UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

Sicurezza Informatica e Internet

A.A. 2015/2016

Build Your Own Botnet v.1

0211577 - Cappello Domenico - domenico.cappello@gmail.com
0213347 - Nazio Alessio - alessio.nazio@gmail.com

Indice

1 Introduzione **1**

1.1 Premessa 1

1.2 HTTP Botnet 2

1.3 Obiettivo 3

1.4 Struttura della relazione 4

2 Stato dell'arte **5**

2.1 Punti cardine 5

2.1.1 HTTP 5

2.1.2 Singleton Pattern 6

2.1.3 Schedulatore 7

2.1.4 ID dei bot e MD5 7

2.1.5 Parsing 8

2.1.6 Multi-piattaforma 8

2.2 Piattaforma di sviluppo, Swing, Awt 9

2.3 Build your own botnet 9

2.3.1 Struttura del progetto 9

2.3.2 ByobComm 10

2.3.3 ByobSingleton 10

2.3.4 ByobTask 11
2.3.5 Byob_v1 11
2.3.6 GUI 11
2.3.7 Parser 17
2.3.8 Tools 18
2.3.9 URLDetails 19

Capitolo 1

Introduzione

1.1 Premessa

Oggigiorno, la maggior parte dei PC utilizza sistemi operativi senza patch e/o senza alcuna sicurezza dietro un *firewall*, rendendoli facili prede per attacchi diretti, orchestrati da malintenzionati, e/o per attacchi di tipo indiretto, mascherati dietro programmi che l'utente usa costantemente (vedi reti *P2P*).

Con l'incremento delle connessioni a banda larga si ha avuto anche un incremento del numero di potenziali vittime di attacchi, con cui i malintenzionati traggono beneficio dalla situazione, utilizzandola a loro vantaggio, sfruttando anche l'automatizzazione di tecniche per la scansione di porzioni della rete che semplifica la ricerca di sistemi vulnerabili. Una volta che un vasto numero di macchine sono state infettate, esse entrano a far parte di "una rete di macchine compromesse che posso essere controllate da remoto[1]", chiamata una **botnet**.

Una *botnet* consiste di tre elementi principali che sono i bot (cioè le macchine infettate che ne fanno parte), il *command and control server* (C&C - da cui ogni bot riceve istruzioni e con cui il malintenzionato ha privilegi amministrativi remoti su tutte le macchine infette) e un *botmaster*; si basa, inoltre, su quattro concetti chiave:

[1] Provos Niels, Holz Thorsten (2007).

1. le *botnet* sono reti, quindi sistemi in cui la comunicazione è importante;
2. le macchine che fanno parte di una *botnet* sono, tipicamente, partecipanti ignari;
3. i *bot* sono controllabili da remoto, permettendo di fare rapporto o ricevere ordini da una struttura C&C (centralizzata o decentralizzata);
4. i *bot* sono controllati da persone con intenti malevoli che fanno capo a qualche forma di attivitá illegale, come la diffusione di un virus, effettuare attacchi DDos, spam, ...

Se il proprio computer diventa parte di una botnet, i criminali creando un scenario di vero e proprio pericolo per ogni infrastruttura IT, in quanto i bot verranno utilizzati con intenzioni malevole.

1.2 HTTP Botnet

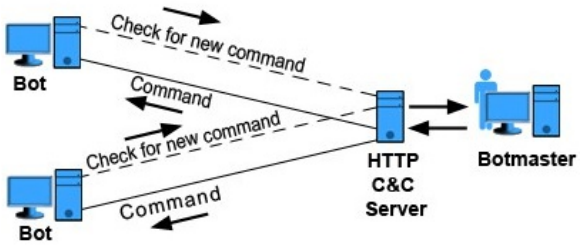

Figura 1.1: Struttura di una botnet

Quando il protocollo HTTP nacque nel 1999, nessuno avrebbe mai pensato che sarebbe stato utilizzato per le botnet. La prima generazione di botnet usava l'*Internet Relay Chat* (IRC[2]) e relativi canali per instaurare un meccanismo di "controllo e

[2]Protocollo di messaggistica istantanea su Internet, che consente sia la comunicazione diretta fra due utenti, che il dialogo contemporaneo di gruppi di persone raggruppati in stanze di discussione dette canali.

comando". I bot IRC seguono lo stesso approccio PUSH di quando ci si unisce ai canali, rimanendo connessi. Essi si connettono ai server IRC e ai canali che sono stati selezionati dal *botmaster* e attendono comandi. Invece di rimanere connessi, i bot HTTP controllano periodicamente per aggiornamenti oppure nuovi comandi: questo modello è detto di PULL e continua ad intervalli regolari definiti dal botmaster, il quale, usa il protocollo HTTP per nascondere le proprie attivitá tra il normale flusso web, riuscendo ad evitare facilmente i metodi di rivelazione come i *firewall*.

