(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 August 2006 (03.08.2006)

PCT

(10) International Publication Number
WO 2006/081459 A2

(51) International Patent Classification:
G06F 15/16 (2006.01)

(21) International Application Number:
PCT/US2006/003003

(22) International Filing Date: 25 January 2006 (25.01.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/647,181	25 January 2005 (25.01.2005)	US
11/339,373	24 January 2006 (24.01.2006)	US

(71) Applicant *(for all designated States except US)*: **WHITEHAT SECURITY, INC.** [US/US]; 3003 Bunker Hill Lane, Suite 106, Santa Clara, CA 95054-1144 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **PENNINGTON, Bill** [US/US]; 48 Norton Avenue, San Jose, CA 95126 (US). **GROSSMAN, Jeremiah** [US/US]; 85 Rio Robles E. # 3318, San Jose, CA 95134 (US). **STONE, Robert** [US/US]; 97 Sherland Avenue # D, Mountain View, CA 94043 (US). **PAZIRANDEH, Siamak** [US/US]; 100 Buckingham Drive # 258, Santa Clara, CA 95053 (US).

(74) Agents: **SULLIVAN, C., Bart** et al.; Townsend and Townsend and Crew LLP, 2 Embarcadero Center, 8th Floor, San Francisco, CA 94111 (US).

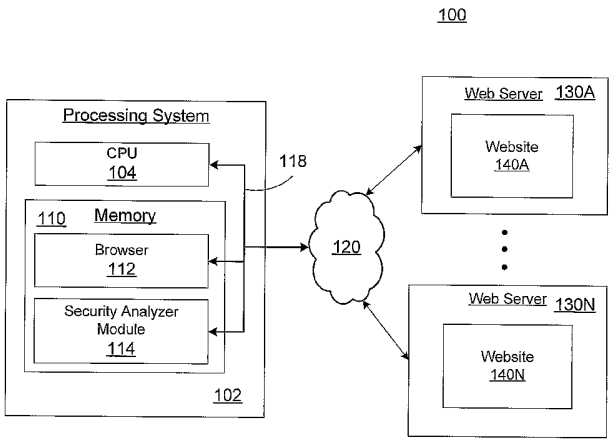
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declaration under Rule 4.17:
— *of inventorship (Rule 4.17(iv))*

[Continued on next page]

(54) Title: SYSTEM FOR DETECTING VULNERABILITIES IN WEB APPLICATIONS USING CLIENT-SIDE APPLICATION INTERFACES



(57) Abstract: An improved method and apparatus for client-side web application analysis is provided. Client-side web application analysis involves determining and testing, using client-side application interfaces and the like, data input points and analyzing client requests and server responses. In one embodiment, a security vulnerability analyzer is employed to analyze web page content for client-side application files, such as Flash files and Java applets, extract web addresses and data parameters embedded in the client-side application file, and modify the data parameters according to user-defined test criteria. The modified data parameters are transmitted as part of a request to a respective web server used to service the client-side application files. The security vulnerability analyzer analyzes the response from the server to ascertain if there are any security vulnerabilities associated with the interface between the client-side application file and the web server.

WO 2006/081459 A2

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM FOR DETECTING VULNERABILITIES IN WEB APPLICATIONS USING CLIENT-SIDE APPLICATION INTERFACES

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Patent Application No. 60/647,181, entitled "System For Detecting Vulnerabilities In Web Applications Using Client-Side Application Interfaces" filed January 25, 2005, which is hereby incorporated in its entirety.

BACKGROUND

[0002] Millions of people worldwide bank, shop, purchase goods and services, perform research, etc. using the World Wide Web. With each transaction, private information, including names, addresses, phone numbers, credit card and debit card numbers, passwords, are routinely transferred and stored in a variety of locations. Billions of dollars and millions of personal identities and private information are vulnerable to hackers who find there way through security vulnerabilities in web-applications.

[0003] One approach to identifying security vulnerabilities in a web application is to examine the code for the web application to identify risk-prone operations within the code. However, this might not provide a complete picture of vulnerabilities, as the execution structure of the code might not be apparent from reviewing the code and the interplay of the examined code and other parts of a web application might introduce other vulnerabilities. Additionally, while examined code might be secure, it is possible for an unauthorized client device to modify client-side code to do something other than what was intended by the web application. Therefore, web application testing should also involve testing the limits of what unauthorized client-side code could perform.

[0004] Difficulties with properly executing security scans of web applications from the client-side include being able to correctly determine the possible paths of interaction between a user and a client and the web application. Often, this is made difficult because the client is provided with code to execute that has varied points of entry to the web application that is not in a readily understood format. For example, if the client-side code comprises a webpage formatted with simple HyperText Markup Language (HTML), a reader or a computer

program can easily determine all of the entry points of that code, such as the form fields and embedded URLs.

[0005] Therefore, what is needed is a system and method to detect security vulnerabilities in server based web applications using client-side applications that are simple to implement and cost effective.

BRIEF SUMMARY

[0006] Embodiments of the invention provide an improved method and apparatus for analyzing client-side web applications to reveal security vulnerabilities in server-side interfaces associated with client-side web applications. In one embodiment, a security vulnerability analyzer is employed to retrieve webpage content from a server in communication with the security vulnerability analyzer. The security vulnerability analyzer retrieves an application file, such as a Flash file (file extension .fla, .swf, and the like), referred to in the webpage content. The security vulnerability analyzer extracts web addresses and associated input data parameters embedded in the application files. The security vulnerability analyzer replaces the input data parameters with user-defined input criteria. A request along with the user-defined input criteria is transmitted to the servers referred to in the web addresses to invoke a response from the servers. The security vulnerability analyzer analyzes the server responses by matching the received responses against a plurality of pre-determined server responses that alone or in combination are indicative of server-side security vulnerabilities.

