Paper1: Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks

This paper discusses randomized instruction set emulator(RISE), which is an instruction set obfuscation technique applied on machine level emulator. RISE is used to provide each binary code copy of its own unique and private instruction set. RISE does not answer all the security issues or attacks, but it's main exercise is to try and stop some of the binary code injections made by attackers through network. The attacks discussed in the paper are binary code injection from the network into executing program including misallocated headers, footer tags and string formats that can write the byte to an arbitrary location without overflowing a buffer. Main goal of RISE is to free user code or program from these malicious attacks. In addition, RISE takes care of the buffer overflow attacks by utilizing the address obfuscation method. But there are some attacks that RISE doesn't provide any defences against, such as data-only attack, which can range from the modification of jump addresses and parameters to call an existing library function, to modification of password files or other critical information.
With respect to performance, RISE-protected program is 5% slower than the program which does not use RISE, due to scrambling and unscrambling of the code before execution. But the response time of the code is still better and reasonable on a low configuration computer such as 200 MHz Pentium computer with 128 MB RAM as used by the authors.

Reference:
Barrantes EG, Ackley DH, Forrest S, Palmer TS, Stefanovic D, Zovi DD. Randomized instruction set emulation to disrupt binary code injection attacks. Proceedings of the 10th ACM conference on Computer and communications security. Washington D.C., USA: ACM; 2003. p. 281-9.

Paper2: Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code

This paper introduces a new method Self-Transforming Instruction Relocation(STIR) that enables binary code to convert to self-randomizing code, each time it is loaded. The input to STIR is only the application binary code without any source code, debug symbols, or relocation information. The output of STIR system is the new binary code whose basic block addresses are dynamically determined at loading time of the program. Therefore it decreases the probability of the attacker to inject any code that matches the newly randomized code developed by STIR. STIR-enabled code randomly reorders the basic blocks in each binary code section every time it is launched, disturbing attempts to predict the locations of gadgets. STIR code is also easily deployable as users only need to apply STIR to their binaries to generate one self-randomizing copy. The binary code is then distributed normally. The randomized code from STIR can overcome the return oriented programming(ROP) attacks.
Although STIR have many advantages, there are some limitations to it. STIR randomizes the code only at load time and not at runtime. For example static JIT compiler can be randomized but JIT compiler code generated at runtime cannot be randomized as this task is difficult to

realize in practice as more complex varieties of aliasing for each round of randomization. Also, STIR method presented in this paper can only be used to randomize the main code but not Windows or Linux libraries. It does not protect against control-flow hijacking attacks that simply call a legitimate computed jump target with corrupted arguments. The authors have also evaluated STIR for both Windows and Linux platforms. In conclusion, STIR in binary code introduces about 1.6% overhead on average to application runtimes.

Reference:

Wartell R, Mohan V, Hamlen KW, Lin Z. Binary stirring: self-randomizing instruction addresses of legacy x86 binary code. Proceedings of the 2012 ACM conference on Computer and communications security. Raleigh, North Carolina, USA: ACM; 2012. p. 157-68.