1.3 Obiettivo

L'obiettivo è quello di sviluppare un software per workstation che definisca bot in grado di contattare delle URL a cui sono associate una serie di variabili:

1. definizione della periodicitá del contatto fissa o variabile in un intervallo temporale;
2. definizione del numero massimo di contatti per ciascuna URL;
3. definizione di una modalitá di "*sleep*", intesa come insieme di condizioni in cui non viene effettuata nessuna azione ;
4. definizione di uno user-agent "*custom*";
5. definizione di un indirizzo e porta di "*proxy*" pubblico.

Tali definizioni saranno effettuate o tramite *file .txt* oppure tramite *Graphic User Interface* (GUI).

Inoltre, contatti alle URL con i loro dettagli, variabili di configurazione, *timestamp*

dei contatti, Sistema operativo utilizzato e *browser* presenti sulla macchina saranno opportunamente salvati in un *file log*.

1.4 Struttura della relazione

L'analisi, progettazione e sviluppo del progetto inizia con un'esposizione del programma implementato, spiegandone la logica, ed esponendo il tentativo di conferire una modalitá di utilizzo immediata, con poche piccole azioni. Nel capitolo successivo si parlerá degli elementi principali del progetto e tutto ciò che li riguarda, evidenziandone il processo di analisi. A seguire, un capitolo dedicato alle metodologie utilizzate per la creazione dei bot facenti parti della rete. Infine, l'ultimo punto, è quello dell'analisi dei casi di test effettuati per un corretto funzionamento del programma.

Capitolo 2

Stato dell'arte

Questo capitolo si sofferma dapprima su una breve introduzione ai punti cardine del progetto, per poi esporre il percorso di progettazione seguito per un suo corretto sviluppo, illustrandone l'utilizzo dal punto di vista dell'utente, attraverso le schermate principali e rafforzandone la logica di funzionamento adottata. In seguito è mostrata l'architettura software con le scelte implementative di rilievo, spiegando aspetti legati al parsing e all'esecuzione delle operazioni.

2.1 Punti cardine

In base ai requisiti richiesti, i punti cardine del progetto sono:

2.1.1 HTTP

HTTP, acronimo di *hypertext transfer protocol*, costituisce il cuore delle azione intraprese dal singolo *bot*.

È un protocollo a livello applicativo utilizzato per la trasmissione di informazioni tramite un meccanismo di richiesta/risposta tra *client* e *server*. All'interno del messaggio di richiesta sono definiti diversi campi tra cui, quelli a noi più tili, il tipo di richiesta (*GET*, *POST*) e lo *UserAgent*, identificativo del tipo di *client* utilizzato. Per il nostro

scopo, il metodo "*GET*" è usato per ottenere il contenuto della risorsa indicata come *URI* (come può essere il contenuto di una pagina *HTML*). Può essere:

- **assoluto**, ossia quando la risorsa viene richiesta senza altre specificazioni;
- **condizionale**, ossia quando la risorsa corrisponde ad un criterio indicato nell'*header*;
- **parziale**, ossia quando la risorsa richiesta è una sottoparte di una risorsa memorizzata.

Per quanto riguarda lo *user-agent*, all'interno dell'*header HTTP*, si hanno informazioni riguardanti il tipo di agente utente, cioè il tipo di *browser* che sta effettuando la richiesta. Se modificato, permette ad un *hacker* di diventare lo *user-agent* che vuole, potendo così selezionare i *payload* di software maligni sulla base del tipo di *user-agent*. Per il nostro scopo il metodo *GET* è stato utilizzato come "assoluto" e si è messo a disposizione dell'utente la possibilità di poter modificare lo *user-agent* a proprio piacimento.

