Tool selection for variable type reconstruction from binary files

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

Remarks: Red text color means that it is not available and must be programmed from scratch (so rather not go this direction). 2. (canditate) – means should be taken seriously into consideration

First, the Hex-Rays tools seems 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 is based on BAP but not open source avaialbe). Plug-ins can be easily created. (canditate)

1. The 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. Comments from D. Brumley: Unfortunately, no. That code base is long gone missing, and wouldn't compile with the latest bap in any case. BAP does have the ability to read in header files, and should be able to do type propagation. I am not able to help with the specifics, but there is fairly extensive documentation in the source base.

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

3. The Dyninst tool 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. Re: Dyninst currently reconstructs data types based on DWARF debugging information and does not try to infer data types when debugging information is not present. So debug information is needed and without we can not infer the data types of the variables. Probably use first use DIA SDK.

4. Binaries are usually stripped from symbol information. We can use the external tool such as Unstrip [1] that restore symbol information to a stripped binary. Discussed in Marlin [2] paper. After this step we can use DIA SDK (probably this is only usable in Windows). (canditate).

5. The angr tool, see [3] "(State of) The Art of War: Offensive Techniques in Binary Analysis" (canditate). currently under investigation. Reply: Currently, angr doesn't include type recovery. Much of the underlying requirements are there, though, and it's something that we're slowly working towards and will definitely happen in the future.

6. PIN tool - TODO

7. IDA Pro (200% less precise than TIE) - TODO

I think a good path to go is to make the variable inference feature of our tool as a plug-in and use each tools one after each other by comparing them w.r.t. their precision.

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"

References:
[1] Paradyn Project. (2011). UNSTRIP [Online]. Available: http:// paradyn.org/html/tools/unstrip.html
[2] A. Gupta et al. Marlin: Mitigating Code Reuse Attacks Using Code Randomization, 326 IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 12, NO. 3, MAY/JUNE 2015
[3] G. Vigna et al. "(State of) The Art of War: Offensive Techniques in Binary Analysis"
[4] Caballero et al. "Type inference on Executables, ACM Journal" 2016. - **this is our most important (comprehensive) reference.**

Here is the latest Dyninst release website:

http://www.paradyn.org/html/manuals.html