Virtual World RE
Gotcha Force

Files patching process v1.0.5

N0 - Public ; N1 - Restricted ; N2 - Confidential ; N3 - Secret

Table of Contents

1. Summary:

Welcome! This guide is intended to explain how to help reverse engineering the GameCube game **Gotcha Force**. The main goal is to better understand some of the strategies to do reverse engineering on the Gocha Force files. Examples are done using the **NTSC GCM/iso** version. Here is presented the global patching process and tools used to achieve this. There is screens to show how to use Hexadecimal editor with bookmarks to investigate files easily with community sharing and a demo to automate the whole patching process using a python script that can be adapted.

Some techniques described in this guide could require some strong basis in informatics like python programming and file formats.

2. What tools do I need?

Hexadecimal editor

First of all you have to find your hexadecimal editor. Values are stored in game files or memory in raw “bytes” or if you prefer “8 bits” by “8 bits”.

The important thing to know about it is that when we store data we use an endianness. This mean that this same “bytes” can be stored in different order:

- Big-endian

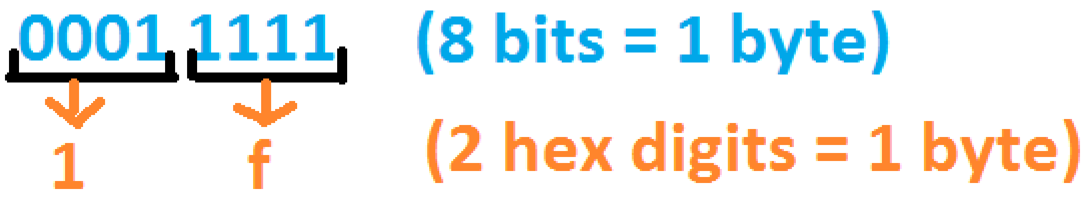
- Little-endian

- (… others are possible)

So bits keep the same order but we store bytes (in files / memories) in one direction or in another. **All the individual bits are not reversed** though. Here is an example. Each 8 bits of the first byte has an index from 0 to 7: "0123 4567". The second byte is "XXXX XXXX". The value is "0123 4567 XXXX XXXX": 2 bytes long.

1. If we store it in **big endian** we will have:
   "0123 4567 XXXX XXXX"
2. And now in **little endian** it will be:
   "XXXX XXXX 0123 4567"

To simplify reading we use hexadecimal representation rather than binary. It's easy to read and convert one byte by splitting the 8 bits in two 4 bits block:

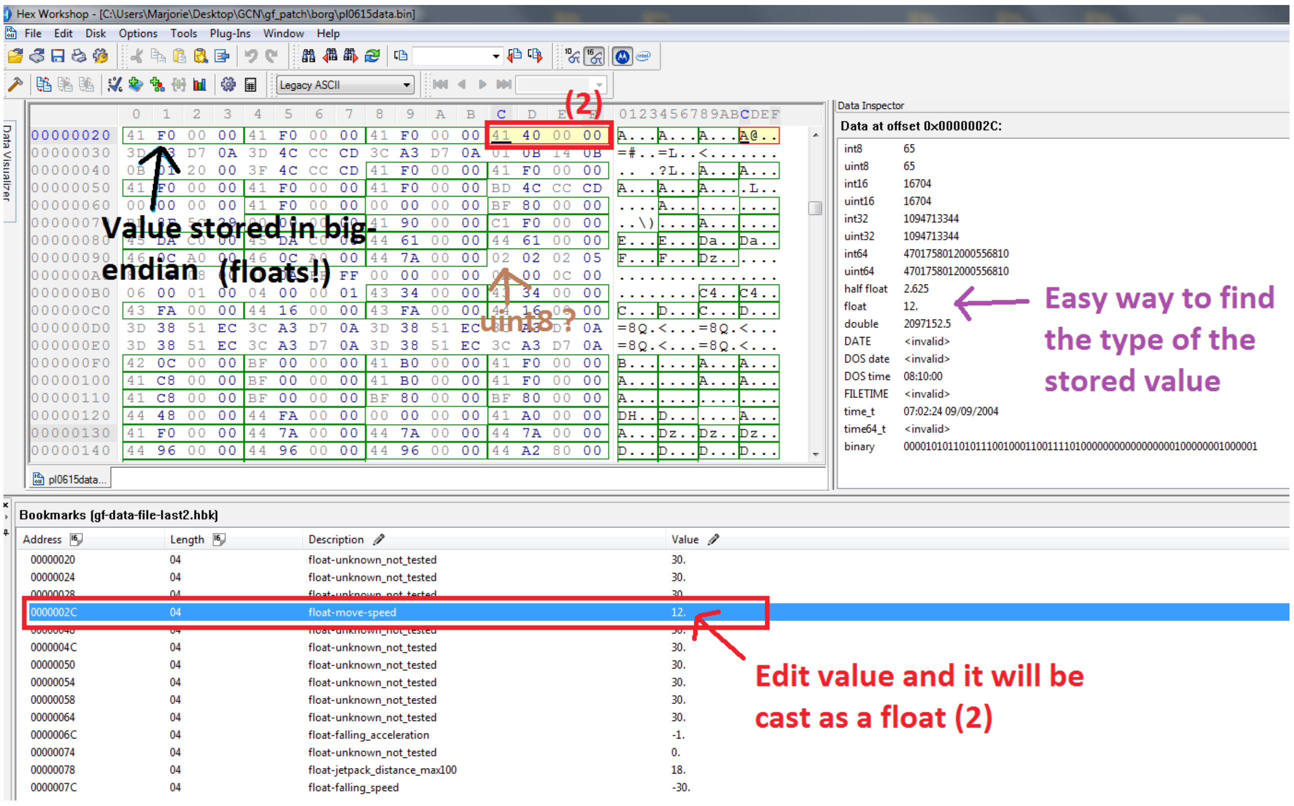


Any hex editor is good if you can:

- patch any byte of the file with a new value or remove / add data
- compare 2 files between them
- See values in different encoding (int8, int16, int32, uint8, float...)
- Use bookmarks or a system to create a memory mapping of the values present in the file.

I use hex workshop and it's bookmarks to easily determine every type of variable and automatically encode in the right format when I change a value. Find a good one!

Here is an example of the work on the standard "plxxxxdata.bin" files of borgs:


Hex Workshop - [C:\Users\Marjorie\Desktop\GCN\gf_patch\borg\pl0615data.bin]
File Edit Disk Options Tools Plug-Ins Window Help
Legacy ASCII
Data Visualizer
pl0615data...
Value stored in big-endian (floats!)
(2)
uint8 ?
Easy way to find the type of the stored value
Edit value and it will be cast as a float (2)