2.1.2 Singleton Pattern

In molte situazioni è necessario che venga istanziato un solo esemplare di una data classe. Per esempio: se si ha un riproduttore di file musicali sarà bene che esso venga istanziato una sola volta per non ritrovarsi due riproduzioni contemporanee; uno *spooler* di stampa dovrebbe tenere una coda unica anche se ci sono piú stampanti attive; il manager del file system dovrebbe essere unico, come pure una cache; ecc. Per garantire questa singola istanziazione basta rendere impossible l'uso del costrutto *new* da parte del programma utente e di fornire un metodo indiretto per ottenere una istanza (l'unica) della classe. A tal fine occorre:

- dichiarare privato il costruttore, in modo che esso possa essere visto solo dall'interno della classe Singleton e non dal programma utente (ciò rende impossibile l'istanziazione di un oggetto dall'esterno della classe Singleton);
- prevedere il metodo (pubblico) statico e cioè di classe, in modo che esso sia comunque visibile. Questo metodo deve istanziare un esemplare se ciò non è ancora accaduto, oppure restituire l'oggetto giá istanziato in precedenza senza istanziare ulteriori esemplari.

Questa seconda scelta è quella applicata nel progetto per la creazione e gestione del *logger* e la creazione di uno schedulatore di task.

2.1.3 Schedulatore

Lo *scheduler* è un componente chiave per l'applicazione.
Grazie a questo, è stato possibile stabilire un ordinamento temporale per l'esecuzione delle azioni del singolo *bot*, così come specificato dall'utente attraverso i parametri di configurazione. La sua implementazione è stata possibile attraverso la creazione di un *executor* a singolo *thread* che schedula comandi da eseguire dopo un certo ritardo o da eseguire periodicamente[1]. Valori di ritardi pari a 0 o negativi (ma non i periodi) sono trattati come richieste di esecuzione immediata.
In sintedi, la schedulazione può essere così pensata: il *thread* dello schedulatore assegna i *task* a un *thread* estratto da un *pool*.

[1]Si noti che se il singolo *thread* termina in maniera anomale durante l'esecuzione, uno nuovo prenderá il suo posto, se necessario, per eseguire *task* successivi.

2.1.4 ID dei bot e MD5

I *bot* devono essere univocamente identificabili.

Questo perchè il malintenzionata che vuole effettuare un attacco (per esempio, *DDoS*) deve sapere quanti *bot* ha a disposizione in un certo lasso di tempo. Inoltre, se il malintenzionato decide di affidare un gruppo di *bot* ad un acquirente esterno (per esempio, per condurre una piccola e veloce campagna di *spam*), mantenere un conteggio della potenza di attacco che possono essere offerti è necessario per accordarsi sul prezzo. Per garantire l'unicitá dell'identificativo di ciascuno, si è deciso di legare l'hardware della macchina al suo sistema operativo tramite*checksum MD5* per ottenere una stringa a 32 caratteri che è utilizzata come identificativo.

Si noti come questo è stato fatto per una integrazione con un futuro sviluppo del progetto.

2.1.5 Parsing

Per *parsing* di una stringa di caratteri si intende l'analisi della stessa per trovare *token*, oggetti o *pattern* per crearne una struttura.

Nel progetto era importante definire un *pattern* per i parametri di configurazione, per rivelare la loro correttezza e capire se si trattasse di un campo facoltativo o meno, per poter così capire se la computazione dovesse fermarsi lì. Quanto detto è stato applicato sia in lettura, sia in scrittura: questo perchè i parametri di configurazione possono essere anche salvati su un file di testo.

2.1.6 Multi-piattaforma

Quando si parla di un programma multi-piattaforma si intende un programma che sia in grado di funzionare correttamente su diversi sistemi operativi.
A tale fine, l'esecuzione di istruzioni "macchina-dipendenti" (come, per esempio, la generazione dell' ID del bot che dipende sia dall'hardware che dal sistema operativo) appariranno all'utente in maniera totalmente trasparente per sistemi operativi come OSX, Windows o Linux.

