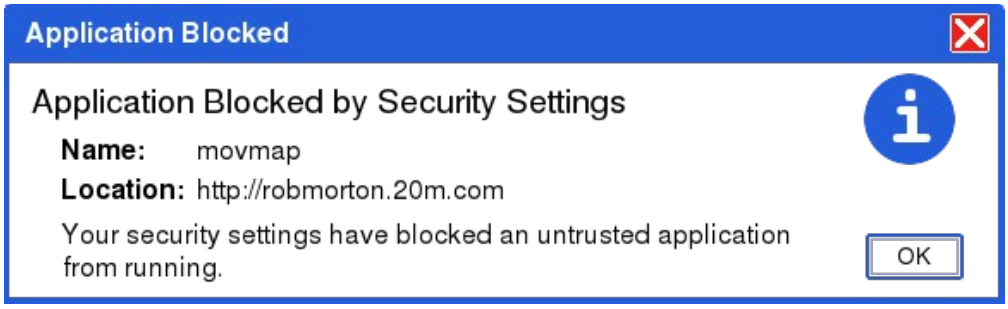
Those Ubiquitous Security Warnings



If you are attempting to view my website using Microsoft Internet Explorer, you may well sooner or later be faced with a warning message like the one shown above. It is an absolutely unfounded warning. There is nothing untrustworthy about any of the vast number of Java applets on my website. Most of them have been working perfectly for over 10 years, with some dating back as far as 1998.

Absolutely none of the Java applets on this website contains any functionality that is even remotely capable of accessing files on your computer. Neither do they pose any other kind of security threat to you or your computer. Notwithstanding, the level of so-called "security" built in to Microsoft Internet Explorer has, over the past few years, become evermore ridiculously paranoid.

With the version of the Oracle Java plug-in current at the time of writing (June 2014), I have not been able to get Microsoft Internet Explorer to run any of the vast number of applets on this website, under any circumstances. This means that Java applets that have been painstakingly designed to illustrate or animate scientific, mathematical and engineering phenomena and principles can no longer be seen. I expect this applies to thousands of other websites all over the world with intellectual and academic content.

I have been programming computers for coming up to 50 years (in September 2014). I do not have a lot of time left but I still have a lot more to do. Consequently I do not propose to waste a substantial part of that time on the gargantuan on-going task of continually revamping all my applets to keep up to date with Microsoft's platform of shifting sand.

In any case, it is not just my time that would be involved because, having revamped an applet, I would then need to get it verified and approved by some kind of self-appointed "authority" in order for it to be granted a digital certificate to enable it to run in the Microsoft Internet Explorer. This would entail having programmers combing through the thousands of lines of my code to verify that it could not be "harmful". I shudder to think how much that would cost but I know it is way beyond anything I could remotely afford.

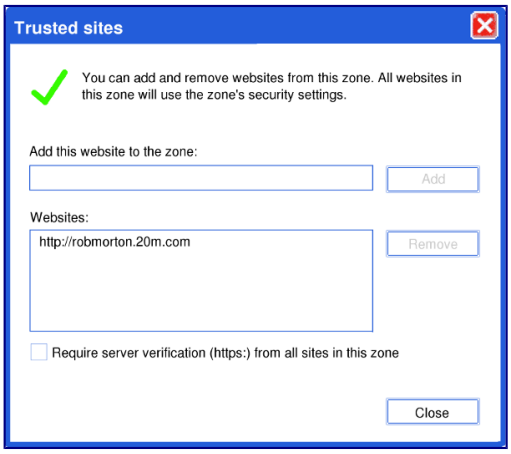
The upshot is that, if you really would like to view my applets in operation, you will need to switch to a secure operating system and browser in which, due to appropriate design, these security issues simply do not arise. An example is the *very easy to install and use* Ubuntu Linux in which I prefer to use the Epiphany web browser. There is, however, a plethora of other browsers of equal merit.

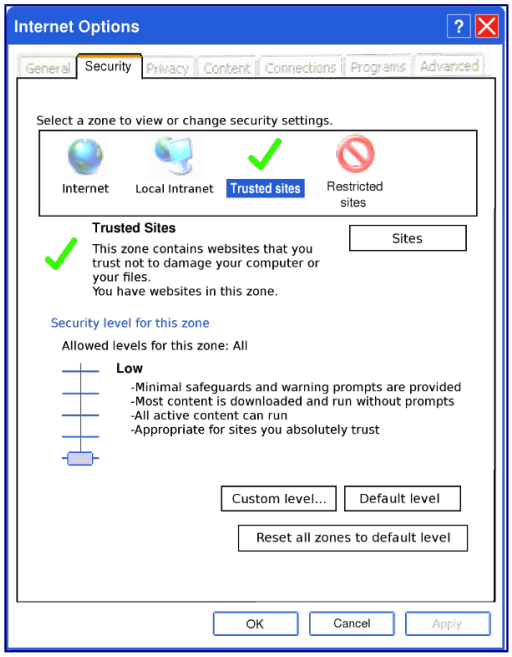
My Fruitless Attempt with IE

I do not generally use the Microsoft Windows operating system. I use Ubuntu Linux for all my day-to-day work. In early June 2014, however, I thought I should check to see how certain new web pages on my website appeared in the current version of Microsoft's Internet Explorer. I therefore fired up my Dell laptop, which still has Microsoft Windows in a spare 20GB partition on its hard drive. I quickly discovered that none of my long-established applets would run any more. Of course, they all still run perfectly well on Ubuntu. I wanted to discover why they would not run any more in Internet Explorer.

I saw that neither my latest applets nor my most ancient ones would run. The problem could not therefore be one of having to recompile with the latest version of the Java compiler. Java was up-to-date and enabled within Internet Explorer. I then tried enabling and permitting absolutely everything in the Advanced Settings section of the browser's Internet Options. I received dire warnings about the extreme risk I was taking operating in this mode. Yet my applets were still blocked from running by the browser's security system.

I set the browser's Internet zone to "trusted sites" and entered my own website as a trusted site. I was careful to un-tick the box, which restricts operation to sites that use the secure socket layer. I was now sure that the browser would run all the applets on my website. But it didn't. I got the same old "Application Blocked" message shown at the top of this page. I tried shutting down the browser, waiting a minute or two, and then restarting it. Still no joy. I shut down my laptop, had a break then started it up again. Still the same.

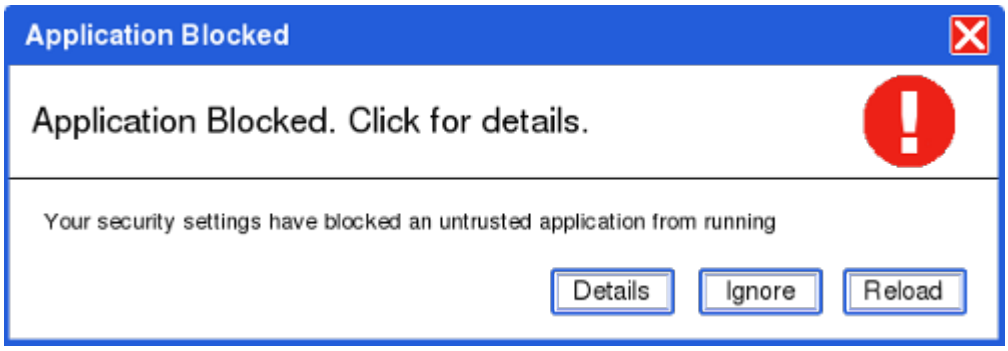




I went back into Internet Options and selected the Security tab. I moved the security slider to the very bottom, as shown on the left. This corresponds to the absolute lowest possible security level. As stated adjacent to the slider, at this level of security "Minimal safeguards and warning prompts are provided, most content is downloaded and run without prompts, *all active content can run*". It warns that this level of security is "appropriate for sites you absolutely trust". I clicked the "Apply" button and tried again to run my applet. Still nothing! In total frustration, I shut down my laptop and returned to my main computer to get on with my work. I later returned to my laptop and tried again to access my website and run an applet. Still no joy.

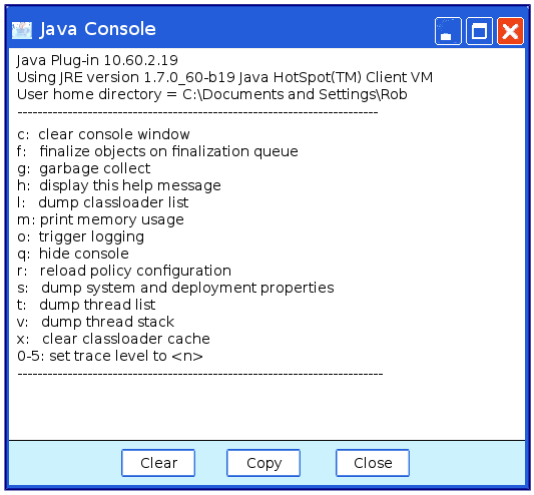
The same "Application Blocked" message was still staring me in the face, so I clicked its "OK" button. It disappeared. Beneath where it had been I saw the outline of my applet's window area with

a red message in the top right-hand corner, which says: Error: click for details. Clicking on the red message brings up the box shown below.



All advanced settings in the browser's Internet Options are set to allow everything without restriction. My website has been set to be specifically permitted to run anything as a trusted site. The security slider in the security tab is set as low as it can go. Yet **still** I am presented with a message that says "Your security settings have blocked an untrusted application from running"! What else can I possibly do? What else is there that I could possibly do? Nothing!

Clicking the "Reload" button simply reloads the page with the same result. Clicking the "Ignore" button dismisses the message box and leaves the applet inoperable. Clicking the "Details" button simply causes the Java console to be displayed, as shown on the right. Its content conveyed nothing useful to resolving the problem at hand. Internet Explorer still would not permit the applets to run, even when I loaded my web pages directly from a web server on my local area network. Nor even when I loaded them from my laptop's own localhost.



To date, I have found no way to set the security levels of the Microsoft Internet Explorer low enough to permit my applets to run with Version 7 of the Oracle Java Runtime System. Consequently, for "uncertified" Java applets, the Microsoft Internet Explorer, with Version 7 of the Oracle Java runtime system, must simply be regarded as a non-Java-capable browser. I do not know whether or not other browsers, running on the Microsoft Windows operating system, are still capable of running Java applets.

