Facoltà di Scienze e Tecnologie

Corso di Laurea Magistrale in Sicurezza informatica

Relazione del progetto Richkware

Framework per la creazione di malware per Windows

Richkware
(*github.com/richkmeli/Richkware*)

Candidato:
Riccardo Melioli
Matricola 247967

Anno Accademico 2016–2017

Indice

Elenco delle figure

Capitolo 1

Descrizione

1.1 Architettura

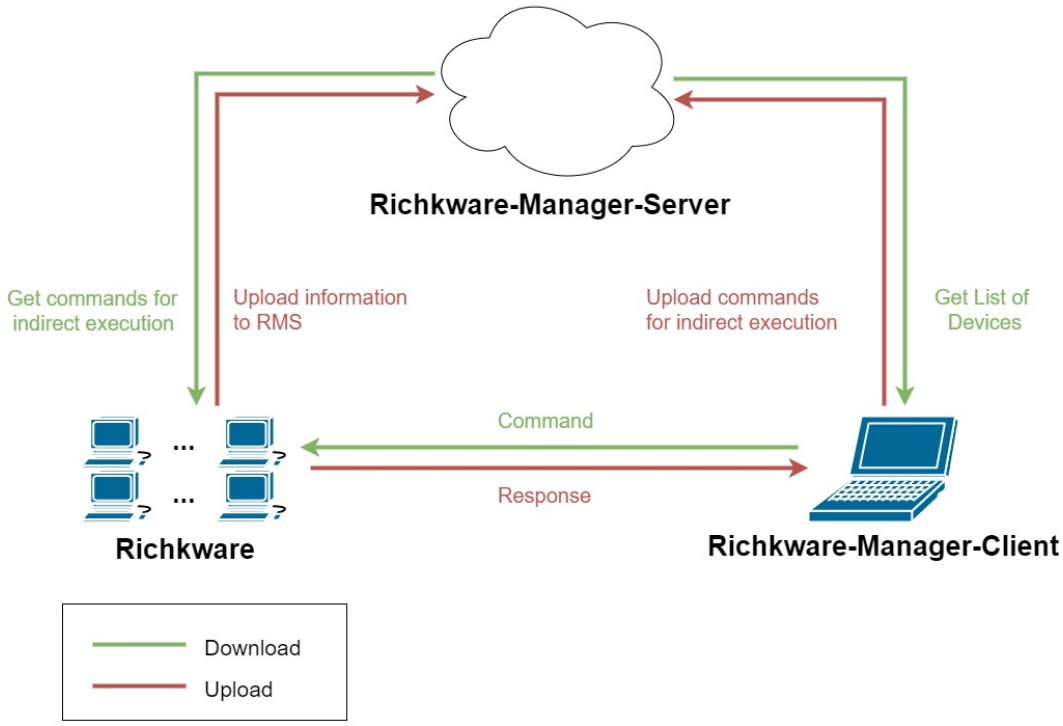


Figura 1.1: Architettura

1.1.1 Richkware

Richkware consiste in una libreria di funzioni, relative alla sicurezza del sistema operativo e delle reti, utilizzabili per la creazione di un applicativo malevolo(malware).

La composizioni di tali funzioni, secondo diverse logiche permettono all'applicativo di poter assumere comportamenti associabili ai seguenti tipi di malware:

- **Virus**: software che si diffonde copiandosi all'interno o sostituendosi ad altri programmi, o in una particolare locazione del sistema operativo, cercando di essere eseguito ogni volta che il file infettato viene aperto.
- **Worms**: software che modifica il computer che infetta, in modo da essere eseguito ogni volta che si avvia la macchina e rimanere attivo finché non si spegne il computer o non si arresta il processo corrispondente. Il worm tenta di replicarsi sfruttando Internet in diversi modi.
- **Bot**: o zombie o roBOT, consentono ai loro creatori di controllare il sistema da remoto. Più BOT tra loro formano una BOTnet, ed i controllori(BOTmaster) di tale rete possono in questo modo sfruttare i sistemi compromessi per scagliare attacchi distribuiti del tipo distributed denial of service (DDoS) contro qualsiasi altro sistema in rete oppure compiere altre operazioni illecite.
- **Spyware**: software che collezionano informazioni sugli utenti senza che loro lo sappiano.
- **Keylogger**: software in grado di intercettare tutto ciò che un utente digita sulla tastiera del proprio, o di un altro computer. In pratica controllano e salvano la sequenza di tasti che viene digitata da un utente.
- **Scareware**: software dannosi o comunque di limitata utilità con l'obiettivo di intimorire l'utente.

Essendo una libreria di funzioni, per ottenere i comportamenti precedentemente elencati, si necessita di creare un "main", che rappresenterà come verranno composte le funzioni e con quali parametri specifici.

1.1.2 Richkware-Manager-Server

Servizio web per la gestione degli host, cioè le varie instanze di Richkware presenti. Memorizza in un database SQL tutte le informazioni relative al malware:

- **Name**: nome del dispositivo in cui è presente il malware
- **IP**: indirizzo IP da cui si connette il malware, questo dato è particolarmente utile perché fornisce dati sulla connessione utilizzata.
- **Server Port**: porta TCP aperta per permettere la connessione da remoto e per poter eseguire comandi o altre funzionalità sulla macchina infetta
- **Last Connection**: ultima data e ora in cui il malware ha contattato il server.
- **Encryption Key**: chiave di crittografia generata lato server, con cui il malware critterà dati e canali di comunicazione.

1.1.3 Richkware-Manager-Client

Client di Richkware-Manager-Server, ottiene la lista di tutti gli host dal server e permette di inviare comandi da eseguire sulla macchina infetta mediante canale sicuro.

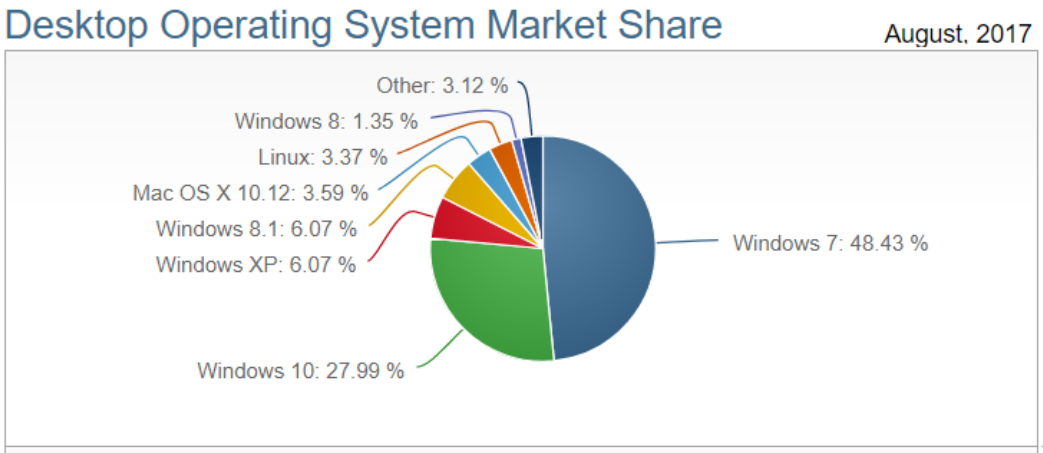
1.2 Obiettivo

L'obiettivo di tale progetto è la creazione di una libreria, a scopo didattico, che permetta lo sviluppo di qualsiasi tipo di malware, in modo versatile e semplice. La semplicità ricercata permette inoltre di creare sistemi più complessi ed "intelligenti".

1.3 Sistema operativo target

Il progetto è stato sviluppato per il sistema operativo Microsoft Windows come obiettivo degli attacchi, questo per motivi di quantità di vulnerabilità solitamente presenti in questo sistema durante l'anno, che potrebbero essere sfruttate per ottenere maggiori funzionalità.

Inoltre Microsoft Windows risulta essere il sistema operativo più diffuso attualmente, permettendo perciò al malware di raggiungere più persone possibili.



OPERATING SYSTEM	TOTAL MARKET SHARE
Windows 7	48.43%
Windows 10	27.99%
Windows XP	6.07%
Windows 8.1	6.07%
Mac OS X 10.12	3.59%
Linux	3.37%
Windows 8	1.35%
Mac OS X 10.11	1.09%
Mac OS X 10.10	0.72%
Windows Vista	0.46%
Windows NT	0.31%
Mac OS X 10.9	0.26%
Mac OS X 10.6	0.09%
Mac OS X 10.8	0.09%
Mac OS X 10.7	0.07%
Mac OS X 10.13	0.02%
Mac OS X 10.5	0.01%
Windows 2000	0.00%
Mac OS X 10.4	0.00%

Figura 1.2: OS share, fonte: netmarketshare.com

Un altro aspetto che si è valutato nella scelta del sistema operativo target è la possibile diffusione del malware, uno scenario tipico in Windows è l'installazione di crack per ottenere gratuitamente servizi che sarebbero invece a pagamento.

