Windows workshop 2010

Understanding Software Dependencies in Windows

Roland Yap

School of Computing

National University of Singapore

Singapore

ryap@comp.nus.edu.sg

Motivation

- Software is complex with a ***ecosystem*** of dependencies
  - Installation can cause other S/W to fail (e.g overwriting old library versions)
  - Uninstallation can cause S/W to fail (e.g removing critical shared libraries)
- Interactions and dependencies between software are difficult to understand
- We explore **visualization** as a tool for understanding software dependencies
  - See ***Y. Wu, R.H.C. Yap and R. Ramnath, Comprehending Module Dependencies and Sharing, ICSE 2010, to appear***

Note: Dave's talk – what is an application?

Examples of dependencies (1)

- Binaries used by **notepad**
  - c:\windows\apppatch\acgenral.dll
  - c:\windows\system32\avgrsstx.dll
  - c:\windows\system32\imm32.dll
  - c:\windows\system32\lpk.dll
  - c:\windows\system32\msacm32.dll
  - c:\windows\system32\msctf.dll
  - c:\windows\system32\msctfime.ime
  - c:\windows\system32\shimeng.dll
  - c:\windows\system32\usp10.dll
  - c:\windows\system32\uxtheme.dll
  - c:\windows\system32\winmm.dll
  - c:\windows\system32\winspool.drv
  - c:\windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll

Examples of dependencies (2)

- Simple boot (only Windows installed)
  - DLLs: 154
  - EXEs: 10
  - Drivers: 1
  - Ime: 1
- Typical boot (Windows + applications)
  - DLLs: 274
  - EXEs: 15
  - Telephony/Modem: 6
  - Drivers: 3
  - ActiveX: 2
  - Ime: 1

What are binaries?

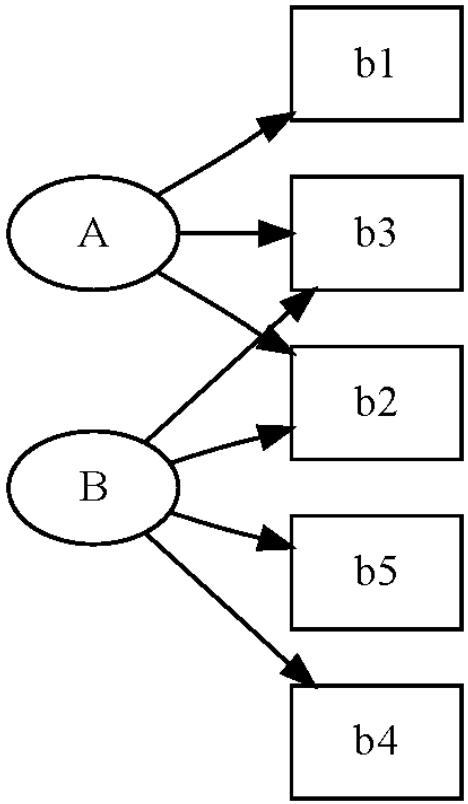
- EXE (executables) are considered binaries
- Other binary types are DLL (dynamic linked libraries), OCX (ActiveX controls), SYS (drivers) and CPL (control panel applets), …