An Inadequate Solution

So, within the context of running my perfectly safe applets, Java 7 is completely dysfunctional, making Microsoft Internet Explorer effectively a non-Java browser. My only remaining option was to uninstall Java 7 and try to reinstall Java 6. This proved to be a little problematic, although I did succeed in the end. Oracle (now the proprietors of Java) give me the distinct impression that they are not at all happy about anybody uninstalling Java 7 and going back to Java 6.

With Java 6 reinstalled, my applets worked fine again and without any silly messages warning me of anything. I reset the browser's security levels back to their defaults and the operating zone back to plain Internet, with the security level slider at "Medium". I closed and re-opened the browser and

accessed my web page with my largest and most complex applet, a fully functional demonstrator of an automatic global navigator. It ran perfectly without any prompts appearing.

The great shame, of course, is that the vast majority of Microsoft Windows users will gullibly submit to the pestering pressure to update to Java 7, which will mean that safe useful illustrative Java applets, on thousands of websites all over the world, will become instantly and permanently inaccessible to them.

The ideal solution, of course, is to abandon both Microsoft Windows and Oracle Java. Switch to the Linux operating system. There you can use the Firefox browser with the Open Source "Iced-Tea" Java plug-in. Firefox will merely ask the first time only if you want to run the Iced Tea plug-in. The Epiphany browser doesn't even do that. There are many other browsers to choose from on Linux. Some of these do not even have the functionality built into them which would enable an applet to access anything outside the sandbox folder, so with them the issue of security does not even arise.

Defamatory Warning Messages

Over the past few years, the security paranoia has manifested itself in different ways in terms of browser behaviour. Currently, this page displays without any trouble because it contains no applets. Notwithstanding, this was not always the case. A few years ago, when attempting to display this very page in Microsoft Internet Explorer, the following message was displayed at the top of the browser window together with an annoying sound effect.

> *"To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options..."*

The page content was not displayed. The options offered allowed you to choose to display the page's content but strongly implied that it was entirely at the user's own risk, against Microsoft's better judgement.

This occurred on practically every one of the thousands of pages on my website. With regard to the whole of my website, this message was completely wrong in its implication and wholly inappropriate. It gave the false impression that I, as the author of this website, were a person of malicious intent.

The repetition of this message, together with its silly sound, on every single page of my website, strongly implied that the content of every one of my web pages contained code designed to access the viewer's computer. It was bound to conjure up the illusion that my web page, were it allowed to be displayed, could access the viewer's personal files and perhaps even disrupt the proper operation of his computer.

And the message was displayed with regard to my website, which clearly bears my name and contact details. It also, in certain parts, contains detailed discourses about my life and work. Though I am not qualified in law, the content of this message does seem to me like a tort of defamation against my character. It certainly branded me as somebody under expert suspicion of having malicious intent towards anybody viewing my website. And this was all wholly untrue.

Even the new 2014 version of the warning message, as shown in the box in the title frame of this page, states that: "Your security settings have blocked an untrusted application from running". This unequivocally states that the Java applet that I have presented to the viewer on my web page is *untrusted*. This effectively states that, as a web author or Java programmer, I am untrustworthy. This is certainly an expert denunciation of my technical ability, if not also a tort of defamation against my character.

At the time of writing, I have been programming for coming up to 50 years, which is a lot longer than Microsoft has existed. The fact that Microsoft states that my applets are untrusted (presumably by themselves) presupposes that their browser security software has examined the code on my web page and thereby determined it to be untrustworthy. But is this really what it's done?

My Website's "Active Content"

On this page, for instance, the so-called "active content" comprises two lines of JavaScript that *would* display, in the browser's Status Bar, the date and time that this page was last modified. It also ensures that this page is loaded correctly into the browser frames. It does absolutely nothing else. Instead of allowing the browser to display this date and time, Microsoft's security software displayed its own *protective shield* icon.

On those of my pages that contain interactive JavaScript forms or Java applets I have seen a message saying that the browser is blocking potentially harmful Active X controls. It then displayed the page without the interactive form or applet. Being thus devoid of its most essential feature, the rest of the content of the page was rendered pointless.

I know nothing about Active X. I do not use it. It exists nowhere on my web site. My pages with interactive JavaScript forms simply assist the user by filling in fields that can be calculated automatically from other information entered earlier in the page. It **does not** contain any mechanism or functionality that is capable of either accessing your computer's file system or of sending any information you enter in the forms back to me.

There are a lot of Java applets on my web site. These are all of a scientific, mathematical or engineering nature. There are no trivial animations or other frilly items of web-designer's self-indulgence. My Java applets operate in what is called a *sand box*. This is a fenced area of memory, within the browser environment, from which no program can possibly access or modify anything in your computer's file system. It simply does not have the capability. The only files my Java applets can access are those on my server. And they only ever read *from* these files: they never write *to* them. So my applets cannot be a threat to your computer or your privacy.

It appears, therefore, that the security program that produces this message is not factually capable of determining whether a script or applet is harmful or not. It simply detects the existence of a script or applet and then makes the blind assumption that if the script has not been written or approved by Microsoft, or some other self-appointed authority, then *ipso facto* it is potentially harmful to your computer. It is a matter of being presumed guilty until I can prove, to Microsoft's satisfaction, that I am innocent. To extricate myself from presumed guilt, I must bear the burden of the proof, including its excruciating cost.

Validating Active Content

In my opinion, it is not beyond the bounds of feasibility or cost for a security program to examine an "unknown" JavaScript program or a Java applet to determine whether or not it is safe to run within your browser.

JavaScript is what is known as an *interpreter* language. To execute a JavaScript program, another program within the browser (called an interpreter) reads a statement of the JavaScript source code, as written by the programmer, interprets it into executable code and then executes it there and then. The interpreter then moves on to the next statement and repeats the procedure for that one and so on to the end of the JavaScript program.

The JavaScript source code, as written by the human programmer, is thus available within the web page's HTML file, which is in the browser's temporary storage area on your computer. Consequently, it is freely available for Microsoft's security software to examine. By parsing each statement of the JavaScript source, the security software could determine whether or not that statement contained any input/output functionality. If so, it could further parse the input/output functionality to determine whether or not that functionality accessed the local file system. If so, red flag. If not, the security software has no valid reason to condemn the program as potentially harmful to your computer. Following this procedure would raise absolutely no red flags on my website.

In my opinion, it is not beyond the bounds of feasibility or cost for a security program to do this. Notwithstanding, from the observed warning messages, I must conclude that, at the time of writing, the security program does not do this. It simply casts suspicion on the web author's JavaScript program without any evidence that it could be harmful.

I think it expedient here to make a point of language. In these warning messages, words like "untrusted" or "could be harmful" are not referring to JavaScript programs or Java applets in general. They are referring to the specific JavaScript program or Java applet on the specific web page being viewed. The security program emitting the message is therefore asserting that the specific JavaScript program or Java applet concerned contains functionality that could, given the right possible set of circumstances, access your file system or harm your computer. None of my JavaScript programs or Java applets contains any such functionality in its code. Consequently, the content of the warning messages and its implications are untrue.

Although there are certain syntactical similarities in their general appearance, Java is not the same as JavaScript. Java is a full rigorous programming language. It has some inherited similarities to C. However, unlike C, it is not a fully compiled language. The Java compiler does not translate what the programmer writes into the computer's own native machine code. Instead it translates it into what is called an intermediate level code. It is kind of compiled half way at compile time and then half-way interpreted at execution time.

> There's a whole history of half-&-half compiled/interpreted languages. Their main merit is small compact executable files, which are, at the same time, reasonably fast in execution.

A Java applet, as downloaded by your browser from my website, is contained in what is called a *class* file, represented by the coffee cup symbol on the left. The class file contains my applet in the form of intermediate-level code, referred to as *bytecode*. When the applet is run within your browser, it is this bytecode that is executed.

Bytecode is by no means as easy to investigate as the original Java source code, as written by the human programmer, from which it was compiled. It is, however, possible to *decompile* bytecode back into a somewhat bland interpretation of the original Java source code. The Java source code thus produced will lack all programmer comments and the context-related names for program variables. Nonetheless, certain crucial pieces of information from the original source code can still be obtained. For example, the bytecode can reveal whether or not the programmer created a file object (for communicating with a file) or a stream object (for communicating with the website's server).

If a file object were created, then perhaps there is reason to suspect that the applet contains functionality for reading from or writing to files on your computer. If it doesn't, there should be nothing to worry about and the applet should be allowed to run without the need to raise any prompts, messages, or alerts on your screen.

If a stream object were created, then further investigation is necessary. The security software must then look to see if the programmer somewhere created a *stream reader* and/or a *stream writer*. If only a stream reader were created then the applet merely wants to read its own auxiliary data from a file on its own web server. This constitutes no threat whatsoever to your computer or any private data that you may have stored on its hard drive. If, on the other hand, the programmer has created a *stream writer* object, then the applet could well be able to send back to its server any information you enter into a form field within the applet's area of the displayed web page.

None of the Java applets on my website contains file objects or stream writers. So they are all inherently safe from your point of view. Notwithstanding, you and Microsoft may prefer to assume that I could be lying. Perhaps I am a dastardly hacker bent on wreaking havoc within your computer and its filesystem. It is for reason of such potential suspicion that I publish the entire source code of all of my applets. The source code of each applet may be displayed in full from a link appearing on the web page within which the applet runs.