Essendo stato sviluppato per Windows si sono utilizzate API che permettono di interagire a basso livello con il sistema, rendendo più difficile il rilevamento per i sistemi di difesa.

1.4 Linguaggio di programmazione utilizzato

Il progetto è stato sviluppato nei linguaggi C++ e Java, nello specifico:

- **Java**: Richkware-Manager-Server e Richkware-Manager-Client
- **C++**: Richkware

Il C++ per il malware è stato scelto perché che essendo un linguaggio compilato viene eseguito direttamente dal sistema operativo e non da un interprete come nel caso dei linguaggi interpretati, dato che tali interpreti talvolta introducono ulteriori livelli di sicurezza ed inoltre potrebbero non risultare presenti su un eventuale macchina target, riducendo, in modo considerevole, il numero di macchine infettabili.

Sono state utilizzate funzioni e classi fornite dalle seguenti librerie:

- **C++ Standard Library**: stdio.h, stdlib.h, conio.h, ctime, string, ...
- **Microsoft**: windows.h, winable.h, winsock2.h, ws2tcpip.h, ...

L'utilizzo delle librerie fornite da Microsoft per Windows ha permesso un integrazione a basso livello e completa con il sistema operativo, permettendo di sviluppare funzionalità che rendono il malware più completo e versatile per varie applicazioni.

1.5 Software Open-source

Il codice sorgente del malware, del client e del server è open-source, per fornire, ai sistemisti o agli addetti alle reti, uno strumento, del quale possano avere completa conoscenza dell'implementazione, e che possano utilizzarlo nelle fasi di testing di sistemi di sicurezza.

Inoltre un software open-source rende possibile a eventuali programmatori esterni di interagire nello sviluppo del software, aggiungendo miglioramenti o segnalando errori.

Gli indirizzi delle repository dove sono contenuti i codici sorgenti sono:

- **Richkware**: https://github.com/richkmeli/Richkware
- **Richkware-Manager-Server**: https://github.com/richkmeli/Richkware-Manager-Server
- **Richkware-Manager-Client**: https://github.com/richkmeli/Richkware-Manager-Client

1.6 Compilazione ed esecuzione

Il malware si può compilare con MinGW da Windows o da ambienti Linux effettuando cross compiling.

Dopo aver creato il main, contenente le funzioni del framework volute, utilizzando il comando 1.1, si crea il file eseguibile.

Vi è la possibilità di utilizzare, come tool di automazione della fase di compilazione:

- **Make**
- **CMake**

Listing 1.1: Compilazione Libreria con il tool Make

```
make
```

Nel Makefile sono raccolti i seguenti passaggi di compilazione e linking.
Il comando 1.2 è un esempio di compilazione, quindi creazione del file oggetto, di uno dei file della libreria.
I parametri utilizzati sono:

- **-c**: Compilare o assemblare i file di origine, ma non linkarli
- **-03**: Generazione di codice ottimizzato, "3" indica il valore di ottimizzazione in una scala da 0 a 3. il codice ottimizzato rende più difficile un eventuale reverse engineering
- **-o**: Mette l'output nel file che segue il parametro.

Listing 1.2: Compilazione Libreria

```
g++ -O3 -c -o Richkware.o Richkware.cpp
```

Con il comando 1.3 effettua il linking dei file oggetto e genera il file eseguibile. Si considera anche presente il file oggetto del "main". I parametri utilizzati sono:

- **-lws2_32**: linka la libreria Windows Sockets API (WSA), nel quale sono definite le funzioni per utilizzare i servizi di rete di Windows.
- **-static-libgcc -static-libstdc++**: linka staticamente(inserisce all'interno del eseguibile, rendendolo standalone) le librerie standard.
- **-o**: Mette l'output nel file che segue il parametro.

Listing 1.3: Linking e creazione eseguibile

```
g++ -static-libgcc -static-libstdc++ -o Richkware.exe
   Richkware.o ... [other file.o]... main.o -lws2_32
```

L'esecuzione del malware avviene, semplicemente eseguendo il file eseguibile con estensione .exe.

Capitolo 2

Funzionalità

Le funzionalità offerte dal framework verranno elencate e illustrate in base al nome dei metodi della classe, raggruppate per tipologia.

2.1 Rete

le funzionalità di rete sono presenti nel file network.h, dove vi sono le seguenti classi:

- **Server**: permette la creazione di un server su una porta TCP, che supporta la crittografia.
- **Network**: permette di comunicare, ricevere e inviare informazioni, in chiaro o in modo crittato, con qualsiasi dispositivo, in particolare, ha metodi specifici per comunicare con Richkware-Manager-Server e Richkware-Manager-Client.
- **Device**: modello, identificante le informazioni da scambiare con Richkware-Manager-Server

Nel file protocol.h vi è il protocollo di comunicazione utilizzato da RMC e Richkware per permettere all'utilizzatore di RMC di poter interagire con la macchina dov'è installata l'istanza di Richkware

2.1.1 Server.Start()

Funzione che permette al malware di creare un server su una porta TCP, nel quale riceverà da eventuali client connessi, comandi da eseguire sulla macchina e restituendo il risultato.

La funzione è multithread e può gestire contemporaneamente 8 client diversi. La funzione identificante il thread è **ClientSocketThread** e come argomento viene passato l'identificativo del client in questione con cui comunicare; il thread ha il compito di ricevere informazioni dal client in questione, che invia dati sulla porta, precedentemente scelta, mediante il metodo **recv**. Tali dati, vengono parsati ed interpretati come comandi ed eseguiti direttamente sul sistema operativo, l'output verrà reindirizzato su un file di testo, che verrà, successivamente aperto ed inviato al client, come risposta del comando richiesto.

I parametri della funzione StartServer sono:

- **port**: porta su cui aprire il socket
- **encrypted**: se abilitare la crittografia

2.1.2 Network.RawRequest()

Funzione che permette al malware di inviare richieste su una porta di un server, ritornando una stringa contenente la risposta. Questa funzione è particolarmente versatile, dato che permette di inviare su una qualsiasi porta una quantità di dati, ad esempio una richiesta HTTP GET:
"$GET/HTTP/1.1\r\n\r\n$"

I parametri della funzione RawRequest sono:

- **serverAddress**: nome Host o indirizzo IP a cui inviare i dati
- **port**: porta a cui inviare i dati
- **request**: stringa di dati da inviare

2.1.3 Network.GetEncryptionKeyFromRMS()

Ottiene la chiave di crittografia generata dal server, per quella specifica istanza di malware, la comunicazione iniziale per creare l'entry nel database del server, viene effettuata in modo crittato con una chiave pre-shared tra RMS e Richkware.

La password ottenuta viene salvata in locale per evitare di ricontattare il server, ad ogni avvio del malware, nel caso in cui il file non venga più trovato nel file system, allora viene ricontattato il server per riottenere la chiave.

I parametri della funzione GetEncryptionKeyFromRMS sono:

- **serverAddress**: nome Host o indirizzo IP da cui ricevere la chiave
- **port**: porta a cui connettersi

2.1.4 Network.UploadInfoToRMS()

Contatta il server, di tipo Richkware-Manager-Server, e gli invia i dati relativi alla macchina, in particolare nome della macchina e porta su cui è attivo il server; non si distingue se il server sia stato attivato o meno con la crittografia perche in automatico viene riconosciuto dal protocollo di comunicazione.

I parametri della funzione GetEncryptionKeyFromRMS sono:

- **serverAddress**: nome Host o indirizzo IP a cui inviare i dati
- **port**: porta a cui connettersi

2.1.5 Protocollo di comunicazione tra Richkware e RMC

Nel file protocol.h vi è il protocollo di comunicazione utilizzato da RMC e Richkware per permettere all'utilizzatore di RMC di poter interagire con la macchina dov'è installata l'istanza di Richkware.

