

# Build your own botnet: BITVIR

Progetto di Sicurezza Informatica ed Internet

Ferrantelli Federico  
Simolo Christian  
Terribili Lorenzo

## Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Specifiche principali dei progetti . . . . .	3
1.2	Considerazioni sul progetto ed estensioni della traccia . . . . .	4
<b>2</b>	<b>La nostra botnet: BiTViR</b>	<b>6</b>
2.1	Struttura della botnet . . . . .	6
2.1.1	Tecnologie . . . . .	6
2.1.2	Assunzioni . . . . .	9
2.1.3	La rete & il problema del <i>rendez vous</i> . . . . .	10
2.1.4	Protocolli di comunicazione . . . . .	12
2.1.5	Struttura dell'applicazione ed entità coinvolte . . . . .	12
2.1.6	Applicazione aggiuntiva: DnSViR . . . . .	14
2.2	Funzionamento dell'applicazione basata sui ruoli . . . . .	15
2.2.1	Bot . . . . .	16
2.2.2	C&C . . . . .	19
2.2.3	Sito internet . . . . .	22
<b>3</b>	<b>Crittografia</b>	<b>30</b>
3.1	Autenticazione . . . . .	31
3.2	Cifratura dei messaggi . . . . .	36
<b>4</b>	<b>Vantaggi delle scelte implementative</b>	<b>38</b>
<b>5</b>	<b>Manuale d'utilizzo</b>	<b>41</b>

# 1 Introduzione

In questo capitolo è contenuta la traccia relativa al progetto svolto, seguita da alcune considerazioni legate al suo sviluppo e dalle estensioni apportate alle specifiche.

## 1.1 Specifiche principali dei progetti

**Traccia del progetto:** Build Your Own Botnet v.1 (BYOB).

*Si sviluppi un software per workstation (no mobile).*

**Obiettivo principale:** il software deve essere in grado di contattare (HTTP GET) una o più URL internet. Per ciascuna URL bisogna prevedere una serie di variabili:

- i.* dettaglio delle URL da contattare (minimo 2 diverse);
- ii.* periodicità di contatto, che può essere fissa (tempo impostato in secondi) o random in un intervallo min-max fisso (tempo sempre impostato in secondi);
- iii.* numero massimo di contatti per ciascuna URL;
- iv.* impostazione di uno “sleep mode”, da intendere come insieme di condizioni per non effettuare alcuna azione (es. giorni dispari della settimana, orario AM o PM, ecc.);
- v.* personalizzazione del campo user-agent (es.: “BYOB v.1”).

I valori delle variabili dovranno essere impostati attraverso un file (di testo) di configurazione.

I contatti ed i parametri di configurazione dovranno esser memorizzati in un file di testo contenente, oltre alle informazioni di configurazione, anche il timestamp di contatto delle URL ed il dettaglio delle URL contattate.

**Estensioni:** le variabili precedenti dovranno essere impostate dall’utente tramite GUI. Aggiungere inoltre tra le variabili impostabili anche l’indirizzo di un proxy pubblico (URL o IP) da usare per i contatti. Raccogliere e scrivere su un file TXT le informazioni relative al Sistema Operativo e al/i browser presenti sulla postazione su cui il software è installato.

**Traccia del progetto:** Build Your Own Botnet v.2 (BYOB2).

*Si sviluppi un web service con due diverse home page.*

**Obiettivo principale:** il web server deve distinguere se è stato contattato canonicamente (es. index.php tramite browser), presentando una prima home page generica, oppure se è stato contattato con una “codifica particolare” (client sviluppato ad hoc), presentando una seconda home page generica.

Non viene fornito alcun vincolo sulla modalità di “codifica particolare”, ma solo alcune possibili indicazioni:

*vi.* user-agent custom (es. “BYOB v.2”);

*vii.* metodo POST in luogo di GET;

*viii.* variabili a seguito della home page

**Estensioni:** in risposta al contatto con “codifica particolare”, il web server può fornire, in luogo della generica home page, un file di testo con un elenco di coppie (ID, VALORE) da memorizzare in locale sul file system del client che contatta la home page (bisogna sviluppare anche il client).

## 1.2 Considerazioni sul progetto ed estensioni della traccia

Come consigliatoci a lezione, essendo il nostro un gruppo di tre persone abbiamo deciso di estendere le tracce ed implementare più funzionalità, ispirandoci alle tracce BYOB e BYOB2 (vedi paragrafi precedenti). Tutte le funzionalità qui citate saranno illustrate in maniera esaustiva nei capitoli 2 e 3 di questo documento, dove sarà possibile analizzare in dettaglio il progetto dai suoi flussi esecutivi ai riferimenti teorici che ne supportano le proprietà crittografiche sulle quali si appoggia.

Si noti che alcuni punti indicati nelle specifiche (ad esempio il punto *v* di BYOB ed il *i* di BYOB2) coincidono logicamente e sono stati trattati come un’unica specifica in grado di gestirne ogni funzionalità. Per una maggiore chiarezza, abbiamo usufruito di diagrammi UML, in particolare del *flow diagram* per rivolgerci agli *use case* principali.

È stata realizzata sia l’applicazione del server C&C relativa alla traccia BYOB sia la controparte e quindi l’applicazione BOT relativa alla traccia BYOB2, trattandole come un’unica applicazione in modo da implementare una tipologia di rete P2P; infine, come vedremo anche più avanti, sono state applicate tutte le estensioni che saranno riscontrate nei paragrafi di questa relazione. L’applicazione raccoglierà tutte le funzionalità a disposizione dei singoli peer della rete, quali: ingresso nella rete, flood delle informazioni, comunicazione con il

vicinato, gestione delle richieste e degli attacchi, ed altro ancora.

Inoltre, l'applicazione prevede l'implementazione di due diverse home page presenti nel pannello di gestione del server raggiungibili a seconda del ruolo assunto dall'utente che ha effettuato il log in:

- la prima, accessibile da ADMIN (ruolo di amministratore), permette sia la gestione degli utenti sia la visualizzazione delle informazioni relative a tutta la botnet;
- la seconda, accessibile da USER (ruolo ricoperto da chi affitta il nostro servizio), permette di monitorare la propria porzione di bot assegnati e di inviare comandi alla botnet stessa.

Infine, non essendo ancora soddisfatti della topologia descritta in BYOB, abbiamo deciso di unire le due applicazioni appena descritte per realizzarne un'unica in grado di comportarsi in entrambe le maniere (sotto determinate politiche che analizzeremo nelle prossime pagine), rendendo la rete una *peer-to-peer* (P2P).

Durante la lettura dei prossimi capitoli, si potranno riscontrare richiami ai requisiti delle specifiche sopra citati, poiché vogliamo facilitare la comprensione e fornire una chiara illustrazione dei meccanismi celati dietro le implementazioni dei diversi requisiti. Allo stesso tempo, è nostra premura mostrare la completezza dell'applicazione ed il rispetto di tutte le specifiche date, oltre a quelle aggiunte da noi stessi.

## 2 La nostra botnet: BiTViR

In questa sezione affronteremo un discorso introduttivo a “BiTViR”, la nostra botnet, analizzandone le entità coinvolte, le funzionalità che ne regolano le interazioni ed i flussi esecutivi più significativi dell’applicazione. Ci si focalizzerà principalmente sullo studio della rete costruita e sulle sue proprietà generali che ne regolano le interazioni tra entità.

Verranno elencate le tecnologie ed i software coinvolti nella sua progettazione con un riguardo particolare agli aspetti implementativi relativi sia al *back-end* sia al *front-end* della applicazione sviluppata.

Tuttavia, in questo capitolo, non sono presenti le configurazioni di sicurezza relative agli accessi ed alle operazioni di codifica (*encrypt*) e decodifica (*decrypt*) dei dati scambiati. Scegliamo di rimandare tutte le discussioni relative alla sicurezza ed alle codifiche coinvolte in BiTViR al prossimo capitolo per non appesantire la lettura laddove sono presenti già grandi quantità di informazioni, e per mantenere le illustrazioni chiare ed intuitive. In questo modo potremo spiegare in maniera più esaustiva alla prossima sezione gli aspetti relativi alle challenge affrontate, il modo in cui vengono interpretati i messaggi ricevuti, ed altro ancora.

### 2.1 Struttura della botnet

Vediamo ora lo scheletro del progetto, e quindi le tecnologie coinvolte, l’architettura della rete sulla quale si regge l’applicazione e le entità che eseguono operazioni al suo interno generando particolari interazioni tra nodi.

#### 2.1.1 Tecnologie

Il progetto è stato scritto in linguaggio Java nell’attuale e recentissima versione 8.

Le novità di questa release interamente *made in Oracle* riguardano l’introduzione di *espressioni Lambda* che permettono la costruzione di *funzioni anonime*, ovvero una funzione che ha un corpo ma non un nome.



Un’espressione lambda definisce una funzione anonima che mantiene lo stato; in altre parole, viene si definisce un metodo senza una dichiarazione e quindi senza nome, modificatori d’accesso, dichiarazione del tipo del valore di ritorno.

Brevemente, tutto ciò ci ha permesso di sviluppare agevolmente le operazioni di testing e di ottenere un codice pulito e più intuitivo. Un esempio pratico delle espressioni Lambda ricorrenti nel codice lo troviamo nel notare l'equivalenza tra i seguenti codici:

```
// Come avremmo fatto finora:
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Una classica sintassi!");
    }
});

// Come possiamo fare ora:
button.addActionListener( (e) -> {
    System.out.println("Una nuova sintassi!");
});
```

Relativamente all'ambito di sviluppo e realizzazione dell'applicazione sono stati coinvolti i seguenti strumenti:

- **Spring**, framework open source nato per gestire la complessità nello sviluppo di applicazioni su Java; permette un facile testing del codice e fornisce una vasta quantità di tecnologie e strumenti che favoriscono il riutilizzo del codice già scritto, evitando così di appesantire ulteriormente il codice. La sua architettura si compone di 5 livelli: core e beans (per la creazione, gestione e manipolazione di oggetti di qualsiasi natura), data access (per l'accesso dei dati, tramite l'appoggio di Hibernate), aspect oriented (funzionalità di programmazione orientata agli aspetti, che permette di evitare l'utilizzo di Enterprise JavaBeans per gestire le transazioni), web (utile allo sviluppo di applicazioni web tramite la realizzazione del pattern architeturale Model-View-Controller) ed infine testing.



- Utilizzeremo principalmente **Spring Boot** che permette di creare un server allineato con l'architettura MVC e di importare facilmente le componenti necessarie e personalizzarle agevolmente. È la soluzione più immediata per la produzione di eseguibili java stand-alone testabili su piattaforma Spring sfruttando tutte le funzioni del framework Spring.

- Inoltre, si è fatto largo uso di **Spring Security**, framework che mette a disposizione le ultime tecnologie in ambito di sicurezza permettendo così l'autenticazione e la configurazione delle autorizzazioni legate ad una applicazione java. Alcuni dei strumenti a disposizione del programmatore sono SSL, database crittografati, gestione dei ruoli e delle password, regolazione degli accessi, permessi ed autorizzazioni, gestione dei cookies, gestione delle sessioni, interfacciamento col DB per l'ottenimento di ruoli prestabiliti. È necessario configurare il tutto per personalizzarlo secondo le proprie preferenze.

**mySql**, è un RDBMS (Relational Database Management System) open-source, ovvero uno strumento di

- amministrazione per la creazione e gestione di database coinvolti in applicazioni web basato sul modello relazionale introdotto da Edgar F. Codd.



**Hibernate**, piattaforma open source che fornisce un servizio di object-relational mapping (ORM) ovvero la

- gestione della persistenza dei dati sul database attraverso la rappresentazione ed il mantenimento su database relazionale di un sistema di oggetti Java.