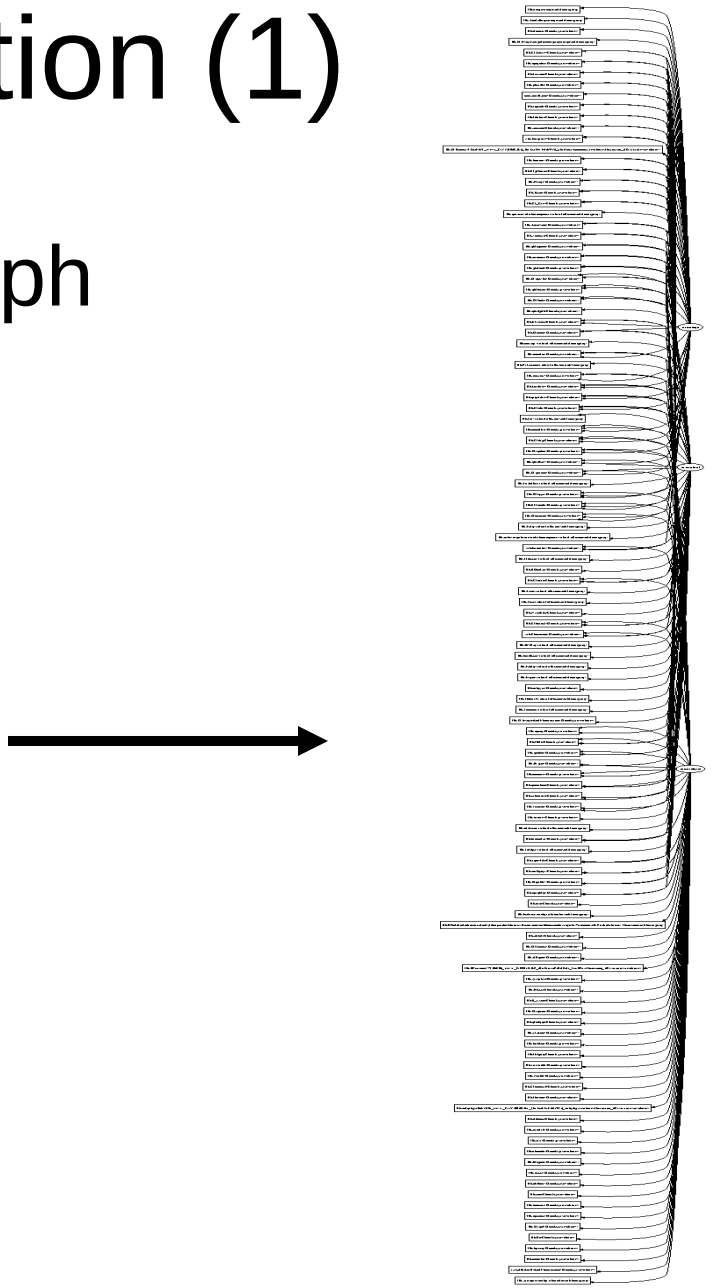
Visualization Objectives

- Visualizing binaries used by a particular executable
- Visualize any commonalities between binaries
- Visualize dependencies between binaries

Visualization (1)

- Basic dependency graph
- Graph is too dense





Revisiting processes (1)

- Process creation: Broken into many native calls.
- Very different from UNIX's fork+execve
- The user space Win32 API CreateProcess()
  1. Open the EXE file (ZwCreateFile)
  2. Create process object (ZwCreateProcess)
  3. Create thread object (ZwCreateThread)
  4. Notify Windows subsystem (csrss)
  5. Start the initial thread (ZwResumeThread)

Note: Recall Dave's talk – NT native system calls

Revisiting processes (2)

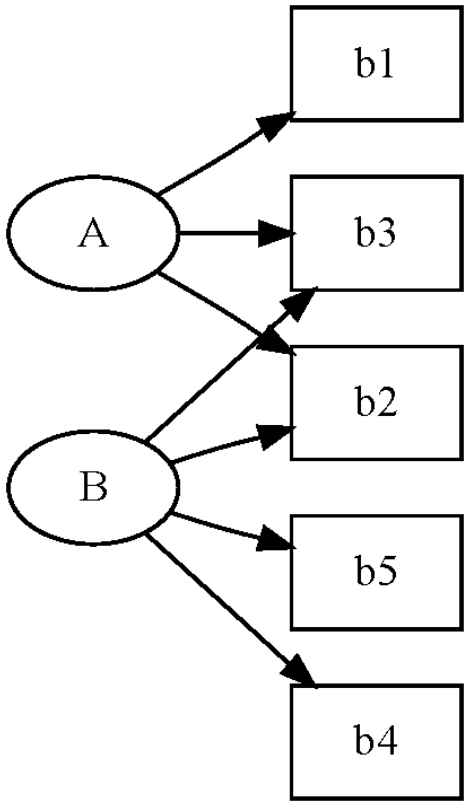
- Binary loading: broken into many native calls
- Similar to UNIX: `dlopen` = `open`+`mmap`+…
- The user space Win32 API LoadLibrary()
  1. Open the binary (ZwCreateFile)
  2. Create a Section object (ZwCreateSection)
  3. Map the binary to VM (ZwMapViewOfSection)
  4. Dynamic linking, relocation…