Le richieste ricevute dal server in Richkware, presente nella classe Server vengono mandate ad un Dispatcher, che smista la richiesta in base a un certo codice, esegue la richiesta e ritorna la risposta, in modo che il server possa comunicarla al client connesso. Il dispatcher è implementato nel seguente modo:

Listing 2.1: Dispatcher comandi

```
... rimozione delimitatori

switch (commandID) {
        case 0:
                response = "***quit***";
                break;
        case 1:
                response = CodeExecution(command);
                break;
        case 2:
                //...
                break;
        default:
                response = "error: Command ID not found\n";
}
```

La sintassi di una richiesta è la seguente:

Listing 2.2: Sintassi Protocollo di comunicazione Richkware e RMC

```
[[1]]ls
```

Il comando precedente, avendo il parametro 1, significa che si richiede l'esecuzione della stringa che segue come un comando da shell, quindi "ls", verrà eseguito dalla shell di Windows e la risposta, cioè quello che verrebbe stampato sulla shell in seguito a quella richiesta viene mandato al client.

2.2 Sistema

le funzionalità riguardanti il sistema sono presenti nel file storage.h, dove vi sono le seguenti classi:

- **SystemStorage**: permette il salvataggio nel sistema di informazioni, in particolare nel registro o nelle cartelle temporanee.
- **Session**: astrae il concetto di salvataggio dati, rendendo disponibile all'implementatore di Richkware, la possibilità di salvare dati non preoccupandosi di percorsi di salvataggio o nomi file, semplicemente permettendo di salvare dati, cancellarli e recuperarli mediante una chiave identificante, come in una struttura dati a mappa/dizionario.

2.2.1 SystemStorage.SaveValueReg() e SystemStorage.LoadValueReg()

Funzioni che permettono di salvare o recuperare delle informazioni su una chiave del registro di sistema.

Le funzioni si comportano in modo diverso a seconda se si hanno i permessi di amministratore per l'esecuzione del malware, riuscendo a salvare lo stato anche nel caso non si abbiano i permessi, quindi solamente per l'utente in cui è in esecuzione il malware.

I parametri della funzione SaveValueReg sono:

- **path**: percorso del registro di sistema in cui salvare i dati
- **key**: nome della chiave del registro di sistema
- **value**: valore da salvare

I parametri della funzione LoadValueReg sono:

- **path**: percorso del registro di sistema in cui recuperare i dati
- **key**: nome della chiave del registro di sistema

void SaveValueToFile(const char* value, const char* path = NULL); std::string LoadValueFromFile(const char* path = NULL);

2.2.2 SystemStorage.SaveValueToFile() e SystemStorage.LoadValueFromFile()

Funzioni che permettono di salvare o recuperare delle informazioni su un file nel sistema.

I parametri della funzione SaveValueToFile sono:

- **name**: nome del file
- **value**: valore da salvare
- **path**: percorso dove salvare il file, di default è la cartella temporanea del sistema.

I parametri della funzione LoadValueFromFile sono:

- **name**: nome del file
- **path**: percorso dove recuperare il file, di default è la cartella temporanea del sistema.

2.2.3 SystemStorage.Persistance()

Funzione che si occupa dell'installazione permanente del malware all'interno del sistema operativo. Le fasi sono le seguenti:

1. **Rinomina**: rinomina il nome dell'eseguibile, in modo da non essere facilmente identificabile, quindi con il nome di un servizio di Windows, ad esempio "winresumer.exe".

2. **Propagazione eseguibili**: gli eseguibili rinominati, vengono dislocati, sia nelle cartelle di sistema, sia nelle cartelle temporanee.

3. **Esecuzione all'avvio**: si salva nel registro, mediante la funzione SaveValueReg (2.2.2), le voci riguardanti i file, facendo in modo che all'avvio vengano eseguiti i file del malware.

2.2.4 Session.SaveSession() e Session.LoadSession()

Funzioni che permettono al malware di salvare o recuperare il suo stato(sessione), all'interno del sistema, in modo tale che al riavvio della macchina possa ripristinare precedenti configurazioni.

Le funzioni utilizzano un attributo della classe, di tipo **map**, con chiave e valore di tipo **string**, nel quale tutte le funzioni della classe possono inserire le proprie informazioni, questo per permettere alle singole funzioni di operare indipendentemente e separare i compiti, riguardo il salvataggio dei dati presenti nella variabile e l'inserimento di tali dati in essa.

Le informazioni vengono salvate nel registro di sistema, mediante la funzione SaveValueReg (2.2.2) e nella cartella temporanea, mediante la funzione SaveValueToFile, con la logica di una struttura dati di un mappa(chiave-valore) sotto una voce difficilmente individuabile all'occhio "non esperto" e il suo contenuto viene crittato.

2.2.5 BlockApps

Classe che permette di che permette di bloccare e sbloccare certe applicazioni nel sistema operativo, mediante l'inserimento e la rimozione del nome dell'applicazione da bloccare nell'attributo della classe di tipo **SharedList**.

La classe BlockApps è formata da un'attributo di classe di tipo **SharedList**, cioè una STD list, nel quale sono gestiti gli accessi concorrenti, ed è utilizzata per inserire le applicazioni da bloccare dal thread **BlockAppsThread**. L'utilizzo di tale meccanismo rende dinamico l'inserimento, di eventuali applicazioni da parte dei thread, pur mantenendo il concetto di thread safety, ed inoltre questo permette di separare la logica di inserimento nella lista, dalla logica di bloccaggio dei programmi il quale nome corrisponde a quello presente nella lista.

Il metodo **start()** della classe **BlockApps** avvia un thread in modo che il lavoro svolto non gravi sulle risorse del sistema. Il metodo **Stop()** ha il ruolo di sospendere il funzionamento di tale thread.

2.2.6 Richkware.IsAdmin() e Richkware.RequestAdminPrivileges()

Funzioni che gestiscono la verifica e la richiesta dei privilegi di amministratore in Windows.

2.2.7 Richkware.StealthWindow()

Funzione che permette di rendere non visibile una finestra scelta nel sistema, pensata principalmente per la finestra del malware, limitando molto la possibilità di essere individuata da un utente.

Per motivi di versatilità si è reso possibile rendere non visibile una qualsiasi finestra, aumentando i possibili impieghi della funzione.

I parametri della funzione StealthWindow sono:

- **window**: nome della finestra che non si vuole rendere visibile

2.2.8 Richkware.OpenApp()

Funzione che permette di aprire un programma qualsiasi. Ad esempio:

- **OpenApp("notepad.exe")**: apre il notepad
- **OpenApp("www.google.com")**: apre il browser di default su quella specifica pagina

I parametri della funzione OpenAPP sono:

- **app**: nome del programma/URL da aprire

2.2.9 Richkware.Keylogger()

Funzione che consente al malware di rilevare qualsiasi pressione effettuata su tastiera o sui pulsanti del mouse, mediante il metodo della libreria di Windows **GetAsyncKeyState**, salvando il contenuto su un file di testo.

La funzione **Keylogger** avvia il thread **KeyloggerThread**, per non gravare sulle risorse del sistema, che si occupera di registrare quali tasti sono digitati e salvarli sul file di testo.

I parametri della funzione Keylogger sono:

- **fileName**: nome del file di testo in cui inserire i tasti digitati

2.3 Crittografia

le funzionalità di crittografia sono presenti nel file crypto.h, dove vi sono le seguenti classi:

- **Crypto**: classe che astrae il resto dell'applicazione dall'algoritmo utilizzato, cioè le altre classi della libreria effettueranno chiamate ai suoi metodi Encrypt e Decrypt, senza preoccuparsi di ulteriori dettagli sull'algoritmo e sull'inizializzazione di esso.
- **RC4**: classe che permette l'utilizzo dell'algoritmo di cifratura a flusso, a chiave simmetrica, RC4.
- **Blowfish**: classe che permette l'utilizzo dell'algoritmo di cifratura a blocchi, a chiave simmetrica, Blowfish.
- **Vigenere**: classe che permette l'utilizzo dell cifrario polialfabetico di Vigenere.

Inoltre vi sono le funzioni riguardanti la codifica delle stringhe in:

- **Base64**: è un sistema di numerazione posizionale che usa 64 simboli.
- **Hexadecimal**: è un sistema di numerazione posizionale che usa 16 simboli.

2.3.1 Crypto.Encrypt() e Crypto.Decrypt()

Funzioni che effettuano rispettivamente la crittazione e la decrittazione di stringhe passate come parametro.

la chiave simmetrica utilizzata per effettuare la crittazione o decrittazione è quella con cui è stata inizializzata l'istanza della classe Crypto.

I parametri delle funzioni Encrypt() e Decrypt() sono:

- **input**: stringa da crittare o decrittare

2.4 Varie (scareware)

2.4.1 Richkware.RandMouse()

Funzione che genera spostamenti casuali del puntatore del mouse, all'interno della risoluzione dello schermo.