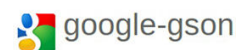
**JSON** (JavaScript Object Notation), formato adatto all'interscambio di dati fra applicazioni client-server, utile per la porzione di codice che mette in comuni-

- cazione l'applicativo con il database e ne permette l'esecuzione di query e la memorizzazione delle informazioni.



Nel dettaglio, è stata coinvolta la recente libreria Google **GSON** per la manipolazione degli oggetti Java in

- grado di convertirli efficientemente in oggetti JSON (e viceversa) e in Java Beans (e viceversa).



**Apache Maven**, si tratta di un sistema dichiarativo per la build automation, utile durante la gestione del progetto Java; sfrutta un costrutto XML particolare denominato POM che descrive le dipendenze fra il progetto e le varie versioni di librerie che sono necessarie all'esecuzione.

- **Maven**

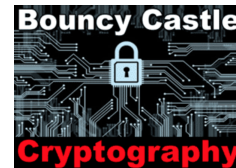




- **Tomcat**, una piattaforma software contenitore servlet open source di Apache che implementa le specifiche JSP e servlet necessarie.



- **Bouncy Castle**, suite crittografica che permette la personalizzazione e l'utilizzo di vaste classi di crittografia per il linguaggio java.



Relativamente all'ambito di testing invece sono stati coinvolti:

- **Mozilla Firefox**, browser web sviluppato da Mozilla; ci è stato utile poiché permette la mutua autenticazione: ovvero, quando il server richiede il certificato, Firefox permette di scegliere tra i certificati aggiunti al browser e di utilizzarne uno in particolare. Questa funzionalità, che ci permette di usufruire al meglio della botnet, non possibile trovarla in tutti i browser di comune uso; Google Chrome, ad esempio, lo permette solo nella versione business.



- **Hamachi LogMeIn**, sistema per la creazione e la gestione di VPN gestito centralmente e comprende il gruppo server (gestito dal venditore) e il software client (che dovrà essere installato sul computer dell'utente). Utilizziamo la versione freeware che ne limita l'uso a 5 computer, che risulteranno così virtualmente collegati nella stessa LAN. Lo scambio di informazioni è isolato da Internet, e tutto il traffico che si scambiano i peer Hamachi è cifrato.



### 2.1.2 Assunzioni

È importante specificare una serie di assunzioni necessarie ad ottenere la miglior performance da BiTViR. Si presume che:

- L'applicazione è stata installata sul computer tramite altri programmi o sfruttando meccanismi di social engineering. Durante l'installazione è richiesto anche il test delle porte aperte da poter utilizzare successivamente, ed in caso di esito negativo, provare ad aprirle. Ci affidiamo al software di Hamachi LogMeIn per costruire una VPN e simulare questa assunzione in fase di testing. L'host risulta quindi infettato;

- L'amministratore dovrebbe mantenere un C&C attivo per permettere la conservazione delle informazioni della rete durante un periodo iniziale nella fase di boot up;
- La rete è già stabilizzata e si trova a regime, assunzione necessaria ad ottenere illustrazioni pulite ed intuitive riguardanti il funzionamento della botnet e l'analisi dei casi iniziali. Eventuali casi particolari possono essere analizzati durante l'esame orale relativo alla consegna di questo progetto, e non verranno pertanto spiegati in questa relazione per non creare ulteriore confusione;
- I nomi scelti per la navigazione (in particolare per DnSViR) sono intuitivi e di scarsa immaginazione, ma permettono una facile comprensione delle sue funzionalità.

### 2.1.3 La rete & il problema del *rendez vous*

BiTViR è una botnet in grado di generare una rete P2P e di esercitare operazioni su di essa; questa scelta è in netto contrasto con la più ovvia struttura centralizzata applicata in molte delle botnet più datate. Lo stato dell'arte delle botnet infatti ci dimostra come una rete centralizzata risulti facilmente una chiara vulnerabilità di progettazione; inoltre, sfruttano la propria struttura P2P per aggirare il cosiddetto problema del *rendez vous*.

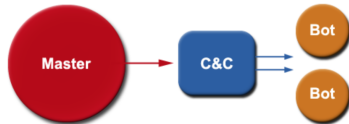


Figura 1: Un esempio di rete centralizzata

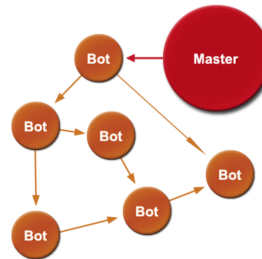


Figura 2: Un esempio di rete decentralizzata

Molte delle botnet meno recenti devono alla loro struttura centralizzata la causa dell'inevitabile individuazione ed abbattimento del C&C, e dunque del Master. Un esempio recente risale al 2008, anno in cui la botnet Srizbi fu responsabile del 93% del volume di spam mondiale generato su Internet a causa della sua infezione e propagazione. Termina la sua attività con la chiusura dell'host provider Janka Cartel, individuato a causa delle vulnerabilità relative alla sua struttura centralizzata (più precisamente, si trattò di una delle ultime e più devastanti Internet Relay Chat botnet).

Questi casi noti hanno incitato i programmatori ad affrontare il problema e complicare la faccenda a chiunque voglia risalire ai nodi principali della rete col fine individuare e localizzare il C&C.

Nel problema del *randez vous* ci si chiede come sia possibile rintracciare il C&C e, eventualmente, analizzarne il comportamento nel caso in cui il C&C venga attaccato o momentaneamente disabilitato. A questo problema rispondiamo con una tecnica implementata solo nelle più recenti botnet: i meccanismi di *rallying*.

Quando parliamo di *rallying* ci riferiamo al prevedere la gestione automatica della dislocazione del C&C; questa tecnica ci permette di recuperare il database ed il controllo della botnet mettendo il nuovo C&C in ascolto per eventuali nuovi comandi richiesti dai bot. I task precedenti infatti risultano ancora avviati fino alla loro terminazione, poiché gestiti esclusivamente dai bot stessi.

Ci affidiamo inoltre alle tecniche di *fast flux*, che analizzeremo più in dettaglio nei prossimi paragrafi, poiché la loro congiunzione con il meccanismo di rallying permette di risolvere del tutto il problema dell'individuazione dei nuovi C&C, e quindi del *randez vous*.

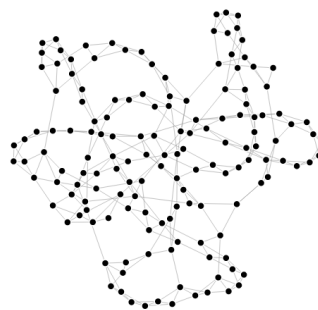
Questo meccanismo da noi implementato riduce l'entropia del numero dei messaggi inviati da ogni bot rendendo difficile scoprire chi sia il vero C&C. La rete generata risulta quindi decentralizzata seppur regolata da un C&C casualmente scelto incaricato di mantenerla efficiente.

Relativamente alla topologia della rete, dopo una breve analisi tra le varie disponibili, è stata scelta quella di un grafo randomico secondo il modello Erdős-Rényi pubblicato nel paper "On Random Graphs" del 1959, al quale aggiungiamo due vincoli:

- Ogni nodo  $x$  deve avere un grado minimo  $\delta(x)$  prestabilito; nel nostro caso, è la funzione logaritmica  $\log_{10} n$ ;
- Il grafo  $G$  generato non deve avere componenti sconnesse

Decidiamo di affidarci a questo modello (come illustrato nell'esempio sulla destra) poiché così possiamo sfruttare le sue interessanti proprietà statistiche.

Un grafo randomico risulta più simile ad una rete tipica della realtà odierna che viviamo, poiché sfrutta distribuzioni probabilistiche applicate ai concetti matematici dei grafi.



In alcuni di paper correlati abbiamo trovato il risultato di alcuni test che elaborano reti con un gran numero di nodi per confrontarle successivamente con modelli alternativi, come ad esempio il modello Watts & Strogatz focalizzato sui cluster di grandi dimensioni presenti in grafi quanto più semplici possibile; i *random graph* sono risultati la categoria di grafi più adeguati al nostro scopo.

Ammettere cluster nella nostra rete avrebbe implicato un rallentamento generale indesiderabile; è stato quindi volutamente scelto di optare per reti sparse, favorendo così un appiattimento della rete stessa col fine di ottenere proprietà omogenee dei nodi, e dunque entropia nulla.

#### 2.1.4 Protocolli di comunicazione

Per lo concerne lo scambio dei messaggi abbiamo deciso di metterci nel quinto livello dello stack TCP/IP dove troviamo lo strato applicativo, utilizzando sia il protocollo HTTP sulla porta 80 sia HTTPS sulla porta 443. Questa scelta è dovuta da diversi fattori:

- La maggior parte del traffico di internet è HTTP, cerchiamo così di rendere più difficile un eventuale monitoraggio della rete (tecniche di sniffing della rete);
- A livello applicativo è più facile instaurare comunicazioni e passarsi i dati;
- Si può utilizzare l'HTTPS per rendere la connessione sicura ed è anche possibile scegliere quale tripla si voglia utilizzare nel TLSv1

#### 2.1.5 Struttura dell'applicazione ed entità coinvolte

In questa sezione verranno descritte le varie funzionalità della nostra applicazione attraverso lo status in cui si troverà (e quindi il ruolo assunto). Come richiesto nelle specifiche, bisogna contattare almeno 2 URL (punto *i* di BYOB) e deve essere impostato un numero massimo di contatti per ciascuna di esse.

Questi requisiti sono stati più che soddisfatti; infatti, le URL da contattare, le abbiamo intese sia come le pagine del sito web sia come i messaggi scambiati nella rete, che, come illustrato a seguire, sono molteplici. Anche il *iii* requisito è stato soddisfatto poiché, durante lo scambio di messaggi, suddetto messaggio viene inviato un numero massimo di volte (impostabile nell'applicazione) finché l'host non lo riceve. Una volta raggiunto, il peer viene considerato offline.

Definiamo e descriviamo ora le varie entità che interagiranno durante l'esecuzione dell'applicazione. Come già accennato, l'applicazione risulta come l'unione di diverse applicazioni, ed il suo comportamento varia a seconda dell'entità che vi interagisce; possiamo quindi considerare diversi status a seconda del comportamento che l'applicazione assume. Gli status possibili dell'applicazione sono:

- C&C; l'applicazione entra in questo status in sole due occasioni: al primo avvio della versione compilata per l'admin (di cui parleremo al prossimo paragrafo) finché non arriva a regime; la seconda occasione si presenta una volta che la rete si è stabilizzata: l'attuale C&C sceglierà il suo successore ogni intervallo di tempo prestabilito. Il C&C deve assolvere diversi compiti, talvolta anche onerosi, come ad esempio mantenere stabile e correttamente aggiornato il grafo della rete, tenere traccia dei bot autenticati, ed altro;
- C&C (SITO); poiché è il C&C a fare da host per il sito, quest'ultimo dovrà tenere traccia degli utenti registrati, dei loro ruoli e dei login effettuati. Inoltre, deve poter eseguire i comandi ricevuti dalle richieste generate nel sito (come ad esempio il flood dei comandi);
- BOT (MALWARE); quando l'applicazione compilata per il bot viene installata su un nuovo host parte nello status di bot. Dopo la fase di autenticazione e quindi dopo l'iscrizione alla botnet, l'applicazione risulta correttamente collegata alla rete e dunque rimane in attesa di eventuali comandi da inoltrare al suo vicinato e da eseguire. Durante il resto del tempo si assicurerà che tutti i suoi vicini siano online;
- APPLICAZIONE AGGIUNTIVA; si tratta di un *mock DNS* denominato Dn-SViR, ovvero una piccola ed intuitiva applicazione in grado di emulare un servizio DNS e permetterci così di implementare la tecnica del *single-flux* spiegata di seguito (vedi applicazione aggiuntiva). Questa soluzione è stata adottata prettamente per ovviare all'acquisto di domini web in un progetto al solo scopo didattico.

