



Booms with Ghidra

Introduction

The Bomb Lab is a reverse engineering challenge designed to enhance understanding of low-level program behavior, assembly analysis, and debugging techniques. The objective is to analyze a compiled binary and provide correct inputs for six sequential phases, along with an optional secret phase, without triggering the `explode_bomb()` function.

This write-up documents the methodology, analysis techniques, and reverse-engineering logic used to successfully defuse all phases using **Ghidra**.

Tools and Resources Used

Primary Tools

- **Ghidra** – Static analysis and decompilation
- **GDB** – Dynamic debugging and runtime verification
- **objdump** – Assembly inspection (supporting tool)

Methodology

The analysis followed a structured reverse-engineering workflow:

1. **Binary loading into Ghidra**
2. **Function identification** using symbol tree
3. **Decompilation review** (C-like pseudocode)
4. **Assembly cross-verification**
5. **Logic reconstruction**
6. **Runtime validation** using GDB

Each phase was analyzed independently to avoid cascading failures.

Phase-Wise Analysis

Phase 1: String Comparison

Objective: Validate a user-provided string against a hardcoded reference string.

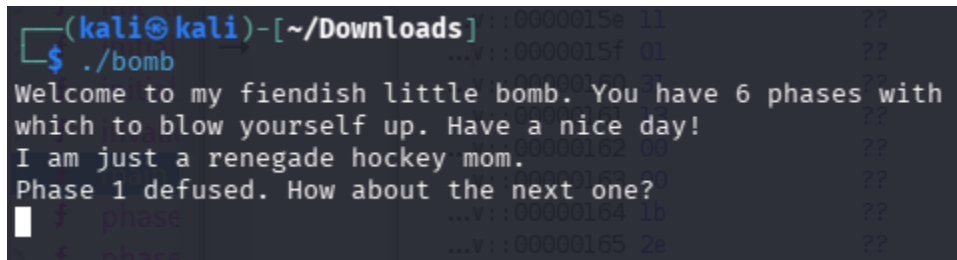
Analysis:

```
2 void phase_1(char input)
3
4 {
5     int result;
6
7     result = strings_not_equal(input, "I am just a renegade hockey mom.");
8     if (result != 0) {
9         explode_bomb();
10    }
11    return;
12 }
```

- The decompiled code revealed a call to `strings_not_equal()`.
- A static string pointer was passed as one argument.
- If the comparison failed, execution branched to `explode_bomb()`.

Answer:

"I am just a renegade hockey mom."



```
(kali㉿kali)-[~/Downloads]
└─$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
└─$
```

Phase 2: Numerical Sequence Validation

Objective: Verify a sequence of six integers follows a specific mathematical rule.

Analysis:

```
2 void phase_2(int input)
3
4 {
5     int *numPointer;
6     long in_FS_OFFSET;
7     int arrayofnum [6];
8     long security_check;
9
10    numPointer = arrayofnum;
11    security_check = *(long *) (in_FS_OFFSET + 0x28);
12    read_six_numbers(input, arrayofnum);
13    if (arrayofnum[0] != 1) {
14        explode_bomb();
15    }
16    do {
17        if (numPointer[1] != *numPointer * 2) {
18            explode_bomb();
19        }
20        numPointer = numPointer + 1;
21    } while (numPointer != arrayofnum + 5);
22    if (security_check == *(long *) (in_FS_OFFSET + 0x28)) {
23        return;
24    }
25    /* WARNING: Subroutine does not return */
26    __stack_chk_fail();
27 }
```

- Input was read using read_six_numbers().
- A loop iteratively validated each element based on the current number is double of previous one.
- Stack offsets indicated an integer array stored locally.

Answer:

1 2 4 8 16 32

```
(kali㉿kali)-[~/Downloads]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
```

Phase 3: Switch Case via jump table

Objective: Match an index-value pair using a jump table.