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00000020  41 F0 00 00 41 F0 00 00 41 F0 00 00 41 40 00 00  A...A...A...A@..
00000030  3D 23 D7 0A 3D 4C CC CD 3C A3 D7 0A 01 0B 14 0B  =#..=L..<.......
00000040  0B 01 20 00 3F 4C CC CD 41 F0 00 00 41 F0 00 00  .. .?L..A...A...
00000050  41 F0 00 00 41 F0 00 00 41 F0 00 00 BD 4C CC CD  A...A...A....L..
00000060  00 00 00 00 41 F0 00 00 00 00 00 00 BF 80 00 00  ....A...........
00000070  BE 8F 5C 29 00 00 00 00 41 90 00 00 C1 F0 00 00  ..\)....A.......
00000080  45 DA C0 00 45 DA C0 00 44 61 00 00 44 61 00 00  E...E...Da..Da..
00000090  46 0C A0 00 46 0C A0 00 44 7A 00 00 02 02 02 05  F...F...Dz......
000000A0  [illegible]          FF 00 00 00 00 [illegible] 00 0C 00  ................
000000B0  06 00 01 00 04 00 00 01 43 34 00 00 43 34 00 00  ........C4..C4..
000000C0  43 FA 00 00 44 16 00 00 43 FA 00 00 44 16 00 00  C...D...C...D...
000000D0  3D 38 51 EC 3C A3 D7 0A 3D 38 51 EC 3C A3 D7 0A  =8Q.<...=8Q.<...
000000E0  3D 38 51 EC 3C A3 D7 0A 3D 38 51 EC 3C A3 D7 0A  =8Q.<...=8Q.<...
000000F0  42 0C 00 00 BF 00 00 00 41 B0 00 00 41 F0 00 00  B.......A...A...
00000100  41 C8 00 00 BF 00 00 00 41 B0 00 00 41 F0 00 00  A.......A...A...
00000110  41 C8 00 00 BF 00 00 00 BF 80 00 00 BF 80 00 00  A...............
00000120  44 48 00 00 44 FA 00 00 00 00 00 00 41 A0 00 00  DH..D.......A...
00000130  41 F0 00 00 44 7A 00 00 44 7A 00 00 44 7A 00 00  A...Dz..Dz..Dz..
00000140  44 96 00 00 44 96 00 00 44 96 00 00 44 A2 80 00  D...D...D...D...
```

Data Inspector

Data at offset 0x0000002C:

int8	65
uint8	65
int16	16704
uint16	16704
int32	1094713344
uint32	1094713344
int64	4701758012000556810
uint64	4701758012000556810
half float	2.625
float	12.
double	2097152.5
DATE	<invalid>
DOS date	<invalid>
DOS time	08:10:00
FILETIME	<invalid>
time_t	07:02:24 09/09/2004
time64_t	<invalid>
binary	0000101011010111001000110011110100000000000000000100000001000001

Bookmarks (gf-data-file-last2.hbk)

Address	Length	Description	Value
00000020	04	float-unknown_not_tested	30.
00000024	04	float-unknown_not_tested	30.
00000028	04	float-unknown_not_tested	30.
0000002C	04	float-move-speed	12.
00000048	04	float-unknown_not_tested	30.
0000004C	04	float-unknown_not_tested	30.
00000050	04	float-unknown_not_tested	30.
00000054	04	float-unknown_not_tested	30.
00000058	04	float-unknown_not_tested	30.
00000064	04	float-unknown_not_tested	30.
0000006C	04	float-falling_acceleration	-1.
00000074	04	float-unknown_not_tested	0.
00000078	04	float-jetpack_distance_max100	18.
0000007C	04	float-falling_speed	-30.

Python 3

Get python 3 to script patching process, use Virtual World RE scripts, and gain lot of time to do easily tests on large amounts of files.

GameCube Emulator

A good GameCube Emulator will help you when dynamically reversing the game:

- Patch the game
- Run and see what's changed
- Patch again
- Run again
- Use dynamic debugging of the game! (change registers or memory values, put breakpoints ...)

Dolphin Emulator is what I use.

GameCube DVD patching tool

You need a tool to export and import back files from and to the Gotcha Force **GCM/iso** file. Virtual Word RE provide a full command line tool, gcmtool.py:

https://github.com/Virtual-World-RE/NeoGF/blob/main/gcmtool

This tool can patch every files from the initial GCM iso like sys files (apploader.img boot.dol) and files inside the DVD changing their length if needed.

AFS patching tool

The afs is a special GameCube file format often used in games that pack the game files. You need a special tool to unpack or pack files to this afs.

Virtual World RE provide afstool.py:

https://github.com/Virtual-World-RE/NeoGF/blob/main/afstool

With the AFS format the main problem is “how the game (boot.dol) retrieve files inside the afs?” afstool.py allow you to use different strategies. You have to configure it to a rebuild by “index” in the Table Of Content (TOC). This will allow you to change length of every file you want to patch.

PZZ patching tool

PZZ is a file format storing multiple files with a compression algorithm. Virtual World RE provide pzztool.py:

https://github.com/Virtual-World-RE/NeoGF/blob/main/pzztool

There is multiple command to optimize patching process by handling compression algorithm during unpack or not.

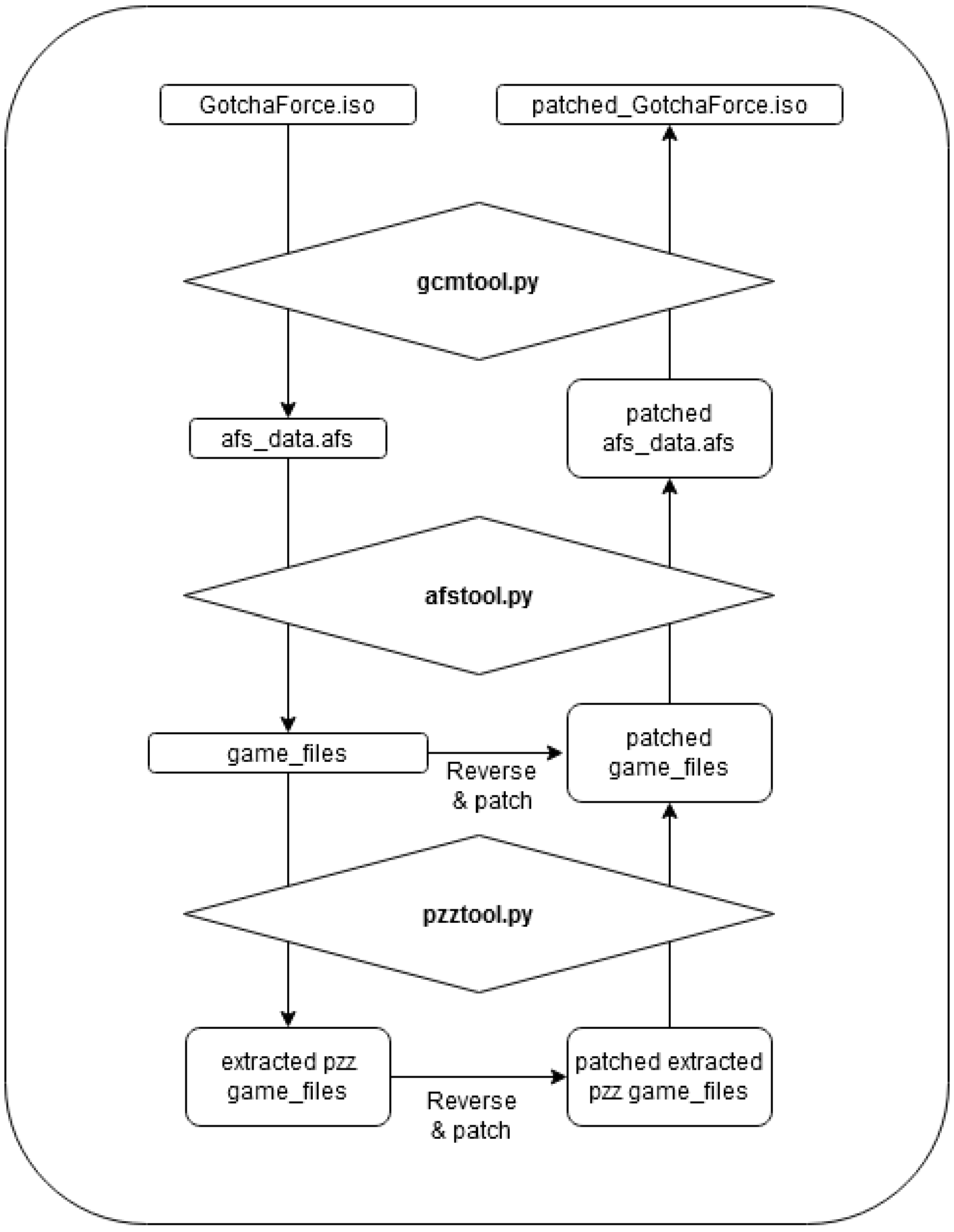
Dolphin Memory Engine (optional)

https://github.com/aldelaro5/Dolphin-memory-engine

This tool can help you to manage the memory during dolphin emulator research sessions.

2. How to patch the Game?

Here is a diagram explaining the different steps to patch the **GCM iso** game:



First of all you have to <u>be organized and use correct naming of files</u> during the patching process. It will help you to find which file are patched and so on.

1. Make a folder named "gf_patch"
2. Put your Gotcha Force iso file "gotcha_force.iso" in it
3. Enter inside the gf_patch folder (with your prompt)

4. Create "tool" folder and download in it the Virtual World RE tools: **gcmtool.py**, **afstool.py**, **pzztool.py**. (Next use tools/gcmtool.py for instance in each command)
5. You will now unpack files from the gf iso into a folder:
   - `gcmtool.py -u gotcha_force.iso iso_unpack` # unpack of the iso
6. Then we extract afs_data.afs:
   - `afstool.py -u iso_unpack/root/afs_data.afs afsdata_unpack` # unpack of the afs
7. Now we will configure the afs to enable changing length of files and remove empty padding to optimize it. Open the configuration file "**afsdata_unpack/sys/afs_rebuild.conf**" and change the following values:
   - In [Default] change **files_rebuild_strategy = index**
   - In [FilenameDirectory] change **toc_offset_of_fd_offset = auto**
   - In [FilenameDirectory] change **fd_offset = auto**
8. Now we rebuild the afs system files into the unpacked afs folder:
   - `afstool.py -r afsdata_unpack` # rebuild of TOC & FD
9. Now we pack the rebuilded afs into the unpacked GCM/iso folder:
   - `afstool.py -p afsdata_unpack iso_unpack/root/afs_data.afs` # pack of the afs
10. Now we rebuild the GCM/iso FST to optimize space and accelerate repack but also to change length of files inside it:
    - `gcmtool.py -r iso_unpack` # rebuild of the FST
11. And finally we pack the files in a new GCM/iso:
    - `gcmtool.py -p iso_unpack gotcha_force_patched.iso` # new patched iso

What to remember?

- Every time you patch one file in the unpacked AFS folder and change it's length you have to rebuild the afs using the -r command on the folder.
- Every time you change the length of a file in the GCM/iso folder you have to rebuild the gcm using the -r command.

Using this process you can change every files as you want.

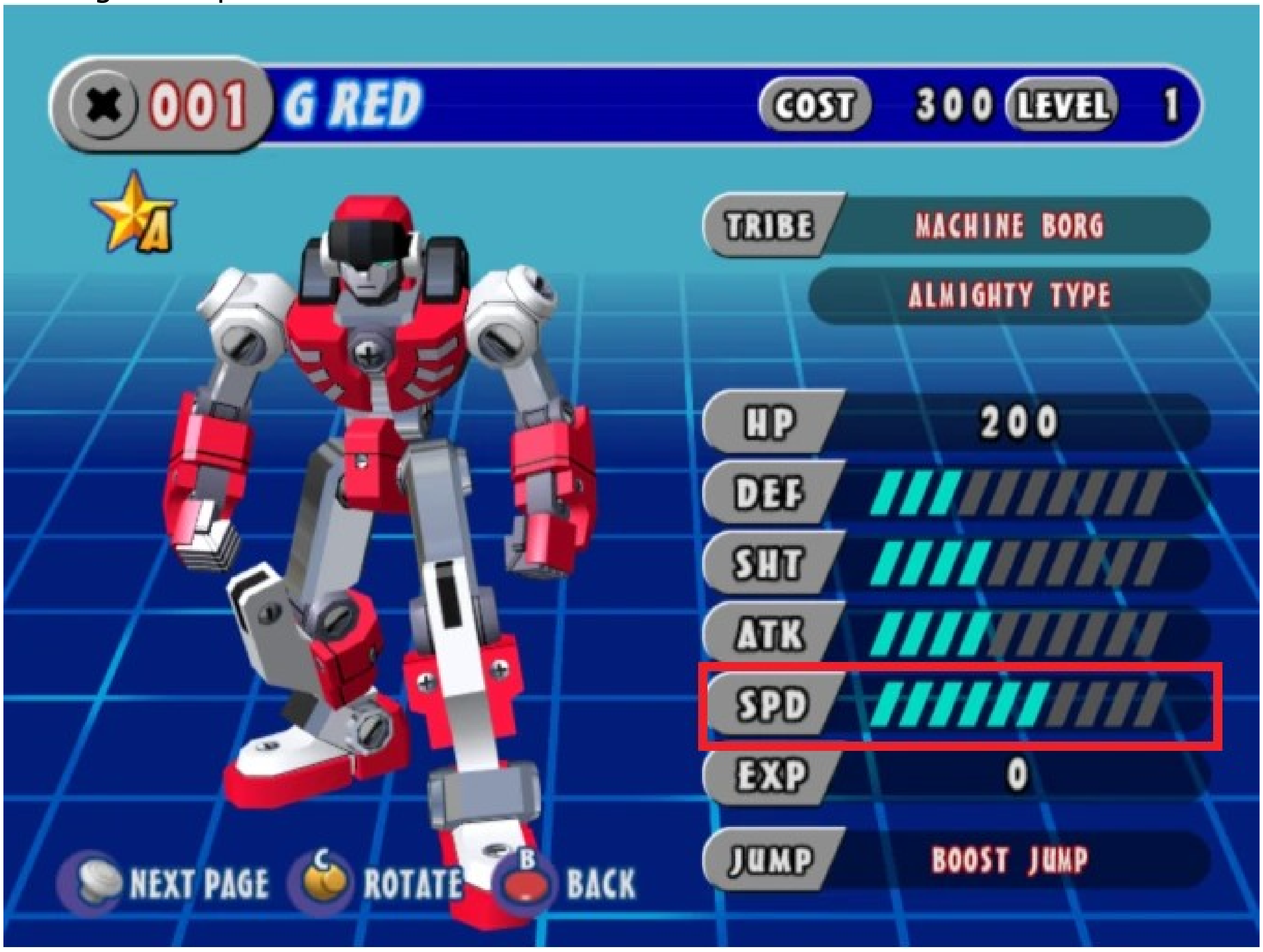
Using a python script to automate the patching process is possible. You change every options on the dolphin emulator configurations to run with max CPU speed, and so on! It could be really quick to patch a file (like shown in Hex WorkShop in the "**What tools do I need**" section) and run to see the impact on the game.
Here is a script that I've made to patch the borg data file (plxxxxdata.bin) and see in-game impact:

https://github.com/tmpz23/scripts/blob/main/gf_research_tool.py

You can edit this to patch an other file. Or ask algoflash on Discord to do this.

3. Demo: change G Red speed in collections menu

In this demo we will patch the speed of G Red displayed in the menu collection → borg description:



This csv contains all borgs that are in the NTSC game. First column describe the filename and second column describe the borg name for instance “pl0615” is the filenames base for the first story borg used in the game “G Red”.

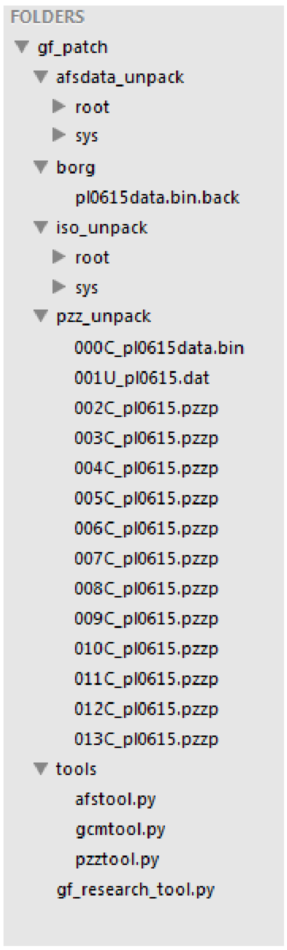
We will now add a cheat code in Gocha Force to have full borg “collection” available. Right click on the game then go to properties and AR Code and add the value “003BFF78 0000CD01” in a new AR Code.

Download this tool:
https://github.com/tmpz23/scripts/blob/main/gf_research_tool.py

Inside this script edit this two variables with the path of your Gotcha Force ROM and with the full path of dolphin emulator exe:

```
# Original Gotcha Force GCM iso PATH
GF_ISO_PATH = Path("ROM/Gotcha Force (USA).iso")
# https://fr.dolphin-emu.org/download/
dolphin_path = Path("C:/Program Files/Dolphin/Dolphin.exe")
```

Then install using this command:
`gf_reasearch_tool.py -i` # it will create "**gf_patch**" folder and download Virtual World RE python tools.



```
FOLDERS
▼ gf_patch
  ▼ afsdata_unpack
    ▶ root
    ▶ sys
  ▼ borg
      pl0615data.bin.back
  ▼ iso_unpack
    ▶ root
    ▶ sys
  ▼ pzz_unpack
      000C_pl0615data.bin
      001U_pl0615.dat
      002C_pl0615.pzzp
      003C_pl0615.pzzp
      004C_pl0615.pzzp
      005C_pl0615.pzzp
      006C_pl0615.pzzp
      007C_pl0615.pzzp
      008C_pl0615.pzzp
      009C_pl0615.pzzp
      010C_pl0615.pzzp
      011C_pl0615.pzzp
      012C_pl0615.pzzp
      013C_pl0615.pzzp
  ▼ tools
      afstool.py
      gcmtool.py
      pzztool.py
    gf_research_tool.py
```

At the left this is what your folder should looks like now.

Keep in mind that you must never add or remove files in the folder "gf_patch/afsdata_unpack" or in "gf_patch/iso_unpack".

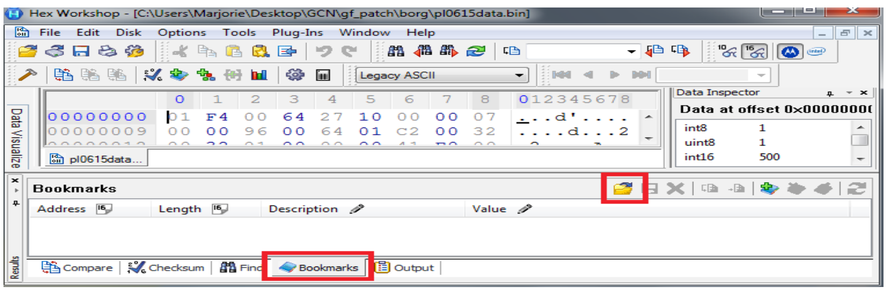
The working directory is the "**gf_patch/borg**".

Copy pl0615data.bin.back and rename it with "**pl0615data.bin**". You will now edit this file to patch the game.

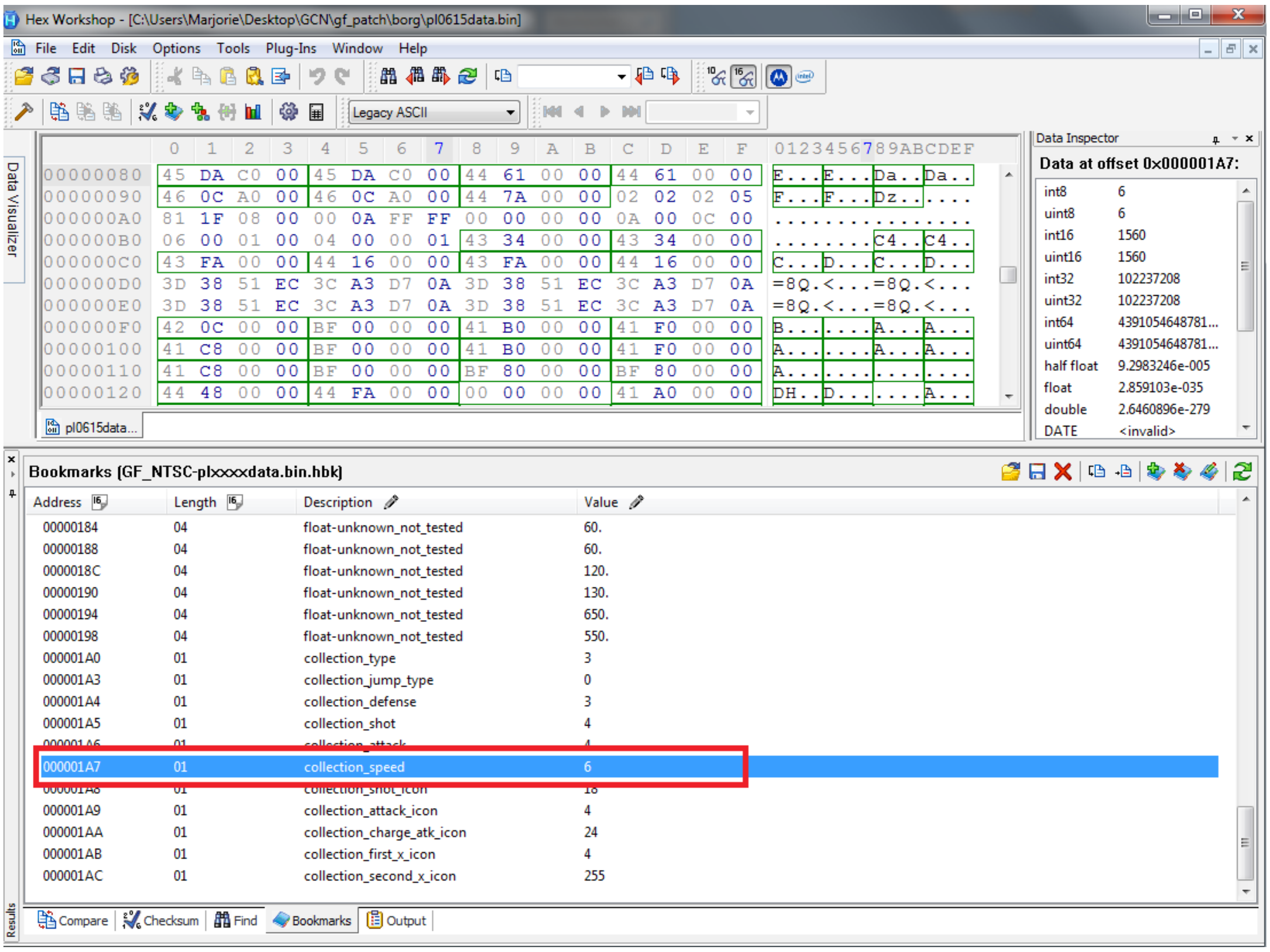
It will automatically patch the afsdata_unpack with:
(1) pl0615data.bin
(2) pl0615.pzz
Both have to contains the sames data to avoid crashes.

Now open **gf_patch/borg/pl0615data.bin** with Hex Workshop and then go to the "bookmarks" tab and open this file:

https://github.com/Virtual-World-RE/NeoGF/blob/main/data/GF_NTSC-plxxxxdata.bin.hbk


Hex Workshop - [C:\Users\Marjorie\Desktop\GCN\gf_patch\borg\pl0615data.bin]
File Edit Disk Options Tools Plug-Ins Window Help
Legacy ASCII
Data Inspector
Data at offset 0x00000000
int8 1
uint8 1
int16 500
pl0615data...
Bookmarks
Address Length Description Value
Compare Checksum Find Bookmarks Output

