Applet Watch-Dog: A Monitor Controlling the Execution of Java Applets

R. Gorrieri, G. Marchetti
Department of Computer Science, University of Bologna
Mura Anteo Zamboni 7, I-40127 Bologna, Italy
Telephone: +39 51 354516 Fax: +39 51 354510
email: gorrieri@cs.unibo.it

Abstract

A monitoring application, called *Applet Watch-Dog*, is proposed to control the execution of malicious Java applets that users can (download and) execute during Web surfing. Typical security attacks that the monitor can stop are related to denial-of-service and antagonism (e.g., obscuring the screen), at the price of a modest degradation of the performance of the Web browser. The application better prevents these security attacks when integrated within a Web browser (even if for simplicity the current prototypal implementation is itself a Java applet to be executed in the same execution space of the other applets). Applet Watch-Dog seems also a necessary tool for any software development environment for Java applets.

Keywords

Web browser, Java applets, network security, denial of service, monitor

1 INTRODUCTION

Java is the most widely used programming language by Internet programmers. One of the reasons for its success relies upon portability of compiled code (called bytecode) as the basis for mobility of Java code: a program written by any programmer in the world can in principle be loaded and executed locally by any user on his machine. Of course, this poses also serious security problems, as the imported programs could damage, even inadvertently, the resources of user's machine. Indeed, one of the most disturbing phenomena that can happen when surfing the Web is to bump into a malicious applet that forces the user to kill the browser or, even worst, to reboot the system. (See, e.g., [6] for a panorama of hostile applets.)

In spite of all the efforts made by Sun Microsystems to improve Java security, bad programmers, hackers and the like can still use Java to build dangerous applications. At present, the only way to contrast this kind of applets is to turn off Java or to load trusted applets only. Neither suggestion is practical nowadays; while the first option simply asks for abstaining from using Java (certainly a drastic solution!), the second option is not available yet, as it poses a lot of problems that are to be solved in a widely accepted manner: Who can sign your applet? Is this signing free? Can you sign yourself your applet? Does the browser realize your signature? And so on.

This paper investigates about security problems in Java and proposes a possible, practical way to defeat many malicious applets that might cause denial-of-service (e.g., obscuring the screen or playing sounds). Instead of constraining Java syntax and execution capabilities (as many researchers have proposed in the last two years, e.g. [7]), we want to find a *language-independent* way to ensure security, so that our proposal can be used in future releases of Java, or even adapted for any kind of language for mobile code.*

The basic idea is to monitor Java applets during Web surfing. Applet Watch-Dog is an application that displays some windows containing information about the current status of the applets under execution and of the resources they are using. The user can decide (typically at installation time) the upper-bounds of usage of any resource. If the applet under execution exceeds one of these limits, then the Watch-Dog immediately kills it. Of course, the monitor is not completely fair w.r.t. applets: there are cases in which non-malicious applets are stopped (e.g., when an applet uses too much of the available memory) and cases when malicious applets are executed unnoticed (e.g., an applet forging new e-mail messages usually does not make any critical use of resources). The former case is not a real problem because the user can always raise the upper bound of the critically used resource to allow the completed execution of the applet. The latter case is more intriguing: even if malicious applets may pass the filter, they are not causing denial-of-service which, after all, is the only security property we intend to guarantee with this application.

For simplicity, the current prototypal implementation of Applet Watch-Dog is itself a Java applet to be executed in the same execution space of the other applets (the sandbox). Obviously, this makes it vulnerable to the attacks of other applets: it is not so difficult to build an applet killer. For this reason, our proposal is to integrate this application within existing Web browsers that are executed outside the sandbox, hence avoiding the risk of being killed by the applets it should control.

The price to pay for monitoring all the applet activities is a modest degradation of the performance of the Web browser (less than 10% of the normal execution time). This is an encouraging result: we can safely use the Watch-Dog without fear of performance loss.

As a final remark, we want to stress that Applet Watch-Dog seems also a necessary tool for any software development environment for Java applets. When executing an applet, it is often the case that the program shows unexpected (and unintended) effects (e.g., it loops forever). If the monitor is integrated in the Java Development Kit [2], then the developer has the capability of stopping/killing the applet without exiting the environment.*

---

*Of course, this is possible only because of our limited aim, i.e., to cope with denial-of-service only. If we want to guarantee secrecy, integrity or authentication, then we may need to work at the level of the language.

