DRAFT

Secure Browsers?

A. L. Dos Santos, R. A. Kemmerer, F. De Paoli

Reliable Software Group
Computer Science Department
University of California
Santa Barbara, CA 93106

August 21, 1997

Abstract Today the World Wide Web is considered to be a platform for building distributed applications. This evolution is made possible by browsers with processing capabilities and by programming languages that allow web designers to embed real programs into HTML documents. Downloading and executing code from anywhere on the Internet brings security problems along with it. A systematic and thorough analysis of security flaws in the browsers and related technology is necessary to reach a sufficient level of confidence. This paper presents some results of ongoing research that has the final goal of developing properties for secure browsers and procedures for secure browsing. The paper describes some experimental attacks that have been carried out by exploiting features of Java, JavaScript and LiveConnect executed by Netscape Communicator 4.01a.

1 Introduction:

The growth of the Internet and the World Wide Web (WWW) during the past few years has been phenomenal. The Internet is currently serving tens of millions of people connected through millions of computers. Most every business and government institution has a web page, and the web and web browsing are fast becoming the primary source of information for people of all ages.

Languages like Java and JavaScript have been developed to embed programs into HyperText Markup Language (HTML) documents (pages). Java applets, which are designed to be downloaded from the web and run directly by the Java virtual machine within a browser, are also increasingly being included in web pages to provide more sophisticated animation and other desirable features. Downloading and executing code from anywhere on the Internet brings security problems along with it. That is, the host computer is open to a variety of attacks, ranging from attacks that simply monitor the environment to export information, to attacks that change the configuration or the behavior of the host (changing files, consuming resources), and finally to attacks that open a back door to let intruders get into the host.

Many security flaws were found in the implementation of the new technologies or in their specification. This paper describes attacks implemented exploiting security flaws of Javascript and LiveConnect, with the final goal of implementing the attacks described in [PSK97] that became unusable with Netscape Navigator 4.0 and later versions. The attacks targeted Netscape's browser because it is one of the most used browsers, although we intend to test our attacks and verify the presence of other flaws in Microsoft Explorer in ongoing work.

The paper is structured in the following way, the next section summarizes the features of Java, Javascript, and LiveConnect that were used in the attack, Section 3 describes the new attacks, and finally, conclusions and future work are presented.

2 Technologies

This section outlines the characteristics of Java and JavaScript, which are the most popular languages for programming the web, and of LiveConnect, which is a technology designed to enable communication beween programs written in these two languages. We refer the reader to our previous paper [PSK97] for details on web browsers and applets.

2.1 Java

The Java language is a general-purpose object-oriented language that was introduced by Sun Microsystems in 1995 [GJS96a]. One of the major design goals for Java was portability. The result is that not only the Java source code, but also the binary code is executable on all processors. This is accomplished by compiling the source code into platform independent bytecode, which is then run by the Java virtual machine.

Some of the features of the Java language that make it simpler and supposedly more secure are that it is strongly typed, there are no preprocessor statements (like C's #define and #include), there are no pointers, no global variables, and no global functions. By keeping the language simple and without many of the error prone features, such as multiple inheritance, it is expected that Java will be more secure.

A Java program is a collection of classes and instances of classes. Each class is compiled into an intermediate format, called *bytecode*, which is then interpreted to execute the program. A major characteristic of Java is that pointers are not supported; *object references* are provided instead. Java also supports dynamic creation of instances and bindings. When a class instance (an object) is needed, it is created explicitly and a reference to it is returned; when a method is invoked on an object, the interpreter selects the method to be executed according to the class hierarchy and method overloading. Object destruction is automatically handled by a *garbage collector*, so that memory management is completely in the control of the interpreter.

The *Java Virtual Machine* is emulated in software and can run on numerous platforms [LY96]. It could also be compiled or implemented directly in microcode or hardware, but currently it is mostly emulated in software. The virtual machine deals with class files, which contain Java virtual machine instructions, a symbol table, and a few other necessary items. Java virtual machine instructions are all one byte long, and that is why they are called bytecodes. Bytecode can also be generated from other high level languages, such as Ada or C, or it could be generated manually.