```
          0  1  2  3  4  5  6  7  8   012345678
00000000  01 F4 00 64 27 10 00 00 07  ...d'....
00000009  00 00 96 00 64 01 C2 00 32  ....d...2
```



Hex Workshop - [C:\Users\Marjorie\Desktop\GCN\gf_patch\borg\pl0615data.bin]

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00000080  45 DA C0 00 45 DA C0 00 44 61 00 00 44 61 00 00  E...E...Da..Da..
00000090  46 0C A0 00 46 0C A0 00 44 7A 00 00 02 02 02 05  F...F...Dz......
000000A0  81 1F 08 00 00 0A FF FF 00 00 00 00 0A 00 0C 00  ................
000000B0  06 00 01 00 04 00 00 01 43 34 00 00 43 34 00 00  ........C4..C4..
000000C0  43 FA 00 00 44 16 00 00 43 FA 00 00 44 16 00 00  C...D...C...D...
000000D0  3D 38 51 EC 3C A3 D7 0A 3D 38 51 EC 3C A3 D7 0A  =8Q.<...=8Q.<...
000000E0  3D 38 51 EC 3C A3 D7 0A 3D 38 51 EC 3C A3 D7 0A  =8Q.<...=8Q.<...
000000F0  42 0C 00 00 BF 00 00 00 41 B0 00 00 41 F0 00 00  B.......A...A...
00000100  41 C8 00 00 BF 00 00 00 41 B0 00 00 41 F0 00 00  A.......A...A...
00000110  41 C8 00 00 BF 00 00 00 BF 80 00 00 BF 80 00 00  A...............
00000120  44 48 00 00 44 FA 00 00 00 00 00 00 41 A0 00 00  DH..D.......A...
```