Analysis:

```
2 void phase_3(int user_input)
3
4 {
5     int tmp;
6     long in_FS_OFFSET;
7     int num;
8     int num1;
9     long security_check;
10
11     security_check = *(long *) (in_FS_OFFSET + 0x28);
12     tmp = __isoc99_sscanf(user_input, "%d %d", &num, &num1);
13     if (tmp < 2) {
14         explode_bomb();
15     }
16     switch(num) {
17     case 0:
18         tmp = 0x274;
19         break;
20     case 1:
21         tmp = 0;
22         break;
23     case 2:
24         tmp = 0;
25         goto fun1;
26     case 3:
27         tmp = 0;
28         goto fun2;
29     case 4:
30         tmp = 0;
31         goto fun3;
32     case 5:
33         tmp = 0;
34         goto fun4;
35     case 6:
36         tmp = 0;
37         goto fun5;
38     case 7:
39         tmp = 0;
40         goto fun6;
41     default:
42         explode_bomb();
43         tmp = 0;
44         goto default_fun;
45     }
```

```

46 tmp = tmp + -0x24c;
47 fun1:
48 tmp = tmp + 0x2b0;
49 fun2:
50 tmp = tmp + -0x7e;
51 fun3:
52 tmp = tmp + 0x7e;
53 fun4:
54 tmp = tmp + -0x7e;
55 fun5:
56 tmp = tmp + 0x7e;
57 fun6:
58 tmp = tmp + -0x7e;
59 default_fun:
60 if ((5 < num) || (num1 != tmp)) {
61     explode_bomb();
62 }
63 if (security_check != *(long *) (in_FS_OFFSET + 0x28)) {
64     /* WARNING: Subroutine does not return */
65     __stack_chk_fail();
66 }
67 return;
68 }

```

- Parses **two integers** using `sscanf`; bomb explodes if fewer than two values are read
- First integer (`num`) controls a **switch statement**
- Switch uses a mix of `break` and `goto`, creating **non-linear control flow**
- Each case determines the **entry point** into an arithmetic sequence
- Arithmetic is performed through **fall-through labels**, accumulating additions and subtractions
- Final computed value depends on **where execution starts**, not just the case number
- Post-computation check restricts `num` to ≤ 5
- Second integer (`num1`) must **exactly equal** the computed value
- Any mismatch triggers `explode_bomb()`
- Phase tests **control-flow tracking**, **switch lowering**, and **arithmetic reconstruction**

Answer:

5 -126

```

(kali@kali)-[~/Downloads]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
5 -126
Halfway there!

```

Phase 4: Recursive Function Analysis

Objective: Provide input that satisfies a recursive computation.

Analysis:

```
2 void phase_4(int input)
3
4 {
5     int numofinput;
6     undefined4 in_register_0000003c;
7     long in_FS_OFFSET;
8     uint first_input;
9     int second_input;
10    long security_check;
11
12    security_check = *(long *)(in_FS_OFFSET + 0x28);
13    numofinput = __isoc99_sscanf(CONCAT44(in_register_0000003c,input),"%d %d",&first_input,
14                                &second_input);
15    if ((numofinput != 2) || (0xe < first_input)) {
16        explode_bomb();
17    }
18    numofinput = func4(first_input,0,0xe);
19    if ((numofinput != 10) || (second_input != 10)) {
20        explode_bomb();
21    }
22    if (security_check == *(long *)(in_FS_OFFSET + 0x28)) {
23        return;
24    }
25    /* WARNING: Subroutine does not return */
26    __stack_chk_fail();
27 }
28
29 int func4(int num_1,int zero,int fourteen)
30
31 {
32     int num;
33     int num1;
34
35     num1 = (fourteen - zero) / 2 + zero;
36     if (num_1 < num1) {
37         num = func4(num_1,zero,num1 + -1);
38         num1 = num1 + num;
39     }
40     else if (num1 < num_1) {
41         num = func4(num_1,num1 + 1,fourteen);
42         num1 = num1 + num;
43     }
44     return num1;
45 }
```

- A recursive function (named func4) was identified.
- The logic resembled a **binary search with accumulated return values**.
- Incorrect traversal paths resulted in failure.

Answer:

3 10

```
(kali㉿kali)-[~/Downloads]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
5 -126
Halfway there!
3 10
So you got that one. Try this one.
█
```

Phase 5: Character Mapping Bitwise Operations

Objective: Transform a 6-character string using a lookup table.

