Analysing Kernel/Modules Interface from binaries

Linux Plumbers, Lisbon, Portugal
Sept 10, 2019.

Dodji Seketeli
dodji@redhat.com
Why?

- Kernel distribution maintenance perspective
  - Usually in LTS versions
- We want to see changes in the interface
  - To review them
  - To automatically alert us on some unacceptable changes
- Looking at binaries directly
  - Catches the impact of what happens to the source code
Be useful to maintainers

$ abidiff binary-v0.o binary-v1.o
Functions changes summary: 0 Removed, 1 Changed, 0 Added function
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

1 function with some indirect sub-type change:

[C]'function int function0(type*, int)' at binary-v1.c:8:1 has some indirect sub-type changes:
  return type changed:
    type name changed from 'int' to 'char'
    type size changed from 32 to 8 (in bits)
parameter 1 of type 'type*' has sub-type changes:
  in pointed to type 'struct type' at binary-v1.c:1:1:
    type size changed from 32 to 64 (in bits)
1 data member insertion:
  'char type::m1', at offset 32 (in bits) at binary-v1.c:4:1
parameter 2 of type 'int' was removed
More than just ELF symbols

• Read debug information and ELF
• Build in-memory internal representation (IR)
  – Graph of functions, variables, types (ABI artifacts)
  – Walk the IR to serialize it to disk, if needed.
• Facilities to compare two IRs
• Build another IR to represent the result of the comparison
• Walk and analyse the graph of changes
  – Decide if a given change is important or not
  – Emit a change report describing the graph of changes
New debug info formats coming

• CTF, BTF, anything more?
• Would be super useful to be able to use those new formats as well
• Interesting features that would be needed
  – Full type description for functions
    • Return and parameter types
  – De-duplication of type descriptions
    • Difficult to have at a full kernel level
Thanks you, gals’n guys!