Scripting and Security in Computer Networks and Web Browsers

6

6.1 Introduction

The rapid growth of the Internet and its ability to offer services have made it the fastest-growing medium of communication today. Today's and tomorrow's business transactions involving financial data; product development and marketing; storage of sensitive company information; and the creation, dissemination, sharing, and storing of information are and will continue to be made online, most specifically on the Web. The automation and dynamic growth of an interactive Web has created a huge demand for a new type of Web programming to meet the growing demand of millions of Web services from users around the world. Some services and requests are tedious and others are complex, yet the rate of growth of the number of requests, the amount of services requested in terms of bandwidth, and the quality of information requested warrant a technology to automate the process. Script technology came in timely to the rescue. Scripting is a powerful automation technology on the Internet that makes the Web highly interactive.

Scripting technology is making the Web interactive and automated as Web servers accept inputs from users and respond to user inputs. While scripting is making the Internet and, in particular, the Web is alive and productive, it also introduces a huge security problem to an already security-burdened cyberspace. Hostile scripts embedded in Web pages, as well as HTML formatted e-mail, attachments, and applets introduce a new security paradigm in cyberspace security. In particular, security problems are introduced in two areas: at the server and at the client. Before we look at the security at both of these points, let us first understand the scripting standard.

J.M. Kizza, *Guide to Computer Network Security*, Computer Communications and Networks, DOI 10.1007/978-3-319-55606-2_6

6.2 Scripting

A program script is a logical sequence of line commands which causes the computer to accomplish some task. Many times we refer to such code as *macros* or *batch files* because they can be executed without user interaction. A script language is a programming language through which you can write scripts. Scripts can be written in any programming language or a special language as long as they are surrogated by another program to interpret and execute them on the fly by a program unlike compiled programs which are run by the computer operating system.

Because scripts are usually small programs, written with a specific purpose in mind to perform tasks very quickly and easily, many times in constrained and embedded environments with abstracted performance and safety, unlike general-purpose programs written in general-purpose programming languages, scripts are not in most cases full-featured programs, but tend to be "glue" programs that hold together other pieces of software.

Therefore, scripting languages are not your general-purpose programming languages. Their syntax, features, library, etc. are focused more around accomplishing small tasks quickly. The scripts can be either application scripts, if they are executed by an application program surrogate like Microsoft spreadsheet, or command line scripts if they are executed from a command line like the Windows or Unix/Linux command line.

6.3 Scripting Languages

CGI scripts can be written in any programming language. Because of the need for quick execution of the scripts both at the server and in the client browsers and the need of not storing source code at the server, it is getting more and more convenient to use scripting languages that are interpretable instead of languages that are compiled like C and C++. The most common scripting languages include:

- JavaScript
- Perl
- Tcl/Tk
- Python
- VBA
- Windows Script Host
- Others including specific mobile device scripting languages

There are basically two categories of scripting languages, those whose scripts are on the server side of the client-server programming model and those whose scripts are on the client side.

6.3.1 Server-Side Scripting Languages

Ever since the debut of the World Wide Web and the development of HTML to specify and present information, there has been a realization that HTML documents are too static.

There was a need to put dynamism into HTTP so that the interaction between the client and the server would become dynamic. This problem was easy to solve because the hardware on which Web server software runs has processing power and many applications such as e-mail, database manipulation, or calendaring already installed and ripe for utilization [1]. The CGI concept was born.

Among the many sever-side scripting languages are ERL, PHP, ColdFusion, ASP, MySQL, Java servlets, and MivaScript.

6.3.1.1 Perl Scripts

Practical Extraction and Report Language (Perl) is an interpretable programming language that is also portable. It is used extensively in Web programming to make text processing interactive and dynamic. Developed in 1986 by Larry Wall, the language has become very popular. Although it is an interpreted language, unlike C and C++, Perl has many features and basic constructs and variables similar to C and C++. However, unlike C and C++, Perl has no pointers and defined data types.

One of the security advantages of Perl over C, say, is that Perl does not use pointers where a programmer can misuse a pointer and access unauthorized data. Perl also introduces a gateway into Internet programming between the client and the server. This gateway is a security gatekeeper scrutinizing all incoming data into the server to prevent malicious code and data into the server. Perl does this by denying programs written in Perl from writing to a variable, whereby this variable can corrupt other variables.