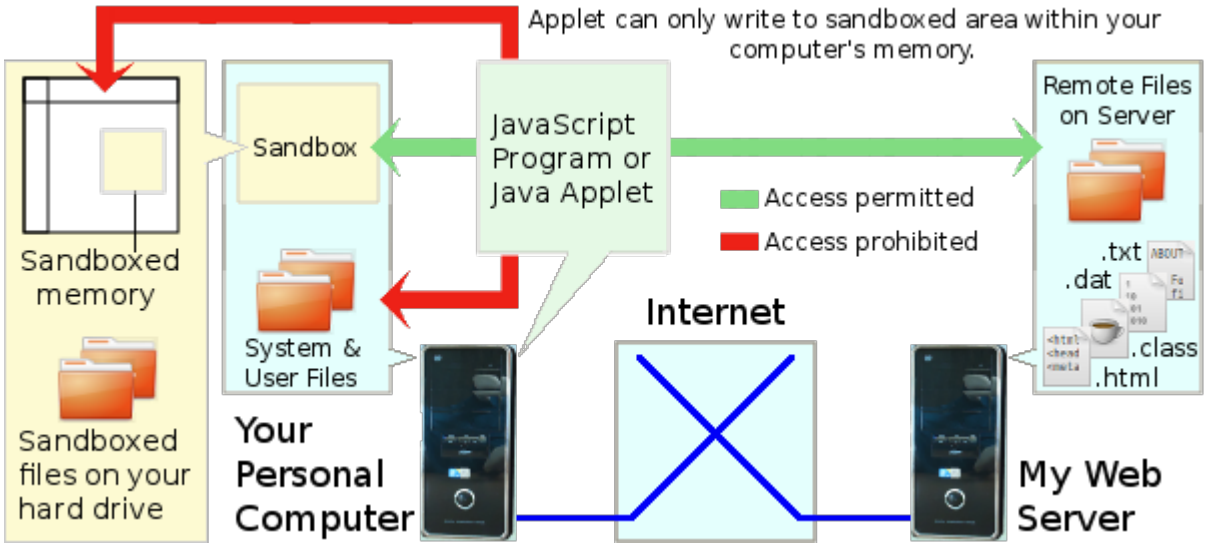
Of course, I could be engaged in some nasty form of skulduggery in which the source code I publish is not the source code from which I compiled the applet. So the source code may be benign while the applet itself is malevolent. This level of paranoia can be assuaged only by modifying the browser's Java runtime environment, within which applets run.

The Java Runtime Environment

As far as I am aware, in the beginning, Java applets had no means of accessing local files within the viewer's computer. A Java applet could only execute Java code, downloaded in .class files, from its web page's server, to create some graphical effect in its appointed area within its web page. An example of this kind of applet is the very first applet that I wrote, which merely displays an animation of Lissajous' Figures. Of course, whether or not your browser allows you to actually see it depends on the level of paranoia of your browser's security settings. This applet neither reads from, nor writes to, any files at all. It simply writes the graphical image on the screen. Yet according to Microsoft Windows and Internet Explorer with Oracle Java 7, it is considered to be far too dangerous to your computer to be allowed to run at all!

Later development enabled a Java applet to read data from other kinds of files stored on the server from which the web page, within which the applet was embedded, was served. Still no functionality existed, within the browser's Java runtime environment, whereby an applet could read from, or write to, any files on your computer's own file system. This allowed an applet to read auxiliary data from its own web server in answer to selections made by the viewer through the applet's user interface (buttons, menus and so on). It also allowed an applet to send data back to its server from data fields within the applet that were filled in by the user. None of my applets sends data from the client side (your computer) back to my server.

Thus Java applets run in what has been termed a *sandbox*. I think the term must come from a box containing sand in which young children are allowed to play. Presumably they are not allowed to play outside the sandbox. So it is with Java applets. They are only allowed to operate within the sandbox area of your computer's memory and disk space. They are not allowed to do anything outside it. The principle of the sandbox, in the context of Java applets, is illustrated below.



My web server accommodates directories (or folders) in which are stored the pages of my website. The basic content of my web pages is contained in HTML files. Illustrations, like the one above, are contained in image files such as PNG or JPG. Some of the content is contained in basic text (TXT) files. The executable program code for my Java applets is contained in what are called CLASS files. These are binary files, which are not generally readable by humans. Some of my Java applets, when running within your browser, pull the data they require, from time to time, from data (DAT) files, which are also stored on my web server.

When you click on a link, that takes you to one of my web pages, your browser downloads the appropriate HTML file from my web server. Your browser then displays my page's content within the browser window. If my page contains an applet, your browser downloads the applet's main CLASS file, from my web server, into your browser's Java runtime environment and sets it running. My applet then writes its visual presentation onto its designated area within the displayed web page.

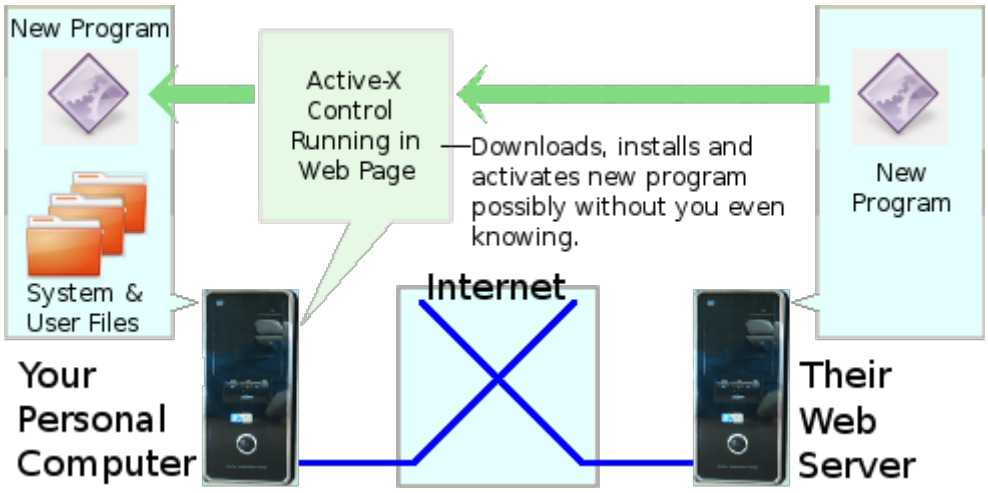
If the applet concerned is of such a type that enables you to select what it should do, then the applet itself may download a data file from my web server and put its contents into a part of your computer's memory. An example is my automatic global navigator. This offers you a selection of routes to fly. When you select a route, the applet downloads the waypoints data for that particular route so that it knows where to go. The area of your computer's memory, into which it downloads its data, is, however, part of the sandbox environment. My applet may not read from, or write to, any part of your computer's memory outside the sandboxed area.

Although none of my applets does this, it is possible for an applet, operating within the sandbox, to send data from your computer to the web server. Notwithstanding, it can only send data, which you the user enter via the applet's data entry fields, menus or tick-boxes within the applet's area of the web page. Since a sandboxed applet cannot read your local filesystem or memory outside the sandbox area, it cannot send any data from these places back to its server. Under this form of Java runtime environment, all was well. All was secure.

But They Opened Pandora's Box

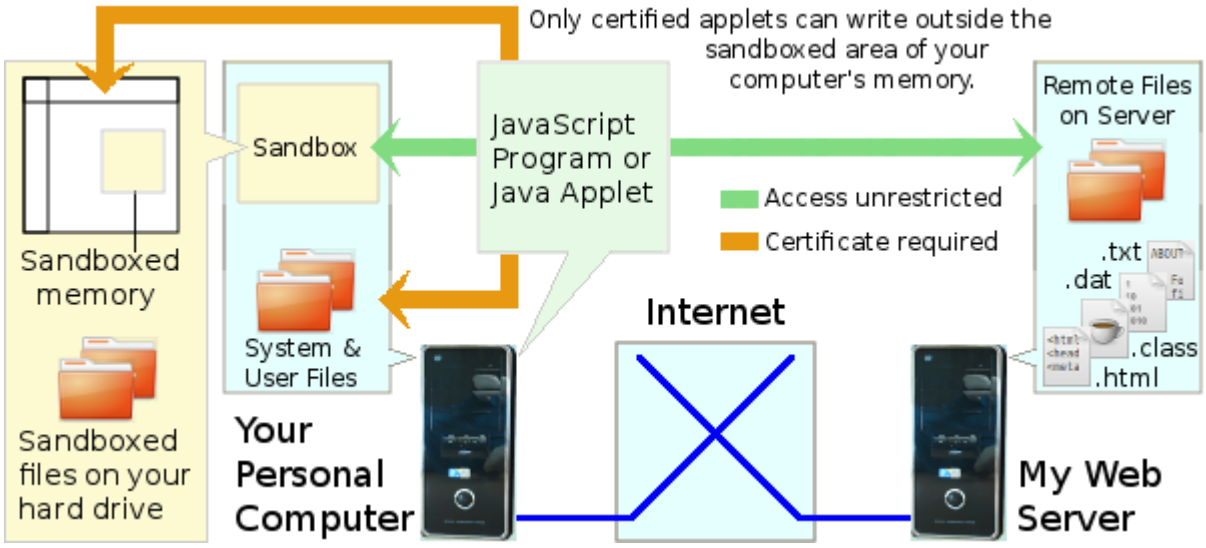
Then, from what I can only deduce to be commercial reasons, it seems to have been decided, by the dominant forces within the IT Industry, to enable applets to access the web viewer's local filesystem. They gave Java applets access to your computer's memory and disk space outside the sandbox. This gave Java applets almost the freedom that I understand to be available to Microsoft's Active X controls.

From the little I have been able to glean about Active X, it seems that a website that you are viewing can download, onto your computer's hard drive, a native program with a browser-based user interface. A native program is one that comprises executable code, compiled specifically for your type of computer and operating system, that runs directly on your computer like any other application program. And, since its operation is spawned by your browser, which you yourself opened, the Active X program has the same access privileges on your local hard drive that *you* have.



The obvious advantage of such a technology is that it enables people to use application programs seamlessly across many computers, connected via the Internet, as if they were all operating within the same computer. And I expect it was pressure from an ignorant market, rather than sound system design, that forced such a technology into being, without any notion of the serious security implications even entering the market's cretinous collective mind.

This made Java applets appear somewhat restricted. I suppose it was for this reason that *the powers that be* opened the way for Java applets to access the web-page viewer's local filesystem. I think this is where the trouble must have begun. To permit certain Java applets to access the local filesystem, the implementation of Java within a web browser needed to be changed along the lines of the model shown below.



The idea is that the mechanism whereby Java applets can read from, and write to, the local filesystem be implemented, but that in order for a particular Java applet to make use of this mechanism, the applet must carry a digital certificate to certify that it be non-malicious.

