[EN] A-Z: FreedroidRPG - save game

FreedroidRPG is an open source hack and slash game. In other words, it's a Diablo clone available on Linux. I became interested in how the game handles untrusted user input, but as it's a single player game supplying malicious input is limited. So I decided to analyze the parser of saved games and whether it may be somehow abused. I found a few memory corruption vulnerabilities and a way to execute arbitrary code.

(/images/freedroid.png)

A single save consists of two files: `NAME.shp` and `NAME.sav.gz`. Let's look into the first one.

NAME.shp

`NAME.shp` is a gzipped text file, so we can get its content using following commands:

```
$ mv NAME.shp NAME.gz
$ gunzip NAME.gz
$ cat NAME
[...]
Levelnumber: 0
xlen of this level: 90
ylen of this level: 90
floor layers: 2
light radius bonus of this level: 12
minimal light on this level: 19
infinite_running_on_this_level: 1
random dungeon: 0
[...]
```

It contains many parameters describing the played level. This kind of file is parsed by functions: `LoadShip`, `decode_level`, `decode_level` in https://gitlab.com/freedroid/freedroid-src/blob/master/src/map.c (https://gitlab.com/freedroid/freedroid-src/blob/master/src/map.c). Some of the parsed values coming from the file are used to calculate offsets to memory. The parsing functions lack validation of read values and do not expect that someone may alter data. Usually, users do not edit their own save files (unless they want to cheat, but it's a different subject), but in some game communities, it is quite common to exchange save game files with other players. So unaware users may download or receive manipulated files leading to some undesired behavior.

First crash

The first problem I found in following code:

```
803     static char *decode_map(level *loadlevel, char *data)
825             this_line = (char *)MyMalloc(4096);

845     for (col = 0; col < loadlevel->xlen; col++) {

850                     tmp = strtol(this_line + 4 * (loadlevel->floor_layers *
col + layer), NULL, 10);
```

At line 825 we can see that `this_line` has a fixed size of 4096 bytes. Then, at line 850 this memory is accessed, but offset is calculated using values `loadlevel->xlen` and `loadlevel->floor_layers`. Both of these values come from the file and `loadlevel` is a structure representing it. So if user edits the `NAME.shp` file and sets big enough values, for example: `floor layers: 48` or `xlen of this level: 2048`, the function will read outside memory buffer.

```
=================================================================
==24662==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xb27ed580 at pc
0xb7aaefd8 bp 0xbfb2a1a8 sp 0xbfb29d7c
READ of size 1 at 0xb27ed580 thread T0
    #0 0xb7aaefd7 in strtol (/usr/lib/i386-linux-gnu/libasan.so.2+0x6efd7)
    #1 0x80fc8cc in decode_map /root/projects/freedroid-src/src/map.c:850
    #2 0x80fe326 in decode_level /root/projects/freedroid-src/src/map.c:1126
    #3 0x80ff639 in LoadShip /root/projects/freedroid-src/src/map.c:1303
    #4 0x8127b85 in load_saved_game /root/projects/freedroid-src/src/saveloadgame.c:366
    #5 0x8128240 in load_game /root/projects/freedroid-src/src/saveloadgame.c:478
    #6 0x810dbfd in load_named_game /root/projects/freedroid-src/src/menu.c:1680
    #7 0x810e57a in do_savegame_selection_and_act /root/projects/freedroid-
src/src/menu.c:1796
    #8 0x810e75c in Load_Existing_Hero_Menu /root/projects/freedroid-
src/src/menu.c:1827
    #9 0x810ecaf in Single_Player_Menu /root/projects/freedroid-src/src/menu.c:1895
    #10 0x8108ecd in Startup_handle /root/projects/freedroid-src/src/menu.c:930
    #11 0x8108b6a in RunSubMenu /root/projects/freedroid-src/src/menu.c:872
    #12 0x8108e2f in RunMenu /root/projects/freedroid-src/src/menu.c:901
    #13 0x8108e4c in StartupMenu /root/projects/freedroid-src/src/menu.c:907
    #14 0x80f6e70 in main /root/projects/freedroid-src/src/main.c:179
    #15 0xb75f7636 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x18636)
    #16 0x805c3ee  (/root/projects/freedroid-src/bin/bin/freedroidRPG+0x805c3ee)

0xb27ed580 is located 128 bytes to the right of 4096-byte region
[0xb27ec500,0xb27ed500)
allocated by thread T0 here:
    #0 0xb7ad6f8e in calloc (/usr/lib/i386-linux-gnu/libasan.so.2+0x96f8e)
    #1 0x814a4fd in MyMalloc /root/projects/freedroid-src/src/text_public.c:68
    #2 0x80fc709 in decode_map /root/projects/freedroid-src/src/map.c:825
    #3 0x80fe326 in decode_level /root/projects/freedroid-src/src/map.c:1126
    #4 0x80ff639 in LoadShip /root/projects/freedroid-src/src/map.c:1303
    #5 0x8127b85 in load_saved_game /root/projects/freedroid-src/src/saveloadgame.c:366
    #6 0x8128240 in load_game /root/projects/freedroid-src/src/saveloadgame.c:478
    #7 0x810dbfd in load_named_game /root/projects/freedroid-src/src/menu.c:1680
    #8 0x810e57a in do_savegame_selection_and_act /root/projects/freedroid-
src/src/menu.c:1796
    #9 0x810e75c in Load_Existing_Hero_Menu /root/projects/freedroid-
src/src/menu.c:1827
    #10 0x810ecaf in Single_Player_Menu /root/projects/freedroid-src/src/menu.c:1895
    #11 0x8108ecd in Startup_handle /root/projects/freedroid-src/src/menu.c:930
    #12 0x8108b6a in RunSubMenu /root/projects/freedroid-src/src/menu.c:872
    #13 0x8108e2f in RunMenu /root/projects/freedroid-src/src/menu.c:901
    #14 0x8108e4c in StartupMenu /root/projects/freedroid-src/src/menu.c:907
    #15 0x80f6e70 in main /root/projects/freedroid-src/src/main.c:179
    #16 0xb75f7636 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x18636)

SUMMARY: AddressSanitizer: heap-buffer-overflow ??:0 strtol
Shadow bytes around the buggy address:
  0x364fda60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x364fda70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x364fda80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x364fda90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x364fdaa0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x364fdab0:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x364fdac0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x364fdad0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x364fdae0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x364fdaf0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x364fdb00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
==24662==ABORTING
```


