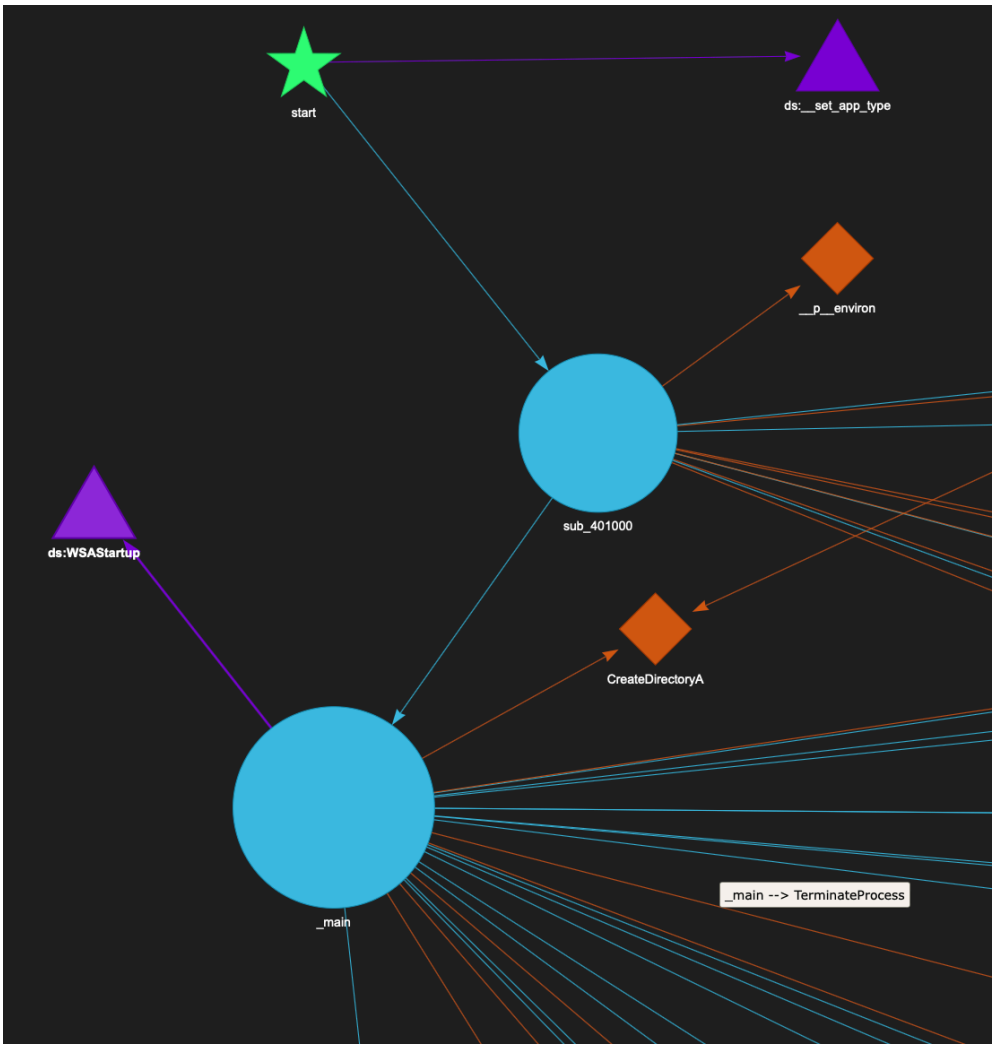
Matthew Crepeau
4/30/2020

Script to Generate Whole-Program Overview Graphs in IDA Pro

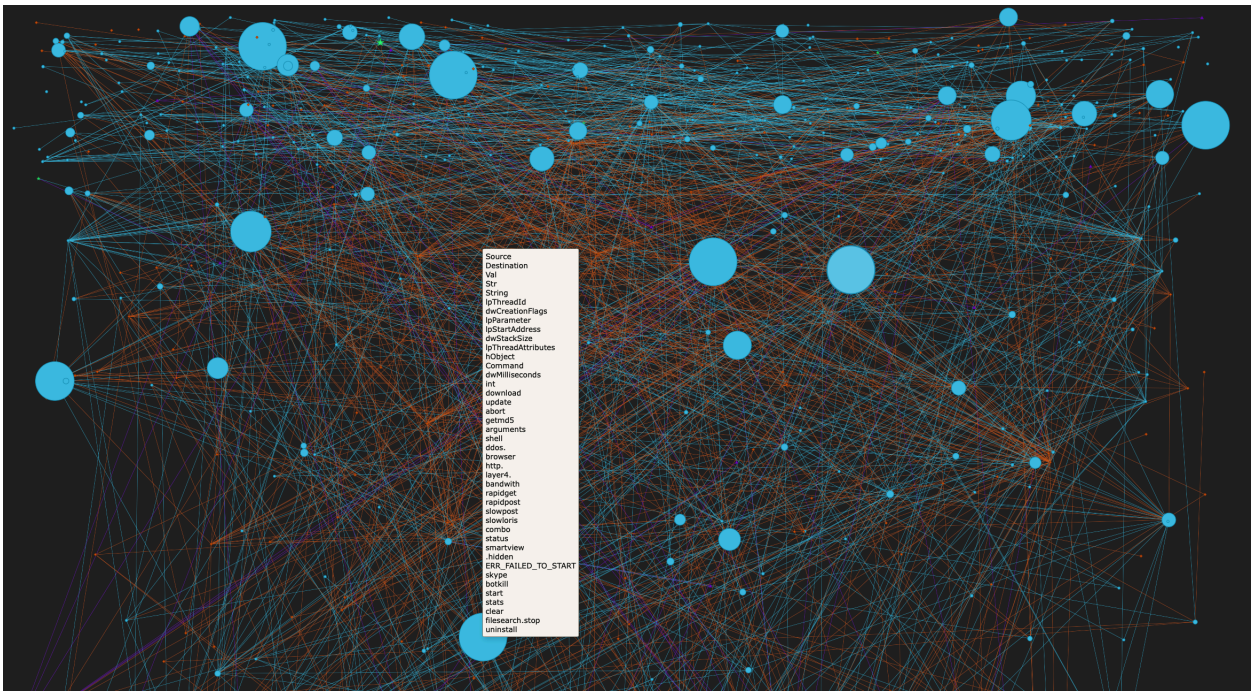
Description

For the second part of this project, I've created an IDA Python script that helps the user identify potential functions of interest. The script scans all of the IDA-identified functions, extracts string references, IDA comments, function comments, and function calls and then generates a graph using the pyvis visualization library. The graph scales each node based on the number of identified string contents, and links nodes together with edges based on function calls. Entry points are drawn on the graph as green stars. Calls to imported library functions are identified with orange edges/diamonds, calls to functions loaded in the data segment are identified with purple edges/triangles, and calls to regular functions are blue edges/circles. When the user hovers over a node, a list of unique strings found in that function are displayed. Hovering over an edge will display a popup telling you which function is calling which.