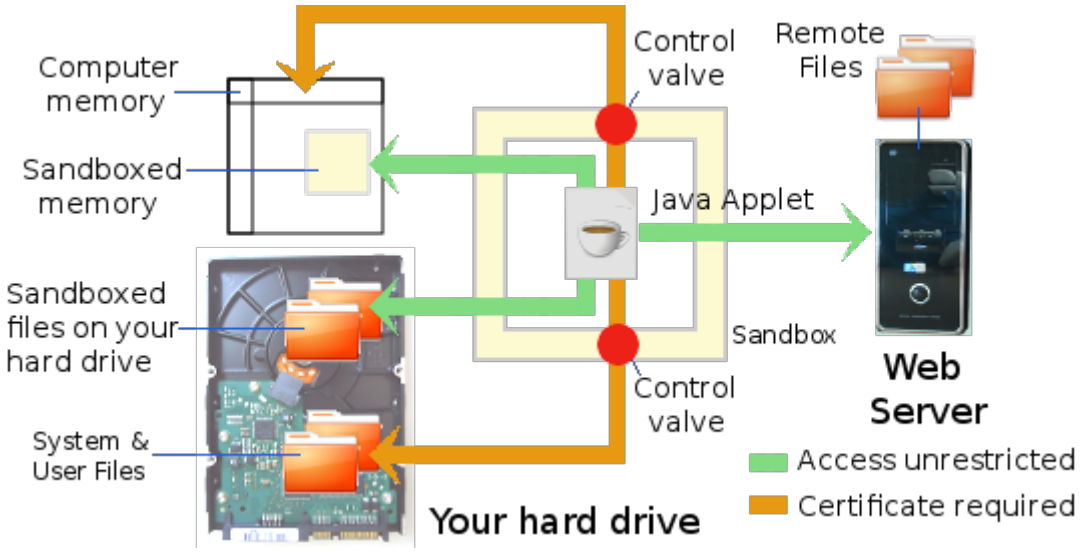
All the applets on my website are uncertified. None of them has a digital certificate issued by some self-appointed authority telling everybody that it is non-malicious. Consequently, under this new model, none of my applets would be able to access the viewer's local filesystem or areas of his computer's memory outside the Java sandbox, which is fine by me.

Large corporations, on the other hand, could have their Java applets certified as safe. Consequently, when downloaded by a browser, the browser would see the digital certificate and so permit the applet to access the viewer's local filesystem and areas of his computer's memory outside the sandbox. This facilitates the implementation of business applications that can operate seamlessly across many computers via the Internet with almost the flexibility of Active X.

Unfortunately, this kind of model requires that the walls of the sandbox be very complicated.

The Selective Sandbox

My perception of how the Java runtime sandbox would have to be implemented to permit selective access to the web-page viewer's local filesystem and computer memory is illustrated below.



The *sandboxed memory* is a part of your computer's memory. It is allocated, as and when necessary, by your browser's Java runtime system, for use by applets running within the web page you are currently viewing. An uncertified applet is prevented, by a "control valve" within your browser's Java runtime system, from addressing any memory outside its allocated area.

None of the applets on my website attempts to access any part of your computer's memory that is outside the walls of the sandbox. Nevertheless, were any applet to attempt to do so, the "control valve" within your browser's Java runtime system would block its attempt. Thus, any uncertified malicious applet that did try to access other parts of your computer's memory would not be able to succeed.

The sandboxed files on your hard drive are contained within your browser's own temporary working directory (or folder). These files may be necessary to facilitate the downloading of applets and their data. The coding within the applet itself, however, is not usually aware of them. Usually, they are downloaded, read from, or written to, by your browser's Java runtime system.

Notwithstanding, it is possible to build coding into an applet that can create, write to, read from and delete files on your computer's local filesystem (hard drive). If such coding be present within an uncertified applet, however, another "control valve" within your browser's Java runtime system will prevent it from doing so. On the other hand, if the applet be certified, your browser's Java runtime system will open the "control valve" to let it create, write to, read from and delete files on your computer's local filesystem.

The *sandbox* of your browser's Java Runtime system thus now operates in two modes or levels of security.

1. For an uncertified applet. An uncertified applet is allowed to read from, and write to, files on the server from which it was downloaded. It has no access whatsoever to the webpage viewer's local filesystem or to any of his computer's memory that lies outside its allocated sandbox area.
2. For a certified applet. A certified applet is allowed to read from, and write to, files on the server from which it was downloaded. It is also allowed access to the webpage viewer's local filesystem and to his computer's memory that lies outside its allocated sandbox area, mostly for the purpose of interacting with other programs through application program interfaces (APIs).

Theoretically, this should cater perfectly for, on the one hand, individuals and organizations with only passive demonstration applets on their websites and also, on the other hand, commerce, which requires applets providing a highly interactive interface between the web page viewer and the server.

Sandbox Shortcomings

Unfortunately, in reality, because of the sheer complexity of the programming in this two-mode version of the sandbox, the walls of the sandbox have become leaky. In other words, the two control valves do not operate absolutely correctly in every possible circumstance. Thus it has proved to be possible for malicious or faulty coding, within an uncertified applet, to access the web page viewer's local file system and computer memory outside its sandbox.

The *powers that be* unilaterally decided that this presented an unacceptable security risk and consequently modified their browser security software. As a result, ordinary web users have been unable to view my perfectly harmless applets without going, past dire security warnings, to change the "safe" default security settings within their browsers to what is declared to be an "unsafe" or "high risk" level. Of course, these false warning messages are gullibly believed by the vast majority of web users.

But apparently, even this was not enough. The *powers that be* finally decided (at the time of writing) that running uncertified applets at all was too risky. Consequently, they modified the browser security software to permit only certified applets to run. Uncertified applets were blocked. There now seems to be no way to set the browser security level low enough to allow uncertified applets to run under any circumstances.

Thus, when presented with the Application Blocked warning, at the top of this article, users viewing one of my web pages containing an applet will be led unequivocally to believe that my applet is a danger to their computer and their privacy and that, by unavoidable implication, I, the author of the page, am an Internet menace. All totally and absolutely untrue. The real menace is their implementation of the Java runtime system: not my applets that run within it.

The Jack-of-all-Trades Browser

From my almost 50 years of programming experience, I am not at all surprised that this situation has come about. It was learned long ago that it is folly to try to create a single *all singing and dancing* program to do everything, which is the case with the current popular web browsers. I seem to recollect that, at one stage, Microsoft even tried to make its operating system desktop effectively a browser or vice versa. I certainly remember various pundits in IT industry magazines suggesting that the browser would eventually be the platform within which all other applications would be run.

> It reminds me of a computer program dating from the 1980s called "The Last One". It was advertised as the last computer program that would ever have to be written. This is because you could use it to specify what you wanted a program to do and it would write the program for you [in BASIC of course]. Where is it now?

The original graphical web browser provided an ideal means of presenting illustrated documentation with facilities for including embedded pictures, diagrams and animations. The purpose of the Hypertext Markup Language (HTML), in which a web page was written, and the Hypertext Transfer Protocol (HTTP), by which its content was distributed, was to facilitate the free and open distribution of information and knowledge via the Internet. Security is exactly what it was **not** about. The Java implementation had a simple completely impervious sandbox with no leaks. The functionality for accessing any local memory or filesystem simply did not exist, and could not therefore be misused by malicious code. Thus, the original graphical web browser was never an appropriate platform for secure business or banking transactions.

Notwithstanding, the corporates of the IT Industry have augmented the original web browser, turning it into a hybrid monster that can, at the same time, be two opposites. Upon a foundation whose brief was the free and open distribution of information, the IT Industry has tried to build a private secure business transaction facility. The original browser was a flexible presentation platform, which gave the viewer the ability to adjust the size and form of the text. Up on this, the IT Industry has tried to construct a fixed "word processor" or "desktop publishing" type of presentation, which constrains documents to the confusing unalterable styles of the graphics artist, with print that is often far too small or indistinct for older people to even read.

If business and commerce want a highly interactive user interface application for the purpose of buying and selling via the Internet, then they should design and produce such an application specifically and exclusively to meet this purpose. It should be built upon its own desktop publishing presentation engine, which is unrelated to HTML. It should communicate via its own underlying secure protocol, which is unrelated to HTTP. Leave the web browser, unadulterated by commerce, to fulfil its original brief of facilitating the free and open distribution of information and knowledge.

It is all a matter of *horses for courses*. To go trekking in the mountains, you use a hill pony. To win the Derby, you use a thoroughbred race horse. I don't think that even the Irish have tried to breed a thoroughbred racing hill pony, which is precisely what the IT Industry has tried to do with the web browser.

Digital Certificates

The self-appointed guardians of people's browsing security could retort that if my applets be benign then I should simply get them all certified. They would then be instantly visible to anybody accessing my website. Notwithstanding, the practicality of my getting my applets certified is not such a simple matter.

As mentioned earlier, it is plain that the security software built into browser runtime systems is not capable of ascertaining whether or not an applet is malicious or otherwise harmful. They simply assume that all non-certified applets *could be* harmful and therefore block them. It is unlikely, therefore, that these self-appointed certifiers possess relatively cheap automatic means of verifying applets by examining either their source code or their compiled byte code.

So how could these self-appointed certifiers go about certifying my applets? The only option left, as far as I can see, is for professional programmers, employed by the certifiers, to scrutinize the thousands of lines of Java source code that make up all the applets on my website. How much would this cost? And who would pay for it? I expect that it would cost a king's ransom and that I would be the one expected to pay the full cost. Since my only income is my reduced UK State Pension of £92·49 a week, this is clearly not going to happen.

But certification would not only involve me in spending far more money than I have. It would also involve a lot of time, which I do not have either, and I would end up with applets of reduced performance.

When a large multi-CLASS applet is run in a web page, only those CLASS files required at the time are downloaded to your browser. Potentially large CLASS files, whose function is not required at the time, are not downloaded unless or until they are needed. As I understand it, the only way I can get an applet certified is to incorporate all its Java CLASS files into a single ".jar" file. This means that for you to be able to see my larger applets running within their respective web pages, your browser would have to download the entire applet before it could run. So I would have the enormous task of ".jar"-filing all my applets, which would then be slower to download and run.

Consequently, in terms of time and cost, getting my applets certified is not a practical option. I must accept that my website's applets will not be permitted to run in the Microsoft Internet Explorer web browser (or perhaps even in any web browser running under the Microsoft Windows operating system). The privilege of being able to enhance freely distributed information, knowledge and ideas by means of applets has become the exclusive reserve of the rich and corporate. The benign individual of limited means has now been well and truly locked out.