Analysis:

```
2 void phase_5(int user_input)
3
4 {
5     int j;
6     int i;
7     long in_FS_OFFSET;
8     uint num1;
9     int num2;
10    long security_checker;
11
12    security_checker = *(long *) (in_FS_OFFSET + 0x28);
13    j = __isoc99_sscanf(user_input, "%d %d", &num1, &num2);
14    if (j < 2) {
15        explode_bomb();
16    }
17    num1 = num1 & 0xf;
18    if (num1 != 0xf) {
19        j = 0;
20        i = 0;
21        do {
22            i = i + 1;
23            num1 = array_3471[(int) num1];
24            j = j + num1;
25        } while (num1 != 0xf);
26        num1 = 0xf;
27        if ((i == 0xf) && (num2 == j)) goto LAB_0010183a;
28    }
29    explode_bomb();
30 LAB_0010183a:
31    if (security_checker == *(long *) (in_FS_OFFSET + 0x28)) {
32        return;
33    }
34    /* WARNING: Subroutine does not return */
35    __stack_chk_fail();
36 }
```


- Parses **two integers** from user input using `sscanf`
- Bomb explodes if **both integers are not provided**
- First integer (`num1`) is **masked with 0xF** (lower 4 bits only)
- If masked value equals 0xF, the phase immediately fails
- Masked `num1` is used as an **index into a fixed integer array**
- Enters a **loop that follows a chain of indices** through the array
- On each iteration:
 - The next index is read from the array
 - A running **sum (j)** is accumulated
 - A **step counter (i)** is incremented
- Loop continues until the value 0xF is reached
- After loop completion:
 - The number of steps must be **exactly 15**
 - The accumulated sum must **match the second input (num2)**
- If either condition fails, `explode_bomb()` is called
- Phase tests **bit masking, array-based traversal, loop control, and state accumulation**
- Includes a **stack canary check** to detect stack corruption

Answer:

```
1  #include <stdio.h>
2
3  int main() {
4      int array[16] = {
5          10,  2, 14,  7,
6          8, 12, 15, 11,
7          0,  4,  1, 13,
8          3,  9,  6,  5
9      };
10
11     int num1 = 5;
12     int j = 0;
13     int i = 0;
14
15     do {
16         i++;
17         num1 = array[num1];
18         j += num1;
19     } while (num1 != 0xf);
20
21     printf("steps: %d\n", i);
22     printf("sum: %d\n", j);
23
24     return 0;
25 }
```

(kali@kali)-[~/Downloads]

\$./bomb

Welcome to my fiendish little bomb. You have 6 phases with which to blow yourself up. Have a nice day!

I am just a renegade hockey mom.

Phase 1 defused. How about the next one?

1 2 4 8 16 32

That's number 2. Keep going!

5 -126

Halfway there!

3 10

So you got that one. Try this one.

5 115

Good work! On to the next...

Phase 6: Linked List Reordering

Objective: Reorder a linked list so that node values are sorted.

Analysis:

```
4 void phase_6(int input)
5
6 {
7     int i;
8     node_structure *current_node;
9     long j;
10    int *piVar1;
11    long index;
12    long in_FS_OFFSET;
13    int six_num [8];
14    node_structure node [6];
15
16    piVar1 = six_num;
17    node[3].next_node = *(node_structure **)(in_FS_OFFSET + 0x28);
18    read_six_numbers(input,six_num);
19    index = 1;
20    while( true ) {
21        if (5 < *piVar1 - 10) {
22            explode_bomb();
23        }
24        j = index;
25        if (5 < (int)index) break;
26        do {
27            if (*piVar1 == six_num[j]) {
28                explode_bomb();
29            }
30            j = j + 1;
31        } while ((int)j < 6);
32        index = index + 1;
33        piVar1 = piVar1 + 1;
34    }
35    index = 0;
36    do {
37        i = 1;
38        current_node = &node1;
39        if (1 < six_num[index]) {
40            do {
41                current_node = current_node->next_node;
42                i = i + 1;
43            } while (i != six_num[index]);
44        }
45        *(node_structure **>(&node[0].value + index * 2) = current_node;
46        index = index + 1;
47    } while (index != 6);
48    *(node_structure **)(node[0]._0_8_ + 8) = node[0].next_node;
49    (node[0].next_node)->next_node = (node_structure *)node[1]._0_8_;
50    *(node_structure **)(node[1]._0_8_ + 8) = node[1].next_node;
```

```

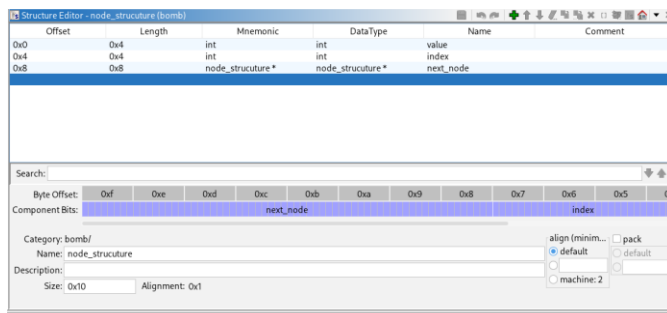
51 (node[1].next_node)->next_node = (node_structure *)node[2]._0_8_;
52 *(node_structure **)(node[2]._0_8_ + 8) = node[2].next_node;
53 (node[2].next_node)->next_node = (node_structure *)0x0;
54 i = 5;
55 piVar1 = (int *)node[0]._0_8_;
56 do {
57     if (*piVar1 < **(int **)(piVar1 + 2)) {
58         explode_bomb();
59     }
60     piVar1 = *(int **)(piVar1 + 2);
61     i = i + -1;
62 } while (i != 0);
63 if (node[3].next_node != *(node_structure **)(in_FS_OFFSET + 0x28)) {
64     /* WARNING: Subroutine does not return */
65     __stack_chk_fail();
66 }
67 return;
68 }