2.2 Piattaforma di sviluppo, Swing, Awt

NetBeans è un ambiente di sviluppo integrato (*IDE - Integrated Development Environment*) multi-linguaggio, scelto dalla Oracle Corporation come *IDE* ufficiale da contrapporre al piú diffuso Eclipse.
NetBeans utilizza due componenti principali: la piattaforma, che comprende una serie di librerie per fornire gli elementi base dell'*IDE* come presentazione dei dati e interfaccia utente, e l'*IDE* vero e proprio, che permette di gestire il controllo e le funzionalitá offerte dalla piattaforma. NetBeans utilizza *Abstract Window Toolkit* (*AWT*), un insieme di API realizzate da Sun che permettono agli sviluppatori di modellare le interfacce grafiche delle finestre, pulsanti e altri elementi visuali. *AWT* fornisce gli elementi grafici base che dipendono dalla piattaforma utilizzata, mentre per gli aspetti di alto livello come gestione di colori e interazione con l'utente è usata la libreria Swing.

2.3 Build your own botnet

Questa sezione tratta del programma e della sua architettura, nonchè delle scelte implementative rilevanti e dei casi d'uso principali.

2.3.1 Struttura del progetto

Il progetto è composto da diverse classi:

- **ByobComm**, responsabile del metodo GET del protocollo HTTP;
- **ByobSingleton**, responsabile del *log* delle azioni intraprese, delle informazioni della macchina dell'utente e dello *scheduling* dei *task* da effettuare;
- **ByobTask**, implementa l'interfaccia *Runnable* per il *multithreading*;
- **Byob_v1**, avvia il programma e l'interfaccia grafica;
- **GUI**, responsabile di tutto quello che riguarda la creazione e gestione della *Graphic User Interface* (quindi, componenti, loro visibilitá, loro aspetto e la loro abilitazione per l'uso);
- **Parser**, responsabile per la creazione di file, lettura e scrittura dei parametri di configurazione inseriti dall'utente, nonchè del loro *splitting* e della loro conversione nel giusto tipo di dato;
- **Tools**, contenente varie funzioni e vari metodi che non appartenevano a nessun'altra classe;
- **URLDetails**, contenente i dati (parametri di configurazione) del singolo *task* da effettuare.

2.3.2 ByobComm

La classe *ByobComm* gestisce tutto quello che riguarda il protocollo HTTP.
Prima di creare una connessione si controlla che tipo di parametri di configurazione ha inserito l'utente e se impattano sull'apertura della connessione. Uan volta istanziato l'oggetto (con o senza *proxy*, in base all'input dell'utente), la connessione è aperta, si setta il metodo di tipo *GET*, la codifica dell'*header* e lo *user-agent* (se è stato fornito). A questo punto si estrae il codice di risposta, prima di chiudere la connessione, che viene analizzato all'interno del programma per i fini progettuali.
Con lo stesso approccio, si era originariamente pensato di creare una funzione booleana per il controllo dell'*URL* inserito dall'utente subito dopo aver premuto il pulsante "*Push*": purtroppo impattava sulle prestazioni dell'interfaccia grafica (il pulsante rimaneva cliccato finchè non era chiusa la connessione).

2.3.3 ByobSingleton

Come suggerisce il nome, la classe *ByobSingleton* è la responsabile per il *singleton* che si occupa di creare il *log* per la trascrizione di:

- contatti e parametri di configurazione;
- *timestamp* di contatto delle URL ed dettaglio delle URL contattate;
- informazioni relative al Sistema Operativo e al/i browser presenti sulla postazione su cui il software è installato.

oltre il *log*, la classe gestisce lo schedulatore dei *task*, di cui si è parlato precedentemente.

2.3.4 ByobTask

La classe *ByobTask* si occupa della definizione del singolo *thread* (o *task*).
È stata focalizzata l'attenzione sul *polling* con intervallo esteso e casuale per verificare se non sono piú rispettate le *sleep condition*: la casualitá è stata dettata dal fatto che se il contatto avveniva sempre alla stessa ora in cui veniva schedulato poteva destare sospetti e, grazie alla sua semplicitá, permette di evitare di dover ogni volta controllare quale/i è/sono la/le *sleep condition*.

2.3.5 Byob_v1

La classe in esame non è nient'altro che la responsabile dell'invocazione della *GUI* (e del settaggio del titolo, assieme alla sua posizione per far in modo che appaia al centro dello schermo, indipendentemente dalla sua grandezza) ed è stata utilizzata a scopo di testing per trovare i giusti comandi da terminale per l'individuazione dei browser, così come richiesti nell'estensione del documento di progetto.