It thus appears to me that the applets on my website can no longer be viewed by people with computers running Microsoft Windows. I expect the same applies to an innumerable number of other websites throughout the world. The Java applet has been to me an enormous aid in illustrating the ideas conveyed by my texts. Its effective demise, for those still running Microsoft Windows, makes me feel as if yet another nail has been driven into the coffin of individual free-expression.

Notwithstanding, anybody running Linux using a web browser with Open Source Java plug-ins can [still] view my applets without restriction.

The Two-Browser Solution

The present integrated Java Runtime System, as described earlier, has selective control valves for allowing certified applets only to access the local memory and filesystem. On top of every other consideration, this is prone to possible malfunction because of its complexity.

The solution is to disintegrate the all-singing-and-dancing browser into two separate applications, namely: an *academic* browser and a *commercial* browser. I use the word *academic* very loosely here. I do not imply that the academic browser should be used only for material having an academic source or approval. I intend it to cover any source of information that is freely distributed on the worldwide web and may contain only *passive* applets, which have no need to access your local disk or APIs.

The *commercial* browser would run only certified applets, which would be allowed to access your local disk and APIs.

Notwithstanding, subtle design or coding errors in the Java Runtime System (especially the sandbox) could still leave your computer vulnerable, even if all applets were non-malicious and perfectly coded. Such errors could, for example, create wild address pointers. These could inadvertently cause a well coded applet to read from or write to a part of your computer's memory outside the sandbox (or API in the case of a certified applet running in the *commercial* browser).

Selective-Compiler Solution

Instead of implementing the control valves in the runtime system, they could, in effect, be placed in the compiler by means of a command-line "sandbox" switch. If I wish to compile my global navigator applet securely, I use the command:

```
javac -sandbox airmap.java
```

When the compiler sees the "sandbox" switch on the command line, it excludes from the compilation any source code that would access the local memory or filesystem of the web page viewer's computer. The compiler would then encrypt a "sandbox" compilation certificate into each resulting CLASS file.

Unfortunately, neither is this solution immune from possible malfunctions in the Java runtime system. There needs to be some way of making it physically impossible for an alien program (such as a Java applet) to access any part of your computer's resources outside its allocated sandbox memory and disk space.

The Only Remaining Options

At the time of writing, the only options left for anyone who would wish to see my applets in operation are as follows.

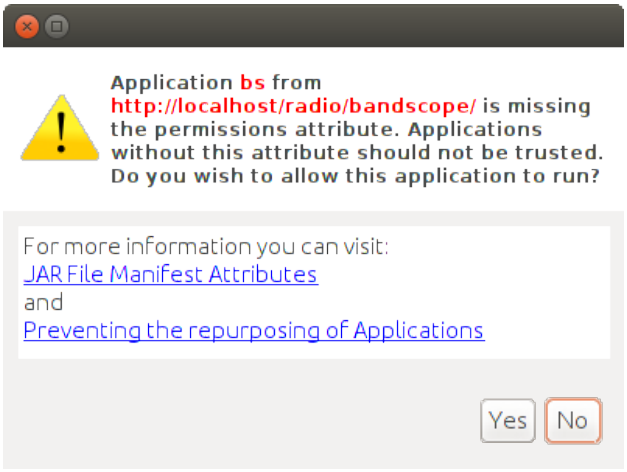
1. Revert to Java 6 and never update it.
2. Upgrade to the Linux operating system.

Under the first option, you will lose out on any possible attempts the vendors may make to correct the flaws and shortcomings in the Java runtime system. You will, of course, be continually pestered to upgrade to the latest version of Java. However, bearing in mind that this will exclude you from what undoubtedly must be the vastly greater proportion of illustrative applets on the Worldwide Web, I think there is really only one way forward.

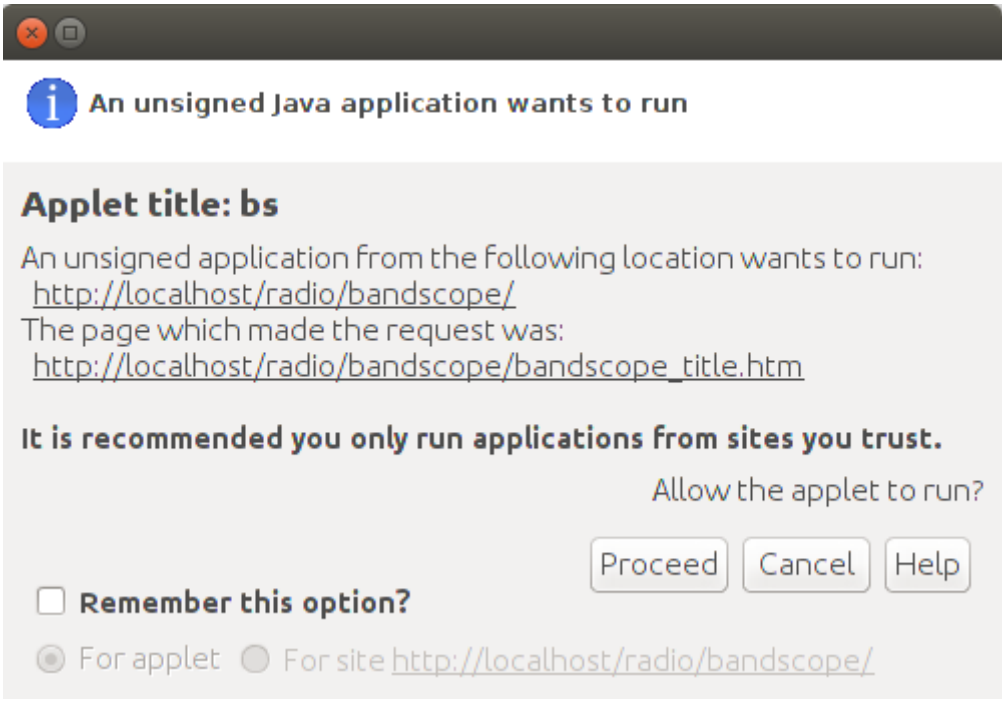
Adjusting For Linux

Traditionally, as far as I have been aware, there have been no problems viewing my applets on Linux-based systems.

Notwithstanding, the latest version of Open Source Java "Open JDK Java 7" with the latest "Iced Tea" Open Source Java browser plug-in, arrives with very paranoid default security settings. As a result, as soon as one of my web pages containing an applet is opened in the Firefox web browser in Ubuntu Linux, a chilling warning appears, as shown on the right. Nevertheless, you still have the option to say that *yes* you do want to run the applet anyway. So you can click "Yes".



But clicking "Yes" invokes a second, larger, warning message. This message positively recommends that you only run applications from web sites that you trust. And why should you trust my web site? You don't know me. You have no idea as to my motives or intentions. On the other hand, if you don't run the applet, you will have little or no idea what I am talking about within the context of the web page.



This message is determinedly trying to induce fear into you concerning the running of my applet. And entirely without qualification. The Iced Tea plug-in knows nothing about my applet. It cannot possibly know. If it did, it would have no reason to display these two rather pointed and dire warnings.

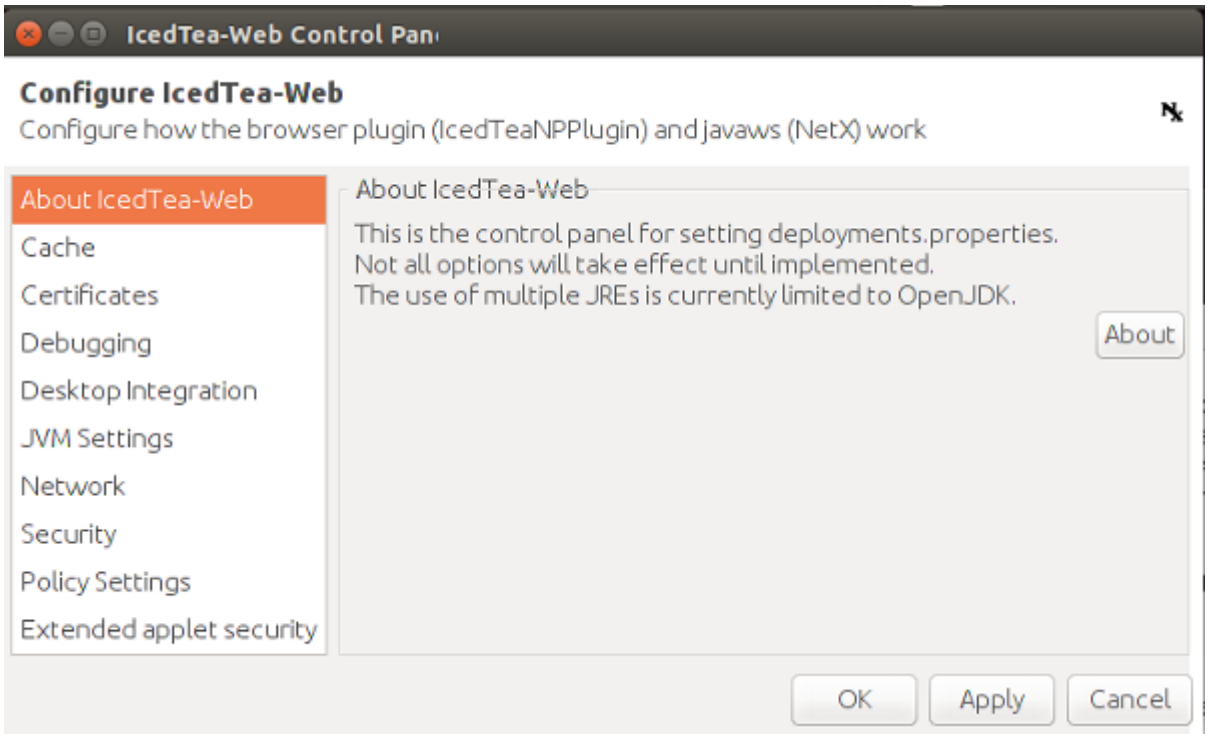
The reason for these warning messages is that the Iced Tea plug-in is set, by default, for what I call "commercial" applets, which may need the freedom to read files located on your computer's hard drive and possibly also to write data to them. They may even need to run other programs already resident within your computer. To run these kind of applets, you really do need to be able to trust implicitly the web site on which the applet is located.