[0007] In one embodiment the invention provides a method for analyzing security vulnerabilities in web-based applications. The method includes extracting path parameters and data parameters from the client-side applications referring to web servers servicing the client-side applications, modifying the path parameters or data parameters with user-defined data to generate test data, transmitting the test data to the web servers, and determining if any of responses received in response to the test data are indicative of security vulnerabilities.

[0008] In another embodiment the invention provides a method of detecting security vulnerabilities in web-based applications. The method includes extracting data from client-side applications to locate data entry points to web servers used to service the client-side applications, and applying user-defined test criteria to the data entry points. The user-defined testing data is configured to cause a web server to respond with one or more predetermined

responses configured to expose security vulnerabilities in client-server interfaces associated with the data input points.

[0009] In another embodiment the invention provides a security analyzer for detecting security vulnerabilities in client-server interfaces. The security analyzer includes a security analyzer engine configured to extract client-server interface data embedded in client-side application files, and generate user-defined testing data from the client-server interface data. The user-defined testing data is configured to cause a web server to respond with one or more predetermined responses that are used to expose security vulnerabilities in client-server interfaces associated with the client-side application files.

[0010] Other embodiments of the invention are described in detail below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] FIG. 1 is a high-level block diagram of system for scanning webpage content and client-side applications to analyze security vulnerabilities in accordance with embodiments of the invention;

[0012] FIG. 2 is a high-level block diagram of a security analyzer software module in accordance with embodiments of the invention;

[0013] FIG. 3 is flow diagram illustrating a method of assessing web application security vulnerabilities in accordance with embodiments of the invention;

[0014] FIGs. 4A-C illustrate an example of a webpage content which includes URL and data parameters extracted from a client-side application file referred to in the webpage content in accordance with embodiments of the invention; and

[0015] FIGs. 5A-C illustrate an example of a method of altering client requests using a pre-defined set of transformation criteria to generate web-application security tests and security vulnerability ranking in accordance with embodiments of the invention.

DETAILED DESCRIPTION

[0016] Embodiments of the present invention provide a system and method to examine webpage content for locations to client-side application files. The paths and associated data parameters embedded in the client side application files are extracted and modified with user-defined test criteria. The modified paths and associated data parameters are transmitted to respective servers referenced in the extracted paths. Responses received from the servers are evaluated to determine if the paths and associated data parameters provide unauthorized network access through the client-side interface and therefore represent a security vulnerability risk.

[0017] FIG. 1 is a high-level block diagram of a system **100** for analyzing webpage content and client-side application files to determine server-side security vulnerabilities. As shown, system **100** includes a processing system **102**, a communication network **120**, and one or more web servers **130A** through **130N** serving respective websites **140A** though **140N**, where "N" is a plurality. For clarity, server **130A** and website **140A** will be used in the following discussion.

[0018] Processing system **102** includes a central processing unit (CPU) **104** and memory **110**. Memory **110** may include a browser **112**, and security analyzer module **114**. Memory **110** is preferably random access memory sufficiently large to hold the necessary programming and data structures required for the operation of processing system **102**. While memory **110** is shown as a single entity, it should be understood that memory **110** may in fact comprise a plurality of modules, and that memory **110** may exist at multiple levels, from high speed registers and caches to lower speed but larger DRAM chips. In one embodiment, processing system **102** may interact with website **140A** through communication network **120**. Communication network **120** may be any network, such as the Internet, a local area network (LAN), a wide area network (WAN), a wireless network, a wire-line network, etc.

[0019] Website **140A** may be stored on a storage device, such as web server **130A**. In one embodiment, web server **130A** is a computer that holds the files for one or more websites, such as website **140A**. In some cases, website **140A** may be split over a number of servers in different geographical locations. Website **140A** may be any software application. Although website **140A** is described, it will be understood that any software application may be analyzed. In one embodiment, website **140A** is a collection of files. For example, website **140A** may include a beginning file called a home page. This home page may be associated

with an identifier, such as a universal resource locator (URL), such as HTTP://www.coopertest.com. The URL corresponds to a file that is being stored. From a home page, other pages may be accessed using links on the home page. The other pages may be associated with other URLs, such as www.coopertest.com/login.html. A person of skill in the art will appreciate additional details regarding web sites that are not described. Although the terms webpages and websites are used herein, it will be understood that these terms may include applications that are not web based.

[0020] When a client, such as processing system **102**, requests a webpage, web server **130A** may use a file stored on the web server **130A** in order to serve the webpage **140A** to the client. The webpage **140A** may then be displayed on an interface, such as web browser **112**. Actions may then be performed with the webpage **140A**. For example, items may be selected ("clicked") to request other webpages, text may be entered, forms may be filled, documents transferred, Flash files (file extension .fla or .swf) or java applet applications may be invoked, etc. Requests indicating these actions may be sent to one or more web servers **130A** for further processing. For example, login information, such as a username and password, may be entered on a webpage in order to login to web site **140A**. In this case, a user may access a restricted webpage that is only accessible if the login information is entered. For example, a restricted webpage may show a user's personal email account information.

[0021] A protocol may be used in communications between browser **112** and web server **130A**. In one embodiment, the HyperText Transfer Protocol (HTTP) is used. Using the example above, login information, such as a username and password, account number or related information, is entered in a webpage sent in a request. The login information may be any information that allows access to restricted parts of website **140A**. The login information sent in the request may allow a user to login into website **130A**. In one embodiment, in response to receiving the login information, credential information may be sent in a response from web server **130A**. Credential information may be any information that may be needed to access the restricted parts of website **140A**. The credential information may be stored and sent with future requests by a client that sent the request. For example, HTTP cookies, URL parameter schemes, or other HTTP headers may be used to retain the credential information for future requests.