2.4.2 Richkware.Hibernation()

Funzione che iberna il computer.

Capitolo 3

Richkware-Manager-Server e Richkware-Manager-Client

In questo capitolo veranno trattati il server e il client di Richkware e quali sono le loro funzionalità.

3.1 Architettura

Il sistema è composto da 3 parti:

- **Istanza di Richkware**: programma malevolo che utilizza la libreria Richkware e ha abilitato i servizi riguardanti lo scambio di informazioni con RMS e RMC.
- **Richkware-Manager-Server**: Server ricevente i dati dall'istanza di Richkware
- **Richkware-Manager-Client**: Client mandante i comandi all'istanza di Richkware.

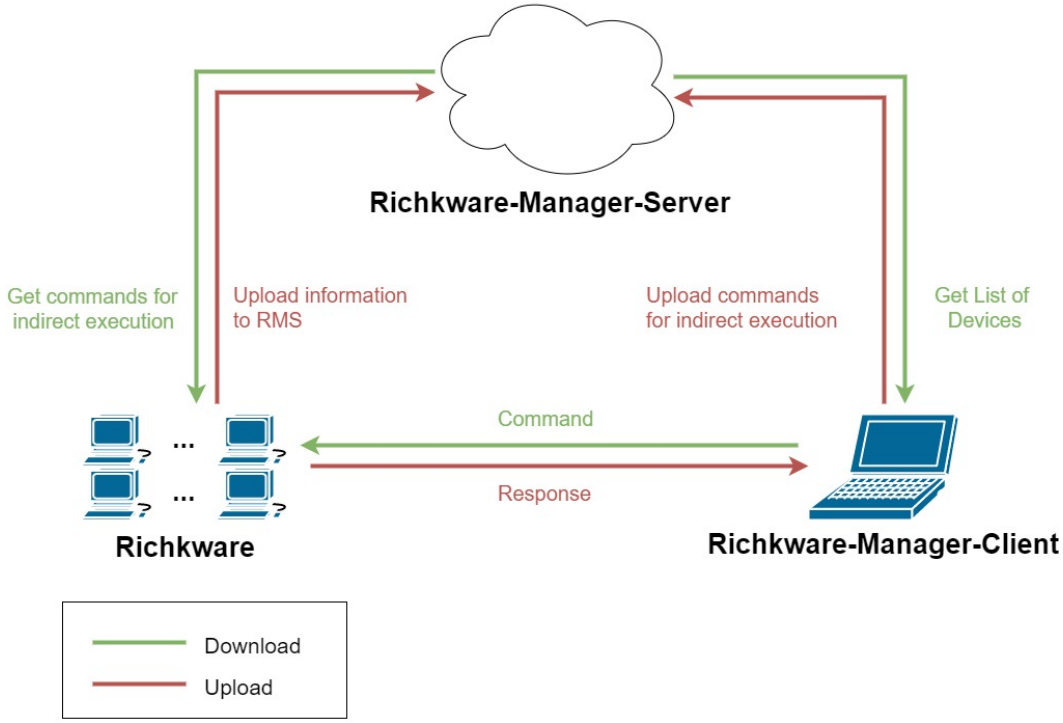


Figura 3.1: Architettura

3.2 Linguaggio di programmazione utilizzato

Richkware-Manager-Server e Richkware-Manager-Client sono stati sviluppati in Java, rendendo perciò molto versatile l'esecuzione indipendetemente dal sistema operativo utilizzato.

Sono state utilizzate le seguenti librerie esterne:

- **javax.servlet-api**: libreria che permette la comunicazione tra classi servlet e l'ambiente runtime.
- **mysql-connector-java**: libreria per l'utilizzo di database mysql
- **gson**: libreria per la conversione di oggetti Java in JSON e viceversa

3.3 Server (Richkware-Manager-Server)

Servizio web per la gestione degli host, cioè le varie instanze di Richkware presenti. Memorizza in un database SQL tutte le informazioni relative al malware:

- **Name**: nome del dispositivo in cui è presente il malware
- **IP**: indirizzo IP da cui si connette il malware, questo dato è particolarmente utile perché fornisce dati sulla connessione utilizzata.
- **Server Port**: porta TCP aperta per permettere la connessione da remoto e per poter eseguire comandi o altre funzionalità sulla macchina infetta

- **Last Connection**: ultima data e ora in cui il malware ha contattato il server.
- **Encryption Key**: chiave di crittografia generata lato server, con cui il malware critterà dati e canali di comunicazione.

Il server interagisce con Richkware, RMC e con i browser mediante le seguenti servlet:

- **DevicesList**: mostra la lista dei dispositivi in una pagina HTML.
- **DevicesListAJAJ**: mostra una stringa JSON contenente la lista dei dispositivi, supporta la crittografia della stringa JSON.
- **GetEncryptionKey**: quando una precisa istanza di Richkware contatta questa servlet, gli viene restituita la sua chiave di crittografia.
- **LoadData**: utilizzata dalle istanze di Richkware per caricare i dati sul server, cioè le informazioni specificate nell'elenco precedente.
- **RemoveDevice**: serve per rimuovere un particolare dispositivo dal database.

Inoltre è presente la JSP (devices_list_AJAJ.jsp), che permette di effettuare una chiamata asincrona alla servlet DevicesListAJAJ, per poi inserire i dati nell'HTML dal JSON ricevuto mediante delle funzioni in javascript.

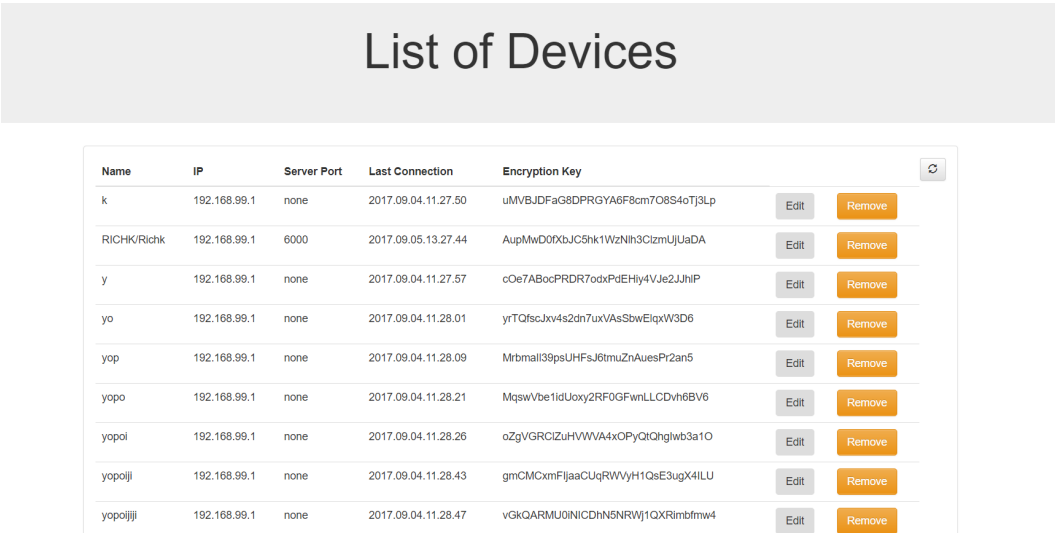


Figura 3.2: Richkware-Manager-Server, devices_list_AJAJ.jsp

3.4 Client (Richkware-Manager-Client)

Client di Richkware-Manager-Server, sviluppato secondo il pattern Model–view–controller, ottiene la lista di tutti gli host dal server e permette di inviare comandi da eseguire sulla macchina infetta mediante canale sicuro.

La View è una GUI (fig. 3.3), interamente sviluppata con la libreria Swing.