*The Java Development Kit already contains *appletviewer* to test applets, but it does not work in presence of several communicating applets because it can load only one applet at a time.

2 BUILDING DANGEROUS APPLETS

Malicious applets are often present on Internet and everybody can easily find one surfing the Web. In this paper we talk especially about denial-of-service and antagonism attacks: Applet Watch-Dog was intentionally built to prevent them.

- **Denial of Service** This attack, making a critical use of resources, can bring a machine to a standstill. The attack can lead the user to kill the browser or reboot the machine (sometimes this is the only way out!). Generally, hackers use some tricks to build such applets: typically, the applet's priority thread is set at MAX_PRIORITY and the *stop()* method is redefined empty so as to prevent the user from stopping the applet. The following are the most frequent malicious applets causing denial-of-service that a web-surfer can find:

  - **Filling a file system:** Silently, the applet occupies more and more memory, so that at the end the system quits for lack of memory.
  - **Using all CPU cycles:** The applet prevents any other process to be executed by disabling any other process or by keeping its priority at the highest level.
  - **Windows and windows...:** This denial-of-service attack consists of opening a lot of windows, so to prevent the user from accessing the screen. Indeed, opening thousands of windows in a short time puts a lot of events in the window manager's queue. This, practically, disables mouse and keyboard since they talk to the machine through window events. The only way out for the user is to reboot the system.
  - **Screen obscuration:** The screen becomes suddenly black and there is no way to access desktop using mouse or keyboard.

- **Antagonism** This is the most frequent threat a user can find surfing the Web and lies in a non-critical use of resources. It is not dangerous but it is very annoying and may require to restart the browser.

  - **Sounds:** An applet may play unwanted sound files.
  - **Images:** An applet may display objectable pictures on a monitor.
  - **Work for me!** This malicious applet works "in the shadow" (for example, integer numbers factoring, ciphertext decryption, etc.) and transmits the results to his creator. At first sight, this kind of applet does not seem hostile at all: it does not try to disturb the user or to critically consume machine's resources, so that the user does not realize to be under attack. Nonetheless, they could originate some illegal situations: e.g., imagine a state employee that imprudently loads an applet that uses public resources for private purposes. Unfortunately, it is difficult to detect such applets because all of their actions are normally permitted by the Java security policy. It would be helpful to see which applets are running: if one of this has not visible effect, the user can reasonably guess that this is an applet of this kind.

People surfing the Web with a Java enabled browser are subject to other classes of security threats too, such as invasion of privacy (mail forging) or system modification (e.g., of files), see [6]. These attacks are outside the scope of Applet Watch-Dog.

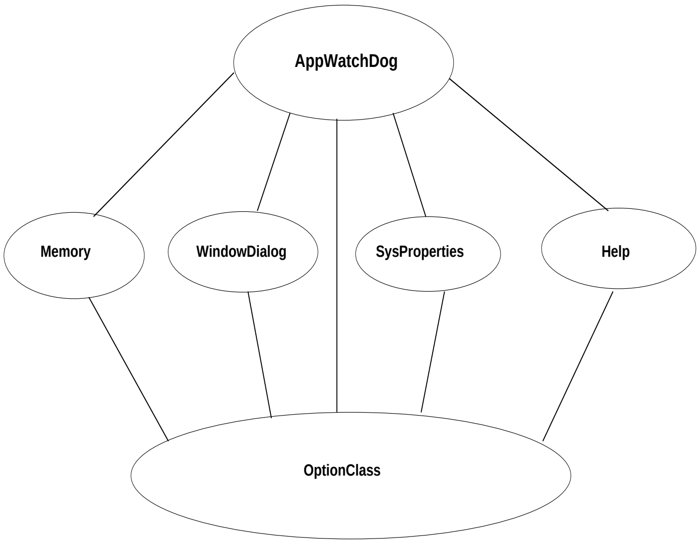

Figure 1 Structure of Applet Watch-Dog

3 APPLET WATCH-DOG

A possible way to make applets execution safer is to create a monitor to control them. This monitor must be included in the browser because a stand-alone application cannot absolutely control what happens inside a browser; furthermore the application must be configurable by the user. The prototype Applet Watch-Dog was developed under Windows95 O.S., built using the Java Development Kit (JDK) version 1.1 (released by Sun Microsystems) and KAWA editor (available at the URL `www.tek-tools.com/kawa`), and was tested on HotJava browser version 1.0β. At the time it was built (summer '97), neither Netscape Navigator nor Microsoft Explorer supported Java class files version 1.1. Program main purposes are:

1. to control all the (applet) *threads* running inside the browser ;
2. to change their status, e.g., or by changing their priority level or by killing them;
3. to be sure that the memory consumption does not exceed pre-defined levels and the screen cannot be obscured by malicious applets.