[0022] In one embodiment, security analyzer module **114** enables the detection and assessment of security vulnerabilities associated with website **140A**. As website **140A** may

have references to more than one web server, security vulnerabilities may be associated with a plurality of web servers **130A** (e.g., web servers **130A-N**). Security vulnerabilities include but are not limited to application specific vulnerabilities, which are security holes within the website's own specific application, and security vulnerabilities embedded in client-side application files, such as Flash files or Java applets, that may be associated with the website **140A**, or other websites, and servers used to service the client-side application files. These security vulnerabilities arise from an application-level interfaces between the client and servers and include, but are not limited to, path vulnerabilities, parameter vulnerabilities, and the like.

[0023] In order to determine server-side security vulnerabilities associated with client-side application files, security analyzer module **114** may be configured to retrieve webpage contents of website **140A** to obtain paths (e.g., web addresses) and data parameters associated with web servers **130A** servicing instructions and content for the webpage page **140A**. In one embodiment, security analyzer module **114** is capable of automatically extracting web addresses and data parameters embedded in client-side application files referred to in the webpage content. This enables the security analyzer module **114** to discover the structure or elements of the of the client-side application interface with the web servers **130A**, particularly the path parameters and data parameters employed in the interface. Security analyzer module **114** is also capable of modifying the data extracted, transmitting the modified data and server request protocols to the respective web server **130A** via communication network **120**, and analyzing response from the web server **130A** in response to the modified data to determine server-side security vulnerabilities associated with the client-server interface.

[0024] For example, to discover server-side security vulnerabilities, security analyzer module **114** may be configured to retrieve and analyze webpage content of website **140A** to locate client-side application files referred to in the webpage content. Security analyzer module **114** is configured to extract path parameters and associated data parameters from the client-side application files which refer to web servers used to service the client-side application files content and instructions thereof. In one embodiment, security analyzer module **114** modifies the path parameters and/or extracted data parameters with user-defined criteria to form test data, transmits the test data to the servers, and analyzes the server responses for server-side security vulnerabilities. Any security vulnerabilities found may be summarized and provided to a user in a security vulnerability assessment as described further below. Advantageously, security analyzer module **114** may be used to detect security

vulnerabilities in the interface between clients and web servers before attackers gain access to personal information, such as user's credit card data and other personal information by exploiting those security vulnerabilities.

[0025] While security analyzer module **114** is capable of performing the processes of parsing and analysis automatically, it is contemplated that a user of the security analyzer module **114** may manually modify the parsing and/or analysis functions to, for example, customize parsing and/or analysis for any portions of the client-side application files and data parameters that meet the needs of a given application. For example, the user may modify the parsing and/or analysis process to eliminate searching for unnecessary files, paths, and the like. Advantageously, providing the user with the ability to manually modify the parsing and/or analysis processes provided by security analyzer module **114** may allow security analyzer module **114** to locate security vulnerabilities more quickly and efficiently.

[0026] FIG. 2 is a high-level block diagram of one embodiment of security analyzer module **114**. Security analyzer module **114** includes a security analyzer engine **202**, user-defined transformation criteria **204**, security test data structure **206**, application data structure **208**, and result data structure **210**. In one embodiment, security analyzer engine **202** may operate as a proxy as known in the art for controlling communication between web server **130A** and processing system **100**. Security analyzer engine **202** may also be configured to control the validation of requests transmitted from the processing system **100** and web server **130A**, for example, using the credential information.

[0027] In one embodiment, security analyzer engine **202** is a software module capable of manually or automatically assessing server-side security vulnerabilities using data embedded in client-side applications. Security analyzer engine **202** may be configured parse webpage content and client-side application files, such as Flash files, to discover URLs, data parameters, and the like. The client-side application files may be developed in any one of a vast option of languages (such as Flash, Perl, PHP, Java, C++, and the like) and may use one or more integration methods (CGIs, servlets, server APIs, etc.). The data parameters are used to call, initiate or trigger various functions, actions or logic of the servers servicing the client-side applications. Each such parameter may have a number of attributes such as data type, length, visibility, and value.

[0028] Once the embedded data has been extracted, security analyzer engine **202** modifies the path parameters and/or data parameters found, transmits requests to the discovered URLs

with the modified paths and/or parameters, analyzes the results of the responses, or lack thereof, from the respective web servers, and ranks the results. Such results may then be used to determine the security vulnerability of the server interface servicing the client-side application. A more detailed description of the processing steps carried out by security analyzer engine **202** is described further below with respect to FIGs. 3-5.

[0029] Transformation criteria **204** includes a plurality of user-defined data such as alphanumeric characters, symbols, and the like that when transmitted to a server **130A**, may, or may not, invoke a response from the server. For example, transformation criteria **204** may include a text character such as an apostrophe, a numeric character such as "1", an algorithm such as "1=1", a symbol, and the like. Such transformation criteria **204** when processed by a server **130A** as part of a request message, may invoke an error message, may invoke no-response, or may invoke an unexpected response from the server **130A** that may lead a user to discern security vulnerabilities, as described further below.

[0030] Transformation criteria **204** may be derived manually or automatically. In one embodiment, transformation criteria **204** is derived empirically from user's observations of security vulnerabilities with respect to, for example, user experimentation. In other embodiments, transformation criteria **204** may be derived automatically using one or more algorithms. In one embodiment, such algorithms employed may generate patterns of data that invoke patterns of web server responses. Such patterns may be used to detect security vulnerabilities.

[0031] Security test data **206** includes data associated with the process of server-side security vulnerability testing. For example, in one embodiment, security test data **206** may include the webpage content, server requests, web addresses extracted from the client-side applications, modified parameters, modified cookies, modified javascript, modified applet parameters, modified database parameters, and the like.