2.3.6 GUI

La classe *GUI* è il cuore dell'applicazione e rappresenta l'interfaccia grafica con cui l'utente fornisce i parametri di configurazione.
A prima vista, l'interfaccia può risultare complicata, ma si è cercato di rendere la modalitá di fruizione più intuitiva possibile, attraverso piccole azioni per il suo utilizzo e con una grafica molto semplice, dato che ci si è soffermati piú sulla sua logica di funzionamento. Le azioni iniziali che l'utente può intraprendere, tramite le *checkbox*, sono due:

1. Aprire un file di configurazione giá in possesso dell'utente;

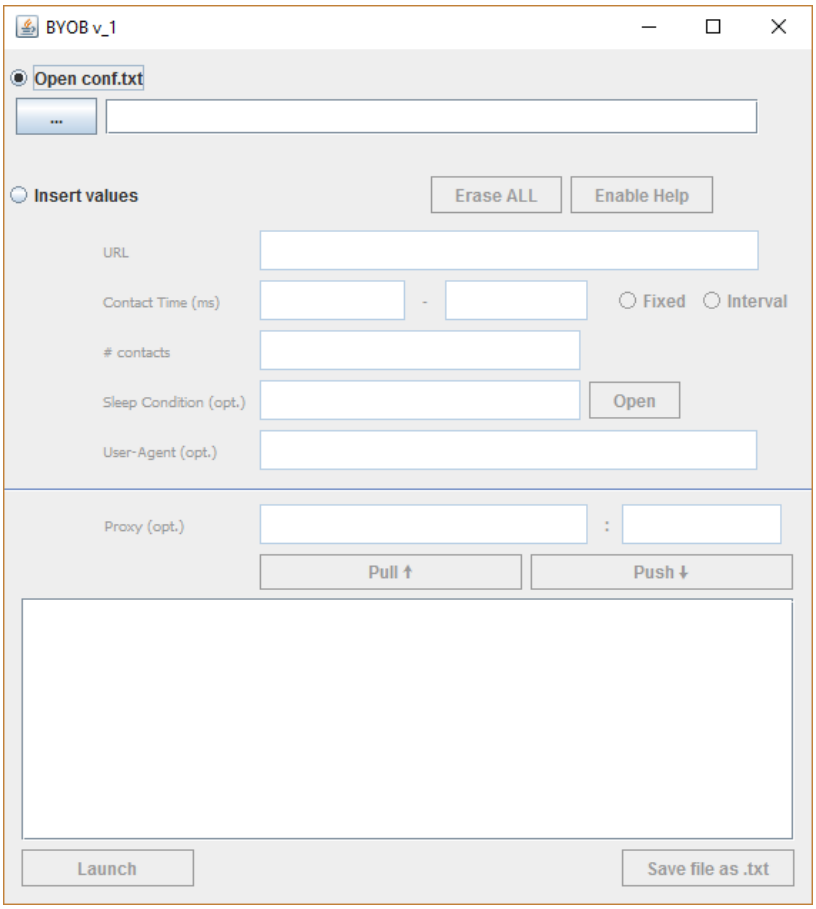

Figura 2.1: Apertura di un file di configurazione

2. Inserire manualmente i parametri di configurazione.

L'apertura di un file di configurazione è riconducibile all'apertura di un qualsiasi file, gesto oramai assimilato nella vita di tutti i giorni. Cliccando sull'apposito pulsante "...", viene aperto il textitJFileChooser, un semplice meccanismo di scelta a disposizione dell'utente (una sorta di *file explorer* per muoversi tra file e cartelle del *file system*). Ad esso è stato applicato un filtro con cui sono visibili solo cartelle e file di testo: questo perchè i parametri di configurazione vengono letti esclusivamente da file con estensione ".txt".