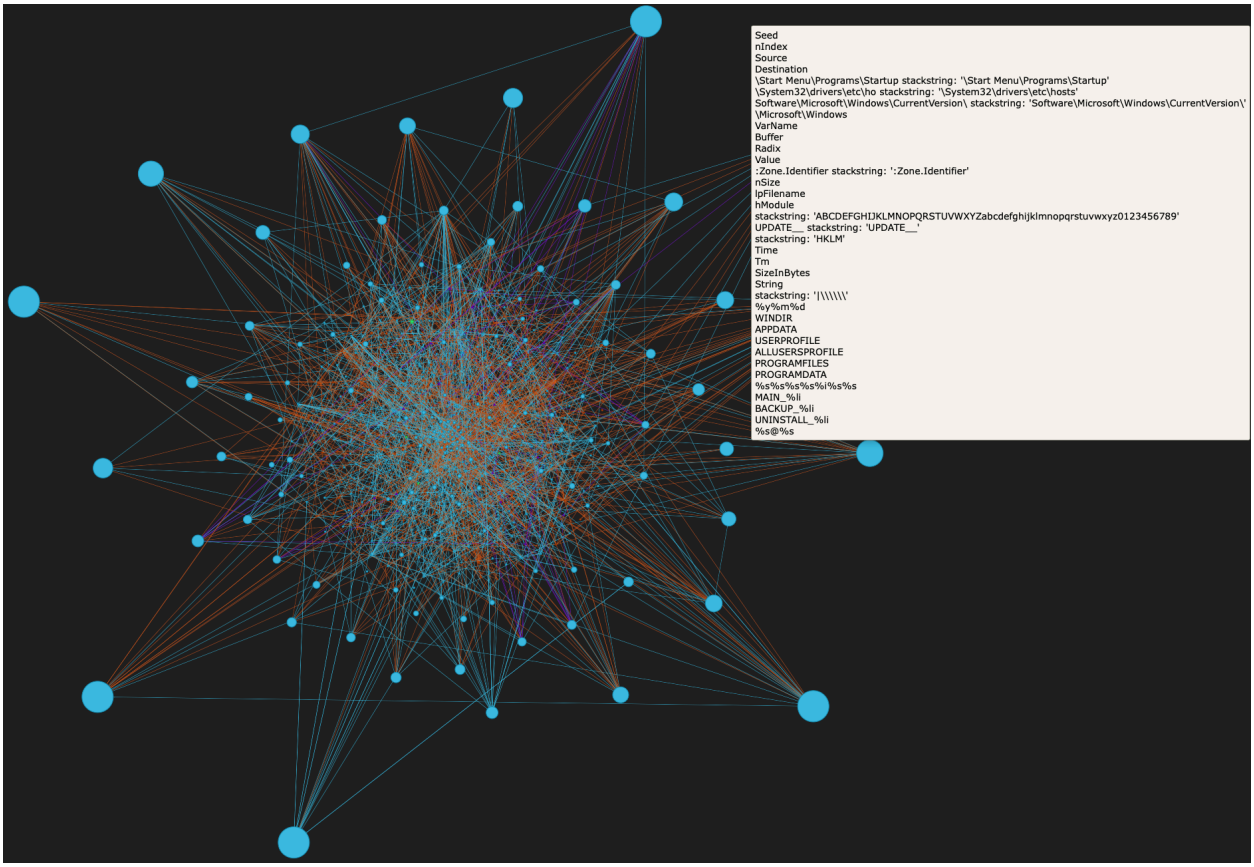


Graph Layouts

The program can be run in two different modes: “hierarchical” and “physics”. The hierarchical model assigns each function a “level” based on when that function is called in the code. The script will start at the entry point and any other identified exports, and then recursively mark functions called with increasing levels. While the order of levels is not necessarily consistent with the real-time calling order of the functions, this layout spreads them out vertically (based on level) to allow the analyst to better trace function calls and program flow, and horizontally (in a random, uniform distribution) to add space between the nodes. Functions that are never directly called (e.g. thread entry points) are at the top of the graph.



The physics model scales the “mass” of each node based on its size. The model is an inverted gravity model, so nodes with more mass exhibit more repulsion on other nodes. This has the effect of pushing the most “interesting” functions to the outer edges of the screen. The user can then zoom in and out and drag nodes around as they examine the functions.



As you can see from the screenshots thus far, the tool provides a useful snapshot of the strings in each function. An analyst can, for example, look at the function above and say (with some amount of confidence) that they think it is responsible for editing the Windows registry and adding something to the startup folder. They can then look the function up in IDA and rename it or add any additional comments.

Usage

Using the script is very straightforward. The only prerequisite is pyvis, which can be installed using the following command:

```
pip install pyvis
```

Once pyvis is installed, open IDA and click File -> Script file, or press Alt+F7. Select the script generateOverviewGraph.py and run it. The script will prompt you to enter the layout mode and an output filename (be sure to include the .hmtl extension). The script will then run and save the output file in your IDA working directory. Open it with a regular web browser and enjoy!

Pro tip: run the FLARE/stackstrings plugin before using this one so that stack strings are included in the results!

Limitations

I struggled greatly with the decision to use pyvis for this project because it has some serious limitations. In the end, however, it gave consistent output and allowed for the level of interactivity that I was looking for. It was also very simple to install and use. Here are the biggest limitations:

- Pyvis no longer appears to be under active development
- An internet connection is required for the HTML output to load
    - It relies on scripts for the vis.js components to work that are downloaded on-the-fly and had no option to allow for offline scripting. I know this is kind of sketchy, and not ideal for a reversing environment where machines might not always be internet-connected.
    - An improvement would be to modify pyvis to use static javascript files
- Pyvis lacks customization options that leave the output cluttered
    - As you can see from the screenshots above, there are a lot of nodes and even more edges. This tends to make the graph very complicated and sometimes overwhelming. I would have loved an option to hide edges unless a node was clicked, but alas, no such option was present.
- Nodes are randomly placed on the canvas, making it hard to keep track of them between runs of the program
    - In the future I'd like to improve the way nodes are placed so it is more deterministic. Another great improvement would be some sort of live-view that updates the graph as you rename functions and change things in IDA.

In light of these limitations, if I were to improve on this program in the future, I would probably find a way to use the Python *networkx* library. I did attempt using this library, but installing the required components (namely matplotlib) proved to be unnecessarily difficult on the VM for some reason. My goal from the beginning was to create an interactive way for reverse engineers to explore the whole-program layout of binaries, and I think pyvis helped provide a good proof-of-concept. With refinement I believe this script could become very useful.

References

https://www.hex-rays.com/products/ida/support/idapython_docs/
https://reverseengineering.stackexchange.com/questions/11024/getting-user-comments-with-idapython-api-user-cmts
https://programtalk.com/python-examples/idc.GetCommentEx/
https://reverseengineering.stackexchange.com/questions/23525/get-input-with-idapython
https://pyvis.readthedocs.io/en/latest/index.html
https://reverseengineering.stackexchange.com/questions/13627/ida-python-list-all-imported-functions
https://reverseengineering.stackexchange.com/questions/19607/idapython-get-call-destination-for-register-operand

I used the above sites mainly as references for how to use certain IDA Python API calls. The logic used in my code was all developed on my own.