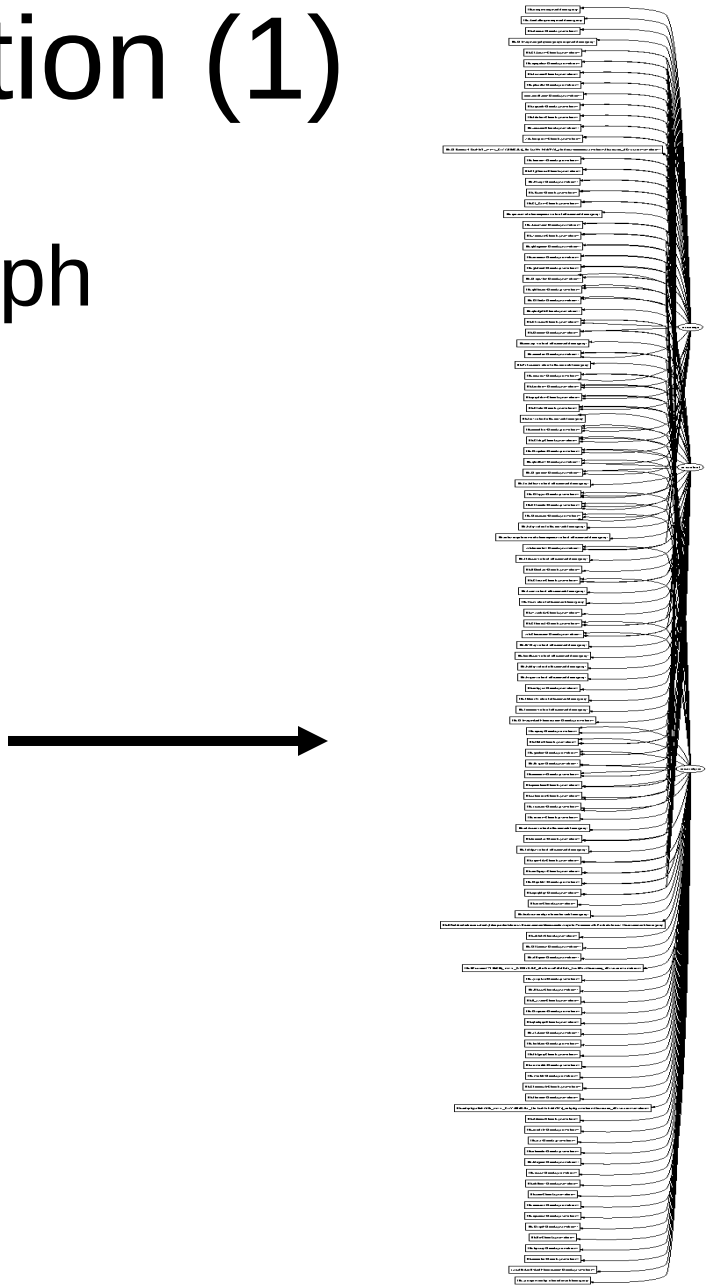
Binary Dependency Visualization

- Two types of nodes: EXE, DLL + etc
- Three types of directed edges
  1. EXE *X* launches another EXE *Y*
  2. EXE *X* load a DLL *Y*
  3. A function in binary *X* calls a function in binary *Y*
- How are binaries shared among programs?
  - **EXE Dependency Graph**
  - Only Type 1 and 2 edge
  - Group DLLs by loader
- How binaries interact?
  - **DLL Dependency Graph**
  - Only Type 2 and 3 edge
  - Group DLLs manually by functionality or software vendor

Visualization (1)

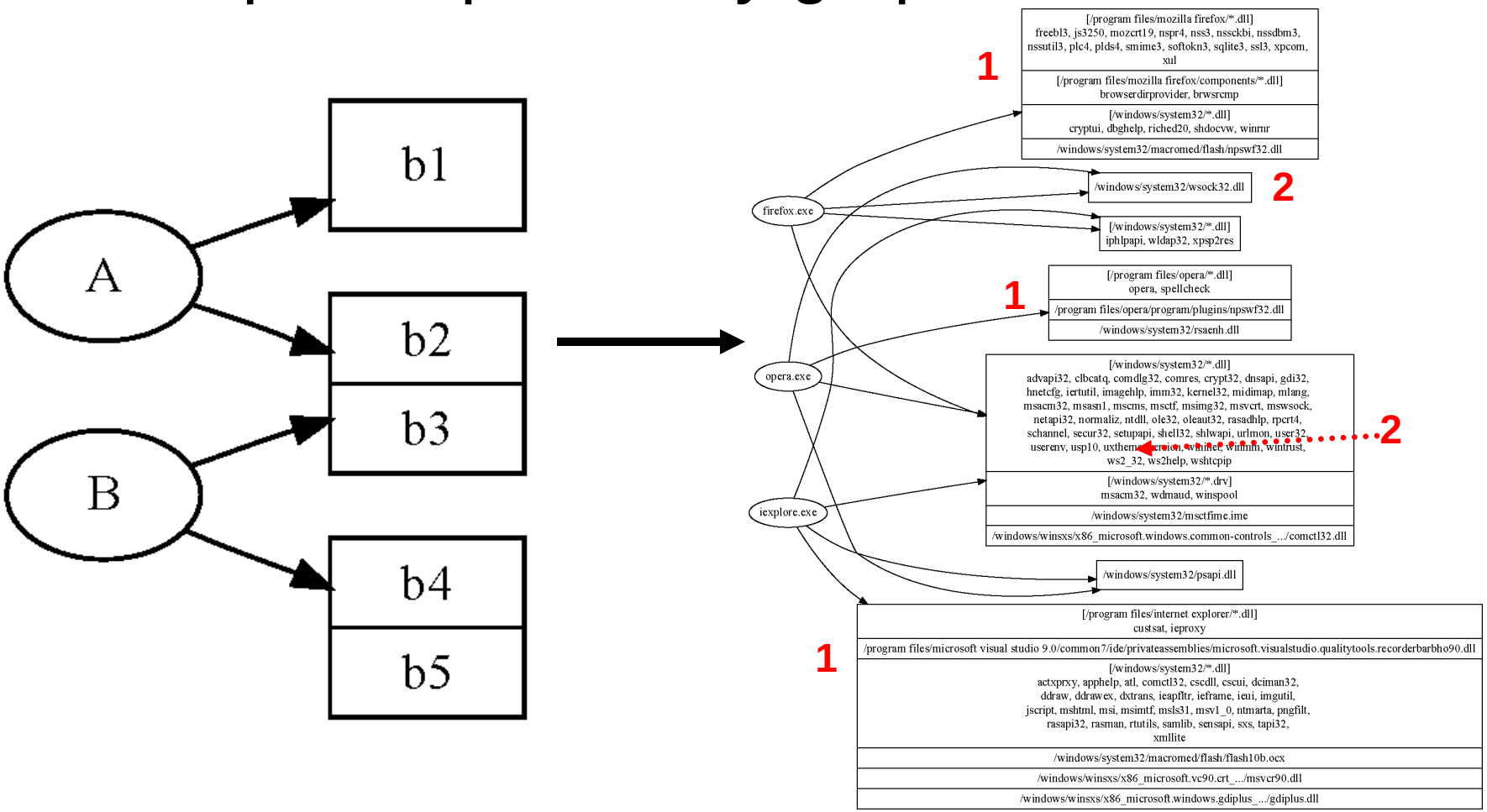
- Basic dependency graph
- Graph is too dense