### Applicativo per l'amministratore

L'amministratore, che possiamo vedere come proprietario della rete nonché primo esecutore della botnet, è dotato dello stesso applicativo diffuso sulla rete come BiTViR con la differenza che la sua versione viene compilata con particolari proprietà abilitate (ed irraggiungibili per l'utente comune da applicativo o a runtime).

Un'efficace ed immediata modifica di pochi valori all'interno dei file *properties* permette dunque di abilitare l'amministratore ad effettuare tutte le operazioni di gestione della rete e della botnet, come ad esempio: iscrivere nuovi utenti, associarvi autorizzazioni e ruoli, monitorare la rete ed altro ancora.

L'applicativo è anche dotato di funzionalità di recupero della rete e, quindi, del controllo del C&C.

### 2.1.6 Applicazione aggiuntiva: DnSViR

Il mock DNS progettato permette un'immediata gestione dei reindirizzamenti differenziando l'esperienza di navigazione degli utenti a seconda del ruolo da essi ricoperto. La distinzione che si può fare tra richieste in GET ed in POST al server permettono un'efficace risposta, facendo risultare il sito inesistente a chiunque non ponga la richiesta nel modo corretto.

DnSViR, mappando le richieste esterne verso BiTViR, è dotato delle seguenti funzionalità:

- /CIAOSONOUNBOT, metodo POST che reindirizza le richieste di connessione al C&C fornendo all'utente la coppia IP e chiave pubblica del C&C corrente;
- /CIAOSONOUNUMANO, metodo GET utilizzato in fase di testing in grado di reindirizzare le richieste di connessione al C&C effettuate tramite URL del browser ad una pagina internet prestabilita;
- /ALTER, metodo POST che consente esclusivamente all'attuale C&C di trasferire ad un nuovo bot il comando, il database sottostante e di aggiornare presso il mock DNS la coppia IP e chiave pubblica con quella del nuovo bot designato;
- /RESET, metodo POST che consente di ripristinare i valori della coppia IP e chiave pubblica del C&C ai valori di default da parte di qualsiasi bot connesso alla rete; questi valori vengono impostati automaticamente a quelli del primo C&C che assume il controllo della botnet. Assieme alla richiesta è necessario fornire una chiave predefinita di 1024 bits denominata *reset master key* e celata all'interno del codice. La chiave è:

```
1TpUXuLLNrC9p7q7QLtJxjurK8ALQ4VJJXh10qsUh9qrO57G5Gmy
9xGHYO7lO2fbw3YKmA2ii8J0Tkk6QqVDOHjmyNFbGV2MmZW9ji
3lxmH53EOhGAwHeXmWcJCr36Z0KbO67EWtT6QX2W28jx9bgZ2
AXkWzfbAT4rnlptmaT5f2DN28FT1KKmAEicW035nWZRy9enilNuih
oUKczn3Sme548EKmDoGWC10BXJKMpeAlrZ802oD7ZqiNs9IJLw8
VC0qs2F6aOXB1GB4foGCW33PMHpkyXuh0BRxWtnqBgiJC5rivN
JEIfISOOcMWRI8sQUTDSaIHjIWGUE0YeNxMVItYMo6rmaUvEI8v
0UHaorSHT80vaIgr0YngWNj1NBcAMF2QZTDkRxLaF1lcbnT7VYjza
BCy7niyYgSKkWNicPZb59ITqsoqeLAG1qtTDWRBt9lylfnMrwnLy0
TZIPIt3hYNJUZV9SoJCJ1LzEoe4kH6VHk4v1VnJGOooyBFfFmx109
TycUqS0hTzDm7TX3EVkQb6bq7mtApBHWkCam2BI6Lf056QrRDy
V6tfm15SXVlMjpX5sKJVB2DGnssujT6F0iGrgsf6LQYXnM5yy24arza
qSzAtiFHb6bW6V6RaIzZ0jcuIzKH77jE7XUUUxlpq5vPmSDCXJ5T8
R5o8Dj3gvilHAIHsvttnwF87kjiftfWvjnrAk9qPhVYZuSjtFxFWODTXhx
UuTzHfx6tn87biAEbo0G89o0h4qj2XI0gvevgOf6Q5s1xqX7Rfv3kC9
ODHzWFmgZBv2i93tPsm9O3vsfawiFVaSbiM8eKs1WUzU92bHt5t1l
USxr0EpZHRGaeYvy6zv3oSjYs6aCxSo9f7qtsFjcsr8oDRs3aSnxLL
```

VZFT1qKT0I9ppwWR4jOoTSQH3EF2ORVDziDxRi91W7pGjPjeBR6  
3AGIMczFB0Jp6Z0DmQkZZmPuCrGjALiL0pZcJnMbQB2g29QI1HZP  
yGf4ujar3JA7Ds5ru0xByLxPzUb0

## Randez vous e single-flux

Come affermato nei paragrafi introduttivi, l'utilizzo della tecnica del *single-flux* (derivata dal *fast flux*) permette di risolvere il problema del randez vous.

Con il termine *fast flux* si fa riferimento ad una tecnica utilizzata nelle botnet basata su DNS che si lega alla combinazione di reti peer-to-peer, sistemi C&C distribuiti, load balancing del web e redirectione di proxy. Permette di rendere le reti di malware più resistenti rispetto alla loro individuazione e alle contromisure adottabili.

Precisamente, con il termine *single-flux* invece, ci riferiamo ad una tipologia di fast flux caratterizzata da molti nodi che all'interno della rete registrano e cancellano il proprio indirizzo come parte della lista degli indirizzi DNS di tipo A per un singolo dominio. Questo sistema sfrutta valori molto bassi di Time To Live (TTL)<sup>1</sup>, per creare una lista dinamica di indirizzi in continuo cambiamento.

In questo modo tutte i bot possono conoscere sempre l'attuale C&C anche dopo che i meccanismi di rallying hanno modificato la struttura della rete.

## 2.2 Funzionamento dell'applicazione basata sui ruoli

In questa sezione analizzeremo i flussi esecutivi dei casi iniziali relativi al funzionamento di BiTViR. Vedremo come reagirà l'applicazione a seconda del ruolo assunto, come già spiegato nella "struttura della botnet". Sarà possibile osservare in dettaglio l'ordine e la consistenza dei messaggi scambiati dalle varie entità della botnet: bot, C&C e mock DNS. Infine, sarà mostrata la realizzazione grafica del *front-end* accessibile dal sito web.

Si noti inoltre che le seguenti sottosezioni sono da intendersi come "*ciò che può fare l'applicazione nel ruolo rispettivamente di bot, di C&C, di C&C (sito)*", differentemente da ciò che intendiamo quando parliamo di *peer*, ovvero l'applicazione a prescindere dello stato in cui si trova (e del ruolo ricoperto) intesa come entità generica della rete.

---

<sup>1</sup>Il TTL indica periodo di validità di un certificato

### 2.2.1 Bot

Il bot effettua diverse azioni, mostreremo qui il funzionamento in dettaglio, fase per fase, della prima connessione al C&C, del flood<sup>2</sup> dei messaggi e dell'esecuzione di un comando ricevuto tramite vicinato.

#### First access:

Questo procedimento (illustrato alla figura 3) viene eseguito ad ogni avvio del bot e permette di risolvere il problema del rendez vous:

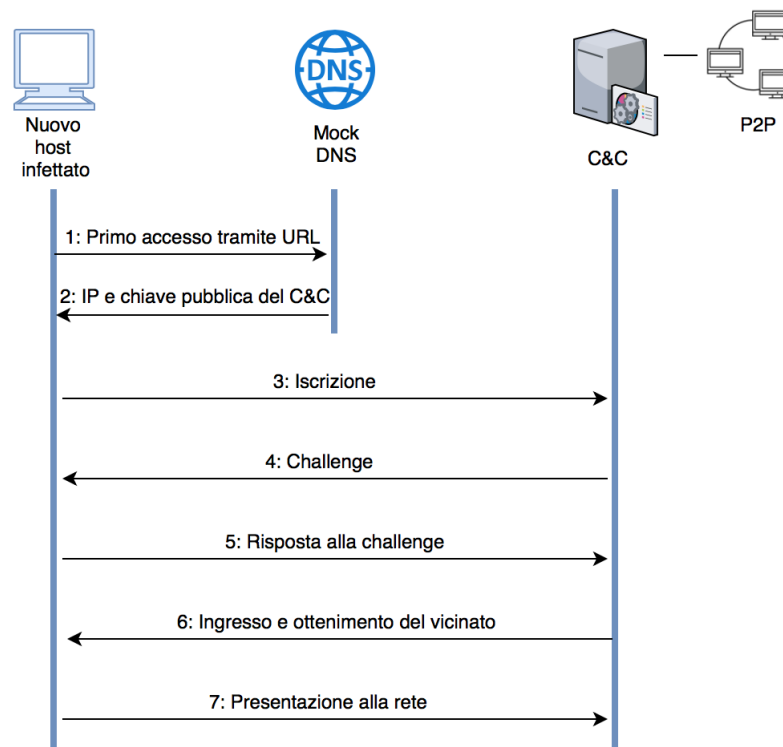


Figura 3: Prima connessione di un bot verso il C&C.

1. Il bot genera una richiesta POST al DNS utilizzando un preciso URL: 005myvt.mwsvr1.com, il quale ritornerà l'indirizzo relativo al mock DNS
2. Il bot genera una nuova richiesta GET all'indirizzo ottenuto al punto precedente aggiungendovi */ciaosounounbot* nell'URL;

<sup>2</sup>Operazione di broadcast di un messaggio in una rete.



3. Il DNS risponderà con la coppia IP e chiave pubblica dell'attuale C&C, indirizzo sfruttabile per effettuare l'accesso al sistema;
4. Il bot contatta il C&C effettuando una richiesta d'iscrizione, inviando il proprio ID firmato dalla sua chiave privata;
5. Il C&C, tramite l'ID del bot ricevuto, tenta di recuperare dal database la chiave pubblica del bot e verificare la firma apposta al messaggio; in caso di esito positivo, si veda direttamente il punto successivo, altrimenti inizia una fase di *challenge*, termine che fa riferimento ad una procedura three-way logicamente equivalente ad un *handshake* con la differenza che l'esito viene decretato dalla risoluzione di un puzzle richiesto dal C&C e risolto dal bot;
  - (a) Il C&C invia al bot (attualmente non identificato) una *challenge* da risolvere;
  - (b) Grazie ad un particolare metodo implementato nell'applicazione (ed illustrato al capitolo successivo) ed grazie ai parametri passati dal C&C, il bot può calcolare il risultato della challenge ed inviarlo al C&C;
  - (c) Il C&C, a questo punto, convalida la risposta del bot e, se l'esito è positivo, assegna un vicinato al bot restituendoglielo in risposta; termina quindi la fase di *challenge*;
6. Il C&C in questo punto della comunicazione conosce il bot; quest'ultimo si unisce alla rete aggiornando i propri parametri, ovvero: riceve il vicinato, lo aggiunge alla propria lista di vicini (finora vuota) e comunica al C&C di essere pronto inviando alcune prime informazioni relative alla macchina infettata; infine, rimane nello stato di attesa.

