I briefly write my ideas here for variable types reconstruction from binary files w.r.t. to the path we should go.

First, the Hex-Rays tools seeams to be the best fit to choose. It is plug-in based, it is good documented but it is not that accurate as TIE (this based on BAP). Plug-ins can be easily created.

Second, TIE is most accurate and better than X-Force tool. Plug-ins for analysis can be built on top. The plug-ins can be written in OCaml, Python and other languages via BAP Server (BAP as a service). We still need to find out if the analysis contained in the TIE paper is really available in the current BAP implementation.

Third, X-Force - source code is not available so I think this is a dead end.

Fourth, Dyninst variable type reconstruction can be obtained. We do not know how accurate it is and under which circumstances it will work. Lets keep this for now as last choice.

In the end as Victor did in his paper “A touch call: …” (he over approximates callee/caller pairs by allowing more callees per caller but still this is lower thant the previous relation which was 1 to N (is the sum of all called in the binary)) we need to further reduce the callee/caller set based on variable types so I think it is not avoidable that we also use a similar style of approximation as he did. We can experiment in my opinion with Hex-Rays and TIE to see for which we get the best result.

A possible approximation could be on the variable types as for e.g., if var x is variable of the caller function e.g., foo(x) and at the callee site we detected lets say 2 possible target locations where first, variable x is of type double and second, variable a is of type float. Since bot this type are super types of int we could allow both types.

In case original x is of type double (type was recuperated by Hex-rays, Tie, etc.) than in case we have the same callee targets for foo(x) but now the types of the the var x at the target are int and float we can allow only the target with type float. This rules one possible target canditate thus, the caller/callee set gets reduced.

The ideas are similar to Victor paper.

see: https://en.wikipedia.org/wiki/C_data_types

long double > double > float > unsigned long long int > signed long long int > unsigned long int > signed long int > unsigned int > unsigned short int > signed short int > unsigned char > signed char > char.

Note: a > b means in this context that a is super type of b.

Strategy: Always allow as target the types (at the calle) if these are subtypes of a supper type which the caller provides

Also for void and void * we have to thing about a schema on how to reduce the caller / callee pairs.

Remarks: We can used DIA SDK (see X-Force paper) to get the ground truth about the binary. Probably we don’t need the LLVM pass from Victor. I asked Vitor about that code and he can give it to me in June after a paper submission deadline.

Text copied from the X-force paper: “To acquire the ground truth, we compile the programs with the option of generating debugging symbols as PDB files, and use DIA SDK to read the type information from the PDB files”