Perl also has a version called Taintperl that always checks data dependencies to prevent system commands from allowing untrusted data or code into the server.

6.3.1.2 PHP

PHP (Hypertext Preprocessor) is a widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. It is an open-source language suited for Web development, and this makes it very popular.

Just like Perl, PHP code is executed on the server, and the client just receives the results of running a PHP script on the server. With PHP, you can do just about anything other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies.

6.3.2 Client-Side Scripting Languages

The World Wide Web (WWW) created the romance of representing portable information from a wide range of applications for a global audience. This was

accomplished by the HyperText Markup Language (HTML), a simple markup language that revolutionized document representation on the Internet. But for a while, HTML documents were static. The scripting specifications were developed to extend HTML and make it more dynamic and interactive. Client-side scripting of HTML documents and objects embedded within HTML documents has been developed to bring dynamism and automation of user documents. Scripts including JavaScript and VBScript are being used widely to automate client processes.

For a long time, during the development of CGI programming, programmers noticed that much of what CGI does, such as maintaining a state, filling out forms, error checking, or performing numeric calculation, can be handled on the client's side. Quite often, the client computer has quite a bit of CPU power idle, while the server is being bombarded with hundreds or thousands of CGI requests for the mundane jobs above. The programmers saw it justifiable to shift the burden to the client, and this led to the birth of client-side scripting.

Among the many client-side scripting languages are DTML/CSS, Java, JavaScript, and VBScript.

6.3.2.1 JavaScripts

JavaScript is a programming that performs client-side scripting, making Web pages more interactive. Client-side scripting means that the code works only on the user's computer, not on the server side. It was developed by Sun Microsystems to bridge the gap between Web designers who needed a dynamic and interactive Web environment and Java programmers. It is an interpretable language like Perl. That means the interpreter in the browser is all that is needed for a JavaScript to be executed by the client and it will run. JavaScript's ability to run scripts on the client's browser makes the client able to run interactive Web scripts that do not need a server. This feature makes creating JavaScript scripts easy because they are simply embedded into any HTML code. It has, therefore, become the de facto standard for enhancing and adding functionality to Web pages.

This convenience, however, creates a security threat because when a browser can execute a JavaScript at any time, it means that hostile code can be injected into the script and the browser would run it from any client. This problem can be fixed only if browsers can let an executing script perform a limited number of commands. In addition, scripts run from a browser can introduce into the client systems programming errors in the coding of the script itself, which may lead to a security threat in the system itself.

6.3.2.2 VBScript

Based in part on the popularity of the Visual Basic programming language and on the need to have a scripting language to counter JavaScript, Microsoft developed VBScript (V and B for Visual Basic). VBScript has a syntax similar to the Visual Basic programming language syntax. Since VBScript is based on Microsoft Visual Basic and unlike JavaScript which can run in many browsers, VBScript interpreter is supported only in the Microsoft Internet Explorer.

6.4 Scripting in Computer Network

The use of scripting in computer network started when network administrators released that one of the best ways to tame the growing mountain of tasks required to have a well functioning network was to use scripting and take advantage of the automated natured of execution of scripts via surrogate programs. This cut down on the need for attention for many, sometimes repeated tasks, in the running of a system network.

According to Allen Rouse [2], scripting lets you automate various network administration tasks, such as those that are performed every day or even several times a day and those performed widely throughout the network like log-in scripts and modification to the registry scripts in a widely distributed network of servers. There are many functions in a daily administration of a network that are performed by scripts including [2]:

- Machine startup
- Machine shutdown
- User log-in and log-out
- Scripting basic TCP/IP networking on clients, including comparisons of GUI and command line tools to analogous scripting techniques
- Extending these scripting techniques to remote and multiple computers
- IP address allocation with DHCP and static IP addresses
- DNS client management
- WINS client management
- TCP/IP filtering, Internetwork Packet Exchange (IPX), and other network protocols
- Managing system time and network settings in the registry
- Addition of new clients to a network
- Client-server information exchange

With all these tasks taken over by scripting, you may notice the tremendous advantages scripting brings to the table for system administration in time savings, network consistence, and flexibility.

By every account, scripting is on the rise with the changing technologies. There is tremendous enthusiasm for growth in the four traditional areas of scripting that include:

- System administration
- Graphical user interface (GUI)
- Internet information exchange (CGI) and the growing popularity of the browser
- Application and component frameworks like ActiveX and others

We will focus on the Internet information exchange in this chapter.