### **Flood di un messaggio:**

1. L'utente invia una richiesta di flood al sito e ne genera il comando;
2. Il sito inoltra al C&C la richiesta interna: costruisce il messaggio a seconda del comando, lo firma e lo invia cifrato;
3. Il C&C inizia la fase di flood informando i suoi bot vicini che, inoltre: ricevono il messaggio, lo decifrano verificando la sua validità tramite controllo della firma, verificano che il messaggio non è stato già ricevuto precedentemente, parsano il comando e lo eseguono; infine, il messaggio viene inoltrato al vicinato di ciascun bot, ad eccezione del mittente del messaggio principale;
4. I bot, a loro volta, ripetono il comportamento del punto precedente ed propagano il messaggio di vicinato in vicinato.

Il costo computazionale di questa operazione è  $\mathcal{O}(n)$ , con  $n$  numero di bots nella rete.

### **Ping verso il vicinato**

Come richiesto nel punto II delle specifiche, è stato impostato un contatto periodico tra i vari peer della rete e lo abbiamo sfruttato per effettuare una continua operazione di ping in modo da capire quali peer vicini sono ancora connessi alla rete, rendendola periodicamente aggiornata.

Più in dettaglio, la sua implementazione prevede che il bot effettui con periodicità un'operazione di *ping* verso i suoi vicini per vedere quali bots risultano ancora attivi; se alcuni di essi non dovessero risultare raggiungibili, saranno raccolti in una lista comunicata al C&C, il quale aggiornerà la lista dei peer connessi ed il grafo dei vicini.

I tempi sono impostabili tramite un file di *properties* presente in BiTViR.

### **Esecuzione degli attacchi:**

Le operazioni di flood permettono la propagazione della richiesta di attacco da effettuare a tutta la rete; l'attacco viene scelto dall'utente tramite l'apposita pagina raggiungibile dal pannello dell'utente connesso e andrà eseguito dalla porzione di bot informati coinvolti.

Esistono moltissimi tipi di attacco; nell'ambito di questo progetto, decidiamo di fornirne solo pochi esempi, spiegando le difficoltà incontrate e le eventuali possibili espansioni future del progetto. Ciò non è che una limitazione dell'enorme quantità di attacchi che è possibile implementare ad un applicativo di tale portata; tuttavia ci limitiamo a fornirne un discorso introduttivo, in quanto parte non richiesta nelle specifiche del progetto.

#### ***Esempio di attacco: SynFlood***

Il SynFlood è un attacco di tipo DDoS *Distributed Denial-of-Service* in grado di inondare il server *target* di richieste SYN lasciandolo in uno stato di *half-open*, ovvero in attesa di ACK mai recapitati dai bot che mantengono aperte le connessioni in tempo indefinito, non permettendo al TCP three-way handshake di concludere con successo la richiesta di connessione.

Questo attacco provoca un *denial of service*, ovvero un malfunzionamento che impedisce agli utenti di accedere ai servizi forniti dal server target, saturando il traffico in entrata.

Teniamo in considerazione lo stack Internet del TCP/IP; utilizzando Java, purtroppo, non siamo potuti scendere di livello (in questo progetto abbiamo lavorato perlopiù nel livello di applicazione) in modo da poter usufruire dei servizi forniti al livello di trasporto. Dopo esserci appositamente documentati su

Internet, abbiamo scoperto che, per raggiungere lo scopo, si ha la necessità di includere nel progetto particolari librerie (diverse per ogni OS) che, se non installate nel sistema, non permettono il corretto funzionamento dell'applicazione.

Inoltre, anche provando a riscrivere la classe relativa alla gestione dei socket, non si riesce ad arrivare all'invio del primo pacchetto TCP. Questo ci ha impossibilitato a realizzare un attacco in grado di sfruttare il SYN del TCP direttamente; è risultato dunque impossibile procedere con la costruzione di un pacchetto TCP ad hoc in modo da sfruttare tecniche d'attacco migliori come il *reflection attack*<sup>3</sup>.

Una valida alternativa potrebbe essere l'esecuzione di un file C o Python esterno all'applicazione Java, ma, anche in questo caso, non si è sempre sicuri del corretto funzionamento. Ciò ci ha portato all'unica implementazione disponibile: sfruttando l'apertura di molteplici socket da diversi thread contemporaneamente, simuliamo un attacco che, esteso alla rete distribuita, provoca un DoS verso il target.

Abbiamo constatato poi che, altri tipi di attacchi, possono essere similmente implementati, tenendo in considerazione il discorso precedente sulla necessità di scendere di livello per ottenere al meglio il risultato desiderato. Qualsiasi altro attacco, dunque, risulta implementabile in una eventuale espansione futura del progetto; noi stessi siamo entrati in possesso di codice Java (non implementato per le motivazioni suddette) riguardo la creazione di keylogger, screen monitor (l'invio di fotografie scattate dalla webcam senza il consenso del fotografato) e stream webcam (l'attivazione della webcam del target ignaro).

## 2.2.2 C&C

L'applicazione, nel ruolo di C&C, permette di effettuare azioni di diversa natura: *azioni automatiche*, ovvero routine per la gestione della rete, indirizzate al mantenimento delle proprietà iniziali nel tempo (quali ad esempio: topologia, nascita e caduta dei nodi, flood dei comandi, migrazione del C&C, ...), ed *azioni a chiamata*, ovvero flussi esecutivi evocati quando necessario.

### Elezione di un C&C:

Si tratta di un'azione *automatica*.

---

<sup>3</sup>L'attacco conosciuto come *reflection attack*, invece di colpire direttamente la vittima, dirige il suo traffico verso un host intermedio (reflector) che lo redirigerà verso la vittima. Sfrutta la tecnica dell'IP spoofing: l'attaccante genera un pacchetto con l'indirizzo sorgente della vittima e l'indirizzo di destinazione del reflector. Il reflector risponde con un pacchetto che però, a causa dello spoofing, avrà come indirizzo quello della vittima. La vittima quindi riceverà pacchetti provenienti dal reflector e non riuscirà a risalire all'attaccante vero.

1. Lo scheduler dei cron-job<sup>4</sup>, allo scadere di un intervallo di tempo prefissato, fa partire la routine per la nuova elezione del C&C. La routine reperisce tramite il database la lista dei bot eleggibili e ne estrae uno random; a questo punto, il C&C proverà a contattare il bot designato con il fine di “trasferirgli il potere”;
2. Il bot designato risponde positivamente ed inizia lo scambio dei dati: inizialmente vengono chieste le entità del database, e successivamente viene memorizzata la topologia della rete attraverso la creazione di un grafo. Una volta terminato lo scambio delle informazioni, il bot informa il C&C di aver ricevuto tutti i dati necessari;
3. A questo punto, il C&C aggiorna i campi IP e chiave pubblica in Dn-SViR con i valori del nuovo bot eletto ed avvia un’operazione di flood per notificare a tutti i bot della rete l’aggiornamento di C&C;
4. Infine, il vecchio C&C entra in uno stato di *backup*, ovvero: per la rete non risulta essere più il C&C, ma a differenza degli altri bot continuerà a mantenere le informazioni dell’attuale C&C relative alla rete fino a quando non ci sarà un’ulteriore elezione. In questo modo, in caso di attacco verso il C&C attuale, o nel caso in cui il C&C vada offline per qualche motivo, il vecchio C&C manterrà le informazioni e potrà riprendere il controllo della rete qualora fosse necessario.

L’illustrazione del procedimento è posto alla figura 4 nella pagina seguente.

Senza dubbio possiamo affermare che questo processo di elezione si evoca in totale autonomia secondo valori di tempo decisi durante la fase progettuale di BiTViR (ed ulteriormente personalizzabili).

Si noti che: l’applicazione è già basata per la gestione di più C&C, mancano solo i messaggi di scambio e delle informazioni tra i vari C&C. Purtroppo a causa delle limitazioni di 5 computer, non siamo riusciti a testare appieno questa parte ed abbiamo preferito implementare questo metodo.

### **Invio comandi:**

Si tratta di un’azione *a chiamata*.

Il C&C riceve i comandi dalle richieste generate dagli utenti che navigano sul sito; a questo punto, il C&C si occuperà di certificare ed avviare le operazioni di flood nella rete. Precisamente:

---

<sup>4</sup>*cron* è una utility di Linux che permette di fare lo scheduling dei comandi indirizzato ad un server in modo da eseguirli automaticamente in un preciso momento di una precisa data. Con il termine *cron-job* ci si riferisce al job schedulato.

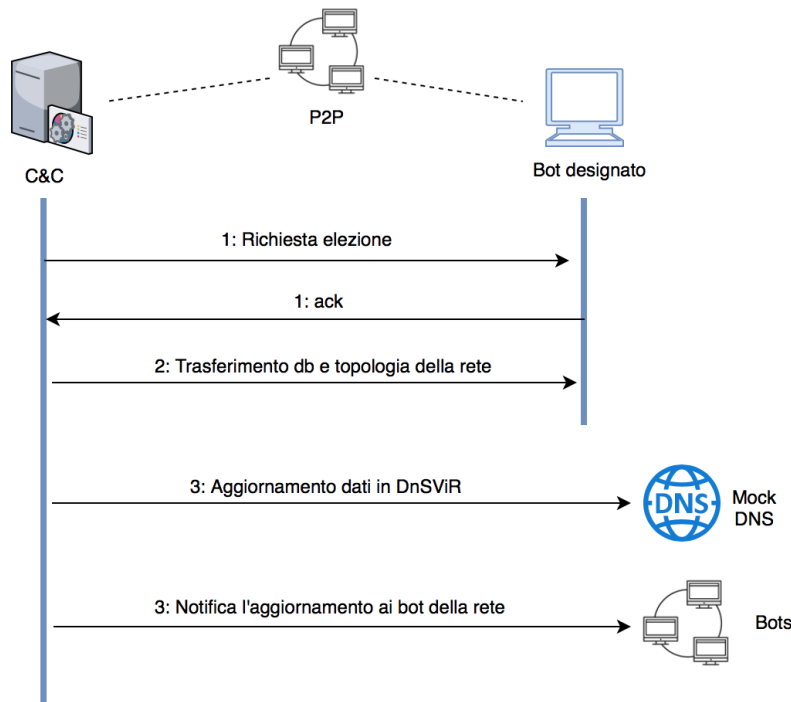


Figura 4: Elezione di un nuovo C&C.

1. L'utente genera una particolare richiesta (passando i parametri relativi alla scelta di attacco che si intende effettuare) verso il C&C cliccando su un bottone presente nella GUI del sito;
2. Il C&C riceve il messaggio ed i suoi relativi parametri, che definisce l'attacco da eseguire. Come già visto nel flood, si dovrà utilizzare una codifica particolare per gli attributi del messaggio, in modo da certificarlo ed infine viene avviata un'operazione di flood inviando il messaggio al vicinato.

### Aggiornamento topologia rete e vicinato

Si tratta di un'azione *a chiamata*. Ci teniamo a puntualizzare tuttavia che la procedura risulta *a chiamata* per il C&C, poiché evocata in risposta ad una procedura *automatica* che, ad intervalli regolari di tempo, fotografano la rete e ne notificano gli aggiornamenti.

Osserviamo che ciò ci permette con efficienza di mantenere le proprietà del grafo, rispettando le assunzioni elencate all'inizio del capitolo. Eventuali future implementazioni di protocolli di rete *instance vector* eviterebbero il controllo

centralizzato della rete implicato nelle operazioni di aggiornamento.

### 2.2.3 Sito internet

Vediamo ora la struttura del *front-end* e quindi la realizzazione del sito internet in grado di comunicare con i bot attraverso pannelli user-friendly minimalisti: uno stile che sposa perfettamente l'idea di BiTViR secondo il nostro immaginario, in netto contrasto con la potenzialità offerta nel back-end, lontano dagli occhi dell'utente.

