Software fault isolation (SFI) offers a technical solution for sandboxing binary code of questionable

provenance that can affect security in cloud computing. Insecure and tampered VM images are one

of the security threats because binary codes of questionable provenance for native plug-ins to a Web

browser can pose a security threat when Web browsers are used to access cloud services.

A recent paper [322] discusses the application of the sandboxing technology for two modern CPU

architectures, ARM and 64-bit x86. ARM is a load/store architecture with 32-bit instruction and 16

general-purpose registers. It tends to avoid multicycle instructions, and it shares many RISC architecture

features, but (a) it supports a “thumb” mode with 16-bit instruction extensions; (b) it has complex

addressing modes and a complex barrel shifter; and (c) condition codes can be used to predicate most

instructions. In the x86-64 architecture, general-purpose registers are extended to 64 bits, with an r

replacing the e to identify the 64 versus 32-bit registers (e.g., rax instead of eax). There are eight new

general-purpose registers, named r8–r15. To allow legacy instructions to use these additional registers,

x86-64 defines a set of new prefix bytes to use for register selection.

This SFI implementation is based on the previous work of the same authors on Google Native

Client (NC) and assumes an execution model in which a trusted run-time shares a process with an

untrusted multithreaded plug-in. The rules for binary code generation of the untrusted plug-in are:

(i) the code section is read-only and is statically linked; (ii) the code is divided into 32-byte bundles,

and no instruction or pseudo-instruction crosses the bundle boundary; (iii) the disassembly starting at

the bundle boundary reaches all valid instructions; and (iv) all indirect flow-control instructions are

replaced by pseudo-instructions that ensure address alignment to bundle boundaries.

The features of the SFI for the Native Client on the x86-32, x86-64, and ARM are summarized in

Table 5.4 [322]. The control flow and store sandboxing for the ARM SFI incur less then 5% average

overhead, and those for x86-64 SFI incur less than 7% average overhead.