I managed to finish reading the paper. It is a very dense piece of work, there is a lot of detail there. I will only give you the highest-level comments, otherwise there would be too much to write. I will emphasize things which have confused me, since these probably could benefit most from improvements.

First of all, based on the title, I expected that you use (C++) types for building the call-graph; in fact, your analysis works on binaries and it does not use the C++ types, but it only uses register types. This is a fine decision, but it is not what expects; perhaps you could change the title to reflect this fact better - perhaps something like CFI binary instrumentation based on binary type information. It seems that you assume statically-linked binaries; if true, you should say it.

I think you have to describe the calling convention somewhere. I was confused by the fact that you analyze x86 code but using Itanium calling conventions.

You don't say anything about control-flow due to C++ exceptions.

Your work has two separate parts: the analysis of the binary, which is based on DynInst for parsing the binaries (BTW: you mention DynInst several times in the text before you say what it is), and enforcement (also based on DynInst for rewriting the binary). For enforcement you have several possible policies, with various degrees of precision/cost trade-offs. I think that the paper should be organized more clearly around these two phases.

In some places the description could be more precise and concise. For example, you should really write the dataflow equations rather than describing in words what is happening.

Also, your description in words of the algorithm for allocating labels for the "fast mode" seems incorrect - based on the order of traversal it may not compute the correct transitive closure of the graph of equivalent labels.

You are using just SPEC INT, not all of SPEC CPU. I don't know how many of these are C++ program, you should indicate that. That's the interesting case for your analysis.

Some terms could probably be improved: "trashed register", "clang miss" (probably better to use false positives and false negatives instead). You use the term "chosen implementation" several times, but I don't know what it means.

BTW: do you have any false positives in the analysis, missing legal edges, which would prevent legal programs for running under some circumstances? It is not clear from the text.

You claim that some attacks are prevented, have you checked by attacking the target and seeing that CFI prevents the hijack?

Tables 2 & 3 seem to be identical. In section VII you have several enumerations with a single item.

This is a very impressive piece of work, thank you for sharing.

Mihai