Second crash

The second problem is related to two similar fragments of code parsing two fragments of the file that don't have a fixed size.

In the save game we can find:

```
[...]
beginning_of_map
116   0 116   0 116   0 116   0 116   0 116   0 116   0 116   0 116   0 116   0 116   0 116
0 116   0 116   0 116   0 116   0 116   0 116 558  47 501   0 524   0 524   0 521  47 502  25
0  23   0   0 521  25   0  23   0  47 500   0 521   0 52
[...]
```

and code responsible for parsing it:

```
825     this_line = (char *)MyMalloc(4096);

837             while (map_begin[curlinepos + nlpos] != '\n')
838                     nlpos++;
839             memcpy(this_line, map_begin + curlinepos, nlpos);
```

Again we meet fixed buffer `this_line`. This part of code tries to copy data resisting between `beginning_of_map` tag and newline character announcing the end of the value. The function assumes that data between the tag and newline character will fit to the `this_line` buffer. It's not true as malicious user can modify the save and put more than 4096 characters in that place. If a such modified save is loaded the game will write data outside the buffer.

```
==24511==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xa6583100 at pc
0xb7a1ba42 bp 0xbfe940c8 sp 0xbfe93c9c
WRITE of size 5000 at 0xa6583100 thread T0
    #0 0xb7a1ba41 in __asan_memcpy (/usr/lib/i386-linux-gnu/libasan.so.2+0x8aa41)
    #1 0xb7a1bc2f in memcpy (/usr/lib/i386-linux-gnu/libasan.so.2+0x8ac2f)
    #2 0x80fc79b in decode_map /root/projects/freedroid-src/src/map.c:839
    #3 0x80fe326 in decode_level /root/projects/freedroid-src/src/map.c:1126
    #4 0x80ff639 in LoadShip /root/projects/freedroid-src/src/map.c:1303
    #5 0x8127b85 in load_saved_game /root/projects/freedroid-src/src/saveloadgame.c:366
    #6 0x8128240 in load_game /root/projects/freedroid-src/src/saveloadgame.c:478
    #7 0x810dbfd in load_named_game /root/projects/freedroid-src/src/menu.c:1680
    #8 0x810e57a in do_savegame_selection_and_act /root/projects/freedroid-
src/src/menu.c:1796
    #9 0x810e75c in Load_Existing_Hero_Menu /root/projects/freedroid-
src/src/menu.c:1827
    #10 0x810ecaf in Single_Player_Menu /root/projects/freedroid-src/src/menu.c:1895
    #11 0x8108ecd in Startup_handle /root/projects/freedroid-src/src/menu.c:930
    #12 0x8108b6a in RunSubMenu /root/projects/freedroid-src/src/menu.c:872
    #13 0x8108e2f in RunMenu /root/projects/freedroid-src/src/menu.c:901
    #14 0x8108e4c in StartupMenu /root/projects/freedroid-src/src/menu.c:907
    #15 0x80f6e70 in main /root/projects/freedroid-src/src/main.c:179
    #16 0xb7548636 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x18636)
    #17 0x805c3ee  (/root/projects/freedroid-src/bin/bin/freedroidRPG+0x805c3ee)

0xa6583100 is located 0 bytes to the right of 4096-byte region [0xa6582100,0xa6583100)
allocated by thread T0 here:
    #0 0xb7a27f8e in calloc (/usr/lib/i386-linux-gnu/libasan.so.2+0x96f8e)
    #1 0x814a4fd in MyMalloc /root/projects/freedroid-src/src/text_public.c:68
    #2 0x80fc709 in decode_map /root/projects/freedroid-src/src/map.c:825
    #3 0x80fe326 in decode_level /root/projects/freedroid-src/src/map.c:1126
    #4 0x80ff639 in LoadShip /root/projects/freedroid-src/src/map.c:1303
    #5 0x8127b85 in load_saved_game /root/projects/freedroid-src/src/saveloadgame.c:366
    #6 0x8128240 in load_game /root/projects/freedroid-src/src/saveloadgame.c:478
    #7 0x810dbfd in load_named_game /root/projects/freedroid-src/src/menu.c:1680
    #8 0x810e57a in do_savegame_selection_and_act /root/projects/freedroid-
src/src/menu.c:1796
    #9 0x810e75c in Load_Existing_Hero_Menu /root/projects/freedroid-
src/src/menu.c:1827
    #10 0x810ecaf in Single_Player_Menu /root/projects/freedroid-src/src/menu.c:1895
    #11 0x8108ecd in Startup_handle /root/projects/freedroid-src/src/menu.c:930
    #12 0x8108b6a in RunSubMenu /root/projects/freedroid-src/src/menu.c:872
    #13 0x8108e2f in RunMenu /root/projects/freedroid-src/src/menu.c:901
    #14 0x8108e4c in StartupMenu /root/projects/freedroid-src/src/menu.c:907
    #15 0x80f6e70 in main /root/projects/freedroid-src/src/main.c:179
    #16 0xb7548636 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x18636)

SUMMARY: AddressSanitizer: heap-buffer-overflow ??:0 __asan_memcpy
Shadow bytes around the buggy address:
  0x34cb05d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x34cb05e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x34cb05f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x34cb0600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x34cb0610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x34cb0620:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x34cb0630: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x34cb0640: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x34cb0650: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x34cb0660: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x34cb0670: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
==24511==ABORTING
```