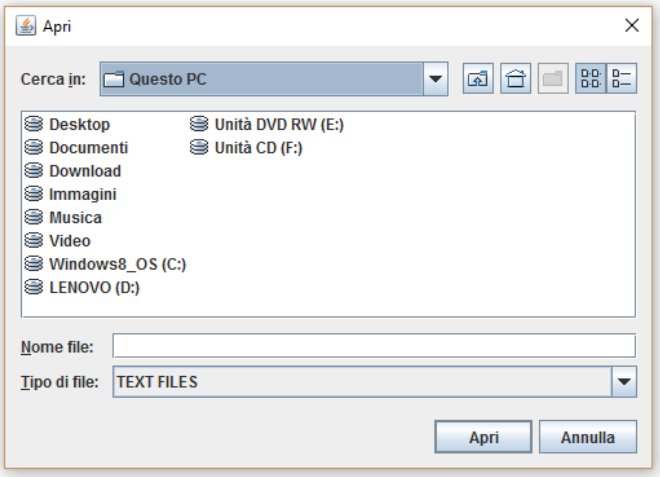

Figura 2.2: Apertura di un file di configurazione

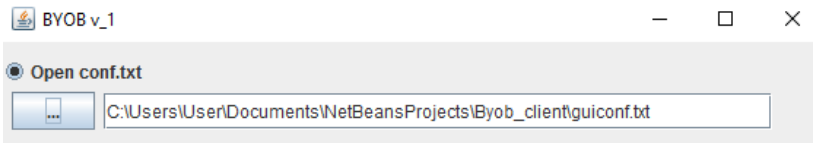

Figura 2.3: Scelta del file di configurazione

Una volta selezionato, il path del file scelto apparirá nella casella di testo relativa, sbloccando il pulsante "Launch" per il *run* dell'applicazione.

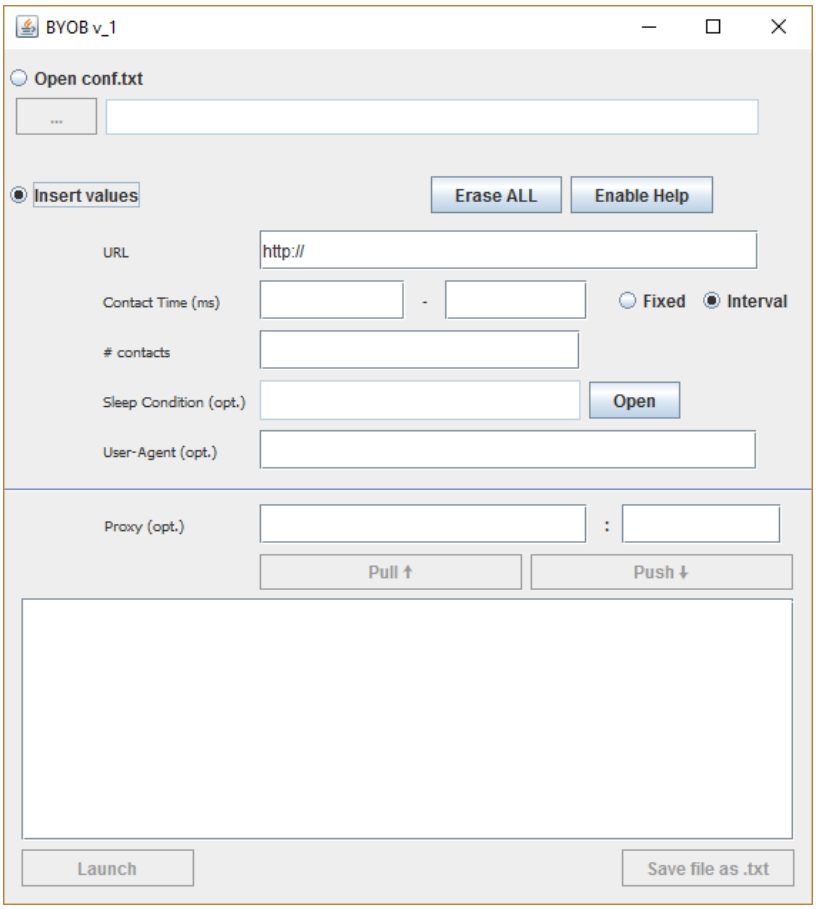

Figura 2.4: Inserimento manuale parametri di configurazione

L'altra opzione è quella dell'inserimento manuale dei parametri di configurazione. Tali parametri sono:

- **URL** da contattare;
- **Contact time**, ossia la periodicitá di contatto che puó essere:
  - Fissa ("*Fixed*");
  - Invervallo ("*Interval*"), cioè all'interno di un intervallo temporale;
- # **contacts**, ossia il numero massimo di contatti per ciascuna URL;