Originally, however, applets were not for commercial use. They were for animating illustrations on web sites. I call these kind of applets "academic" applets. By this, I do not mean that they need to originate from a university or academic institute of any kind. I use the term merely to indicate that the originator of the applet wrote it simply as an aid for making an explanation clearer by means of animation or to facilitate interactive participation by the reader. Such an applet does not require any access to your computer's filesystem.

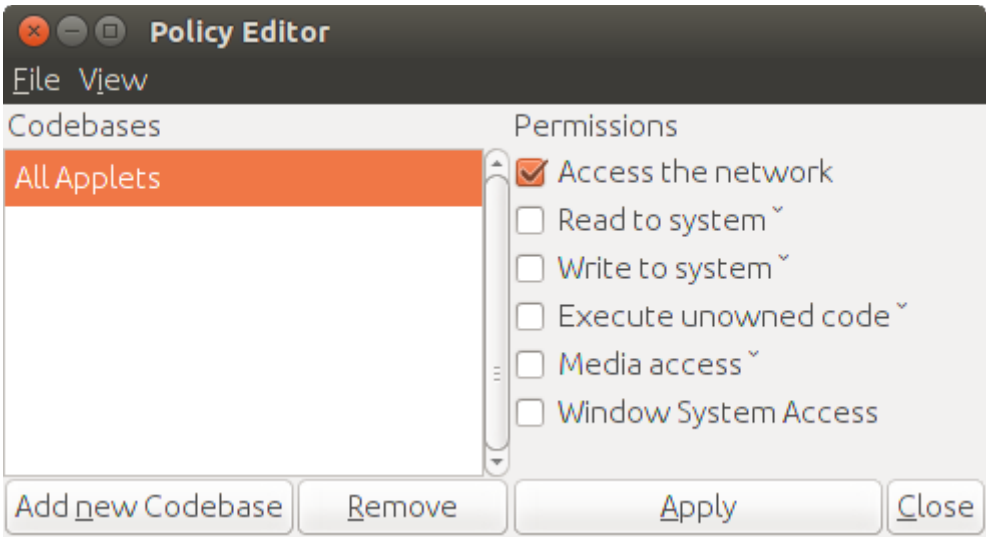
Happily, you can configure Iced Tea to run these "academic" applets safely without the display of any fear-invoking warning messages. I will go through the procedure for Ubuntu 14.04. There could be a fairly obvious corresponding procedure for other Linux operating systems.

1. Open Unity Dashboard.
2. Select the "Applications" tab.
3. Click on the Iced Tea icon.

The "IcedTea-Web Control Panel" window should open. Within this window alter the settings to be as follows:



4. Under "Cache" tick "Keep temporary files on my computer" and set amount of disk space to 199MB.
5. Under "Security" tick all the boxes.
6. Under "Policy Settings" open "Simple Editor", which should display as follows:



7. Select "All Applets" and tick "Access the network". This allows it to download its own data from its own web site.
8. Untick all the other boxes so applets cannot access your computer or your file system, or mess with your browser.
9. Under "Extended applet security" set "Low Security - All, even unsigned, applets will run". This is safe since you have forbidden applets to access your local filesystem.

Now you should be able to view my web pages that contain applets without any dire warnings appearing. Beware, however, that "commercial" applets will now not be able to access your local filesystem or interface with other programs running on your computer.

Perhaps the Open Source Community should create a variant of Firefox [we could call it Firefly] which is set by default to run "academic" rather than "commercial" applets.

End of Java Support

A scheduled update of the Ubuntu 16.04 operating system, in the second week of February 2017, included an update of the Firefox web browser. After this update, I discovered that none of my applets even appeared. Firefox ceased to be a Java-capable web browser. I had been aware of the threat to terminate Java support for quite some time. Other browsers had long since ceased to be Java-capable. The sentiment from the browser providers had been that Java was simply too unsafe to continue using in a browser, which seems to me somewhat incongruous.

Thus, a marvellous medium for animating illustrations for all kinds of scientific and mathematical phenomena, which had been with us since 1996, practically coincident with the beginning of the Worldwide Web, was thus dead. It left almost a hundred of my web essays useless. With the applet areas blank, the text was referring to nothing. It was a disaster.

Fortunately, the **Opera Browser** still runs my applets perfectly. So, if you want to see the original authentic applets running within their respective web page contexts, use the Opera browser. Perhaps some other browsers still support Java.

Notwithstanding, with the more popular browsers now blind to Java applets, some kind of solution was necessary. Although it can never replace the sheer convenience and slickness of presentation provided by a proper embedded applet, the Java Web Start application is a kind of workable option.

The 'Java Web Start' Solution

So, from 15 February to 17 March 2017 I had to drop my current projects entirely and concentrate on creating Java Web Start versions of all the applets I had written since 1997. This was a period of very intensive work, which had to include the learning curve for adapting to the Java Web Start way of writing programs. I left the original applets in place so that any Java-capable browser could still run them in their original form, because having an applet embedded within the web page that describes it was, is, and always shall be, by far the better option.

Notwithstanding, I needed to avoid a blank white area appearing, in place of the applet itself, in non-Java browsers. I used the option, which has always been there, to precipitate the automatic display of a static image in place of an applet in non-Java browsers. So now, in non-Java browsers like Firefox and Chromium, a static image of the applet appears in place of the functional applet. This static image of the applet is also a hyperlink to a ".jnlp" file, which launches a Web Start version of the original applet. Thus, provided the person viewing my web site has Java Web Start installed and enabled, clicking on the image opens a dialogue box inviting him to give his permission for the Web Start version of the applet to be launched. And all, supposedly, would be well.

But unfortunately, all is not well. The viewer is presented with a similar set of dire warnings about the authenticity and safety of my Web Start applet. With some of my applets, this has been avoided by encapsulating all of them in a ".jar" file sealed with what is called "sandbox" permissions. However, most of my applets are not permitted to run as Web Start applets with "sandbox" permissions. The only other permissions setting for a sealed ".jar" file is "all-permissions". This gives the Web Start applet full rights to create, access and modify files anywhere within the web site viewer's computer. That's dangerous.

The irony is that none of my applets needs "all-permissions". None of them needs to create, read-from or write-to any files outside the sandbox. But the fact that they merely need to *read* data from their respective ".jar" files requires that they must be granted "all-permissions". This is a serious flaw in the logic of Java security design. There is no reason, that I can see, why a Web Start applet's own ".jar" file should not be considered part of the sandbox. Simply reading data from its own ".jar" file cannot logically present any kind of risk to the web site viewer's computer. Yet for a Web Start applet to be permitted to do this, it must be sealed in a ".jar" file and given "all-permissions". Otherwise, it won't work. This is a completely unnecessary and highly disruptive requirement.

The adjacent table shows, what I would call a sensible logic of security for Java applets and Java Web Start application programs, in terms of read, write and execute permissions shown in Unix rwx (read, write, execute) notation.

SB	Int	All	Zone
`---`	`---`	`rwx`	client
`r--`	`r--`	`rwx`	jar
`r--`	`rw-`	`rwx`	server

These permissions are shown for the three minimum permission levels under which I think Java applets and Java Web Start application programs should be allowed to run. These are "sandbox permissions" [SB], "interactive permissions" [Int] and "all-permissions" [All].

None of my applets needs to read-from, write-to or execute any file within the client's [the web page viewer's] computer file system. Logically, therefore, they should all be able to run freely at the "sandbox" permissions level.

Some commercial Java applets and Java Web Start application programs involve the client user filling in on-screen forms and sending the in-fill to a vendor's server. These require what I have called an "interactive" permissions level. The need to read data from and write data to the vendor's remote server. Such programs could maliciously *phish* the personal data voluntarily submitted by a naive client. But they cannot access his computer's file system.

For corporate applications, which need to operate seamlessly across the Internet between the workstations of staff at diverse locations, the "all-permissions" level is required. This permits a program to read-from, write-to and execute files located on any of the participating computers across the Internet.

Persistent Certification Problem

The problem of certification remains. Every time anybody views one of my web pages containing an applet, he is asked whether or not he will trust the applet author [me] who is completely unknown to him. I, as an applet author, am not recognised by any self-declared authority as trustable. If he wishes to run the applet, the viewer must then "go out on a limb" and confirm his trust in me by accepting my certificate, which is frankly stated as being unknown and unrecognised by any competent authority.

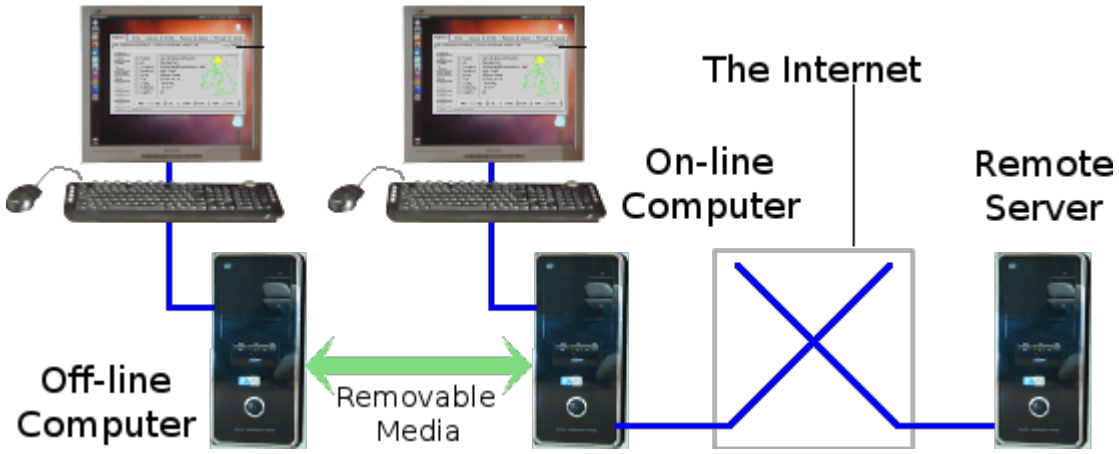
For the certificate, of the key with which I sign my applets, to be recognised by a "competent" authority, I would have to pay such an authority a lot of money. If I were requested or required to endorse somebody else's programs, I would need to scrutinise all their source code. Presumably, they would have to do the same in order to endorse my programs as trustworthy. The cost of such an undertaking would be astronomical in both time and money. Obviously, this is simply not going to happen. There is absolutely no way in which I could afford the fees of Certification Authorities in order to be registered as a legitimate and trustworthy applet author, even though this does not alter the fact that I am.