Besides supporting the design of complete applications, Java supports the implementation of small applications, called *applet*s, which can be embedded into HTML pages. A special tag in a downloaded HTML file tells the web server to download the bytecode for a Java applet, and the most popular web browsers, such as Netscape Navigator and Microsoft Explorer, include support for the execution of applets. Applets can be used for implementing small stand-alone applications, as well as for implementing clients of client/server applications.

A page may have more than one applet embedded in it. In this case, all of the applets are downloaded together. Because a thread is associated with each applet more than one applet can run concurrently within the same web-page context. The class *Applet* defines a set of methods to control applet behavior. These methods can be overridden by programmers to get the desired behavior from their applets. The most important methods are *init*, *start* and *stop*. When an applet is downloaded it is automatically created, initiated and started: the methods *init* and *start* are invoked and the applet is labeled *active*. When the page that embeds an applet is not displayed (the browser is iconified or shows another page), the execution of that applet is suspended (i.e., the method *stop* is invoked and the applet is labeled *inactive*). If the page is displayed again, the applet execution is resumed (i.e., the method *start* is invoked again and the applet is labeled *active*). It is possible to keep an applet running even when it is inactive, however. This is accomplished by overriding the *stop* method so that it never returns.

2.2 Javascript

Javascript is an object-based, loosely-typed, scripting language that has been specifically designed by Netscape to include programs (scripts) into HTML pages. It gives the programmer a higher degree of access to the browser than Java does. In Javascript functions can be declared that are not methods of a particular object and that can be used with any reference to an object.

Javascript was designed to provide Internet developers with an environment that is easy to program, and it results in programs that run on both the client side, embedded in the HTML language, and on the server side, where it is called LiveWire. On the server side, LiveWire is used as common gateway interfaces (CGI) are. That is, they are accessed by a URL and execute a program in the server with access restrictions specified on the web server. The requirements of being flexible and providing a friendly environment for the programmer make Javascript subject to many security flaws as is demonstrated in [DIGI97].

Javascript provides the programmer with many pre-defined objects, such as history, text, windows. Many of these refer to browser properties, such as location, history, navigator, and window, or to the HTML page being shown in the browser. This is where the language provides the programmer with a good deal of manipulation of the user's browsers, which results in very pleasant pages and sometimes unsafe and perhaps annoying pages.

One particularly dangerous, although very useful, object is the form object. The form is sent to a CGI or LiveWire in the server URL taken from its ACTION field, and this program may send back an answer in Multipurpose Internet Mail Extensions (MIME) format that will be shown in the window or frame taken from the TARGET field. Another way to send a form is through e-mail, and for this one just specifies in the ACTION field "mailto:" with a valid e-mail address. This is a very flexible and dangerous feature.

The security policy of Javascript, implemented in Netscape Navigator, prevents some of its builtin objects and methods from accessing privileged information. In this way, although Javascript has a history object, an access to its fields returns the null string. In Navigator 3.0 Javascript introduced data tainting in order to allow applets that need this information access to it. Data tainting enables Javascript script to access this information but not to send it to a server without the consent of the user. The only requirement for data tainting to be enabled is for the user to set an environment variable called NS_ENABLE_TAINT [NET97b]. Netscape abandoned this technology in Navigator 4.0 and above, which is part of Netscape Communicator, in favor of a much better technology that uses digital signatures to give capabilities to Javascript scripts [NET97c].