Alla fine del capitolo sono mostrati gli screenshot relativi (figura 6) a tutte le pagine web che saranno qui di seguito citate.

#### Save Walter White!

Il sito è stato dotato di due differenti home page (che chiameremo *index*). L'*index* principale, al quale è possibile accedervi tramite URL, si presenta come una vecchia pagina HTML anni '90 ispirata ad un ipotetico sito internet appartenente all'universo di Braking Bad, una nota serie TV del momento (vedi figura 7).

L'*index* secondaria, invece, si riferisce, come vedremo a breve, rispettivamente al pannello admin o al pannello user; per potervi accedere si necessita di una *codifiche particolari* che saranno illustrate nel prossimo capitolo.

Questa pagina è l'unica accessibile da protocollo HTTP non sicuro.

#### Log in

Prima di poter accedere alle pagine index è necessario autenticarsi ed assumere un ruolo. La pagina di login si presenta con una form nella quale possiamo inserire username e password per accedere al pannello principale di BiTViR (vedi figura 8).

Il procedimento percorre i seguenti passi (vedi figura 5):

1. L'utente genera una richiesta GET al DNS utilizzando un preciso URL mappato dall'applicazione (vedi */ciaosonounumano*) tramite codifica particolare;
2. Il DNS risponderà reindirizzando l'utente all'attuale C&C;
3. L'applicativo dell'utente (ad esempio Firefox) contatta il C&C richiedendo la pagina iniziale;

4. Il C&C risponde con l'*index principale* (la pagina fake);
5. L'utente genera una richiesta particolare per ottenere la pagina di login, mostrando così di essere un utente iscritto a BiTViR intenzionato ad accedere al proprio pannello; la richiesta particolare deve essere generata come da specifiche: noi abbiamo scelto che da questo momento in poi, tutte le richieste (questa inclusa), avverranno attraverso un certificato rilasciato dall'amministratore all'user (vedi capitolo successivo, Crittografia);
6. il C&C risponde con la pagina di login;
7. L'utente, a questo punto, procede con l'autenticazione inserendo i propri dati nella form;
8. Il C&C verifica e convalida l'autenticazione dell'utente e risponde con l'*index secondaria*, mostrando rispettivamente il pannello admin se il login è stato effettuato da un utente amministratore o con il pannello user altrimenti.

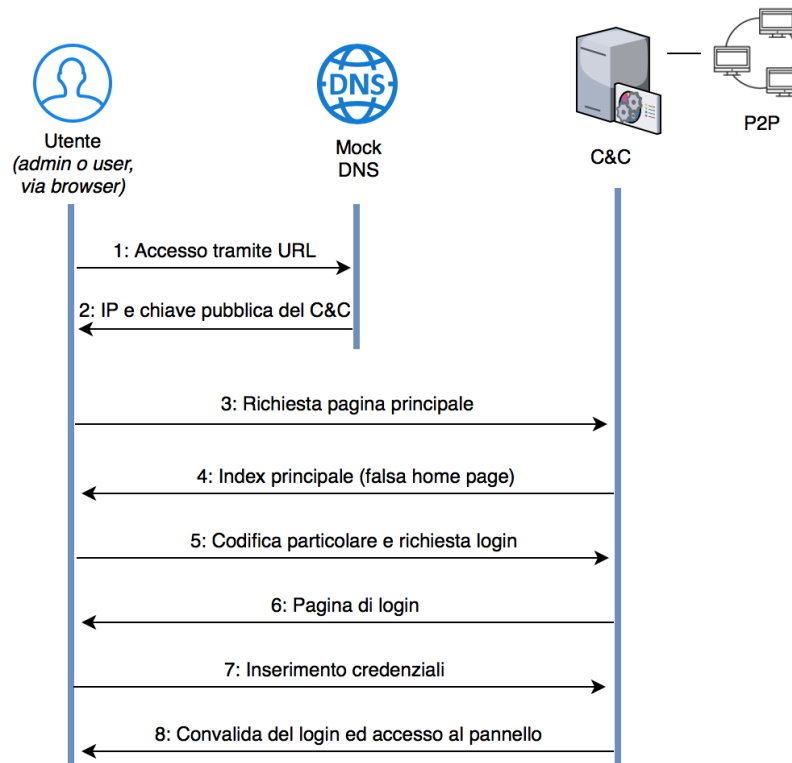


Figura 5: Log in dell'utente.

### **Pannello *admin***

Effettuando il log in da amministratore sarà possibile sfruttare il menù posto sulla sinistra per accedere ai seguenti pannelli (vedi figura 9 relativa al pannello admin):

- *Registration*: permette la creazione di nuove utenze in grado così di accedere alla botnet; vedi figura 11;
- *Add bot*: permette l'assegnazione dei bot agli user della rete; vedi figura 12;
- *Delete bot*: permette la revoca dei bot assegnati agli user della rete; vedi figura 13.

### **Pannello *user***

Il menù posto sulla sinistra permetterà all'utente di accedere ai seguenti pannelli (vedi figura 10 relativa al pannello user):

- *Bots*: permette di visualizzare la porzione di bots assegnati all'utente autenticato. Questa pagina può essere considerata, come richiesto nell'estensioni di BYOB2, come un file di testo nel quale vengono visualizzati la lista dei bots assegnati all'utente. Nel momento in cui l'utente visualizza la pagina con la lista dei bot può facilmente salvare l'intera pagina come documento HTML ottenendo la lista formattata e con un CSS allegato, invece che solo il file di testo con le coppie di ID (anche l'admin può avere accesso a questa pagina, ma non vede solo la porzione di bot assegnati ma tutta la botnet);
- *Attack*: permette di inviare una richiesta di attacco ai bot assegnatogli; vedi figura 13.

### **404 Not Found**

Qualsiasi URL si decida di comporre per tentare una navigazione sul sito si otterrà un errore di tipo 404 Page Not Found; abbiamo deciso di sbizzarrirci e mostrare all'utente la cara vecchia pagina d'errore tipica delle più datate versioni di Internet Explorer (vedi figura 6). Un omaggio nostalgico che si legava bene con lo stile adottato per l'index principale.

Questo comportamento è stato permesso dall'implementazione di una classe Java per l'*error handler* in grado di mappare gli “/error” intercettati durante le richieste al server che non trovano un mapping valido all'interno del controller del sito. Tutte le eccezioni reindirizzeranno alla pagina di 404, confondendo l'utente non autorizzato.








## The page cannot be displayed.

The page you are looking for is currently unavailable. The Web site might be experiencing technical difficulties, you may need to adjust your browser settings.

---

Please try the following:

- Click the  **Refresh** button, or try again later.
- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- To check your connection settings, click the **tools** menu, and then click **Internet Options**. On the **Connections** tab, click **Settings**. The settings should match those provided by your local area network (LAN) administrator or Internet service provider (ISP).
- If your Network Administrator has enabled it, Microsoft Windows can examine your network and automatically discover network connection settings.  
If you would like Windows to try and discover them, click  **detect Network Settings**.
- Some sites require 128-bit connection security. Click the **Help** menu and then click **About Internet Explorer** to determine what strength security you have installed.
- If you are trying to reach a secure site, make sure your Security settings can support it. Click the **Tools** menu, and then click **Internet Options**. On the advanced tab, scroll to the Security section and check settings for SSL 2.0, SSL 3.0, TLS 1.0, PCT 1.0.
- Click the  **Back** button to try another link.

Cannot find server or DNS Error  
Internet Explorer

Figura 6: 404 Page Not Found.

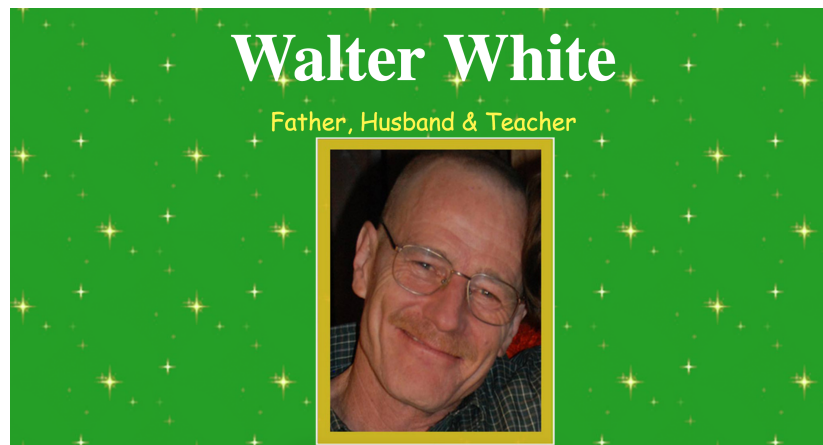
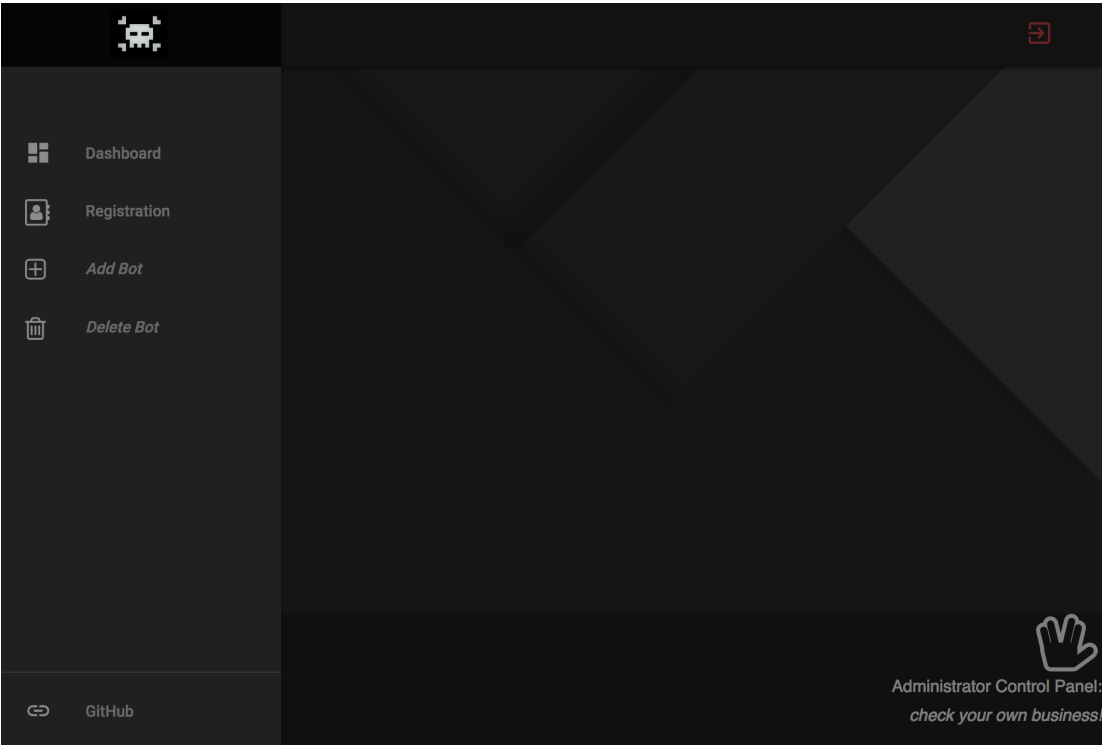


Figura 7: Index principale

A dark gray login form with a subtle geometric pattern. It contains three input fields: the first is labeled "Username" with a person icon, the second is labeled "Password" with a lock icon, and the third is a checkbox labeled "Remember me". Below these fields is a large, bold, white "LOGIN" button.