```

- Reads **six integers** into an array using read_six_numbers
- Each input number is validated to ensure it is in the **range 1–6**
- Nested checks ensure **all six numbers are unique** (no duplicates allowed)
- Each input value represents a **position in a predefined linked list**
- For each number:
 - Traverses the linked list starting from node1
 - Selects the node at the specified position
 - Stores selected node pointers in a new array order
- Reconstructs a **new linked list** using the selected nodes, in input order
- Explicitly updates next_node pointers to form the new list
- Traverses the reconstructed list to verify **node values are in non-increasing (descending) order**
- If any node's value is less than the next node's value, the bomb explodes
- Phase enforces correctness through **range checking, uniqueness, pointer manipulation, and sorting logic**
- Includes a **stack canary check** to detect stack corruption

Answer:

Define a node:



```
(kali㉿kali)-[~/Downloads]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
5 -126
Halfway there!
3 10
So you got that one. Try this one.
5 115
Good work! On to the next ...
5 4 3 1 6 2
Congratulations! You've defused the bomb!
```

Secret phase: Binary Tree Traversal

Trigger Mechanism

- Activated after defusing Phase 6 via additional input validation.

Objective: Find an integer that produces a specific traversal path in a binary search tree.

Analysis:

```
2 void secret_phase(void)
3 {
4 {
5     int return_value;
6     char *__nptr;
7     long input;
8
9     __nptr = (char *)read_line();
10    input = strtol(__nptr, (char **)0x0, 10);
11    if (1000 < (int)input - 1U) {
12        explode_bomb();
13    }
14    return_value = fun7(&n1.value, (int)input);
15    if (return_value != 5) {
16        explode_bomb();
17    }
18    puts("Wow! You've defused the secret stage!");
19    phase_defused();
20    return;
21 }
22 }
```

```

2 int fun7(int *n1,int user_input)
3
4 {
5     int return_value;
6
7     if (n1 != (int *)0x0) {
8         if (user_input < *n1) {
9             return_value = fun7(*(int **)(n1 + 2),user_input);
0             return_value = return_value * 2;
1         }
2         else {
3             return_value = 0;
4             if (*n1 != user_input) {
5                 return_value = fun7(*(int **)(n1 + 4),user_input);
6                 return_value = return_value * 2 + 1;
7             }
8         }
9         return return_value;
0     }
1     return -1;
2 }

```

- Reads a **single line of input** from the user using `read_line()`
- Converts the input to an **integer** using `strtol`
- Validates that the input is **within a specific range** (≤ 1000)
- Calls a recursive function `fun7` with the input and the root of a **binary search tree** (`n1.value`)
- `fun7` returns a value that encodes the **path taken in the tree** (e.g., left/right traversal)
- If the return value does not equal **5**, the bomb explodes
- Successfully passing this phase prints a **congratulatory message** and calls `phase_defused()`
- Tests the following skills:
 - **Binary search tree traversal**
 - **Recursive function reasoning**
 - **Input validation and path encoding**
 - Understanding **recursive control flow** and **tree-based constraints**

Answer:

Defining the secret_node

Structure Editor - secret_node (bomb1)

| Offset | Length | Mnemonic | DataType | Name | Comment |
|--------|--------|--------------|--------------|---------|---------|
| 0x0 | 0x4 | int | int | value | |
| 0x4 | 0x4 | int | int | padding | |
| 0x8 | 0x8 | secret_node* | secret_node* | left | |
| 0x10 | 0x8 | secret_node* | secret_node* | right | |