Javascript enables a script to open a new browser and gives back a handle to this browser. Using this handle one should be able to access any method or object of the new browser that does not access or return privileged information. In particular, the URL object and the location object should not return any information about what the user is viewing in the other browser. When the object URL is accessed it always returns a null string. However we found that by accessing the object location using the expression *val = "" + win.location*, the URL of the page that is shown in the popped up browser is returned. Since the URL that is returned is the one that was specified by the script in the open method, this is not very useful information. If the user changes the page displayed in the popped up browser, then the expression *val = "" + win.location* returns the literal string "[object Location]". Thus, by repeatedly making these accesses the script that opened the browser could determine when the user finishes viewing the first page. This, however, does not appear to give any useful information to the script. Therefore, this implementation, although strange, at first appears to be safe. We found, however, that if the popped up browser is closed and the script evaluates the expression *val = "" + win.location,* then the URL of the page that was last displayed in the browser is returned. We describe two attacks in this paper that can be implemented using the value acquired exploiting the above flaw.

2.3 LiveConnect

LiveConnect is a technology developed by Netscape and incorporated in its browser. It was developed as an answer to page developers who wanted to be able to do inter application communication at the client side [LEV96]. It enables communication between any two of Java applets, Javascript and Plug-ins [NET97a].

The communications that may occur are:

a) A Javascript script may call methods from Java applets or Java builtin methods;
b) A Javascript script may call Plug-in functions;
c) A Java applet may call Javascript functions, both builtin and user defined;
d) A Java applet may call Plug-in functions;
e) A Plug-in may define Java classes;
f) A Plug-in may call Java and Javascript functions.

The restriction imposed on this communication is that any communication must be between an applet and a script contained in the same page and Plug-ins that were loaded by this page. Also, since one of the Java, Javascript, and Plug-in can access builtin functions of the other, they should not be able to subvert the security feature in place at the accessed technology.

This technology gives an Internet developer many opportunities for interaction with the user that were not available before. Applets can work together with Javascript and Plug-ins, giving the user all the features available in all three technologies. Also they can open channels of communication in the client side without the need for interaction with the server. This was a feature requested by many Web page developers.

The consequence of LiveConnect adding all these additional features is that they open new doors to security flaws. Concerns that were nonexistent before, like a Javascript script being able to open a socket connection [GAR97], are real threats now, since the script can communicate with a Java applet and ask it to open the connection on the script's behalf. This connection could be used to send the server information that previously was only able to leave the client as a form submission.

3 Experiments

In this section we describe the experiments carried out by the Reliable Software group at UCSB. Our original goal when these experiments began was to test the reliability of LiveConnect and check for implementation flaws. Although almost all attacks exploit the weakness of tying together different technologies, which were designed independently, the last attack described could be carried out by just exploiting a flaw in Javascript.

The first attack described was implemented against Netscape Communicator 4.01 and does not work against the newly released Netscape Communicator 4.01a. It exploits an implementation flaw in LiveConnect, which gathers information about the activities of a user browsing the Web. This information, the URL of pages the user has viewed, could be used to implement the attacks described in [PSK97].

The second attack, which works against Netscape Communicator 4.01a, is a denial of service attack. It exploits a non-documented feature of the objects used by LiveConnect. It disables part of LiveConnect, not allowing communication between Java applets and Javascript, after a user has viewed a honey-pot page at the attacker's server.

The third and fourth attacks are new implementations of [PSK97]. After [PSK97] was written Netscape released Communicator 4.0, which fixed the flaw that enabled one applet in a page to get thread names in other pages. This makes the attacks described in [PSK97] only effective with Navigator version 3.01 or lower. However, by exploiting a flaw in Javascript and using the ability for communication between Java applets and Javascript we were able to implement the same attacks. We also got the added bonus of not being restricted by pages that carry applets.

3.1 LiveConnect attacks

As with all commercial products, which need to meet tight deadlines, LiveConnect had implementation flaws that could be exploited. One powerful attack, which exploits LiveConnect to enable an attacker to track user activities, gets information entered in forms and information on the cookie file, has not yet been fixed by Netscape (August 14th, 1997). It is called the Singapore Privacy Bug, and Netscape claims they have already identified and fixed Netscape to deal with it. In [NET97d] they claim that the new version of Netscape, which patches the flaw will be released in a few weeks, but this has not yet happened.

