Problems when running Untrusted Java threads and ways to solve them

Jiazhen Chen
jche117@ec.auckland.ac.nz
Department of Computer Science
University of Auckland, New Zealand

Abstract

The popularity and Java application and applets invite more hackers to break in to the Java system. The main potential threats rose by Herzog and Shahmehri [1] are unsafe termination, resource control and isolation between Java threads. This report is focused on criticisms of the paper published by Herzog and Shahmehri. Several criticisms has been raised and backed up by the book by McGraw and Felten[4] and paper published by Felten[3]. The main criticism is that this paper has reviewed old ideas which have been foreseen by the book [4] five years ago, and the authors did not do a good job reviewing the old ideas. Another criticism is that the authors never mentioned in detail how to prevent malicious thread and control what an untrusted thread can do. Those two additional resources carried on the problems raised by [1] and explained the suggestions to counter those attacks in detail.

1 Introduction:

Java thread system is known efficient because Java threads are designed to be cooperative and friendly with each other. In comparison to the normal processes, the Java threads require less overhead therefore requires less memory and computing power to manipulate. To be cooperative, the access control between Java threads are less restricted than the control between normal processes. The security between threads therefore is also less controlled. This approach is based on the assumption of "Java threads within one JVM are created by one application". In other words, threads that can manipulate each other are created by one application, so they suppose to trust each other. This is true in the case of local Java application, but in case of network applications such as Java Applet, RMI application and Java Servlet this is done in different way which untrusted code is running with normal threads in a same

container. Herzog and Shahmehri have suggested three main potential threats when running untrusted threats as a java thread which contained in the same JVM with other normal threads. The problems were clearly presented and suggestions are given. Several doubts have risen in my head after reading this paper. My main criticism is the paper is reviewing the security problems which have been foreseen by McGraw and Fenten five years ago or even longer. It is perfectly fine to review somebody's work. Herzog and Shahmehri however did a bad job explaining their solutions to the problems because the solutions are too simple for readers to understand. My second criticism is the authors have not mentioned the real harms that Java security issues can cause to the system. A question "Why do we have to spent all the effort on some problems that are not that serious" can be asked. Authors never mentioned the cost to spend on building the security mechanism. My last criticism is that authors have mentioned the problem "Isolation between the threads" and "resource control" as the main problems, but the suggestions given to solve those problems is not explained in details and no reference has been given for further research. The structure of this paper is focused on those three criticisms. Section two reviews the Java virtual machine and its relationships with Java processes and Java threads. Section three review the problems identified in this paper. Section four focused on the first criticism: The authors did a bad job reviewing the ideas more than five years ago. Section five focused on the second criticism: The harms that caused by the problems listed in this paper are not very serious therefore the solutions suggested by the authors are too expensive to implement. Section six focused on the last criticism: The solutions to the isolation problem and resource control problem presented in this paper are too simple to understand. Solutions given by the user is too simple to convince the reader that their suggestions can counter the problems. Section seven is the conclusion of this paper.

2 Relationships between Java Virtual Machine, Process and Java Thread

Java Virtual Machine is a specification for an abstract computer [2]. Each time a Java process is created, it is assigned to a JVM which acts like a real machine. The process has the full control of the JVM's computer resources. This approach provides isolation between each process and avoids malicious contact between processes. In theory threads created by one process are contained within one JVM which other

threads have no accesses to. The threads can safely manipulate each other without worrying about interventions from other untrusted threads. This approach is much more efficient than normal process systems.

3 Problems

Consequences of running untrusted service as a thread can be serious. Based on the findings of Herzog and Shahmehri [1], problems of running untrusted threads within the same JVM with other threads can be summarized in three main points.

3.1 Safe Termination

If an untrusted thread is running within the same JVM, it could prevent other threads from running. The reason for this to happen is “The stop method within the thread class is not safe to use”. Therefore the system can not force the thread to terminate once it is running indefinitely. Malicious thread can be intentionally written in infinite loop, which keep them alive as long as they want. According Herzog and Shahmehri [1], the execution environment will stay in a hung state if the malicious thread running infinitely and system trying to shut it down or uninstall the application that thread belongs to. Safe termination already been used as one of the most popular Java applet attack on the internet. Herzog and Shahmehri provided little details of how the attack can be organised.

3.2 Resource Control

According to Herzog and Shahmehri [1], Java has a good access control system. Once the access is been granted, the thread can have the full control of system resources been assigned to it. Therefore a malicious thread could try to get the permission and use that permission to control all the resources, which prevent other threads running duel to the lack of system resources. An example has been give by the author[1], if a file “malicious.jar” is allowed to write to /tmp/a, it can fill the disk by write empty data such as zero. Once the disk has been filled, no other threads could perform the write command to that disk and also leads to system shut termination since there is no disk space to store temporary system files.