Search:

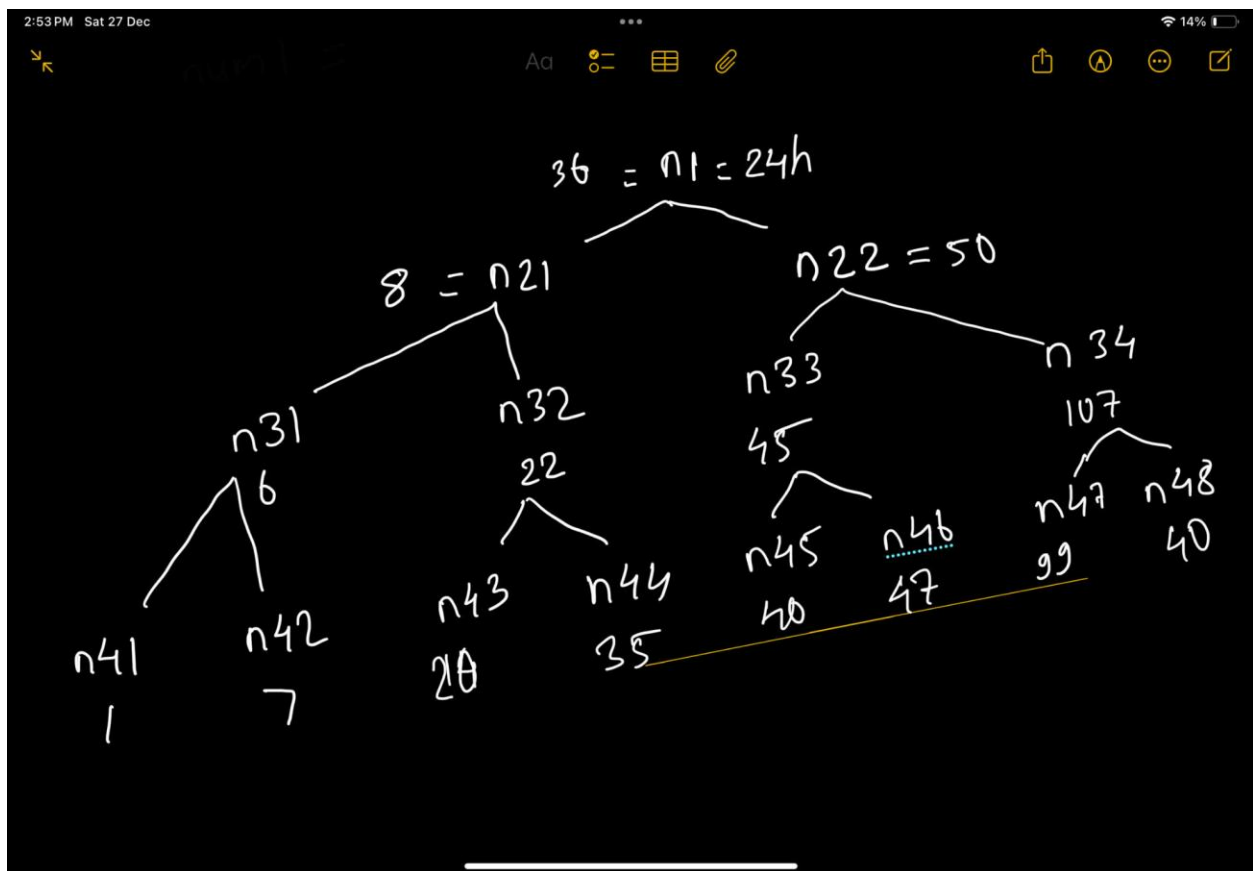
Byte Offset: 0x17 0x16 0x15 0x14 0x13 0x12 0x11 0x10 0xf 0xe 0xd 0xc

Component Bits: right left

Category: bomb/
Name: secret_node
Description:
Size: 0x18 Alignment: 0x1

align (minimum...): ☒ default ☐ machine: 2 ☐ pack ☐ default

Reconstructing Binary Tree:



Answer:

```
(kali㉿kali)-[~/Downloads]
$ cat input.txt
I am just a renegade hockey mom.
1 2 4 8 16 32
5 -126
3 10 DrEvil
5 115
5 4 3 1 6 2
47
```