A more usable Visualization: EXE Dependency Graph

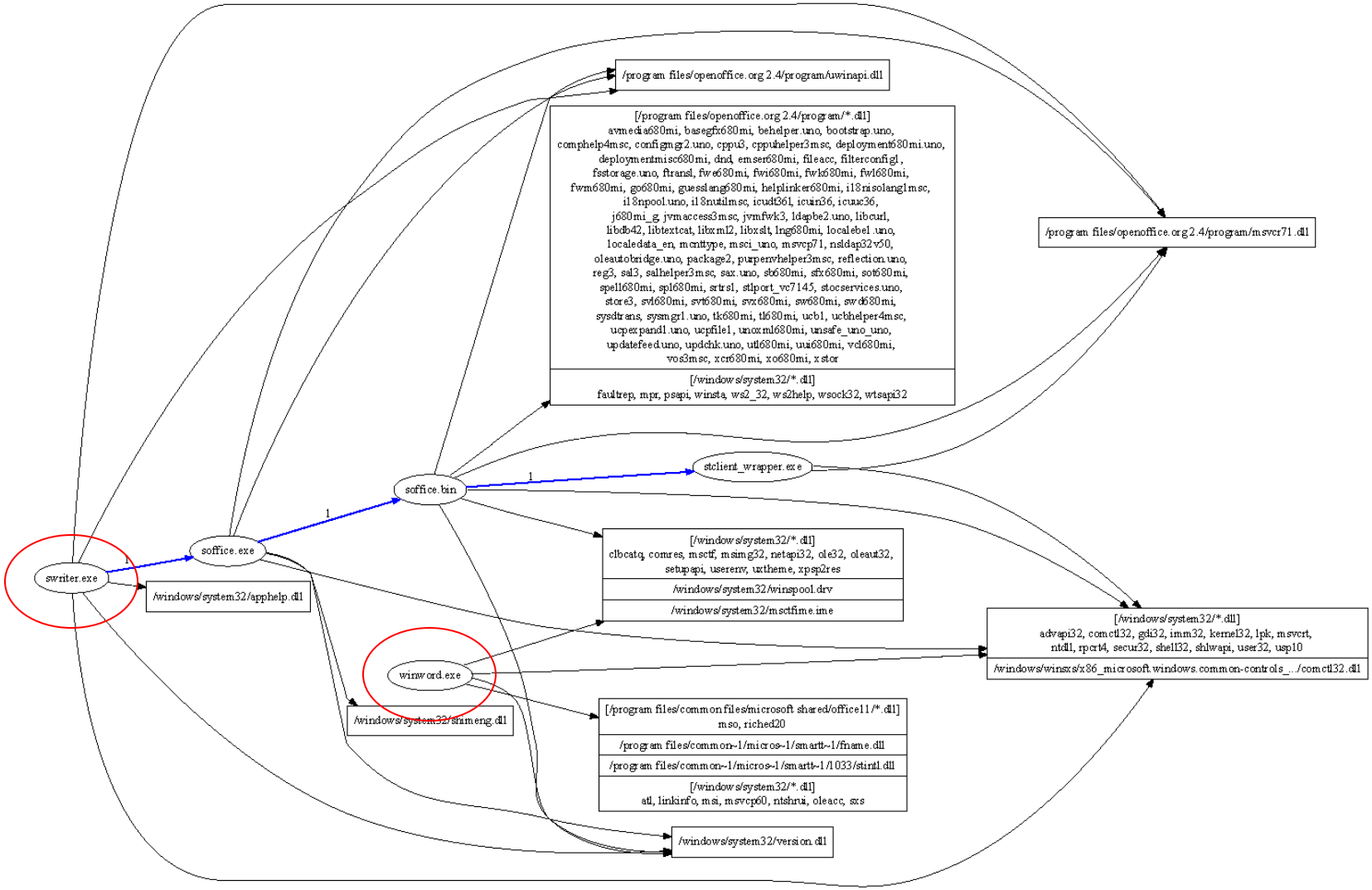
- Grouped dependency graph



Collecting binary dependency information

- EXE *X* launches another EXE *Y*
  - Use the kernel `PsSetCreateProcessNotifyRoutine()` API
- EXE *X* load a DLL *Y*
