

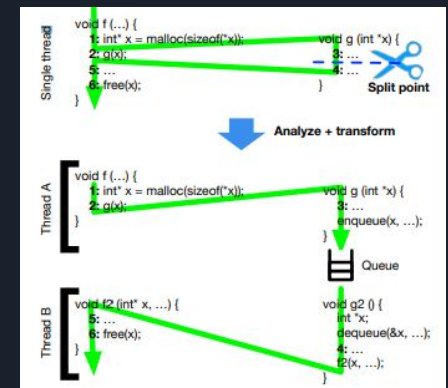
# Debugging strongly-compartmentalized distributed systems

Recording: [https://youtu.be/16\\_bMEAjLwI](https://youtu.be/16_bMEAjLwI)

Henry Zhu  
Nik Sultana  
Boon Thau Loo

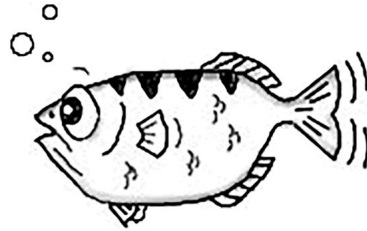
# Compartmentalization

- Limit access of information to entities
- In other words, break up a system into different parts with different levels of access.
- Hard to get right
  - Transfer of data between compartments might be corrupted
  - Information leak
  - Compartmentalized software might operate differently from original software
    - Might deal with privilege levels
  - What tools do I have available to debug these cases?



# Existing tools for debugging compartmentalized software

- GDB
  - Works at a process level
  - Privileges dictated by OS
  - Doesn't recognize compartments
- LLDB
  - LLVM/Clang debugger
  - Works similarly to GDB
- Two prototypes to solve problem
  - Custom Debugger
  - GDB Stub Debugger



**GDB**  
The GNU Project  
Debugger



# Custom Debugger

- Custom built debugger that recognizes compartments
- Has 12 commands -- of which 11 is a subset of GDB (read/write memory, breakpoints, variable printing)
- Parses DWARF - common debug information generated by compilers
- Ability to switch between compartments



# GDB Stub Debugger

- Implements a stub that follows the GDB protocol (communication back and forth from the GDB client, or the user)
- Features most of the commonly used GDB commands (breakpoints, reading/writing to variables, back-tracing)
- Rather than having the difference be at the process level, the GDB Stub debugger operates at the compartment level

# GDB Debugger

```
8 struct extension_data
   ext_verify_login_to_arg(Info* info)
9 {
10  struct extension_data result;
11  result.bufc = sizeof(*info);
12  memcpy(result.buf, info, sizeof(*info));
13  return result;
14 }
```

```
1 // lower privileged compartment
2 (gdb) b login_interface.c:49
3 Breakpoint 1 at 0x4011a5: file login_interface.c, line
   49.
```

```
4 (gdb) c
5 Continuing.
```

```
6 Breakpoint 1, ext_verify_login_to_arg (info=0
   x7ffef16fd11a0) at login_interface.c:49
7 49      memcpy(result.buf, info, sizeof(Info));
8 (gdb) n
9 (gdb) p *info
10 $2 = {name = 0x404e88 "admin", password = 0x404e88 "
   admin"}
```

```
11 (gdb) p (char*)result.buf
12 $3 = 0x7ffef16fd0f68 "\210N@"
13 (gdb) p result.bufc
14 $4 = 16
```

# Custom debugger

```
Prompt ("stop" to quit): connect login_compartment
```

```
Prompt ("stop" to quit): list variables
```

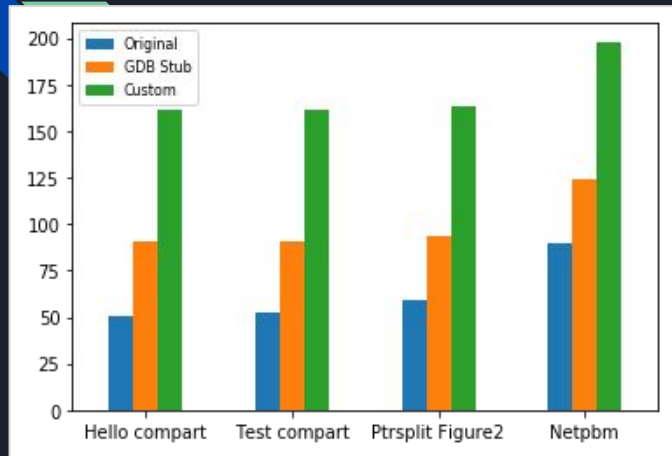
Num	File	Function	Line	Name
0	login_interface.c	ext_verify_login_to_arg	55	result

```
Prompt ("stop" to quit): r 0 0
```

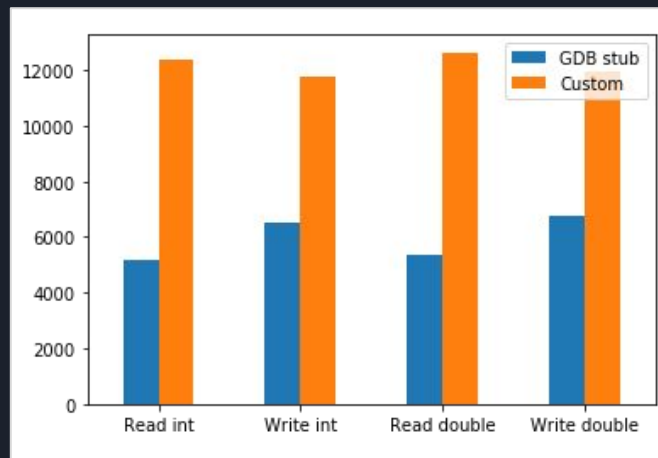
```
{
  "name": "extension_data",
  "address": "0x7fff32eeeb50",
  "val": [
    {
      "name": "bufc",
      "address": "0x7fff32eeeb50",
      "type": null,
      "size": 8,
      "val": 16
    },
    {
      "name": "buf",
      "address": "0x7fff32eeeb58",
      "type": null,
      "size": 512,
      "val": null
    }
  ]
}
```

```
8 struct extension_data
  ext_verify_login_to_arg(Info* info)
9 {
10 struct extension_data result;
11 result.bufc = sizeof(*info);
12 memcpy(result.buf, info, sizeof(*info));
13 return result;
14 }
```

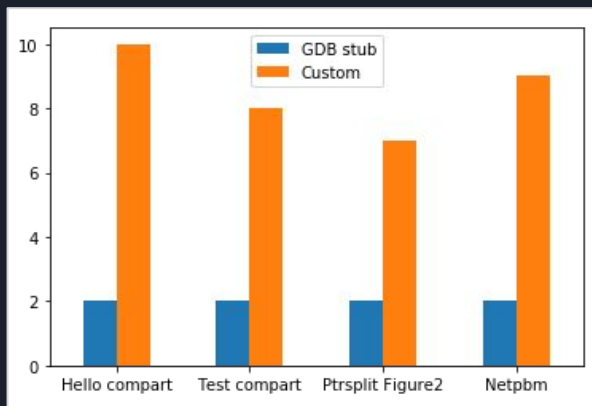
## Binary Size(kilobytes)



## Time overhead(CPU cycles)



## Source line changes







# Comparison

- Custom debugger is not as simple to use as GDB
  - Some actions require more commands for custom debugger than for GDB
  - Preprocessing step required to parse debug information before debugging the program
- GDB
  - Cannot switch between compartments
    - Each GDB stub only knows the compartment it is attached to
  - Cannot disconnect and reconnect to compartments
    - Background listener threads are needed to support reconnections