[0032] Application data **208** may include client-side application files referred to in the webpage being analyzed (e.g., webpage **140A**). Examples of client-side applications may include, but not are limited to, Sun Microsystems' Java Applets code, Macromedia's Flash Movies applications, XUL, and Microsoft's ActiveX Controls code. These types of applications provide an alternative application programming interface to web server **130A** that may be separate from the typical HTML and Javascript interfaces.

[0033] Results data structure **210** includes results associated with the process of server-side security vulnerability testing. For example, results data **210** may include errors and messages received from servers in response to the transformation criteria **204**, a listing of security issues, a user-defined relationship between errors and transformation criteria **204**, reports, security assessments, and the like.

[0034] FIG. 3 is flow diagram illustrating a method **300** of assessing web application security vulnerabilities. In one embodiment, method **300** is initiated at step **302** when, for example, processing system **102** is used to analyze a webpage for client side security vulnerabilities. At step **304**, webpage content is received by security analyzer engine **202** for processing. For example, as illustrated in Fig. 4A, a webpage content requested from URL "HTTP://www.coopertest.com" is located and stored in security test data structure **206** for processing by security analyzer engine **202**.

[0035] Depending on the type of client-side application, different techniques may be employed to identify the available URLs contained within the client-side application. For example, at step **306**, security analyzer engine **202** parses the contents to locate references (e.g., paths) to client-side applications such as Java applets, Flash files, and the like. For example, FIG. 4B illustrates the content of webpage "www.coopertest.com" having references to a Flash file location "path_to_flash_movie.swf." and applet with references to an applet file path_to_applet.class.

[0036] In one embodiment, a user may modify the parsing step to tailor the parsing to a particular need. For example, the user may modify the parsing process to eliminate searching for image files such as GIF, JPEG, BMP, and the like. In other embodiments, the user may modify the parsing step to search for references to specific file types such as Flash applications, Applet applications, and other types of client-side application files. Advantageously, providing the user with the ability to modify the parsing step allows security analyzer engine **202** to parse the webpage content more quickly to help a user locate security vulnerabilities more quickly and efficiently.

[0037] At step **308**, security analyzer engine **202** retrieves the client-side application files discovered during parsing and stores them, for example, in application data structure **208**. Security analyzer engine **202** extracts path parameters and associated data parameters from the client-side application files. In one embodiment, security analyzer engine **202** parses the client-side application to extract the path parameters and associated data parameters from the

client-side application files using a variety of methodologies. This may be done when, for example, security analyzer engine **202** is configured to search through the client-side application file for the path parameters and associated data parameters. For example, to parse the client-side application file, the security analyzer engine **202** may use the file format of the client-side application file published, for example, by the vendor of the client-side application. In other embodiments, the format may be learned through reverse engineering procedures. Alternatively, the security analyzer engine **202** may analyze the client-side application file code, which may be encoded in various formats, to identify data matching the standard structure of path and data parameters.

[0038] In one embodiment, security analyzer engine **202** employs binary analysis to find path parameters and associated data parameters from the client-side application files. For example, path parameters and associated data parameters may be part of an ASCII string in a binary file. Once located, security analyzer engine **202** searches the strings for indications of path parameters and associated data parameter within the strings. For example, the string may include path parameters such as "http", "/", www, and the like.

[0039] In another embodiment, security analyzer engine **202** employs run-time monitoring to derive path parameters and associated data parameters. In this embodiment, a client-side application file is activated in a "sandbox" environment to simulate the execution process of a client-side application in a controlled environment. Sandboxing refers to a process of activating the client-side application file in an isolated environment and monitoring its operation. In the sandbox environment, upon execution the client-side application file may attempt to make outbound HTTP requests. If the client-side application attempts to remotely connect, security analyzer engine **202** captures path parameters and associated data parameters from such HTTP requests sent, for example, from processing system **102** to web server **140A**, via network **120**. The captured path parameters and associated data parameters may be stored, for example, in security test data structure **206**, for real-time or later analysis.

[0040] At step **310**, once the path parameters and associated data parameters are determined, the next step is to analyze messages that flow or would flow between a client and a web server in order to discover data input points of the web application interface with external clients. Examples include links, forms, fixed fields, hidden fields, menu options, etc. The security analyzer engine **202** may send an HTTP request derived from each of the path parameters (e.g., URL) identified in the previous step and receive the associated responses

from the web server **130A**. The web server responses can then be analyzed to discover the various URLs and data input points contained within the website and web applications.

[0041] Web applications interface with clients using a multitude of data parameters that may be data input points. Security analyzer engine **202** may employ a variety of techniques to discover such data input points. In one embodiment, for well understood URL structures which usually exhibit a well known format with a query string "? name1=value1&name2=value2", security analyzer engine **202** may parse the URL string to extract "value1" and "value2". For example, as illustrated in FIG. 4C, employing the extraction process step, security analyzer engine **202** extracted http://host/webapp.cgi?name=ActualValue 406, from the flash file "flash_movie.swf". In this example, the path parameter http://host/webapp.cgi? and associated data parameter "ActualValue" were embedded in flash_movie.swf. In this illustration, the data parameter "ActualValue" represents a plurality of application data entry parameters, symbols, algorithms, etc. that are provided by, for example, the company who generated flash_movie.swf. In one embodiment, security analyzer engine **202** marks the data entry parameter "ActualValue" as a data entry point that may be stored, for example, in security tests data structure **206**.

[0042] In other embodiments, for non-standard URL structures, security analyzer engine **202** may employ a deterministic approach to locate data input points. For example, security analyzer engine **202** may compare a set of URLs, examining where the URL's change on byte-by-byte position basis. Changes in bytes with respect to given byte position between URLs may represent a data input point.