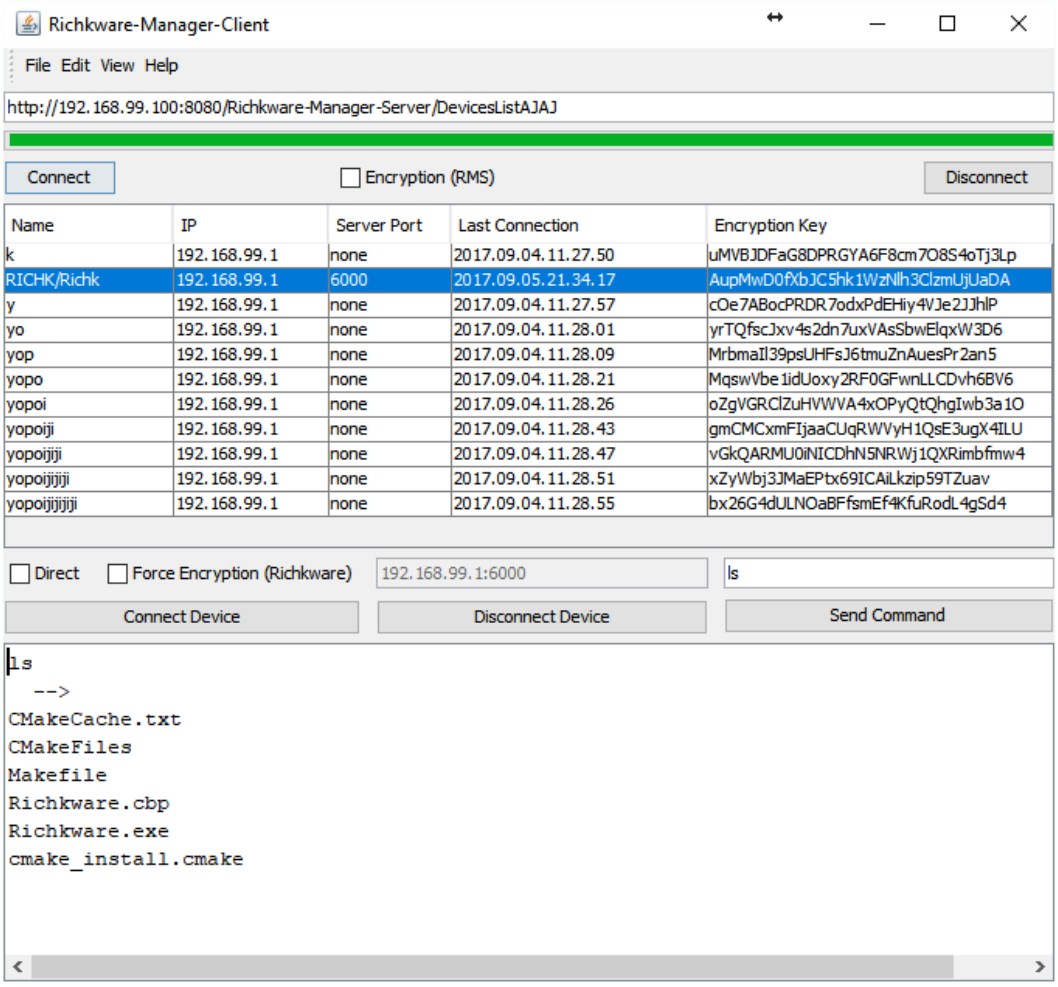


Figura 3.3: Richkware-Manager-Client GUI

Per ottenere la lista dei dispositivi dal server, RMC effettua una chiamata alla servlet DevicesListAJAJ, che ritorna un JSON contenente le informazioni di tutti i dispositivi presenti. La chiamata nel codice di RMC viene effettuata dalla classe Network, che racchiude tutte le funzionalità per la comunicazione di rete con le varie parti. Inoltre dentro tale classe, vi sono anche i metodi per l'invio di comandi all'istanza di Richkware secondo un determinato protocollo, visto nel capitolo (2.1.5).

Capitolo 4

Crittografia

In questo capitolo verrà spiegato come sia stata applicata la crittografia alle varie parti del progetto.

Principalmente ha trovato impiego nella protezione dei canali di comunicazione tra Richkware, RMC e RMS, ma anche per la protezione dei dati conservati su file o voci di registro da parte di Richkware, infatti qualsiasi informazione salvata nel sistema dall'istanza di Richkware viene crittata.

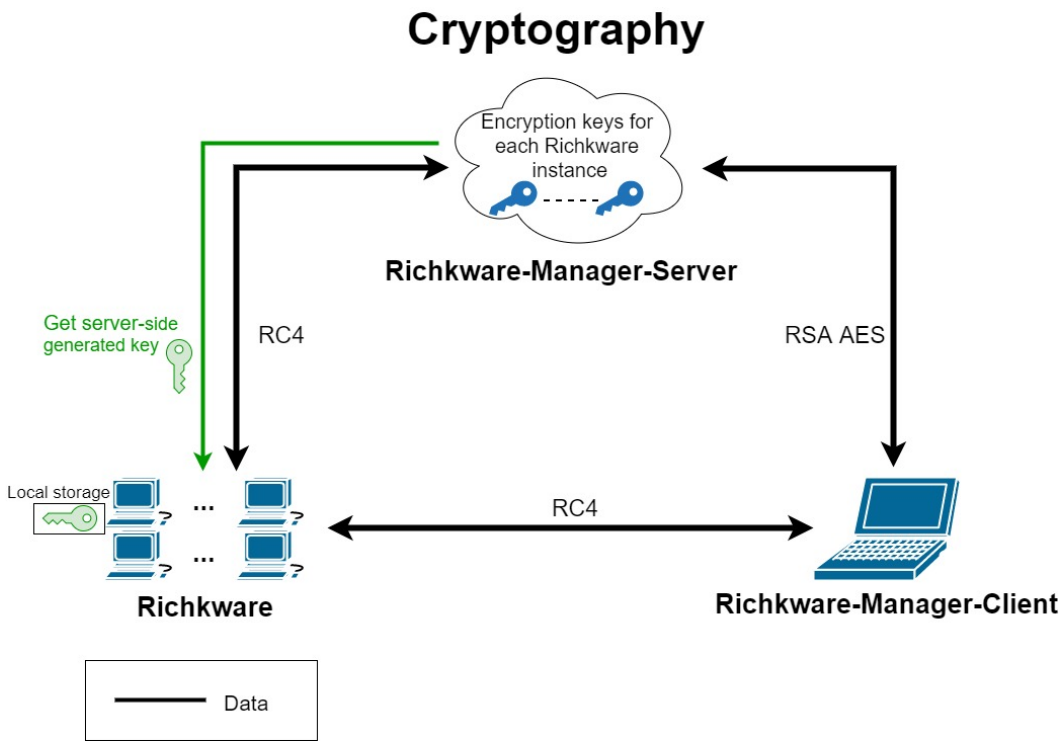


Figura 4.1: Sistema Crittografico

4.1 Chiave di crittografia

Come viene mostrato dall'immagine 4.4, l'istanza di Richkware, al suo primo avvio, può contattare, in modo crittato(con chiave pre-shared tra Richkware e RMS) il server per chiedere la chiave di crittografia per quella precisa istanza, che poi, una volta ricevuta, andrà a sostituire la chiave pre-shared dentro (hardcoded) l'istanza di Richkware. Per mantenere in memoria la chiave ricevuta dal server, quindi uno "stato" dell'applicazione in sé, viene salvata la chiave in un file locale (fig. 4.2), in modo che ai successivi avvii, se tale file è presente si andrà a leggere il contenuto del file senza bisogno di contattare nuovamente il server, per riottenere la chiave.

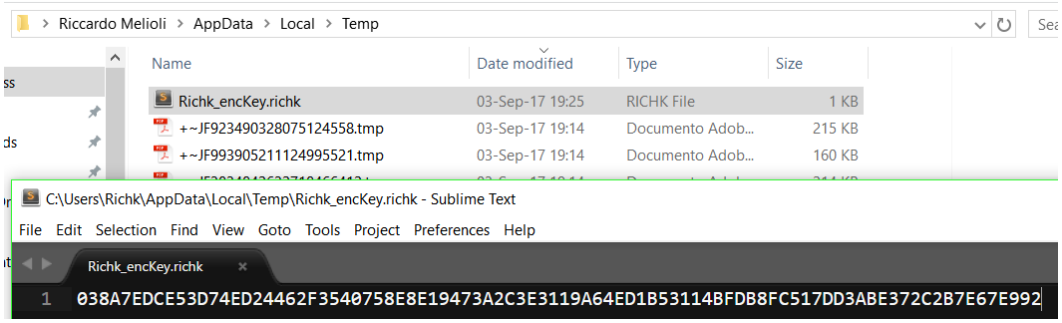


Figura 4.2: Chiave salvata localmente

La corrispondenza della chiave utilizzata su Richkware e quella presente su RMS, è fondamentale, dato che quando RMC contatta il server per avere la lista dei dispositivi e le loro informazioni, ottiene anche la chiave di crittografia, che verrà utilizzata da RMC per rendere sicuro il canale di comunicazione con l'istanza di Richkware.