  - Use the kernel `PsSetLoadImageNotifyRoutine()` API
- A function in binary *X* calls a function in binary *Y*
  - Instrument code execution

Comparing Microsoft Word and Open Office Writer



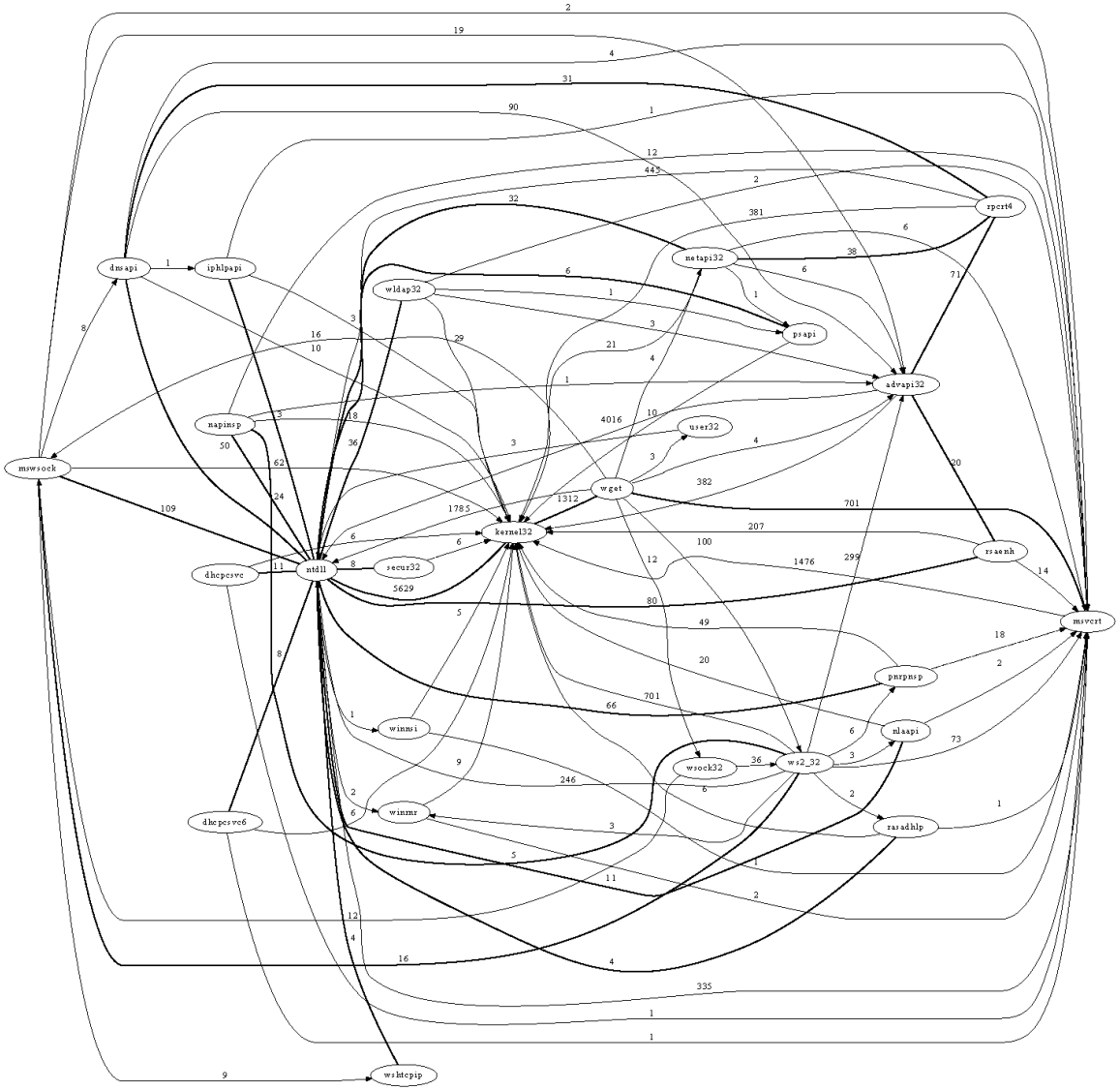
DLL Dependency Graph: actual binary usage