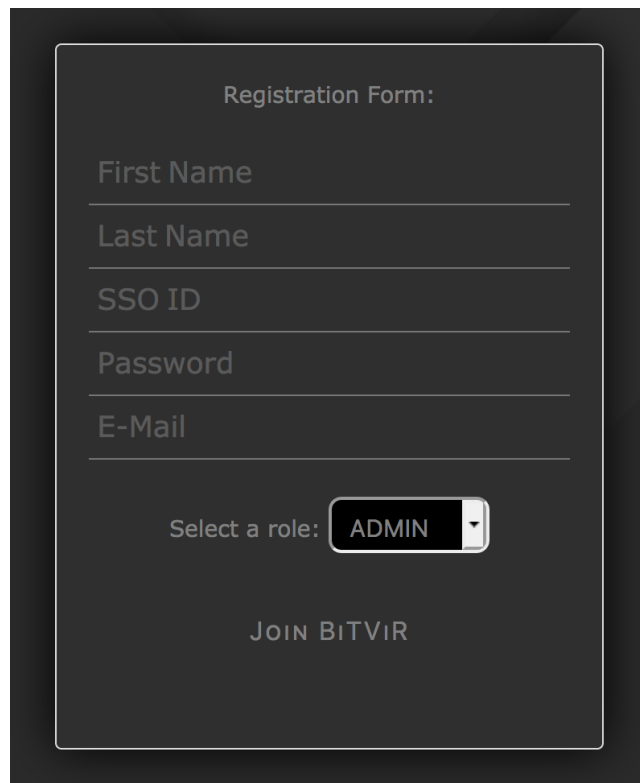
Figura 8: Login



(a) Figura 9: Index secondaria: pannello admin



(b) Figura 10: Index secondaria: pannello user



Registration Form:

First Name

Last Name

SSO ID

Password

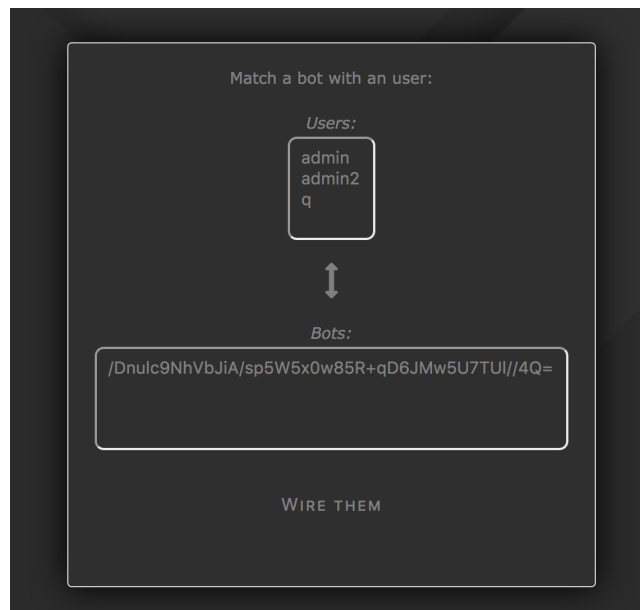
E-Mail

Select a role: ADMIN

JOIN BITVIR

The registration form is a dark-themed interface with a white border. It contains six input fields for user registration: First Name, Last Name, SSO ID, Password, and E-Mail. Below these fields is a dropdown menu for selecting a role, currently showing 'ADMIN'. At the bottom of the form is a button labeled 'JOIN BITVIR'.

(a) Figura 11: Registrazione di un nuovo utente



Match a bot with an user:

Users:

admin  
admin2  
q

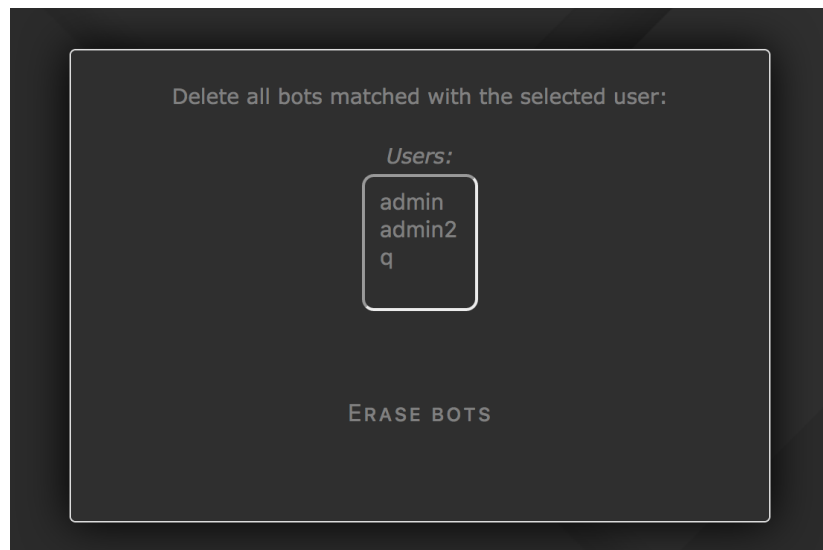
Bots:

/Dnulc9NhVbJiA/sp5W5x0w85R+qD6JMw5U7TUI//4Q=

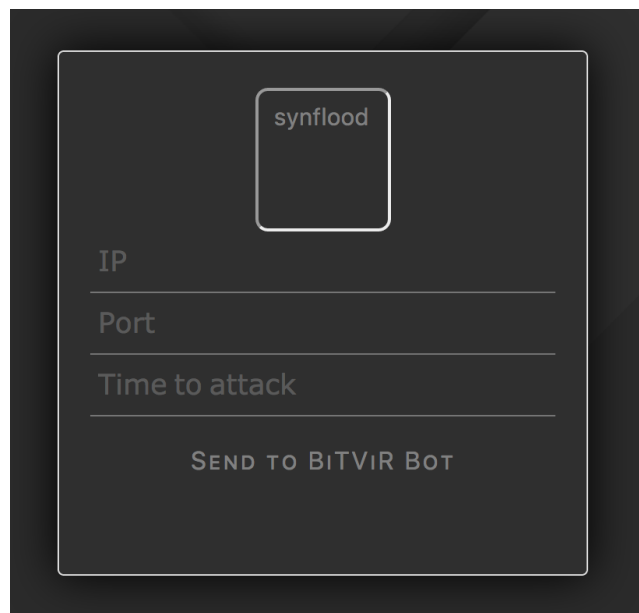
WIRE THEM

The 'Match a bot with an user' interface is a dark-themed screen. It features two lists: 'Users' containing 'admin', 'admin2', and 'q', and 'Bots' containing a long alphanumeric string. A double-headed vertical arrow is positioned between the two lists. At the bottom is a button labeled 'WIRE THEM'.

(b) Figura 12: Assegnamento di bots ad un utente



(a) Figura 12: Rimozione delle allocazioni di bots ad un utente



(b) Figura 13: Effettuare un attacco

### 3 Crittografia

In questa sezione analizzeremo i dettagli implementativi relativi alla crittografia coinvolta nel progetto che sono stati ritenuti più adeguati durante la fase di progettazione di BiTViR. Il nostro intento fin da subito è quello di sfruttare metodi di codifica noti per essere ritenuti i più efficienti tra quelli pubblici, in modo da ottenere uno “strato” di sicurezza con estrema semplicità (grazie alla grande mole di documentazioni e librerie reperibili su Internet) che risulti quasi inviolabile.

Questa sicurezza nella crittografia coinvolta viene logicamente applicata ai metodi che regolano le interazioni tra le entità della botnet analizzate al capitolo precedente.

È importante sottolineare che la nostra applicazione è dotata di *due coppie* differenti di chiavi (pubblica e privata): una coppia *primaria* verrà usata per effettuare la cifratura dei messaggi da inviare nella rete, una coppia *secondaria* per firmare i dati. Quest’ultima coppia è necessaria ad irrobustire la sicurezza dell’applicazione poiché è sempre consigliato non riciclare la stessa chiave per funzionalità diverse; l’utente, infine, ne viene a conoscenza dopo l’operazione di challenge iniziale.

L’ausilio di Spring Security e Bouncy Castle, come vedremo, ci ha permesso così di usufruire facilmente di: HTTPS, TLS/SSL, RSA, AES, SHA256, HMAC e dei certificati e di essere personalizzate con estrema facilità.

#### HTTPS

Prima di iniziare a spiegare quali metodi abbiamo implementato in BiTViR e come funzionano, è bene sottolineare che tutti gli scambi di messaggi tra le varie entità della rete avvengono esclusivamente attraverso il protocollo HTTPS sopra descritto. Inoltre, non contenti, abbiamo deciso di personalizzarlo in modo da accettare solo ed esclusivamente le ultime versioni, rendendo disponibili solamente alcune specifiche triple da noi scelte, in modo da garantire un altissimo livello di sicurezza.

Per queste richieste è stato generato un certificato diverso da quelli che verranno utilizzati dalle nostre reimplementazioni spiegate nei paragrafi successivi.

Il TLS (Transport Layer Security), successore di SSL (Secure Sockets Layer), è un protocollo crittografico del livello di presentazione (e quindi lo strato applicativo dello stack TCP/IP) molto usato, in grado di fornire una comunicazione sicura end-to-end.

Le triple di algoritmi SSL utilizzati dalla nostra applicazione sono:

- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384;

- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256;
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256;
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256;
- TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256.

Constatiamo che, inserendo questo strato, abbiamo reso le comunicazioni di BiTViR sicure dall'ascolto e dall'interferenza da parte di persone indesiderate. Ciò significa che due peer (qualsiasi entità della rete essi siano), una volta eseguito il protocollo SSL, potranno comunicare in modo sicuro.

In ogni caso, questo non ci permette ancora di garantire che il peer con il quale stiamo comunicando non sia un'entità malevola intenta a fingersi tale; per questo motivo sono stati implementati tutti i successivi metodi di codifica ed autenticazione degli utenti che completeranno il capitolo a seguire.

Ci teniamo ad iniziare da una premessa riguardante la cifratura asimmetrica coinvolta, fornita dall'algoritmo RSA (Rivest-Shamir-Adleman, basato sul sistema creato nel 1976 da Whitfield Diffie e Martin Hellman). Questo algoritmo viene utilizzato sia in modalità di cifratura dei dati (dopo la fase di handshake del TLS) sia per permettere ai C&C di firmare i comandi destinati alla rete. RSA fornisce, infatti, un metodo crittograficamente sicuro per lo scambio di una chiave di sessione tra due delle entità della rete. Tale chiave verrà usata per le comunicazioni con cifratura simmetrica, meno costosa e più immediata.

### 3.1 Autenticazione

Abbiamo implementato varie tipologie di autenticazione, necessarie a rendere più resistente e dettagliato l'intero processo che lo regola. Avere a disposizione diverse implementazioni ci permette di affrontare l'autenticazione con ogni evenienza; talvolta può ritenersi necessaria la conoscenza da parte del bot di un'informazione, talvolta ci si può affidare al certificato, e così via.

Vediamo nel dettaglio quali tipi di autenticazione vengono forniti in BiTViR, distinguendone i metodi che permettono l'handshake tra i vari peer della rete, ovvero: tra utente $\leftrightarrow$ C&C, tra bot $\leftrightarrow$ bot, e tra bot $\leftrightarrow$ C&C. Lo scenario che analizzeremo ora è quello in cui sono presenti due diverse entità che vogliono condividere un segreto (la chiave che sarà possibile utilizzare per le comunicazioni future).

Spring Security permette la gestione dei ruoli e automaticamente delle pagine con i loro permessi, ovvero, tutta la sottosezione *admin* è disponibile solo a chi si autentica utilizzando le credenziali di un utente amministratore salvato

nel database. Stessa cosa avviene per il profilo utente.