Another attack, which was implemented by our group at UCSB, exploits the fact that the implementation of LiveConnect in Navigator 4.01 goes against the need-to-know security principle. An applet communicates with Javascript through LiveConnect using an object called JSObject. If an applet has its stop method overwritten, as was done in [PSK97], it will continue to run even after the user is no longer displaying the page that loaded the applet. If it calls a method of JSObject after the page that loaded the applet is not shown anymore, the method throws a JSException exception. The toString method of this JSException is translated to the string: "netscape.javascript.JSException: access disallowed from scripts at http://www.cs.ucsb.edu/~andre/livecon/ to documents at *the URL address*", with *the URL address* filled in with the URL address of the page being shown. A side effect of LiveConnect throwing this exception is that it kills the applet. This occurs because the prototypes of the JSObject methods do not specify that they throw any exceptions.

An attack implemented using the flaw described above, could get the URL of a page the user was viewing. It is not possible to do anything with this information, however, since the applet is killed before being able to use it. In our attack, the solution for this problem was to hand-edit the JSObject.class class library. The JSObject methods are native methods, and they can therefore throw exceptions without specifying this in their prototypes. It is an easy task to change the JSObject.class class library to specify that the methods throw JSException exception, because the specification of a class file is publicly available [LY96]. With this change, our Java applet is able to catch the exception, which stops it from being killed. Using this implementation we were able to implement attacks similar to the ones described in [PSK97], without the restriction that the target pages needed to have applets. Netscape released Navigator 4.01a, which fixed this problem printing "another domain" in the place of *the URL address.*

3.2 LiveConnnect and a denial of service attack

Denial of service attacks were never considered high priority by Sun [DFW96], and this is likely true for Netscape too. This is because the prevention of denial of service attacks is difficult, but not impossible [GAR97]. We implemented a denial of service attack that can have effects that are less obvious or harder to deal with than the ones reported in [DFW96] and [GAR97].

The denial of service attack disables the ability of an applet to communicate with Javascript using LiveConnect. This attack is very difficult to detect or deal with because of its stealth nature. It does not affect any other part of the browser other than the Java-Javascript communication. For a user, the result of this attack would be that some applets do not behave properly. The user, not having any additional clues, would likely blame Netscape for having a buggy browser. The number of such applets nowadays are small, but as the number of Java applets using LiveConnect increases the problem would scale accordingly. Using tools like Jamba [AIM97], which creates these applets or by directly using the LiveConnect technology will also increase the number of these applets in use.

The first step of this attack is to have an applet that overwrites the stop method and that uses the JSObject.class class library prepared as described in the previous section. In our implementation the applet has an infinite loop with a call to the sleep method and a call to a JSObject method (we used the eval method) in its stop method. The applet will be prevented from stopping because it had its stop method overwritten and the exception will be caught by the applet, preventing it from being killed. The effect is that due to an implementation bug Netscape will believe that any other applet that calls a JSObject method, when this malicious applet is running, is the offending applet. This other applet will receive the same JSException that the offending applet receives, and because it has not prepared JSObject.class it will not catch the exception and will be killed.

After the applet is ready, we could embed it in a honey-pot page that will be viewed by users. The applet and the honey-pot page would be difficult for a user with no experience in programming applets to trace, because the bug would be hidden until an applet that uses LiveConnect tries to run. We tested this denial of service attack using two different servers at the same domain. Although there is a possibility that the attack only works for Web servers in the same domain we don't believe that this is true. We were also careful to use IP numbers to address the Web severs running in different hosts.

3.3 Building a dossier of the user

This attack exploits the flaw in Javascript described in section 2.2. It uses this flaw to identify the URL a user is viewing in a browser that was popped up by a script. This information can then be sent to a Java applet, which is allowed to have a socket connection to the host it came from. Through this connection the information can be sent to the attacker's host.

