<u>**Enhancements over the conference version**</u>

In our earlier work, presented at EuroSys 2013, we presented some techniques essential for converting executables to the high-level intermediate representation (IR) of an existing compiler. Our prevision work segmented the flat address space in an executable by converting physically addressed stack to an abstract stack and by promoting identified scalar variables to symbols. In this work, we have extended the above techniques in the following directions.

1. Improved Introduction: We have updated the introduction to reflect the increasing importance of binary analysis and rewriting for security purposes. Our previous introduction only compared the source-level and binary-level program analysis in general. There has been a rapid rise in cyberattacks and the updated introduction effectively demonstrates the ability of our framework in solving some of the critical challenges involved in preventing such attacks. The updated introduction also reflects the applicability of our framework in mobile domain.

2. Scope: In our earlier work, we proposed a mechanism to only promote scalar memory locations to symbols. In this work, we extend our technique to promote identified array variables to symbols in the intermediate representation. Promotion of stack array accesses to symbolic accesses enhances the precision of several subsequent analyses in IR such as alias analysis. We demonstrate that our techniques result in the promotion of 30% of identified arrays to symbols. (Section 5.3)

3. Precision: In this work, we propose three techniques to improve the precision of scalar symbol promotion. These techniques were driven by our observation of promotion decision bottlenecks in our benchmarks. These techniques include - enhancement of symbol promotion in presence of variable argument procedures, enhancement in presence of several executable specific artifacts such as indirect control transfer instructions, and modeling aliased memory locations as structure types. Our techniques collectively improve the precision of symbol promotion by 8.5% on average on our benchmarks. (Section 6 and Section 8.3)

4. Analysis: The promotion of memory locations to symbols essentially recovers the data-flow graph corresponding to the original program by enhancing the density of data-flow graph in a program. We define a new metric to capture this density of the data-flow graph. Our results demonstrate that symbol promotion results in an improved value of this density metric, signifying an enhancement in the precision of subsequent data-flow analysis. (Section 8.4)

5. Extension to other ISA: Our earlier work implicitly assumed the presence of 32 bit x86 ISA. In this work, we demonstrate that there is no inherent limitation in extended our work to other architectures. We present a discussion on how to extend and apply our symbol promotion and abstract stack mechanisms to x86-64 ISA and ARM ISA. (Section 7)