Despite the growing popularity of interpreted or byte-compiled languages, C/C++ and other languages targeting native code are still dominantly used for system programming. Programs compiled to native code present a set of challenges compared to alternatives. In particular, in this work we focus on how they can be efficiently analyzed, how existing security measures (known as "binary hardening techniques") perform, and how new ones can be introduced to secure features that have received little attention.

We propose rev.ng a binary analysis framework based on QEMU, a popular dynamic binary translator and emulator, and LLVM, a mature and flexible compiler framework. rev.ng can easily handle a large number of architectures and features a set of analyses to recover basic blocks locations, function boundaries and prototypes in an architecture- and ABI-independent way. rev.ng can be used for instrumentation, debugging, decompilation, retrofitting of security features and many more purposes. Our prototype encompasses about 17 kSLOC of C++ code and has been publicly released under a Free Software license. The core component of rev.ng is revamb: a static binary translator which can accurately identify all the basic blocks, and, in particular, the targets of indirect jumps for switch statements. Along this work, we will make heavy use of analysis techniques popular in the compiler literature, such as Monotone Frameworks, to recover an accurate control-flow graph, identify
function boundaries and the number and location of function arguments and return values. We will also discuss how rev.ng can handle native dynamic libraries, how it can be easily employed for instrumentation purposes, how it can be extended to handle even more architectures and how its performance compares to tools with analogous purposes such as QEMU, Valgrind, Pin and angr. We also study two often overlooked features of C/C++ programs: variadic functions and the RELRO link-time protection mechanism. We propose HexVASAN, a sanitizer for variadic functions to ensure that the number and type of arguments used by the variadic function match those passed by the caller, and leakless, an exploitation technique to bypass the RELRO protection in its several forms.

Despite the growing popularity of interpreted or byte-compiled languages, C/C++ and other languages targeting native code are still dominantly used for system programming. Programs compiled to native code present a set of challenges compared to alternatives. In particular, in this work we focus on how they can be efficiently analyzed, how existing security measures (known as "binary hardening techniques") perform, and how new ones can be introduced to secure features that have received little attention. We propose rev.ng a binary analysis framework based on QEMU, a popular dynamic binary translator and emulator, and LLVM, a mature and flexible compiler framework. rev.ng can easily handle a large number of architectures and features a set of analyses to recover basic blocks locations, function boundaries and prototypes in an architecture- and ABI-independent way. rev.ng can be used for instrumentation, debugging, decompilation, retrofitting of security features and many more purposes. Our prototype encompasses about 17 kSLOC of C++ code and has been publicly released under a Free Software license. The core component of rev.ng is revamb: a static binary translator which can accurately identify all the basic blocks, and, in particular, the targets of indirect jumps for switch statements. Along this work, we will make heavy use of analysis techniques popular in the compiler literature, such as Monotone Frameworks, to recover an accurate control-flow graph, identify function boundaries and the number and location of function arguments and return values. We will also discuss how rev.ng can handle native dynamic libraries, how it can be easily employed for instrumentation purposes, how it can be extended to handle even more architectures and how its performance compares to tools with analogous purposes such as QEMU, Valgrind, Pin and angr. We also study two often overlooked features of C/C++ programs: variadic functions and the RELRO link-time protection mechanism. We propose HexVASAN, a sanitizer for variadic functions to ensure that the number and type of arguments used by the variadic function match those passed by the caller, and leakless, an exploitation technique to bypass the RELRO protection in its several forms.