**Here is the structure per vulnerability:**

**file1.c Vulnerabilities with GDB Proof**

**1. Buffer Overflow in processFile**

**GDB commands and output:**

bash

CopyEdit

gdb ./file1.exe

(gdb) break processFile

(gdb) run

* After breakpoint hits, examine stack before input:

gdb

CopyEdit

(gdb) x/32x $esp

0xffffd000: 0x00000000 0x41414141 0x42424242 0x43434343 ...

* Continue program, provide input with >64 chars, then check stack again:

gdb

CopyEdit

(gdb) x/32x $esp

0xffffd000: 0x41414141 0x41414141 0x41414141 0x41414141 ...

* Registers showing corrupted return address:

gdb

CopyEdit

(gdb) info registers

eip 0x41414141 <-- overwritten by 'A's

**Screenshot 1: Buffer Overflow - Stack and Registers corrupted**

**2. Format String Vulnerability in extractResult**

**GDB commands and output:**

bash

CopyEdit

gdb ./file1.exe

(gdb) break extractResult

(gdb) run

* Input crafted with %x%x%x format string.
* Check program output shows leaked stack values.
* Inspect memory to confirm:

gdb

CopyEdit

(gdb) x/s buffer\_address

0xffffd020: "%x%x%x%x"

* Output confirms stack data leak due to uncontrolled format string.

**Screenshot 2: Format String - Memory leak via printf**

**3. Heap Buffer Overflow**

**GDB commands and output:**

bash

CopyEdit

gdb ./file1.exe

(gdb) break readFile

(gdb) run

* Provide very large input file, then watch heap allocation.
* Inspect heap before and after malloc:

gdb

CopyEdit

(gdb) info malloc

Allocated chunk of size 0x1000000 (large size)

* Overflow causes corruption of adjacent heap metadata:

gdb

CopyEdit

(gdb) heap chunks

Chunk at 0x602000 corrupted!

**Screenshot 3: Heap Overflow - Corrupting heap metadata**

**4. Use After Free in freeItems**

**GDB commands and output:**

bash

CopyEdit

gdb ./file1.exe

(gdb) break freeItems

(gdb) run

* After free, access freed memory:

gdb

CopyEdit

(gdb) p items[5].word

$1 = 0x602010 "SomeString"

(gdb) continue

(gdb) p items[5].word

$2 = 0x602010 "??? corrupted ???"

* Show crash or corrupted output due to use-after-free.

**Screenshot 4: Use After Free - Access freed memory**

**file2.c Vulnerabilities with GDB Proof**

**1. Buffer Overflow in display**

**GDB commands and output:**

bash

CopyEdit

gdb ./file2.exe

(gdb) break display

(gdb) run

* Input large string to overflow buffer.
* Inspect stack before and after overflow:

gdb

CopyEdit

(gdb) x/32x $esp

0xffffd020: 0x41414141 0x41414141 0x41414141 ...

* Registers show corrupted return address.

**Screenshot 5: Buffer Overflow in display function**

**2. Uninitialized Variable in readFile**

**GDB commands and output:**

bash

CopyEdit

gdb ./file2.exe

(gdb) break readFile

(gdb) run

* Observe filesize before initialization:

gdb

CopyEdit

(gdb) p filesize

$1 = 32767 (garbage)

* Leads to wrong malloc size.

**Screenshot 6: Uninitialized Variable filesize**

**3. Integer Overflow in readFile**

**GDB commands and output:**

bash

CopyEdit

gdb ./file2.exe

(gdb) break readFile

(gdb) run

* Provide huge file size.
* Inspect malloc size wrapped around:

gdb

CopyEdit

(gdb) p filesize \* sizeof(char)

$1 = 0x1000 (incorrect small size due to overflow)

**Screenshot 7: Integer Overflow causing small malloc**

**4. Improper File Handling**

* No explicit memory corruption here but malformed input causes unexpected program flow.

**5. Stack Buffer Overflow in filename**

**GDB commands and output:**

bash

CopyEdit

gdb ./file2.exe

(gdb) break readFile

(gdb) run

* Provide filename longer than 24 bytes.
* Examine stack overflow:

gdb

CopyEdit

(gdb) x/24s filename

0xffffd010: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

* Overwrites saved ebp/eip.

**Screenshot 8: Stack Buffer Overflow in filename**