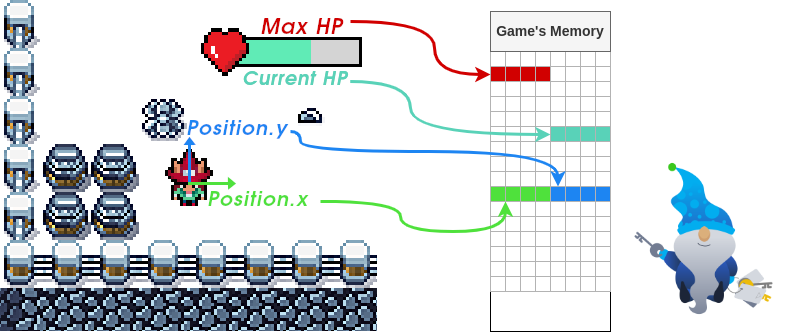
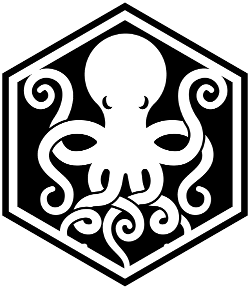
The Memory of a Program

Whenever we execute a program, all data will be processed somehow through the computer's RAM (Random Access Memory). If you think of a videogame, your HP, position, movement speed and direction are all stored somewhere in memory and updated as needed as the game goes.



If you can modify the relevant memory positions, you could trick the game into thinking you have more HP than you should or even a higher score! This sounds relatively easy, but a program's memory space is vast and sparse, and finding the location where these variables are stored is nothing you'd want to do by hand. Hopefully, some tools will help us navigate memory and find where all the juicy information is at.

Be sure to hit the **Start Machine** button before continuing. The machine will start in a split-screen view. In case the VM is not visible, use the blue Show Split View button at the top-right of the page. All you need for this challenge is available in the deployable machine. If you prefer to do so, however, you can download and install Cetus on your own machine by downloading it [from here](https://github.com/Qwokka/Cetus/releases/download/v1.03.1/Cetus_v1.03.1.zip).



The Mighty Cetus

Cetus is a simple browser plugin that works for Firefox and Chrome, allowing you to explore the memory space of Web Assembly games that run in your browser. The main idea behind it is to provide you with the tools to easily find any piece of data stored in memory and modify it if needed. On top of that, it will let you modify a game's compiled code and alter its behaviours if you want, although we won't need to go that deep for this task.

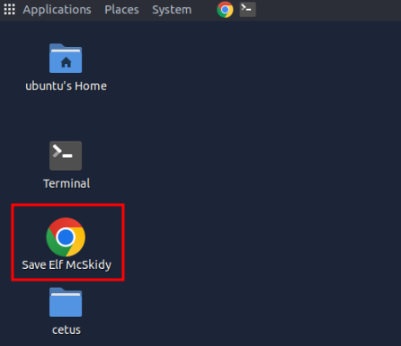
Cetus is already installed on Chrome in your deployed machine, so you can use it straight away for the rest of the task. If you find the game runs slowly when using the in-browser machine, you can always install Cetus on your machine and do the task from there, following the indications given below.

*Installing Cetus on Firefox (Click to read)*

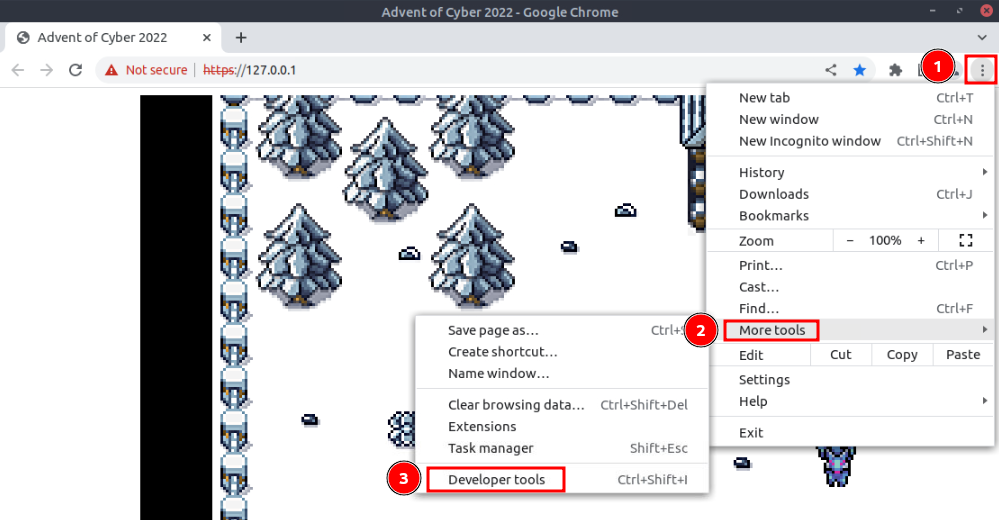
*Installing Cetus on Chrome (Click to read)*