Applet Watch-Dog was not built for security purposes only: it can be also used as a development tool, to help programmers in testing and controlling their applications.

3.1 Applet Watch-Dog architecture

Figure 1 shows the structure of Applet Watch-Dog. The source code is divided into the six files listed below:

- **AppWatchDog** This is the principal class, i.e. the application kernel. It contains the *run()* method that continues its cycle until the user exits the applet. AppWatchDog class calls methods inside other classes when it needs them (e.g. setting refresh time or thread's priority value). The thread list is updated every 200 ms.* The *stop()* method is defined empty so to continue the execution even when the user changes web page.
- **Memory** This class monitors the memory consumption, showing it in a colored graphic pie (total memory, free memory and used memory). To clean up unused memory, we have inserted a call to the *gc()* method that starts the Garbage Collector.
- **WindowDialog** This class manages dialog windows for user inputs.
- **SysProperties** It shows the system properties, such as Java Classpath or Browser used.
- **Help** It contains the Help Menu showing explanations for all items in the MenuBar.
- **OptionClass** Since the principal application thread continues its execution while moving from a menu item to another one, the problem about how to communicate between classes is solved by OptionClass, that contains user choices that are available from every point inside the program.

3.2 Usage

The Applet Watch-Dog prototype can be loaded including the following HTML instruction in a Web page:
`<applet code=AppWatchDog.class width=...height=...></applet>`
or it can be loaded at browser start-up, setting in a suitable way the *start-up page*. When loaded, Watch-Dog begins to monitor all threads present in the browser, showing them in a column list where the user can see the name, the priority and the type of each thread. Each element of this list can be selected by simple click (for changing its priority or for killing it) or double click (for showing the elements of a group). The application is always active, even if the user changes the page or continues web surfing.
Applet Watch-Dog appears in a separated frame, so the user has a "console" from which he can directly see and control all the other applets running in the browser. It is something like the "Java console" under Netscape Navigator, but with different purposes: indeed the "Java console" only shows the Java interpreter execution and has no way to control applets execution.

Menus

There are four items in the MenuBar: **Watch, Actions, Options** and **Help**. Let us see them in detail.

- **Watch:** It is the principal menu and it is used to observe applets execution. It contains six submenus as follows: **Threads** (showing all the threads in the frame), **Applet threads only** (showing only the applet threads), **Memory Resources** (showing the available memory and the used memory. When the user

*We consider this value a good compromise between CPU consumption and data refresh.

chooses this item, he can then activate the Memory Usage Options (see below) to control applets consuming a quantity of memory exceeding a predetermined value. The monitored memory is that used by the browser.), **CPU Usage** (due to the impossibility of including other (machine) code or system calls in an applet, it has not been implemented yet – see Section 4 for more details), **System Properties** (showing the system properties), **Exit** (that exits the program).

- **Actions:** The Actions menu allows the user to kill a thread or change its priority. Using the **Kill Applet Threads** submenu, a user can kill the selected thread. Using the **Change Priority Settings** submenu, a user changes the thread's priority. Recall that the new value cannot be higher than that of the owner ThreadGroup, see [1].
- **Options** This menu allows the user to configure Applet Watch-Dog; because of general security policy (referred to untrusted applets, see [5]), we decided to avoid reading and writing a file, so the options are valid for one session only.

  – **CPU usage Options** Shows a window dialog with four *checkboxes* for a limit on the CPU usage. This option window is available, but the corresponding functionality is not implemented yet (see Section 4 for more details).
  – **Memory Usage Options** This option causes the halting of all applets running in user's browser (except Applet Watch-Dog, of course) when the memory consumed by the browser exceeds the fixed limit. *
  – **Priority Options** Choosing this item, the user can see a window dialog that allows him to stop every applet having a determined prioriy. For example, stop all applets having MAX_PRIORITY set.
  – **Refreshing Time** To combat the applets that obscure the screen, it is possible to refresh the main window. As a default value, the refreshing time is set at 0 minutes (no refresh), but the user can change this value as he wants. In this way the user can then manually kill the applet by selecting it from the Actions menu.