4.2 Interazione tra le parti

4.2.1 RMS - RMC

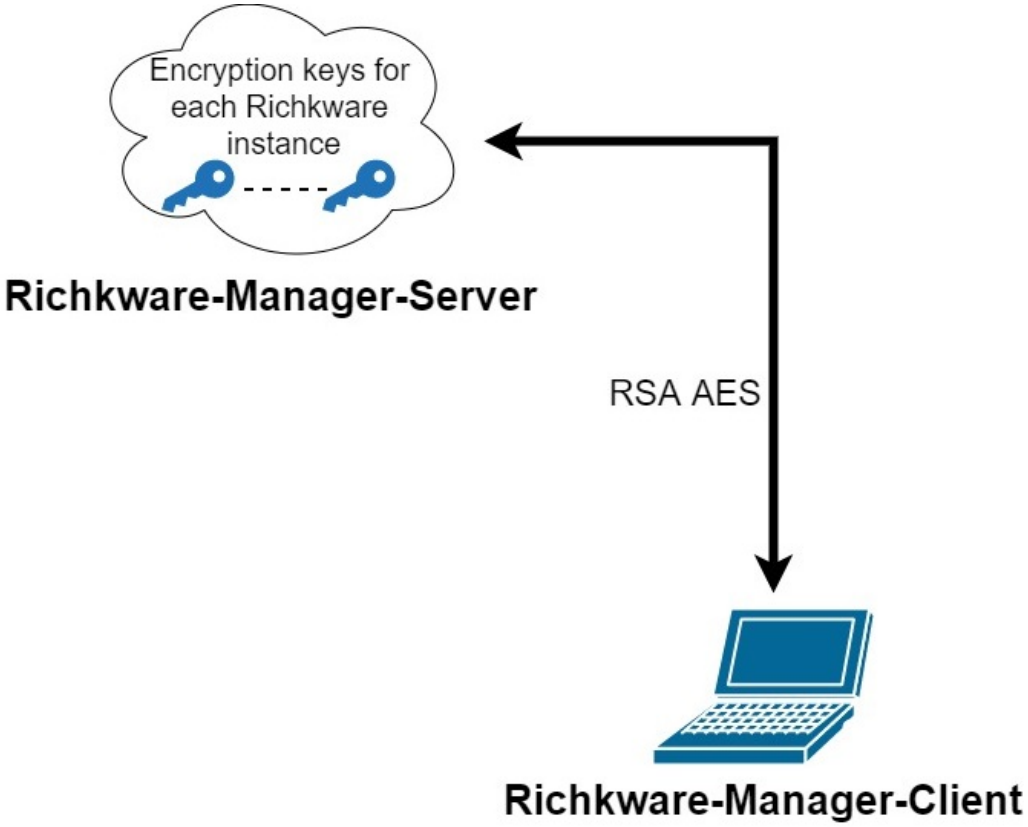


Figura 4.3: RMS - RMC

Per la comunicazione tra RMS e RMC, si utilizzano gli algoritmi di crittografia RSA e AES. RSA essendo un algoritmo asimmetrico, risulta essere più sicuro, permettendo alle parti di scambiarsi dati senza dover avere un segreto scambiato in precedenza, in questo caso verrà scambiata la chiave; Viene utilizzato per scambiare solo la chiave e non per l'intera comunicazione perché è particolarmente lento e "pesante" e comporterebbe ad un overhead eccessivo.

Ricapitolando le fasi:

1. RMC genera la sua coppia di chiavi RSA
2. RMC invia la sua **chiave pubblica RSA** a RMS
3. RMS
   (a) genera la sua coppia di chiavi RSA e la chiave AES che verrà utilizzata per crittare i messaggi.
   (b) firma la chiave AES con la sua chiave privata
   (c) critta con la chiave pubblica di RMC il pacchetto formato dalla firma e il messaggio(Chiave AES)

(d) invia a RMC il pacchetto formato da: **chiave pubblica di RMS**, e **pacchetto crittato contenente la firma e messaggio**.

4. RMC decritta con la sua privata e verifica il contenuto con la chiave pubblica di RMS.

5. RMC può utilizzare la chiave AES per decifrare i dati successivamente ricevuti

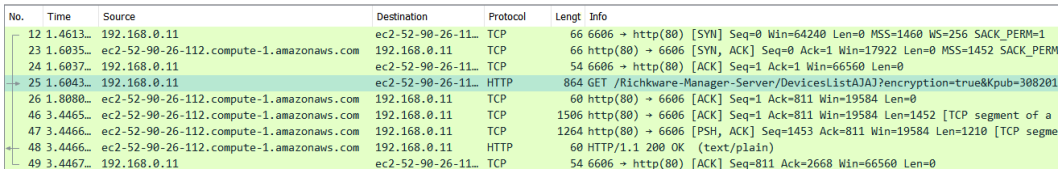


Figura 4.4: Scambio chiavi tra RMC e RMS

Listing 4.1: Pacchetto da RMC a RMS

```
GET /Richkware-Manager-Server/DevicesListAJAJ?
encryption=true&Kpub=3082012230......03010001 HTTP/1.1
User-Agent: Java/1.8.0_91
Host: rms-richk.rhcloud.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Listing 4.2: Pacchetto da RMS a RMC

```
HTTP/1.1 200 OK
Date: Sun, 03 Sep 2017 16:29:25 GMT
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=1F479C58D327620765A0D431A15DCB84; Path
   =/
Richkware-Manager-Server/; HttpOnly
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/plain
{'encryptedAESsecretKey' : '8F01D550529.....C2205EC',
        'signatureAESsecretKey' : '496D636344......773D3D',
        'kpubServer' : '3082012.......03010001',
        'data' : '33A07E.......BFDCEC'}
```

4.2.2 Richkware - RMC

Figura 4.5: Richkware - RMC

Per la comunicazione tra Richkware e RMC, si utilizza l'algoritmo RC4, particolarmente veloce e performante. RMC viene a conoscenza della chiave da utilizzare con Richkware per la comunicazione, contattando il server RMS.

Come visto nel capitolo (2.1.5) vi è un protocollo di comunicazione alla base, la crittografia si applica ad ogni pacchetto scambiato nel protocollo, tranne che all'**handshake** iniziale, appunto per sapere se Richkware supporti o meno la crittografia. Se il canale è insicuro, quindi un utente malevolo volesse inserirsi nella comunicazione e cambiare il messaggio di handshake da canale crittografato a non crittografato, si può impedire tale attacco abilitando la modalità del client "**Force Encryption**" che permette al client di ignorare l'handshake e procedere con la crittografia in ogni caso. Per utilizzare questa modalita in RMC, il server in Richkware deve aver abilitato la crittografia.

4.2.3 Richkware - RMS

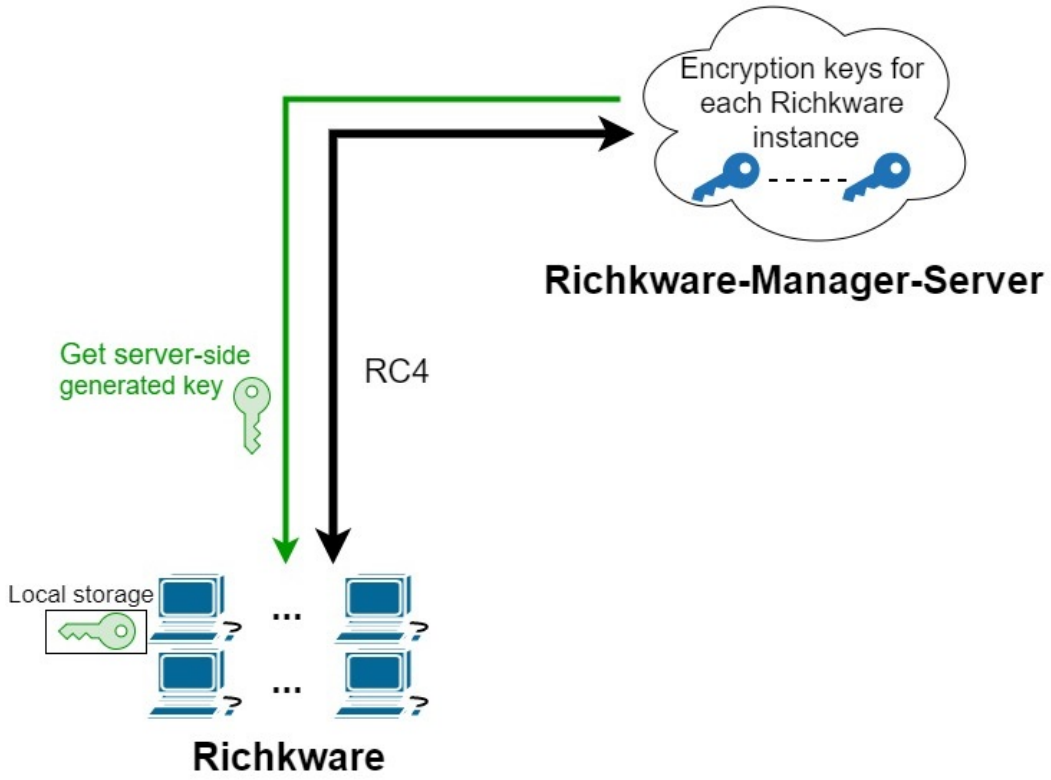


Figura 4.6: Richkware - RMS

Per la comunicazione tra Richkware e RMS, si utilizza l'algoritmo RC4, particolarmente veloce e performante. Al primo avvio Richkware utilizza una **chiave hardcoded** al suo interno, appositamente scelta dal creatore dell'istanza di Richkware, quindi presente anche sul server RMS, dopo di che, quando verrà stabilito il contatto tra Richkware e RMS, l'istanza otterrà la chiave di crittografia dal server e utilizzerà sempre quella, questo per rendere più sicuro il funzionamento ed evitare attacchi di tipo **disassembling** fatti all'eseguibile dell'istanza di Richkware.

4.3 Algoritmi utilizzati

Gli algoritmi utilizzati sono:

1. **RC4**: utilizzato principalmente da Richkware e dalle altre parti quando comunicano con lui. Questo algoritmo è stato scelto perche particolarmente performante e veloce, perciò difficilmente dovrebbe gravare sulle risorse del pc.
2. **AES**: utilizzato da RMC e RMS per la crittografia dei dati.
3. **RSA**: utilizzato da RMC e RMS per la crittografia della chiave AES scambiata
4. **SHA256**: utilizzato da RMC e RMS per la firma e la verifica dei messaggi.

5. **Base64**: utilizzato da tutte e 3 le parti per codificare i messaggi prima di inviarli, per evitare problemi di rappresentazione o troncamento.

6. **Hex**: utilizzato da tutte e 3 le parti per codificare i messaggi prima di inviarli, per evitare problemi di rappresentazione o troncamento.

Capitolo 5

Applicazioni

In questo capitolo veranno mostrati degli esempi di combinazione di funzioni offerte dalla libreria per creare dei malware con funzionalità diverse.

5.1 Esecuzione comandi da remoto

Per effettuare questo tipo di attaccato o creazione di una backdoor nel sistema operativo, si possono utilizzare le funzionalità di rete, **Server.Start()** e **Network.RawRequest()**.

Con il comando 5.1 si crea un server sulla porta 6000, in ascolto di comandi. Il server RMS su cui verranno caricate le informazioni di questa istanza di Richkware è all'indirizzo IP "192.168.99.100" sulla porta "8080".

Listing 5.1: Avvio server su porta 8000