If a normal thread has been granted the access and can not have the resources it wants, the ability of Java threads can be greatly reduced. Authors mentioned little about how to prevent untrusted code getting the access of complete system resources. This approach will be later reviewed based on another author's publication [3].
To summarize, resource control can be indeed a serious problem to Java system.

3.3 Isolation

Isolation between Java threads is not completely secured. Herzog and Shahmehri[1] suggests the class loading isolation mechanism provided by Java is not a perfect isolation mechanism. The authors also suggested interference between Java threads within the same JVM can still happen. Mutable parts of system classes such as static fields and methods can leak object references. Little details have been provided about the harm of those interferences.
According to authors, better isolation is achieved by the Multitasking Virtual Machine. The idea of MVM is to share the application code whenever possible and replicate the code only when needed. In contrast to Class loading mechanism which do not share code enough and replicate too much.
To summarize, the isolation problem is not well explained by the authors. No example has been provided about the isolation. The consequences of isolation problem are also not well explained.

3.4 Author's suggestions to the problems

3.41 Unsafe thread termination

According to the author's suggestion [1], the system should consider running the untrusted code as a separate process in its own Java Virtual Machine if the safe termination is an issue for the container. Process states can be tracked by the system where thread can not. Therefore they can be monitored and controlled by the operating system more effectively. Since each process is running on their own virtual machine, the untrusted threads and the normal threads are more isolated.
Author did suggest that this approach would greatly increase startup overhead and increase use of system memory because of the additional process. Author however

mentioned little about the situation where untrusted thread force to run within the same JVM with normal threads duel to the limitation of memory space. Off course there is no suggestion for that particular situation as well.

3.42 Insufficient Isolation

The insufficient isolation between threads provides the opportunity for the malicious threads to start a denial-of-service attack for other applications. According to author's example, malicious thread could hijack the shared system class such as finaliser queue which may invoke the garbage collector. Garbage collector is used to reclaim memory from objects no longer in use and returning it to the system [2]. When garbage collector is invoked, all the other threads will be suspended while it runs. No explanation and solution has been given by the user except a reference to the other people's work. The authors never mentioned the prevention of the denial-of-service attack which will be explained in detail based on another academic paper.

3.43 Lack of resource control

Author did suggest the access to the CPU should be controlled, which prevents one thread starves other threads within the same JVM. The solution to that is running each process in its own virtual machine. How can we manage the resources under the situation where only one JVM can be used and all the threads include trusted and untrusted are forced to run within the same JVM? Author gave no clear explanation of his own problem.

To summarize, authors gave a detailed description of the problems that they found. Suggestions made by the authors are not explained in details, which can not be very helpful to the reader. More importantly, many of the problems have already been foreseen before this paper was published.

4. Critical comment one: Authors review and suggestions are not well explained

Three problems presented in the paper [1] have been foreseen by the book [3] “Securing Java” by Gary McGraw and Edward W.Felten. The most important thing is that in paper [1] authors did not explain the harms that those security issues can cause. The main aim in this paper [1] is to examines the risks associated with Java threads that run untrusted code and presents existing research solutions. Obviously they did not present all the potential problems and the existing solutions. McGraw and Felten did a great job foreseen the potential threats and ways to solve the problems. Yet Herzog and Shahmehri never referenced their work. Based on the comparison between the two works, McGraw and Felten gave more details on problems and well explained in solutions.
Gary McGraw and Edward W.Felten categorized the potential Java threats into four major attacks.

System Modification: This is considered the most severe attack by the Author. It can be implemented through Java applets which can then modify system through the browser. The Java defence against this kind of attack is however very strong.

Invasion of Privacy: This kind of attack can be also implemented through applets which can disclose private information of the user to the public. The consequence of this attack is considered moderate to the user and has strong defence from Java.

Denial of Service: Consider to be serious attack by the author. This attack can bring a system to a standstill. Malicious applets can achieve this by make resources unavailable which then require the system to be rebooted. The defence against this kind of attack is weak.

Antagonism: This attack basically annoys the user by making unwanted sound through the speaker or displaying weird pictures on the screen. The consequences of these attacks are considered moderate by the author and Java has weak defence against this kind of attacks.

From the four main attacks summarized by McGraw and Felton, we can easily discover the similarities of findings between [1] and [4]. System Modification attacks is generally caused by the lack of isolation between threads because lack of the

isolation between threads could allow malicious threads to modify the system. Denial of service attacks can be caused by the unsafe termination and resource control problem because its attack is based on holding up the resources or running malicious thread infinitely which prevent other threads running. On top of that, McGraw raised the issues such as the invasion of the privacy and Antagonism. The book by McGraw and Felton is published at the year of 1999. The paper published by Herzog and Shahmehri is published at the year of 2004. The security has already been foreseen by McGraw and Felton. Which in later sections of their, they have provided details of defence against those attacks with comprehensive explanation and real Java code. Herzog and Shahmehri in the other hand published the similar content and provided little explanation and example five years later after McGraw and Felton's book.