In an attempt to combat this problem, I understand that many applet authors have formed a group of mutual endorsement called Open SSL, although it is doubtful whether they would be recognised by browser providers. So all certificates endorsed by Open SSL groups will still be stated as unrecognised by browser messages. The only positive thing is that it appears that, once a viewer of any of my web pages has accepted my certificate to run one of my Web Start applets, then that applet, plus all my other applets, will run immediately without the intervention of the dire warning dialogues.

The original concept of the sandboxed applet was a stable platform for over 20 years from before 1997 to 2017. It is suddenly not there. Its replacement takes illustrative programs outside of their embedded web page contexts and creates an impossible authenticity problem. I cannot imagine the commercial interests of the Internet resolving this problem. I think it must be done - and inevitably will be done - by the Open Source community. In the meantime we are stuck.

The Only Secure Solution

The only foolproof way to achieve this is to allow alien programs to run only within a physically separate computer. Thus you have an off-line computer containing all your personal application programs and personal data. This is the computer on which you do your private work. You also have another separate computer (connected to the Internet) in which alien programs (such as Java applets) are allowed to run. You never leave or store any personal or otherwise sensitive data on your on-line computer. You use your on-line computer for surfing the worldwide web, sending receiving emails and exchanging files over the Internet. You also use it to serve your website and to serve files on other networks such as FTP, eDonkey or Gnutella.



You do all your normal private work on your off-line computer. When you need to send a finished piece of work across the Internet to somebody else, you proceed as follows. 1) encrypt your finished work on your off-line computer, 2) copy the encrypted file to a memory stick, 3) plug the memory stick into your on-line computer, 4) transmit the encrypted file to its destination, 5) delete the encrypted file from the memory stick. When you receive an encrypted file from a colleague across the Internet, copy the file onto the memory stick, plug the memory stick into your off-line computer, decrypt the file and wipe the encrypted version from the memory stick.

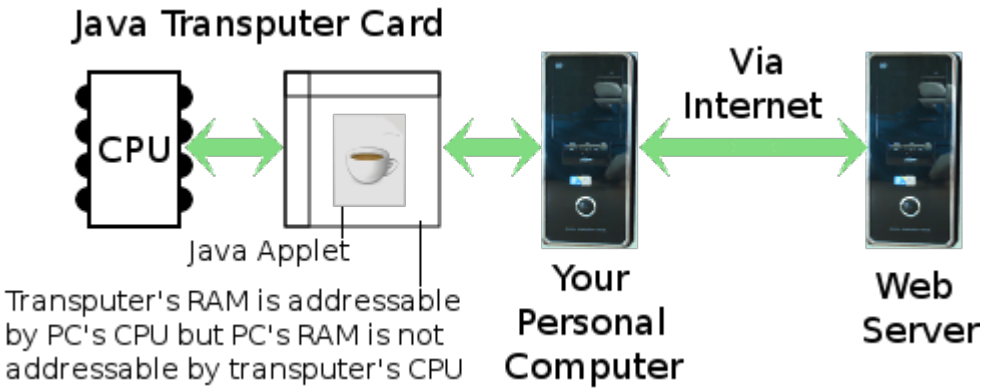
Of course, you must follow all the safety steps for ensuring that nothing malicious becomes transported from your on-line computer to your off-line computer via the memory stick. Always use a basic underlying format (such as ext2) for your memory stick. Never leave your memory stick where others could maliciously record something on it you don't know about. Lock it away when not in use. Never use any form of auto-starting or auto-running on the memory stick. Only ever read from it the encrypted file you put there: nothing else.

You also need to archive your emails in your off-line computer. You must use a basic email client on your on-line computer. This client should have the minimum necessary and sufficient functionality for the sending and receiving of emails. It must be able to save emails in passive files (with no active content such as macros). These files you transfer to the email archive on your off-line computer and delete them from your on-line computer.

Although this is probably the safest solution to the problem of Internet security, it is nonetheless extremely irksome and cumbersome to use. It is necessary to make some form of compromise between security and convenience.

The Java Transputer Idea

The best compromise, that I have so far been able to think of, is what I would call the Java Transputer. I got the idea from the Inmos Transputer of the 1980s. The Java Transputer could initially be implemented as a PC expansion card. Later, it could be integrated to become part of the PC motherboard. Thereby, it could be applied to laptops and even smaller devices. The essential principle of the Java Transputer is shown below.



The transputer comprises a central processing unit (CPU), random access memory (RAM), input/output (I/O) bus and all the associated circuitry necessary for these to function as an independent computer. My own preference is as follows. The CPU is a reduced instruction set computer (RISC) chip driven by re-flashable microcoding which enables it to execute up-to-date Java byte code directly within a Java runtime environment. It is thus, in effect, a Java computer complete with its own (non-disk) operating system.

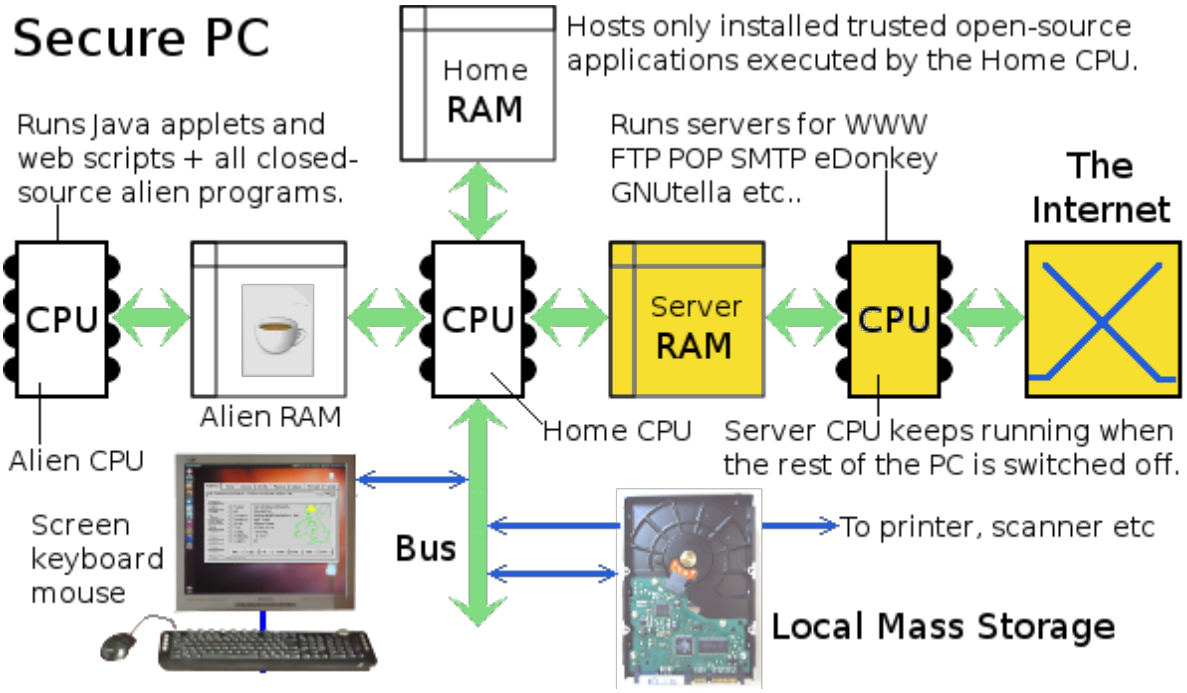
The transputer's CPU can address all of the transputer's memory but it is physically unable to address any of its host PC's memory. The PC's CPU, on the other hand, is able to address all of the transputer's memory. The PC's CPU sees the transputer's memory as a separate memory page outside its own normal addressing range. The PC's CPU cannot run any executables within the transputer's memory.

When an alien applet needs to be run, the PC stores the alien applet's executable byte code into the transputer's memory. It then advises the transputer's runtime system that it has a new applet to run. The transputer then starts to execute the alien applet together with any other alien applets it may have already running. Since the transputer's CPU is itself fundamentally incapable of addressing anything outside its own physical memory, no alien applet running within it can possibly interfere with the PC's resources.

Perhaps the Java Transputer idea should be thought of as a prelude to a complete re-think of PC architecture in the light of the Internet. I think that the PC of the future should comprise three quasi-independent machines: one to run the user's own applications and host his personal data, another to run alien applications such as Java applets and a third to manage the serving and safe transfer of data over the Internet.

A Secure Personal Computer

My own conception of this kind of secure personal computer is illustrated in the following block diagram. The diagram shows logical connections between components rather than physical ones.



The Home CPU (Central Processing Unit), the Home RAM (Random Access Memory), the Bus and attached peripherals form, what is, in principle, the traditional personal computer part of this new architecture. I shall call it the Home Transputer. The Alien CPU and Alien RAM form, in concept, what I have previously described as the Java Transputer but which I shall hereafter refer to as the Alien Transputer. The Server CPU and Server RAM form a Server Transputer. This, however, is able to run trusted programs written in native code (compiled C-source etc.).

The Home CPU is able to address not only the Home RAM but also the Server RAM and the Alien RAM. The Alien CPU is physically unable to address any memory except for its own Alien RAM. The Server CPU is physically unable to address any memory except for its own Server RAM. Even the Home CPU cannot address the Home RAM, Server RAM and Alien RAM as a single contiguous address space. To the Home CPU, each of the three RAMs is in a separate address space. This ensures that alien programs are physically constrained from ever being able to address computing resources outside the Alien RAM.

Alien programs are not allowed to access local mass storage. All their necessary temporary files are stored in the Alien RAM. An alien program is, however, permitted access, via the Internet, to the remote computer from which it was served. It is also allowed input from the keyboard and mouse, plus output to a designated area of the screen. But this is all done indirectly. Only the Home CPU has access to the PC's local peripherals. The Home CPU thus mediates between physical local peripherals and an alien program needing access to them.