This attack was implemented by tracking activities by the host, not by users. Some alternatives for tracking activities by the user are:

a) Use cookies which uniquely identify a user, as is done by DoubleClick Corporation [STE97]. Although this would not give the user's login name it has the advantages of using cookies, which the user is already used to accepting, and being persistent among Web surfing sessions.
b) Use the automatic submission of forms, as is done in [PSK97]. This would give the e-mail address of the user but must rely on the user accepting, or not understanding and accepting, the submission of a form by e-mail. This assumes the user has set the option in Netscape communicator 4.01a to be warned by this. If the user did not set the option or he/she is just too used to clicking yes for every warning message, then this is the best option.
c) Use social engineering to convince the user to submit a form or to have a username/password to access the attacker's page. Depending on what information is available on the attacker's server this can easily be achieved.

The attack starts with the user accessing a honey-pot page on the attacker's server. This page has a Javascript script and a Java applet. The script, when loaded, opens a new browser with a page from the attacker's server. The script also has a function that when called closes the browser that was opened, gets the location that was opened on the browser, and reopens it at that location, returning the location.

Next the Java applet opens a socket connection to the attacker's server. It then gets the IP address of the user's host and sends it through the connection. At a set interval of time the applet uses LiveConnect to call the Javascript function and gets back the URL name of the page that the user was looking at. The applet sends this information to the attacker's server through the connection.

In this attack, the interval used to collect the data is very important. If it is too small the user will notice a frequent flicker of his/her browser. If it is too long the URL's for some pages will not be collected.

This attack is likely to succeed because many PC users are used to having their browsers full screen. In Windows98 beta there is an easy way to open a Web browser, which is opened by default full screen. The browser being full screen has the advantage of not alerting the user about having two browsers opened. Another advantage of having full screen browsers is the result of the way Javascript scripts runs in the browser. Because the script is interpreted in browse space there is no repainting of the page in the browser until the script finishes. Given this property, if a loop is used for the delay in the script, in the interval of time that the second browser is closed the page on the first browser will not be repainted. Therefore, the first browser will not display any information that could lead the user to perceive the attack.

Even if the browser is not full screen the attack is likely to succeed because with a correct interval of time between samples, the user may believe that this flicker is not something to be concerned about. Also there are many pages on the Web that open a second browser.

3.4 Stealing privileged information, accounts and passwords

This attack exploits the same flaw on Javascript as the previous attack. It uses the information collected in the same way as the previous attack to identify a target page. However, when the target page is identified it opens a copy of the page, which is located on the attacker's server instead of the page located on the real server.

This attack needs only a Javascript script. It uses setTimeout to recursively call the same function after a delay. This function has the same functionality as the one used in the previous attack. The difference is that when a user is viewing a target page, instead of reopening it, a copy is opened in the attacker's server.

This attack, as in the previous one, starts when the user fetches a honey-pot page from the attacker's server. This page has a Javascript script that opens a new browser and monitors the user activity. When the user is viewing the target page, or perhaps one of a set of target pages, the browser that is being controlled by the script opens a malicious copy of the page on the attacker's server. In the same manner as is done in [PSK97], from this point on the attacker's server has complete control of the user's interaction with the pages that he believes are in another server. The forms filled in and the hyperlinks followed by the user from this moment on are at the control of the attacker, who probably faked some of the links and will send the user back to the real Web world after the targeted privileged information has been acquired.

The target page could be the log-on page of an online bank or some other page that requires a user id and password. The target page could also be a page that the user believes to be safe for typing credit card information. After the information is captured the user could clean up, closing the monitoring browser, which is no longer needed.

The same comments about the need for two browsers being opened that was presented in the previous attack applies to this attack. The same technique of faking the status line done in [PSK97] may also be implemented in this attack.

The user may notice both attacks if he tries to go back in the history list, using the back button. Since the browser is being opened over and over again, all history information is only available at the interval of time that it remains undisturbed. Also any scrolled text will be reset when the intervention occurs.

4. Conclusions

In this paper a number of attacks were described. All except the first attack are still effective with the latest version of the Netscape browser (Communicator 4.01a). The first attack was presented for its pedagogical value. It shows that in an earlier version of Communicator the need-to-know principle was forgotten, resulting in a flaw that could be exploited against a user.