[0043] At step **312**, security analyzer engine **202** generates modified data entry parameters by replacing the ones discovered during the parsing step with user-defined transformation criteria. Such user-defined transformation criteria may be derived from virtually any source and may be formed from alphanumeric characters, symbols, algorithms, binary strings, and the like, with respect to the type of path parameters and client-side applications being analyzed. For example, as illustrated in FIG. 5A, transformation criteria may be represented by the term "TestCriteria" which can take on a plurality of attributes with respect to common gateway interface (CGI) web-application "webapp.cgi" (CGI is a standard for interfacing external applications with information servers, such as HTTP or web servers). In one example, as shown in FIG. 5B, TestCriteria is an apostrophe (') character. In another example, as shown in FIG. 5C, TestCriteria is an algorithm "1=1".

[0044] In one embodiment, security analyzer engine **202** retrieves predetermined transformation criteria from transformation criteria data structure **204**. Such predetermined transformation criteria may be derived empirically from user observations of responses from the web server 130 with respect to known web server responses and user observed responses that may lead the user to deduce a security vulnerability. For example, a user may rely on a string such as "SH1214!SGS#&" that is known to produce an error message, such as "DNS Error", from web servers most of the time.

[0045] A user may develop other transformation criteria that produces unexpected results recognized by the user but not necessarily recognized by others. In one embodiment, the user may develop predefined text field parameters, numeric filed parameters, database parameters (e.g., SQL, etc.), parameter names, additional path parameters, configured to invoke a known server response or a server response recognized by the user but not necessarily recognized by others. The predetermined transformation criteria may be stored in, for example, security tests data structure **206**.

[0046] At step **314**, security analyzer engine **202** transmits the transformation criteria to its respective web server according to the path parameters. For example, as illustrated in FIG. 5A a request **506** containing the data parameter "TestCriteria" **504** is transmitted to a web server called "host". As shown in FIG. 5B and 5C, the data parameter for "TestCriteria" **504** is an apostrophe and an algorithm 1=1, respectively, sent to the "host" web server. Security analyzer engine **202** is configured to transmit the transformation criteria any number of ways. For example, the transformation criteria may be sent as a HTTP request, such as GET or POST as known in the art.

[0047] At step **316**, security analyzer engine **202** receives responses, or no responses, from the web servers (e.g., web server **130A-N**). As described herein, the responses, or lack thereof, may be known, unexpected, or known to a user of the security analyzer engine **202**. Security analyzer engine **202**, stores the responses in results data structure **210**, and examines the responses, or lack thereof, from the web servers responding to the transformation criteria to determine if any security vulnerabilities exist.

[0048] In one embodiment, the security analyzer engine **202** compares the responses received to a list of responses and associated security vulnerabilities stored, for example, in security test data structure **206**. For example, as illustrated in FIG. 5A, an analysis process provides a rank of security vulnerability test **510**. As illustrated at FIG. 5B, a web server

provides a response **508** of "ODBC Drivers error '80040e14" in response to the modified transformation criteria of "name=' " where the modified data parameter sent was an apostrophe ('). Security analyzer engine **202** compares the response "ODBC Drivers error '80040e14" **508** to a list of responses and corresponding security vulnerabilities that are well known or defined by the user, and provides a risk analysis **510** of "Level A". As illustrated at FIG. 5C, a web server provides a response **508** of "file system error" in response to the modified transformation criteria of "name=1=1 ", where the modified data parameter sent was the algorithm "1=1". Security analyzer engine **202** compares the response "file system error" **508** to a list of responses and corresponding security vulnerabilities and provides a risk analysis **510** of "Level B" , for example, where "B" may be indicative of a greater security risk than level "A". Security analyzer engine **202** may compile the risk analyses to form a risk security vulnerability assessment. Security analyzer engine **202** ends the method **300** at step **318**.

[0049] The present invention can be implemented in the form of control logic in software or hardware or a combination of both. The control logic may be stored in an information storage medium as a plurality of instructions adapted to direct an information-processing device to perform a set of steps disclosed in embodiments of the present invention. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the present invention.

[0050] The above description is illustrative but not restrictive. Many variations of the invention will become apparent to those skilled in the art upon review of the disclosure. The scope of the invention should, therefore, be determined not with reference to the above description, but instead should be determined with reference to the pending claims along with their full scope or equivalents.

WHAT IS CLAIMED IS:

1. A method for analyzing security vulnerabilities in web-based applications, the method comprising:

extracting path parameters and data parameters from the client-side applications referring to web servers servicing the client-side applications;

modifying the path parameters or data parameters with user-defined data to generate test data;

transmitting the test data to the web servers; and

determining if any of responses received in response to the test data are indicative of security vulnerabilities.

2. The method of claim 1, further comprising parsing web page content to determine locations to client-side applications.

3. The method of claim 1, wherein extracting comprises searching for one or more web application URLs contained within a compiled client-side software application or file associated with the client-side software application.

4. The method of claim 1, wherein transmitting the test data comprises generating HTTP requests for transmission to at least some of the web servers.

5. The method of claim 1, wherein extracting the web addresses comprises simulating the execution process of a client-side application and capturing remote connection attempts to one or more of the web servers.

6. The method of claim 1, wherein determining comprises transmitting an HTTP request derived from at least some of the path parameters and receiving responses from the servers in response to the HTPP request.

7. The method of claim 1, wherein determining comprises searching a database of web server messages and associated security vulnerabilities.

8. A computer readable medium, computational apparatus, or server computer comprising computer code for performing the method of claim 1.

9. A method of detecting security vulnerabilities in web-based applications, the method comprising:

extracting data from client-side applications to locate data entry points to web servers used to service the client-side applications; and

applying user-defined test criteria to the data entry points, wherein the user-defined testing data is configured to cause a web server to respond with one or more predetermined responses configured to expose security vulnerabilities in client-server interfaces associated with the data input points.

