What are the problems solved by this paper?

Attackers (hackers) try to gain control of a system and to access information present on that system. To do this, hackers exploit certain vulnerabilities in programs. The most common vulnerabilities are ‘buffer overflows’ and ‘format strings’. Using these vulnerabilities the hackers can insert malicious instructions and/or data into any memory location that is writable by the hacked program. If the hacker is able to insert the code and/or data he desires, he can effectively take control of the program. If the program is a ‘privileged’ program, like a daemon, the hacker can gain control of the system. Thus, such vulnerable programs pose a security risk.

This paper tries to reduce most of these security risks by using some hardware techniques. With suitable support from the OS, the techniques can help prevent hackers from gaining control of the system.

What are the approaches attempted by this paper?

The paper hardware techniques, with some support from the OS. They call their technique *Dynamic Instruction Flow Tracking*. If a program reads input data from ‘untrusted’ sources (e.g. disk or network I/O) then the OS marks these data as spurious. Basically, all the data that are not generated under program control can be considered spurious. If the data is from a trusted source, or was generated under program control, then it is marked as authentic. A one-bit tag is used for this purpose (‘0’ => authentic, ‘1’ => spurious). In the processor, each register is tagged, while in the memory, data are tagged at various granularities. The setting of these tags is done by the OS when the data are loaded. After tagging, the processor tracks the usage of these data. This is done by tracking various dependencies to track any new values produced by the spurious data and tagging the new values based on certain rules. Whenever any spurious data are used as instructions or jump target addresses, the processor raises an exception and the OS can terminate the attacked process. Thus, even the attack cannot be prevented, the hacker cannot gain control of the system.

The authors also describe two security policies and discuss how tags can be used efficiently to minimize the overhead associated with tagging data. Sometimes, data that has been marked spurious may be actually legitimate and so it will be incorrect to raise exceptions when such data are used as instructions or jump targets. The authors describe how such incorrect exceptions can be avoided by using some software techniques, or by help from the OS.

What are the main conclusions of this paper?

The authors tested some benchmarks with various vulnerabilities. They found that their approach can effectively detect any attempt to hack into the system by feeding in malicious data as input to privileged programs. An important observation was that their approaches do not generate any false alarms, i.e. any legitimate uses of spurious data are allowed, and unnecessary exceptions are not raised.

The overall hardware cost is not much. Also the performance overheads are quite acceptable. In case of policy 1, the overheads are around 0.26% and in case of policy 2, it is around 4.5%. The performance
overheads are also very small - 0.02% in case of policy 1 and 0.8% in case of policy 2. The binary annotation that is needed to allow legitimate uses of spurious data in policy 2, can be done once when the program in installed on the system or at the time of loading by the OS. Thus, they avoid recompilation of the programs.