The Usability of a Debugger Designed for Compartmentalized Systems

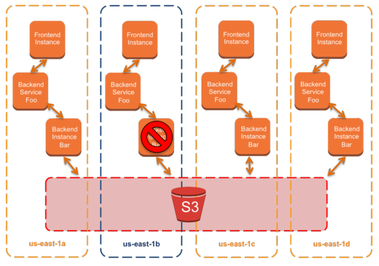
Junyong Zhao, Henry Zhu, Nik Sultana and Boon Thau Loo
Department of Computer and Information Science
School of Engineering and Applied Science, University of Pennsylvania

Introduction

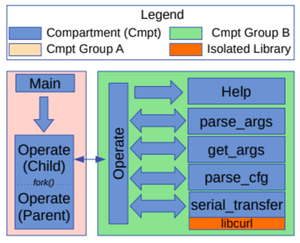
- Compartmentalization

  Breaking software into compartments is a well-known security technique for security, it will restrict the impact of errors in a part of the program.
- Debugging

  Compartmentalization makes debugging harder, as the developer might need to access all compartments with different privileges. Remote debugging is also required if the system is distributed on separate machines.

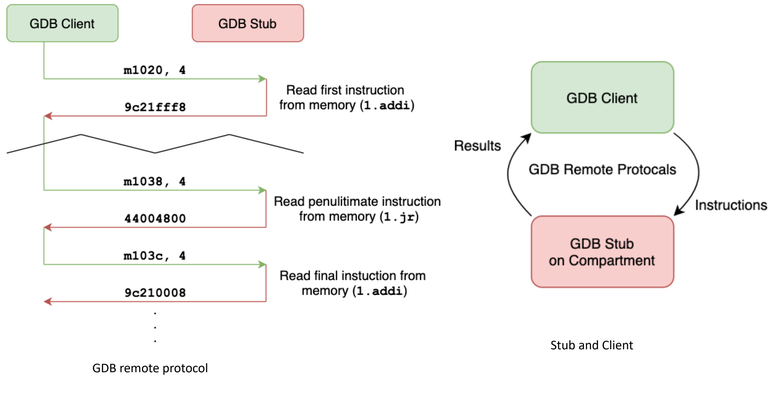


https://aws.amazon.com/cn/blogs/architecture/aws-and-compartmentalization/



A Case Study of Fine-Grained Software Compartmentalization using cURL
S.Carrasquillo, J.Zhao, H.Zhu, N.Sultana and B.T.Loo

Debugger - Back End



GDB remote protocol

Stub and Client

- Back-End Design

  We used the GDB's standard remote protocol at the backend so the client could talk remotely to the stub we developed. And the stub will run the debugging instructions on distributed compartments.

Motivation

- Research at Penn

  This is part of a research project at Penn focusing on making compartmentalization of software easier through a new tool-chain we are developing.
- Current Status

  There are no existing solutions for debugging compartmentalized software in our research. We want a debugger that is compartment-aware and does not make compromises.

Example

GDB client 1

Debugging Compartment 1

```
(gdb client 1): SetupDebugger (port=1234) at gdbstub_sys.c:544
(gdb client 1): 544     }
n
(gdb client 1): main (argc=1, argv=0x7fffaa467f28) at login_compartment.c:36
(gdb client 1): 36          Info info = { .name = INFO_VERIFY_NAME, .password = INFO_VERIFY_PASSWO
RD };
n
(gdb client 1): 38          struct extension_data arg = ext_verify_login_to_arg(&info);
n
(gdb client 1): 39          int logged_in = ext_int_from_arg(compart_call_fn(verify_login_ext, arg
));
p info
(gdb client 1): $1 = {name = 0x407b18 "admin", password = 0x407b18 "admin"}
c
(gdb client 1): Continuing.
switch 2
Switched to client 2
(gdb client 2): GNU gdb (Ubuntu 7.1                    i1.1
(gdb client 2): Copyright (C) 2016 Free Software Foundation, Inc.
(gdb client 2): License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
(gdb client 2): This is free software: you are free to change and redistribute it.
(gdb client 2): There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
(gdb client 2): and "show warranty" for details.
(gdb client 2): This GDB was configured as "x86_64-linux-gnu".
(gdb client 2): Type "show configuration" for configuration details.
(gdb client 2): For bug reporting instructions, please see:
(gdb client 2): <http://www.gnu.org/software/gdb/bugs/>.
(gdb client 2): Find the GDB manual and other documentation resources online at:
(gdb client 2): <http://www.gnu.org/software/gdb/documentation/>.
(gdb client 2): For help, type "help".
(gdb client 2): Type "apropos word" to search for commands related to "word".
(gdb client 2): SetupDebugger (port=1235) at gdbstub_sys.c:544
(gdb client 2): 544     }
n
(gdb client 2): other_preinit_fn () at login_interface.c:33
(gdb client 2): 33      }
n
(gdb client 2): compart_start (new_compartment_name=0x407b06 "login compartment")
(gdb client 2):     at compart.c:872
(gdb client 2): 872                 compart_as(comparts[i].name);
c
(gdb client 2): Continuing.

The client is no longer interactive, please switch to another client!
Launched client numbers:  [1, 2]
All client terminated, exiting...
```

switching

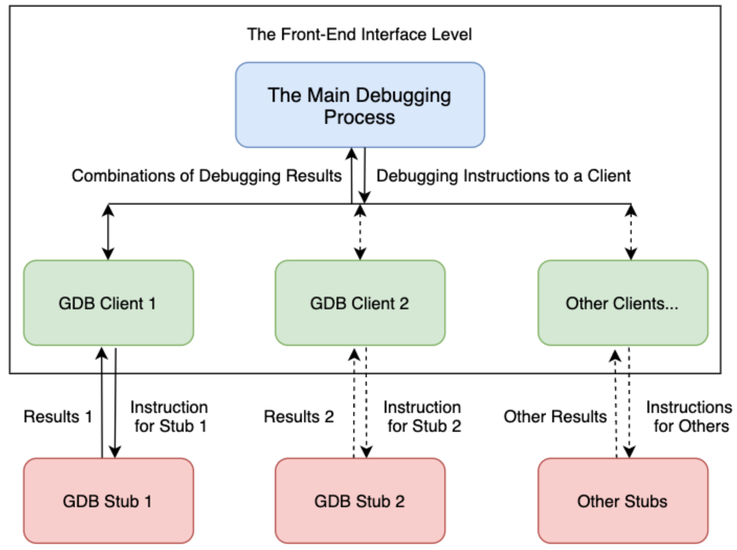
GDB client 2

Debugging Compartment 2

- An Example

  The example output of switching between two clients and running some basic debugging command could be captured from the above figure. As we could see, the interface is working and debugging two compartments.

Debugger - Front End



The Back-End Debugging Sessions

- Front-End Design

  Based on GNU's GDB, we used a Python interface to launch and manage all the GDB client corresponding to each compartment. Users could send instructions to clients to debug, and also switch client for debugging different compartments.
  There is only going to be one line of communication active, as the debugger will show nothing unless the user gives instructions (indicated by solid line).

Further References

For more informations about the debugger and libcompart, please see https://cobre.cis.upenn.edu/

Or contact me at junyong@seas.upenn.edu