10. The method of claim 9, further comprising parsing webpage content to locate the data entry points.

11. The method of claim 9, wherein extracting data comprises parsing a client-side application file to extract the data.

12. The method of claim 9, wherein extracting data comprises operating the client-side application and monitoring the communication between the client-side application and a web server used to service the client-side application.

13. The method of claim 9, wherein extracting data comprises monitoring changes in data parameter byte positions with respect to a plurality of URLs.

14. A computer readable medium, computational apparatus, or server computer comprising computer code for performing the method of claim 9.

15. A security analyzer for detecting security vulnerabilities in client-server interfaces, the security analyzer comprising:

a security analyzer engine configured to extract client-server interface data embedded in client-side application files, generate user-defined testing data from the client-server interface data, wherein the user-defined testing data is configured to cause a web server to respond with one or more predetermined responses that are used to expose security vulnerabilities in client-server interfaces associated with the client-side application files.

16. The security analyzer of claim 15, wherein the security analyzer engine is configured to extract the location of the client-side application from webpage content.

17. The security analyzer of claim 15, wherein the security analyzer engine analyzes the client-side application files to determine one or more addresses of web-servers used to service the client-side applications.

18. The security analyzer of claim 15, wherein the security analyzer engine is configured to parse the client-side applications for path parameters or data parameters.

19. The security analyzer of claim 18, wherein the path parameters comprise URLs.

20. The security analyzer of claim 15, wherein the security analyzer engine is configured to monitor changes in data parameter byte positions with respect to a plurality of URLs.

21. The security analyzer of claim 15, wherein the security analyzer engine is configured to monitor client-server communication of at least one client-side application.

22. The security analyzer of claim 15, wherein the user-defined testing data comprises user-defined transformation criteria specifically named or selected by a user.

23. The security analyzer of claim 15, wherein the security analyzer engine replaces extracted path parameters or data parameters with user-defined testing criteria configured to invoke a response from a web server indicative of one or more security vulnerabilities.

24. The security analyzer of claim 23, wherein the user-defined testing criteria comprises a string of alphanumeric characters, or an algorithm, or symbol.

25. The security analyzer of claim 23, wherein the user-defined testing criteria comprises transformation criteria that produces unexpected results recognized by the user as one or more potential security vulnerabilities.

26. The security analyzer of claim 15, wherein the interface data comprises forms, or fixed fields, or hidden fields, or menu options.

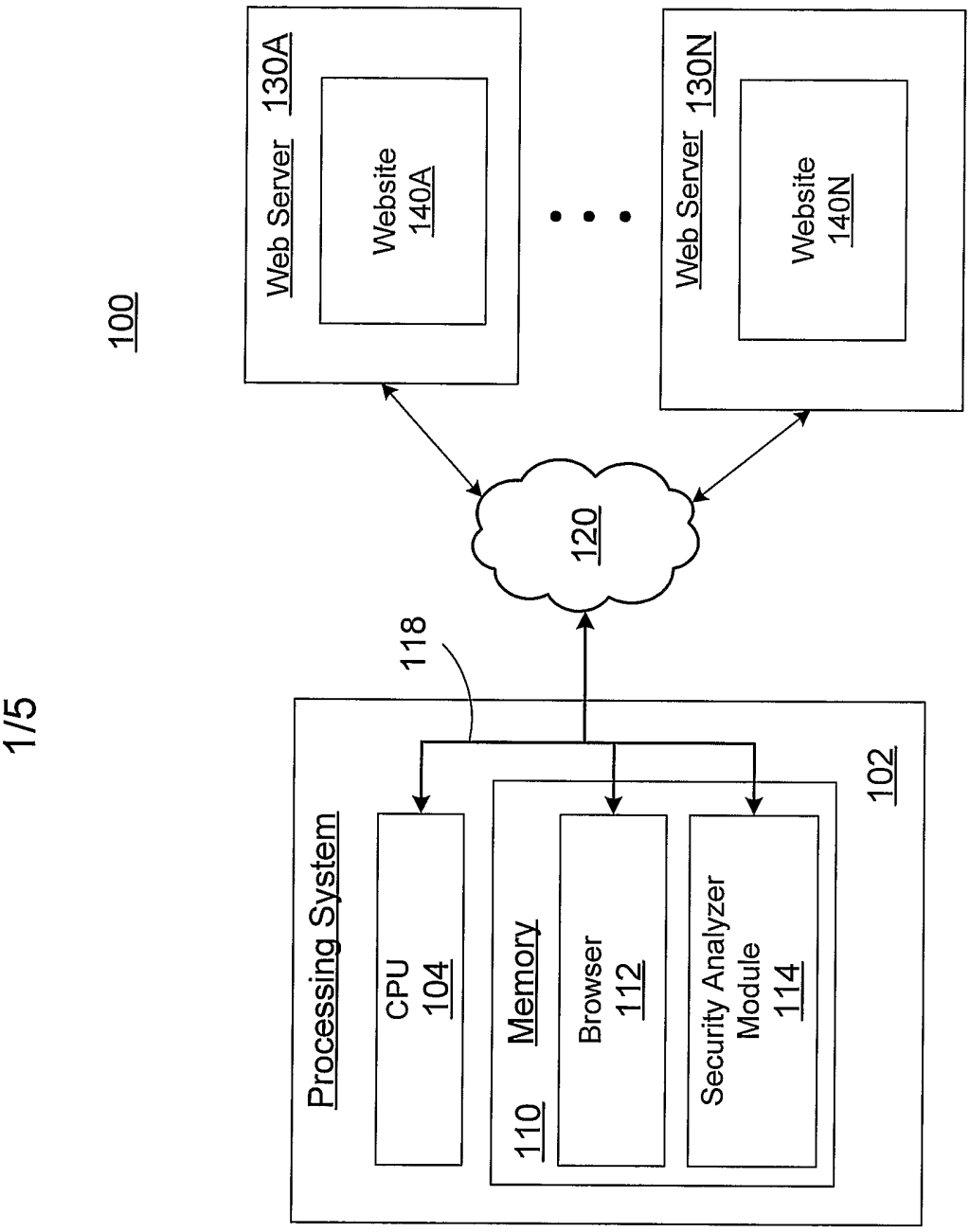

100
Web Server 130A
Website 140A
Web Server 130N
Website 140N
120
118
Processing System
CPU 104
Memory 110
Browser 112
Security Analyzer Module 114
102