6.4.1 Introduction to the Common Gateway Interface (CGI)

One of the most essential useful areas in network performance when scripting plays a vital role is in the network client-server information exchange. This is done via a *Common Gateway Interface* or *CGI*. CGI is a standard to specify a data format that servers, browsers, and programs must use in order to exchange information. A program written in any language that uses this standard to exchange data between a Web server and a client's browser is a *CGI script*. In other words, a CGI script is an external gateway program to interface with information servers such as HTTP or Web servers and client browsers. CGI scripts are great in that they allow the Web servers to be dynamic and interactive with the client browser as the server receives and accepts user inputs and responds to them in a measured and relevant way to satisfy the user. Without CGI, the information the users would get from an information server would not be packaged based on the request but based on how it is stored on the server.

CGI programs are of two types: those written in programming languages such as C/C++ and Fortran that can be compiled to produce an executable module stored on the server and scripts written in scripting languages such as PERL, Java, and Unix shell. For these CGI scripts, no associated source code needs to be stored by the information server as is the case in CGI programs. CGI scripts written in scripting languages are not compiled like those in nonscripting languages. Instead, they are text code which is interpreted by the interpreter on the information server or in the browser and run right away. The advantage to this is you can copy your script with little or no changes to any machine with the same interpreter and it will run. In addition, the scripts are easier to debug, modify, and maintain than a typical compiled program.

Both CGI programs and scripts, when executed at the information server, help organize information for both the server and the client. For example, the server may want to retrieve information entered by the visitors and use it to package a suitable output for the clients. In addition, GCI may be used to dynamically set field descriptions on a form and in real time inform the user on what data has been entered and yet to be entered. At the end, the form may even be returned to the user for proofreading before it is submitted.

CGI scripts go beyond dynamic form filling to automating a broad range of services in search engines and directories and taking on mundane jobs such as making download available, granting access rights to users, and order confirmation.

As we pointed out earlier, CGI scripts can be written in any programming language that an information server can execute. Many of these languages include script languages such as Perl, JavaScript, TCL, Applescript, Unix shell, VBScript, and nonscript languages such as C/C++, Fortran, and Visual Basic. There is dynamism in the languages themselves; so, we may have new languages in the near future.

6.4.1.1 CGI Scripts in a Three-Way Handshake

As we discussed in Chap. 3, the communication between a server and a client opens with the same etiquette we use when we meet a stranger. First, a trust relationship must be established before any requests are made. This can be done in a number of ways. Some people start with a formal "Hello, I'm... " and then "I need ..." upon which the stranger says "Hello, I'm ..." and then "Sure I can ...." Others carry it further to hugs, kisses, and all other ways people use to break the ice. If the stranger is ready for a request, then this information is passed back to you in a form of an acknowledgement to your first embrace. However, if the stranger is not ready to talk to you, there is usually no acknowledgment of your initial advances, and no further communication may follow until the stranger's acknowledgment comes through. At this point, the stranger puts out a welcome mat and leaves the door open for you to come in and start business. Now, it is up to the initiator of the communication to start full communication.

When computers are communicating, they follow these etiquette patterns and protocols, and we call this procedure a handshake. In fact, for computers it is called a three-way handshake. A three-way handshake starts with the client sending a packet, called a *SYN* (short for synchronization), which contains both the client and server addresses together with some initial information for introductions. Upon receipt of this packet by the server's welcome open door, called a *port*, the server creates a communication socket with the same port number such as the client requested through which future communication with the client will go. After creating the communication socket, the server puts the socket in queue and informs the client by sending an acknowledgment called a *SYN-ACK*. The server's communication socket will remain open and in queue waiting for an ACK from the client and data packets thereafter. The socket door remains half-open until the server sends the client an ACK packet signaling full communication. During this time, however, the server can welcome many more clients that want to communicate, and communication sockets will be opened for each.

The CGI script is a server-side language that resides on the server side, and it receives the client's SYN request for a service. The script then executes and lets the server and client start to communicate directly. In this position, the script is able to dynamically receive and pass data between the client and server. The client browser has no idea that the server is executing a script. When the server receives the script's output, it then adds the necessary protocol data and sends the packet or packets back to the client's browser. Figure 6.1 shows the position of a CGI script in a three-way handshake.