Accessing Cetus

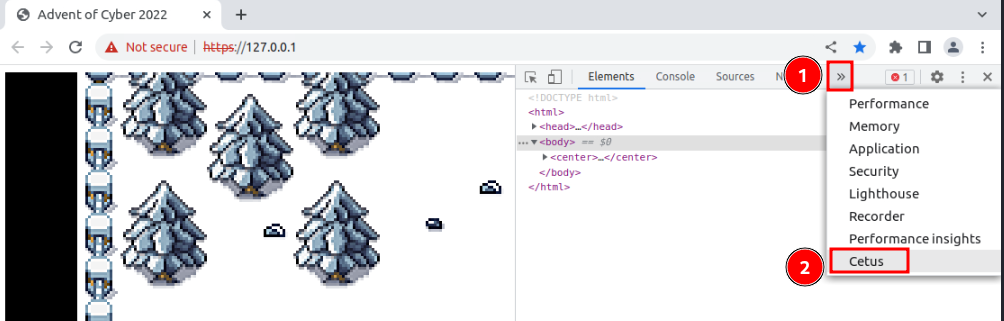
To open the game, go to your deployed machine and click the "Save Elf McSkidy" icon on the desktop. This will open Google Chrome with Cetus already loaded for you.



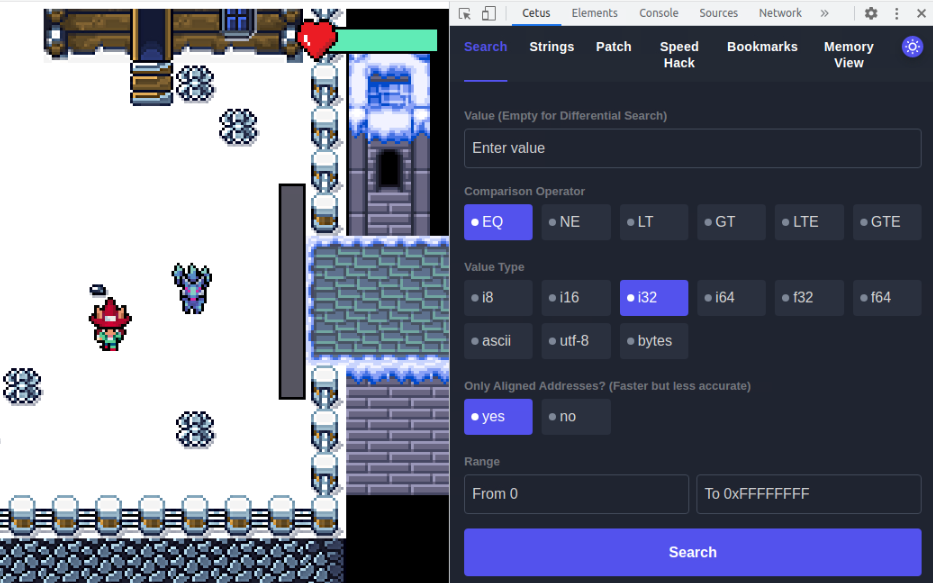
To find Cetus, you need to open the Developer tools by clicking the button on the upper-right corner of Chrome, as shown in the figure below:



Cetus is located in one of the tabs there:



With Cetus open, hit the refresh button to reload the game. If you installed Cetus on your machine, you can find the game at <https://MACHINE_IP/>. Cetus should detect the web assembly game running and show you the available tools:

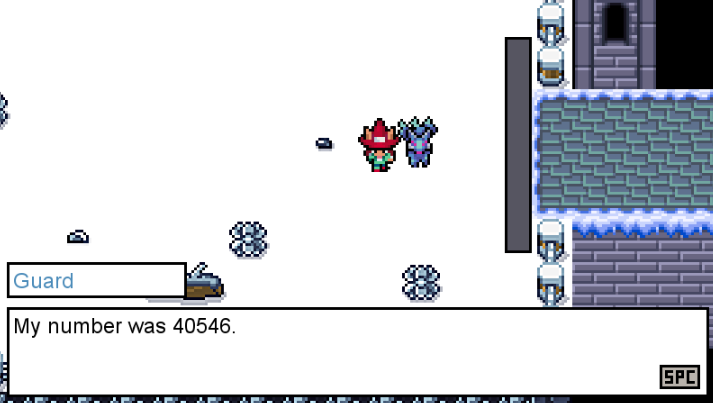


**Note:** If Cetus shows the "Waiting for WASM" message, just reload the game, and the tools should load.

Guess the Guard's Number

If you walk around the game, you will find that the guard won't let you leave unless you guess a randomly generated number. At some point, the game must store this number in memory. Cetus will allow us to pinpoint the random number's memory address quickly.