FIG. 1

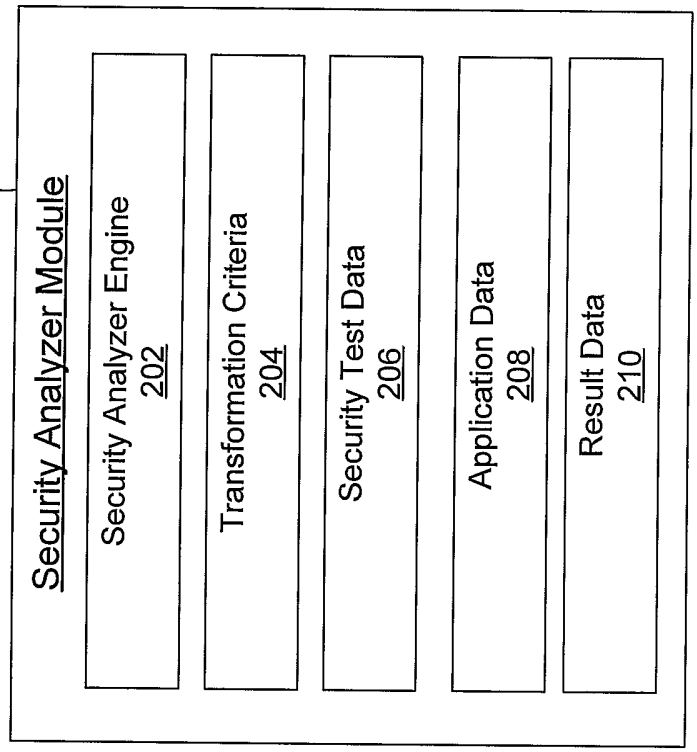

114
Security Analyzer Module
Security Analyzer Engine
202
Transformation Criteria
204
Security Test Data
206
Application Data
208
Result Data
210