```
int main() {
        Richkware richkware("Richk","richktest","
            192.168.99.100", "8080");

        richkware.network.server.Start("6000");
        ...
}
```

5.1.1 Client RMC

In figura 5.1 è mostrato il funzionamento lato client(attaccante), dove è utilizzato RMC, che esegue il comando **"ls"** sulla macchina dove è installato il malware, e ottiene l'output del comando.

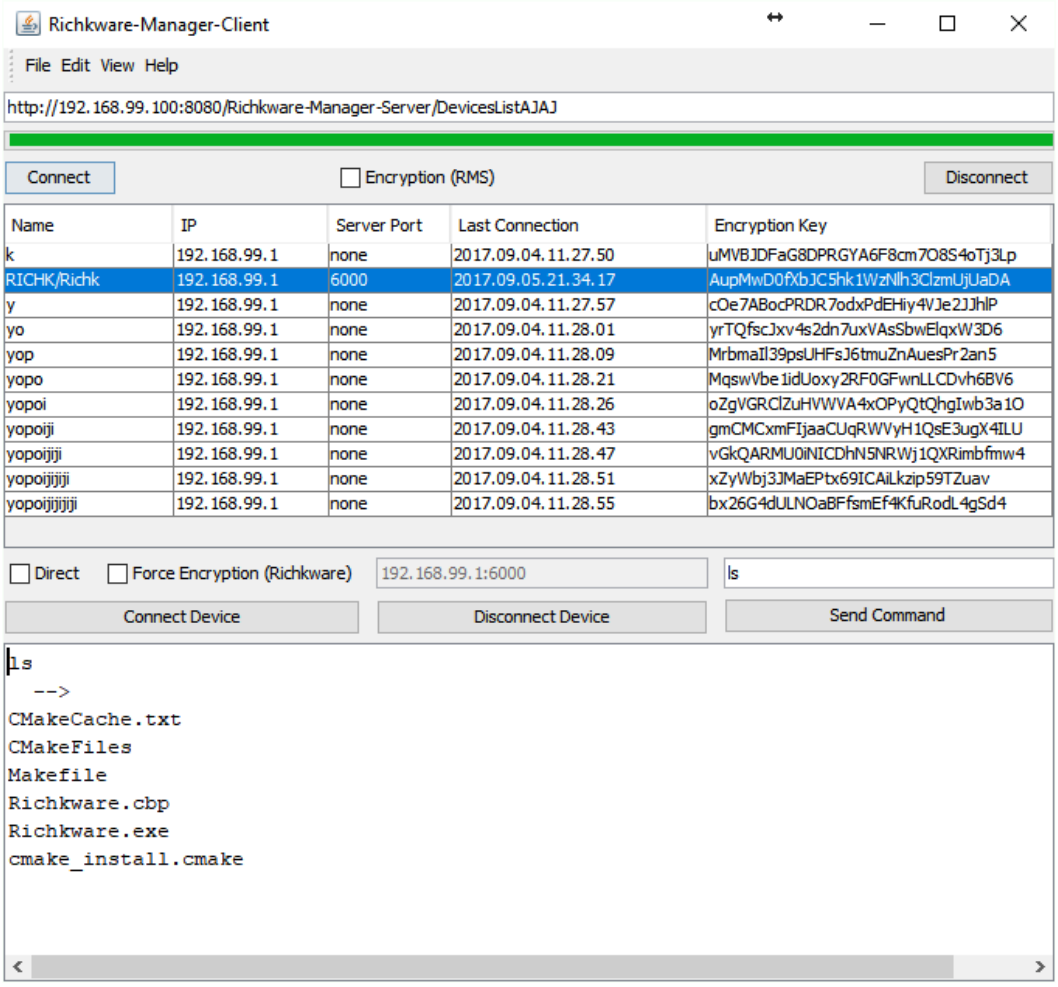


Figura 5.1: Richkware-Manager-Client: comando "ls"

5.1.2 Client telnet, mediante linea di comando

I comandi possono essere eseguiti anche mediante linea di comando, come in figura 5.2

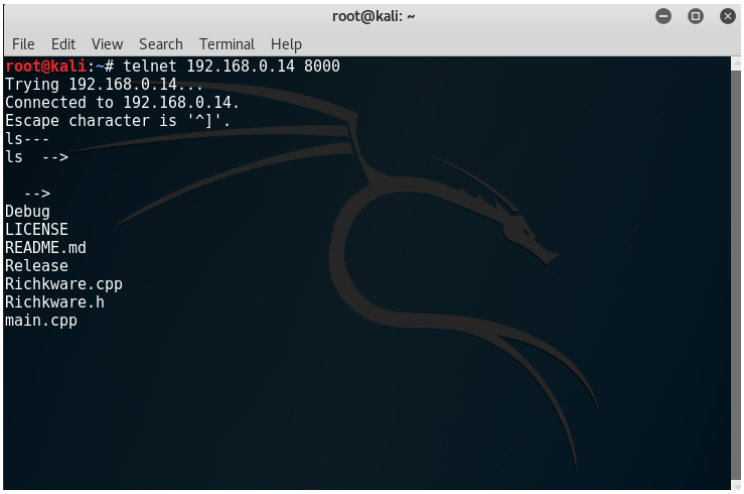


Figura 5.2: Macchina Attaccante: invio comandi da console

In figura 5.3 sono mostrati i pacchetti scambiati dai 2 host durante la comunicazione del comando e la ricezione dell'output, utilizzando il tool di sniffing wireshark.

59	7.336821	192.168.0.16	192.168.0.14	TCP	74 49378 → 8000 [SYN]
60	7.336925	192.168.0.14	192.168.0.16	TCP	74 8000 → 49378 [SYN, ACK]
61	7.337152	192.168.0.16	192.168.0.14	TCP	66 49378 → 8000 [ACK]
76	9.746349	192.168.0.16	192.168.0.14	TCP	73 49378 → 8000 [PSH, ACK]
77	9.758553	192.168.0.14	192.168.0.16	TCP	75 8000 → 49378 [PSH, ACK]
78	9.758772	192.168.0.16	192.168.0.14	TCP	66 49378 → 8000 [ACK]
79	10.107324	192.168.0.16	192.168.0.14	TCP	68 49378 → 8000 [PSH, ACK]
80	10.120365	192.168.0.14	192.168.0.16	TCP	143 8000 → 49378 [PSH, ACK]
81	10.120555	192.168.0.16	192.168.0.14	TCP	66 49378 → 8000 [ACK]
99	16.655235	192.168.0.16	192.168.0.14	TCP	66 49378 → 8000 [FIN, ACK]
100	16.655328	192.168.0.14	192.168.0.16	TCP	66 8000 → 49378 [ACK]
101	16.669061	192.168.0.14	192.168.0.16	TCP	143 8000 → 49378 [PSH, ACK]
102	16.669229	192.168.0.16	192.168.0.14	TCP	60 49378 → 8000 [RST]

Figura 5.3: Pacchetti scambiati durante l'attacco

In figura 5.4 è mostrato lo scambio dei payload tra i 2 host.

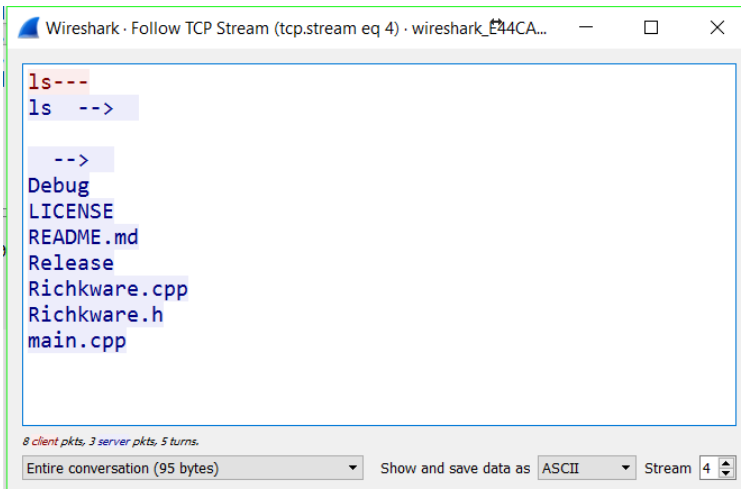


Figura 5.4: Payload scambiati durante l'attacco

La funzione **RawRequest** può essere utilizzata per effettuare la richiesta del comando da eseguire in modo passivo, senza bisogno di aprire un socket in ascolto su una porta, ma contattando un server, che restituirà il comando specifico da eseguire.

5.2 Blocco antivirus e applicazioni di sistema

Per bloccare eventuali antivirus o programmi che potrebbero disturbare il funzionamento del malware si può utilizzare la classe **BlockApps**, che permette di chiudere programmi in base al loro nome.

Nel programma 5.2 vengono bloccati per 5 secondi i programmi "Windows Defender", che è l'antivirus di default di Windows 10, "Command Prompt" per evitare che la vittima usi la console di Windows e "Registry Editor" che è di particolare importanza dato che bloccando l'editor del registro di sistema la vittima non può modificarne le voci, non potendo quindi rimuovere eventuali voci inserite per l'installazione del malware nel sistema (Persistance).

Listing 5.2: Blocco antivirus e applicazioni di sistema