Second crash, part 2

The problem is exactly the same for parsing this part of the save.

```
[...]
wp
Nr.=  0 x=  27 y=  17 rnd=0      c:    1  53
[...]
```

However, it occurs in another function:

```
875 static char *decode_waypoints(level *loadlevel, char *data)
896     this_line = (char *)MyMalloc(4096);

902     while (wp_begin[curlinepos + nlpos] != '\n')
903                     nlpos++;
904             memcpy(this_line, wp_begin + curlinepos, nlpos);
```

If data between `wp` and newline character is longer than 4096 bytes, than the same vulnerability occurs - data is written outside the buffer.

NAME.sav.gz

The second file is also a gzipped text file, but its content is quite different.

```
game_config{
played_game_act = [=[act1]=],
}
tux_t{
current_game_date = 0.208548,
current_power_bonus = 0,
power_bonus_end_date = -1.000000,
current_dexterity_bonus = 0,
dexterity_bonus_end_date = -1.000000,
light_bonus_end_date = 0.000000,
speed = {
x = 0.000000,
y = 0.000000,
},
```

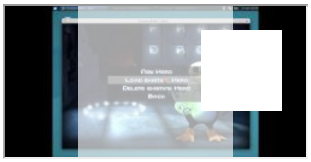
If we look into the code (https://gitlab.com/freedroid/freedroid-src/blob/master/src/savestruct_internal.c) parsing we will notice that it's Lua code being interpreted while loading the save.

```
void load_game_data(char *strin)
{
        // Add the table constructors called by Lua when parsing a savegame

[...]
        // Parse the configuration file, calling table constructors to create the
        // associated C data structures

        run_lua(LUA_DIALOG, strin);
}
```

So, in this case, the attacker has a much easier job. It's possible to put any Lua code we want, for example `os.execute("xcalc")`.



Summary

Even single player games are susceptible to malicious input. Every kind of data we deliver to our games (save games, mods, additional maps, etc.) can be a way of triggering potential vulnerabilities in code. In the case of FreedroidRPG, it was possible to cause some (but I don't know if exploitable in practice) memory corruption and what is worse, execute arbitrary code.

- https://bugs.freedroid.org/b/issue951 (https://bugs.freedroid.org/b/issue951)
- https://bugs.freedroid.org/b/issue952 (https://bugs.freedroid.org/b/issue952)
- https://bugs.freedroid.org/b/issue953 (https://bugs.freedroid.org/b/issue953)

Written on February 12, 2020 by Michał Dardas

Name and surname / company*

E-mail*

Telephone (optional)

Message*

☐ **I agree to the processing of my personal data and sending the offer.**
Rules for the processing of personal data.

send

LogicalTrust sp. z o.o.
sp. k.
NIP: 8952177980
KRS: 0000713515
office@logicaltrust.net (mailto:office@logicaltrust.net)
Key: PGP/GPG (/logicaltrust.gpg)

al. Aleksandra Brücknera 25-43
51-411 Wrocław, Poland, EU
T.: +48 71 738 24 35 (tel:+48717382435)
K.: +48 514 812 431 (tel:+48514812431)

LOGICALTRUST