- **Sleep conditions** (opzionale), ossia l'insieme di condizioni temporali in cui non viene svolta alcuna azione; sono state divise in condizioni sui giorni e condizioni sulle ore della singola giornata, ognuna caratterizzata da una lettera dell'alfabeto o ad una stringa vuota nel caso di nessuna restrizione:

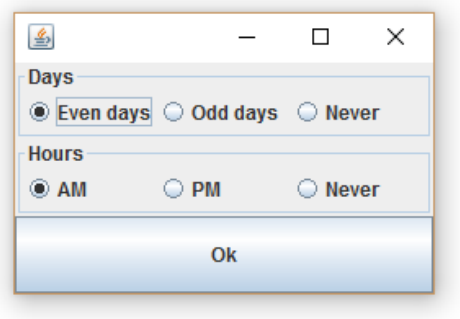

Figura 2.5: Sleep conditions

  - Condizione sui giorni pari (E), sui giorni dispari (O), nessuna restrizione;
  - Condizione sulle prime dodici ore (A), condizione sulle ultime 12 ore (P), nessuna restrizione;

- **User-Agent** (opzionale), ossia la modifica da apportare allo User-Agent del protocollo HTTP;

- **Proxy** (opzionale), ossia l'indirizzo e porta di un proxy pubblico da utilizzare.

Una volta inseriti nelle relative caselle di testo, il pulsante "*Push*" permette di visualizzare tali parametri (come apparirebbero se fossero stati scritti sul file di configurazione) nell' area di testo sottostante: si noti come, inserendo altri parametri, essi vengano appesi a quelli giá presenti. Il pulsante "*Pull*" permette di estrarre dall'area di testo l'ultimo inserimento effettuato per una eventuale modifica dei parametri non correttamente inseriti.

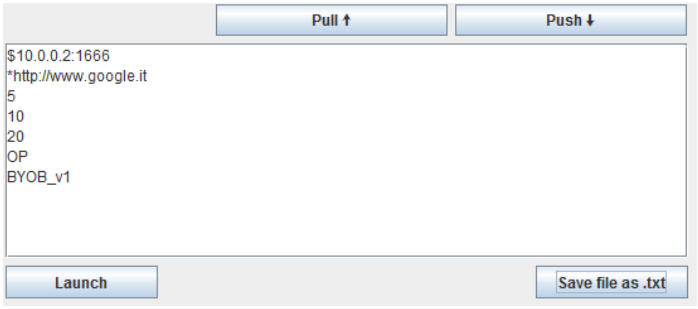

Figura 2.6: Push - Inserimento dell'input nell'area di testo

L'inserimento manuale ha il fine di salvare tutti i parametri di configurazione all'interno di un file di testo (azione svolta dal pulsante "Save file as .txt"), con un approccio simile a quello utilizzato per l'apertura del file di configurazione, cioè attraverso il textitJFileChooser.

GUI user-friendly Per rendere il tutto semplice e a portata di qualsiasi utente, il *toogle* "*Enable Help*" permette di invocare dei *tooltips* sulle caselle di testo per meglio capire che dato inserire e in che formato. Nell'esempio fornito, per sapere come correttamente inserire il parametro *URL*, basta posizionarsi col mouse sulla casella di testo corrispondente e attendere qualche secondo.