We also described how a denial of service attack can be used to defame a new technology or a browser. This attack differs from other attacks that exhaust resources [DFW96] in the sense that it is very specific and has consequences only to a small portion of the browser and pages displayed by the browsers.

The last two attacks presented give the same results as the attacks described in [PSK97]. While in [PSK97] the tracking of user activity would only work for pages with applets, in the case described in this paper this restriction does not apply, which makes the attacks much more powerful.

The attack against a user's privacy described in section 3.3 shows that new technologies that combine individual parts must take in consideration not only the security features of these individual parts, but also how the security features of these parts are affected when they are combined. Javascript or Java alone have serious restrictions on what they are able to accomplish, but when they are put together they can be combined to attack the user.

The lack of the need for an applet in the target page in the attack described in section 3.4 opens the possibility of attacking any page on the Web. This has much more serious consequences than the attack described in [PSK97].

In the last two attacks described the sample rate is the most important point to make the attacks effective. The sample rate must be adjusted to a value such that the casual user will not get suspicious. These rates can be even more frequent on stand alone web browsers, like web boxes and web computers, which besides having a full screen browser make flickering more acceptable.

5 Future Work

This paper is the result of our ongoing effort to determine the needs for secure browsers and secure procedures for browsing.

We are currently running experiments for determining other security flaws in the browsers and in new technologies. We are also testing these attacks against Microsoft Explorer.

There is much that can be improved to get better stealth characteristic on these attacks. We are testing some of these solutions and investigating how much more we can exploit the security flaws identified.

From this research we expect to learn how to design procedures and counter measures to enable users to use commercial browsers without the fear of having their privacy violated or even worse of having their privileged information released. The best solution we believe would be for users to be better educated. But we believe that with all these new technologies appearing at such a fast pace and with all the new user-friendly, but not security friendly, hardware that gives easy access to the World Wide Web, this will likely be impossible. To find solutions for this problem, or part of it, is our goal.

6 References

AIM97 Aimtech Corp, "Jamba The professional design tool for creating dynamic Java applets", Jamba web site, 1997, http://www.jamba.com:80.

DFW96 Drew Dean, Edward W. Felten, and Dan S. Wallach, "Java Security: From HotJava to Netscape and Beyond", Proc. Of 1996 IEEE Symposium on Security and Privacy. http://www.cs.princeton.edu/sip/pub/secure96.html

DIGI97 Digicrime site. http://www.digicrime.com.

GAR97 Simson Garfinkel and Gene Spafford, "Web security & commerce", O'Reilly & Associates, June 1997.

GJS96 James Gosling, Bill Joy, and Guy Steele, The Java Language Specification, ISBN 0-201-63451-1, The Java Series, Addison Wesley, 1996. http://www.nge.com/home/java/spec10/index.html.

LEV96 Jason Levitt, "Internetview: An Application Infrastructure", Information Week, June 1996, http://techweb.cmp.com/iw/582/82iojl.htm.

LY96 Tim Lindholm and Frank Yellin, "The Java™ Virtual Machine Specification", September 1996, http://www.javasoft.com/docs/books/vmspec/html/VMSpecTOC.doc.html.

NET97a Netscape, "LiveConnect", Netscape web site, 1997, http://home.netscape.com/comprod/products/navigator/version_3.0/building_blocks/liveconnect/index.html.

NET97b Netscape, "Using data tainting for security", Netscape web site, 1997, http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html.

NET97c Netscape, "Signed Scripts", Netscape web site, 1997, http://developer.netscape.com/library/documentation/communicator/jsguide/js1_2.htm.

NET97d Netscape, "The Singapore Privacy Bug", Netscape web site, July 25th 1997, http://home.netscape.com/info/security-doc.html.

PSK97 Flavio De Paoli, Andre L. Dos Santos and Richard A. Kemmerer, "Vulnerability of "Secure" web browsers", to appear in 20th National Information Systems Security Conference, October 1997, http://www.cs.ucsb.edu/~andre/nissc97.ps.