- **Help:** This menu helps the user in using the application.

3.3 What you can do and what you cannot

Applet Watch-Dog allows users to contrast malicious applets running in their browsers. In Table 1 we can see which kinds of bad applets it can stop.

An applet may be automatically killed by the Watch-Dog when, e.g., it uses of a resource more than the predetermined (by the user) amount. In this class, we can put the first applet in the table (memory allocation). Many of the other applets listed above are automatically killed if their priority level is MAX_PRIORITY, as in the case of windows creation. However, there are other cases in which the user has to intervene to manually kill the applet from the suitable menu. This is the case, for instance, of screen obscuration or of a work-for-me applet. Especially delicate is

*An even more subtle check could be to set a memory limit to each thread, rather than to the whole browser; unfortunatley, the Java language had, at the time the Watch-Dog was implemented, no ways to do that; also in this case, trusted applets could help, by allowing the use of native code or system calls.

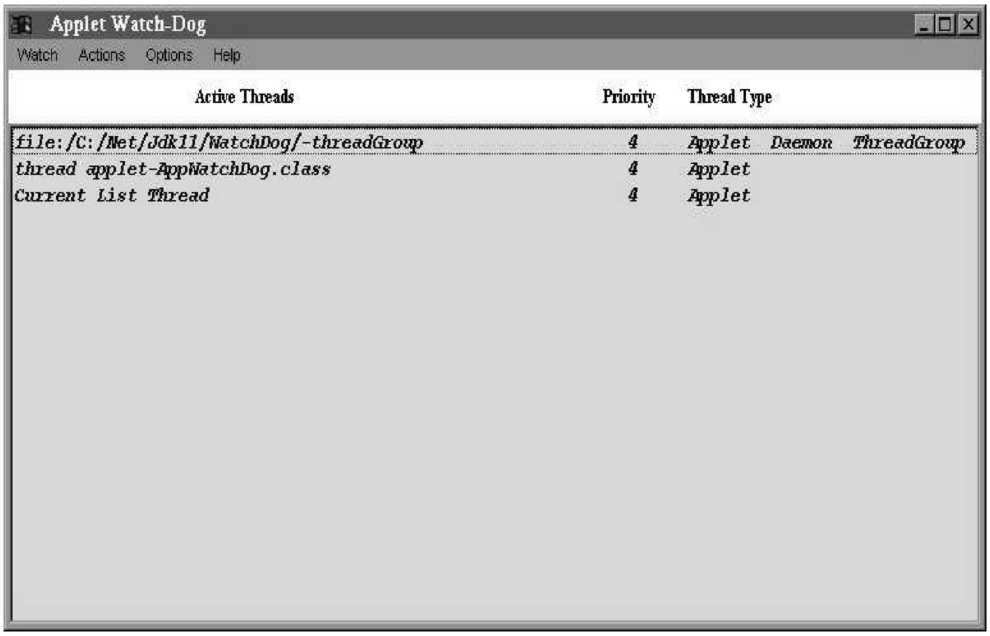

Figure 2 Applet Watch-Dog: Applet Thread Window

Malicious Applet Type	Stopped by Applet Watch-Dog
Memory allocation	YES
Windows creation	YES
Play sounds	YES
Display obscene pictures	YES
Work for me!	YES
Screen obscuration	YES
Mail forging	NO

Table 1 Malicious applet stopped by Applet Watch-Dog

the latter case, as it is not always immediate to realise when a work-for-me applet is active.
Summing up, Applet Watch-Dog stops all the applets causing denial-of-service and most of those causing antagonism. Unfortunately, it does not stop applets causing privacy attacks. One can think of introducing specific checks for coping with specific problem. Nonetheless, in general we cannot reasonably expect to be able to prevent all these problems.