- Some definitions:
  - An EXE-DLL dependency in a DLL Dependency Graph is when there is has a control transfer from code in executable x to code in DLL y. We say that x has an EXE-DLL dependency on y.
  - A DLL-DLL dependency in a DLL Dependency Graph is when there is has a control transfer from code in DLL x to code in DLL y. We say that x has a DLL-DLL dependency on y

Visualizing binaries executed

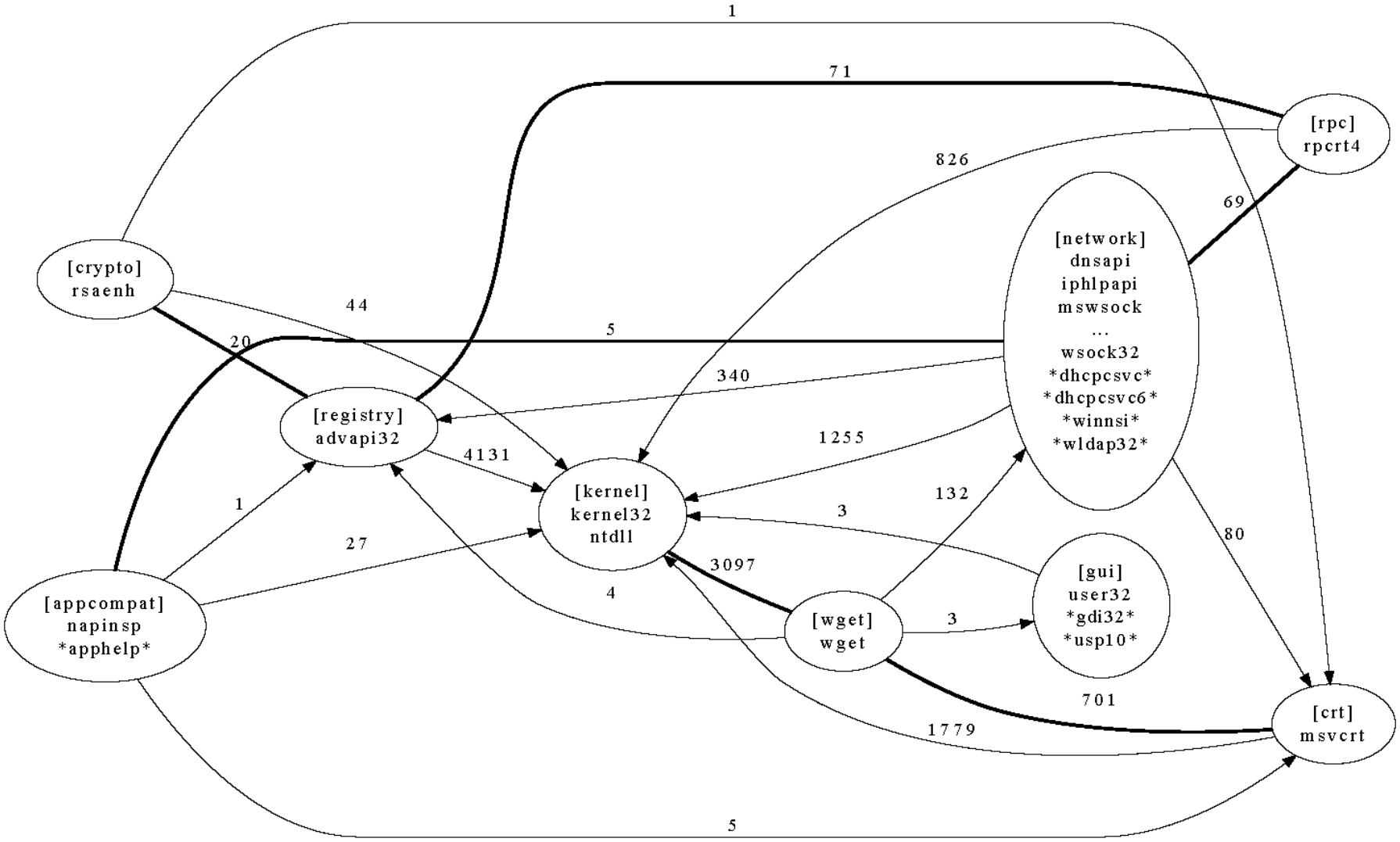
- Call graph is large.
- Group functions to images => DLL dependency graph.
- DLL dependency graph is still large.
- Group DLLs by properties:
  - By functionality: graphics, audio, network…
  - By vendor: microsoft, adobe…
  - By path: C:\windows\system32*.dll, D:\vmware*.dll…