The solutions provided by McGraw and Felton to count those four attacks are following:

System Modification: According to author's opinion [4], system modification attacks have not yet been seen outside the lab. Because Java has a sandbox security mechanism which means all the untrusted code is contained inside the sandbox. Once a malicious thread is inside the sandbox, it does not have the permission to modify the system such as writing files to the local drive. This approach limits the harm that a malicious thread can do. This is one of the reason attacks based on Java language is not widely implemented. Security problem however still exists which can not be overlooked.

Invasion of Privacy: This kind of attacks typically implemented by forging the mail. Outsider can gain enough information from the user by forging the mail through malicious threads. The defence from Java against this kind of attack is strong because the sandbox security mechanism. Once the thread is contained within the sandbox, it can not reach the file I/O. However if the malicious attack is implemented by a Java applet, it always have a channel open back to the host which it can send all the information it got from the threads within the same container. Defence such as disabling the network ports which prevents the applet send back the useful information to the host can be effective depends on the situation.

Denial of Service

Denial of service attack is the most common Java security concern. This is because it is easy to implement and current security model has little defence against this threat. Authors have listed few examples of Denial of service attacks:

- Completely filling a file system
- Using up all available file pointers
- Allocating all of a system's memory
- Using all of the machine's cycles(CPU time) by creating many high priority threads

Fortunately the harm done by this attack is considered to be moderate because by killing the process or at most reboot the system could stop malicious threads consume all the system resources. Nothing in the system can be changed by this attack since the malicious threads are contained inside the sandbox.

Antagonism

The best solution suggested by the user is to just shut down the application that annoys you. This approach will not cause any inconsistency simply because those attacks are not harming your system. Restart the application and avoid running the annoy applet again seems to be the best solution.

By comparing the paper by Herzog and Shahmehri [1] with the book by McGraw and Felten[2], I found McGraw and Felten's Book covers more Java thread security issues than the paper published by Herzog and Shahmehri. Herzog and Shamehri never mentioned the attacks such as Antagonism and Invasion of Privacy. Also the suggestions such as Lack of resource control by Herzog and Shahmehri are not well explained in details and no useful references have been given to some of the solutions. (For example no reference has been given by the authors to the suggestion "Lack of Resource Control" in Page 27). The paper published five years later than the book by McGraw and Felten, which means they have plenty of time to review the old ideas. At least they can cover all the issues which have been pointed out in the past and give detailed suggestions and references to the problems.

5. Critical comment two: The damage is not critical to make major changes of JVM

From the above section, we can see the two attacks (Denial of service and Antagonism) that are most likely to be successful but they can do little harm to the system [4] . The two attacks that cause harms to the system can be defended by Java security mechanism (System modification and Invasion of Privacy). This is one of the main reasons why people often overlooked the Java security issues. One question is how important those security issues are in the real life. The motivation behind Herzog and Shahmehri is those issues can be very critical, therefore needs to spend a lot of effort change the way how threads work. Based their suggestions, Java needs to redefine its security policies, re construct existing security managers and restrict the ability of threads by isolate the threads. Should we ask ourselves, do we need to spend a lot of effort on some security issues that can do limited harms? If they do cause serious harms, show us the evidence. McGraw and Felton listed few things what Untrusted services can and can not do.

Untrusted Java code can not do (most important parts):

- Read, write, delete and modify files on the client file system
- Create a directory on client file system
- List the contents of a directory
- Check to see whether a file exists
- Obtain information about a file
- Create a network connection
- Listen for or accept network connections
- Obtain user's username or home directory name through any means
- Define system properties
- Make the Java interpreter exit
- Create a SecurityManager
- Create a ClassLoader

Etc…

Untrusted Java code can do

- Access to the CPU of the client machine
- Access to the memory which to build objects
- Access to the web server which this code is downloaded

Based on McGraw and Fenten's opinion [4], there are little things that an untrusted code can do which is critical. Indeed we can not force the threads to restrict the usage of the computer resources because it would greatly reduce the flexibility and ability of threads. After all, who needs an applet in the browser that can't do anything? The harm is limited. Therefore even if an attack is successful we can recover it relatively easy so it is unnecessary to reduce the efficiency of threads by enforcing more rules on it. The most harmful attack with a high success rate is the denial of the service attack. If we assume the attack is successful, we don not have to terminate the thread; we can just terminate the whole process without effecting the operating system and file system. If Herzog and Shahmehri think the problems are critical, they should at least identify the consequences of those attacks.