An alien program, such as a Java applet, is able to communicate with the Home Transputer by means of messages. For this purpose, some space within the Alien RAM is designated as a message buffer. When the alien program wishes to write something to its display window on the computer's screen, it encapsulates what it wants to write, and where it wants to write it, in the form of a message. It places the message in the message buffer within the Alien RAM. The Home Transputer checks for new messages in the Alien RAM. On finding the alien program's output message, the Home Transputer reads it and expedites it, placing its content in the appropriate place within the alien program's window on the computer screen. The reverse takes place for input from the keyboard and mouse.

The same message principle is used for communication between the Server Transputer and the Home Transputer. Active programs running within the Server Transputer communicate with the Home Transputer via the Server Message Buffer. The Home Transputer scrutinizes all message traffic passing between the Server Transputer and the Alien Transputer. The message-passing process is implemented as an intermediate layer immediately above the Home Transputer's operating system. This message layer is the only means of access between the Home Transputer, the Alien Transputer and the Server Transputer. Its message analysis procedures thus form an ultra secure validation mechanism for all that passes between the three transputers.

A file received from a remote computer, via the Internet, appears as a message available within the Server Transputer's message buffer. The presence of such a message is signalled to the Home Transputer. A client program, running in the Home Transputer, is then used - with or without manual intervention - to approve and accept the file and move it to the Home Transputer's mass storage. The reverse is done to send a file from the Home Transputer's mass storage.

Essential to the design of this secure PC is uninterruptible power. Battery backup is a must. When the computer is switched off, the Server Transputer, which consumes relatively little power, remains running. It thus continues to serve files and receive emails.

Secure Software

The major culprit in the case of the overwhelming insecurity, which pervades the Internet today, is, without any doubt, bad software. In this context, the term *bad software* does not mean software that is error-prone or lacking in functionality. On the contrary, bad software is, for the most part, that which has too much functionality.

I remember well the many discussions I had with a colleague, in the early 1980s, about good programming practice. At the time, as today, certain dominant players in the IT Industry built proprietary extensions into programming languages. Using such extensions in applications inevitably caused an immense amount of unwelcome trouble. They may have been superficially helpful in their way. Nonetheless, sooner or later, programs would no longer run when one or other proprietor of such extensions unilaterally and arbitrarily updated its extensions. Furthermore, such extensions rendered it impossible to migrate an application to another operating system platform.

We therefore rapidly adopted a firm policy we called RISP. When programming an application, we would use the minimum necessary and sufficient resources of the language to program the application in hand. The RISP acronym meant Reduced Instruction Set Programming. We coined the term by analogy with RISC (Reduced Instruction Set Computing), which refers to a type of CPU design.

I continued the RISP philosophy beyond the mere coding of an application into its very design. The philosophy stipulates that you should only build into an application the minimum necessary and sufficient functionality to do the essential job. No additional bells and whistles. By restricting an application's functionality to only that which is essential, you also restrict its vulnerability to attacks. The most basic and most useful functionality of an application is generally of no use at all to an attacker. It is the non-essential bells and whistles that offer him a way in. So, for example, why make email communications vulnerable by giving email clients the functionality to run macros? People can send and receive all the information they want without them - and safely.

By following technical best practice, we would have a safe Internet with full user control over privacy and publicity. So why is the reality so insecure, riddled as it is with such glaring vulnerabilities? It's all a matter of motive.

If the market were of a mind to accept expert advice as to the best form and design for software, then we would all be using secure compact functional applications, the use of which would be easy and self-evident. Instead, we are forced to pay for, and endure, expensive time-consuming training courses to learn by rote how to use an over-complicated application, which does not seem to adhere to any logical structure or procedures. And the reason for this is that the application was not designed according to technical best practice but by the blind ignorant desires of the market.

To me, the mind of the market appears as that of an "I want" tantrum-throwing toddler who really does not know what it wants or what is best or even safe for it. A fickle desire enters its tiny mind. It then bleats and pesters for the technical experts of the industry to fulfil that desire. The result is an increasingly cumbersome and complicated application that was "never made" but just "growed" like Topsy. But why would the experts of the industry be party to such folly? Because giving the market what it wants is the way to make money. And the motive is money: not good safe secure software.

Security is a Money Spinner

Selling robust self-evidently usable software merely earns the price of each copy sold. Selling frumpy market-designed software as "easy-to-use" also earns the price of each copy sold. However, its complexity and illogicality precipitate a training course gravy-train, which stands to gain far more income than selling the software in the first place.

Furthermore, the security vulnerabilities, inherent within such software, precipitate an on-going war of attacks and fixes, which creates and sustains a whole additional *security* dimension to the software industry. Thus the user ends up paying an additional on-going subscription for keeping the security software up-to-date to combat the ever-multiplying hordes of mutating viruses.

To create and sustain the twin spoof necessities of training and security, a powerful yet precisely tuned marketing strategy is required. It must instil, within the market's collective mind, two notions:

1. The software industry comprises a mere handful of upstanding companies who are squarely on the user's side and endeavour to supply good reliable products.
2. All the insecurity trouble experienced by users is the work of rebellious IT students and other irresponsible nobodies, whose avowed intent is to sow mischief.

Within the second notion, all who create software or web content of any kind, who are not members of the industry's corporate elite, necessarily fall under the classification of "irresponsible nobody". The significant corollary to this is that whereas members of the industry's corporate elite are *trusted*, irresponsible nobodies are not. Hence the message box at the head of this article, which clearly labels my applet as *untrusted*. This forthrightly implies - nay, states - that I am therefore an *untrusted* source.

What can easily be missed here is the fact that merely describing something as *untrusted* is not a complete statement. It requires qualification. To complete the statement, it is necessary to know:

1. by whom is it untrusted?
2. for what purpose is it untrusted?
3. why is it untrusted?

By whom is my applet untrusted? One can only assume that my applet is untrusted by whoever wrote the warning message, namely: the proprietors of the Internet Explorer browser and its Java runtime system. The obvious and intended further implication is that because *they* do not trust my applet then *you* (the hapless and vulnerable web page viewer) should not trust it either.

For what purpose do *they* not trust my applet? The only answer apparent to me is that they do not trust it to fulfil, correctly and safely, its intended purpose as stated by its author. The only place where I explicitly state my applet's intended purpose is in its source code. Although there is a link to the source code of each of my applets on the web page in which it appears, most people probably won't look there. Nevertheless, by placing my applet within my web page, I implicitly state its intended purpose by context.

The warning message implies that my applet is unfit for that purpose. By so doing, it brands me, the author of the applet, as a "cowboy" programmer, whose dodgy software you (the viewer of my web page) are best advised not to trust. In fact, since early 2014, it does not matter whether you trust my applet or not. The browser simply will not allow the applet to run under any circumstances. It won't even let *me* run my own applet on my own computer!

Why do *they* not trust my applet? To me, the warning implies that whoever caused it to be displayed on my screen had a substantiated reason for having done so. It gives me the distinct impression that the displayer of the warning has, by some concrete means (manual or automatic), conducted a technical examination of the functionality of my applet and positively found it to be, to at least some degree, malicious or dangerous to the computer of whoever views the web page containing it.

The fact is that my applet is perfectly benign. It simply does the job that its context implies. Furthermore, its source code is freely available for examination, so any competent Java programmer could verify that it is safe to run. Notwithstanding, these self-appointed judges of my applet display their warning and unconditionally block my applet from running unless the viewer of my web page is prepared to "go out on a limb" and accept my unregistered certificate. The only conclusion I can therefore draw is that they condemn my applet out of hand without having examined it and for no better reason than that they know nothing about it.

A Strategy For Exclusion

This out-of-hand condemnation may be seen as an unfortunate collateral effect of protecting the majority of web users against the dangers of malicious attackers.

> I'm afraid I don't believe the story about the young Chinese whiz-kid hackers bent on wreaking havoc to gain some sick satisfaction that they have the skill and power to do so. It doesn't jibe. They cannot possibly have the necessary experience, skill, knowledge or resources for this.

Whatever the cause of this real or perceived insecurity, it is proving to be a very effective means - intentional or otherwise - of forcibly excluding from the Worldwide Web all content providers, with the exception of a self-chosen clique of large corporate players who have the vast financial resources necessary for getting all their applets "certified" as safe. So perhaps they are the culprits regarding Internet insecurity. Perhaps the whole security issue is created by those who want to make the provision of Web content exclusive to themselves, and thereby control what people think.

It reminds me of what happened to the habitable land of this planet. Once it was free for all to roam and from which to take their necessary and sufficient needs of life. But gradually, the hapless majority were excluded from it as it was claimed and fenced by an exigent few. Finally, the will of this favoured few became protected by laws such as the “Inclosures Acts”. The upshot was that a privileged elite became the possessors of all the land and resources of the Earth, throwing the dispossessed majority into frantic competition, going cap-in-hand to their local lord in desperate attempt to trade their labour for their needs of life.

Likewise, the Worldwide Web was once a soapbox on which any could stand up and have his say. But the exigent rulers of this world quickly saw this as an open danger. But to whom? To the vast majority of web viewers? No. To themselves. They were afraid of that majority of web viewers being exposed to ideas and opinions that could prejudice the security of the privileged minority. These non-conformist voices had to be stopped.

So far, they have succeeded in disallowing the free and open viewing of applets. What next? I fear that very soon you will only be allowed to view web content from an exclusive group of certified websites. Indeed, the time may soon be upon us when certain operating systems will no longer allow you to run any native program that has not been certified by some self-appointed corporate authority. Thus will dawn the age of the non-programmable computer. Such a retrogression inevitably occurs once any new device or infrastructure is observed to enhance the individual's means and freedom of self-expression.

Like radio and television before it, those who govern us want the Worldwide Web under their strict control. They, and they alone, must have access to all means of shaping public opinion. By passively allowing a powerful corporate minority to exclude the rest of us from visibility on the Worldwide Web, those with political power are able to maintain their stranglehold on all means of shaping public opinion. And it is far easier for political power to control a small clique of corporates than a myriad independent free-thinking individuals.

Related Articles: Who Owns Cyberspace? and About Software