As a first step, talk to the guard and try to guess the number randomly. You probably won't guess it first try, but take note of the guard's number.



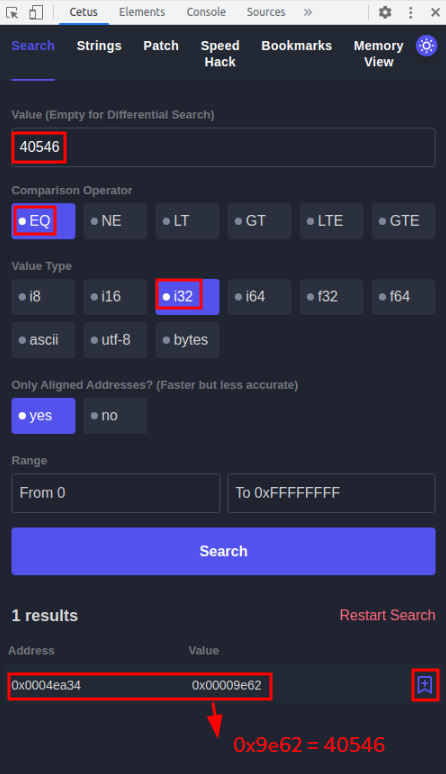
You can use Cetus to find all the memory addresses used by the game that match the given value. In this case, the guard's number is probably a regular integer, so we choose i32 (32-bit integer) in Value Type.

Cetus also allows you to search for numbers with decimals (usually called floats), represented by the f32 and f64 types, and for strings encoded in ascii, utf-8 or bytes. You need to specify the data type as part of your search because, for your computer, the values 32 (integer) and 32.0 (float) are stored in different formats in memory.

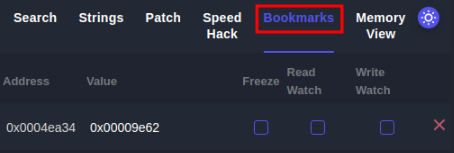
We will use the EQ comparison operator, which will search for memory addresses which content is equal to the value we input. Note that you can also search values using any of the other available operators. For reference, this is what other operators do:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| EQ | Find all memory addresses with contents that are **equal** to our inputted value. |
| NE | Find all memory addresses with contents that are **not equal** to our inputted value. |
| LT | Find all memory addresses with contents that are **lower than** our inputted value. |
| GT | Find all memory addresses with contents that are **greater than** our inputted value. |
| LTE | Find all memory addresses with contents that are **lower than or equal to** our inputted value. |
| GTE | Find all memory addresses with contents that are **greater than or equal** to our inputted value. |

Since the guard uses a random number, you will likely find the memory address on the first try. Once you do, click the bookmark button on the right of the memory address:

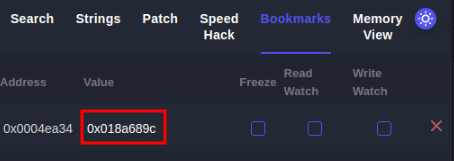


You can then go to bookmarks to see your memory addresses:



Note that Cetus uses hexadecimal notation to show you the numbers. If you need to convert the shown numbers to decimal, you can use [this website](https://www.rapidtables.com/convert/number/hex-to-decimal.html).

With Cetus on the bookmarks tab, talk to the guard again and notice how the random number changes immediately. You can now guess the number:



Convert the number from hexadecimal to get the guard's number (0x005c9d35 = 6069557). You defeated the guard (sort of)!

**Note:** You can also modify the memory address containing the random number from the bookmarks tab. Try restarting the game and changing the guard's number right before the guard asks you for your number. You should now be able to change the guard's number at will!