6. Critical comment three: Never mentioned untrusted threats prevention

One of other things that Herzog and Shahmehri never mentioned is how to avoid the situation untrusted code can modify other threads within the same JVM. Author explained in few sentences which suggest we should keep them isolated and with a fine grained thread access control. A normal thread access control either gives all threads the same permissions or no threads been given the permission. A fine grained thread access control focus on granularity of thread group which gives different threads with different permission. This approach separates trusted and untrusted threads into two groups which untrusted threads are given minimal access permission. This approach is never explained in details or referenced to additional readings. The interactions between threads are very important security issue which author noticed. The reason why they did not go in deep remains unknown.

The co – author of the [2] E.W.Felten published an academic paper in 1997[3] which specifically addressed this problem. Traditional JVM has two properties which allow the check to be successful:

- Every class in JVM which came from the network is loaded by classloader and included a reference to that classloader. Local class has a special system classloader. So they can be distinguished just by looking at the classloader. The class from network generally considered to be untrusted.
- Every frame on the call stack includes a reference to the class running in that frame. This property can be used for debugging and diagnostic. Untrusted code can be then monitored by the system.

Once the classes are distinguished, different security policies are then applied to different groups of threads [3]. However despite threads are distinguished, they are still contained within the same sandbox. Both trusted and untrusted codes are not allowed to write, read and perform other useful operations. To address this inflexibility, Authors suggest three strategies to solve the problem.

Capabilities: According to [3], a Capability is an unforgeable pointer to a controlled system resource. Based on security policy, a particular thread is given the capabilities for whatever resources the thread is allowed. This way it is more flexible to assign the permissions to different group of threads.

Extended stack introspection: Three basic primitives are necessary to use this approach: enablePriviledge (target), disablePriviledge(target), checkPriviledge(target). When a critical resources need to be protected, a target such as untrusted code needs to be defined for resource access and the system must call checkPriviledge(target) before starts the untrusted code. EnablePriviledge is called when checkPriviledge is passed which allows the target to access the particular resource. And disablePriviledge is called after the target has finished using the resources.

Namespace Management: In OO languages, classes represent the resources such as file system and network. For example “File” class represent the file resources. By applying namespace management, we could make this file class invisible when

untrusted code is running which prevents the attacking on the file system. More detailed is explained in the paper [3].

7 Conclusions

Although Herzog and Shahmehri have provided a good overview of the potential threats when running untrusted threads within the same JVM with normal threads. I have several critical comments after reading the paper. Herzog and Shahmehri have suggested large improvements of JVM are needed to ensure the system resources are well controlled and interactions between threads are more isolated. They never mentioned the performance issue and the effort need to make all that happen. Based on [4], we found that the damage the potential security problems raised by Herzog and Shahmehri can do to the system is very limited due to the protection of sandbox mechanism. Even if one of the attacks is successful, system is able to recover with limited cost. The serious attack suggested by Herzog and Shahmehri that can do severe harm is rarely seen and hard to be successful. Therefore at this point we do not need to change entire Java structure to meet Herzog and Shahmehri's requirements in order to defend the potential Java threats. In addition Herzog and Shahmehri published the paper in 2004 and all the problems they presented are foreseen by McGraw and Felten who published the book in 1997. McGraw and Felten provided real code examples and ways to defend the threats which Herzog and Shahmehri mentioned little. Therefore there is an ethical problem here. Is it ethical to use other people's ideas and publish the paper as their own research findings? The paper [1] published five years later than the book [2] which give Herzog and Shahmehri plenty of time to come up with better suggestions and exploit new Java threats.
Herzog and Shahmehri spend many pages explaining the importance of resource control and isolation. The suggestion they provided is not detailed enough to persuade reader to believe their suggestions could counter those kinds of attack. [3] Proposed a comprehensive overview of this problem and designed three strategies to defend and prevent this kind of attack.

Acknowledgement:

I like to thank Mr. Clark Thomborson for his guidance and suggestions for this term paper. I also like to thank my friend Gilbert Notoatmodjo for his assistance on grammar checking of this report. My warm thanks go to Jonathan Wright who critically reviewed my paper and made me restructure the whole thing!!

Reference

[1] A. Herzog, N Shahmehri, "Problems Running Untrusted Services as Java Threads", in Certification and Security in Inter-Organizational E-Services, IFIP 18th World Computer Congress, ed. Nardelli et al., Aug 2004, pp. 19-32.

[2] A.Silberschatz, P.Galvin, "Applied Operating System Concepts 2nd edition" Chapter 5

[3] Dan S. Wallach, Dirk Balfanz, Drew Dean, Edward W. Felten
October 1997 ACM SIGOPS Operating Systems Review , Proceedings of the sixteenth ACM symposium on Operating systems principles, Volume 31 Issue 5

[4] G. McGraw, E,W, Felten. "Securing JAVA Getting Down to Business with Mobile Code".