The CGI scripts reside on the server side, in fact on the computer on which the server is, because a user on a client machine cannot execute the script in a browser on the server; one can view only the output of the script after it executes on the server and transmits the output using a browser on the client machine the user is on.

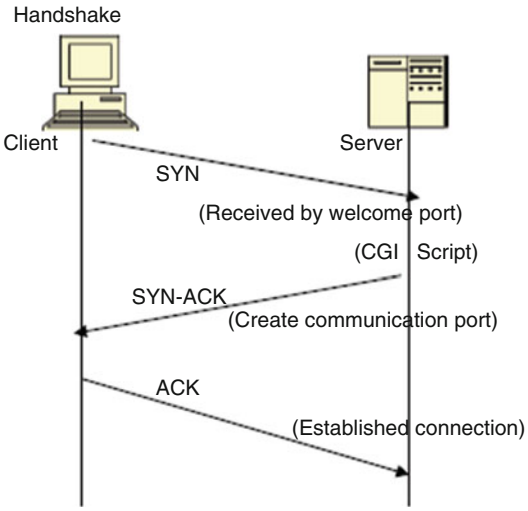

Fig. 6.1 The position of a CGI script in a three-way handshake

6.4.2 Server-Side Scripting: The CGI Interface

In the previous section, we stated that the CGI script is on the server side of the relationship between the client and the server. The scripts are stored on the server and are executed by the server to respond to the client demands. There is, therefore, an interface that separates the server and the script. This interface, as shown in Fig. 6.2, consists of information from the server supplied to the script that includes input variables extracted from an HTTP header from the client and information from the script back to the server. Output information from the server to the script and from the script to the server is passed through environment variables and through script command lines. Command line inputs instruct a script to do certain tasks such as search and query.

6.5 Computer Networks Scripts and Security

As we have pointed out above, by all accounts, scripting is on the rise with the changing technologies. There is tremendous enthusiasm for growth in the four traditional areas of scripting that include:

- System administration
- Graphical user interface (GUI)
- Internet information exchange (CGI) and the growing popularity of the browser
- Application and component frameworks like ActiveX and others

As scripting grows, so are will its associated security problems. Hackers are constantly developing and testing a repertoire of their own scripts that will compromise other scripts wherever they are on the Web, in the computer network system or

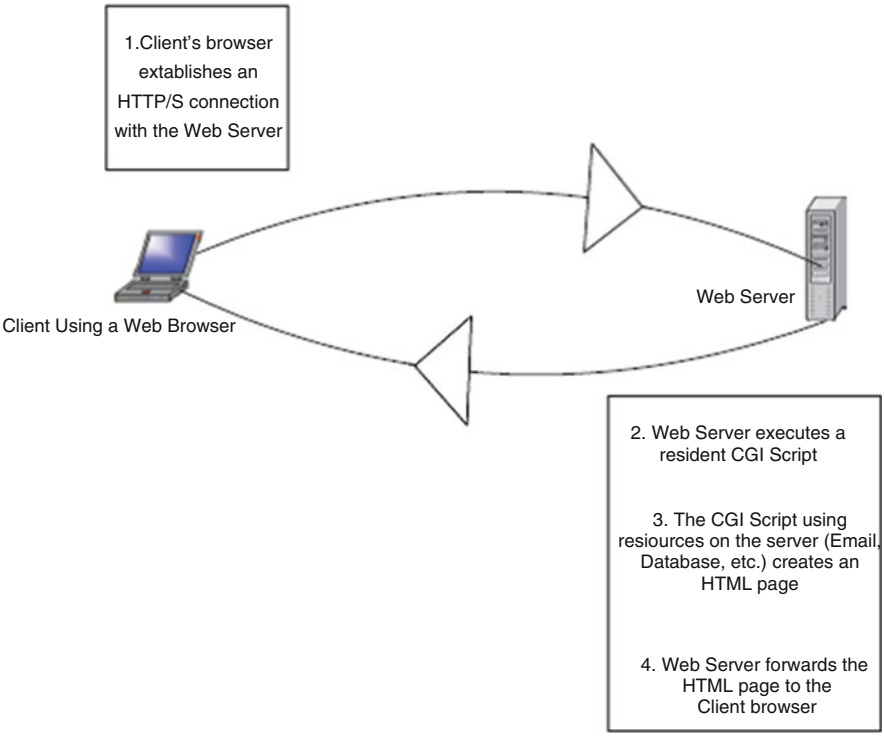

Fig. 6.2 A client CGI script interface

in applications. The most common of such hacker techniques today include Web cross-site scripting or XSS or CSS. Cross-site scripting allows attackers of Web sites to embed malicious scripts into dynamic unsuspecting Web and network scripts. Although this is a threat to most scripts, we will focus our script security discussion on the CGI scripts.