Getting through the bridge

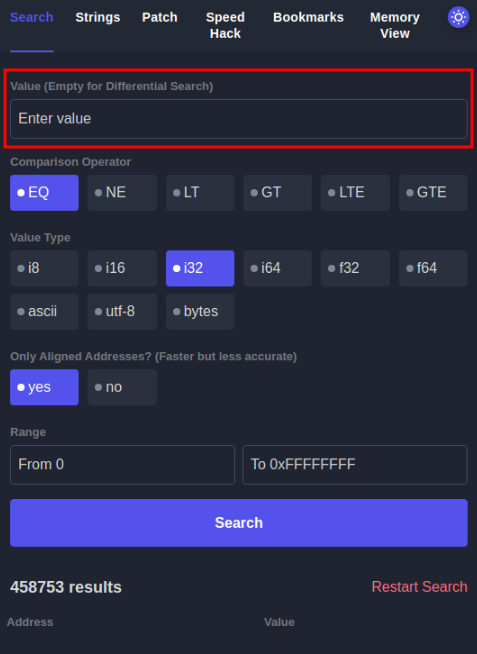
You are now out of your cell, but you still have to overcome some obstacles. Can you figure out how?



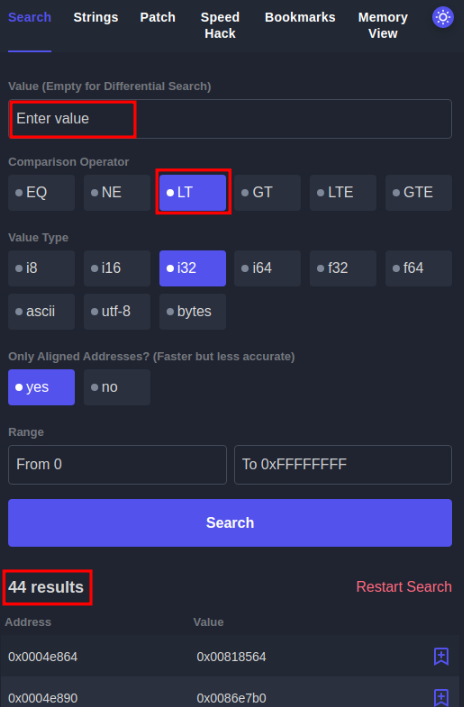
While you are wondering what other data in memory could be changed to survive the bridge, Elf Recon McRed tells you that he read about **differential search**. Differential Search, he said, allows you to run successive searches in tandem, where each search will be scoped over the results of the last search only instead of the whole memory space. Elf Recon thinks this might be of help somehow.

To help you better understand, he used the following example: suppose you want to find an address in memory, but you are not sure of the exact value it contains, but you can, however, manipulate it somehow by doing some actions in the game (you could manipulate the value of your position by moving, for example). Instead of doing a direct search by value as before, you can use differential search to look for memory positions based on specific **variations on the value**, rather than the value itself.

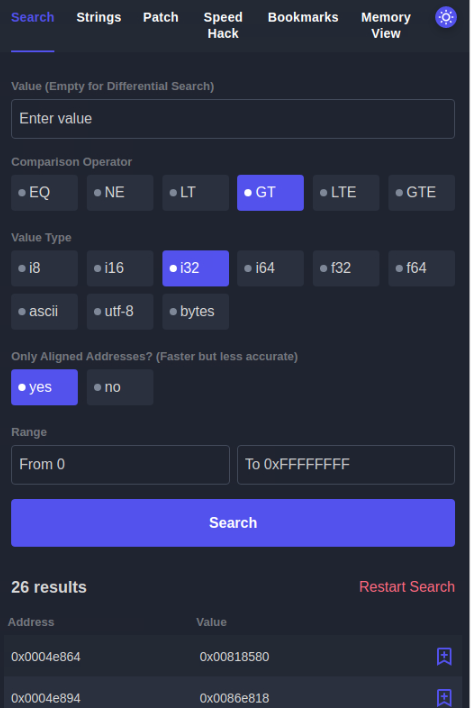
To start the differential search mode, your first search needs to be done with an empty value.



This will return the total number of memory addresses mapped by the game, which is 458753 in the image above. Now, suppose you want to know which memory addresses have decreased since the last search. You can run a second search using the LT operator without setting a value to search:



The result above tells us that only 44 memory positions of the total of 458753 have decreased in value since the last search. You can of course, continue to do successive searches. For example, if you now wanted to know which of the 44 resulting memory addresses from the first search have increased their value, you could simply do another search with the GT operator with no value again.



The result tells us that from the 44 memory addressed from the last search, only 26 have increased in value. If you are searching for a particular value, you can continue to do more searches until you find the memory address you are trying to get.

Armed with this knowledge, can you identify any parameters you'd like to search on memory to allow you to cross the bridge? The elves surely hope you do, as getting McSkidy out of the game now depends on you!