Did this paper address an important issue? Explain.

As has been shown by recent worm attacks and noted in the other papers, worms, viruses and other malicious programs can wreak havoc with Internet infrastructure and also cripple critical computing facilities. These worms usually exploit some vulnerabilities in the programs to launch such attacks. Software techniques have been proposed to defend against such attacks, but such techniques incur significant performance overhead. Also, if the software technique depends on a ‘library’ of known or detected worms, then it becomes ineffective against mutating (polymorphic) worms.

These days mobile computing devices (mobile phones, PDAs, etc.) are proliferating. These devices are also being targeted by worms. Since these devices have very limited power resources, incurring the performance overhead mentioned above is not a feasible approach. Also, these devices have limited storage capacities, and so maintaining a library of known worms is also not feasible.

Thus, all the computer markets (server, desktop and embedded) will benefit if there can be a cheap (in terms of hardware, power consumption, and performance penalty) solution by which worms can be detected and thwarted before they damage/affect any computing and/or communication resources. This paper is presenting a new technique, called Dynamic Information Flow Tracking, by which, with a little support from the OS and with little hardware overhead, worms can be detected and thwarted. Hence, this paper is dealing with an important issue.

Are the proposed approaches valid? Describe its strength and weakness.

Yes, the proposed approaches are valid. For any worm to gain control of a program, it has to ‘hijack’ the CFG of the program and start executing malicious code by transferring control to the malicious code. The sources from which such codes/data are input are usually external sources like network I/O, disk or user input. The proposed technique requires that the OS mark data from such ‘untrusted’ sources as spurious. In the hardware, tags are maintained to track the flow of spurious data. There are well defined rules by which these tags are set, propagated or reset. If any spurious data is being used as a jump target or as an instruction, then the hardware raises an exception and the OS can terminate the attacked process.

Strengths

Since it is a hardware technique, it does not depend on known ‘libraries’ of worms and viruses. It can detect use of spurious data as instructions/jump target at runtime and cause an exception. The hardware overhead associated is quite low. Also, the authors propose ways to reduce the overhead of storing a large number of tags in the memory, by having various granularities of data blocks that are tagged. A potential limitation of the scheme will be that even if ‘spurious’ data is used in a legitimate manner, there will be false alarms. But the authors propose software techniques to overcome this limitation. The support required from the OS is minimal and can be easily provided by the OS. Considering the importance of thwarting worm attacks, the degradation in performance and the memory overheads are very small. This is especially true if this technique is compared with other
software based approaches. Due to these reasons, this technique can be readily applied to mobile devices.

**Weaknesses**

Unfortunately, since this is a hardware technique, it will be applicable only to new systems with such hardware support built into them. This technique does nothing to protect the already existing machines from worm attacks. These machines will have to continue to rely on ‘traditional’ software based methods. Also, since this technique is only marking the data from untrusted sources as spurious, it does not provide any protection from any inherently malicious program. For e.g., suppose a program is corrupted by a new virus (which is not yet detected) and some malicious code has been inserted into the binary of the program by the virus. Next time the program is executed, the malicious code will look like a part of the ‘original’ program to the hardware, and the hardware will not be able to detect that there is any malicious activity going on. Also, there is a check to ensure that legitimate use of spurious data does not cause any alarm. This fact can be exploited by a carefully crafted worm to defeat the hardware and gain control of the system. (Of course, this will be very difficult, but the possibility exists.)

**Do the results support the conclusions? Explain.**

The testing of the techniques was interesting, since this is not a ‘routine’ performance evaluation test. Here the authors had to locate vulnerabilities in the programs that they were running and stage attacks to try to exploit these vulnerabilities. The results show that all the attacks were detected and stopped, by both the ‘security policies’. Also, they have verified that there are no false alarms, by running the SPEC benchmarks (this is because they annotate the binary to verify that any use of spurious data is legitimate). As mentioned earlier, the performance degradation and the memory overheads are quite small even while using the more elaborate ‘policy 2’. Thus the authors conclusion that their approach is efficient and effective is valid.

**Describe the potential future works?**

The current techniques are only useful for stopping a worm from getting control of a program (and thereby the system). But it does not prevent the attack from happening in the first place. If such ‘attack detection’ mechanism can be built, then the overall security infrastructure will become very strong and attacks will not be able to cause even temporary disruption of computing and/or communication facilities.

In this technique, if a attack occurs and is detected, the program needs to be terminated. This requirement can itself be a very serious limitation. (Consider an attack on a nuclear plant’s thermal management system! We can’t afford to shut down the program and start it again.) May be, by keeping some history (some kind of check-point) of where in the program the malicious spurious data was inserted, the system can ‘roll-back’ to that state and re-start from there, instead of shutting down completely. Of course, this kind of check-pointing may require large space overheads and may also degrade the performance.

Another point is that if an attack occurs, then some part of the data being used by the program may get corrupted. The techniques in this paper do not prevent such corruption, and there is no recovery mechanism. A check-pointing based technique may be used to solve this problem.