### Tramite challenge iniziale

All'avvio di BiTViR, il bot tenta di iscriversi alla botnet comunicando con il C&C. Per ottenere ciò, abbiamo implementato due differenti metodi di challenge: una nel caso in cui il bot non si sia mai iscritto alla rete, ed una nel caso in cui lo abbia già fatto in precedenza.

Nel primo caso (*prima iscrizione del bot*, vedi flusso al capitolo precedente), la challenge presentata da BiTViR è stata interamente progettata ad-hoc da noi e si basa sulla risoluzione di un puzzle tramite un particolare algoritmo implementato nell'applicazione e su una chiave implementata nel codice (il segreto). Ciò permette al bot di autenticarsi con il C&C dimostrando di conoscere tale segreto, e quindi la chiave. Per evitare che venga utilizzata sempre la stessa chiave nelle autenticazioni, verrà eseguito un particolare algoritmo che opera come descritto qui di seguito:

1. il bot invia una richiesta di challenge al C&C;
2. il C&C invia un messaggio cifrato in AES al bot (vedi cifratura tramite AES in seguito), contenente due numeri random il primo dei quali indica il numero di iterazioni necessarie dell'algoritmo;
3. Il bot, a questo punto, utilizza un seme (*seed*) contenuto nel codice col fine di generarvi numeri random e fermarsi solo una volta che il numero delle iterazioni equivale al valore inviato dal C&C. Il risultato di queste iterazioni lo chiameremo *chiave challenge*. A questo punto tale chiave viene cifrata tramite HMAC con l'ausilio del secondo numero random ricevuto dal C&C. Il valore ricavato, infine, viene inviato al C&C;
4. Il C&C ne conferma la validità. Questa chiave verrà utilizzata per uno scambio iniziale di alcune informazioni tra bot e C&C, tra i quali la chiave pubblica del bot che sarà utile per firmare le successive comunicazioni.

Analizziamo la robustezza della challenge proposta: iniziamo che abbiamo una chiave da 64 bit che è a conoscenza del C&C e del bot che si sta escludendo; inoltre, un malintenzionato potrebbe collezionare molteplici pacchetti di challenge-response, vista l'alto traffico generato tra i vari peer; quindi, abbiamo fatto inviare dal C&C al bot due valori da considerare come vettore di inizializzazione, in modo da poter aver  $2^{32} \times 2^{64}$  possibili combinazioni di challenge per un'unica chiave. Ciò rende molto più costosa la memorizzazione di tutte le possibili challenge in modo da poter applicare metodi di criptoanalisi su di esse.

Nel secondo caso (*il bot è già iscritto alla rete*), invece, poiché il server già conosce la chiave pubblica del bot e quest'ultimo risulta autenticato tramite



challenge, allora il bot stesso invierà tramite RSA il proprio *ID* univoco cifrato tramite chiave pubblica del C&C e firmato con la propria chiave privata.

Il C&C decifrerà il messaggio e lo verificherà tramite la chiave pubblica (corrispondente all'*ID* ricevuto) recuperata dal proprio database. Se non la trova, presenta la challenge precedentemente illustrata.

Analizziamo la robustezza della seconda challenge proposta: si basa su RSA con chiavi a 4096 bit, più che sufficienti per le tecnologie odierne.

In definitiva, riassumiamo dicendo che, la challenge coinvolge tutte le comunicazioni  $bot \rightarrow bot$  e la prima comunicazione  $bot \rightarrow C\&C$ .

### Tramite user-agent

Come richiesto nel punto *v* delle specifiche di BYOB e nel punto *vi* di BYOB2, abbiamo implementato un'ulteriore autenticazione automatica che avviene su tutte le richieste tra due peer, attraverso il campo *user-agent*.

Sfruttando il framework di Spring, abbiamo implementato un metodo automatico che, a tutte le richieste in uscita generate dalla nostra applicazione, aggiunge un campo nell'header chiamato "x-user-agent" ed il ruolo in cui si trova l'applicazione. Grazie a Spring ci è bastato definire un filtro da aggiungere alla catena dei filtri automatica che effettua il *parsing* di ogni richiesta ricevuta, controllando così, tramite *regular expression*, la presenza del campo nell'header HTTPS.

### Tramite certificato

Tenendo a mente le specifiche iniziali (si il punto I di BYOB2), abbiamo fatto in modo che, per accedere al sito correttamente, bisogna presentare una richiesta particolare; abbiamo scelto di far coincidere questa particolarità con l'utilizzo dei certificati SSL tramite protocollo HTTPS.

Come già detto nel discorso introduttivo, tutte le richieste che circolano nella rete avvengono tramite SSL; precisiamo che, in realtà, vi è un vincolo maggiore ossia la *mutua autenticazione* dell'HTTPS: la necessità di presentare un certificato *particolare* durante tutte le comunicazioni.

Questo *particolare* certificato è firmato dalla nostra CA (Certification Authority)<sup>5</sup>. Abbiamo generato una CA con un certificato a 8192 bit che rimarrà

---

<sup>5</sup>La CA è un ente di terza parte fidato, pubblico o privato, abilitato a rilasciare un certificato digitale tramite firma con la propria chiave privata in grado di attestare l'autenticità delle identità legate ad una chiave pubblica.

in possesso solamente dell'amministratore della botnet. I certificati che abbiamo generato e che saranno illustrati a seguire, per essere accettati durante la mutua autenticazione, devono essere firmati esclusivamente dalla nostra CA; l'applicazione quindi rifiuta automaticamente tutti gli altri certificati.

I certificati si presentano in due differenti forme, ovvero:

**certificato dell'applicazione:** è presente in BiTViR un certificato firmato dalla CA ed utilizzato durante l'invio di richieste. Quando l'applicazione riceve una richiesta, controlla che il certificato sia correttamente firmato dalla CA, confrontandolo con ciò che ha salvato all'interno di un *trust store*<sup>6</sup>;

**certificato dell'utente:** sfruttando il TTL (Time To Live) possiamo fornire una sorta di abbonamento periodico all'utente; volendo affittare la botnet dietro retribuzione monetaria (esclusivamente in bitcoin) per un periodo limitato ad una nuova utenza, procederemo in questo modo: una volta ricevuto correttamente il pagamento concordato, generiamo un certificato con una scadenza prefissata e lo inviamo all'utente assieme al certificato contenente la chiave pubblica della nostra CA (in modo da poter completare la mutua autenticazione) e i suoi relativi dati di accesso al sito. In questo modo, una volta scaduto il certificato, l'utente non avrà alcuna possibilità di accedere al proprio pannello e lasciare i dati salvati nel server nel caso in cui in futuro decidesse di riaffittarla.

In definitiva, il certificato da noi proposto è il seguente: chiavi RSA da 4096 bit, utilizzando per il calcolo della chiave pubblica lo standard X.509, e per il calcolo della chiave privata lo standard PKCS12.

Per uno sguardo concreto ai certificati contenuti in BiTViR suggeriamo l'elenco dettagliato fornito al capitolo conclusivo della relazione.

## Tramite RSA

L'RSA (Rivest-Shamir-Adleman, 1977) è un algoritmo di crittografia a chiave pubblica che fornisce un metodo di comunicazione e cifratura dei dati in grado di assicurare confidenzialità, integrità, autenticità e non ripudiabilità, proprietà che lo rendono uno degli algoritmi più efficienti al pubblico.

La sua sicurezza è dovuta alla robusta teoria matematica che lo sostiene: sfruttando le proprietà di asimmetria dovute alla presenza di due chiavi per ciascun entità, è in grado di usare entrambe le chiavi per cifrare informazioni e

---

<sup>6</sup>Un *trust store* è un particolare key store, ovvero una lista contenente esclusivamente chiavi pubbliche fidate.

di sfruttare l'altra per l'eventuale decifratura.

Questo algoritmo di crittografia viene utilizzato nei seguenti casi:

- Nelle comunicazioni  $bot \rightarrow C\&C$  quando l'applicazione risulta avviata e dopo essersi già iscritto una prima volta (vedi "tramite challenge iniziale");
- Nelle comunicazioni  $bot \rightarrow bot$ . I bot comunicano con gli altri bot del proprio vicinato (come secondo la topologia della P2P costruita dal C&C). Il vicinato, composto da coppie IP, chiave pubblica, inoltreranno ai vicini utilizzando la stessa challenge effettuata al primo avvio tra bot e C&C; a seguito della challenge, il bot appena connesso si comporterà da bot, mentre quello già iscritto si comporterà da C&C;
- Per firmare le comunicazioni  $C\&C \rightarrow bot$ . Tutti i messaggi vengono creati dal C&C attuale, il quale, a seconda del comando da eseguire, restituirà un diverso messaggio; in ogni caso, il messaggio ritornato verrà firmato con la chiave privata del C&C in modo che i bot, prima di inoltrarlo a loro volta e di eseguirne il comando, possano essere sicuri dell'autenticità del messaggio;

### Tramite login

Vediamo ora come opera l'autenticazione tra utente e C&C (sito).

Quando un utente vuole effettuare il log in al sito per accedere al proprio pannello, può farlo inserendo le credenziali composte da username e password fornite dall'admin nell'apposita form disponibile alla pagina di log in.

Parlando della persistenza, e quindi della sicurezza del nostro database, facciamo notare che: oltre ad aver mascherato il database e ad averlo reso inaccessibile direttamente, abbiamo implementato due diversi tipi di sicurezza; analizziamole.

Il primo tipo riguarda *SQL Injection*, ovvero, prima di effettuare il parsing di eventuali SELECT destinate al database, queste vengono purificate da eventuali caratteri speciali che ne permetterebbero l'aggiunta di comandi al loro interno.

Il secondo tipo di sicurezza è legato alla password memorizzata all'interno della persistenza; ciò avviene tramite l'ausilio di una funzione di hash SHA512 implementata ad-hoc. Questa funzione coinvolta risulta adattiva, ovvero, è in grado di complicare il calcolo di cifratura in modo da contrastare l'aumento di capacità computazionale reperibile nel tempo (e quindi resistere ad attacchi di

forza bruta<sup>7</sup>); inoltre, incorpora un sale<sup>8</sup> legato alla password stessa: tale sale, sfruttato per generare confusione nella password cifrata, si appoggia ad un vettore di inizializzazione costruito con i primi byte della password stessa. Questo evita il riciclo di vettori di inizializzazione pre-impostati (che genererebbero sale identici) e garantisce una difesa concreta contro eventuali vulnerabilità da attacchi tabella arcobaleno<sup>9</sup>.

Tra le precedenti tecnologie coinvolte e non più utilizzate in BiTViR ricordiamo l'algoritmo "bcrypt", il cui sale però risultava identico per ogni password (poiché generato a partire dallo stesso vettore di inizializzazione) e la cui cifratura risultava meno forte rispetto a quella adottata per la versione definitiva del progetto. Ci siamo limitati ad ispirarci alla sua struttura algoritmica, per poi fornirne un'implementazione personale.

## 3.2 Cifratura dei messaggi

Vediamo in dettaglio gli algoritmi proposti per effettuare la cifratura (*encode*) e decifratura (*decode*) dei messaggi scambiati nella rete.

Sono stati implementati algoritmi di cifratura simmetrica ed asimmetrica con le medesime motivazioni che ci hanno spinto ad implementare diversi metodi di autenticazione: avere un panorama più vasto di metodi a disposizione permette di utilizzarli in modo migliore, abbracciando ogni necessità dell'utente e tenendo a mente la priorità posta nella sicurezza del progetto. Inoltre, ciò è stato reso più facile dalle vastissime librerie disponibili tra Spring Security e Bouncy Castle.