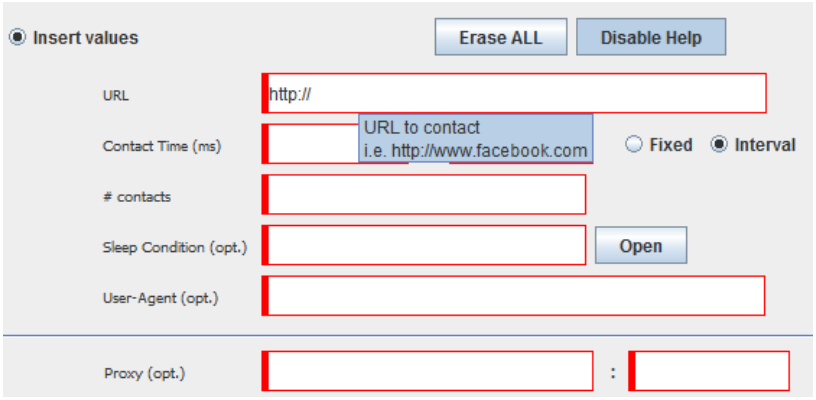

Figura 2.7: Utilizzo dell'help

Per quanto riguarda il pulsante "*Erase ALL*", la sua implementazione è nata dalla volontá di fornire all'utente un metodo veloce per cancellare ogni inserimento nelle caselle di testo nel caso di molteplici errori.
Infine si è scelto di fornire un *warning message*, con l'approccio di un *pop-up*, in caso di errato e/o mancato inserimento di uno o piú parametri tramite il pulsante "*Push*": l'utente, a questo punto, puó decidere se tornare indietro e correggere da solo l'errore oppure utilizzare dei valori di *default* per i parametri di configurazione.

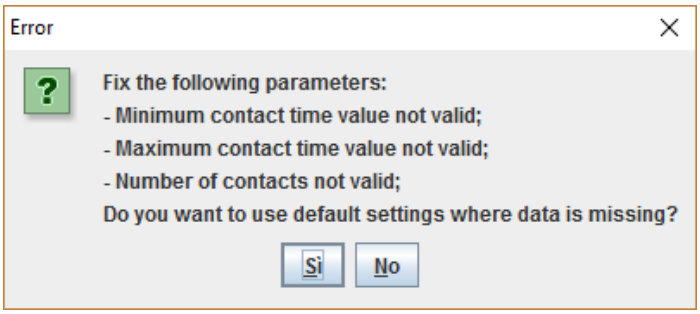

Figura 2.8: Esempio di messaggio d'errore per dati mancanti e/o errati

2.3.7 Parser

La classe *Parser* si occupa di tutti gli aspetti legati al controllo dell'*input*, sia dal punto di vista sintattico che dal punto di vista alfanumerico, ma anche la correttezza di *pattern*, come nel caso venga inserito un indirizzo IP per il *proxy*. Altre responsabilitá della classe sono:

- lettura e scrittura del file contenente i parametri di configurazione;
- creazione di una lista di istanze della classe *URLDetails*, una per ciascun contatto da effettuare;

- conversione dei parametri di configurazione nel tipo corretto (per fornire un esempio, il numero di contatti da stringa a intero);
- controllo della validitá di ciascun parametro di configurazione (se il parametro è un numero, se gli estremi degli intervalli sono uno maggiore dell'altro, ...).

2.3.8 Tools

La classe *Tools* contiene al suo interno, tra varie funzioni e vari metodi, la generazione dell' identificativo del *bot*.
Tale generazione è effettuata tramite la ricerca del sistema operativo della macchina ospitante per poi lanciare, tramite riga di comando, delle istruzioni con cui ottenere dati riguardanti l'*hardware* della stessa. Queste informazioni sono poi "date in pasto" ad un funzione *hash* con cui è generato l'ID. La decisione di utilizzare entrambe le informazioni, sia *hardware* che *software* è stata dettata dalla volontá di ridurre le possibilitá di ID uguali, grazie all' indirizzo *MAC* univoco della macchina e al sistema operativo che può trovarsi su piú macchine differenti. La funzione *hash* usata è MD5, fornita dalla classe *MessageDigest*, perchè è veloce da generare e non si avevano grosse pretese di sicurezza (non si tratta di una password ma bensì di un identificativo che servirá al *botmaster* per capire quante unitá ha a disposizione per un eventuale attacco).
Questa classe si occupa anche della schedulazione dei *task*, tramite l'unica istanza della classe *ByobSingleton*. Altri metodi e funzioni della classe hanno lo scopo trovare informazioni sulla macchina ospitante (sistema operativo, architettura, browser instal-

lati, lanciare istruzioni da riga di comando) ed effettuare i controlli per la generazione del *warning message* della classe GUI.

2.3.9 URLDetails

URLDetails è la classe che contiene tutte le informazioni inserite dall'utente per ciascun contatto che il *bot* deve effettuare. Oltre agli attributi, che coincidono con i parametri di configurazione forniti dall'utente, mette a disposizione i metodi getXX e setXX (dove XX è il parametro di configurazione) per recuperare e settare il loro valore.