Grouping by functionality
wget: DLL dependency without grouping

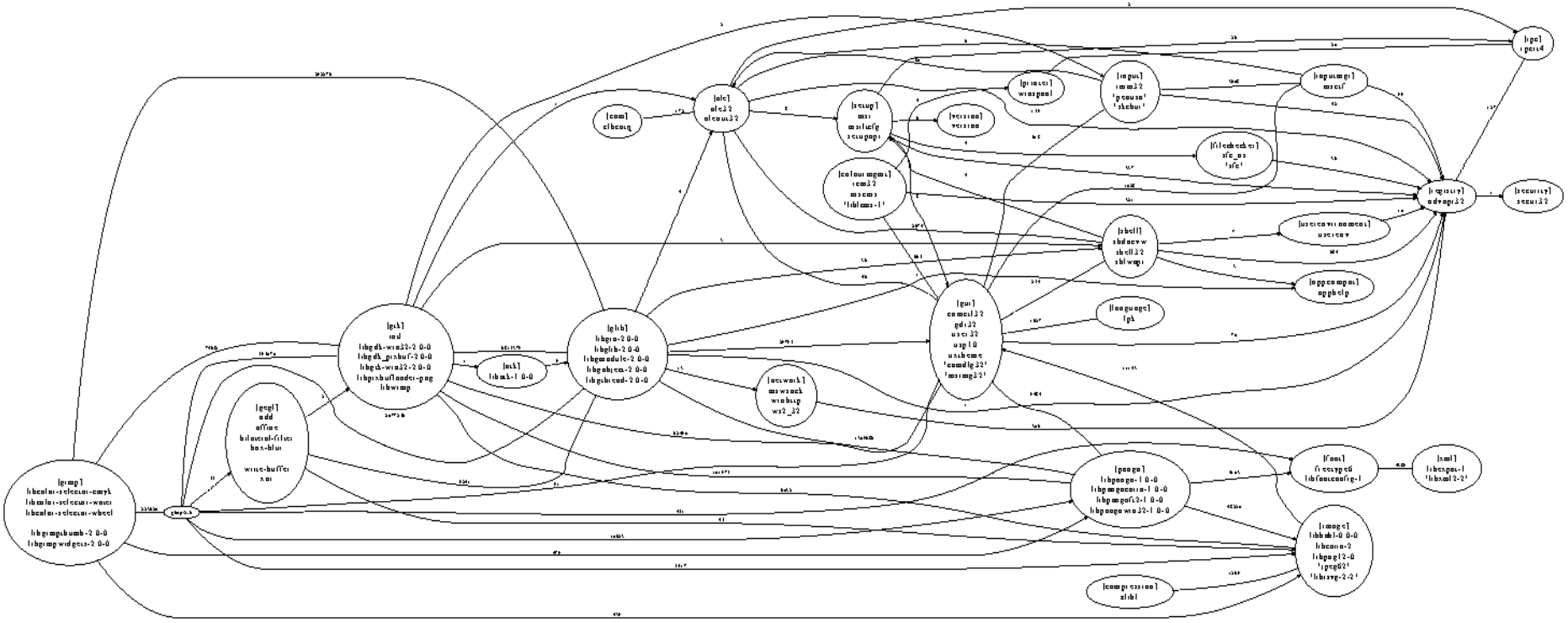


Grouping by functionality

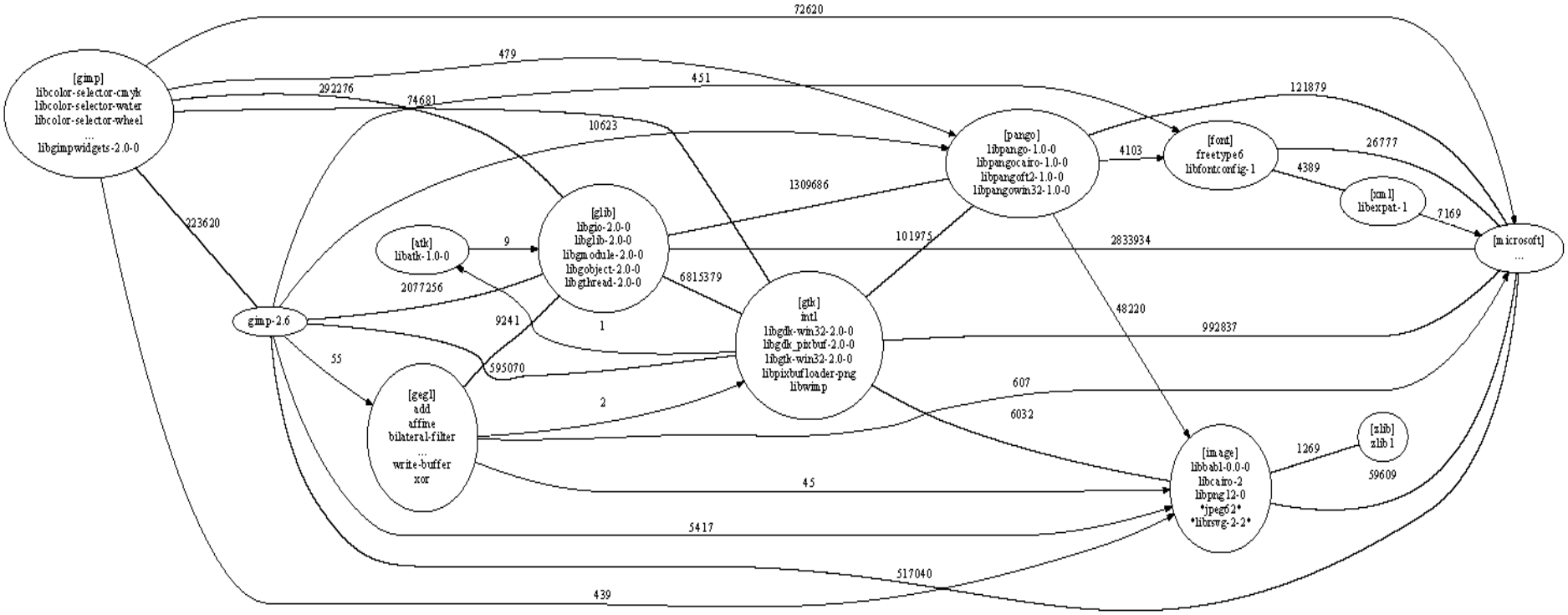
wget: DLL dependency with grouping



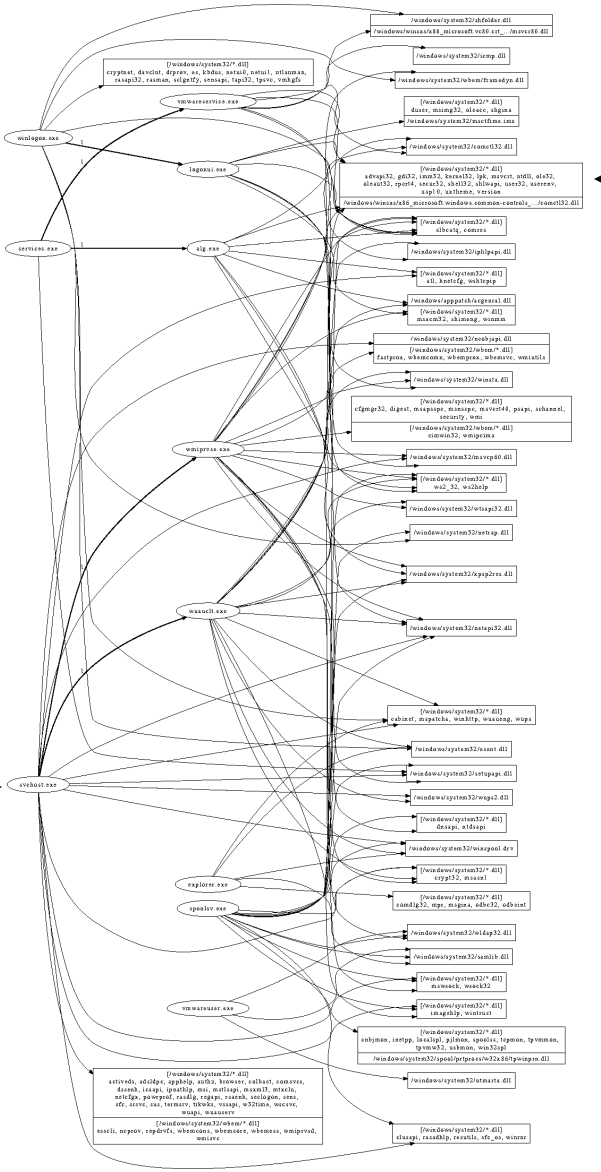
Examples of grouping
By functionality (GIMP)



Examples of grouping
By software vendor (GIMP)



Boot trace


Most common dlls
e.g kernel32, gdi32
Svchost using
a lot of dlls

Conclusion

- Actual software dependencies quite complex in Windows
- 2 effective visualizations:
  - EXE Dependency Graphs: commonality between binaries
  - DLL Dependency Graphs: actual binary usage