### Cifratura simmetrica

Per cifrare i messaggi end-to-end, che coinvolgono coppie di bot, è stato scelto l'algoritmo AES (Advanced Encryption Standard), conosciuto anche come Algoritmo di Rijndael.

AES è un algoritmo di cifratura a blocchi utilizzato come standard americano. Si tratta di una rete a sostituzione e permutazione (e non una rete di Feistel, come avveniva per DES) che implementa il principio crittografico di Shannon

---

<sup>7</sup>Attacco che consiste nel verificare tutte le soluzioni teoricamente possibili fino a che si trova quella effettivamente corretta.

<sup>8</sup>Sequenza casuale di bit che, se utilizzata assieme ad una password verso una funzione (di hash) unidirezionale, fornisce un output statisticamente più resistente ad eventuali attacchi di tipo dizionario, ovvero attacchi che sfruttano parole chiave e parole frequenti nel linguaggio umano. Molti algoritmi di tipo PRNG (Pseudo-Random Number Generator) sfruttano valori di sale pre-impostati.

<sup>9</sup>è una tabella che può essere usata per il recupero delle chiavi di cifratura in chiaro partendo da chiavi in formato hash generate da una funzione crittografica di hash.

(confusione e permutazione). Elabora blocchi di dimensione fissa (128 bit) con chiavi da 128/192/256 bit; abbiamo impostato di base la nostra versione del cifratore AES con le seguenti proprietà: *cipher mode CBC chaining*, e padding PKCS5. Le due chiavi coinvolte son: una per il vettore di inizializzazione CBC (16 byte) ed una per l'AES (128 bit), entrambe memorizzate all'interno del codice.

La cifratura simmetrica viene utilizzata nei seguenti ambiti:

- Lettura e scrittura dei dati su file, in questo modo possiamo salvare informazioni sulla macchina dell'host evitando che possano essere decifrati da qualcuno;
- Comunicazioni  $Bot \rightarrow C\&C$  una volta che i bot sono avviati. Ciò significa che, dopo la prima autenticazione del bot, le comunicazioni verso il C&C avverranno inviandogli come parametro l'*idBot* cifrato con AES.

Per effettuare il flood di un messaggio  $M$  tramite questa tecnica di crittografia viene elaborata la seguente stringa  $M'$  (successivamente cifrata con AES):

- Creazione di un *ID* del messaggio tramite applicazione della funzione di hash SHA256 relativa ad  $M$  e di un valore random denominato *nonce*;
- Separatore ortografico: "|";
- Comando d'attacco richiesto dal bot;
- Separatore ortografico: "|";
- Firma digitale passando come parametri il comando richiesto dal bot e la chiave privata del C&C, esattamente come abbiamo visto nell'*autenticazione tramite RSA*.

## 4 Vantaggi delle scelte implementative

Elenchiamo di seguito i vantaggi ottenuti dalle scelte implementative adottate, spesso coincidenti con le motivazioni che ci hanno spinto ad effettuare determinate scelte invece di altre. Sono stati riscontrati vantaggi nella rete, nell'implementazione del sito e nell'uso della crittografia interna alla rete.

### Rete P2P

Alcuni dei vantaggi più rilevanti di aver implementato una rete peer-to-peer per BiTViR possono essere riscontrati nel fatto che:

- L'entropia del numero di messaggi inviati da ogni bot è asintotica allo 0, affermazione che possiamo vedere sotto altri punti di vista come: *tutti inviano approssimativamente la stessa quantità di messaggi* o, in maniera meno forte *la varianza decresce*; grazie a ciò otteniamo che il C&C si mimetizza tra i bot; quest'ultimi, inoltre, avendo un carico di lavoro distribuito omogeneamente, garantiscono che nessuno all'interno della rete possa assumere un ruolo più centrale rispetto ad altri peer. La diretta conseguenza è che si evitano così colli di bottiglia dovuti ad peer particolarmente lenti (o con scarse capacità computazionali);
- I C&C migrano autonomamente nel tempo, rendendo quasi impossibile prevedere il prossimo spostamento e quindi rintracciare il bot designato a ricevere il comando della botnet; questa tecnica permette di aggirare preventivamente eventuali attacchi tramite sniffer indirizzati a BiTViR;
- La topologia viene generata in maniera random tramite appositi algoritmi secondo il modello di random graph Erdős-Rényi con i vincoli precedentemente illustrati; questo garantisce alta reliability, e, quindi, la rete risulta sia connessa in qualsiasi istante sia aggiornata in tempo reale automaticamente;
- (non implementato, causa deadline): più C&C aumentano la probabilità che la rete possa sempre eseguire i comandi.

### Nell'implementazione del sito

- HTTPS rende sicuro il traffico da sniffing e modifiche non autorizzate;

- Sono stati integrati gli header CSFR<sup>10</sup> per evitare replay attack<sup>11</sup> e le vulnerabilità legate a questo derivato del cross-site scripting<sup>12</sup>;
- La mutua autenticazione permette sia di assicurarsi di effettuare il log in presso il giusto sito sia che il client è in possesso di un certificato valido emesso correttamente dalla nostra CA;
- L'amministratore può regolare l'accessibilità al sito dei vari utenti iscritti sia tramite la distribuzione di certificati ad-hoc sia tramite il pannello fornito in BiTViR nel *back-end* per la gestione degli utenti; ciò permette di tener traccia delle utenze anche dopo la scadenza del certificato, facilitando futuri rinnovi;
- L'amministratore, tramite il suo pannello, può sia assegnare nuovi bot ad un utente che revocargli i bot precedentemente assegnatigli;
- L'utente, dotato del proprio pannello, può visionare i bot assegnatigli dall'admin ed inviare loro dei comandi; tale comando verrà inoltrato a tutta la rete, ma solo quelli che gli risulteranno assegnati potranno eseguire l'attacco desiderata;
- È disponibile un punto di ripristino all'interno di DnSViR tale che, con l'utilizzo di una chiave segreta memorizzata all'interno del codice, permette di aggiornare il C&C con il valore del primo C&C della rete (eletto come C&C di default).

## Nell'uso della crittografia interna alla rete

- Ogni elemento possiede una doppia coppia di chiavi pubblica e privata per RSA: una coppia sarà utilizzata durante le connessioni SSL, mentre l'altra per firmare i messaggi interni alla rete;
- Tutte le connessioni HTTPS vengono effettuate tramite uno dei due certificati come spiegato precedentemente;
- L'autenticazione al primo avvio viene effettuata tramite una challenge che implica la conoscenza di una chiave segreta e di un algoritmo per effettuare il calcolo della challenge; inoltre il risultato finale viene inviato tramite HMAC;

---

<sup>10</sup>Con il termine *cross-site request forgery* ci riferiamo ad una vulnerabilità a cui sono esposti i siti web dinamici che ricevono richieste da un client senza meccanismi di verifica sull'intenzionalità dell'invio della richiesta o meno.

<sup>11</sup>Il *replay attack* è un attacco che consiste nell'impossessarsi di una credenziale di autenticazione comunicata da un host ad un altro, e riproporla successivamente simulando l'identità dell'emittente.

<sup>12</sup>Il *cross-site scripting (XSS)* sfrutta la fiducia di un utente in un particolare sito, mentre il CSFR sfrutta la fiducia di un sito nel browser dell'utente.

- Solo i C&C possono inviare comandi, i quali vengono appositamente firmati;
- I bot inoltrano ed eseguono solo quei messaggi la cui provenienza è da loro stessi verificata;
- Tutti i messaggi che passano nella rete vengono cifrati ulteriormente in AES;
- I bot possono eseguire piccole operazioni con il vicinato solo dopo che i due bot si sono entrambi autenticati;
- (*non implementati:*) database criptato; aggiornamento automatico della password per la crittografia simmetrica (AES) nel tempo.



## 5 Manuale d'utilizzo

Sfruttiamo questi ultimi paragrafi per illustrare il contenuto del materiale consegnato in allegato alla relazione e spiegare come avviare il progetto in tutte le sue funzionalità.

### Certificati dell'applicativo

L'applicazione è dotata dei seguenti certificati, più volte coinvolti durante l'utilizzo della stessa. Vediamoli in dettaglio:

- *ca.crt*: è il certificato della CA, tramite il quale siamo in grado di firmare i certificati che riteniamo affidabili; è il certificato più importante di BiTViR e rappresenta l'autorità in grado di concordare i permessi all'uso della botnet. Questo file è destinato esclusivamente all'amministratore, che ne farà un uso adeguato firmando tutti e soli quei certificati appartenenti ad utenti della rete regolarmente iscritti;
- *SIKeyStore.jks*: è il certificato tramite il quale, un utente, può facilmente instaurare connessioni SSL ed effettuare comunicazioni sicure;
- *truststore.jks*: contiene tutte quelle chiavi pubbliche di cui un utente può fidarsi con certezza (in primis la chiave pubblica della CA da noi autorizzata);
- *UserSite* si compone di 3 diversi certificati:
  - *UserSite.p12*, certificato contenente la coppia di chiavi relative all'utente iscritto;
  - *UserSite.csr*, certificato estratto dal file P12 contenente la chiave pubblica dell'utente;
  - *UserSite.crt*, certificato analogo al file CSR firmato dalla nostra CA.

### Contenuto del pacchetto rar

All'interno del pacchetto rar in allegato a questa relazione si troveranno i seguenti file:

- *BitVir-0.1.0-Bot.jar* è il JAR relativo alla botnet nel ruolo di BOT (consigliabile nella fase di testing, poiché si appoggia al C&C correttamente avviato ed impostato disponibile al link sopra citato);
- *BitVir-0.1.0-Cec.jar* è il JAR relativo alla botnet nel ruolo di C&C (si sconsiglia di avviare questo applicativo in fase di testing poiché necessita di configurare le password per il database, oltre alle altre impostazioni dei file *properties*);
- *BotNet* cartella contenente il codice sorgente della botnet;

- *DnsNet* cartella contenente il mock DNS.

Link al download del rar:

<https://mega.nz/#!FAtwnDZT!KuQcsusFw2214A8U--98JYmIRnLbD3eAfvbRZFzw5xs>

## Testare la botnet

Con il fine di testare BiTViR forniamo temporaneamente una rete VPN tramite il software Hamachi LogMeIn, al quale è possibile accedere tramite le credenziali:

ID Rete: SIIBOTTEST  
Password: sicurezza2016

Una volta effettuato il log in ci si troverà davanti un host connesso alla rete: si tratta del nostro C&C. Il mock DNS risulterà online e raggiungibile all'indirizzo:

005myvt.nwsvr1.com

Accedendo a questo sito si verrà reindirizzati automaticamente all'attuale C&C e si potrà navigare tra i pannelli del sito relativo alla botnet.

Nel caso in cui si voglia testare la botnet, a questo punto basterà avviare, da terminale, BiTViR tramite la linea di comando:

```
java -jar BitVir-0.1.0-Bot.jar
```

L'host che avvierà l'applicativo a questo punto assume il ruolo di BOT e sarà pronto a comunicare con il C&C. Per comodità e facilità di testing, evitiamo la descrizione dell'intera procedura che necessita di almeno tre host: uno farà da DNS e da C&C, ed un paio da bot, passandosi il controllo del C&C tra di loro. L'applicazione compilata come bot, infatti, come visto in dettaglio nei capitoli precedenti, può assumere ruoli diversi (compreso il ruolo di C&C) in maniera del tutto autonoma e celata agli occhi dell'utente.

Nel caso in cui si voglia una dimostrazione completa del procedimento di utilizzo della botnet, diamo completa disponibilità per stabilire un incontro.