```
int main() {
        ...
        richkware.blockApps.start();

        richkware.blockApps.dangerousApps.add("App1");
        richkware.blockApps.dangerousApps.add("App2");
        richkware.blockApps.dangerousApps.add("App3");

        Sleep(5000);
        richkware.blockApps.stop();
        ...
}
```

5.3 Installare il malware in modo persistente

Per effettuare l'installazione del malware in modo persistente, quindi quando il sistema viene riavviato, il malware riparte, si può utilizzare la funzione **Persistance**.

Listing 5.3: Persistance

```
int main() {
        ...
        richkware.systemStorage.Persistence();
        ...
}
```

Nelle immagini 5.5 e 5.6 si può osservare che il file eseguibile è stato inserito tra i file di sistema e che verrà eseguito all'avvio, perché è presente nei valori del registro nel percorso "...CurrentVersion/Run".

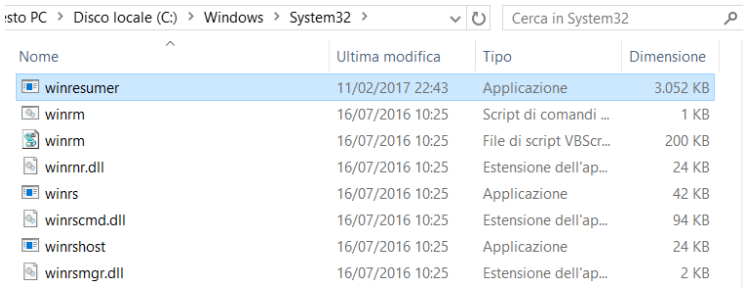


Figura 5.5: Macchina vittima: directory di sistema

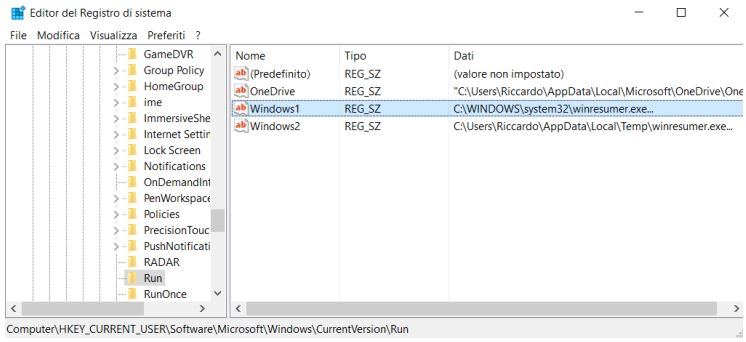


Figura 5.6: Macchina vittima: Registro di sistema

5.4 Keylogger

Per effettuare questo tipo di attacco che consiste nell'intercettare tutto ciò che un utente digita sulla tastiera del computer, si utilizza il metodo **Keylogger**.

Listing 5.4: Keylogger

```
int main() {
        ...
        richkware.Keylogger();
        ...
}
```

Nell'immagine 5.7 si può osservare che sono stati registrati i tasti digitati e salvati nella cartella temporanea "**.../AppData/Local/Temp**", in particolare nel file dal nome dato come parametro alla funzione **Keylogger** in 5.4.

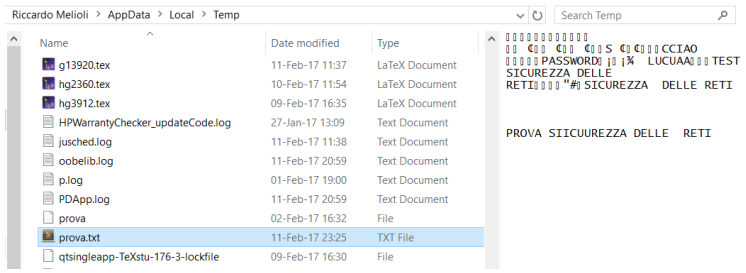


Figura 5.7: Macchina vittima: output Keylogger in una cartella temporanea

Conclusioni e sviluppi futuri

In questa relazione si è illustrato il framework open-source Richkware per la creazione di malware, che possono svolgere diverse funzioni. Sono stati spiegati i metodi della libreria e le possibili applicazioni di tali metodi per effettuare attacchi in determinati scenari.

I possibili sviluppi futuri sono:

- creazione di un main "intelligente", che applichi concetti di intelligenza artificiale, sia nel mantenersi nascosto nel sistema operativo sia nel decidere che comportamenti assumere in base alla situazione nel sistema.
- estendere la libreria con nuone funzioni, ad esempio per creare un ransomware.

Bibliografia

Alfieri, Roberto
2015 «Slide del corso di Reti degli elaborati, UNIPR».

AV-Test 2016 , av-test.org.

Cimato, Stelvio
2016 «Slide del corso di SSR, UNIMI».

Kaspersky Lab 2016 , kaspersky.com.

Securelist 2016 , securelist.com.

Silberschatz, Abraham
2014 *Sistemi Operativi. Concetti ed esempi*, Pearson.

Tanenbaum, Andrew S.
2011 *Reti di Calcolatori*, Pearson.

Wikipedia 2016 , wikipedia.com.