6.5.1 CGI Script Security

To an information server, the CGI script is like an open window to a private house where passersby can enter the house to request services. It is an open gateway that allows anyone anywhere to run an executable program on your server and even send their own programs to run on your server. An open window like this on a server is not the safest thing to have, and security issues are involved. But since CGI scripting is the fastest-growing component of the Internet, it is a problem we have to contend with and meet head on. CGI scripts present security problems to cyberspace in several ways including:

- *Program development*: During program development, CGI scripts are written in high-level programming language and compiled before being executed, or they are written in a scripting language and they are interpreted before they are executed. In either way, programming is more difficult than composing documents with HTML, for example. Because of the programming complexity and owing to lack of program development discipline, errors introduced into the program are difficult to find, especially in noncompiled scripts.
- *Transient nature of execution*: When CGI scripts come into the server, they run as separate processes from that of the host server. Although this is good because it isolates the server from most script errors, the imported scripts may introduce hostile code into the server.
- *Cross-pollination*: The hostile code introduced into the server by a transient script can propagate into other server applications and can even be retransmitted to other servers by a script bouncing off this server or originating from this server.
- *Resource guzzling*: Scripts that are resource intensive could cause a security problem to a server with limited resources.
- *Remote execution*: Since servers can send CGI scripts to execute on surrogate servers, both the sending and receiving servers are left open to hostile code usually transmitted by the script.

In all these situations, a security threat occurs when someone breaks into a script. Broken scripts are extremely dangerous.

Kris Jamsa gives the following security threats that can happen to a broken script [2]:

- Giving an attacker access to the system's password file for decryption.
- Mailing a map of the system which gives the attacker more time offline to analyze the system's vulnerabilities.
- Starting a log-in server on a high port and telneting in.
- Beginning a distributed denial-of-service attack against the server.
- Erasing or altering the server's log files.
- In addition to these others, the following security threats are also possible [3]:
  - Malicious code provided by one client for another client: this can happen, for example, in sites that host discussion groups where one client can embed malicious HTML tags in a message intended for another client. According to the Computer Emergency Response Team (CERT), an attacker might post a message like:
    Hello message board. This is a message.
    <SCRIPT>malicious code</SCRIPT>
    This is the end of my message.
    When a victim with scripts enabled in his or her browser reads this message, the malicious code may be executed unexpectedly. Many different scripting tags that can be embedded in this way include <SCRIPT>, <OBJECT>, <APPLET>, and <EMBED>.

– Malicious code sent inadvertently by a client: When a client sends malicious data intended to be used only by itself. This occurs when the client relies on an untrustworthy source of information when submitting a request. To explain this case, CERT gives the following example. An attacker may construct a malicious link such as [3]:
<A *HREF* = "*http://example.com/comment.cgi?*
mycomment = <SCRIPT>malicious code</SCRIPT>">
Click here </A>
When an unsuspecting user clicks on this link, the URL sent to example.com includes the malicious code. If the Web server sends a page back to the user including the value of *mycomment*, the malicious code may be executed unexpectedly on the client.

All these security threats point at one security problem with scripts: they all let in unsecured data.

6.5.1.1 Server-Side Script Security

A server-side script, whether compiled or interpreted, and its interpreter are included in a Web server as a module or executed as a separate CGI binary. It can access files, execute commands, and open network connections on the server. These capabilities make server-side scripts a security threat because they make anything run on the Web server unsecure by default. PHP is no exception to this problem; it is just like Perl and C. For example, PHP, like other server-side scripts, was designed to allow user-level access to the file system, but it is entirely possible that a PHP script can allow a user to read system files such as /etc/passwd which gives the user access to all passwords and the ability to modify network connections and change device entries in /dev/ or COM1, configuration files /etc/ files, and .ini files.

