB (total throughput) were received, and number of virtual users was between 88 and 89.

Table 3. Static web content: Comparison between the response times through DSSA or WCVR systems, in seconds, of all requests during the test on IIS and Tomcat web servers.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average 90% (request)	Average Page Response Time
DSSA	IIS	0.165	4.03	4.48	4.06	3.84
WCVR	IIS	0.073	1.06	4.12	0.951	1.63
DSSA	Tomcat	0.117	3.86	4.71	4.04	3.96
WCVR	Tomcat	0.07	1.93	90.3	0.834	3.46

As a result, the end-to-end performance of WCVR system is clearly very effective. This is because the overhead of WCVR system on the both web servers (IIS, and Tomcat) are minimal and very low in comparison with the DSSA mechanism (see Table 3). The average response time (request) of WCVR on IIS was 1.06 seconds and on Tomcat was 1.93 seconds where the average response time (request) of DSSA on IIS was 4.03 seconds and on Tomcat was 3.86 seconds. The results indicate that the WCVR is less costly in performance terms to verify the server-side static web content against tampering attacks on IIS and Tomcat web servers.

Case Study - Dynamic Web Content

Experiment three is designed to show end-to-end performance measurement through the proposed WCVR system over wired network on Tomcat web server for verification of server-side dynamic web content integrity. In this experiment three, the response times for Scenario A (No mechanisms or systems for tamper protection, and recovery) and Scenario B (With WCVR system) were collected.

In this part of study, 5421 hits were created, 5397 resources were requested out, and 1.55MB (total throughput) were received. Number of virtual users were launched in this test was between 198 and 204.

As is shown in Table 4, the end-to-end performance of

Table 4. Dynamic Web Content: Comparison between the response times through without verification system or WCVR systems, in seconds, of all requests during the test on Tomcat web server.

System	Web Server	Graph Min (request)	Average Response Time (request)	Graph Max (request)	Average Page Response Time
Without verification system	Tomcat	0.029	3.14	4.86	3.24
WCVR	Tomcat	0.219	3.95	6.01	3.97

WCVR system is nearly effective and acceptable because the overhead of WCVR system on Tomcat web server is minimal in comparison with response times for Scenario A. The average response time (request) of WCVR on Tomcat was 3.95 seconds, whereas the average response time (request) through without verification system on Tomcat was 3.14 seconds. It is suggested that the results indicate the WCVR is nearly similar response time to execute and verify the server-side static web content against tampering attacks on Tomcat web server compared without verification system. Therefore, the WCVR satisfies the performance objective for integrity verification of server-side web content.

6. Conclusions

Data integrity can be violated on the server even though the communication channel between the server and client is secure. We have presented a novel WCVR system. The framework architecture of WCVR system consists of a number of components: web register, response hashing calculator, integrity verifier, and recovery.

To conclude, the performance of WCVR system is nearly effective on Tomcat and IIS web servers. Therefore, the WCVR has satisfied the performance objective and acceptable.

References

[1] S. Aljawarneh, C. Laing, and P. Vickers. Security policy framework and algorithms for web server content protection. In *ACSF '07*, Liverpool, UK, 12–13 July 2007. Liverpool John Moores University.

[2] S. Aljawarneh, C. Laing, and P. Vickers. Verification of web content integrity: A new approach to protecting servers against tampering. In M. Merabti, editor, *PGNET 2007 The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, Liverpool, UK, 28–29 June 2007. Liverpool John Moores University.

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

[4] 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.

[5] 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.

[6] 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.

[7] R. Probert, B. Stepien, and P. Xiong. Formal testing of web content using TTCN-3. In *TTCN-3 User Conference 2005*, 2005.

[8] C. Reis, J. Dunagan, H. 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.

[9] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, pages 223–238, 2004.

[10] D. Scott and R. Sharp. Specifying and Enforcing Application-Level Web Security Policies. *IEEE. Knowl. Data Eng*, 15(4):771–783, 2003.

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

[12] 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 B. Christianson, J. A. Malcolm, and M. Roe, editors, *Security Protocols Workshop*, volume 2845 of *LNCS*, pages 158–178. Springer Verlag, 2002.

[13] B. Wong. Sizing up Your Web Server. Sun World Online, Oct 1997. http://sunsite.uakom.sk/sunworldonline/swol-10-1997/swol-10-sizeserver.html.