Data Inspector

Data at offset 0x000001A7:

int8	6
uint8	6
int16	1560
uint16	1560
int32	102237208
uint32	102237208
int64	4391054648781...
uint64	4391054648781...
half float	9.2983246e-005
float	2.859103e-035
double	2.6460896e-279
DATE	<invalid>

Bookmarks (GF_NTSC-plxxxxdata.bin.hbk)

Address	Length	Description	Value
00000184	04	float-unknown_not_tested	60.
00000188	04	float-unknown_not_tested	60.
0000018C	04	float-unknown_not_tested	120.
00000190	04	float-unknown_not_tested	130.
00000194	04	float-unknown_not_tested	650.
00000198	04	float-unknown_not_tested	550.
000001A0	01	collection_type	3
000001A3	01	collection_jump_type	0
000001A4	01	collection_defense	3
000001A5	01	collection_shot	4
000001A6	01	collection_attack	4
000001A7	01	collection_speed	6
000001A8	01	collection_shot_icon	18
000001A9	01	collection_attack_icon	4
000001AA	01	collection_charge_atk_icon	24
000001AB	01	collection_first_x_icon	4
000001AC	01	collection_second_x_icon	255

Change the value of collection_speed with “10” and save. Then enter the command:

```
gf_reasearch_tool.py -pr
```

This command will patch the borg pzz after patching the data file inside at position 000. Then it will patch the unpacked afsdata with both (data.bin and .pzz) and pack the afs into the unpacked gcm and pack the gcm into gf_patched.iso and run it into dolphin emulator.