Since databases have become an integral part of daily computing in a networked world and large databases are stored on network servers, they become easy prey to hostile code. To retrieve or to store any information in a database, nowadays you need to connect to the database, send a legitimate query, fetch the result, and close the connection all using a query language, the Structured Query Language (SQL). An attacker can create or alter SQL commands to inject hostile data and code, or to override valuable ones, or even to execute dangerous system-level commands on the database host.

6.5.2 JavaScript and VBScript Security

Recall that using all client-side scripts like JavaScript and VBScript that execute in the browser can compromise the security of the user system. These scripts create hidden frames on Web sites so that as a user navigates a Web site, the scripts running in the browser can store information from the user for short-time use, just like a cookie. The hidden frame is an area of the Web page that is invisible to the

user but remains in place for the script to use. Data stored in these hidden frames can be used by multiple Web pages during the user session or later. Also, when a user visits a Web site, the user may not be aware that there are scripts executing at the Web site. Hackers can use these loopholes to threaten the security of the user system.

There are several ways of dealing with these problems including:

- Limit browser functions and operations of the browser scripts so that the script, for example, cannot write on or read from the user's disk.
- Make it difficult for others to read the scripts.
- Put the script in an external file and reference the file only from the document that uses it.

6.5.3 Web Script Security

Our discussion of script security issues above has centered on CGI scripts stored and executed on the server. However, as the automation of the Web goes into overdrive, there are now thousands of Web scripts doing a variety of Web services from form filling to information gathering. Most of these scripts either transient or reside on Web servers. Because of their popularity and widespread use, most client and server Web browsers today have the capability to interpret scripts embedded in Web pages downloaded from a Web server. Such scripts may be written in a variety of scripting languages. In addition, most browsers are installed with the capability to run scripts enabled by default.

6.6 Dealing with the Script Security Problems

The love of Web automation is not likely to change soon, and the future of a dynamic Web is here to stay. In addition, more and more programs written for the Web are interacting with networked clients and servers, raising the fear of a possibility that clients and servers may be attacked by these programs using embedded scripts to gain unauthorized access.

It is, therefore, necessary to be aware of the following:

- Script command line statements: Scripting languages such as PERL, PHP, and the Bourne shell pass information needed to perform tasks through command line statements which are then executed by an interpreter. This can be very dangerous.
- Clients may use special characters in input strings to confuse other clients, servers, or scripts.
- Problems with server side include user-created documents in NCSA HTTPd that provide simple information, such as current date, the file's last modification date, and the size or last modification of other files, to clients on the fly. Sometimes this information can provide a powerful interface to CGI. In an unfortunate

situation, server-side scripts are a security risk because they let clients execute dangerous commands on the server.

We summarize the three concerns above in two good solutions: one is to use only the data from a CGI, only if it will not harm the system; and the second is to check all data into or out of the script to make sure that it is safe.

Exercises

1. How did CGI revolutionize Web programming?
2. What are the differences between client-side and server-side scripting? Is one better than the other?
3. In terms of security, is client-side scripting better than server-side scripting? Why or why not?
4. Suggest ways to improve script security threats.
5. Why was VBScript not very popular?
6. The biggest script security threat has always been the acceptance of untrusted data. What is the best way for scripts to accept data and preserve the trust?

Advance Exercises

1. The most common CGI function is to fill in forms; the processing script actually takes the data input by the Web surfer and sends it as e-mail to the form administrator. Discuss the different ways such a process can fall victim to an attacker.
2. CGI is also used in discussions allowing users to talk to the customer and back. CGI helps in creating an ongoing dialog between multiple clients. Discuss the security implications of dialogs like this.
3. CGI is often used to manage extensive databases. Databases store sensitive information. Discuss security measures you can use to safeguard the databases.

References

1. Sol S. Server-side scripting. http://www.wdvl.com/Authoring/Scripting/WebWare/Server/
2. Rouse A. Understanding the role of scripting in network administration. http://www.techrepublic.com/article/understand-the-role-of-scripting-in-network-administration/

Additional References

1. The World Wide Web Security FAQ. http://www.w3.org/Security/Faq/wwwsf4.html
2. Jamsa K (2002) Hacker proof: the ultimate guide to network security, 2nd edn. Thomason Delmar Learning
3. CERT® Advisory CA-2000–02 Malicious HTML Tags Embedded in Client Web Requests. http://www.cert.org/advisories/CA-2000–02.html