FIG 2

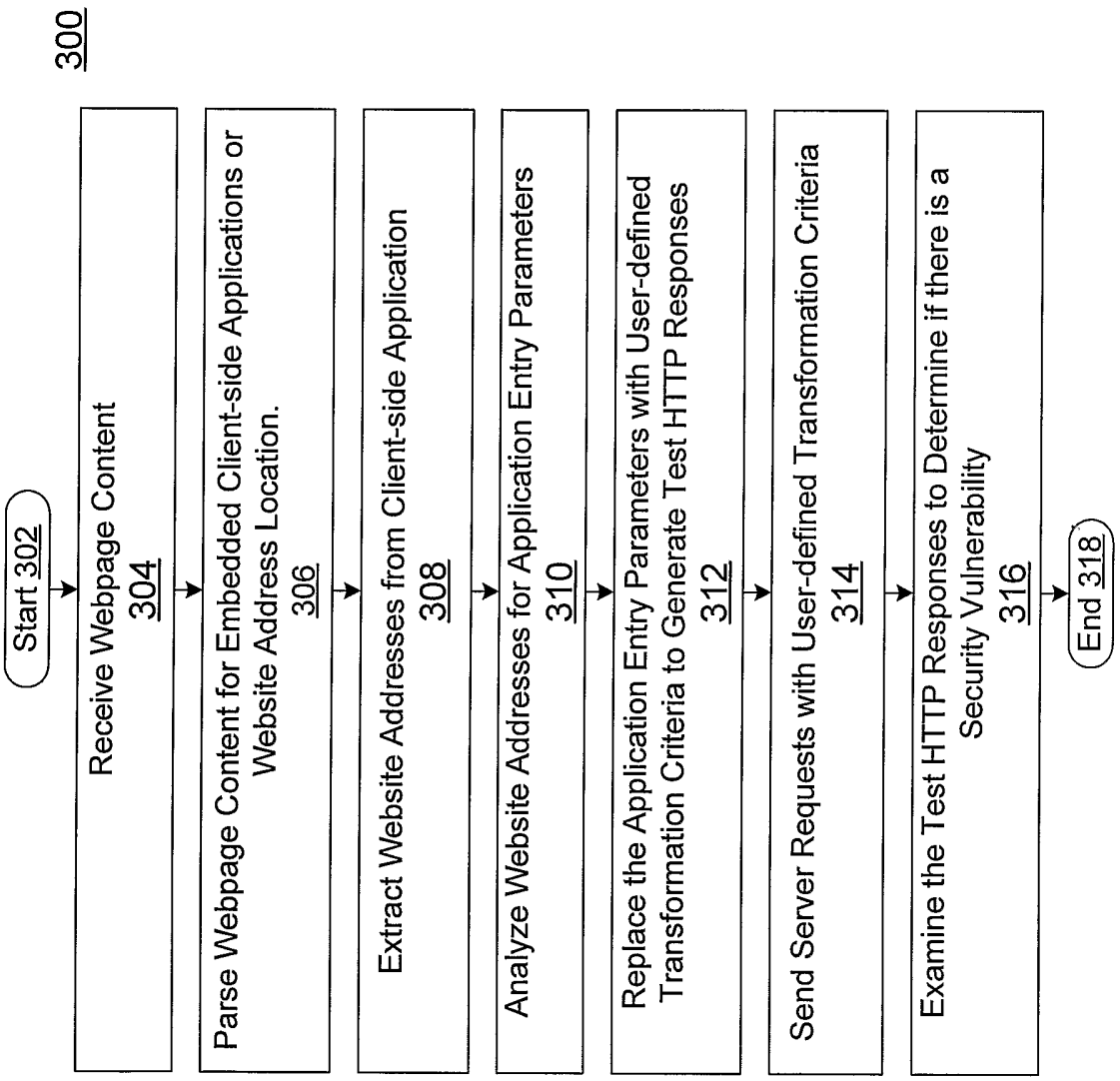

300
Start 302
Receive Webpage Content 304
Parse Webpage Content for Embedded Client-side Applications or Website Address Location. 306
Extract Website Addresses from Client-side Application 308
Analyze Website Addresses for Application Entry Parameters 310
Replace the Application Entry Parameters with User-defined Transformation Criteria to Generate Test HTTP Responses 312
Send Server Requests with User-defined Transformation Criteria 314
Examine the Test HTTP Responses to Determine if there is a Security Vulnerability 316
End 318

FIG. 3

FIG. 4A



FIG. 4B

```
<html>
<object classid="clsid:D2" codebase="http://download.coopertest.com/cabs/flash/swflash.cab">
<param  value="path_to_flash_movie.swf" />
<embed  src="path_to_flash_movie.swf" type="application/x-shockwave-
flash" pluginspage="http://www.macromedia.com/go/getflashplayer"></embed>
</object>

<applet code=path_to_applet.class>
<param name="ActualName1" value="ActualValue1">
</applet>
</html>
```

404

FIG. 4C

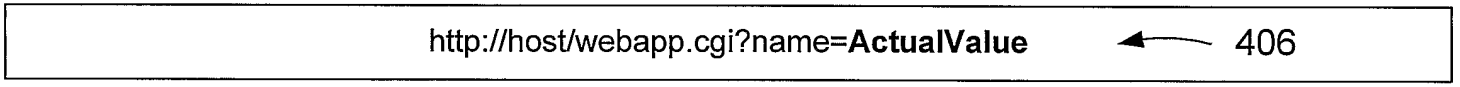


FIG. 5A

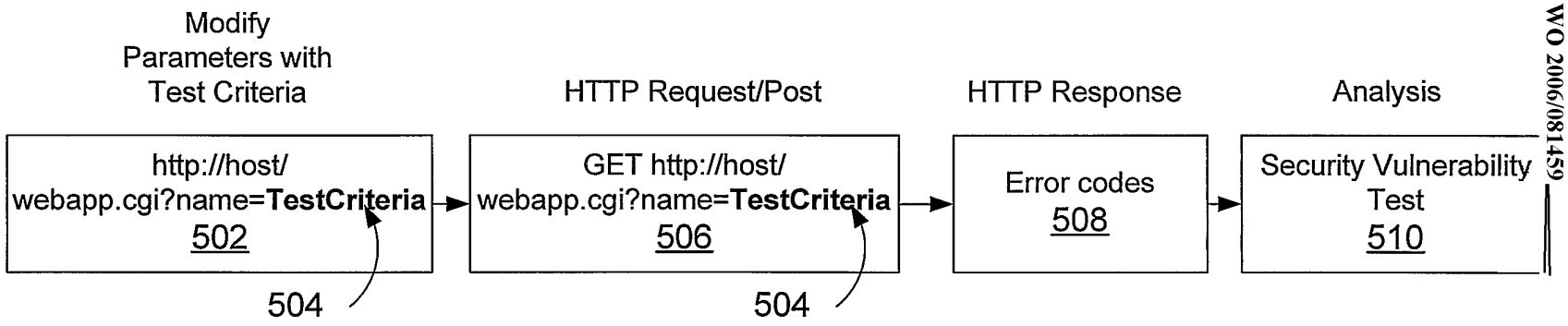


FIG. 5B

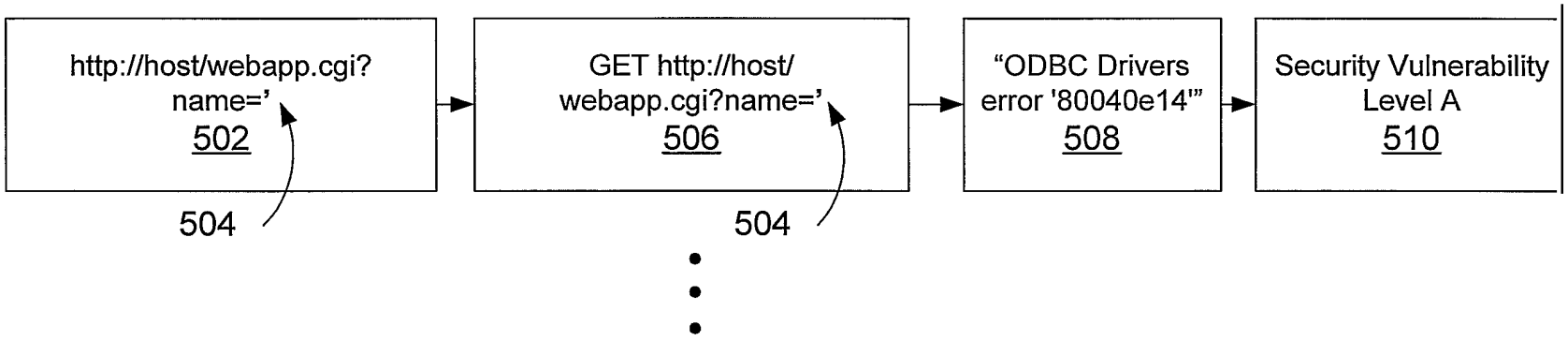


FIG. 5C