3.4 Applet Watch-Dog performance

Using Symantec Norton System Monitor on a Pentium 133 Mhz with Windows95 OS, we have tested the CPU consumption when Applet Watch-Dog runs. During normal background execution, the CPU consumption increases of 3-4 %. Of course, this value is different when Applet Watch-Dog is used by the user to kill or change priority or set options, that is when it does not run in background.
This application, however, introduces a small delay in loading other applets, a delay that can be evaluated in less than 10% of total loading time. Considering the number of applets that a user normally loads during his Web surfing, Applet Watch-Dog can be used without relevant performance loss.

4 CONCLUSION

The Applet Watch-Dog is a useful tool to control every applet a user can download. Despite of the merits of the proposed application, still there are some relevant weakness points in its current implemented version, that are related to the general Java security policy (see [4], [5] or [8] for the structure of the Java Security Model). Among these weaknesses, we mention the following:

- Possible sensitivity to the browser: every browser has a different security policy implemented through the Security Manager.
- CPU Usage option: not implementaed in the current release, as Java has no way to access directly this resource.
- Save options: it would be useful that every Watch-Dog session could begin using the options saved last time.

All these problems are mainly due to the fact that, for security reasons, Java forces the execution of all the applets in the same execution space of the browser (the sandbox). Therefore, the Applet Watch-Dog is subject to the security policy of the browser (first problem), cannot use system call or import native machine code (second problem) and, finally, cannot write any permanent file because this is outside the sandbox (third problem).
A solution to these problems seems to be very close now that *trusted* applets [2] are allowed in recent releases of some browsers, like Netscape Navigator 4.0. An applet is trusted when the author of the applet is certified by some trusted company (Verisign, for Netscape 4.0), so that the author takes full responsibility for the possible damages caused by the applet execution. These applets are executed outside the sandbox, hence they are allowed to do many operations that are potentially dangerous for security and so forbidden for normal applets, like writing or reading a file, connecting to any host, executing System calls, including native code [3], etc...
We plan to implement a second release of the Applet Watch-Dog that uses trusted applets to improve its functionalities also by solving the weakness points above.
As mentioned above, Applet Watch-Dog could behave in different ways, in conformity with the browser where it has been loaded, because every browser has a different security policy implemented through the Security Manager. Moreover, the Watch-Dog is vulnerable, as it runs in the same space (the sandbox) where all the other applets run; indeed, it is not so diffuclt to design an applet killer.

The solution suggested above is to make trusted the Watch-Dog. In this way, the Watch-Dog is executed outside the sandbox and so untrusted applets cannot kill it. Another, maybe even safer, way to solve the problem is to integrate Applet Watch-Dog into the browser. This solution is even safer than the previous one as it is subject not even to potential attacks of trusted applets. In this way, the browser's functionality would be enhanced with the features of the Applet Watch-Dog.

5 ACKNOWLEDGEMENTS

The second author would like to thank Andreas Sterbenz (Graz University Of Technology) for his help and suggestions during his master thesis work.

REFERENCES

[1] M. Campione and K. Walrath. The java tutorial, September 1996. Http://ftp.javasoft.com/docs/tutorial.ps.tar.Z.

[2] D. Flanagan. *Java in a Nutshell.* O'Reilly & Associates Inc., second edition, 1997.

[3] J. Gosling and K. Arnold. *The Java Programming Language.* Addison-Wesley, 1996.

[4] Sun Microsystems Inc. Javasoft forum 1.1, September 1996. Http://java.sun.com/forum/securityForum.html.

[5] Sun Microsystems Inc. Frequently asked questions - applet security. Technical report, Sun Microsystems, February 1997. Http://www.javasoft.com/sfaq.

[6] G. McGraw and E. W. Felten. *Java Security. Hostile Applets, Holes, and Antidotes.* Wiley Computer Publishing, 1997.

[7] T. Riechmann. Secure java: A security model based on transitive capabilities for distribueted java. Technical report, University of Erlangen-Nurnberg, February 1997.

[8] A. Sterbenz. An evaluation of the java security model. Technical report, Graz University Of Technology, Austria, March 1996. Http://www.iaik.tu-graz.ac.at/PEOPLE/sterbenz.html.

6 BIOGRAPHIES

R. Gorrieri (Ms '86, PhD '91 from Pisa University) is associate professor of computer science since 1992. He is member of the executive board of EATCS and secretary of IFIP TC1. His current research interests include foundations of computer security and theory of concurrent, distributed computation.

G. Marchetti graduated from the University of Bologna in 1997. He is currently with Sinergia, a software house in Imola, Italy.