Then ingame go to the menu “collection” and view the borg “G Red”. You should now see “10” in the speed value.

With the -pr argument you can test all values in this file. But it will work if you edit the script to achieve this with another file. The main advantage is that it is really fast and allow to see changes in multiple prompt and dolphin emulator windows.

How to help?

You now get the patching process and you can automate it using a python script. The main goal is to find what values in the borg data files are used for. To optimize the process there are commands that use dichotomic search algorithms to patch a range of value inside the plxxxxdata.bin.

The idea of this algorithm is to patch half of the bytes of both files:

- pl0615data.bin
- pl0615/000C_pl0615data.bin

And if there is a change ingame then patch half of the half and see if the change is here or not. If not your searched offset responsible of the change is in the other half than the half you have patched. So cut the other half in two and patch one of the two part to see if the searched offset is in it. Repeat until you find the real offset and values that are stored in it.

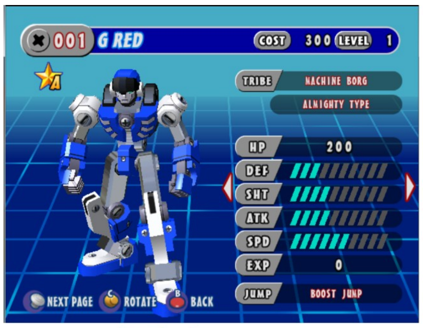
Here is a trick:
For a known offset you can list ALL the values that are taken in all plxxxxdata.bin files with a simple python script.

See gf_research_tool.py help and code for further information. And you can adapt it for the file you want.

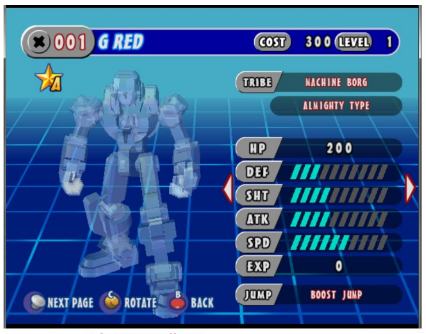
Conclusion

You are now a confirmed GameCube files reverse engineer! Come on discord and share the new offset you have found and their impact on the game! Pl0615 (G Red) is the first borg of the story so use it for your tests.

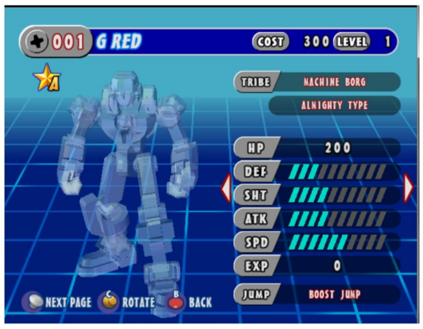
Also you can switch borgs by renaming all their files (plxxxx.pzz and plxxxx*) and erasing the targeted borg you want to change. Here is a screen of changing files to test ingame the different G Red _mdl:



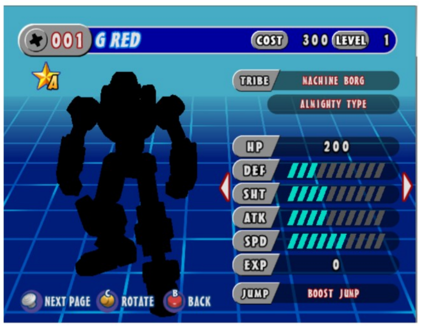
pl0615b_mdl.arc



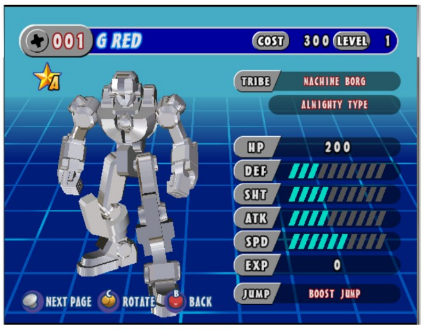
pl0615c_mdl.arc



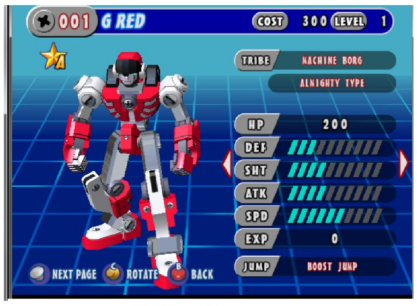
pl0615g_mdl.arc



pl0615k_mdl.arc



pl0615s_mdl.arc



pl0615_mdl.arc